NEURAL NETWORKS AND ADAPTIVE COMPUTERS

THEORY AND METHODS OF STOCHASTIC ADAPTIVE COMPUTATION

HUAIYU ZHU

University of Liverpool

Ph.D. Thesis 1993

Neural Networks and Adaptive Computers: Theory and Methods of Stochastic Adaptive Computation

Thesis submitted in accordance with the
requirements of the University of Liverpool
for the degree of Doctor in Philosophy
by Huaiyu Zhu

June 1993

Department of Statistics and Computational Mathematics
University of Liverpool

NEURAL NETWORKS AND ADAPTIVE COMPUTERS:
THEORY AND METHODS OF STOCHASTIC ADAPTIVE COMPUTATION

HUAIYU ZHU

Abstract

This thesis studies the theory of stochastic adaptive computation based on neural networks. A mathematical theory of computation is developed in the framework of information geometry, which generalises Turing machine (TM) computation in three aspects — It can be continuous, stochastic and adaptive — and retains the TM computation as a subclass called "data processing". The concepts of Boltzmann distribution, Gibbs sampler and simulated annealing are formally defined and their interrelationships are studied. The concept of "trainable information processor" (TIP) — parameterised stochastic mapping with a rule to change the parameters — is introduced as an abstraction of neural network models. A mathematical theory of the class of homogeneous semilinear neural networks is developed, which includes most of the commonly studied NN models such as back propagation NN, Boltzmann machine and Hopfield net, and a general scheme is developed to classify the structures, dynamics and learning rules.

All the previously known general learning rules are based on gradient following (GF), which are susceptible to local optima in weight space. Contrary to the widely held belief that this is rarely a problem in practice, numerical experiments show that for most non-trivial learning tasks GF learning never converges to a global optimum. To overcome the local optima, simulated annealing is introduced into the learning rule, so that the network retains adequate amount of "global search" in the learning process. Extensive numerical experiments confirm that the network always converges to a global optimum in the weight space. The resulting learning rule is also easier to be implemented and more biologically plausible than back propagation and Boltzmann machine learning rules: Only a scalar needs to be back-propagated for the whole network.

Various connectionist models have been proposed in the literature for solving various instances of problems, without a general method by which their merits can be combined. Instead of proposing yet another model, we try to build a modular structure in which each module is basically a TIP. As an extension of simulated annealing to temporal problems, we generalise the theory of dynamic programming and Markov decision process to allow adaptive learning, resulting in a computational system called a "basic adaptive computer', which has the advantage over earlier reinforcement learning systems, such as Sutton's "Dyna", in that it can adapt in a combinatorial environment and still converge to a global optimum.

The theories are developed with a universal normalisation scheme for all the learning parameters so that the learning system can be built without prior knowledge of the problems it is to solve.

Acknowledgements

I would like to express my sincere thanks to Dr. R. Wait for his help and supervision during the preparation and writing of this thesis. Without his patient scrutiny of so many revised versions the pile of half-baked ideas which was to become this thesis would never become this thesis.

I would also like to express my gratitude to the British Council, The State Educational Commission of China, and the Y. K. Pao Foundation, who co-sponsored this research through the Sino-British Friendship Scholarship Scheme (SBFSS).

I want to thank Prof. J. G. Taylor of King's College London for inspiring communication and for introducing me to the literature of reinforcement learning.

I am grateful to my wife for her understanding, encouragement and support during difficult times. A number of ideas in the thesis also initialised during our discussion.

Thanks are also due to the staff in the Department of Statistics and Computational Mathematics for their assistance and advice, and my other friends and colleagues for their help and understanding while I was completing this thesis.

To
My Parents
and
My Wife

Contents

List of Figures

Part I

Fundamentals

Chapter 1

Introduction

1.1 Motivation

It has long been a dream of people to design a machine which can think. The advent of computers had been the hope of many that this might become a reality. Much has been achieved by the computer sciences toward this goal, but the traditional symbolic artificial intelligence failed in a crucial point: however smart a computer can be in doing many jobs, it does not behave like human beings. The most difficult problems for computers are just those problems which are most simple for humans, even for some other animals. Does this reflect some fundamental differences between a machine and a conscious being? Or is this merely because the computers of today are not yet complicated enough? This is the core of the current intensive debate concerning the so-called "strong-AI thesis" [Sea80]. Although it is accepted common sense in the scientific community that there is no mystic force behind the phenomenon of intelligence, it is evident that the current computational theory lacks something essential to account for it. This is even more remarkable considering that Turing machines (TM) are universal computational machines.

Recent exciting developments in connectionist models of computation [RM86, MR86] point out two possible extensions to TM which seems to be essential for intelligence: the stochastic nature of general computation and the accumulation of information. Although these ideas were already present in Turing's first account of artificial intelligence [Tur50], they remained largely ignored in the later developments of computational science. In this thesis we try to develop a unified theory and general methods of stochastic adaptive computations based on neural networks (NN). The results suggest some new directions in which the debate on strong-AI thesis might be resolved, in science rather than philosophy.

For the purpose of introduction, a brief recollection of the particular train of thought leading to this theory is probably more informative. The starting point is the back-propagation learning rule [RHW86] and the results that feedforward NNs are in a sense universal approximators of functions [Fun89, HSW89]. This means that NN can be regarded as a parameterised function which can adjust itself to suit the demand. The question I asked was: "Can such devices learn to play chess?" This question is quite legitimate since the rules of chess are well defined. It is also significant since, as many pioneers of computational theory had recognised [Wie48, Tur50, Sha50, Tur53], if computers can learn to play games like chess, they can do things hitherto considered as the reserve of human intelligence.

One aspect of games like chess of particular importance to computational theory is that it is impossible to enumerate all the possibilities of the game, even with the resource of the universe [Sha50]. This makes it a necessity that any successful method for such problems be adaptive, or incremental, so that it can provide a solution with limited resources, and can improve the solution whenever more resources are available.

What a human player knows about chess which makes him a good player even before he can actually enumerate all the possibilities? A human player of chess learns two basic things: how to evaluate each board position; and which move is likely to lead to good positions. The first fundamental question arises: What does it means that a position is "good"? After a moment of reflection, one can be sure that the optimality of a position must be a measure monotonically related to the probability of winning. Since the rules of chess is deterministic, the players must play stochastically [1].

Here we restate our principal observations: First, for a system to be intelligent, it must be able to learn more than what it is programmed for; and second, in order for the concept of learning to be well defined in this sense, the system itself must act stochastically, independently of whether the environment is stochastic.

Assuming that the players use stochastic strategies, a learning machine for playing chess can, at least in principle, be constructed with two NNs, one for evaluation of board positions and the other for producing the moves stochastically. It is obvious, however, that there is nothing special to chess in this formulation: the learning machine thus constructed would seem to incorporate all the ability of human problem solving in an "arbitrary" environment. How far this approach can go in the direction of building a general problem solving machine can only be decided by quantitative studies, of which this thesis is intended to be a first step.

As is evident from the above discussion, our concept of "computation" is extended from that based on the formal definition of TMs. To avoid discussions which go too far afield in this introduction, we simply call our new concept "stochastic adaptive computation" while the classical one as "TM computation". We return to motivation in the conclusions in §10.5.

1.2 Theoretical Framework

As soon as one tries to build a theory of "intelligent behaviour" based on so-called "mechanical elements", one realizes that there is a vast level of complexity. The complexity of a theory does not depend on what it studies, but on the "distance" between its overall structure and its fundamental building blocks. We call this the *scope of complexity*. The most ambitious objective along the direction outlined above seems to be to account for the intelligent behaviour of an animal in complex environments based on the details of neurons in its brain.

In this thesis we focus our attention on the more realistic scope of complexity: down to some regular and mathematically well defined neurons, up to the discrete time decision

[1] The reason for stochastic behaviour in chess-playing is different from traditional situations where there are stochastic elements in the environment: If the two players are assumed to be of similar mentality, neither player can be less stochastic than his environment, which consists of the other player and the deterministic rules. The degree of stochasticity can be measured, for example, by the entropy of the action selection process.

problems in some Markovian environments [2].

Even this is too complex, so the theory is divided into two levels: the theory of "adjustable information processors" and the theory of "adaptive computers". The former takes into account the lower part of the scope of complexity, while the latter the upper part. This framework is illustrated schematically in the series of four "zoom in" figures, Figure 1.1–1.4, where each succeeding figure details part of its predecessor. We shall return to these figures in the reversed order in the main body of the thesis. (Figure 1.4 in §4.2, Figure 1.2 & 1.3 in §4.3, and Figure 1.1 in Chapter 9.)

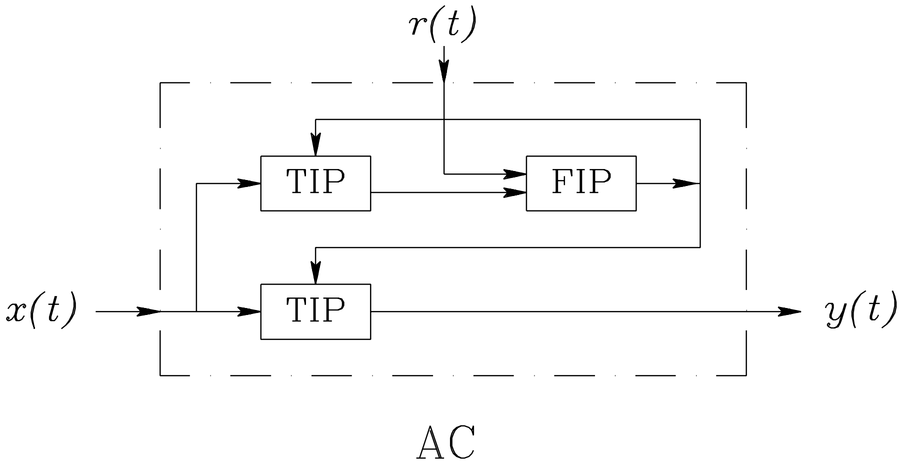

Figure 1.1: An adaptive computer (AC) constructed with trainable information processors (TIP) and fixed information processors (FIP).

$x(t)$–input, $y(t)$–output and $r(t)$–reward.

The concept of AIP is an abstraction of the TMs and a subset of NNs (all those to be studied here). The former is called a programmable information processor (PIP) and the latter a trainable information processor (TIP). Each AIP realizes a (stochastic) mapping from its input to its output depending on some parameters (abstraction of programs or connection weights) which can be adjusted to increase the expectation of *immediate evaluation* of the input-output pair. Whether this is done artificially (by some outside programmer) or automatically (by AIP itself) differentiates between PIP and TIP. A physical device implementing a fixed (stochastic) mapping is called a fixed information processor (FIP), which is equivalent to a communication channel [Sha48] although with quite different purposes. A PIP with a given program is a deterministic FIP.

The concept of AC is an abstraction of animal brains and artificial adaptive computational devices. An AC works in a certain, but quite arbitrary environment, receiving a *time sequence* of inputs, emitting a *time sequence* of outputs, receiving a *time sequence* of rewards in return, and modifying internal parameters so as to increase the utility of each state, which is the expectation of a particular form of accumulated reward it would receive in the future following that state. If an AC can be modelled by a Markov decision process (MDP), it is called a basic adaptive computer (BAC).

[2] In the concluding chapter we shall make some suggestions on how the top line might be raised to include more complex environments.

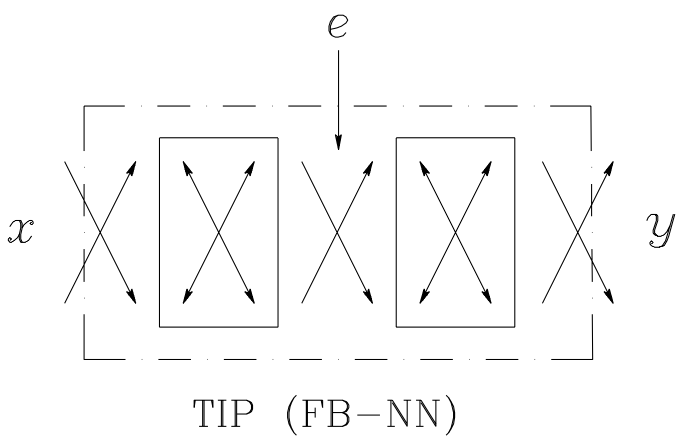

Figure 1.2: One type of trainable information processor: simple-layered SQFB-NN.

x–input, y–output, e–evaluation.

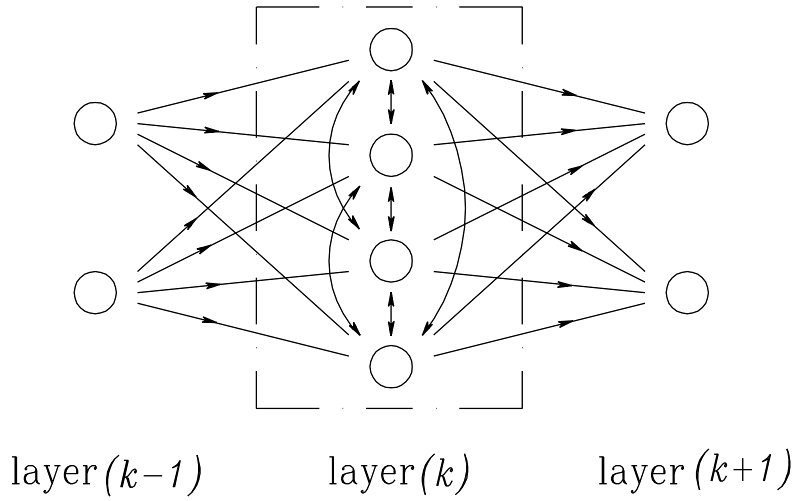

Figure 1.3: One layer of a SQFB-NN.

Each circle is a neuron. Note that there are bidirectional connections inside each layer, and unidirectional connections between layers.

Both TIP and AC are systems which accumulate information while processing it. What differentiates TIP and AC is that for the former the adjustment of the parameter only depends on the evaluation of the input-output pairing. It does not depends on how the past might influence the present, nor how the present might influence the future. Nor does it take into account that the environment might change in response to its own change.

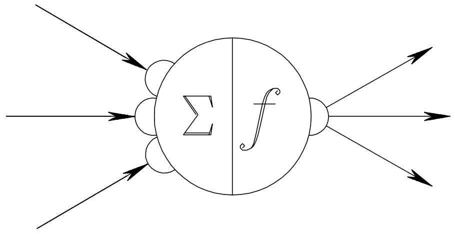

Figure 1.4: A typical quasilinear neuron.

The input is linearly combined (Σ), and nonlinearly transformed (f). The output is broadcast to other neurons along connections. In biological analogy, the big circle is the *soma*, the semi-circles to the left are *synapses*, the semi-circle to the right is the *axon*, and the arrowed lines are *dendrite*.

---

* * *

The distinction between adjustable information processor (AIP) and adaptive computer (AC) is one of the most important feature in the structural framework of our theory. The basic idea behind this is that AC can be constructed from modules each composed of a TIP and some FIP. As a metaphor, we consider TIP and FIP as standard components and the AC as a complete machine. Neural networks fit in this picture as certain "universal" TIPs. When we study NNs, we only consider what it is required as a TIP; when we study ACs, we take for granted that there are TIPs for implementing required stochastic mappings.

The significance of such a "division of labour" can be somehow explained through the concept of credit assignment problem (CAP) for learning machines [Min61], although we shall seldom have chance to mention this concept in technical discussions. The study of TIPs corresponds to the so-called structural credit assignment problem (SCAP), where given an evaluation of a system, it is required to assign the credit proportionately to various subsystems. The study of ACs corresponds to the so-called temporal credit assignment problem (TCAP), where given the evaluation of a mission, ie., a sequence of decisions, it is required to assign the credit proportionately to various steps in its execution. There is also a biological analogy: the study of TIPs and ACs correspond respectively to neurological and behavioural studies.

Our theoretical framework puts strong constraints on both TIPs and ACs: The TIPs must be able to be connected in a modular structure; It is not enough that NNs are designed as independent models, as is being done in most NN research nowadays. The ACs must be stable; this cannot be derived simply from the fact that all its components are stable in a stationary environment. Considerations of such kind will sometimes be explicitly stated, but mostly implicitly applied throughout the thesis.

* * *

We try to provide both the mathematical theory and computational methods, the latter having further requirements. Our goal is to see how much of the constructions of pure mathematics can be carried out by physical devices restricted in the *universe*. Considering that there are only about 10^{10} neurons in the human brain [And83] and about 10^{80} atoms in the whole observable universe [Dav82], the following concepts, closely related to the concept of NP-completeness [GJ79], are very important for comparing the computational viability of various theories.

A finite set X is said *enumerable* [3] if, by a proper coding method, all of its elements can be examined, either simultaneously, or sequentially over a tolerable duration of time, in a physical device of current technology. A set X is *combinatorially enumerable* if its elements can be labelled by a coding system A^n, where A is the alphabet, n is the maximum length of code, such that $nA := \{(k, a) : \ k \in \mathbb{N}_n, \ a \in A\}$ is enumerable. If X is (combinatorially) enumerable, we also say that $|X|$, the number of members of X, is so. For example, sets of 100, 2^{100} and $2^{2^{100}}$ elements are enumerable, not enumerable but combinatorially enumerable, and not combinatorially enumerable, respectively [4].

A *combinatorial optimisation problem* is a global optimisation problem over a combinatorially enumerable set. By definition, only approximate solutions are in general achievable. To date, the most efficient general methods to combinatorial optimisation problems are stochastic methods, such as simulated annealing (SA) method and genetic algorithms (GA). This is no coincidence, and its root can only be explained by the information theory.

Classical computational theories have always been theories of certainties, in which the structure "if ... then ... else ... " takes care of all the imaginable possibilities. As soon as we are faced with problems for which it is even impossible to imagine all the possibilities before they actually occur, the *amount of information* in a problem becomes more important than the *amount of data*. The value of these two measures diverges as the probability distribution becomes less uniform. The stochastic methods have the ability to process an enumerable amount of information embodied in a combinatorially enumerable amount of data, by dealing with some parts of the data with diminishing, but still positive, probability.

There are at least two other reasons why stochastic computations are important for intelligence: (1) The environment might be stochastic; (2) In game theory, the optimal strategy may exist only if mixed strategies are allowed [5]. In these two "classical" reasons: either the condition or the solution of a problem is stochastic. However, as we have just seen, facing with combinatorial problems, even if the condition and the optimal solution are both deterministic, it may still be necessary to use a stochastic method.

One of the mathematical foundations of stochastic computation is the theory of Markov chains (MCs). Since in our theory the computation is also adaptive, so that the MC itself will also move as learning proceeds, we must consider the space of all the MCs. Fortunately, the fundamental concepts for such consideration have already been

[3] This has nothing to do with the concept of countable, sometimes also called denumerable, in set theory.

[4] Although these concepts depend on the "current technology", these statements in this example have an absolute meaning if physical constraints are taken into account. The number $2^{100} \approx 10^{30}$ is more than 100 times larger than the total number of neurons in the brains of all the humans ever lived.

[5] The terms "policy" and "strategy" are used interchangeably. Same applies to "stochastic policy" and "mixed strategy".

provided by the theory of information geometry (IG), which is the study of manifolds of probability distributions[AH89, AKN92].

* * *

Our main purpose of studying NN is to develop more powerful computational devices. Therefore we shall mainly examine them from mathematical, and sometimes engineering, points of view. These "artificial neural networks" (ANNs) are systems that we can control their internal construction. On the other side of the same coin are the "biological neural networks" (BNNs), which we know work very well, but we do not know how. Although the study of both ANNs and BNNs may be beneficial to each other in several ways, our theory does not logically depend on the study of BNNs.

Numerous neural network models have appeared in the literature. It is impossible to define NN in such a way as to include everything which has been called NN without including something definitely not to be called NN. We shall give formal definitions for what we call the class of quasilinear neural networks (Q.NN). Our theory is further restricted to a subset of Q.NN, the class of homogeneous semilinear neural networks (H.S.NN). This restriction is justified by the following three simple reasons: the class of H.S.NN includes many of the most popular neural networks; it is general enough as a class of TIPs; and it is regular enough to allow a complete mathematical theory. Intuitively, the term "semilinear" means the input of each neuron is a linear combination of the inputs to the network and the outputs of all the other neurons. The term "homogeneous" means that the output of each neuron depends on the input of that neuron by a rule which is the same for all the neurons in the network.

1.3 Outline of the Thesis

The next two chapters, Chapter 2 and Chapter A provide background material. Chapter 3 sets up the basic theoretical framework of this thesis. In the following four chapters, Chapters 4–7, we study neural networks as realizations of trainable information processors (TIPs), which solve the structural credit assignment problem. In Chapters 8 and 9, the concept of time is introduced into the problem. We study adaptive computers, which solve temporal credit assignment problem, provided that TIPs are available. Chapter 10 summarises the results, discusses various implications, and suggests future research directions.

Chapter 2: Related work to this research are reviewed. The emphasis is on the most significant and general related results, and no attempt is made at sorting out the first contributor.

There is no need to know anything about biology, engineering, or physics in order to logically follow this thesis, but a background in related fields would certainly help in comprehending the reason behind each development.

The review is divided into three parts.

In §2.1, various of neural networks are reviewed from the computational point of view. That is, we are mainly concerned with the most simple structures which can perform general tasks, instead of those models which are most descriptive of BNN.

In §2.2, we review work related to our adaptive computers. An emphasis is on work which is most related to Markov decision processes (MDPs) (Chapter 8) and basic adaptive computers (BACs) (Chapter 9).

In §2.3, we review work related to our approach to stochastic adaptive computation. In particular, we review the related subjects in the following fields: probability theory, information theory, statistical mechanics, combinatorial optimisation (computational complexity, simulated annealing and genetic algorithms)

Chapter 3: We develop a mathematical theory of adaptive information processing in general, based on probability theory and the theory of information geometry (IG).

We shall formulate the *entropy principle*, which generalises the well-known maximum entropy principle, and show its relation with the simulated annealing (SA) method. We shall also apply the entropy principle to parameterised MCs, which forms the basis of the later chapters.

The concept of trainable information processor (TIP) is also defined in this chapter, which serves as an interface between NNs and ACs: each NN is a system whose functionality is a TIP, while an AC is a system whose main components are TIPs.

Chapter 4: We develop a mathematical theory of homogeneous semilinear neural networks (H.S.NN).

The basic definitions of structure and dynamics of quasilinear neural networks (Q.NN) are give in §4.2, and a classification is given in §4.3. In the rest of the thesis we focus on to a smaller class, H.S.NN, which from there on is also abbreviated as NN when there is no danger of confusion.

The prefix notations S- (stochastic), D- (deterministic), C- (continuous), Q- (quantised, or discrete), F- (feedforward, multilayer, or associative), B- (feedback, symmetric connection, or correlative) are used to denote the classification of Q.NN. As listed in §4.3.4, the class of Q.NN includes some of the most well-known NNs. The intrinsic relations between SQ-NN and DC-NN are also discussed (§4.4). We also study the FB-structure which generalises F- and B- structures (§4.5).

In §4.6 we give a general definition of neural network learning rules. Various existing learning rules are studied and generalised. In particular, we shall show that most existing gradient-following (GF) learning rules can be transformed into simulated annealing (SA) learning rules, thus avoiding convergence to local optima in the learning process.

Chapter 5: We study some of the most important special examples of H.S.NN, including the DCF-NN (back-propagation network), the DCB-NN (Hopfield net), the SQF-NN (belief network) and the SQB-NN (Boltzmann machine). We shall also study DCFB-NN as a generalisation of DCF-NN and DCB-NN.

Chapter 6: We develop the mathematical theory of SQFB-NN. The main thrust of this chapter is the derivation of the learning rules, based both on GF and SA. A numerical example is provided in which GF learning rule inevitably leads to local optima. This is overcome by the SA learning rule.

Our mathematical derivation of the learning rules reveals the reason behind and shortcomings of the so-called "Hebbian learning" mechanism, which have been used extensively *ad hoc* in many existing learning rules.

Chapter 7: We study various aspects concerning the implementation of SQFB-NN.

The core of a stochastic neural network is a Gibbs sampler (GS) (defined in §3.4.1), which is also widely used in other stochastic computations. In §7.2 we derive a very efficient implementation of GS on a conventional computer, whose application should not be restricted to neural networks.

In §7.3.3, we analyse a typical, and yet tractable example, the encoder problem, to derive theoretical relations between various parameters of learning and the performance of the network. We give a mathematical treatment of learning which has quantitative theoretical predictions subject to comparison with experiments. These theoretical results can only be understood in the sense of information theory, and are, to our best knowledge, completely new.

Many numerical tests are performed for the SQFB-NN with the SA learning rule. In §7.5 we present and analyse the results concerning various aspects of learning. This is also compared with the Boltzmann machine (BM) learning rules. In fact, we shall show that there are actually two brands of BM learning rules, one corresponds to GF and the other to SA.

Chapter 8: We develop a mathematical theory of Markov decision process (MDP), which generalises the classical theory of dynamic programming (DP) and controlled Markov chains. Our generalisation is that the entropy principle (§3.4) in applied to the decision module (policy function in DP), so that the resulting learning rule is a multistage variant of simulated annealing which avoids local optima. This is necessary for implementing the MDP on neural networks. In our formulation many different problems in DP are treated in exactly the same way (§8.5.3).

Chapter 9: We study how to implement the basic adaptive computer (BAC) based on NN. Many possible variations to the main algorithm are given. Simulation results on two test problems are given in §9.7, one is a game of conflict of two players, where the solution contains many alternating regions of local optima. We show that it never converges to a global optimum if GF learning rules are used, and that it always converges to a global optimum if SA learning rules are used. Another problem reported in the same section also contains local optima. It also has an enormous state space. As far as we know, such solutions have never been reported before.

In §9.5, we derive a normalisation convention which is universally applicable to BACs. Since it is difficult to get an exact theoretical result, we adopt a variant of the convention in §7.3.3. With this normalisation convention, we use the same program with same parameters to learn the two very different games.

Chapter 10: We summarise our results. We also further discuss the implications of this research on various related research fields: artificial intelligence, computer architecture, neural physiology and behavioural sciences, probability theory and information theory. We also comment on further researches in these directions, as well as on neural network theory and adaptive computer theory.

Appendix A: There are quite a few new notations introduced in this thesis. They are described in this appendix, together with a review of prerequisite mathematics.

Appendix B: This appendix lists explanation of all the acronyms used in the thesis.

Chapter 2

Review of Related Work

The general problem we are going to study is stochastic adaptive computation in a certain general environment. Related work can be roughly divided into three classes: methods for solving structural credit assignment problems (SCAPs), methods for solving temporal credit assignment problems (TCAPs), and fundamental theory concerning general credit assignment problems for stochastic adaptive computation.

Those pertaining to SCAPs include various neural networks and their learning rules. Those pertaining to TCAPs include game theory, the theory of dynamic programming, and various reinforcement learning methods. The fundamental theory pertaining to general credit assignment problem include theories on stochastic information processing, Markov chains and information geometry.

We shall review only references which we consider to be landmarks in the development of the theory, excluding numerous special methods which may be very useful to specific problems in practice but which do not feature in our general theory. Many similar concepts have been given various names in various research fields over the years. We shall give precise definitions of the terms used in the thesis, we shall not attempt to mention the different and often conflicting definitions in the literature.

2.1 SCAP: Neural Networks

Neural networks are also called connectionist models or parallel distributed processing (PDP) models. The reference book [Was90] includes over 4000 entries up to 1989, where 1988 alone counts for more than 1000.

As we consider neural networks as physical realizations of trainable information processors (TIPs), our theory will be focussed on the class of homogeneous semilinear neural networks (H.S.NN), with supervised or reinforcement learning rules (these terms were briefly explained in §1.3, and will be defined fully in Chapter 4) [1].

* * *

There are generally three aspects of neural networks which are of mathematical interest: the structure, the dynamics and the learning rules.

[1] This choice is based on the conviction that on the one hand H.S.NN form a quite general and rich subset of TIPs, and on the other they have handy physical interpretations. While the latter property does not guarantee more applications or easier manufacturing, it does make it much easier to obtain deep theoretical results necessary for analysis, design and application of neural networks, especially when several neural networks are to be connected together in a complex system.

The first formal theory of neural network structures and dynamics is usually attributed to [MP43, PM47], where it was shown that the DQF-NN (deterministic quantised feedforward neural network) can implement any logical calculations, provided that there are enough neurons and the connection weights are right. The first NN learning rule is usually attributed to [Heb49, p.62]. The first formal theory on a type of artificial neural networks with a learning rule is usually traced to [Ros58, Ros59], where a perceptron network is actually one layer of DQF-NN preceded by a layer of fixed processors [MP69]. The first rigorous theory on what can and cannot be done by a certain type of neural networks was given by [MP69], where they showed that single layer perceptrons are limited in their representational capabilities.

There were various types of neural networks developed since 1960's, which we shall not study here. These include, most notably, various models developed by Grossberg and colleagues [2], see [Gro86] and references therein, and those of Kohonen, see [Koh77, Koh84]. An introduction to these can be found in [BJ90]. A survey of structures and dynamics of deterministic neural networks are given by [Gro88]. It especially covers those well known in the biological sciences and intended primarily as models of BNN.

Many learning rules have been proposed for various neural networks. They are usually divided according to the learning "mode" into unsupervised learning (UL), supervised learning (SL), and reinforcement learning (RL). We classify the learning rules into maximum evaluation (ME) learning rules and maximum likelihood (ML) learning rules. We shall show that ME learning can be transformed into ML learning but not the reverse. RL rules belong to ME learning rules, some SL rules, such as the back propagation (BP) rule [RHW86, Wer74, Par85] are special cases of RL. Some other SL rules, such as the Boltzmann machine (BM) learning rule [AHS85, HS86] is a ML learning rule. Furthermore, any UL rule must also have some kind of performance evaluation, the only difference being that they are built-in, but we shall not consider UL rules here.

In general, learning rules are optimisation methods, almost all the currently available learning rules are based on gradient-following (GF) which may lead to local optima [MP88]. Simulated annealing (SA) [KGV83] can be used to avoid local optima in an optimisation problem. One objective of this thesis is to show that SA can be adapted to replace GF in virtually all the learning rules.

* * *

The beginning of the current "rigorous development period" for neural networks was marked by [Hop82] who proved the stability of DQB-NN (deterministic quantised feedback neural network) by deriving a Lyapunov function for the network dynamics. This was generalised to SQB-NN (stochastic quantised feedback neural network) in [AHS85, HS86], which they call "Boltzmann machine" (BM). The BM is perhaps the most versatile neural network in many respects, although the computation on BM simulated on a conventional computer is expensive [3]. These authors also derived the BM learning rule, which is a prototype of the ML learning rule. The BM learning rule was given as a gradient-following (GF) rule, but in their experiments with BM they actually used the entropy principle implicitly. Unfortunately the way they presented it as a technical skill prevented others from recognising its importance. SQB-NN are usually used in combinatorial optimisation problems [AK89b, KA89, AK89a], but other applications

[2] It should be mentioned here that according to our terminology, many of Grossberg's models are more like ACs than TIPs.

[3] However, as mentioned in [Hin89a] the BM implemented on special chips might be over a million times faster than simulation [AA87].

are possible (For example, the prediction module in §9.2).

Perhaps the most widely studied and used neural network is the DCF-NN (deterministic continuous feedforward neural network), called variably "feedforward perceptrons", "multilayer perceptrons", and "back-propagation networks". It was discovered independently by several authors [RHW86, Cun85, Par85, Wer74]. It is most famous for its learning rule, called back-propagation (BP) rule, which is the prototype for ME learning rules. A great volume of research has been devoted to various aspects of DCF-NN, of particular importance are the approximation properties [Fun89, HSW89, HSW90, Hor91]. For our purpose these results can be generally stated as showing that the DCF-NN is a universal approximator of arbitrary functions in $[\mathbb{R}^m \to \mathbb{R}^n)$, provided that there are enough neurons in the hidden layer. An upper bound for the number of neurons needed for a two-hidden layer DQF-NN was given in [BL91]. This result is immediately applicable to DCF-NN and SQF-NN (stochastic quantised feedforward neural network), as they are generalisations of DQF-NN. Other studies in DCF-NN include how to accelerate convergence of learning [Sam91, RIV91, Fah88], and other aspects of learning [Sus92]. There are probably well over a thousand different published applications of DCF-NN, covering virtually every field where adaptive approximation of a nonlinear function might be of some use. A glimpse of sample applications is provided by [MHP90].

The DQB-NN was also generalised in [Hop84] to DCB-NN (deterministic continuous feedback neural network), usually called "(continuous) Hopfield net", the stability of which was also guaranteed by a Lyapunov function. The ME learning rule for DQB-NN was developed by [Pin87, Alm87, Alm89b, Alm89a]. DCB-NN was also studied as mean field (MF) approximation of SQB-NN [PA87], where another learning rule was derived from BM learning rule, which is also a GF rule [Hin89b]. The applications of DCB-NN include, but are not restricted to, various optimisation problems [TH86, HT86, PH89, dBM90].

The development of SQF-NN has several origins, most of them are related to DQF-NN, ie. logical circuits. One origin was the theory of "stochastic learning automata" [NT74], where each automaton was actually a neuron, and many of them were connected to perform certain tasks [Lak81, NL77, BA85, NT89]. A network of learning automata without loops is a SQF-NN. Another derivation was from the Bayesian inference, this results in "belief networks" [Pea87]. The ME learning rules for SQF-NN was studied extensively in [Wil92, Wil87, Wil88, Wil90]. Recently the ML learning rule for SQF-NN was studied in [Nea92]. It was cited in [PH89] that a similar learning rule was studied for SQF-NN in [BW88].

* * *

A general framework for structure and dynamics of PDP models has been described in [RHM86]. In particular it defines the classes of quasilinear and semilinear neural networks, for which we shall develop a mathematical theory. See [Sej81, Pin88] for other attempts on general theory of NN.

Concerning neural network structures, we have seen that the feedforward (F-) and feedback (B-) neural networks have been proved stable. One of our contributions will be the identification of a class of structures, the FB-structure, which guarantees stability and is a natural generalisation of F- and B-structures.

Concerning the dynamics of NN, there is a formal duality between SQ-NN and DC-NN, informally discussed in [Hop89, Ami89]. This concept turned out to be very fruitful in our theory, both in designing networks of rich and versatile dynamics, and in providing penetrating insights of the properties of the networks from different points of view. It should be noted that the formalism in [Sej81] even allows SQ and DC to be viewed as extremes of a one-parameter family. We discuss this briefly in §4.4.

Neural network learning rules are surveyed in [Hin89a], this reference contains original ideas on various important issues of learning. Learning based on a genuine entropy principle was given in [LTS90], although their method is somewhat theoretical rather than practical.

Much of this research was inspired by the two volume "PDP book" [RM86, MR86], which is something of a "connectionist manifesto". It covers much of the philosophical background of the resurgence of interest in neural network computing, as well as technical details of many of today's well-known neural network models.

2.2 TCAP: Reinforcement Learning

Several related research fields are concerned with the problem of "optimisation over time", including games theory, dynamic programming (controlled Markov chains, Markov decision processes), learning automata, reinforcement learning, and adaptive control theory.

Games theory is a general mathematical theory concerning the "optimal behaviour" of many players in an environment [vNM47]. Its most profound consequence is that in general there can be no consistent definition of optimal actions which is universally valid, but it is possible to define optimal strategies under very weak conditions, provided that "mixed strategy", also called "stochastic policy", is allowed.

One of the basic methods in games theory is to transform a multistage game into an equivalent single stage game (in the sense that there is a one-one correspondence between their optimal solutions), by remembering the whole history of moves. We call this "average over histories". This is enough for the analytical study of the characteristics of optimal solutions, but is deficient as a computational method for finding optimal solutions, for the following reasons: (1) For many interesting games, such as Markovian games, it is not necessary to remember the whole history; (2) For most non-trivial games it is never possible to base the decision on the whole history of past moves (the number of histories usually grows exponentially with the number of steps of the game); (3) It assumes that each player seeks an "optimal strategy" based on the assumption that all the other players take their optimal strategy.

The theory of dynamic programming (DP) [Bel57, How60, BD62, DL76] provides a practical computational method to solve a special type of game, "games against Nature", in which an "agent" (a certain player) improves its policy (strategy) in an environment (all the other players and the rules of the game) which is stationary (ie., the transition probability of the environment does not change). Its main idea is to use the smallest amount of data necessary, instead of the whole history, to describe the "state", so that the game remains Markovian. This implies the so-called "optimality principle": If a history is generated by an optimal policy, then the policy is optimal at each state in the history. In other words, if a policy is optimal at a state, it will still be optimal if more about the past is known. Therefore it is possible to define a scalar evaluation, called "utility", of each state. The optimality theorem of dynamic programming says that optimal policy can be obtained by choosing actions at each state to maximise the expected utility, we call this as "average over states". The mathematical part (ie. excluding the model building) of DP is also called theory of controlled Markov chains (CMC) or Markov decision processes (MDP) [Ros83, DY79, Der70]. A quite intuitive introduction to the general applicability of utility function was given in [CM59].

* * *

It can be seen that games theory treats the environment as being of infinite intelligence (every player chooses optimal strategy), while the DP theory treats the environment as being of zero intelligence (Nature will neither cooperate nor compete). The decision problems facing humans or other higher animals have environments more general than those in both dynamic programming and game theory: there are various agents in the environment with various degrees and types of intelligence. The behaviour of the environment is a crucial *empirical factor* regarding the performance of the agent, it cannot be dispensed with or assumed away. Furthermore, it is impossible, and obviously unnecessary, for a naturally intelligent agent to enumerate all the states and actions before choosing an action. Technically speaking, the classical DP theory treats the states as data, not as information. More specifically, if two states are slightly different but correlate strongly, this property will not be utilised.

The assumption of a stationary environment in DP implies that there is always a deterministic optimal solution. This turned out to be both fortunate (the early development of DP theory and computer implementation) and unfortunate: Since the deterministic policies are mappings from the state space to a discrete space, the action space, they can be represented in tabular form of size equal to the product of the size of state space and action space. So for a rather long period there had been little effort to parameterise the policy in DP. The first completely parameterised RL method seems to be the rather neglected reference [Wit77], but his parameterisation is in the form of a large look-up table of probability distributions, and he did not mention the relation with either DP or RL. A remarkable feature of this reference is that it presented a theory, rather than a method specifically designed for a few problems.

The diverse research field of reinforcement learning (RL) [4] originates from the study of animal behaviour under "classical conditioning" (Pavlov conditioning) [SB90]. One of its main features is to (continuously) parameterise the utility function and/or the policy, so that they can be improved "incrementally". Unlike DP, there are many methods but only a few scattered theories in RL, since the relationship between the parameterisation and the actual application is usually a crucial factor in the overall performance, but this relationship most often has only been analysed with regard to specific test problems (eg. [Tes92, Lin92]).

The (implicit) application of RL as a computational method can be traced back to the "checker playing program" of Samuel [Sam59], in which the evaluation is represented by a polynomial. Since pure polynomials are not good for this purpose (and, indeed, not good for most approximation problems), some "heuristic rules" were used in the program for managing a set of preferred polynomial terms.

Remark 2.2.1 Heuristic rules became rather popular in AI research [NS72], but there has never been any solid theoretical foundation [Dre79][5]. They are only "successful" in

[4] The term "reinforcement learning" have been used with two different meanings: In one sense, it means to learn something when the only feedback is a scalar evaluation (contrast to supervised learning); In another sense, it refers to learning in a stochastic environment without using exhaustive enumeration methods (contrast to DP).

[5] As will be discussed in §10.5, it is doubtful that there will ever be any formal theory on heuristic rules.

a few cases where sufficient amount of heuristic rules have been accumulated by *human* experts, such as in chess or checker playing. Despite their continued popularity in AI research, we shall not discuss these methods any further.

The theory of learning automata (LA) was developed from yet another origin [NT74, BA85], which attempts to parameterise the policies of simple agents in a game-like environment. Since the utility function usually does not feature in these theories, they have about the same efficiency as "average over histories", although their main virtue is that there is no need to enumerate the histories. These methods are also called backpropagation through time (BTT) [Wer90a].

The first "average over states" method in which both the policy and evaluation are parameterised is usually traced to [BSA83]. The policy is also represented in the form of a look-up table, and is applicable to agents with only two choices of actions [6]. Use of neural network to approximate the policy and utility function was proposed in [And89], and was later generalised to problems with more than two actions in the Dyna model [Sut90]. In this model the policy is represented as a Boltzmann distribution over actions. This is of fundamental importance, for it allows the introduction of information theory into the decision theory. We call these two parameterisations the "evaluation module" and the "policy module" in the "basic adaptive computer".

Most of RL research concentrates on the evaluation module, ie., they assume that there is a mechanism to choose the optimal option, given the evaluation. The temporal difference (TD) method [Sut84, Sut88] for predicting the expected value of a function of the states of a Markov chain can be used for updating the evaluation module [BSW90]. Its convergence was proved with a local representation of state space [Day92]. It extracts the first order statistics in such a way as to achieve a balance between minimum bias and minimum variation. Another method for updating the evaluation module is the Q-learning method [Wat89], which requires a local representation of action space as well. Its convergence was shown in [WD92], under the assumption that the agent has an ability similar to "dreaming" which can replay state-action sequences with appropriate frequency. This is necessary since the decision module simply select the optimal action deterministically according to the evaluation module. Various models based on these methods have been tested on specific problems in [Lin92, dRMT92, Sin92]. A review of RL can be found in [Bar90].

* * *

These above methods (game theory, dynamic programming and reinforcement learning) either update the policy in the action space (deterministic policies), or in the policy space by gradient following. The former is not applicable to large complex problems, while the latter only converge to local optima, which are in general abundant in non-trivial problems, although absent in most of the test problems used in the literature. To overcome local optima and to avoid exhaustive search, the representation must be in such a form that any two deterministic policies are "adjacent" to each other. This is in general only possible if such policies are (topologically) vertices of a simplex. However, the only way to effectively explore a simplex of a combinatorial dimension is to use a stochastic method by which the probability distribution corresponds to a point on the simplex. This makes the stochastic methods such as simulated annealing effective in this context.

[6] This is the only non-trivial problem in which a decision problem is also a control problem. Suppose a n-simplex is also a n-cube. Then $n+1 = 2^n$, which implies $n = 0$ or $n = 1$. (See §3.3 for explanation)

The theories discussed above are decision theories. A related large research area is the theory of adaptive control, which also deals with "optimisation over time". Although these two terms are often used interchangeably (see [FS90] and many other references in [MSW90]), they are almost mutually exclusive and complementary to each other in mathematical terms. The difference between them is that the former is concerned with global optimisation while the latter concerned with differentiable optimisation. Control problems are usually defined by an "objective functional", usually of bounded second order derivative, on a "control space", which is usually a finite dimensional continuum. The decision problems in general can only be defined by a functional on a "policy space", usually a simplex, spanned by a "action space", which is usually a discrete set. Any method capable of global optimisation of a general non-differentiable functional over a continuum must necessarily deal with infinite amount of information, and is physically impossible. We shall not be concerned with control problems in the thesis except in §10.4.2, where the relations between decision and control theories are further explored. For a survey of adaptive control see [WRS89] and the collection [MSW90].

The theories mentioned above are based on the assumption that the "state of world" at any moment can be perceived directly by the agent, and processed in detail. In practice such an assumption may not be granted, and the problem does not belong to MDP. Many specific models have been proposed in the literature to deal with these issues separately, with rather artificial criteria for success (§10.4.2).

A comprehensive discussion of important problems relating to the aforementioned methods was given in [Tes92]. Some of them will be solved satisfactorily by our methods, some will be given practical solutions without theoretical proofs, while others remain open problems, although our methodology may be useful in future solutions.

2.3 Information Theory and Related Topics

The concepts of entropy, Boltzmann-Gibbs distribution and statistical ensembles, and their relations to irreversible processes (the Second Law of Thermodynamics) and disorder, have been known in statistical mechanics since the end of last century [LL80].

Shannon established the mathematical foundation of information theory in the monumental papers [Sha48], in which it became perfectly clear that *entropy* is a purely mathematical concept, just as probability is, and it is a more fundamental concept than the physical concept of irreversibility. One of the most significant contributions of [Sha48] was to show that entropy is the *only* measure of *disorder*, or *missing information*, under some very reasonable conditions one would usually attribute to such a measure.

Some further significant contributions to information theory include [Kol56b] (information of continuous variables), [Khi57] (entropy of a Markov chain), and [Kul59] (the information divergence, also known as Kullback separation, and the cross entropy between two distributions).

The *maximum entropy principle* (MAXENT) [Jay57a, Jay57b] is a further attempt at clearing the relation between the mathematical theory of information and the physical theory of statistical mechanics. One interpretation of MAXENT is that statistical mechanics can be based on the single axiom that the known "conservative variables", such as energy, are the only ones which can be deduced from observing the macro states of a physical system. We use the concept of entropy in the purely mathematical sense, so we shall not go into the details of statistical physics. There is a detailed discussion of

MAXENT in [TTL84] from a physicist's point of view.

Another interpretation, perhaps the intended one, is that MAXENT is the only natural combination of Laplace's "principle of insufficient reasons" and Shannon's characterisation of entropy as a measure of missing information: Whenever there is a choice under uncertainty, the best strategy is to adopt a maximum entropy distribution, given the same level of value.

An early attempt at applying the Monte Carlo method to the calculation of state variables in massively interacting systems led to the Metropolis method [MRR+53], which is one of the so-called importance weighted methods [HH64]. The Monte Carlo method involves a change of problem, which we call the *Monte Carlo principle* [7]: instead of finding analytically the average of a random variable over a distribution which is computationally intractable, an average over samples of the random variable is used. The convergence is then defined in the sense of probability.

The Metropolis method is now known as a *Gibbs sampler* (GS), one of the first in a large family, which produces an irreducible Markov chain whose stationary distribution is a certain Boltzmann (Gibbs) distribution. A continuous version of GS was provided in [Gla63]. Some of the most important properties of GSs have been studied in [GG84], and a survey of their applications was given in [Yor92].

The study of NP-complete problems reveals that many combinatorial optimisation problems that are routinely encountered are inherently intractable if exact global optimisation is attempted [GJ79]. When there are certain structures on the space of alternatives, local deterministic methods such as gradient-following (GF) methods will usually only lead to local optima. When GF methods are combined with the Monte Carlo principle, the resulting methods are called stochastic GF methods. In a sense, such methods are only qualitative methods, since gradient alone does not tell how much change should be made with limited supporting information.

Monte Carlo methods were further developed into the simulated annealing method [KGV83], which solves the combinatorial optimisation problems efficiently, by equating energy with the cost function to be optimised. This involves a further change of problem, which we call the *entropy principle*. The detailed analysis will be given in §3.4, where its relation to the MAXENT and linear programming methods are also elucidated.

The SA method is the first computational method in which the computational effort is directly related to the *mutual information* [Kul59] between data and performance, not on data alone. In fact, we shall show that the entity which is usually called *temperature* is simply a measure of the *value* of information in this context.

One major theoretical breakthrough in the study of the SA method occurred in the research of cooling schedules which, in our interpretation, is the study of how to assign the value to the missing information. The results of *constant thermodynamic speed* (CTS) [NS88, AHM+88, SNH+88] were based on purely theoretical considerations, and are of a quite universal character. Their interpretation of the approach to equilibrium as the approximation in the sense of Kullback separation is very convenient for our applications (also see [BGMT89]), and our experiments in §7.5 provide quantitative support for the CTS schedule. A short survey of cooling schedules was given [HCH91], and global optimisation methods are surveyed in [SSF92]

A different approach to the Monte Carlo principle is through the stochastic neural

[7]Something is called a *principle* if it involves a change of problem, which cannot be justified on purely logical ground. The reason this is allowed is that the original problem itself is usually an idealisation.

networks (S-NN), such as Boltzmann machine [AHS85, HS86]. A S-NN is a variant GS which, instead of producing a sample in each member of a sequence of distributions approaching the Boltzmann distribution, produces a parameterised (in the form of neural network connection weights) distribution which can be continuously modified. The necessary theoretical tools for dealing with this has been provided by the theory of information geometry (IG), which studies the differential manifold of probability distributions [AKN92, AH89, BNCR86]. The parameterised distributions of a S-NN is restricted to a submanifold of the information manifold. The learning rules of S-NN specify movements in this submanifold.

Closely related to S-NN and SA are the genetic algorithms (GA) [Hol75], a comparison of GA and SA can be found in [Dav87]. Each of these three (S-NN, SA, GA) has an underlying stochastic model described by a Gibbs ensemble which changes with time so that the proportion of good solutions increases. In the original SA method exactly one sample from this ensemble is realized at each step. In S-NN infinitely many samples can be realized, but the distribution is restricted to a submanifold of much lower dimension. The GA realizes as many samples as the population size, and allows unrestricted distribution, but the equilibrium distribution will depend on the representational scheme. It is probably safe to say that in general, the pure SA is most suitable for solving a one-off problem, the learning in S-NN is most suitable for adaptation in a stationary environment, and the GA is capable to adapt to any changing environment, but is not as efficient as learning in a stationary one.

Chapter 3

Adaptive Information Processing

3.1 Introduction

In this chapter we develop the theory of *adaptive information processing*. Our usage of the term *information processing* allows for stochastic systems, and, therefore, is significantly different from that of classical theories [NS72]. In fact our emphasis will be on those abilities of stochastic systems which a deterministic information system, which we call *data processing systems*, can not have.

3.2 Review of Probability Theory

Our references to probability and stochastic processes are [Kol56a, CM65, GS82, GS74].

Definition 3.2.1 Let $[\Omega, \mathcal{F}, \Pr]$ be *probability space*. Let $[A, \mathcal{A}]$ be an arbitrary *measure space*. Then the space of *random objects* in A is defined as

$$A_\Omega := \mathcal{M}[\Omega \to A) := \{f \in [\Omega \to A) : \ f \text{ is measurable}\}.$$

Remark 3.2.1 The couple $[\Omega, \mathcal{F}]$ is usually called a *measure space*. We shall not worry about the measure-theoretical aspect of the probability theory, as we are only interested in probabilities on finite sets.

It is always possible to identify $A \subseteq A_\Omega$. The random objects need not be restricted to real numbers. For example, they can be *random variables* (when $A = \mathbb{R}$ or $\mathbb{C}$), *random vectors* (when $A = \mathbb{R}^n$) or *random mappings* (when $A = [X \to Y)$) where X and Y are measurable spaces. The random mapping is called *random process* when $X = \mathbb{R}, \mathbb{R}_+, \mathbb{Z}$ or $\mathbb{N}$, and *random field* when $X = \mathbb{R}^n$. We use the term "random" interchangeably with "stochastic".

Notation. Let q be a property on X, $x, y \in X_\Omega$, and $\xi, \eta \in X$. Then

$$\Pr\{q(x)\} := \Pr(\{\omega \in \Omega : \ q(x(\omega))\}).$$

$$p_x(\xi) := \Pr\{x = \xi\}. \qquad P_{y|x}(\eta|\xi) := \Pr\{y = \eta | x = \xi\}. \qquad P_f(\eta|\xi) := \Pr\{f(\xi) = \eta\}.$$

These notations will be used throughout the thesis.

Let x, y be random variables. Then the *mathematical expectation* of x is denoted $\langle x \rangle$, while the *covariance* of x, y is denoted $\langle x, y \rangle$. Let C be an *event*. The notation $\langle \cdot \rangle_C$ and $\langle \cdot, \cdot \rangle_C$ denote the mathematical expectation and covariance conditional on C:

$$\langle x \rangle_C := \sum_\xi \Pr\{x = \xi | C\} \xi, \qquad \langle x, y \rangle_C := \langle (x - \langle x \rangle_C)(y - \langle y \rangle_C) \rangle_C .$$

We also define

$$\langle x, y, z \rangle := \langle (x - \langle x \rangle)(y - \langle y \rangle)(z - \langle z \rangle) \rangle .$$

The notation $\langle x \rangle_y$ denote a random variable which is the expectation of x conditional on the value of y:

$$\langle x \rangle_y (\omega) := \sum_\xi \Pr\{x = \xi | y = y(\omega)\} \xi.$$

When considering a stochastic process, the mathematical expectation is also called *ensemble average*, and the notations $[\cdot]$ and $[\cdot, \cdot]$ are used for concepts corresponding to *time average* and its approximation.

The space of random mappings can also be embedded as a subspace of $[X_\Omega \to Y_\Omega)$. For each $f \in [X \to Y)_\Omega$, let $f^* \in [X_\Omega \to Y_\Omega)$:

$$f^*(x) := y : \; y(\omega) = f(x(\omega), \omega).$$

Define $[X \xrightarrow{P} Y) := \{f^* : f \in [X \to Y)_\Omega\}$. We shall identify f^* with f. Intuitively, a stochastic mapping is an operation which maps a random object *randomly* to another random object. It is always possible to identify $[X \to Y) \subseteq [X \xrightarrow{P} Y)$, which is usually called *function of random variables*. It is also quite clear that $y = f(x) \implies \Pr\{y = f(x)\} = 1$. It is also true that $[X \xrightarrow{P} Y) \subseteq [X \to Y_\Omega)$.

Definition 3.2.2 (Independence) Let $[\Omega, \mathcal{F}, \Pr]$ be a probability space, $[A, \mathcal{A}]$ and $[B, \mathcal{B}]$ be two measure spaces, $x \in A_\Omega$, $y \in B_\Omega$, then $\{x, y\}$ is *independent* if

$$\forall X \in \mathcal{A}, Y \in \mathcal{B} : \; \Pr\{x \in X, y \in Y\} = \Pr\{x \in X\} \Pr\{y \in Y\}. \tag{3.2.1}$$

Remark 3.2.2 As noted in [Kol56a, p. 8], from an axiomatic point of view, the theory of probability would be no more than a sub-branch of set theory and measure theory, if not for the additional notion of *independence*, which makes it an entire branch of mathematics [1]. For the concept of random mappings to be useful, we shall need the independence of the function and its operand.

Theorem 3.2.1 *Let X and Y be finite sets, $f \in [X \xrightarrow{P} Y)$, $x \in X_\Omega$, and $y \in Y_\Omega$. If $\{f, x\}$ is independent and $y = f(x)$, then*

$$P_{y|x}(\eta|\xi) = P_f(\eta|\xi).$$

[1] This concept ensures that by statistics of the values of random variables it is in general not possible to determine the identity of the underlying sample, thereby leaving a "random element". Otherwise, as the theory itself is deterministic, as any scientific theory should be, it will never be able to describe truly random phenomenon.

Proof. It follows directly from the definitions that

$$\begin{aligned}\Pr\{y=\eta, x=\xi\} &= \Pr\{\omega:\ f(x(\omega),\omega)=\eta,\ x(\omega)=\xi\}\\ &= \Pr\{\omega:\ f(\xi,\omega)=\eta,\ x(\omega)=\xi\}\\ &= \Pr\{\omega:\ f(\xi,\omega)=\eta\}\Pr\{\omega:\ x(\omega)=\xi\} = P_f(\eta|\xi)P_x(\xi).\end{aligned}$$

□

Remark 3.2.3 In the axiomatic system of probability, two objects are not independent unless explicitly stated so. In the world of electronic computers, however, two variables are not dependent unless explicitly stated. It seems unlikely that there is an expression which reconciles differences between them while remaining concise. We shall sometimes resort to the convention of saying some object x "only depends" on some object y, meaning that x is independent of everything else *mentioned before* except y.

Remark 3.2.4 The concept of random mapping is quite tricky, and has usually been avoided by only considering random variables. It is unavoidable to us, however, since in our computational methods, randomness is actively and intentionally generated, instead of being a result of "uncontrollable factor" in the environment.

Two special examples of stochastic mappings are the mappings of random variables, where the mapping is deterministic, and the stochastic processes (fields), where the operands of the mapping is deterministic. They are automatically independent to their arguments, since either $\Pr\{\omega:\ f(\xi)=\eta\}\in\{0,1\}$ or $\Pr\{\omega:\ x=\xi\}\in\{0,1\}$.

We use the notation "rnd" to denote the stochastic mapping which maps a set randomly to one of its members. Let X be a finite set and $p\in\mathcal{P}(X)$. Then $x:=\mathrm{rnd}(X,p)\in X_\Omega : \Pr\{x=\xi\}=p(\xi)$. When p is omitted uniform distribution is implied, ie., $p(\xi):=1/|X|$. This is not a very rigorous concept so we shall only use it as a short hand notation.

3.3 Information Geometry

We shall use the basic concepts of *differential geometry* as studied in [Die60], including *differentiable manifold*, *coordinate system*, *isomorphism*, *quotient space*. Since this material is standard and our usage is rather intuitive, no review is provided in Appendix A. Intuitively, a manifold is a "smooth curve", or "smooth body", etc., depending on its dimension, but it is not necessarily a *submanifold* in the Euclidean space of the same dimension. Introduction to the theory of information geometry can be found in [AKN92], but our notation is somewhat different.

3.3.1 Information manifolds

Let Ω be a sample space, $X=\{x_1,\ldots,x_n\}$ be a finite set. Define $\mathcal{P}(X):=\{p_x : x\in X_\Omega\}$, the set of all the probability distributions on X. Let $\mathcal{P}(X)\cong\Delta^{n-1}$ by $p_i=p_x(x_i)$, where $p\in\Delta^{n-1}$ and $x\in X_\Omega$. In particular, let $\mathcal{P}(\mathbb{N}_n)\cong\Delta^{n-1}$. With differential structure of Δ^{n-1}, $\mathcal{P}(X)$ becomes an $(n-1)$-dimensional differentiable manifold, called the *information manifold* based on X [AKN92] [2]. The *standard differential*

[2] The term *probability manifold* might be more descriptive, but we shall adhere to the standard terminology.

manifold Δ^{n-1} has a role in stochastic computation similar to that of $\mathbb{R}^n$ in numerical algebra. Any $(n-1)$-subvector of p can be used as a coordinate system on Δ^{n-1}, but none of such coordinate systems is symmetric with respect to all x_i.

With $\mathcal{P}(X) \cong \Delta^{n-1}$, each $p \in \Delta_0^{n-1} := \{p : \exists j : p_i = \delta_{ij}\}$ is a *vertex* of Δ^{n-1} and corresponds to a *deterministic distribution* $p_x(x_i) = \delta_{ij}$. The set of all the deterministic distributions is denoted $\mathcal{P}_0(X)$, which can also be naturally identified with X. We say that $\mathcal{P}(X)$ is *spanned* by $\mathcal{P}_0(X)$, since $p_i = \sum_j p_j \delta_{ij}$.

Remark 3.3.1 The manifold $\mathcal{P}(X)$ is actually a *Riemannian manifold* if endowed with the *Fisher information matrix* [Kul59] defining the Riemannian metric [AKN92], by which $\mathcal{P}(X)$ is a curved manifold. The Fisher information matrix is closely related to the concept of covariance, which is of the utmost importance for all the learning rules for stochastic neural networks, and underlies all the "Hebb-like" learning rules. The so-called dually flat property of $\mathcal{P}(X)$ in [AKN92] is also closely related to the P-coordinates and T-coordinates to be defined below. (Compare Theorem 3.4.5 with the properties of Riemannian metric of Δ^{n-1} [AKN92].) We shall not pursue this issue further.

Notation. Let $p, q \in \mathbb{R}^n$. Then

$$p \sim q \;:\Longleftrightarrow\; p_i q_j = p_j q_i, \quad \forall i, j.$$

This notation will be used throughout the thesis.

Since we shall not use the Riemannian properties of $\mathcal{P}(X)$, it is more convenient to regard $\mathcal{P}(X)$ as a simplex embedded in $\mathbb{R}^n$. This enables us to use coordinates which are symmetric to all the x_i.

Definition 3.3.1 Let $P \in \mathbb{R}^n$, $P \sim p$, where $p \in \Delta^{n-1}$. Then P is called a *P-coordinate system* for p.

Any P-coordinate system defines an isomorphism between $\mathcal{P}(X)$ and an $(n-1)$-simplex embedded in $\mathbb{R}^n$.

Definition 3.3.2 Let $T \in \mathbb{R}^n$, $\exp(T) \sim p$, where $p \in \Delta^{n-1}$. Then T is called a *T-coordinate system* for p. The coordinates T_i are called *tendencies*.

Any T-coordinate system defines an isomorphism between $\mathcal{P}(X)$ and a $(n-1)$ dimensional linear subspace of $\mathbb{R}^n$. For any given T-coordinate system, $Z := \sum_i \exp T_i$ is called the *partition function*.

Theorem 3.3.1 *Any T-coordinate system is uniquely determined by the partition function Z.*

$$p_i = \exp T_i / Z, \qquad \forall i. \tag{3.3.1}$$

Definition 3.3.3 The *Kullback separation* (also called *information divergence*) of the probabilities $p, q \in \Delta^{n-1}$ is

$$K(p,q) := \sum_i p_i \log \frac{p_i}{q_i}.$$

The Kullback separation is a measure of difference between two probability distributions in the information theory [Kul59]. It is not symmetric between the two distributions, but otherwise satisfies the axioms of a pseudo-metric: $K(p,q) \geq 0$, $K(p,q) = 0 \iff p = q$ [AKN92].

Notation. Let $p \in \Delta^{n-1}$, and $\forall k \in \mathbb{N} : p^k \in \Delta^{n-1}$. Then

$$p^k \to p :\iff K(p^k, p) \to 0. \tag{3.3.2}$$

3.3.2 Information mappings

Given two information manifolds $\mathcal{P}(X)$ and $\mathcal{P}(Y)$, a *conditional distribution* is a point $P \in [X \to \mathcal{P}(Y)) =: \mathcal{P}(Y|X)$. It defines an *information mapping* $P^* \in [\mathcal{P}(X) \to \mathcal{P}(Y))$:

$$P^* p = q : \quad q(\eta) = \sum_\xi p(\xi) P(\eta|\xi), \qquad \forall \xi \in X, \ \eta \in Y.$$

It can also be represented by *conditional tendency* $T \in [Y|X \to \mathbb{R})$:

$$\exp T(\cdot|\xi) \sim P(\cdot|\xi), \qquad \forall \xi \in X.$$

The set of all information mappings forms the *information mapping manifold* $\mathcal{P}[X \to Y) := \{P^* \in [\mathcal{P}(X) \to \mathcal{P}(Y) : P \in \mathcal{P}(Y|X))\}$. The submanifold of *deterministic information mappings* from X to Y is defined as

$$\mathcal{P}_0[X \to Y) := \{\pi \in \mathcal{P}[X \to Y) : \ \pi(\mathcal{P}_0(X)) \subset \mathcal{P}_0(Y)\},$$

which is the set of mappings taking deterministic distributions to deterministic distributions. The following identifications are obvious,

$$\mathcal{P}_0[X \to Y) \cong [X \to Y) \subseteq \mathcal{P}(Y|X) \cong \mathcal{P}[X \to Y) \subseteq [\mathcal{P}(X) \to \mathcal{P}(Y)). \tag{3.3.3}$$

The information mapping manifold is closely related to the set of stochastic mappings. If $x \in X_\Omega$ then $p_x \in \mathcal{P}(X)$. If $f \in [X \xrightarrow{P} Y)$ then $P_f \in \mathcal{P}(Y|X) = \mathcal{P}[X \to Y)$.

Definition 3.3.4 (Parameterisation) Let $S \subseteq \mathcal{P}(X)$ and let A be an arbitrary index set, then a surjection $r \in [A \to S]$ is called an *A-parameterisation of S*. Likewise, let $C \subseteq \mathcal{P}[X \to Y)$ and let A be an arbitrary index set, then a surjection $r \in [A \to C]$ is called an *A-parameterisation of C*.

Notation. Given $X \cong \mathbb{N}_n$, called the *state space*. Let $x \in X_\Omega$ be a unique *state variable*, with $p_x = p \in \Delta^{n-1}$. Then each $v \in \mathbb{R}^n \cong [\mathbb{N}_n \to \mathbb{R})$ corresponds to a random variable $v(x) \in \mathbb{R}_\Omega : v(x)(\omega) := v(x(\omega))$. We identify $v(x)$ with v. This notation will be effective throughout the rest of this chapter.

Theorem 3.3.2 *Let $u, v \in \mathbb{R}^n$. Then $\langle v \rangle = \sum_i p_i v_i$, $\langle u, v \rangle = \langle (u - \langle u \rangle)(v - \langle v \rangle) \rangle$. Let $\bar{u}, \bar{v} \in \mathbb{R}$. Then*

$$[u, v] := \langle (u - \bar{u})(v - \bar{v}) \rangle = \langle u, v \rangle + \delta u \delta v, \tag{3.3.4}$$

where $\delta u = \langle u \rangle - \bar{u}$ and $\delta v = \langle v \rangle - \bar{v}$.

Corollary 3.3.3 *For any $\bar{u}, \bar{v} \in \mathbb{R}$:*

$$|[u,v] - \langle u,v \rangle| = |\langle u \rangle - \bar{u}| \; |\langle v \rangle - \bar{v}| \,. \tag{3.3.5}$$

Corollary 3.3.4 *For any $\bar{u}, \bar{v} \in \mathbb{R}$:*

$$\langle u,v \rangle = \langle uv \rangle - \langle u \rangle \langle v \rangle = \langle (u - \bar{u})(v - \langle v \rangle) \rangle = \langle (u - \langle u \rangle)(v - \bar{v}) \rangle \,. \tag{3.3.6}$$

Theorem 3.3.5 *Let $u, v, w \in \mathbb{R}^n$, then the following holds.*

$$\begin{aligned} \langle u,v,w \rangle &= \langle (u - \langle u \rangle)(v - \langle v \rangle), w \rangle \\ &= \langle uv - u \langle v \rangle - v \langle u \rangle, w \rangle \\ &= \langle uv, w \rangle - \langle u \rangle \langle v, w \rangle - \langle v \rangle \langle u, w \rangle \\ &= \langle uvw \rangle - \langle u \rangle \langle vw \rangle - \langle v \rangle \langle uw \rangle - \langle w \rangle \langle uv \rangle + 2 \langle u \rangle \langle v \rangle \langle w \rangle \,. \end{aligned} \tag{3.3.7}$$

Any permutation of u, v, w also equals the above.

3.4 Entropy and Simulated Annealing

3.4.1 Entropy, Boltzmann distribution and Gibbs samplers

Definition 3.4.1 Let $p \in \Delta^{n-1}$, then the *entropy* H of p is

$$H := -\langle \log p \rangle = -\sum_i p_i \log p_i.$$

Example 3.4.1 Let $n \in \mathbb{N}$. $\forall k \in \mathbb{N}_n$ let $p_k = 2^{-k}$, and $p_{n+1} = 2^{-n}$. Then $H = 2 - 2^{1-n} \in [1,2]$. Therefore a non-degenerate (ie. nowhere zero) distribution on a large set may still have a small entropy.

Remark 3.4.1 The convention of signs in physics results in extra minus signs in most formulae we shall study. For convenience, we shall use concepts which correspond to the negative of variables in physics, such as energy, entropy, free energy, etc. We shall often refer to them without the phrase "negative". Care should be taken, however, when one tries to transfer the conclusions directly from texts on statistical mechanics.

Definition 3.4.2 (Boltzmann distribution) Let $e \in \mathbb{R}^n$, $\mu \in \mathbb{R}$. Then

$$BD(e, \mu) := p \in \Delta^{n-1} : \; p \sim \exp(e\mu)$$

is called the *Boltzmann distribution* corresponding to $[e, \mu]$, and $Z := \sum_i \exp(e_i \mu)$ is called the corresponding *partition function*.

Remark 3.4.2 Any $p \in \mathbb{R}^n$ is the Boltzmann distribution corresponding to many different $[e, \mu]$, such as $[T/\mu, \mu]$, where T is an arbitrary T-coordinate. The two names "Boltzmann distribution" and "Gibbs distribution" seems equally popular, the historical origin is from statistical mechanics [LL80].

Remark 3.4.3 By definition, the computation of Z involves enumeration of all the states, which is generally intractable, see [GG84, p. 725].

Definition 3.4.3 (Gibbs sampler) Let $X \cong \mathbb{N}_n$, $p \in \Delta^{n-1}$. Then

$$GS(p) := \left\{\tau \in [\mathbb{N} \to [X \xrightarrow{P} X)) : \forall x \in X_\Omega : p^k := p_{\tau^k(x)} \to p,\ (k \to \infty)\right\},$$

where $\tau^0(x) := x$, $\tau^k(x) := \tau_k(\tau^{k-1}(x))$. Any stochastic mapping process $\tau \in GS(p)$ is called a *Gibbs sampler* corresponding to p.

3.4.2 Maximum Entropy Principle and characterisation of Boltzmann distribution

Consider the following problem:

> Given $e, g \in \mathbb{R}^n = [\mathbb{N}_n \to \mathbb{R})$ and $\langle e \rangle = e^0 \in \mathbb{R}$, where $p \in \Delta^{n-1}$ is unknown, find $\langle g \rangle$.

This problem is under-determined, since there are many different p satisfying the condition. The *Maximum Entropy Principle* (MAXENT) [Jay57a] is the transformation of the above problem into the following well defined problem

$$p : \max H : \langle e \rangle = e^0,\ \langle \mathbf{1} \rangle = 1. \tag{3.4.1}$$

setting the solution as $\langle g \rangle := \sum_i p_i g_i$.

The problem (3.4.1) is a *constrained linear optimisation problem*. It can be solved by the *Lagrange multiplier method*: as a *stationary point* of the *Lagrangian*

$$L := H + \lambda\left(\langle \mathbf{1} \rangle - 1\right) + \mu\left(\langle e \rangle - e^0\right).$$

The stationary points can be found by

$$\frac{\partial L}{\partial p_i} = 0 \implies p_i = \exp\left(\mu e_i - 1 + \lambda\right) = \exp(\mu e_i)/Z,$$

where $Z := \exp(1 - \lambda)$.

$$\frac{\partial L}{\partial \lambda} = 0 \implies \langle \mathbf{1} \rangle = 1 \implies Z = \sum_i \exp(\mu e_i),$$
$$\frac{\partial L}{\partial \mu} = 0 \implies \mu : \langle e \rangle = e^0.$$

Denote the corresponding variables in the MAXENT solution by a superscript 0. Then $p^0 = BD(e, \mu^0)$. For fixed $e \in \mathbb{R}^n$, the solution p^0 is determined by any of the following: e^0, μ^0, λ^0, H^0, Z^0, provided that they are in the range of a possible solution.

Remark 3.4.4 The solution of (3.4.1) exists if and only if $\min e \le e^0 \le \max e$. In fact, in a typical example in §7.3.2, $e^0 = \tanh \mu$, if e is so normalised that $\max e = 1$ and $\min e = -1$.

Assumption 3.4.4 The MAXENT has no constraint on the sign of μ. By possibly changing the sign of e_i, it is always possible to make $e^0 \ge \bar{e} := \sum e_i/n$, and hence $\mu \ge 0$. We shall always assume this is so from now on.

Theorem 3.4.1 *Let $e \in \mathbb{R}^n$, $\mu \in \mathbb{R}$, $p, p' \in \Delta^{n-1}$, $p' = BD(e, \mu)$ with partition function Z', and $f := e\mu - \log p$. Then*

$$\langle f \rangle = -K(p, p') + \log Z'. \tag{3.4.2}$$

Notation. In the rest of this chapter, assume $e \in \mathbb{R}^n$, $\mu \in \mathbb{R}_+$, $p \in \Delta^{n-1}$, $S := -H = \sum_i p_i \log p_i$, $f := e\mu - \log p$.

Theorem 3.4.2 *The following holds:*

1. *(Maximum entropy) Suppose $\bar{e} \le e^0 < \max e$. Then*

$$\max H : \langle e \rangle = e^0, \ \langle \mathbf{1} \rangle = 1 \iff p = p^0.$$

2. *(Minimum energy) Suppose $\log n \ge H^0 > 0$. Then*

$$\max \langle e \rangle : \ H = H^0, \ \langle \mathbf{1} \rangle = 1 \iff p = p^0.$$

3. *(Minimum free energy) Suppose $0 \le \mu^0 < \infty$. Then*

$$\max \langle e \rangle \mu + H : \ \mu = \mu^0, \ \langle \mathbf{1} \rangle = 1 \iff p = p^0.$$

This theorem is well known [Jay57a]. It can be proved by directly solving these optimisation problems, by using duality theorems in the theory of convex optimisation, or by translating the corresponding results from statistical mechanics [LL80, Ami89].

The above characterisation of the Boltzmann distribution has several different interpretations, all of them based on the fact that entropy is the measure of disorder, or missing information. The following two are most interesting to us.

Example 3.4.2 (Statistical Physics) Consider a physical system having *micro states* $i \in \mathbb{N}_n$. Regard $-e_i$ as the *energy*, $\theta := 1/\mu$ as *temperature*, and $-f_i$ as the *free energy*. Then the corresponding *macro state* variables are energy $-\langle e \rangle$, temperature θ, free energy $-\langle f \rangle$ and entropy $-\langle \log p \rangle$. The Boltzmann distribution p^0 is the equilibrium distribution in the *Gibbs canonical ensemble* [LL80, Ami89].

Example 3.4.3 (Decision theory) Suppose an agent is to choose actions $a \in \mathbb{N}_n$ with distribution $p \in \Delta^{n-1}$. Let $e \in \mathbb{R}^n$ be an *evaluation function* on $\mathbb{N}_n$. Then the expected return is $\langle e \rangle$. If e_i is only known after action i is chosen, and if such choices are to be repeated for sufficiently large number of trials, then it is worthwhile to explore the options. Consider H as a measure of diversity, and θ as *unit value of information*. Then the *compound evaluation* $\langle f \rangle$ is the combined return from the gain in the expected evaluation and the gain of information valued by θ.

Remark 3.4.5 We shall use $\mu = 1/\theta$ more often than θ. This is not only because many formulae are much simpler, but also because the solution to the above problems has a singularity at $\theta = 0$, but not at $\mu = 0$.

3.4.3 The Entropy Principle, the Monte Carlo Principle and simulated annealing method

There are two more steps needed to transform the above theoretical results into powerful computational methods. The first is the Entropy Principle and the second is the Monte Carlo Principle.

Given $e \in [\mathbb{N}_n \to \mathbb{R})$. Consider the *discrete optimisation problem*

$$\max e_i : \; i \in \mathbb{N}_n. \tag{3.4.3}$$

The solution set is $\mathbb{N}_n^* := \{i \in \mathbb{N}_n : \; e_i = \max e\}$. Each $i \in \mathbb{N}_n^*$ is called a "ground state" in physics. In general, finding a solution of (3.4.3) requires the enumeration of $\mathbb{N}_n$. Now consider the related *linear optimisation problem*

$$\max \langle e \rangle : \; p \in \Delta^{n-1}. \tag{3.4.4}$$

The solution set is $\mathcal{P}^* := \{p \in \Delta^{n-1} : \; p_i > 0 \implies e_i = \max e\}$, the set of distributions on $\mathbb{N}_n^*$. These two problems are equivalent in the sense that $i \in \mathbb{N}_n^* \iff \exists p \in \mathcal{P}^* : \; p_i > 0$. Since $\mathbb{N}_n \cong \Delta_0^{n-1}$, $\mathcal{P}^*$ is the *convex hull* of $\mathbb{N}_n^*$.

Definition 3.4.5 (Entropy Principle) The *Entropy Principle* is the transformation of problem (3.4.4) into the following associated problem

$$\forall \mu \in \mathbb{R}_+ : \; \max \langle e \rangle \mu + H : \; p \in \Delta^{n-1}. \tag{3.4.5}$$

Remark 3.4.6 This can be interpreted as the *barrier function method* of solving *constrained optimisation problem* [Fle87], where H acts as the *barrier function*. The entropy is a very good barrier function: it is symmetric, finite, and convex, and it has infinite derivative at the boundary of the simplex [Sha48, Kul59]. In many aspects it can be regarded as the optimal one. (See [Jay57a] for similar ideas from a slightly different point of view.)

Theorem 3.4.3 *Denote the solution of (3.4.5) as p^μ, then as $\mu \to \infty$, $p^\mu \to p^* \in \mathcal{P}^*$: $p_i^* = 1/m$ if $i \in \mathbb{N}_n^*$ and $p_i^* = 0$ otherwise, where $m := |\mathbb{N}_n^*|$. That is, p^* is the uniform distribution on $\mathbb{N}_n^*$.*

Although exact solution of (3.4.5) is at least as difficult as that of (3.4.3), the solution process (3.4.5) provides approximate solutions for each μ.

Remark 3.4.7 The transform from exact to approximate solution cannot be justified on logical grounds. It is only valid when it is allowed in the real problem. Sometimes such approximate solutions are even more important than the exact solution, especially for most practical problems where (3.4.3) is only an idealisation.

The Entropy Principle effectively transforms a global optimisation problem on a finite set into a sequence of convex optimisation problem on a simplex. This may not seem a good transformation (the simplex method in linear optimisation uses almost exactly the opposite procedure), but here comes the most important advantage of the Entropy Principle: $p \in \Delta^{n-1}$ can be interpreted as a probability distribution, so it need not be explicitly computed.

Definition 3.4.6 (Monte Carlo Principle) The *Monte Carlo Principle* is the transformation of the problem

$$\text{Find } p \in \Delta^{n-1} : C(p) \tag{3.4.6}$$

into the problem

$$\text{Find } x \in X_\Omega : C(p_x), \tag{3.4.7}$$

where $X \cong \mathbb{N}_n$ and C in a certain condition on Δ^{n-1}.

Definition 3.4.7 (Simulated annealing) Let $e \in \mathbb{R}^n$, $\tau \in GS(BD(e, \mu))$, $\mu^0 \in \mathbb{R}_+ \cup \infty$. The following random process is called a *simulated annealing method* (SA) corresponding to $[e, \mu^0]$, denoted $SA(e, \mu^0)$.

1. Set $\mu := 0$, $x^0 = \mathrm{rnd}(\mathbb{N}_n)$.
2. Set $x^{k+1} := \tau_k(x^k)$.
3. Iterate from step 2, until $p^k := p_{x^k} \approx BD(e, \mu)$.
4. Increase μ, $\mu < \mu^0$.
5. Iterate from step 2, until $\mu \approx \mu^0$.

There are actually two variants of SA: The *infinite SA method* ($SA0$) takes $\mu \to \infty$, which gives an efficient combinatorial optimisation method, as in [KGV83]. The *finite SA method* ($SA1$) takes $\mu \to \mu^0 < \infty$, which gives a method of accelerating a Gibbs sampler, as in [AHS85].

Theorem 3.4.4 *In the $SA0$ method, $p^k \to p^*$. In the $SA1$ method, $p^k \to BD(e, \mu^0)$.*

Proof. The solution sequence x^k produced from steps 2–3 in the SA method satisfy $p^k \to p^\mu$. For $SA0$, $p^\mu \to p^*$, hence $\langle e(x^k) \rangle \to \max e$. For $SA0$, $p^\mu \to BD(e, \mu^0)$. □

Remark 3.4.8 To use a SA method, one has to accept the Entropy Principle, the Monte Carlo Principle, and provide a practical Gibbs sampler. The SA0 method provides a sequence of "approximate" solutions to the optimisation problem without the need of enumerating all the possible solutions, but the "approximation" is in the sense of probability, not in the numerical sense, ie., $e(x^k)$ does not necessarily increase with k.

Remark 3.4.9 Both SA0 and SA1 are quite standard and are important in our later considerations. As computational methods, they must be regarded as separate methods, since transformation from one to the other involves singularity.

3.4.4 Parameterised distributions

In many applications, it is not possible to find an exact GS for arbitrarily given $[e, \mu]$. One has to use a parameterised GS which can approximate the required GS by modifying the parameters. This is so for NN learning rules.

Definition 3.4.8 (Parameterised Gibbs sampler) Let $X \cong \mathbb{N}_n$, $W = \mathbb{R}^M$, $p \in [W \to \Delta^{n-1})$. Then

$$GS_W(p) := \left\{ \tau \in [\mathbb{N}|W \to [X \xrightarrow{P} X)) : \forall w \in W : \tau_w \in GS(p_w) \right\}.$$

Any $\tau \in GS_W(p)$ is called a *parameterised Gibbs sampler* (PGS) corresponding to p.

Let $f := e\mu - \log p$. It follows Theorem 3.4.1 that step 2 in SA method can be characterised by $\langle f(x^k) \rangle \to \max_\xi f(\xi)$. This suggests how SA can be adapted to use PGSs.

Remark 3.4.10 The gradient-following (GF) method of increasing $\langle e \rangle$ is to set $\Delta w \sim \partial \langle e \rangle / \partial w$. The *entropy principle* (EP) requires $\Delta w \sim \partial \langle f \rangle / \partial w$, while increasing μ slowly enough so that the minimum of $\langle f \rangle$ is almost reached at each step.

Definition 3.4.9 (Parameterised simulated annealing) Let $e \in \mathbb{R}^n$, $W = \mathbb{R}^M$, $p \in [W \to \Delta^{n-1})$, $\tau \in GS_W(p)$, $\mu^0 \in \mathbb{R}_+ \cup \infty$. The following random process is called a *parameterised simulated annealing method* (PSA) corresponding to $[e, \mu^0]$, denoted $SA_W(e, \mu^0)$.

1. Set $\mu := 0$, $w = 0$, $x^0 = \mathrm{rnd}(\mathbb{N}_n)$.
2. Set $x^{k+1} := \tau_k(x^k)$.
3. Iterate from step 2, until $p^k := p_{x^k} \approx BD(e, \mu)$.
4. Modify w to increase $\langle f \rangle$.
5. Iterate from step 2, until $\langle f \rangle \approx \max_w \langle f \rangle$.
6. Increase μ, $\mu < \mu^0$.
7. Iterate from step 2, until $\mu \approx \mu^0$.

The following theorems are related to the PSA method. They are of fundamental importance to learning rules of stochastic neural networks.

Theorem 3.4.5 *Let $p \in \Delta^{n-1}$, $T \in \mathbb{R}^n$, $p \sim \exp T$. Then*

$$\frac{\partial p_i}{\partial T_j} = (\delta_{ij} - p_j) p_i = \frac{\partial p_j}{\partial T_i}. \tag{3.4.8}$$

For any $u, v \in \mathbb{R}^n$:

$$\sum_{ij} (\delta_{ij} - p_j) p_i u_i v_j = \langle u, v \rangle . \tag{3.4.9}$$

Proof. It follows (3.3.1) and the chain rule that

$$\begin{aligned} \frac{\partial p_i}{\partial T_j} &= \frac{1}{Z^2} \left(\frac{\partial e^{T_i}}{\partial T_j} Z - e^{T_i} \frac{\partial Z}{\partial T_j} \right) \\ &= \frac{1}{Z^2} \left(e^{T_i} \delta_{ij} Z - e^{T_i} \sum_k e^{T_k} \delta_{jk} \right) = p_i \delta_{ij} - p_i p_j . \end{aligned}$$

The identity (3.4.9) follows (3.3.6):

$$\sum_{ij}(\delta_{ij} - p_j)p_i u_i v_j = \langle u(v - \langle v \rangle)\rangle = \langle u, v \rangle .$$

□

Corollary 3.4.6 *There is a function F such that $p_i = \partial F/\partial T_i$. By integration it follows that $F = \log Z$.*

Notation. In the following three theorems, $M \in \mathbb{N}$, $W = \mathbb{R}^M$, and $\partial_k := \partial/\partial w_k$.

Theorem 3.4.7 *Let $T \in C^1[W \to \mathbb{R}^n)$, $p \in C^1[W \to \Delta^{n-1})$, and $\exp T \sim p$. Let $\mu \in \mathbb{R}_+$, $e \in \mathbb{R}^n$. Denote $f = e\mu - \log p$, $S := -H = -\sum_i p_i \log p_i$. Then*

$$\partial_k \langle e \rangle = \langle \partial_k T, e \rangle , \qquad \partial_k S = \langle \partial_k T, T \rangle , \tag{3.4.10}$$

$$\partial_k \langle f \rangle = \langle \partial_k T, e\mu - T \rangle . \tag{3.4.11}$$

Proof. For any $v \in \mathbb{R}^n$,

$$\sum_i \partial_k p_i v_i = \sum_i \sum_j \frac{\partial p_i}{\partial T_j} \partial_k T_j v_i = \sum_i \sum_j (\delta_{ij} - p_j)\, p_i \partial_k T_j v_i = \langle \partial_k T, v \rangle .$$

Therefore,

$$\partial_k \langle e \rangle = \sum_i \partial_k p_i e_i = \langle \partial_k T, e \rangle ,$$

and

$$\partial_k S = \sum_i \partial_k p_i \log p_i + \sum_i \partial_k p_i = \sum_i \partial_k p_i \log p_i = \langle \partial_k T, T - \log Z \rangle = \langle \partial_k T, T \rangle .$$

So,

$$\partial_k \langle f \rangle = \langle \partial_k T, e\mu - T \rangle .$$

□

Theorem 3.4.8 *Let $p \in C^1[W \to \Delta^{n-1})$, $v \in \mathbb{R}^n$. Then*

$$\left\langle \frac{\partial \log p}{\partial w} v \right\rangle = \frac{\partial \langle v \rangle}{\partial w}. \tag{3.4.12}$$

Proof. By direct calculation.

$$\left\langle \frac{\partial \log p}{\partial w} v \right\rangle = \sum_i p_i \frac{\partial \log p_i}{\partial w} v_i = \sum_i \frac{\partial p_i}{\partial w} v_i = \frac{\partial \langle v \rangle}{\partial w}.$$

□

Theorem 3.4.9 *Let $T \in C^2[W \to \mathbb{R}^n)$, $p \in C^2[W \to \Delta^{n-1})$, and $\exp T \sim p$. Let $v \in \mathbb{R}^n$. Then*

$$\partial_k \partial_j \langle v \rangle = \langle \partial_k T, \partial_j T, v \rangle + \langle \partial_k \partial_j T, v \rangle . \tag{3.4.13}$$

Proof. Apply Theorem 3.4.7 twice, we get

$$\begin{aligned}\partial_k \partial_j \langle v \rangle &= \langle \partial_k T, (\partial_j T - \langle \partial_j T \rangle)\, v \rangle + \langle - \langle \partial_k T, \partial_j T \rangle\, v \rangle + \langle \partial_k \partial_j T, v \rangle \\ &= \langle \partial_k T, \partial_j T v - \langle \partial_j T \rangle\, v - \partial_j T \langle v \rangle \rangle + \langle \partial_k \partial_j T, v \rangle \\ &= \langle \partial_k T, \partial_j T, v \rangle + \langle \partial_k \partial_j T, v \rangle .\end{aligned}$$

□

Definition 3.4.10 In stochastic computations involving GSs, a variable is called *locally computable* if it is either (1) known explicitly, (2) a deterministic function of a locally computable variable, or (3) an average of locally computable variables whose variation is at most polynomial in T_i.

Remark 3.4.11 Usually T_i is explicitly known, but p_i is not locally computable. So $\exp T_i$ and $\langle T \rangle$ are locally computable, but Z is not locally computable, although $1/\log Z = \langle \exp(-T_i) \rangle$. This is because p_i is inversely proportional to $\exp(-T_i)$, so that if this formula is to be used to compute Z using the Monte Carlo Principle, it is necessary to sample almost all $i \in \mathbb{N}_n$ before it is reasonably accurate. Since $\log p_i = T_i - \log Z$, with T_i known, the computation of $\log p_i$ is as difficult as that for Z. It is a remarkable fact that the above theorems are valid for any tendency T_i, not just $\log p_i$. This is not so for Markov chains, as we shall see in the following sections.

3.5 Markov Chains and Parameterised Markov Chains

Markov chains (MC) have been studied from many different perspectives. Since we are interested in how a Markov chain might be changed, we shall use the terminology of information mappings. We shall only consider Markov chains on finite state spaces.

3.5.1 Markov chains

Definition 3.5.1 A *Markov chain of infinite length* is a couple $[X, x]$, where $X = [X_k : k \in \mathbb{N}]$, $x = [x_k : k \in \mathbb{N}]$, $\forall k \in \mathbb{N} : x_k \in (X_k)_\Omega$, with the *Markov property*

$$\Pr\{x_k | x_j : j < k\} = \Pr\{x_k | x_{k-1}\}, \qquad \forall k \in \mathbb{Z}_+. \tag{3.5.1}$$

A *Markov chain of finite length* $n \in \mathbb{N}$ is defined by replacing $\mathbb{N}$ with $\mathbb{N}_n$ in the above.

Notation. In this section we do not distinguish between row vectors and column vectors. Let $I, J \subseteq \mathbb{N}$, $\xi \in X$. Then $\xi_I := [\xi_i : i \in I]$. $I < J :\iff \forall i \in I, j \in J : i < j$. If $I \cap J = \emptyset$ then $IJ := I \cup J$.

Definition 3.5.2 Given a MC $[X, x]$. Define the *joint probability* p, the *transition probability* P_k, the *marginal probability* P_I, and the *conditional probability* $P_{I|J}$, respectively, by

$$p(\xi) := \Pr\{x = \xi\}. \tag{3.5.2}$$

$$P_k(\xi_k | \xi_{k-1}) := \Pr\{x_k = \xi_k | x_{k-1} = \xi_{k-1}\}. \tag{3.5.3}$$

$$P_I(\xi_I) := \Pr\{x_I = \xi_I\}. \tag{3.5.4}$$

$$P_{I|J}(\xi_I | \xi_J) := \Pr\{x_I = \xi_I | x_J = \xi_J\}. \tag{3.5.5}$$

Theorem 3.5.1 *Let $[X, x]$ be a MC. Then $\forall \xi \in X$:*

$$\begin{aligned}
p(\xi) &= p_0(\xi_0) \prod_{k>0} P_k(\xi_k|\xi_{k-1}), \\
p_I(\xi_I) &= \sum_{\xi_{\mathbb{N}\setminus I}} p(\xi), \\
P_{I|J}(\xi_I|\xi_J) &= p_{IJ}(\xi_{IJ})/p_J(\xi_J).
\end{aligned}$$

If $j = \max J < I$, then $P_{I|J} = P_{I|j}$. In particular, The Markov property is $P_{k|\mathbb{N}_{k-1}} = P_k$.

Therefore a MC is characterised by a couple $[X, P]$, where $X = [X_k : \ k \in \mathbb{N}]$, $P = [P_k : \ k \in \mathbb{N}]$, and

$$P_k \in \mathcal{P}[X_k|X_{k-1}), \qquad \forall k \in \mathbb{Z}_+.$$

Theorem 3.5.2 *Let $X = [X_k : \ k \in \mathbb{N}]$, $x = [x_k : \ k \in \mathbb{N}]$,*

$$x_0 \in (X_0)_\Omega, \quad \tau_k \in [X_{k-1} \xrightarrow{P} X_k), \quad x_k = \tau_k(x_{k-1}), \qquad \forall k \in \mathbb{Z}_+.$$

Suppose that $\{x_0\} \cup \{\tau_k : \ k \in \mathbb{Z}_+\}$ is independent. Then $[X, x]$ is a MC.

Proof. From definition and the independence condition,

$$\begin{aligned}
\Pr\{x_{\mathbb{N}_k} = \xi_{\mathbb{N}_k}\} &= \Pr\{\omega : \ x_0(\omega) = \xi_0, \forall k : \tau_k(\xi_{k-1}, \omega) = \xi_k\} \\
&= \Pr\{\omega : \ x_0(\omega) = \xi_0\} \prod_k \Pr\{\omega : \ \tau_k(\xi_{k-1}, \omega) = \xi_k\} \\
&= \Pr\{x_0 = \xi_0\} \prod_k P_{\tau_k}(\xi_k|\xi_{k-1}), \qquad \forall \xi \in X.
\end{aligned}$$

It is also obvious that $\{\tau_k, x_{k-1}\}$ is independent. Therefore,

$$\begin{aligned}
\Pr\{x_k = \xi_k | x_{\mathbb{N}_{k-1}} = \xi_{\mathbb{N}_{k-1}}\} &= P_{\tau_k}(\xi_k|\xi_{k-1}) \\
&= \Pr\{x_k = \xi_k | x_{k-1} = \xi_{k-1}\}, \qquad \forall \xi \in X.
\end{aligned}$$

This proves the theorem. □

If all the state spaces X_k are identical, and if all the transition probabilities P_k are identical, the MC is called *homogeneous*, otherwise it is *inhomogeneous*. A MC of finite length can be similarly described, by replacing $\mathbb{N}$ with $\mathbb{N}_n$.

Remark 3.5.1 Although MCs can be studied in their own rights without mentioning information geometry, when one tries to approximate some MC by a sequence of other MCs, it is necessary to put them in a same space in order to study the convergence.

3.5.2 Entropy of a Markov chain

Notation. Denote $\mathbf{k} := \mathbb{N} + k = \{k, k+1, \dots\}$. Let $v \in [X_k \to \mathbb{R})$, then define $\langle v \rangle_{k-1} \in [X_{k-1} \to \mathbb{R})$:

$$\langle v \rangle_{k-1}(\xi_{k-1}) := \sum_{\xi_k} P_k(\xi_k|\xi_{k-1}) v(\xi_k).$$

Define the *total entropy* $S_{\mathbf{k}}$, the *local entropy* S_k, and the *transferred entropy* $\langle S_{\mathbf{k+1}} \rangle_{k-1}$ at step k, respectively, by

$$S_{\mathbf{k}}(x_{k-1}) := \sum_{\xi_{\mathbf{k}}} P_{\mathbf{k}|k-1}(\xi_{\mathbf{k}}|\xi_{k-1}) \log P_{\mathbf{k}|k-1}(\xi_{\mathbf{k}}|\xi_{k-1}), \tag{3.5.6}$$

$$S_k(\xi_{k-1}) := \sum_{\xi_k} P_k(\xi_k|\xi_{k-1}) \log P_k(\xi_k|\xi_{k-1}), \tag{3.5.7}$$

$$\langle S_{\mathbf{k+1}} \rangle_{k-1} (\xi_{k-1}) = \sum_{\xi_k} P_k(\xi_k|\xi_{k-1}) S_{\mathbf{k+1}}(\xi_k). \tag{3.5.8}$$

Remark 3.5.2 These are generalisations of the definitions given in [Khi57]. Entropy of infinite steps may be infinite, in that case, the above formulae are interpreted by substituting $\mathbf{k} := \mathbb{N}_{k,m} = \{k, k+1, \ldots, m\}$, where m is a number larger than any k which is of interest.

Theorem 3.5.3 *The entropy has the following properties.*

$$\langle S \rangle : = \langle S_{\mathbf{1}} \rangle_{-1} = \sum_{\xi_0} p_0(\xi_0) S_{\mathbf{1}}(\xi_0) = S_{\mathbf{0}} - S_0. \tag{3.5.9}$$

$$S_{\mathbf{k}}(\xi_{k-1}) = S_k(\xi_{k-1}) + \langle S_{\mathbf{k+1}} \rangle_{k-1} (\xi_{k-1}) \tag{3.5.10}$$

$$S_{\mathbf{k}}(\xi_{k-1}) = \sum_{\xi_{\mathbf{k}}} P_{\mathbf{k}|k-1}(\xi_{\mathbf{k}}|\xi_{k-1}) \sum_{i=k}^{\infty} \log P_i(\xi_i|\xi_{i-1}), \tag{3.5.11}$$

Proof. Decompose the conditional probability

$$P_{\mathbf{k}|k-1}(\xi_{\mathbf{k}}|\xi_{k-1}) = \sum_{\xi_k} P_k(\xi_k|\xi_{k-1}) P_{\mathbf{k+1}|k}(\xi_{\mathbf{k+1}}|\xi_k),$$

and substitute it into (3.5.6), (3.5.7) and (3.5.8). □

Theorem 3.5.4 *Let* $P_k \in [W \to \mathcal{P}[X_k|X_{k-1})).$ *Then*

$$\frac{\partial S_{\mathbf{k}}(\xi_{k-1})}{\partial w} = \frac{\partial S_k(\xi_{k-1})}{\partial w} + \sum_{\xi_k} \frac{\partial P_k(\xi_k|\xi_{k-1})}{\partial w} S_{\mathbf{k+1}}(\xi_k) + \sum_{\xi_k} P_k(\xi_k|\xi_{k-1}) \frac{\partial S_{\mathbf{k+1}}(\xi_k)}{\partial w}.$$

The term $\partial S_{\mathbf{k}}(\xi_{k-1})/\partial w$ is called the *total entropy gradient*, the term $\partial S_k/\partial w$ is called the *local entropy gradient*, the term $\langle \partial S_{\mathbf{k+1}}(\xi_k)/\partial w \rangle$ is called the *transferred entropy gradient* and the term $(\partial P_k/\partial w) S_{\mathbf{k+1}}$ is called the *shift entropy gradient*.

Theorem 3.5.5 *Let* $[X, x]$ *be a parameterised MC, and* T_k *be corresponding tendencies. That is,* $P_k \sim \exp T_k$, $T_k \in [W \to [X_{k-1} \to \mathbb{R}^{|X_k|})).$ *Then*

$$\frac{\partial S_k(\xi_{k-1})}{\partial w} = \left\langle \frac{\partial T_k(\cdot|\xi_{k-1})}{\partial w}, T_k(\cdot|\xi_{k-1}) \right\rangle.$$

$$\sum_{\xi_k} \frac{\partial P_k(\xi_k|\xi_{k-1})}{\partial w} S_{\mathbf{k+1}}(\xi_k) = \left\langle \frac{\partial T_k(\cdot|\xi_{k-1})}{\partial w}, S_{\mathbf{k+1}}(\cdot) \right\rangle.$$

Definition 3.5.3 Let $[X, x]$ be a MC, $f_k, g_k, h_k, \dots \in [X_k \to \mathbb{R})$. Then the recursive formulae

$$f_k = F_k(f_{k+1}, g_k, h_k, \dots),$$

where F_k is pointwise except possibly involving taking average, is called *backpropagation process*. Denote $f_k \in BP(g_k, h_k, \dots)$. Likewise, denote $f_k \in F(g_k, h_k, \dots)$ if

$$f_k = F_k(g_k, h_k, \dots).$$

Proposition 3.5.6 *Let $T_k \in L[W \to [X_{k-1} \to \mathbb{R}^{|X_k|}))$. Then $S_{\mathbf{k}} \in BP(S_k)$, $\partial S_k/\partial w \in F(T_k)$, $\partial S_{\mathbf{k}}/\partial w \in BP(\partial S_k/\partial w, S_{\mathbf{k+1}})$.*

The total entropy gradient can be computed by a back propagation process through all the local gradients and shift gradients. The local entropy gradient can be computed from any tendency function. The computation of shift entropy gradient require the knowledge of entropy of following stages, which is in general intractable.

3.5.3 Modifying a Parameterized Markov Chain

Consider a parameterized MC $[X, x]$, depending on a parameter w. Let $r = [r_k : k \in \mathbb{N}]$ be a sequence of *reward functions* $r_k \in [X_{k-1} \times X_k \to \mathbb{R})$. Define the *total evaluation* $e_{\mathbf{k}}$ at k, the *local evaluation* e_k and the *transferred evaluation* $\langle e_{\mathbf{k+1}} \rangle_{k-1}$, respectively, by

$$e_{\mathbf{k}}(\xi_{k-1}) := \sum_{\xi_{\mathbf{k}}} P_{\mathbf{k}|k-1}(\xi_{\mathbf{k}}|\xi_{k-1}) \sum_{i=k}^{\infty} r_i(\xi_i|\xi_{i-1}), \tag{3.5.12}$$

$$e_k(\xi_{k-1}) := \sum_{\xi_k} P_k(\xi_k|\xi_{k-1}) r_k(\xi_k|\xi_{k-1}), \tag{3.5.13}$$

$$\langle e_{\mathbf{k+1}} \rangle_{k-1}(\xi_{k-1}) = \sum_{\xi_k} P_k(\xi_k|\xi_{k-1}) e_{\mathbf{k+1}}(\xi_k). \tag{3.5.14}$$

Theorem 3.5.7

$$\langle e \rangle := \langle e_{\mathbf{1}} \rangle_{-1} = \sum_{\xi_0} p_0(\xi_0) e_{\mathbf{1}}(\xi_0), \tag{3.5.15}$$

$$e_{\mathbf{k}}(\xi_{k-1}) = e_k(\xi_{k-1}) + \sum_{\xi_k} P_k(\xi_k|\xi_{k-1}) e_{\mathbf{k+1}}(\xi_k). \tag{3.5.16}$$

So $\langle e \rangle$ is multilinear in the input probability p_0 and the transition probabilities P, but linear in the joint probability distribution p.

Theorem 3.5.8

$$\frac{\partial e_{\mathbf{k}}(\xi_{k-1})}{\partial w} = \sum_{\xi_k} \frac{\partial P_k(\xi_k|\xi_{k-1})}{\partial w} (r_k(\xi_k|\xi_{k-1}) + e_{\mathbf{k+1}}(\xi_k)) + \sum_{\xi_k} P_k(\xi_k|\xi_{k-1}) \frac{\partial e_{\mathbf{k+1}}(\xi_k)}{\partial w}.$$

The first term at right hand side is called *combined local evaluation gradient* (since it is a combination of local gradient and shift gradient) and the second term is called the *transferred evaluation gradient*.

Proposition 3.5.9 *$e_{\mathbf{k}} \in BP(e_k)$, $\partial e_{\mathbf{k}}/\partial w \in BP(r_k, e_{\mathbf{k+1}})$.*

The combined local gradient can be locally computed recursively from the transferred evaluation, which in turn can be locally computed from r_{k+1}. Therefore the total gradient can be computed through a *back-propagation process* (BP) through all the combined local gradient.

The formulae (3.5.10) and (3.5.16) are called *average over states*, while (3.5.11) and (3.5.12) are called *average over histories*. It is well known (eg. [Sut88]), that average over states is in general much more efficient than average over histories for long Markov chains, since the number of histories usually grows in exponential with the length of the chain.

Remark 3.5.3 If k represents layers of a neural network this gives back-propagation learning rule. If k denotes time then (3.5.12) gives *back-propagation through time* (BTT) [Wer90a]. The direct application of BTT for a long temporal sequence is not practical, however, since it is average over histories. The dynamic programming method (DP), which uses averaging over states, is much better, and will be studied in detail in Chapter 8.

The gradient-following (GF) method of improving the evaluation of the (parameterized) Markov chain is to modify the parameter w according to $\Delta w \sim \partial\langle e\rangle/\partial w$. Define

$$f_{\mathbf{k}} = e_{\mathbf{k}} - \theta S_{\mathbf{k}}, \qquad f_k = e_k - \theta S_k.$$

By applying the entropy principle and the Monte Carlo principle we get the SA method which is to modify w according to $\partial\langle f\rangle/\partial w$, while reducing θ slowly enough so that the minimum of $\langle f\rangle$ is almost reached at each step.

There is a significant difference between multistage and single stage stochastic computation. For multistage problem the total entropy gradient involves the shift entropy gradient, which cannot be computed locally. This is a fundamental problem of multistage stochastic adaptive computation. Our treatment of this problem in all the subsequent chapters is simply to discard the shift entropy gradient. We have not found any theoretical justification for doing so, although our numerical experiments do not give any counter-examples (§7.5) in places where one might expect counter-example to occur.

Remark 3.5.4 There may seem to be another way to apply the entropy principle to MC, which is to use each individual P_k, instead of the joint distribution p. However with this formulation the evaluation is *multilinear* in the probabilities, instead of linear. So the equivalence to linear optimisation problem is destroyed.

3.6 Adjustable Information Processors

Definition 3.6.1 A (discrete time) *adjustable information processor* (AIP) is a tuple

$$[T, X, Y, R, W, V, U, \zeta, \chi, \delta, \kappa, \Sigma], \tag{3.6.1}$$

such that

$$
\begin{aligned}
&T = \mathbb{N},\\
&\zeta \in [T \times W \to [X \xrightarrow{P} R)),\\
&\chi \in [T \to [R \xrightarrow{P} Y)),\\
&\delta \in [T \times W \times R \times V \xrightarrow{P} W),\\
&\kappa \in [T \times W \times R \times V \xrightarrow{P} U),\\
&\Sigma \subseteq X_\Omega \times \mathcal{E} \times \Psi,
\end{aligned}
$$

where

$$
\begin{aligned}
\mathcal{E} &:= [T \times \mathcal{P}(X) \times \mathcal{P}(Y|X) \xrightarrow{P} \mathbb{R}),\\
\Psi &:= [T \times X \times Y \times R \times W \xrightarrow{P} V),
\end{aligned}
$$

and satisfying

1. $\forall [x, \varepsilon, \psi] \in \Sigma : \; \{\zeta_t, \chi_t, \delta_t, \kappa_t, x_t, \varepsilon_t, \psi_t : t \in T\}$ is independent.
2. $\zeta_t : t \in T$ have identical distributions. The same is true for ζ substituted by any of χ, δ, κ, x, ε or ψ_t.

The sets T, X, Y, R, W, V and U are called *time domain*, *input space*, *output space*, *internal state space*, *parameter space*, *auxiliary input space* and *auxiliary output space*, respectively. They together forms the *structure of the AIP*.

The mappings ζ, χ, δ and κ are called *processing operator*, *transmission operator*, *development operator* and *auxiliary output operator*, respectively. They together forms the *dynamics of the AIP*.

The sets $\mathcal{E}$, Ψ and Σ are called *evaluation space*, *assistance space* and the *environment space*, respectively. Any $[x, \varepsilon, \psi] \in \Sigma$ is called an *environment*. x is called the *input*, ε the *evaluation operator*, and ψ the *assistance operator*.

The set $[X \xrightarrow{P} R)$ is called the *space of reasonable processing*, and the set $\zeta(W)$ is called the *space of realizable processing*.

For each environment $[x, \varepsilon, \psi] \in \Sigma$, the AIP produces the following stochastic process:

$$
\begin{aligned}
r_t &= \zeta_{t,w_t}(x_t),\\
y_t &= \chi_t(r_t) = \varphi_{t,w_t}(x_t),\\
v_t &= \psi_t(x_t, y_t, r_t, w_t),\\
w_{t+1} &= \delta_t(w_t, r_t, v_t),\\
u_t &= \kappa_t(w_t, r_t, v_t),\\
e_t &= \varepsilon_t(p_x, P_{w_t}),\\
E(w_t) &:= \left\langle \epsilon(p_x, P_{w_t}) \right\rangle,
\end{aligned}
$$

where the *throughput operator* $\varphi \in [T \times W \to [X \xrightarrow{P} Y))$: $\varphi_{t,w} := \chi_t(\zeta_{t,w})$, with conditional probability distribution $P_{t,w} := P_{\varphi_{t,w}} \in \mathcal{P}(Y|X)$.

Notation. Since most of the above refer to the same t, we shall often omit reference to t, while using a plus sign to represent $t+1$ and minus sign for $t-1$. More specifically, the above relations are abbreviated as

$$\begin{aligned} &r = \zeta_w(x), \qquad y = \chi(r) = \varphi_w(x), \qquad \varphi_w = \chi(\zeta_w), \\ &v = \psi(x, y, r, w), \qquad w_+ = \delta(w, r, v), \qquad u = \kappa(w, r, v), \\ &e = \varepsilon(p_x, P_w), \qquad E(w) = \langle \epsilon(p_x, P_w) \rangle . \end{aligned}$$

Definition 3.6.2 An AIP is said to be compatible with an environment $[x, \varepsilon, \psi] \in \Sigma$ if $\langle e_+ \rangle \geq \langle e \rangle$. An AIP is said to be compatible with the environment space Σ if it is compatible with all its members.

Obviously an AIP must be compatible with its environment space to be useful.

Definition 3.6.3 An AIP is said *deterministic* if ζ, χ and δ are so. Otherwise it is said to be *stochastic*. An AIP is said to be *discrete* or *continuous* if W is so.

Notation. The *black-box* *assistance* *space* $\Psi_0 := \{\psi \in \Psi : \psi(x, y, r_1, w_1) = \psi(x, y, r_2, w_2)\} \cong [T \times X \times Y \xrightarrow{P} V)$.

Definition 3.6.4 An AIP is called a *trainable information processor* (TIP) if

$$\forall [x, \varepsilon, \psi] \in \Sigma : \; \exists \psi_0 \in \Psi_0 : \; \langle e_+ \rangle \geq \langle e \rangle \, , \tag{3.6.2}$$

Otherwise it is called a *programmable information processor* (PIP).

A TIP is usually continuous and often stochastic. A PIP is usually discrete and deterministic.

Remark 3.6.1 An AIP is a TIP if and only if its performance can improve even when the assistance from environment regards it as a black-box.

The following terms are usually used. Any $w \in W$ is usually called the *weights* for TIP and *program* for PIP, δ is usually called *learning* for TIP and *operating system* for PIP, ψ is usually called *training* for TIP and *programming* for PIP.

Example 3.6.1 A piece of software in a digital computer (when under development) is a discrete deterministic PIP with its programmer as environment, given the following correspondence: w—program, x—input, r—trace of running the program, y—output, e—evaluation of program, v—intended change to program, u—error signal, ζ—CPU, χ—output devices, δ—operating system, κ—error checking mechanism, ϵ—objective specifications, ψ—programming.

Part II

Neural Networks

Chapter 4

Theory of H. S. Neural Networks

4.1 Introduction

In this chapter we shall develop a mathematical theory of homogeneous semilinear neural networks (H.S.NN), which does not depend on any of its physical interpretation. We start from the formal definition of a somewhat larger class of neural networks, quasilinear neural network (Q.NN), and build mathematical theory of neural network structure, dynamics and learning rules.

Our main theory will be restricted to H.S.NN which will be more than enough for our later considerations. The class of H.S.NN includes some of the most intensively studied and commonly used neural networks, including back-propagation networks, Boltzmann machine, and Hopfield net, and many others.

The theory in this chapter may seem rather abstract, even with our illustrated examples. We postpone until the next chapter a detailed study of the most important specialisations, including the standard NNs and their generalisations.

4.2 Definitions and Basic Properties

The class of quasilinear neural networks (Q.NN) and its subset semilinear neural networks (S.NN) were first formalised in [RHM86]. Our definition is slightly different: In our definitions, the class of Q.NN is somewhat larger while that of S.NN is somewhat smaller than those in [RHM86]. A *quasilinear neural network* (Q.NN) is a trainable information processor with the following structure and dynamics.

4.2.1 Structure

Definition 4.2.1 The *structure* of Q.NN is defined by an *input size*, *output size*, and *network size*,

$$m \in \mathbb{N}, \qquad n \in \mathbb{N}, \qquad N \in \mathbb{N},$$

two *connection sets*

$$I \subseteq \mathbb{N}_m \times \mathbb{N}_N, \qquad J \subseteq \mathbb{N}_N \times \mathbb{N}_N,$$

and two *output selectors* $C, D \in \mathbb{R}^{n \times N}$.

Define the *connection weight spaces*

$$\mathbb{R}_I^{N\times m} := \{A \in \mathbb{R}^{N\times m} : \ [j,i] \notin I \implies A_{ij} = 0\} \cong \mathbb{R}^{|I|},$$

$$\mathbb{R}_J^{N\times N} := \{B \in \mathbb{R}^{N\times N} : \ [j,i] \notin J \implies B_{ij} = 0\} \cong \mathbb{R}^{|J|},$$

Define the *weight space*

$$W := \mathbb{R}^M \cong \mathbb{R}_I^{N\times m} \times \mathbb{R}_J^{N\times N} \times \mathbb{R}^N,$$

where $M := |I| + |J| + N$.

We shall sometimes refer $\mathbb{N}_m$, $\mathbb{N}_N$ and $\mathbb{N}_n$ as the *input set*, *neuron set* and *output set*. Their members are called *input units*, *neurons* and *output units*, respectively. They are collectively known as *units*, or *nodes*. Members of I and J are called *connections*. If $[i,j] \in J$ then i and j are called connected.

Remark 4.2.1 Note that in our formulation, a neuron *is* a number $i \in \mathbb{N}_n$. We do not consider input units as part of the network. Therefore our neuron set is the union of what is commonly called set of "hidden-neurons" and set of "output-neurons". The output selectors are usually simple *projection matrices*, and often one of them is zero. If they correspond to projection of $\mathbb{N}_n$ to a subset of $\mathbb{N}_N$, then this subset can be identified with the output set, and the remaining set of $\mathbb{N}_N$ is called the *hidden neuron set*, its members called *hidden neurons*. These concepts are not applicable to more general cases.

Remark 4.2.2 The set I is composed of connections $[j,i]$ from input unit j to neuron i. Likewise the set J is composed of connections $[j,i]$ from neuron j to neuron i. These connections are directional. The connection sets I and J are relations on $\mathbb{N}_m \times \mathbb{N}_N$ and on $\mathbb{N}_N$, respectively.

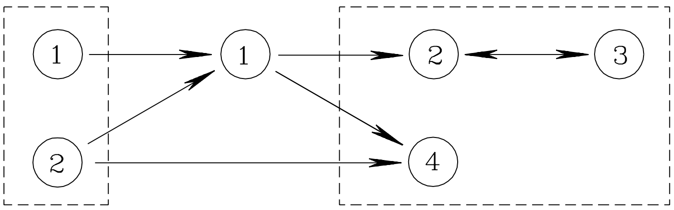

Figure 4.1: Structure of an example neural network.

Nodes in dashed boxes are so-called "input" and "output" neurons.

Example 4.2.1 Consider a simple network as shown in Figure 4.1. We have $m = 2$, $N = 4$. The input set is $\mathbb{N}_2 = \{1,2\}$, the neuron set $\mathbb{N}_4 = \{1,2,3,4\}$, The connection sets are

$$I = \{[1,1],[2,1],[2,4]\}, \qquad J = \{[1,2],[1,4],[2,3],[3,2]\}.$$

If we take $n = 3$ then the output set is $\mathbb{N}_3 = \{1,2,3\}$. The output selectors $C, D \in \mathbb{R}^{3\times 4}$. Suppose that

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad D = 0.$$

Then the output is selected from neurons $\{2,3,4\}$. So the hidden neuron set is $\{1\}$.

4.2.2 Dynamics

Definition 4.2.2 The *dynamics* of a Q.NN is determined by two *weight matrices* $A \in \mathbb{R}_I^{N\times m}$ and $B \in \mathbb{R}_J^{N\times N}$, a *bias vector* $b \in \mathbb{R}^N$, a set of *output ranges* $\mathcal{R} = \{R_i : \ i \in \mathbb{N}_N\}$, where $R_i \subseteq \mathbb{R}$, and an *activation rule* $\widetilde{f} \in \prod_{i\in\mathbb{N}_N}[\mathbb{R} \xrightarrow{P} R_i)$, with conditional probability $P_i \in \mathcal{P}(R_i|\mathbb{R}), \forall i \in \mathbb{N}_N$. Given an *input* $x \in \mathbb{R}^m$ the network generates an *output* $y \in \mathbb{R}^n$ by $y = \varphi(x)$ where the *network dynamics* $\varphi \in [\mathbb{R}^m \xrightarrow{P} \mathbb{R}^n)$ is generically described by the set of equations

$$s^0 := Ax + b \in \mathbb{R}^N, \tag{4.2.1}$$

$$s = Br + s^0, \qquad r = \widetilde{f}(s), \tag{4.2.2}$$

$$y = Cr + Ds. \tag{4.2.3}$$

The vector $s \in \mathbb{R}^N$ is called the *neuronal input vector*. The vector $r \in \prod \mathcal{R}$ is called the *neuronal output vector*. They are collectively called *neuronal states*. The equation $s = Br + s^0$ is called *influence equation* or *propagation equation*. The equation $r = \widetilde{f}(s)$ is called the *activation equation*. They are collectively called the *network equations*.

Remark 4.2.3 A "quasilinear neuron" was schematically depicted in Figure 1.4. A neural network in general is a nonlinear, high dimensional, and often stochastic system. The term "quasilinear" describes the fact that nonlinearity and dimensionality are separate: a high dimensional vector is first linearly combined into a single number and then nonlinearly (and stochastically) transformed. An important property of Q.NN is the so-called $S\Sigma$ exchange property [Gro88]: the network equations are equivalent to either of the following two equations

$$r = \widetilde{f}(Br + s^0), \quad \text{or} \quad s = B\widetilde{f}(s) + s^0.$$

The extended vector $w := [A, B, b] \in W$ is call *weight vector*. Each component of A and B is called a *connection weight*, or *synapse*. The stochastic function $\widetilde{f} \in \prod_i[\mathbb{R} \xrightarrow{P} R_i)$ is a diagonal mapping in $[\mathbb{R}^N \xrightarrow{P} \prod \mathcal{R})$ defined by $(\widetilde{f}(s))_i := \widetilde{f}_i(s_i)$. The function $f_i(s_i) := \langle r_i \rangle_{s_i} = \sum_{r_i} r_i P_i(r_i|s_i)$ is called the *activation function*.

The bias b can be theoretically merged into the connection matrix A [AHS85]. This is done by introducing an imaginary input node 0 which is connected to all the neurons in the network, and extending the matrix A with a column 0, such that $x_0 = 1$ and $\forall i : \ A_{i0} = b_i$. With this formal transform the propagation equation becomes $s^0 = Ax$.

Remark 4.2.4 Some authors use the diagonal of B to represent the bias of the neurons [AK89b]. This is only applicable when the states of the neurons are in $\{0, 1\}$.

The network equations have a recursive form: r depends on s, and vice versa. There are several possible interpretations of the network equations, each of them giving rise to a different dynamics. We concentrate in the so-called attractor dynamics.

Definition 4.2.3 Let $a \in \mathbb{R}^n$. A *sequential updating*

$$\Delta a : \ a = f(a, b), \ b = g(a, b) \tag{4.2.4}$$

is a Markov random process $[a^t, b^t : t \in \mathbb{N}]$ generated by transitions $[i^t, f^t, g^t : t \in \mathbb{N}]$ such that

$$i^t \in (\mathbb{N}_n)_\Omega,$$
$$a_j^t = \begin{cases} f_j^t(a^{t-1}, b^{t-1}), & j = i^t, \\ a_j^{t-1}, & j \neq i^t, \end{cases}$$
$$b^t = g^t(a^t, b^{t-1}).$$

The random process $[i^t : t \in \mathbb{N}]$ is called *update sequence*. If $i^t = \mathrm{rnd}(\mathbb{N}_n)$, (4.2.4) is called *random sequential updating*. For each t, i^t is called the *current unit*.

Definition 4.2.4 For the *attractor dynamics*, φ is defined by the *stationary probability distribution* of the neuronal states. More precisely, for each input x, the network is subject to a sequential updating with the network equations, until the *neuronal distributions* p_r and p_s converge to stationary distributions,

$$p_r(r) = \sum_s p_s(s) \prod_i P_i(r_i|s_i), \qquad p_s(s) = \sum \left\{ p_r(r) : \ s = Br + s^0 \right\}.$$

Then the output y is an arbitrary sample y taken from the distribution

$$p_r(y) = \sum \left\{ p_r(r) : \ y = Cr + Ds, \ s = Br + s^0 \right\}.$$

When $|R| = \infty$, the summation is replaced by integration.

Remark 4.2.5 The opposite of "attractor dynamics" is the so-called *recurrent dynamics*, in which a time sequence of input produces a time sequence of output. We do not study recurrent NN, since the adaptive computer to be studied in Chapter 9 is more powerful than simple recurrent NNs, and it is composed of attractor networks plus a simple delay element.

Remark 4.2.6 Here we are intentionally not very specific about these equations since a rigorous general account is considerably more complicated than required for the two special subsets of networks, the SQ-NN and DC-NN, defined in §4.3, which are all we are interested in.

Example 4.2.2 Continuing with Example 4.2.1 with reference to Figure 4.1. The weight matrices will have the following form.

$$A = \begin{bmatrix} * & * \\ 0 & 0 \\ 0 & 0 \\ 0 & * \end{bmatrix}, \qquad B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ * & 0 & * & 0 \\ 0 & * & 0 & 0 \\ * & 0 & 0 & 0 \end{bmatrix}.$$

The weight vector $w = [A_{11}, A_{12}, A_{42}, B_{21}, B_{41}, B_{32}, B_{23}, b_1, b_2, b_3, b_4]^T$.

4.3 Classifications and Restrictions

In order that a stationary solution to the network equations exists, some conditions must be imposed on the structure and dynamics of the network. For this purpose we give a comprehensive classification of Q.NN, which is more or less in line with the folklore usage of corresponding terms.

4.3.1 Classification of structures

A summary of relations and orders is given in Appendix A. The connection set J induces a pseudo order $\precsim$ on the neuron set, the minimal pseudo order containing J, which further induces an equivalence $\sim$, a strict partial order $\prec$, a partition of $\mathbb{N}_N$ into clusters, and a partial order $\preceq$ on the cluster set, in the standard manner. Intuitively, The relation $i \precsim j$ means that there is a sequence of connections forming a *path* from i to j.

Notation. Connections are also denoted by $(i_1 \to i_2 \to \dots) := \{[i_1, i_2], [i_2, i_3], \dots\}$. Denote

$$(4.3.1)\quad (i \xrightarrow{a} j \xrightarrow{b} k \xrightarrow{c} \dots) :\iff (i \to j \to k \to \dots), (B_{ji} = a),\ (B_{kj} = b),\ \dots).$$

Example 4.3.1 Continuing with Example 4.2.1 with reference to Figure 4.1. The pseudo order $\precsim$ is

$$\{[1,2], [1,3], [1,4], [2,2], [2,3], [3,2], [3,3]\},$$

which can be more compactly expressed as $\{[1, \{2,3\}], [1,4]\}$. The equivalence $\sim$ can be expressed by the partition $\{1, 2, \{3,4\}\}$. The strict partial order $\prec$ is $\{[1,2], [1,3], [1,4]\}$.

Definition 4.3.1 A connection $[i, j] \in J$ is called an *F-connection* (*unidirectional connection*), if $[j, i] \notin J$. If a connection is not unidirectional it is called a *bidirectional connection*. A connection $[i, j] \in J$ is called a *B-connection* (*symmetric connection*), if $B_{ij} = B_{ji}$ and $i \neq j$.

Proposition 4.3.1 *Symmetric connections are bidirectional, but the converse is not true.*

Definition 4.3.2 A Q.NN is said to be *feedforward*, denoted by prefix F-, if

$$\forall i :\ [i, i] \notin J,\ [i]_\sim = i.$$

If a Q.NN is not feedforward it is called *recursive*.

Remark 4.3.1 The term "recursive" describes the *global structure* of a network. It should be distinguished from the term "recurrent" which describes the *global dynamics* of a network. In general a recurrent network is recursive. The converse is not true. It should be noted that the two terms have been used in the literature interchangeably to mean either concepts, or even both.

Definition 4.3.3 A Q.NN is called *layered* if the neuron set $\mathbb{N}_N$ can be decomposed into an ordered sequence of disjoint subsets $\{L_k :\ k \in \mathbb{N}_L\}$, each of them called a *layer*, such that

$$[i, j] \in J \implies \exists k, l \in \mathbb{N}_L :\ k < l,\ i \in L_k,\ j \in L_l.$$

It is called *simple-layered* if

$$[i, j] \in J \implies \exists k \in \mathbb{N}_{L-1} :\ i \in L_k,\ j \in L_{k+1}.$$

Theorem 4.3.2 *A Q.NN is F-Q.NN if and only if the pseudo order $\precsim$ is a partial order, if and only if $\precsim$ can be extended to an order, and if and only if the network is layered.*

Proof. The first two assertions are properties of the corresponding relations. A Q.NN where J can be extended to an order is obviously layered, where each neuron can be taken as a layer. The converse is also true since by arbitrarily ordering neurons in each layer, a layered network is also ordered. □.

Example 4.3.2 Consider the F-Q.NN described by two segments $J = (1 \to 2 \to 3 \to 4) \cup (1 \to 5 \to 4)$, where the notation is obvious. Any one of the following three orders $[1,2,3,5,4], [1,2,5,3,4]$, and $[1,5,2,3,4]$ contains J. So the extensions of J is not unique. This means that the concept of layer is not intrinsic. This F-Q.NN is not simple-layered. In any pseudo order containing J, the four neurons $\{1,2,3,4\}$ must be in four different layers. There is no way to fit neuron 5 into the chain $[1,2,3,4]$, either as a separate layer, or as part of a layer, so as to make a simple-layered network.

Definition 4.3.4 A F-Q.NN is called *fully connected* if

$$[i,j] \in J \iff \exists k,l \in \mathbb{N}_L : \ k < l, \ i \in L_k, \ j \in L_l.$$

Similarly, a simple-layered F-Q.NN is called fully connected if

$$[i,j] \in J \iff \exists k \in \mathbb{N}_{L-1} : \ i \in L_k, \ j \in L_{k+1}.$$

Such networks can be described by $N_0\text{-}\cdots\text{-}N_L$ where $N_k = |L_k|$, and $L_0 := \mathbb{N}_m$.

Remark 4.3.2 The concept of fully connectedness depends on the network type.

Definition 4.3.5 A Q.NN is called *feedback*, denoted by prefix B-, if the matrix B is symmetric with zero diagonal, ie.,

$$(\forall i : \ [i,i] \notin J), \quad (\forall i,j : \ B_{ij} = B_{ji}).$$

Remark 4.3.3 A Q.NN is a B-Q.NN if and only if all its connections are B-connections. In an F-Q.NN all the connections are F-connections, but the converse is not true. Unlike B-structure, F-structure is not determined by local properties. We say that the F-property is global while the B-property is local.

Example 4.3.3 Consider the network with structure $J = (1 \to 2 \to 3 \to \cdots \to N \to 1)$. The fact that this is not an F-structure can only be known by examining all the connections.

Remark 4.3.4 B-Q.NN is only a small subset of all the recursive Q.NN.

Definition 4.3.6 A Q.NN is called *feedforward-feedback*, denoted by prefix FB-, if

$$(\forall i : \ [i,i] \notin J), \quad (\forall i,j : \ i \sim j \implies B_{ij} = B_{ji}).$$

Proposition 4.3.3 *The cluster set of a FB-Q.NN is layered.*

Definition 4.3.7 A FB-Q.NN is called *simple-layered* if the cluster set is simple-layered in the sense of F-Q.NN.

Remark 4.3.5 Recall that a FB-Q.NN was depicted in Figure 1.2 & 1.3. One layer of FB-Q.NN is also a B-Q.NN (Figure 1.3).

Example 4.3.4 Continuing with Example 4.2.1, with reference to Figure 4.1. All the connections are F-connections except $[2,3]$ and $[3,2]$. If $B_{23} = B_{32}$ then this connection is a B-connection. The network is neither a F-Q.NN nor a B-Q.NN. It is an FB-Q.NN if and only if the cluster $\{2,3\}$ can itself be regarded as a B-Q.NN, ie., $B_{23} = B_{32}$.

4.3.2 Classification of dynamics

Definition 4.3.8 A Q.NN is called a *continuous Q.NN*, denoted C-Q.NN, if each R_i is an interval. It is called a *quantized Q.NN*, or *discrete Q.NN*, denoted Q-Q.NN, if each R_i is a discrete set. It is called a *binary Q.NN* if each R_i contains exactly two members.

Remark 4.3.6 C-Q.NN and Q-Q.NN only covers a subset of possible networks. We do not study mixed networks. On the other hand, Q-Q.NN can be regarded as limit of C-Q.NN in many aspects, as we shall see shortly in §4.4.

Definition 4.3.9 If the dependence of r_i on s_i is deterministic for each i, ie., if $r_i = f_i(s_i)$, the network is called a *deterministic Q.NN*, denoted D-Q.NN. Otherwise the neural network is called a *stochastic Q.NN*, denoted S-Q.NN.

Remark 4.3.7 Note that the above definition depends only on local activation rules, not on the manner such activation rules are utilized globally. This means, for example, we classify the (discrete) Hopfield net [Hop82] as DQB-Q.NN, since in it each neuron is activated deterministically, although it uses the random sequential updating.

Remark 4.3.8 The DCB-NN discussed in [Hop84, Alm87] are slightly more general, using $f_i(s_i) = \tanh(\mu_i s_i)$ as activation function, or, equivalently use $B_{ij}\mu_j =: B'_{ij}$ as connection weight. Since we are only interested in homogeneous networks, and since there seems to be no effective and significant application of this property, we have defined B structure and FB-structure as above, but they are easily extended in this direction.

4.3.3 Restrictions

Definition 4.3.10 A Q.NN is called a *semilinear neural network* (S.NN) if for all i, (1) P_i is continuous in s_i, (2) the activation function f_i is monotonic and differentiable, and (3) the output range R_i is bounded.

Definition 4.3.11 A Q.NN is said *homogeneous*, denoted by the prefix H., if $R_i = R \subseteq \mathbb{R}$ and $P_i = p \in \mathcal{P}(R|\mathbb{R})$, independent of i.

Remark 4.3.9 The concept of homogeneity seems have not appeared in the literature. This concepts is very important for us, however, since it greatly assists the analysis of the network, while at the same time does not restrict the performance of the network. Minor generalization to non-homogeneous networks are possible, and will be noted when appropriate, but we shall not go into this detail without proper reason.

Notation. We use the following form of notation to denote all the combinations

$$\{D, S\}\{C, Q\}\{F, B, FB\} - \{H.\}\{Q., S.\}NN,$$

where zero or one symbol from each set is chosen.

Assumption 4.3.12 From now on, we shall restrict our attention to the class of H.S.NN which satisfies three further conditions: (1) it is either DQ-, DC- or SQ-, (2) it is binary if it is a Q-Q.NN, and (3) it has the tanh activation function, ie., $f(s) = \tanh s$.

Remark 4.3.10 This assumption also means for C-Q.NN we set $R = [-1, 1]$, while for Q-Q.NN we set $R = \{-1, 1\}$. There is no fundamental difference between this scaling of R and any other, thanks to the quasilinear property, but [RIV91] suggests that the gradient of weights for DC-Q.NN with $R = [-1, 1]$ is *better conditioned* than that with $R = [0, 1]$, resulting in faster learning.

Remark 4.3.11 The scaling certainly matters in physical implementations, affecting damage tolerance. There is an interesting explanation to the biological phenomenon of "phantom limbs" [Mel92]. The biological neural networks have the scaling $[0, 1]$, as dictated by physical restrictions, so the "no-signal" level of activation is about 0.5. The constant zero activation level from neurons of the lost limb would therefore be interpreted as a signal by the central nervous system.

4.3.4 Important Examples

Notation. As we shall not discuss NN other than H.S.NN, the abbreviation NN will be used to denote H.S.NN satisfying Assumption 4.3.12.

In summary, we use the prefixes D, S, C, Q, F, B to represent the properties "deterministic", "stochastic", "continuous", "quantized", "feedforward" and "feedback" of H.S.NN, which we abbreviate as NN. Important examples of this classification include the following.

DQF-NN: This includes the so-called logical circuits or multilayer (hard-limit) perceptron network. The perceptron network described in [Ros58, Ros59] can be regarded as single layer DQF-NN.

DQB-NN: This is the (discrete) Hopfield net [Hop82],

DCF-NN: This is the most famous multilayer (soft) perceptron network [RHW86], also called back-propagation network.

DCB-NN: This is the continuous Hopfield net [Hop84, HT86], the mean field (MF) BM [PA87] or deterministic BM [Hin89b], and the feedback perceptron network [Alm87, Pin87] [1].

SQF-NN: This is called belief network [Nea92, Pea87], and simple reinforcement learning networks [Wil87, Wil92].

SQB-NN: This is the most famous Boltzmann machine (BM) [AHS85, HS86].

[1] They are in fact the same network with minor variations in detail.

Remark 4.3.12 Readers familiar with neural network literature would find that many of the most important neural networks are included in this classification. The most noticeable exceptions maybe the so called "higher order networks", also called "sigma-pi" networks, in which the input to a neuron is a higher order polynomial of x and r.

Remark 4.3.13 The networks which do fall in this classification may account for most of the applications. In fact, by far the majority of studies and applications on neural networks are concentrated on the class of fully connected layered or simple-layered DCF-NN.

Remark 4.3.14 As we have introduced the FB-structure, the above six classes are unified into three classes: DQFB-NN, DCFB-NN and SQFB-NN. In §4.4 we shall show that DQ-NN is quite uninteresting mathematically, and can be viewed as extremes of the former two (§4.4). In fact, as we shall see in the next section, the SQ-NN and DC-NN can be incorporated as extremes of an even larger class of neural networks, which we call the Sejnowski class, and denote as TNN. The class of FB-TNN will possess almost all the properties the subclasses have, including stability conditions, and learning rules. A proper study of FB-TNN would be very rewarding mathematically, but unfortunately we do not have enough space to cover it here.

The remainder of this chapter is devoted to discussion of the issues mentioned in the last paragraph. In the next chapter we shall study the DCF-NN, DCB-NN and SQB-NN, since not only they illustrate many features of the more general class, but they are also very important for application, and warrant a study in their own right. The two chapters following that will be devoted to the study of the theory and implementation of SQFB-NN, which is one of the two major themes of this thesis.

4.4 Duality between DC- and SQ-NN

There is a one-one correspondence between DC- and SQ-NN, which we call the *duality* between these networks. This correspondence seems to be widely recognized (Cf. [Hop89, Ami89]), although there seems no explicit statement to this effect in the literature.

Theorem 4.4.1 *Let* $R = \{-1, 1\}$, $\widetilde{R} = [-1, 1]$. *For each* $f \in [\mathbb{R} \to \widetilde{R})$*: define* $\widetilde{f} \in [\mathbb{R} \xrightarrow{P} R) : P_{\widetilde{f}} = p$*:*

$$p(\pm 1|s) := \frac{1}{2}(1 \pm f(s))$$

For each $\phi \in [\mathbb{R} \xrightarrow{P} R)$*: define* $\widehat{\phi} \in [\mathbb{R} \to \widetilde{R})$ *:*

$$\widehat{\phi} = \langle \phi \rangle = P_\phi(1|s) - P_\phi(-1|s).$$

Then $\widehat{\widetilde{f}} = f$, $P_{\widetilde{\widehat{\phi}}} = P_\phi$.

Proof. By direct calculation from definition.

$$\widehat{\widetilde{f}}(s) = \left\langle \widetilde{f}(s) \right\rangle = P_{\widetilde{f}}(1|s) - P_{\widetilde{f}}(-1|s) = \frac{1}{2}(1 + f(s)) - \frac{1}{2}(1 - f(s)) = f(s).$$

$$P_{\widetilde{\widehat{\phi}}}(\pm|s) = \frac{1}{2}\left(1 \pm \widehat{\phi}(s)\right) = \frac{1}{2}(1 \pm (P_\phi(1|s) - P_\phi(-1|s))) = P_\phi(\pm|s).$$

□

This duality is not symmetric between these two networks. The dynamics of SQ-NN are much richer than DC-NN. For a *dual pair* of SQ-NN and DC-NN, $\left\langle r_i^{\mathrm{SQ}} \right\rangle = r_i^{\mathrm{DC}}$.

Corollary 4.4.2 *Let* $f = \tanh$, $p = P_{\widetilde{f}}$. *Then*

$$p(\pm 1|s) = 1/\left(1 + e^{\mp 2s}\right),$$

$$\frac{p(1|s)}{p(-1|s)} = \frac{e^s}{e^{-s}}.$$

In the above, p is called the *local Boltzmann distribution*, $\widetilde{\tanh}$ is called the *random tanh function*, and $p(s) = \frac{1}{2}(\tanh s + 1) = 1/(1 + e^{-2s})$ is called the *logistic function*.

Remark 4.4.1 If $f(s)$ is replaced by $g(s) := f(\mu s)$, and $\mu \to \infty$, then both $g \to \operatorname{sign}$ and $g_p \to \operatorname{sign}$, the *sign function*. So DQ-NN can be viewed as extremes of both DC-NN and SQ-NN. On the other hand, DQ-NN is less useful in practice owing to the above singularity which prevent it to have a reasonable learning rule [BR92, Shv88]. This is hardly surprising, since for DQ-NN the parameterization of realizable processing by the weight vector is discontinuous.

When the states of both SQ-NN and DC-NN in a dual pair are being referred at the same time, there need to be separate names for them. By an analogy with statistical mechanics, but not rigorous correspondence, we call r_i^{SQ} the *local micro state*, r^{SQ} the *global micro state*, r_i^{DC} the *local macro state*, r^{DC} the *global macro state*, $p_i(r_i^{\mathrm{SQ}})$ the *local grand state*, and $p(r^{\mathrm{SQ}})$ the *global grand state*. The grand states are distributions of the micro states, while the macro states are the averages of the micro states.

Remark 4.4.2 Global micro state is specified by the collection of all the local micro states. On the other hand, global grand state is not determined by all the local grand states, unless the states of all the neurons are statistically independent, in which case $p(r) = \prod_i p_i(r_i)$.

Remark 4.4.3 In statistical physics, the *mean field* (MF) model of a stochastic system is a deterministic system in which each subsystem reacts only to the local macro states of other subsystems, but independent to their local micro states. The global grand state can then be identified with the collection of all the local grand states. The duality between SQ- and DC-NN is equivalent to the statement that each DC-NN is the MF system of a unique SQ-NN.

The duality between SQ- and DC-NN has an even deeper origin than the MF interpretation of DC-NN. In fact, they are two extremes in a same one-parameter family, proposed as early as [Sej81]. The Sejnowski class of NN is such that the neuronal input is the *decaying time average* (*leaky integration*) of the input we discussed above. The network equation is therefore

$$\tau \frac{ds}{dt} + s = Ax + B\widetilde{f}(s),$$

where τ is a new parameter which indicates the time factor in the leaky integration, which we call *activation time factor*. We denote by TNN the class of H.S.NN having the above dynamics with tanh activation function. It is obvious that if $\tau = 0$, the neuronal input s is the instant linear composition of r and x, and TNN reduces to SQ-NN, described by

$$s = Ax + B\tilde{f}(s).$$

On the other hand, as $\tau \to \infty$, the neuronal input s becomes the time average of linear compositions of r and x of infinite duration, which equals the ensemble average, so TNN reduces to the MF approximation of SQ-NN, namely, the DC-NN, described by

$$s = Ax + Bf(s),$$

where $f = \left\langle \tilde{f} \right\rangle$.

Remark 4.4.4 It is quite remarkable that the formulation in [Sej81] have all the essential ingredients of today's models: It is both semilinear and stochastic. Therefore it encompasses all the NN mentioned in §4.3.4. All the later developments on these NN can be viewed as adding stability conditions and learning rules to various subclasses.

4.5 Generality of FB-NN and Stability

Theorem 4.5.1 *FB-structure is a generalization of F- and B-structures.*

Proof. For both F- and B-structures, $[i, i] \notin J, \forall i$. Suppose that the J is an F-structure. Then

$$\forall i, j \in \mathbb{N}_N : \ i \sim j \implies i = j \implies B_{ij} = B_{ji}.$$

So it is also a FB-structure. On the other hand, suppose that J is a B-structure, then

$$\forall i, j \in \mathbb{N}_N : \ (B_{ij} = B_{ji}) \implies (i \sim j \implies B_{ij} = B_{ji}).$$

So it is also a FB-structure. □

As a digression we give a characterization of F-, B- and FB-structures in matrix form. Suppose that the neurons in an FB-NN is ordered according to that of the cluster set. Consider the input as cluster 0. Then the matrix $[A, B]$ can be decomposed into blocks [2]

$$[A, B] = \begin{bmatrix} A_{10} & B_1 & & & \\ A_{20} & A_{21} & B_2 & & \\ \vdots & \vdots & \ddots & \ddots & \\ A_{L0} & A_{L1} & \cdots & A_{L,L-1} & B_L \end{bmatrix}$$

where each A_{kl} is the connection from cluster l to cluster k and each B_k is the connection within cluster k. A_{kl} can be arbitrary matrix, while each B_k is symmetric with zero diagonal. In a simple-layered structure $A_{kl} \neq 0 \implies l = k - 1$. Denote $A_k := A_{k,k-1}$. Then the connection is completely described by the matrices $\{A_k, B_k : \ k = 1, \ldots, L\}$. If there is only one layer, it reduces to the B-structure. If each layer has only one neuron then $B_k = 0$, and it reduces to the F-structure.

[2] Note a slight change in notations: A_{kl} is a submatrix of B.

Remark 4.5.1 The FB-structure can be conveniently considered as a two level structure where the coarse structure (that of the clusters) is of F-type while the fine structure (within each cluster) is of B-type. We shall also say the F-structure and B-structure of a FB-NN.

From examples in §4.3.4, we can see that the FB-structure is quite general. Can it be further generalised while retaining all the good properties? The major reason of studying F and B structures is that they guarantee stability. We give some intuitive arguments showing that unless there are more powerful tools for guaranteeing stability of recursive networks, the FB-structure is the most general stable network.

Definition 4.5.1 A NN is called *strongly stable* if for any updating sequence, the neuronal distribution converges to a stationary distribution, which might depend on the initial state and the updating sequence. A NN is called *weakly stable* if for each updating sequence produced from a sequence of identical independent distributions with nonzero probability, the neuronal distribution converges to a stationary distribution, which might depend on the initial state and the updating probability.

Proposition 4.5.2 *A FB-NN is strongly stable.*

Remark 4.5.2 As remarked previously, we have not pin-pointed the dynamics of neural networks we are studying. Therefore we cannot actually "prove" this proposition. An intuitive argument goes like this. (1) Each cluster is a B-structure, which allows a Lyapunov function, guaranteeing strong stability. The convergence of neuronal output distribution of each cluster follows that of the neuronal input distribution, since they are continuously dependent. (2) By an argument similar to the mathematical induction, but applied to the partial order of the F-structure, the strong stability of the FB-structure is also guaranteed. In the two special cases which concerns us, it has been proved [Hop82, Hop84, AHS85] that DCB-NN and SQB-NN are strongly stable. (See also §5.4, §5.2, and §6.2.) Therefore the SQFB-NN and DCFB-NN are also stable.

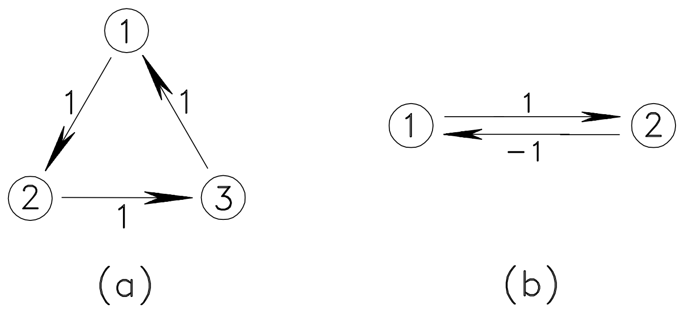

Figure 4.2: Two simple structures which are not strongly stable.

Example 4.5.1 Consider the DQ-NN $1 \xrightarrow{1} 2 \xrightarrow{1} 3 \xrightarrow{1} 1$. (Figure 4.2 (a)) Suppose that the initial state is $[+ + -]$. By updating the neurons in the sequence $[1, 3, 2, 1, 3, 2, \ldots]$, the state of the neurons will be $[-, +, -, +, -, +, \ldots]$. So the network is not strongly stable.

Example 4.5.2 Consider the DQ-NN $1 \stackrel{1}{\rightarrow} 2 \stackrel{-1}{\rightarrow} 1$. (Figure 4.2 (b)) Start with the initial state is $[++]$. By updating the neurons in the sequence $[1, 2, 1, 2, \ldots]$, the state of the neurons will be $[-, -, +, +, -, -, \ldots]$. So the network is not strongly stable.

Now we show that the FB-structure is closed under a certain composition operation. Given two classes of structures a and b. Let $a \circ b$ denote the class of structures formed by replacing each node in an a-structure with a b-structure, and replacing each connection in the a-structure by a bundle of a-connections in the same direction.

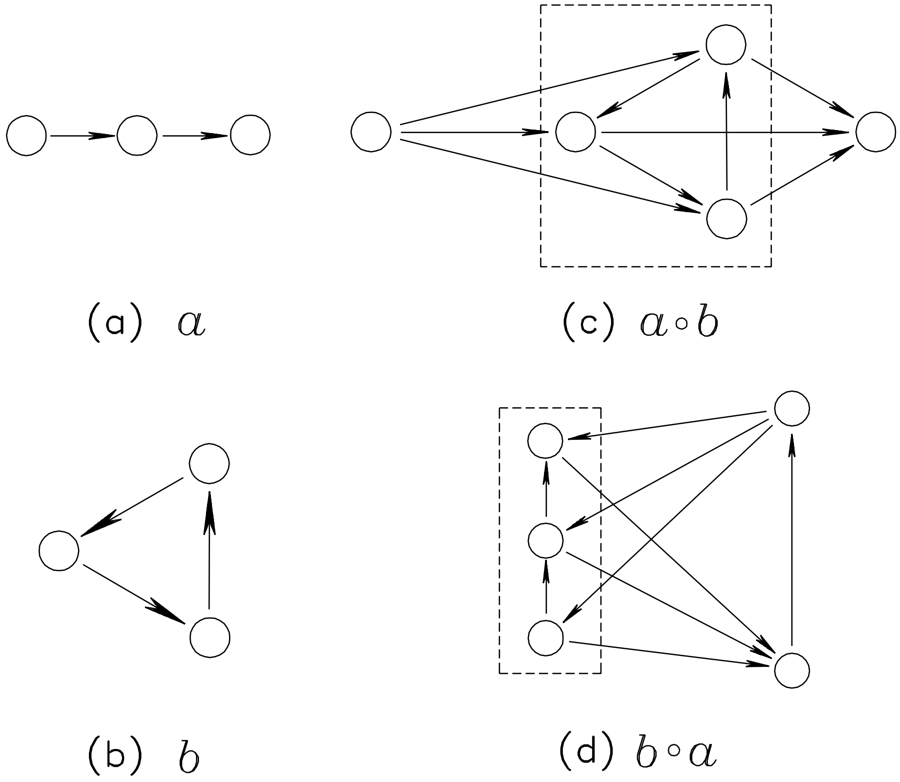

Figure 4.3: Composition of two structures.

Example 4.5.3 Consider the networks in Figure 4.3. The network in (c) is formed by replacing one node in (a) with the network in (b). The network is (d) is formed by replacing one node in (b) with the network in (a).

Remark 4.5.3 Although this concept of "composition of structures" is intuitively simple, a rigorous formulation is very complicated and beyond the scope of this document. We shall rely on the intuitive understanding. Even if the statements may be incorrect in general, it is certainly correct for the two particular structures F and B, which are all that we are interested in.

It is obvious that $(a \circ b) \circ c = a \circ (b \circ c)$. It is also easy to verify that $FB = F \circ B$, $F \circ F = F$, and $B \circ B = B$. So we have $F \circ FB = FB$, and $FB \circ B = FB$. Therefore the class FB-NN will not be enlarged as long as F-structure is" enclosing from outside" while B-structure is "embedded inside".

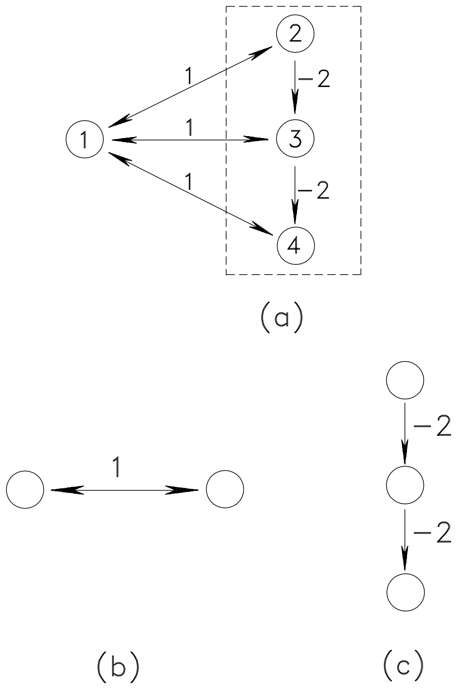

Figure 4.4: A BF-NN composed of a B-NN and F-NN.

Example 4.5.4 $B \circ F$ is not in general strongly stable. Consider a DQ-NN depicted in Figure 4.4 (a), which is formed by composing the B-NN in (b) and F-NN in (c). Suppose the initial state is $[+ + ++]$. Then updating recursively in the sequence $[4, 3, 2, 1, 4, 3, 2, 1, \ldots]$ will result in an oscillation of states with period 4.

Remark 4.5.4 If it can be proved that the only way to guarantee the strong stability of a cluster is that it is of a B structure, then the FB-structure is the most general extension of F and B structures with strong stability. Unless some fundamentally different scheme is found, the discussion concerning generality of FB-structures will remain valid. It is well-known that a dynamic system is stable if and only if it possess a Lyapunov function. The only known universal method of deriving a Lyapunov function is by integration, which necessarily leads to symmetric connections (apart from diagonal scaling as in [Hop84, Alm87]).

Remark 4.5.5 The F and B structures are uniform: both are described by the number of input units, hidden units and output units, since there is only one kind of connections in each of them. The FB-structure is not uniform: The number of hidden layers and the number of neurons in each hidden layer must be specified, since inter and intra layer connections are of different types. The FB-structure, although more general than both F- and B-structures, is not formed by a general "FB-connection" which takes F- and B-connections as extremes. It is only a mixed structure in which F- and B-connections coexists. It is an interesting open question whether there is a useful and general type of connections which takes the above two as extremes. The main obstacle is that the stability of computation is guaranteed by different mechanisms for F and B connections. The B connections results in stable computation because there is an energy function, while for F connections it is because there is no feedback so that input can be considered as constant.

Remark 4.5.6 It may be the case that any NN is weakly stable [Pin88]. What we are interested in are those NN for which stationary distribution does not depend on the updating probability. We conjecture that this is only true if and only the network is strongly stable.

4.6 Learning Rules

4.6.1 General definition

A NN is a trainable information processor (3.6) with *input space* X, *output space* Y, *internal state space* R, *weightspace* W, *auxiliary input space* U and *auxiliary output space* V. Given an input $x \in X_\Omega$, its *computation process*

$$r = \zeta_w(x), \qquad y = \varphi_w(x), \tag{4.6.1}$$

is intertwined with the *learning process*

$$v = \psi(x, y), \qquad w_+ = \delta(w, r, v), \tag{4.6.2}$$

and the *evaluation process*

$$e = \varepsilon(p_x, P_w). \tag{4.6.3}$$

A *learning rule* is a tuple $[\delta, \Sigma]$ such that $\forall [x, \varepsilon, \psi] \in \Sigma$ and $\langle e_+ \rangle \geq \langle e \rangle$.

Remark 4.6.1 Some earlier learning rules, such as the *perceptron learning rule* [Ros59, MP69], have evaluation operators which depend explicitly on w, so that they are only applicable to particular network structure. These are called *direct learning rules*. We shall only consider learning rules which depends on w only implicitly through P_w, so that the evaluation can be written as $E(p, w) = \epsilon(p, P_w)$. These are called *indirect learning rules*.

Notation. Given an environment space Σ, denote

$$X_\Omega{}' := \{x : [x, \varepsilon, \psi] \in \Sigma\}, \tag{4.6.4}$$

$$\mathcal{E}' := \{\varepsilon : [x, \varepsilon, \psi] \in \Sigma\}, \tag{4.6.5}$$

$$\Psi' := \{\psi : [x, \varepsilon, \psi] \in \Sigma\}. \tag{4.6.6}$$

In almost all the good learning rules, $X_\Omega{}' = X_\Omega$, ie., any input distribution is allowed. On the other hand, there is necessarily strong constraints on ε and ψ.

Notation. $\forall p \in P(X), \mathcal{P} \in \mathcal{P}(Y|X) : pP \in \mathcal{P}(X \times Y) : pP(\xi, \eta) := p(\xi)P(\eta|\xi)$.

Definition 4.6.1 Define

$$\mathcal{E}_E := \left\{ \varepsilon : \varepsilon(p, P) = \sum_{\xi} pP(\xi, \eta) \langle e(\xi, \eta) \rangle, e \in [X \times Y \xrightarrow{P} \mathbb{R}^N) \right\}. \tag{4.6.7}$$

(4.6.8)

$$\mathcal{E}_L := \left\{ \varepsilon : \epsilon(p, P) = aK(pP, pP') + bK(pP', pP), P' \in \mathcal{P}(Y|X), a, b \geq 0, a + b > 0 \right\},$$

Each $\varepsilon \in \mathcal{E}_E$ is called a *point evaluation operator* corresponding to the *evaluation function* $e \in [X \times Y \xrightarrow{P} \mathbb{R}^N)$. Each $\varepsilon \in \mathcal{E}_L$ is called a *distribution evaluation operator* corresponding to the *target distribution* $P' \in \mathcal{P}(Y|X)$. If $\mathcal{E}' \subseteq \mathcal{E}_E$ the learning process is called *maximum evaluation* (ME) learning. If $\mathcal{E}' \subseteq \mathcal{E}_L$ the learning process is called *maximum likelihood* (ML) learning.

Almost all the currently known learning rules are special instances of either ME or ML.

Notation. $\Delta w := w_+ - w$, $\Delta e := e_+ - e$.

Definition 4.6.2 The learning rule is called a *gradient following* (GF) learning rule if $\langle \Delta w \rangle \sim \partial E(w)$. It is further called a *deterministic GF rule* if $< \Delta w \sim \partial E(w)$, and a *stochastic GF rule* otherwise.

Theorem 4.6.1 *Let $E \in C^1[W \to \mathbb{R})$, $\langle \Delta w \rangle = \alpha \partial E(w)$, $\alpha \in \mathbb{R}_+$. Then for α sufficiently small, $\langle \Delta e \rangle \geq 0$.*

Proof. Denote $\partial_i := \partial / \partial_{w_i}$. Then

$$\langle \Delta e \rangle = \langle \partial E(w') \Delta w \rangle = \partial E(w') \langle \Delta w \rangle = \alpha \sum_i \partial_i E(w') \partial_i E(w),$$

where $w' \in [w, w_+]$. So for α small enough $\langle \Delta e \rangle \geq 0$. □

GF methods are by far the most widely used learning methods to date, since they are so simple yet always guarantee local improvement, and are perceived as a first step to more powerful methods. Unfortunately, the first step cannot always proceed to the hoped-for second step [Dre79, MP88]. In fact, we shall see that the GF methods almost always produce undesirable results for many interesting problems. We shall also see later that simulated annealing (SA) methods are almost as simple as GF, and overcome most of the undesirable characteristics. Fortunately, it turns out that SA is applicable to most SQ-NN. A variant of SA can also be applied to DC-NN in a different sense.

Many different learning rules are applicable to various networks [Hin89a]. They are generally classified by the "learning mode" into supervised learning (SL), reinforcement learning (RL), and unsupervised learning (UL), depending on the nature of the assistance operator.

In the SL mode, there is an input distribution $p_x \in \mathcal{P}(X)$ and a *teaching mapping* $\varphi' \in [X \xrightarrow{P} Y)$ with distribution $P' = P_{\varphi'} \in \mathcal{P}(Y|X)$. Usually $v = \partial E(w)$. SL can be either ME learning or ML learning. If the evaluation operator is defined by (4.6.8), the learning rule is belong to ML (eg. learning rules in [AHS85]). On the other hand, if there is a distance function d on the output space Y so that $e(x, y) := -d(\varphi'(x), y)$ is well defined, and if the evaluation operator is defined by (4.6.7), the learning rule belongs to ME (eg. learning rules in [RHW86, Alm87])

The RL mode belongs to ME, where $v = e$. It applies to SQ-NN with $|Y| < \infty$.

In the UL mode, $V = \emptyset$, but $\mathcal{E}'$ is rather restricted. Usually $X = Y$ and $e(x, y) = -d(x, y)$, with d being a particular distance function. The learning rules are usually direct. Therefore such learning rules are extremely limited in application.

4.6.2 GF-ME learning in DC-NN

The dynamics of DC-NN $r = \zeta_w(x)$, $y = \varphi_w(x)$, where $w = [A, B]$, is defined by

$$s = Ax + Br, \quad r = f(s), \quad y = Cr + Ds,$$

where C, D are fixed.

Theorem 4.6.2 *Let $v^T = \partial e / \partial y$, and*

$$\Delta w^T = \alpha \left[v^T \frac{\partial y}{\partial w} \right], \quad u^T = v^T \frac{\partial y}{\partial x}. \tag{4.6.9}$$

where $[\cdot]$ is some unbiased running average over a sequence of presentations. Then

$$\left\langle \Delta w^T \right\rangle = \alpha \frac{\partial \langle e \rangle}{\partial w}, \quad u^T = \frac{\partial e}{\partial x}.$$

These can be used to modify w to increase $\langle e \rangle$. If Δw is sufficiently small, $\langle e \rangle$ is almost certain not to decrease.

Theorem 4.6.3 *Let $e \in C^1_2[X \times Y \to \mathbb{R})$, $\rho(B \partial f(s)) < 1$. Denote*

$$u^T := \frac{\partial e}{\partial x}, \quad v^T := \frac{\partial e}{\partial y}, \quad u_1^T := \frac{\partial e}{\partial r}, \quad v_1^T := \frac{\partial e}{\partial s}. \tag{4.6.10}$$

Then

$$\frac{\partial \langle e \rangle}{\partial A} = \left\langle x v_1^T \right\rangle, \quad \frac{\partial \langle e \rangle}{\partial B} = \left\langle r v_1^T \right\rangle, \quad u = A^T v_1, \tag{4.6.11}$$

and u_1, v_1 are solutions of

$$u_1 = C^T v + B^T v_1, \quad v_1 = D^T v + \partial f(s) u_1. \tag{4.6.12}$$

Proof. By the chain rule,

$$\begin{aligned}
\frac{\partial \langle e \rangle}{\partial A} &= \left\langle \frac{\partial e}{\partial s} \frac{\partial s}{\partial A} \right\rangle = \left\langle v_1^T E x \right\rangle = \left\langle x v_1^T \right\rangle, \\
\frac{\partial \langle e \rangle}{\partial B} &= \left\langle \frac{\partial e}{\partial s} \frac{\partial s}{\partial B} \right\rangle = \left\langle v_1^T E r \right\rangle = \left\langle r v_1^T \right\rangle, \\
u^T &= \frac{\partial e}{\partial x} = \frac{\partial e}{\partial s} \frac{\partial s}{\partial x} = v_1^T A,
\end{aligned}$$

where E is the exchange tensor. Equation (4.6.12) follows directly from the chain rule.
□

Remark 4.6.2 The introduction of the exchange tensor makes it possible to avoid explicitly using the subscripts corresponding to neurons as in the usual derivation of learning rules in the literature which tends to confuse rather than inform.

Remark 4.6.3 The purpose of the auxiliary output is that if two DC-NN are cascaded

$$y_1 = \varphi_1(x_1, w_1), \quad x_2 = y_1, \quad y_2 = \varphi_2(x_2, w_2),$$

the learning process of the connected network is equivalent to a learning process in which $v_1 = u_2$. This is a form of backpropagation.

4.6.3 GF-ME learning in SQ-NN

The gradient-following (GF) RL in SQF-NN have been discussed in detail in [Wil87, Wil92].

Theorem 4.6.4 *Let $p_0 \in \mathcal{P}(X)$, $e \in [X \times Y \to \mathbb{R})$. Define $e'(y|x) := \langle e(x,y) \rangle$. Let $W = \mathbb{R}^M$, $P \in [W \to \mathcal{P}(Y|X))$, $P_w \sim \exp T_w$. Then*

$$\frac{\partial \langle e \rangle}{\partial w} = \left\langle \left\langle \frac{\partial T_w}{\partial w}, e \right\rangle_x \right\rangle. \tag{4.6.13}$$

Proof. The conclusion (4.6.13) follows directly from Theorem 3.4.7 and the fact that $\langle e \rangle = \langle \langle e \rangle_x \rangle$. ☐

The stochastic GF algorithm is to set

$$\Delta w = \left[\left[\frac{\partial T_w)}{\partial w}, e \right]_x \right],$$

where $[\cdot]$ is some average over a sequence of trials, and $[\cdot,\cdot]_x$ is taken from some process approximating $\langle \cdot, \cdot \rangle_x$, such as $[u,v] = [(u-[u])(v-[v])]$, or $[u,v] = [(u-[u])v]$, etc. This will be further discussed in §6.4.

Remark 4.6.4 If the SQ-NN receives input from a DC-NN, the auxiliary output can be set to

$$u^T := \frac{\partial \log P(y|x)}{\partial x} e(y|x).$$

When two SQ-NN are cascaded, the auxiliary output is the scalar $e(x,y)$.

Chapter 5

Classical Models of H. S. Neural Networks

5.1 Introduction

In this chapter we study three of the most important special types of H.S.NN: the DCF-NN, also known as feedforward perceptrons, multilayer perceptrons, back-propagation network; the SQB-NN, also known as the Boltzmann machine; and the DCB-NN; also known as the (continuous) Hopfield net, the deterministic Boltzmann machine, and the feedback perceptrons.

5.2 SQB-NN: Boltzmann Machine

The SQB-NN is known as the Boltzmann machine (BM) (with trivial variance) [AHS85, HS86]. There are other variations of the BM, we shall not consider the case where the assignment of input, hidden and output nodes can be decided after learning, which although very useful in many special applications, seems unlikely to be applicable as module in a complex model.

5.2.1 Structure and dynamics

The structure of SQB-NN is specified by the number m of input nodes, n of output nodes, N of neurons, and two output selectors $C, D \in \mathbb{R}^{n\times N}$.

The dynamics is specified by input $x \in \mathbb{R}^m$, output $y \in \mathbb{R}^n$, neuronal state $r \in R^N$ where $R = \{-1, 1\}$, connection matrices $A \in \mathbb{R}^{N\times m}$, $B \in \mathbb{R}^{N\times N}$ such that $B_{ij} = B_{ji}$ and $B_{ii} = 0$. The bias $b \in \mathbb{R}^N$ can be merged into A by setting $x_0 = 1$ and $A_{i0} = b_i$. The weight is $w = [A, B]$.

Definition 5.2.1 A *Boltzmann machine* (BM) is a SQB-NN with $n \leq N$, $C = [0, I_n]$ and $D = 0$.

Definition 5.2.2 (Dynamics of SQB-NN) $r = \zeta_w(x)$, $y = \varphi_w(x)$:

1. Set $s^0 = Ax$, $r^0 = \mathrm{rnd}(R^N)$.

2. Do random sequential update, until equilibrium

$$\Delta r: \; r = \tilde{f}(s), \qquad s = s^0 + Br. \tag{5.2.1}$$

where $\tilde{f} = \widetilde{\tanh}$ is the random tanh function.

3. Output $y = Cr + Ds$.

Notation. Define the *tendency function* $T \in [R|X \to \mathbb{R})$: $T(r|x) = r^T Ax + \frac{1}{2} r^T Br$. Let $\eta \in R^N$, $a \in R$, then denote $\eta^{i(a)} := \eta' : \; \eta'_i = a, \; \eta'_j = \eta_j, \forall j \neq i$. Define $\Delta T \in [R|X \to \mathbb{R}^N): \; \Delta_i T(\eta|x) := (T(\eta^{i(+)}|x) - T(\eta^{i(-)}|x)$. Denote

$$P^{i(a)}(\eta) := \Pr\left\{r^t = \eta^{i(a)} | r^{t-1} = \eta, i^t = i\right\},$$
$$P^{i(a|b)}(\eta) := \Pr\left\{r^t = \eta^{i(a)} | r^{t-1} = \eta^{i(b)}, i^t = i\right\}.$$

Theorem 5.2.1 $\sum_\eta T(\eta|\xi) = 0, \; \forall \xi \in \mathbb{R}^m$.

Proof. Since B is symmetric with zero diagonal, $T(\eta|\xi)$ is the sum of a linear function and a 2-linear function of $\{\eta_i : i\}$ for each ξ. So the conclusion follows from a simple property of multilinear functions. A direct proof is as follows: It is obvious that $\sum_\eta \eta^T A \xi = 0, \; \forall \xi$. For $N = 1$, it holds trivially that

$$\sum_\eta \eta^T B \eta = 0. \tag{5.2.2}$$

Suppose (5.2.2) holds for a particular $N-1$. Denote $\forall \eta \in R^N : \eta =: [\eta'; \eta_N]$, where $\eta' \in R^{N-1}$, $\eta_N \in R$. Since $\operatorname{diag} B = 0$, it is possible to denote

$$B =: \begin{bmatrix} B' & B_N \\ B_N^T & 0 \end{bmatrix},$$

such that $B' \in R^{(N-1)\times(N-1)}$ with $\operatorname{diag} B' = 0$. So

$$\sum_\eta \eta^T B \eta = \sum_{\eta_N} \sum_{\eta'} \eta'^T B' \eta' + \sum_{\eta'} \sum_{\eta_N} 2 \eta'^T B_N \eta_N = 0.$$

The theorem is therefore proved by induction. □

Lemma 5.2.2 *With (5.2.1), it holds that*

$$\Delta T(r|x) = 2s. \tag{5.2.3}$$

For each i, s_i does not depend explicitly on r_i.

Proof. Since B is symmetric with zero diagonal, $\frac{1}{2} r^T Br$ is a 2-linear function on r. Therefore

$$\Delta T(r|x) = 2(Ax + Br) = 2s,$$

and s_i does not depend explicitly on r_i. □

Theorem 5.2.3 (Detailed balance) *The random sequential updating with random tanh function implies*

$$\frac{P^{i(+|-)}(\xi)}{P^{i(-|+)}(\xi)} = \exp\left(\Delta_i T(\xi|x)\right).$$

Proof. From Definition 4.2.3 and Lemma 5.2.2,

$$\begin{aligned} P^{i(a)}(\xi) &= \Pr\left\{\widetilde{\tanh}(s_i^{t-1}) = a | r^{t-1} = \xi\right\} \\ &= \Pr\left\{\widetilde{\tanh}(\Delta_i T(\xi|x)/2) = a\right\}. \end{aligned}$$

It follows Corollary 4.4.2 that

$$\frac{P^{i(+)}(\xi)}{P^{i(-)}(\xi)} = \exp\left(\Delta_i T(\xi|x)\right).$$

Also from Lemma 5.2.2, $P^{i(a)} = P^{i(a|+)} = P^{i(a|-)}$. So the theorem is proved. $\square$

Definition 5.2.3 Let $g \in [R^N \to \mathbb{R})$, define the *time average*

$$[g(r)]_x := \lim_{T\to\infty} \frac{1}{T}\int_0^T g(r^t)\,dt = \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T} g(r^t),$$

where $r^{t+\delta t} := r^t$, $\forall t \in \mathbb{N}$, $\delta t \in [0,1)$.

Theorem 5.2.4 (Boltzmann distribution) *The random sequential updating (5.2.1) converges to a stationary distribution $P \sim \exp T$ with partition function $Z(x) = \sum_r \exp(T(r|x))$. Furthermore, $\forall g \in [R^N \to \mathbb{R})$: $[g(r)]_x = \langle g(r)\rangle_x$.*

The proof of this theorem can be found in, for example, [GG84, Ami89, AK89b]. See also [MRR+53, KGV83, AHS85, HS86] for supplementary information. The main idea is that since the probability of flipping each r_i is non-zero, the random process $[r^t : t]$ is a homogeneous ergodic Markov chain with an equilibrium distribution independent of the initial state r^0, and that the detailed balance ensures that the only equilibrium distribution is the Boltzmann distribution.

Remark 5.2.1 The random sequential updating (5.2.1) realizes a Gibbs sampler $GS(BD(T,1))$. It can be accelerated by the finite simulated annealing (SA1) method, as being the common practice in the literature since [AHS85]. Since this hinders the parallelism of the network, and since this does not alter the final result in any way, we shall not consider this possibility any more. Some other authors (eg. [AK89b]) apply the infinite SA method (SA0) to the BM. The resulting $P(r|x)$ is a uniform distribution on $\{r : \max T(r|x)\}$. This can be used as an optimization method if T is interpreted as an objective function. We shall not consider this alternative either, since the most important feature of NN is that the objective function can be learnt instead of being given analytically.

5.2.2 GF-ML Learning rules

The SQB-NN can have several different learning rules, GF-ME, GF-ML, SA-ME and SA-ML. The BM learning rule [AHS85] is in theory a GF-ML learning rule, but the experiments reported in that paper is actually performed with a crude form of SA-ML learning (§6.6 and §7.6). In this section we only consider GF-ML learning rule. Later we shall show its relation to GF-ME learning rule.

There are two different kinds of ML learning for BM [LL88], using the cross entropy (CE) $K(P,P')$ and the reversed cross entropy (RCE) $K(P',P)$, respectively.

Notation. Let $v \in \mathbb{R}^N$. Denote $\langle v \rangle' := \sum_\xi P'_\xi v_\xi$.

Theorem 5.2.5 *Let $P \in [W \to \mathcal{P}(R^N|X))$, $P \sim \exp T$, $P' \sim \exp T'$. Then*

$$\frac{\partial K(P',P)}{\partial w} = \left\langle \frac{\partial T}{\partial w} \right\rangle - \left\langle \frac{\partial T}{\partial w} \right\rangle', \tag{5.2.4}$$

$$\frac{\partial K(P,P')}{\partial w} = \left\langle \frac{\partial T}{\partial w}, T - T' \right\rangle. \tag{5.2.5}$$

Proof.

$$\begin{aligned} \frac{\partial K(P',P)}{\partial w} &= \frac{\partial}{\partial w} \sum_\xi P'_\xi (\log P'_\xi - \log P_\xi) \\ &= -\sum_\xi P'_\xi \frac{1}{P_\xi} \frac{\partial P_\xi}{\partial w} \\ &= -\sum_\xi P'_\xi \frac{1}{P_\xi} P_\xi \sum_\eta (\delta_{\xi\eta} - P_\eta) \frac{\partial T_\eta}{\partial w} \\ &= \left\langle \frac{\partial T}{\partial w} \right\rangle - \left\langle \frac{\partial T}{\partial w} \right\rangle'. \end{aligned}$$

The third step uses Theorem 3.4.7.

$$\begin{aligned} \frac{\partial K(P,P')}{\partial w} &= \frac{\partial}{\partial w} \sum_\xi P_\xi (\log P_\xi - \log P'_\xi) \\ &= \sum_\xi \frac{\partial P_\xi}{\partial w} \log \frac{P_\xi}{P'_\xi} + \sum_\xi \frac{\partial P_\xi}{\partial w} \\ &= \sum_\xi P_\xi \sum_\eta (\delta_{\xi\eta} - P_\eta) \frac{\partial T_\eta}{\partial w} \log \frac{P_\xi}{P'_\xi} \\ &= \left\langle \frac{\partial T}{\partial w}, \log \frac{P}{P'} \right\rangle = \left\langle \frac{\partial T}{\partial w}, T - T' \right\rangle. \end{aligned}$$

The third step uses the fact that $\sum_\xi P_\xi = 1$, Theorem 3.4.7, and (3.3.6). □

Definition 5.2.4 (GF-ML learning rule for SQB-NN) $w_+ = \delta(w, r, v)$: The *BM learning rule* [AHS85] is defined as GF learning on RCE,

$$\Delta w^T \sim \left[\left[\frac{\partial T}{\partial w} \right]_x - \left[\frac{\partial T}{\partial w} \right]'_x \right], \tag{5.2.6}$$

The *variant Boltzmann machine learning rule* (VBM) [LL88] is defined as GF learning on CE,

$$\Delta w^T \sim \left[\frac{\partial T}{\partial w}(T - T' - [T - T']_x)\right]. \tag{5.2.7}$$

It should be remarked that there is no practical implementation for the VBM learning rule as a ML rule, but it is closely related to a good ME rule (§6.6).

5.3 DCF-NN: Multilayer Perceptron

The DCF-NN is often called *back-propagation network*, *multilayer perceptron network* or *feedforward perceptron network* [RHW86, Wer74, Cun85, Par85]. We give formulation for simple-layered DCF-NN, which consists of a sequence of layers of neurons where each neuron only activate neurons in immediately succeeding layer. The general DCF-NN is similar, with slightly more complicated notations.

5.3.1 Structure and dynamics

The structure of the DCF-NN is in general specified by input size m, output size n and network size N. By far the most commonly used DCF-NN is fully-connected simple-layered, which can be described as N_0-N_1- ... -N_L where L is the number of layers, N_k is the number of neurons in layer k, and $N_0 = m$. The connection matrix is $A = [A_1, \ldots, A_L]$ where $A_k \in \mathbb{R}^{N_k \times N_{k-1}}$. The output selectors C and D are fixed matrices, usually being identity matrices or zero matrices, or other projection matrices. The most commonly used combination is $C = 0$, $D = [0, \ldots, I_n]$.

Remark 5.3.1 There are several conventions for counting the number of layers in a layered network. We adopt the convention of counting the number of layers of adjustable connections. When counting layers of neurons, all those following a layer of adjustable connections is counted. The input layer is not considered as a part of the network. Therefore, a DCF-NN with one hidden layer is said to have two layers (usually called a "three layer network" or a "one hidden layer network").

Definition 5.3.1 (Dynamics of DCF-NN) $r = \zeta_w(x)$, $y = \varphi_w(x)$:

1. Input layer. Set input $r_0 = x$.
2. Hidden layers. For $k = 1, \ldots, L$, do
$$s_k = A_k r_{k-1}, \quad r_k = f(s_k).$$
where $f = \tanh$.
3. Output layer. Set $y = Cr_L + Ds_L$,

It has been proved that a two layer DCF-NN can approximate almost arbitrary functions given enough neurons in the hidden layer [Fun89]. The function to be approximated need not to be continuous [HSW89], it is only assumed to be in an appropriate Sobolev space (For introduction to Sobolev spaces see [Ada75]). The approximation is in the sense of a corresponding Sobolev norm, which can also take into account of the derivatives [HSW90] (in the weak sense if the function is discontinuous). The activation function f can be any monotonic and bounded function for these approximation theorems to hold [Hor91].

Remark 5.3.2 These approximation properties are quite important theoretically, for they show that the multilayer networks overcome the major difficulties of single layer networks [MP69]. This also provides a justification that we are to restrict our attention to the class of H.S.NN. On the other hand, they are more than can be utilised, since what we are concerned is not what the network can represent in principle, but what it can learn to represent (§6.5).

Remark 5.3.3 The function $f = \tanh$ is likely to be an optimal activation function in some sense owing to its close relationship with information theory (§4.4). As a precaution, we have also tried to replace it with several other activation functions in our experiments from time to time. In most occasions no discernible change was found in the performance, while in the remaining cases the performance seems to deteriorated. The functions we tried include atan and $f(s) := s/(1+|s|)$. We do not pursue this issue further.

5.3.2 GF-ME learning rules

Notation. Let $e \in C_2^1[X \times Y \to \mathbb{R})$. Define

$$v^T := \frac{\partial e}{\partial y}, \quad u^T := \frac{\partial e}{\partial x}, \quad v_k^T := \frac{\partial e}{\partial s_k}, \quad u_k^T := \frac{\partial e}{\partial r_k}.$$

Theorem 5.3.1 *Let x, y, r, s satisfy Definition 5.3.1, $e \in C_2^1[X \times Y \to \mathbb{R})$. Denote $\langle e \rangle := \sum_x p(x) e(x, y)$. Then*

$$\frac{\partial \langle e \rangle}{\partial A_k} = \left\langle r_{k-1} v_k^T \right\rangle,$$

where u_k, v_k are solutions of

$$u_L = C^T v, \quad v_L = \partial f(s_L) u_L + D^T v,$$

$$u_k = A_{k+1}^T v_{k+1}, \quad v_k = \partial f(s_k) u_k,$$

*Proof*It follows from the chain rule that

$$u_k = A_{k+1}^T v_{k+1}, \quad v_k = \partial f(s_k) u_k,$$

$$\frac{\partial \langle e \rangle}{\partial A_k} = \left\langle \frac{\partial e}{\partial s_k} \frac{\partial s_k}{\partial A_k} \right\rangle = \left\langle v_k^T E r_{k-1} \right\rangle = \left\langle r_{k-1} v_k^T \right\rangle.$$

□

The GF-ME learning rule for DCF-NN is usually called *back-propagation* (BP), since gradients are propagated in the network in the reversed direction of the forward computation.

Definition 5.3.2 (GF-ME learning rule for DCF-NN) Set $w_+ = \delta(w, r, v)$, $u = \kappa(w, r, v)$, by

1. Output layer. Set $v = \frac{\partial e}{\partial y^T}$. Compute

$$u_L = C^T v, \quad v_L = \partial f(s_L) u_L + D^T v,$$

$$\Delta A_L = \alpha \left[v_L r_{L-1}^T \right].$$

2. Hidden layers. For $k = L-1, \ldots, 1$, do

$$u_k = A_{k+1}^T v_{k+1}, \quad v_k = \partial f(s_k) u_k,$$

$$\Delta A_k = \alpha \left[v_k r_{k-1}^T \right],$$

   where $[\cdot]$ is some running average process.

3. Input layer. Set $u = u_0 = A_1^T v_1$.

The auxiliary output u is needed when two such networks are cascaded, and can otherwise be discarded.

5.4 DCB-NN: Continuous Hopfield Net, Mean field Boltzmann Machine and Feedback Perceptron

The DCB-NN can be introduced from three different points of view:

- As a continuous variant of Hopfield net [Hop84], (DQB $\to$ DCB).
- As a variation of feedforward networks [Alm87, Pin87], (DCF $\to$ DCB).
- As a mean field Boltzmann machine [PA87, Hin89b]. (SQB $\to$ DCB).

They can be proved to define the same input-output relation, or, using a term used in [Wil90], to be "input-output equivalent".

5.4.1 Structure and dynamics

The structure of the DCB-NN is the same as that of the SQB-NN. Let A, B, C and D be matrices representing the synapse from the input to the neurons, among the neurons and from the neurons to the output, respectively. C and D can be arbitrary but fixed. B is symmetric with zero diagonal. The network of [Hop84, Alm87] allows slightly general B, which is equivalent to scaling the output of each neuron independently. Since we are only interested in homogeneous networks we shall not discuss this possibility.

Definition 5.4.1 (Dynamics of DCB-NN) $r = \zeta_w(x)$, $y = \varphi_w(x)$:

1. Input. Set $s^0 = Ax$.

2. Solve the equation

$$s = s^0 + Br, \quad r = f(s), \tag{5.4.1}$$

   where $f = \tanh$.

3. Output. Set $y = Cr + Ds$.

The recursive equation (5.4.1) can be solved by the following three different methods.

Process (P0) (5.4.1) is iterated and the fixed point is taken as solution [PA87].

Process (P1) The solution is defined as the attractor of differential equation [Alm87]

$$s = Br + s^0, \quad \frac{dr}{dt} = \alpha(f(s) - r). \tag{5.4.2}$$

Process (P2) The solution is defined as the attractor of differential equation [Hop84]

$$\frac{ds}{dt} = \alpha(Br + s^0 - s), \quad r = f(s). \tag{5.4.3}$$

In the above α is some positive constant.

It is easy to see that if any of these three processes do converge to some fixed point, this fixed point will be a solution of (5.4.1). We shall show that (P1) and (P2) are equivalent, in the sense that their solutions are related by an differential-algebraic relation.

Theorem 5.4.1 *Both (5.4.2) and (5.4.3) imply*

$$\frac{ds}{dt} = \alpha(Bf(s) + s^0 - s). \tag{5.4.4}$$

Denote the solution of (P1) and (P2) by $[r_1, s_1]$ *and* $[r_2, s_2]$*, respectively, then for any* t*,* $s_1 = s_2$ *at* t *implies*

$$s_1 \equiv s_2, \quad \frac{dr_1}{dt} = \alpha(r_2 - r_1). \tag{5.4.5}$$

Proof. Suppose (5.4.2) hold. Then

$$\frac{ds}{dt} = B\frac{dr}{dt} = \alpha B(f(s) - r) = \alpha(Bf(s) + s^0 - s).$$

This is also trivially implied by (5.4.3). This proves the first part of the theorem.

The solution to the differential equation (5.4.4) is unique in both directions for any initial value. Therefore $s_1 = s_2|_t$ implies $s_1 \equiv s_2$. This, in turn, implies that

$$\frac{dr_1}{dt} = \alpha(f(s_1) - r_1) = \alpha(f(s_2) - r_1) = \alpha(r_2 - r_1).$$

This proves the second part of the theorem. □

Remark 5.4.1 Each of (5.4.4) and (5.4.5) might have more than one solutions, depending on the initial state. In that case we define the solution obtained by starting from $s = Ax$ and $r = 0$ as the required solution.

5.4.2 Free energy and stability of dynamics

The stability of (P1) and (P2) are guaranteed by the existence of a Lyapunov function, which is usually called "energy", following [Hop84]. We shall show that this function actually corresponds to "free energy" when the network is considered as a mean field system of the Boltzmann machine. Define

$$T(r|x) := \frac{1}{2} r^T B r + r^T A x.$$

Then

$$\frac{\partial T(r|x)}{\partial r^T} = Br + s^0.$$

Now we interpret DCB-NN as the mean field model of SQB-NN whose state is r^{SQ}, where $\left\{ r_i^{\mathrm{SQ}} \right\}$ is statistically independent, ie., $r_i = \left\langle r_i^{\mathrm{SQ}} \right\rangle$ and

$$\Pr \left\{ r_i^{\mathrm{SQ}} = \pm 1 \right\} = p_i^{\pm} := (1 \pm r_i)/2.$$

The entropy $-S$ of the network is the sum of the entropies of all the neurons

$$S = \sum_i (p_i^+ \log p_i^+ + p_i^- \log p_i^-).$$

It follows that

$$\frac{\partial S}{\partial r^T} = \frac{1}{2} \log \frac{1+r}{1-r} = \tanh^{-1} r. \tag{5.4.6}$$

Taking $-T$ as energy and introducing an artificial temperature θ, the free energy $-f$ of the network is defined by $f = T - \theta S$. Combining the gradients for T and S gives

$$\frac{\partial f}{\partial r^T} = Br + s^0 - \theta \tanh^{-1} r,$$

Process (P2) is therefore equivalent to (with $\theta = 1$)

$$\frac{ds}{dt} = \alpha \frac{\partial f}{\partial r^T} \implies \frac{df}{dt} = \frac{1}{\alpha} \frac{ds^T}{dt} \frac{dr}{dt} > 0.$$

Considering Theorem 5.4.1, this proves the following

Theorem 5.4.2 *Both (P1) and (P2) are stable.*

Remark 5.4.2 The above proof follows [Hop84]. An equivalent proof of stability was directly applied to (P1) in [Alm87]. The free energy $-f$ is just the "energy" in [Hop84], with the correspondence $\alpha \sim 1/C_i$, $\theta \sim 1/R_i$, and $S \sim \int \tanh^{-1}(r) dr$, where C_i and R_i are called capacitance and resistance in [Hop84]. This fact the f corresponds to free energy was mentioned in [Hin89b] without detail.

5.4.3 GF-ME learning rules

Notation. Let $e \in C_2^1[X \times Y \to \mathbb{R})$. Define

$$v^T := \frac{\partial e}{\partial y}, \quad u^T := \frac{\partial e}{\partial x}, \quad v_1^T := \frac{\partial e}{\partial s}, \quad u_1^T := \frac{\partial e}{\partial r}.$$

Theorem 5.4.3 *Let x, y, r, s satisfy Definition 5.4.1, $e \in C_2^1[X \times Y \to \mathbb{R})$. Denote $\langle e \rangle := \langle (x,y) \rangle = \sum_x p(x) \sum_y P(y|x) e(x,y)$. Then*

$$\frac{\partial \langle e \rangle}{\partial A} = \left\langle x v_1^T \right\rangle, \quad \frac{\partial \langle e \rangle}{\partial B} = \left\langle r v_1^T \right\rangle,$$

where u_1, v_1 are solutions of

$$u_1 = B v_1 + C^T v, \quad v_1 = \partial f(s) u_1 + D^T v. \tag{5.4.7}$$

Proof. It follows from the chain rule that

$$u_1 = C^T v + B^T v_1, \quad v_1 = D^T v + \partial f(s) u_1,$$

$$\frac{\partial \langle e \rangle}{\partial A} = \left\langle \frac{\partial e}{\partial s} \frac{\partial s}{\partial A} \right\rangle = \left\langle v_1^T E x \right\rangle = \left\langle x v_1^T \right\rangle.$$

$$\frac{\partial \langle e \rangle}{\partial B} = \left\langle \frac{\partial e}{\partial s} \frac{\partial s}{\partial B} \right\rangle = \left\langle v_1^T E r \right\rangle = \left\langle r v_1^T \right\rangle.$$

Equation (5.4.7) has a solution if and only if $\rho(\partial f(s) B) < 1$, which is always satisfied when the equilibrium (5.4.1) is reached [Alm87]. □

Definition 5.4.2 (GF-ME learning rule for DCB-NN) Set $w_+ = \delta(w, r, v)$, $u = \kappa(w, r, v)$, by

1. Output layer. Set $v = \frac{\partial e}{\partial y^T}$.
2. Iterate, until convergence

$$u_1 = B v_1 + C^T v, \quad v_1 = \partial f(s) u_1 + D^T v.$$

   Set

$$\Delta A = \alpha \left[x v_1^T \right], \quad \Delta B = \alpha \left[r v_1^T \right],$$

   if there is no restriction on B, or

$$\Delta B = \alpha \left[\frac{1}{2} (v_1 r^T + r v_1^T) \right],$$

   if the the condition $B = B^T$ is imposed, where $[\cdot]$ is some running average process.

3. Input layer. Set $u = A^T v_1$.

This process is propagation on a "transposed network". It converges since (5.4.7) is linear with $\rho(\partial f(s) B) < 1$.

5.5 DCFB-NN: Generalization of DCF-NN and DCB-NN

The DCF-NN and DCB-NN can be generalized to the DCFB-NN, which can be conveniently considered as the result of embedding several DCB-NN in a DCF structure.

5.5.1 Structure and dynamics

For clarity we discuss simple-layered DCFB-NN, which has L layers of neurons, where there are unidirectional inter layers connections as in DCF-NN, and symmetric bidirectional intra layer connections as in DCB-NN. Denote by A_k the connection from layer $k-1$ to layer k, and by B_k the connection within layer k. The output selectors C and D are some fixed matrices, usually being identity matrix or zero matrix.

Definition 5.5.1 (Dynamics of DCFB-NN) $r = \zeta_w(x)$, $y = \varphi_w(x)$

1. Input layer. Set input $r_0 = x$.
2. Hidden layers. For $k = 1, \ldots, L$, solve the equation
$$s_k = A_k r_{k-1} + B_k r_k, \quad r_k = f(s_k).$$
3. Output layer. Set $y = Cr_L + Ds_L$.

Theorem 5.5.1 *The dynamics of DCFB-NN is strongly stable.*

Proof. By induction on the F-structure of the DCFB-NN. The cluster 0, the input, is stable. For each k, if r_{k-1} be fixed, then r_k is stable since this is effectively a DCB-NN. It is also evident that the fixed point of r_k depends continuously on r_{k-1}, with a norm independent of r_{k-1}. Therefore as r_{k-1} converge to a fixed point r_k also converge to a corresponding fixed point. $\square$

Remark 5.5.1 It is easy to verify that if all the $B_k = 0$ then DCFB reduce to DCF, and that if $L = 1$ then DCFB reduce to DCB.

5.5.2 GF-ME learning rule

The GF-ME learning rule of DCFB-NN is derived similar to those of DCF-NN and DCB-NN.

Notation. Let $e \in C_2^1[X \times Y \to \mathbb{R})$. Denote
$$v^T := \frac{\partial e}{\partial y}, \quad u^T := \frac{\partial e}{\partial x}, \quad v_k^T := \frac{\partial e}{\partial s_k}, \quad u_k^T := \frac{\partial e}{\partial r_k}.$$

Theorem 5.5.2 *Let x, y, r, s satisfy Definition 5.5.1, $e \in C_2^1[X \times Y \to \mathbb{R})$. Denote $\langle e \rangle := \langle (x, y) \rangle = \sum_x p(x) \sum_y P(y|x) e(x, y)$. Then*

$$(5.5.1) \qquad \frac{\partial \langle e \rangle}{\partial A_k} = \left\langle \frac{\partial e}{\partial s_k} \frac{\partial s_k}{\partial A_k} \right\rangle = \left\langle r_{k-1} v_k^T \right\rangle, \quad \frac{\partial \langle e \rangle}{\partial B_k} = \left\langle \frac{\partial e}{\partial r_k} \frac{\partial r_k}{\partial B_k} \right\rangle = \left\langle r_k v_k^T \right\rangle,$$

where u_k and v_k are solutions of

$$u_L = B_L v_L + C^T v, \quad v_L = \partial f(s_L) u_L + D^T v,$$
$$u_k = A_{k+1}^T v_{k+1} + B v_k, \quad v_k = \partial f(s_k) u_k.$$

Proof. Apply the chain rule, similar to §5.3 and §5.4, □

Definition 5.5.2 (GF-ME learning rule for DCFB-NN) Set $w_+ = \delta(w, r, v)$, $u = \kappa(w, r, v)$, by

1. Output layer. Set $v = \frac{\partial e}{\partial y^T}$. Solve

$$u_L = B_L v_L + C^T v, \quad v_L = \partial f(s_L) u_L + D^T v,$$

   Update

$$\Delta A_L = \alpha \left[v_L r_{L-1}^T \right], \quad \Delta B_L = \alpha \left[v_L r_L^T \right].$$

2. Hidden layers. For $k = L-1, \ldots, 1$, solve

$$u_k = A_{k+1}^T v_{k+1} + B_k v_k, \quad v_k = \partial f(s_k) u_k.$$

   Update

$$\Delta A_k = \alpha \left[v_k r_{k-1}^T \right], \quad \Delta B_k = \alpha \left[v_k r_k^T \right],$$

   where $[\cdot]$ is some running average process.

3. Input layer, $u = u_0 = A_1^T v_1$.

When the condition $B_k = B_k^T$ is imposed, the above rules should be changed in the form of $\Delta B = \frac{\alpha}{2}(v r^T + r v^T)$.

Remark 5.5.2 Although DCFB-NN is a generalization of many existing C-NN, we do not study it further, since in the next two chapters we shall study its stochastic dual, the SQFB-NN, which is not only much more powerful, but also with better physical understanding. Most results on SQFB-NN can be immediately transformed back to the DCFB-NN.

Chapter 6

Theory of SQFB Neural Networks

6.1 Introduction

In this chapter we study the mathematical theory of the SQFB-NN, including its structure, dynamics and learning rules.

A focal point of this chapter is the learning rule. We derive both the gradient-following (GF) and simulated annealing (SA) learning rules for maximum evaluation (ME) learning. Most learning rules in the literature are GF rules. We show by experiments that GF learning almost always converges to local optima while the SA learning rule almost always converge to global optima.

6.2 Structure and Dynamics

Recall the following notations for SQFB-NN: m—number of input nodes, n—number of output nodes, N—number of neurons, number of clusters L, sizes of clusters $[N_k : k \in \mathbb{N}_L]$, and $C, D \in \mathbb{R}^{n \times N}$—output selectors. Denote $N_0 := m$. The dynamics are specified by input $x \in \mathbb{R}^m$, output $y \in \mathbb{R}^n$, neuronal state $r = [r_k : k \in \mathbb{N}_L]$, $r_k \in R^{N_k}$ where $R = \{-1, 1\}$, and connection matrices

$$A = [A_{kl} \in \mathbb{R}^{N_k \times N_l} : \ l < k, \ k \in \mathbb{N}_L],$$

$$B = [B_k \in \mathbb{R}^{N_k \times N_k} : \ k \in \mathbb{N}_L],$$

with $(B_k)_{ij} = (B_k)_{ji}$ and $(B_k)_{ii} = 0$. The bias $b \in \mathbb{R}^N$ can be merged into A by setting $x_0 = 1$ and $A_{i0} = b_i$. The weight is $w = [A, B]$.

The layered structure can be expressed as N_0-N_1- ... -N_L, The ith component of r_k is denoted r_{ki}, which is the state of neuron i in layer k.

Assumption 6.2.1 For simplicity, we shall only consider the simple-layered SQFB-NN

$$A_{kl} \neq 0 \implies l = k - 1.$$

The network is called *multilayer Boltzmann machine* if $N_L = n$, $C = [0, I_n]$, $D = 0$.

The dynamics of the network is described by two nested Markov chains. The propagation of information among the layers is described by a finite length inhomogeneous Markov chain $[r_k : k \in \mathbb{N}_{0,L}]$, while the computation inside layer k is described by a homogeneous Markov chain $[r_k^t : t \in \mathbb{N}]$.

Definition 6.2.2 (Dynamics of SQFB-NN) $r = \zeta_w(x)$, $y = \varphi_w(x)$:

1. Input $r_0 = x$.
2. For $k = 1, \cdots, L$, do
   (a) Set $s_k^0 = A_k r_{k-1}$, $r_k^0 = \mathrm{rnd}(R^{N_k})$.
   (b) Do random sequential update until equilibrium

$$\Delta r_k : \; r_k = \tilde{f}(s_k), \; s_k = s_k^0 + B_k r_k. \tag{6.2.1}$$

   where $\tilde{f} = \widetilde{\tanh}$ is the random tanh function.
3. Output $y = r_L$.

The following notations are similar to their counterparts in §5.2, except that there is one for each layer. The matrices A_k and B_k define the *conditional tendency function* of layer k:

$$T_k(r_k|r_{k-1}) := r_k^T A_k r_{k-1} + \frac{1}{2} r_k^T B_k r_k.$$
$$\Delta_i T_k(r_k|r_{k-1}) := T_k(r_k(i+)|r_{k-1}) - T_k(r_k(i-)|r_{k-1}).$$

Remark 6.2.1 For a general SQFB-NN, there are connections between any two layers. Denote $\overline{r}_{k-1} := [r_0, \ldots, r_{k-1}]$. Denote by A_{kj} the connection from layer j to layer k where $j < k$. Then

$$T_k(r_k|\overline{r}_{k-1}) = \sum_{j=0}^{k-1} r_k^T A_{kj} r_j + \frac{1}{2} r_k^T B_k r_k.$$

All the results below can be easily extended to this general case.

For each layer k, the dynamics of SQFB-NN is exactly that of a BM, with x replaced by r_{k-1}. The following results follow easily from their counterparts in §5.2.

Lemma 6.2.1 *With (6.2.1), it holds that*

$$\Delta T_k(r_k|r_{k-1}) = 2 s_k. \tag{6.2.2}$$

For each i, s_{ki} does not depend explicitly on r_{ki}.

Proof. Similar to Lemma 5.2.2. □

Notation. Denote

$$P_k^{i(a)}(\xi) := \Pr\left\{ r_{ki}^t = \xi^{i(a)} | r_k^{t-1} = \xi, i_k^t = i \right\},$$
$$P_k^{i(a|b)}(\xi) := \Pr\left\{ r_{ki}^t = \xi^{i(a)} | r_k^{t-1} = \xi^{i(b)}, i_k^t = i \right\}.$$

Theorem 6.2.2 (Detailed balance) *The random sequential updating with random tanh function implies*

$$\frac{P_k^{i(+|-)}(\xi)}{P_k^{i(-|+)}(\xi)} = \exp\left(\Delta_i T_k(\xi|r_{k-1})\right). \tag{6.2.3}$$

Definition 6.2.3 Let $g \in [R^{N_k} \to \mathbb{R})$, define the *time average*

$$[g(r_k)]_{k-1} := \lim_{T\to\infty} \frac{1}{T}\int_0^T g(r_k^t)\,dt, = \lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T} g(r_k^t), \tag{6.2.4}$$

where $r_k^{t+\delta t} := r_k^t$, $\forall t \in \mathbb{N}$, $\delta t \in [0,1)$, and the *ensemble average*

$$\langle g(r_k)\rangle_{k-1} := \sum_x g(x)\Pr\{r_k = x | r_{k-1}\}.$$

Let $u, v \in [R^{N_k} \to \mathbb{R})$ then $\langle u, v\rangle_{k-1} := \Big\langle (u - \langle u\rangle_{k-1})(v - \langle v\rangle_{k-1})\Big\rangle_{k-1}$.

Theorem 6.2.3 (Boltzmann distribution) *The random sequential updating of layer k converges to a stationary distribution $P_k \sim \exp T_k$ with partition function $Z_{k-1}(r_{k-1}) = \sum_\eta \exp(T_k(\eta|r_{k-1}))$. Furthermore, $\forall g \in [R^{N_k} \to \mathbb{R})$: $[g(r_k)]_{k-1} = \langle g(r_k)\rangle_{k-1}$.*

Remark 6.2.2 The condition (6.2.3) only defines a one-parameter family of processes, since the two probabilities of flipping a bit can be scaled by a same factor. Some of the more important in this family are the Metropolis process [MRR+53], the Glauber process [Gla63] and the local Boltzmann process [AHS85]. They will be discussed in detail in §7.2, when an efficient implementation on conventional computers is also introduced.

6.3 Relations with Other Networks

6.3.1 Relation with SQB-NN (Boltzmann machine)

The SQFB-NN is a generalization of SQB-NN (§5.2). Therefore the structure of SQFB-NN is more flexible than that of SQB-NN. On the other hand, there is a BM corresponding to each SQFB-NN, formed by replacing each unidirectional connection with a symmetric connection. The advantage of SQFB-NN is that it allows for uni-directional channels, which are unavoidable in many circumstances. For example, the SQFB-NN may be trained as a data compressor, with half of the network as an encoder and half of the network as a decoder. In some other applications it may be necessary to cascade two SQB-NN, so the overall network will be a SQFB-NN.

6.3.2 Relations with DCF-NN (feedforward perceptron)

The deterministic dual (the mean field model) of SQFB-NN is the DCFB-NN (§5.5), which generalizes DCF-NN (multilayer perceptron) (§5.3). This makes it possible to extend some theorems previously proved for DCF-NN to SQFB-NN.

Theorem 6.3.1 (Approximating SQF-NN by DCF-NN) *Suppose that $B_k = 0$ for all k. Then*

$$\langle r_k\rangle_{k-1} = \tanh(A_k r_{k-1})$$

Proof. Since $B_k = 0$, $s_{ki} = (A_k r_{k-1})_i$ is independent of r_k, therefore

$$\langle r_{ki}\rangle_{k-1} = \frac{\exp(2(A_k r_{k-1})_i) - 1}{\exp(2(A_k r_{k-1})_i) + 1} = \tanh((A_k r_{k-1})_i).$$

$\square$

When the number of neurons in each layer is sufficiently large, the effect of r_{k-1} on r_k can be approximated by the mean field approximation $\langle r_k \rangle = \langle \tanh(A_k r_{k-1}) \rangle \approx \tanh(A_k \langle r_{k-1} \rangle)$, see [PA87] and [Hop89].

As has been mentioned earlier (§5.3), it is known that a two layer simple-layered fully-connected DCF-NN with sufficient number of neurons in the hidden layer is a universal approximator of functions and gradients in the sense of Sobolev space [Fun89, HSW89, HSW90]. These results directly apply to the SQFB-NN, since all of them are based on the assumption of a network of sufficiently large number of neurons. [BL91] gives upper bounds on the number of neurons needed for approximation for a three-layer DQF-NN, this result is also applicable to SQFB-NN, since SQFB-NN generalizes SQF-NN which in turn generalizes DQF-NN. However, these bounds are not quite practical since they are combinatorial.

6.3.3 Relation with DCB-NN (continuous Hopfield net, etc.)

The DCFB-NN is also a generalization of the DCB-NN (§5.4), such as continuous Hopfield net [Hop84], the feedback perceptron network [Alm87], the mean field (MF) model of BM [PA87, Hin89b]. It should be noted that in general the SQFB-NN cannot be replaced by its MF approximation DCFB-NN without compromising its representational power. In particular, SQFB-NN can represent the correlations between the states of neurons in each layer, which are absent in the MF approximation. Only under the condition that the correlation between neurons in one layer has little effect on that in the subsequent layer can the SQFB-NN be replaced by DCFB-NN in all layers except the last, which is still needed for producing correlated output [Hop89].

6.4 GF-ME Learning Rule

In this section we consider maximum evaluation (ME) learning for SQFB-NN based on gradient following (GF). In §6.6 the simulated annealing (SA) learning rule is studied, as well as the relation between ME learning and maximum likelihood (ML) learning.

Definition 6.4.1 Let $e \in [X \times Y \xrightarrow{P} \mathbb{R})$. Define $e_L(r_L|r_0) := \langle e(x, y) \rangle$. For $k \in \{1, \ldots, L\}$, define recursively $e_{k-1} \in [R^{N_{k-1}}|R^{N_0} \to \mathbb{R})$:

$$e_{k-1}(r_{k-1}|r_0) := \langle e_k(r_k|r_0) \rangle_{k-1} .$$

Notation. By a slight abuse of notation, we denote $T_k := T_k(r_k|r_{k-1})$, $P_k := P_k(r_k|r_{k-1})$, $P_{k|j} := P_{k|j}(r_k|r_j)$, $p_k := p_k(r_k)$, and $e_k := e_k(r_k|r_0)$. We also use r_k to denote both the random variable and its value, whenever there is no danger of confusion.

Theorem 6.4.1 *Let* $0 \le k \le j \le L$. *Then*

$$e_k = \langle e_j \rangle_k . \tag{6.4.1}$$

Proof. Induction on k. (6.4.1) is true for $k = L$. Suppose it is true for a particular $k \le L$. For all j such that $0 \le k \le j \le L$,

$$e_{k-1} = \sum_{r_k} P_k e_k = \sum_{r_k} P_k \sum_{r_j} P_{j|k} e_j = \sum_{r_j} P_{j|k-1} e_j$$

So it is also true if k is replaced by $k-1$. □

Corollary 6.4.2 *Let $p_0 \in \mathcal{P}(X)$. Then for each $0 \le j \le L$.*

$$\langle e \rangle = \sum_{r_0} p_0 \sum_{r_j} P_{j|0} e_j. \tag{6.4.2}$$

$$e_k = \langle e_L \rangle_k . \tag{6.4.3}$$

Remark 6.4.1 Note that e_k and $\langle e \rangle$ depend on A_k and B_k through P_k for each k.

Let $e \in [X \times Y \xrightarrow{P} \mathbb{R})$. Denote $w_k := [A_k, B_k]$. The stochastic GF-ME learning rule is defined by

$$\langle \Delta w_k \rangle = \alpha \frac{\partial \langle e \rangle}{\partial w_k^T}. \tag{6.4.4}$$

Lemma 6.4.3 *Let $p_{r_0} = p_0 \in \mathcal{P}(X)$. Then*

$$\frac{\partial \langle e \rangle}{\partial w_k^T} = \sum_{r_0} p_0 \sum_{r_{k-1}} P_{k-1|0} \frac{\partial e_{k-1}}{\partial w_k^T} = \left\langle \frac{\partial e_{k-1}}{\partial w_k^T} \right\rangle .$$

Proof. Differentiating (6.4.2) with $j = k-1$ gives the desired result since $P_{k-1|0}$ does not depend on w_k. □

Lemma 6.4.4 *The following holds.*

$$\frac{\partial e_{k-1}}{\partial w_k^T} = \left\langle \frac{\partial T_k}{\partial w_k^T}, e_k \right\rangle_{k-1} .$$

Proof. This follows immediately from Theorem 3.4.7 and the definition of e_{k-1}. □

Lemma 6.4.5 *The definition of T_k implies*

$$\frac{\partial T_k}{\partial r_k^T} = A_k r_{k-1} + B_k r_k, \qquad \frac{\partial T_k}{\partial r_{k-1}^T} = A_k^T r_k, \tag{6.4.5}$$

$$\frac{\partial T_k}{\partial A_k^T} = r_k r_{k-1}^T, \qquad \frac{\partial T_k}{\partial B_k^T} = r_k r_k^T, \tag{6.4.6}$$

where the last gradient is taken subject to the condition that $B_k^T = B_k$.

Theorem 6.4.6 *The following holds.*

$$\frac{\partial \langle e \rangle}{\partial A_k^T} = \left\langle \langle r_k r_{k-1}^T, e_k \rangle_{k-1} \right\rangle, \qquad \frac{\partial \langle e \rangle}{\partial B_k^T} = \left\langle \langle r_k r_k^T, e_k \rangle_{k-1} \right\rangle \tag{6.4.7}$$

The first factor in the covariance expression is called *Hebbian factor*. The second factor is called the *reinforcement factor*. The Hebbian factor is also called "characteristic eligibility" in reinforcement learning literature [Wil87, Wil92], following [BSA83].

Notation. Denote by $[\cdot]_{k-1}$ and $[\cdot]$ some approximate time averages approximating $\langle \cdot \rangle_{k-1}$ and $\langle \cdot \rangle$, respectively. The operator $[\cdot]_{k-1}$ can be obtained at the same time when r_k is equilibrating. The operator $[\cdot]$ is computed over the whole learning process. Denote by $[\cdot, \cdot]_{k-1}$ some approximate covariance average process. The exact implementation of these averaging schemes will be studied in detail in §7.4.1.

Definition 6.4.2 (GF-ME learning rule for SQFB-NN) $w_+ = \delta(w, r, v)$, $u = \kappa(w, r, v)$:

1. Output layer. Set $v = e_L$.
2. Hidden layers. For $k = L, \ldots, 1$, update
$$\Delta A_k = \alpha \left[r_k r_{k-1}^T, v\right]_{k-1}, \qquad \Delta B_k = \alpha \left[r_k r_k^T, v\right]_{k-1}.$$
3. Input layer, $u = v$.

Remark 6.4.2 This learning rule, in which the connection between neurons i and j are updated according to $\langle r_i r_j, e\rangle$, is remarkably similar to the Hebbian rule, in which the connection between i and j is updated according to $\langle r_i, r_j\rangle$. This is what is usually called "locally computable": the modification of the connection between two neurons only depends on the activity of these two neurons.

Remark 6.4.3 The GF-ME learning rule for SQFB-NN is much more likely to be used by biological neural networks. The Hebbian rule has the defect that the correlation between two neurons are always enhanced, irrespective of its merit. In our learning rule, two neurons will be positively or negatively correlated if and only if this will increase total evaluation.

Given some running average process, Two simple realizations of the learning rule SQFB-GF-ME can be derived from Corollary 3.3.4 and Theorem 6.4.6.

Definition 6.4.3 (Learning rule SQFB-GF-ME1) Update w_k by

$$\Delta A_k = \alpha \left[\left(r_k r_{k-1}^T - [r_k]_{k-1} r_{k-1}^T\right)(e_L - b)\right], \tag{6.4.8}$$
$$\Delta B_k = \alpha \left[\left(r_k r_k^T - \left[r_k r_k^T\right]_{k-1}\right)(e_L - b)\right], \tag{6.4.9}$$

where b is an arbitrary constant. It can be zero, in particular.

Remark 6.4.4 If $[\cdot]_{k-1} = \langle\cdot\rangle_{k-1}$, the result would be invariant for any b.

Definition 6.4.4 (Learning rule SQFB-GF-ME2) Update w_k by

$$\Delta A_k = \alpha \left[\left(r_k r_{k-1}^T - a\right)(e_k - e_{k-1})\right], \tag{6.4.10}$$
$$\Delta B_k = \alpha \left[\left(r_k r_k^T - a\right)(e_k - e_{k-1})\right], \tag{6.4.11}$$

where a is an arbitrary constant, which can be 0 in particular.

Remark 6.4.5 The function e_{k-1} can be implemented by some DC-NN, which can be updated at the same time layer k is equilibrating. If e_{k-1} were estimated accurately, the result would be invariant for any a.

The running average schemes will always introduce random errors. Considering (3.3.4), we can combine the above two methods to form a more accurate method.

Definition 6.4.5 (Learning rule SQFB-GF-ME3) Update w_k by

$$\Delta A_k = \alpha \left[\left(r_k r_{k-1}^T - [r_k]_{k-1} r_{k-1}^T \right) (e_L - [e_L]) \right], \tag{6.4.12}$$

$$\Delta B_k = \alpha \left[\left(r_k r_k^T - [r_k r_k^T]_{k-1} \right) (e_L - [e_L]) \right], \tag{6.4.13}$$

Remark 6.4.6 Unlike backpropagation (BP) or Boltzmann machine (BM) learning rule, the learning rule derived above is biologically plausible. Call the product $r_i r_j$ the *common activation* of i and j. We can imagine the following scenario in a BNN: the density of a certain chemical substance in each synapse keeps track of a running average of the common activity of pre- and post-synaptic neurons. There is a certain diffuse substance in the network which registers the evaluation for the whole network, and another substance keeps track of its average. The product of the common activity minus its mean and the product of the evaluation minus its mean is then used to modify the synapse. Whether this is actually used in the BNN is of course an empirical question which, given such a specific description, seems not a very difficult task for neuroscientists to answer in the near future. Given that only running averages are used in the above learning rule, there seems unlikely any reason that the BP rule, which requires propagating signal in the opposite direction in each synapse, and the BM rule, which requires two phases of learning, would have been used in BNN. It should also be noted that there is also a very coherent hypothesis regarding how nervous systems would have developed by natural selection: if by any chance the correlation of activity between two cells can be modified, the natural selection would certainly favour those which the modification increase the expected evaluation.

Remark 6.4.7 The above learning rule answers the question of choosing "reinforcement baseline" posed in [Wil87, Wil92]: in the learning rule

$$\delta w_k = \alpha \left\langle \frac{\partial \log p_k}{\partial w_k^T} (e_k - b) \right\rangle, \tag{6.4.14}$$

what should be the best choice of b, the *reinforcement baseline*? It follows Theorem 3.4.7&3.4.8 that this is exactly the GF-ME learning rule. So it follows Corollary 3.3.3 that the answer is $\langle e \rangle$, which confirms his empirical observation.

6.5 GF Learning Converge to Local Optimum in Weight Space

As was pointed out in [MP88], most of the existing learning rules for neural networks are based on gradient-following (GF), and are vulnerable to local optima in the weight space. Numerical experiment on the GF learning rule for SQB-NN [PA87] show that $\langle e \rangle$ as a function of the connection weights is plagued with local maxima. See also Figure 7.19 & 7.20.

Local optima are unavoidable since in general the evaluation function $\langle e \rangle$ is not convex with respect to w. In this section we analyse a particular test problem on which experiments using the GF learning rule generally result in convergence to local optima even for a single layer network. Analysis on this problem will lead to the simulated annealing learning rule discussed in the next section.

Example 6.5.1 Let $m = n = 4$, $X = Y = R^4$, where $R = \{-1, 1\} \cong \{-, +\}$. Each element in R^4 is called a *code*. The input $x \in X$ is generated with $x_i = 1 \iff i = I_{in}$, where the random integer $I_{in} = \mathrm{rnd}(\mathbb{N}_4)$. The output $y \in Y$ is also transformed back to an integer $I_{out} \in \mathbb{N}_4$ by the following rule: Let $J := \{j : x_j = 1\}$; If $J \neq \emptyset$ then $I_{out} = \mathrm{rnd}(J)$; If $J = \emptyset$ then $I_{out} = \mathrm{rnd}(\mathbb{N}_4)$. The evaluation function is defined by $e(x, y) = |I_{out} - I_{in}|^2$.

Obviously there is a unique optimum for this problem, the identity mapping, denoted as $\varphi_1 := [+---|1; -+--|2; --+-|3; ---+|4]$, or simply $[+---; -+--; --+-; ---+]$. Now we use a one layer SQFB-NN (ie., a SQB-NN) with structure N_0-N_1 where $N_0 = N_1 = 4$. The attractor φ_1 can be approximated by this network in many ways; one of the most obvious is to let the connection between the corresponding neurons of the input and output layers grow to positive infinity, while leaving all the other weights zero.

We conducted both stochastic simulation experiments and deterministic (enumerative) numerical simulations with the SQFB-GF-ME3 learning rule on this problem. The main results of stochastic simulations are summarized as the following:

1. When starting from the totally random mapping $\varphi_0 : P_{\varphi_0}(\eta|\xi) = 1/16$ (the mapping corresponding to $w = 0$), the network always converges to the mapping $\varphi_2 := [++--; -+--; --+-; --++]$ (20 experiments).

2. When the weights are started with some random values (independent normal distribution with $\langle w \rangle = 0$ and $\sigma_w := \sqrt{\langle w^2 \rangle} \in \{0.1, 0.2, 0.5, 1.0, 2.0\}$, 10 experiments for each σ_w), many other attractors appear. All of them are deterministic.

3. As σ_w increase the number of attractors increase, and their distribution become more even. Then most frequently appeared attractors include

$$\begin{aligned}
\varphi_3 &:= [+---; -+--; --+-; --++], \\
\varphi_4 &:= [++--; -+--; --+-; ---+], \\
\varphi_5 &:= [++--; ++--; --+-; --++], \\
\varphi_6 &:= [++--; -+--; --++; --++],
\end{aligned}$$

   and φ_1.

4. φ_1 never dominates for any σ_w.

5. If the weights are hand adjusted to within a certain neighbourhood of that of φ_1 (eg., $A = \mu I, B = 0$ with $\mu > 3$), they will converge rapidly to φ_1.

These observations tell us the following : (1) There are many local optima of $\langle e \rangle$ on W; (2) φ_0 is within the attracting basin of φ_2; (3) φ_0 is near the attracting basin of φ_3, φ_4, φ_5 and φ_6 than φ_1; (4) There are many other attractors not much farther away from φ_0 than φ_1. Therefore, with the GF learning rule the network may not learn something which it is capable of representing. It is most likely to be trapped in local optima for this particular example.

Due to the slowness of stochastic simulations, it is not possible to conduct a more complete experiment on this example by stochastic simulation. However, since this problem is still rather small, it is possible to use deterministic (enumerative) method to simulate the stochastic process of learning, that is, to compute the probabilities directly instead of using GS.

Proposition 6.5.1 *In the Example 6.5.1,* $|X| = 4$, $|Y| = 16$, $\dim \mathcal{P}(Y) = 15$, $\dim \mathcal{P}(Y|X) = 60$, $|\mathcal{P}_0(Y|X)| = |Y|^{|X|} = 2^{16} = 65536$, $\dim \mathcal{P}(Y|X) = 15^4 = 50625$, $\dim W = 26$.

The result of deterministic simulation is summarised as follows (with a slight change of notation: $\{0, 1\}$ is used in place of $\{-, +\}$). We define a deterministic attractor φ as $\forall \xi : \exists \eta : P_\varphi(\eta|\xi) > 1/2$. This is justified by the observation that $P(\eta|\xi) \to 1$ if it is larger than $1/2$ at some stage.

1. $\sigma_w = 0.1$. Total 1 deterministic attractor, occurs 100 times in 100 experiments:

$$(100)\ \begin{matrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{matrix}$$

2. $\sigma_w = 0.2$. Total 2 deterministic attractors, occur 100 times in 100 experiments:

$$(99)\ \begin{matrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{matrix} \qquad (\ 1)\ \begin{matrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

Bitwise average over attractors:

$$\begin{matrix} 1.00 & 1.00 & 0.00 & 0.00 \\ 0.00 & 1.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.99 & 1.00 \end{matrix}$$

3. $\sigma_w = 0.5$. Total 6 deterministic attractors, occur 100 times in 100 experiments:

$$(69)\ \begin{matrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{matrix} \qquad (16)\ \begin{matrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix} \qquad (12)\ \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{matrix}$$

$$(\ 1)\ \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix} \qquad (\ 1)\ \begin{matrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{matrix} \qquad (\ 1)\ \begin{matrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{matrix}$$

Bitwise average over attractors:

$$\begin{matrix} 1.00 & 0.86 & 0.00 & 0.00 \\ 0.01 & 1.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 1.00 & 0.01 \\ 0.00 & 0.00 & 0.82 & 1.00 \end{matrix}$$

4. $\sigma_w = 1.0$. Total 14 deterministic attractors, occur 100 times in 100 experiments. The most frequently occurred 8 of them are:

$$(41)\ \begin{matrix}1&1&0&0\\0&1&0&0\\0&0&1&0\\0&0&1&1\end{matrix}\qquad (15)\ \begin{matrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&1&1\end{matrix}\qquad (15)\ \begin{matrix}1&1&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{matrix}\qquad (\ 7)\ \begin{matrix}1&1&0&0\\0&1&0&0\\0&0&1&1\\0&0&0&1\end{matrix}$$

$$(\ 7)\ \begin{matrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{matrix}\qquad (\ 3)\ \begin{matrix}1&0&0&0\\1&1&0&0\\0&0&1&0\\0&0&1&1\end{matrix}\qquad (\ 3)\ \begin{matrix}1&1&0&0\\1&1&0&0\\0&0&1&0\\0&0&1&1\end{matrix}\qquad (\ 2)\ \begin{matrix}1&1&0&0\\0&1&0&0\\0&0&1&1\\0&0&1&1\end{matrix}$$

Bitwise average over attractors:

$$\begin{matrix}1.00&0.70&0.00&0.00\\0.11&1.00&0.00&0.00\\0.00&0.00&1.00&0.13\\0.00&0.00&0.65&1.00\end{matrix}$$

5. $\sigma_w = 2.0$. Total 31 deterministic attractors, occur 99 times in 100 experiments. The most frequently occurred 8 of them are:

$$(17)\ \begin{matrix}1&1&0&0\\0&1&0&0\\0&0&1&0\\0&0&1&1\end{matrix}\qquad (\ 9)\ \begin{matrix}1&1&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{matrix}\qquad (\ 7)\ \begin{matrix}1&1&0&0\\0&1&0&0\\0&0&1&1\\0&0&1&1\end{matrix}\qquad (\ 6)\ \begin{matrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&1&1\end{matrix}$$

$$(\ 5)\ \begin{matrix}1&0&0&0\\1&1&0&0\\0&0&1&1\\0&0&0&1\end{matrix}\qquad (\ 5)\ \begin{matrix}1&1&0&0\\1&1&0&0\\0&0&1&0\\0&0&1&1\end{matrix}\qquad (\ 5)\ \begin{matrix}1&1&0&0\\1&1&0&0\\0&0&1&1\\0&0&1&1\end{matrix}\qquad (\ 4)\ \begin{matrix}1&0&0&0\\0&1&0&0\\0&0&1&1\\0&0&1&1\end{matrix}$$

Bitwise average over attractors:

$$\begin{matrix}0.99&0.64&0.04&0.00\\0.32&0.96&0.08&0.00\\0.00&0.13&0.95&0.36\\0.00&0.01&0.58&0.96\end{matrix}$$

6. $\sigma_w = 8.0$. Total 402 deterministic attractors, occur 495 times in 500 experiments. Bitwise average over attractors:

$$\begin{matrix}0.73&0.63&0.31&0.17\\0.47&0.75&0.49&0.18\\0.17&0.46&0.75&0.47\\0.19&0.33&0.61&0.70\end{matrix}$$

7. $\sigma_w = 8.0$. Total 99 deterministic attractors, occur 100 times in 100 experiments. Bitwise average over attractors

$$\begin{matrix}0.69&0.72&0.41&0.17\\0.49&0.73&0.55&0.22\\0.15&0.55&0.68&0.44\\0.25&0.40&0.56&0.71\end{matrix}$$

The last two items are intended to have a sense of the number of deterministic attractors. It is obvious that the experimental size of 500 is still quite small relative to the number of deterministic attractors (potentially 65536 candidates). The reason that there are some "non-deterministic attractors" (1%) seems due to the termination criteria of the simulation program. All of them are of the form of a half-half mixture of two neighbouring deterministic attractors. It seems likely that given enough iteration, all of them would converge to one of their constituents.

For this relatively simple example we can give an intuitive explanation as to why the network converged to those particular mappings. This is based on a further observation that the learning process can be somewhat divided into two phases. In the first phase the network "tries" to avoid giving $|I_{out} - I_{in}| = 3$, since this is the most important single factor contributing to the overall evaluation. Since the correspondence between the input and the output is not well established, it is very much of the matter of what output should be given in any case, rather than for a given input. What is learned is a stochastic mapping near the mappings

$$\varphi_7 := [-+--;-+--;--+-;--+-].$$

$$\varphi_8 := [++--;++--;--++;--++].$$

In other words, it is a mixture of φ_6, φ_7 and other deterministic mappings, with various probability.

In the second phase the network converges to an attractor determined by the results of the first phase. This can be described as locally adjusting among some already rather good options. However at this stage the weights are already in the attracting basin of local maximum. Further learning only makes the weights tend towards infinity.

Summarizing, the reason that the network converges to local optima is that it adopts whatever the easiest modification which enables it to do better than before, and is unable to survey the other possibilities after becoming fixed.

6.6 Simulated Annealing Learning Rule

One advantage of SQ-NN is that it is natural to implement the parameterised simulated annealing to avoid local optimum. The idea of incorporating SA in the learning process was also suggested in [Hin89a], in relation to the BM learning rule. To the best of our knowledge, we seem to be the first to show its general necessity and applicability to both the ME and ML learning rules. In this section we discuss SA-ME learning rule for SQFB-NN.

6.6.1 Algorithm of SA learning rule

The analysis in §6.5 suggests that to avoid local maxima, there must be some mechanism to ensure that the network is not committed to a particular deterministic mapping in the early stage of learning. This means restraining the magnitude of T_k while maximizing e_k in the learning process. The most mathematically coherent way of doing this is to replace the reinforcement factor e_k in the GF-ME learning rules with $e_k - \theta T_k$, where θ is a positive parameter, which is decreased toward zero as learning proceeds. Like the GF rules, there are several variants to the SA rule. To avoid repetition we only describe the SA-ME3 rule here.

Definition 6.6.1 (Learning rule SQFB-SA-ME3) Update w_k by

$$\Delta A_k = \alpha \left[\left(r_k r_{k-1}^T - [r_k]_{k-1} r_{k-1}^T \right) (e_L \mu - T_k - [e_L \mu - T_k]) \right],$$
$$\Delta B_k = \alpha \left[\left(r_k r_k^T - [r_k r_k^T]_{k-1} \right) (e_L \mu - T_k - [e_L \mu - T_k]) \right],$$

Theorem 6.6.1 *Define* $S_{k-1} := \langle \log P_k \rangle_{k-1}$, $f_{k-1} := e_{k-1}\mu - S_{k-1}$. *Then*

$$\frac{\partial f_{k-1}}{\partial w_k^T} = \left\langle \frac{\partial T_k}{\partial w_k^T}, e_k \mu - T_k \right\rangle_{k-1} \tag{6.6.1}$$

Proof. This immediately follows Theorem 3.4.7 and the fact that $P_{k-1|0}$ does not depend on w_k. □

One consequence of this theorem is that there is no need to compute the entropy $\log P_k$ explicitly. All that is required is T_k, which is always available. This is extremely important since computation of $\log P_k$ is as intractable as computation of the partition function Z_{k-1}, which is as intractable as enumerating all the possible states r_k (§3.5.3).

Remark 6.6.1 For network with more than one layer, our proposed learning rule is to train each layer with reference to its own entropy. As was discussed in §3.5.3, computing the gradient of the total entropy of all the layers requires the explicit computation of entropy in all the subsequent layers, which is as intractable as computing the partition function. Therefore using entropy in each layer separately seems the only practical option. Although we have not found theoretical justification for this, we found numerically that the network always converges to the global optimum if the cooling process is slow enough(See §7.5 for details of results).

6.6.2 Simulations of SA-ME3 learning rule

Numerical experiments on Example 6.5.1 with SA-ME3 learning rule show that this indeed makes the network converges to the global maximum.

In 20 stochastic simulations with starting $w = 0$ and 20 simulations with $\sigma_w = 0.5$, the network always converges to global optimum φ_1, independent of the speed of annealing (γ) and learning (α), as long as they are not large enough to make the computation unstable. The tolerable range are, for example, $\alpha < 0.1$, $\gamma < 0.01$. In §7.4 we shall have more details on these parameters.

In about 30 deterministic simulations with starting $w = 0$, and five groups of 15 deterministic simulations, each group with random starting w for each $\sigma_w \in \{0.1, 0.2, 0.5, 1.0, 2.0\}$, the network also always converges to the global optimum φ_1, independent of the speed of annealing and learning. In such deterministic simulations the time factor (§7.4) can be set to unity, since the statistics instead of samples are used.

This, together with the results in §6.5 shows that (1) At $\mu = 0$ there is only one attractor φ_0; (2) As $\mu \to \infty$ this attractor smoothly moves to φ_1; (3) φ_1 is the global optimum at $\mu = \infty$; (4) At $\mu = \infty$ the starting point φ_0 lies in the attracting basin of φ_2, instead of φ_1. Therefore the SA method is both necessary and sufficient for this particular problem.

An interesting observation from the deterministic simulations is that even if w originally lies in the attracting basin of other attractors (in the sense of e), it will still find its way back to φ_1, even at late stage for μ as large as 5. This rather unexpected phenomenon indicates that for this particular example, the local optima only exists if $\mu = \infty$. However, even if this is true, it is still not practical to set μ to a very large number and wait for the network to converge to global optimum, since the time it takes to transfer from one local optima to another is asymptotically proportional to $\exp(2\mu)$. (For example, if $\mu = 5$ the speed is about 22000 times slower.) In §7.3.1 we shall discuss time scale in more detail.

6.7 Information-theoretical interpretation of SA learning rule

For the sake of brevity, we only consider one layer SQFB-NN, ie. SQB-NN. The network is described by $m, n, M, N \in \mathbb{N}$, $X = R^m$, $Y = R^n$, $W = \mathbb{R}^M$. The environment is characterised by $e \in [Y|X \to \mathbb{R})$ and $p \in \mathcal{P}(X)$. Denote $\mathcal{M} := \mathcal{P}(Y|X)$, then $P \in [W \to \mathcal{M})$. Define $\mathcal{M}_W := \{P_w : w \in W\}$. Then $\dim \mathcal{M} = 2^m(2^n - 1)$, $\dim \mathcal{M}_W \le M \in O(N^2)$.

Definition 6.7.1 For $\mu \in \mathbb{R}_+$, define the *equivalent environmental distribution* $P' := BD(e, \mu) \in \mathcal{P}(Y|X)$:

$$(6.7.1) \quad T' := e\mu, \qquad Z'(\xi) := \sum_\eta \exp T'(\eta|\xi), \qquad P'(\eta|\xi) = \exp T'(\eta|\xi)/Z'(\xi).$$

Define the *equivalent environmental free energy*

$$\langle f' \rangle_x = \sum_\eta P'(\eta|x)(e(\eta|x)\mu - \log P'(\eta|x)).$$

Theorem 6.7.1 *Let $f := e\mu - \log P$, then*

$$-K(pP, pP') = \langle f \rangle - \langle \log Z' \rangle, \qquad -\frac{\partial K(pP, pP')}{\partial w} = \frac{\partial \langle f \rangle}{\partial w},$$

and

$$K(pP, pP') = 0 \iff \langle f \rangle = \min.$$

Proof. It can be easily checked that

$$\begin{aligned} -K(pP, pP') &= -\langle K(P, P')_x \rangle = \sum_{x\eta} p(x)P(\eta|x)\left(\log P'(\eta|x) - \log P(\eta|x)\right) \\ &= \sum_{x\eta} p(x)P(\eta|x)\left(e(\eta|x)\mu - \log Z'(x) - \log P(\eta|x)\right) = \langle f \rangle - \langle \log Z' \rangle. \end{aligned}$$

The proof of the theorem is completed by noting the Z' does not depend on w. □

This means that the SA-ME learning rule is equivalent to approximating $P' \in \mathcal{M}$ by $P \in \mathcal{M}_W$ in the sense of minimising $K(P, P')$. This interpretation was inspired by [NS88], where P' is called *target distribution* and P is called *system distribution*. As $\mu \to \infty$, P' moves towards $\partial\mathcal{M}$, the boundary of $\mathcal{M}$, while P follows in $\mathcal{M}_W$. It is *a priori* not clear whether $P \to \partial\mathcal{M}$ and whether $\langle e \rangle$ will increase, since $\mathcal{M}_W$ is in general a curved submanifold in $\mathcal{M} \cong (\Delta^{2^n-1})^{2^m}$, such that $\dim \mathcal{M}/\dim \mathcal{M}_W$ can be enormous.

Proposition 6.7.2 *For a fully connected 4-4 SQB-NN with input in R^4,* $\dim \mathcal{M} = 2^4(2^4-1) = 240$. $\dim W = 4(4+1) + \frac{1}{2}4(4-1) = 26$. $|[X \to Y)| = |\mathcal{P}_0(Y|X)| = |Y|^{|X|} = 6.6 \times 10^{18}$. *For a fully connected 8-8 SQB-NN with input in R^8,* $\dim \mathcal{M} = 2^8(2^8-1) = 65280$. $\dim W = 8(8+1) + \frac{1}{2}8(8-1) = 26$. $|[X \to Y)| = |\mathcal{P}_0(Y|X)| = |Y|^{|X|} = 10^{616}$.

The success of the SA learning rule seems to suggest that the submanifold is "sufficiently flat" so that the initial global optimum will always deform to the final global optimum as $\mu \to \infty$. Considering Proposition 6.7.2, we have to simplify the problem to a certain degree to make is testable by experiments. We therefore consider a network with $X = \emptyset$ here. Our first guess is the following conjecture.

Conjecture 6.7.1 Let $Y \cong \mathbb{N}_n$, $W = \mathbb{R}^N$, $e \in \mathbb{R}^n$, and $\forall k \in \mathbb{N}_N : T_k \in \mathbb{R}^n$. Define $\forall w \in W : T_w := \sum_k w_k T_k \in \mathbb{R}^n$, $\forall \mu \in \mathbb{R}_+ : f := e\mu - \log P_w \in \mathbb{R}^n$, $P_w \in \mathcal{P}(Y) : P_w \sim T$. Define $W_\mu := \{w \in W : \langle f \rangle_w = \max_w \langle f \rangle_w\}$. Then

$$\exists w^* \in C[\mathbb{R}_+ \to W) : \forall \mu \in \mathbb{R}_+ : w^*_\mu \in W_\mu. \tag{6.7.2}$$

Note that f depends on both w and μ.

As it stands, this conjecture is not correct, due to the following counter-example.

Example 6.7.1 Let $N = 3$, $n = 3$, $M = 1$, $e = [2, -3, 1]$, $T_1 = [2, 1, -3]$. Then W_μ is not continuous on μ. There is one branch of local optima $w^1_\mu \to -\infty$ and another $w^2_\mu \to \infty$, such that $\exists \mu^0 : \forall \mu < \mu_0 : W_\mu = \left\{w^1_\mu\right\}, \forall \mu > \mu_0 : W_\mu = \left\{w^2_\mu\right\}$.

The curves of $\langle f \rangle$ and $\langle f \rangle / \mu$ versus w on $\mathcal{M}_W$ are shown in Figure 6.1 (a) and (b). The contours of $\langle f \rangle$ and $\langle f \rangle / \mu$ over μ and w are shown in Figure 6.1 (c) and (d). For $\mu < 0.5860$ there is only one global optimum at $w^1_0 = 0$. As $\mu \to \infty$, $w^1_\mu \to -\infty$. However, for $\mu \geq 0.5860$, there appears a new local optima w^2_μ, $w^2_0 = 1.64$, and $w^2_\mu \to \infty$ as $\mu \to \infty$, rapidly overtaking w^1_μ to become a global optima. A more vivid picture of $\langle f \rangle$ versus μ and w are shown in Figure 6.2.

The contour of $\langle f \rangle$ on the whole $\mathcal{M}$, superimposed with the position of $\mathcal{M}_W$, is shown in Figure 6.3. Corresponding 3D graphs are shown in Figure 6.4 (without the position of $\mathcal{M}_W$.)

It is quite clear from these graphs that the main reason for the failure to converge to global optimum is that $\mathcal{M}_W$ is too curved: it is possible to have more than two points on $\mathcal{M}_W$ which are locally closest to P' in the sense of $K(P, P')$.

Although Example 6.7.1 is almost the most complicated of such type which can be represented graphically, it is too simple to correspond to any interesting neural network. A two neuron network (without input) will have four states, while a one neuron network, having only two states, does not allow for $\mathcal{M}_W$. Considering the quasilinear neural networks, it is clear that T_k are not allowed to be arbitrary function. Therefore we make the following more realistic conjecture.

Conjecture 6.7.2 Let $R = \{-1, 1\}$, $Y = R^n$, $W = \mathbb{R}^M$, $e \in [R^n \to \mathbb{R})$, and $\forall k \in M : T_k \in (L + L^2)[R^n \to \mathbb{R})$, where $(L + L^2)$ denote the space of sum of linear functions and 2-linear function. Define $\forall w \in W : T_w := \sum_k w_k T_k \in (L + L^2)[R^n \to \mathbb{R})$. $\forall \mu \in \mathbb{R}_+ : f := e\mu - \log P_w \in [R^n \to \mathbb{R})$, $P_w \in \mathcal{P}(Y) : P_w \sim T$. Define $W_\mu := \{w \in W : \langle f \rangle_w = \max_w \langle f \rangle_w\}$. Then

$$\exists w^* \in C[\mathbb{R}_+ \to W) : \forall \mu \in \mathbb{R}_+ : w^*_\mu \in W_\mu. \tag{6.7.3}$$

Proposition 6.7.3 *Let notation be as in Conjecture 6.7.2, then* $\dim \mathcal{M}_W = |\{[i, j] : i \in \mathbb{N}_n, j \in \mathbb{N}_{i-1}\}| = \frac{1}{2}n(n+1)$, *and* $\dim \mathcal{M} = 2^n - 1$. *For* $n = 1$, $\dim \mathcal{M}_W = \dim \mathcal{M} = 1$. *For* $n = 2$, $\dim \mathcal{M}_W = \dim \mathcal{M} = 3$. *For* $n = 3$, $\dim \mathcal{M}_W = 6$, $\dim \mathcal{M} = 7$. *For* $n \geq 3$, $\dim \mathcal{M}_W < \dim \mathcal{M}$.

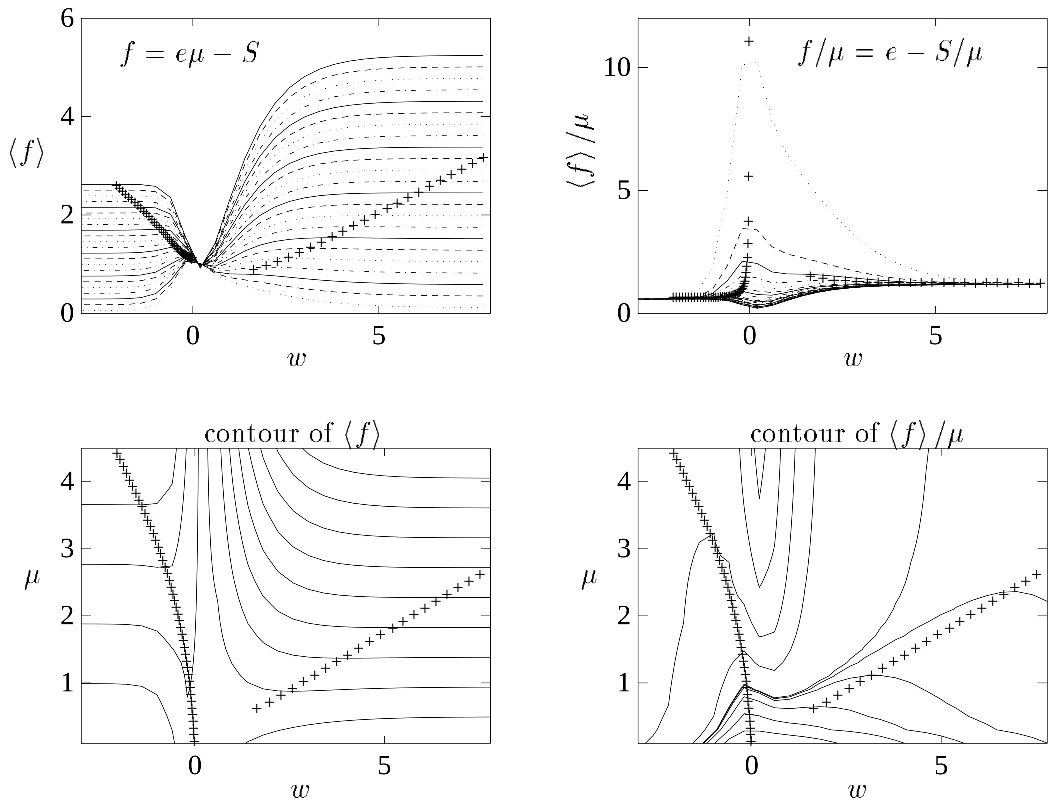

Figure 6.1: Curves of $\langle f \rangle$ versus w for various μ and contours of $\langle f \rangle$ over w and μ on $\mathcal{M}_W$

The curves represented by + indicate the location of local optima. The local optima to the left are w^1_μ; those to the right are w^2_μ, which only exist for $\mu \geq 0.5860$.

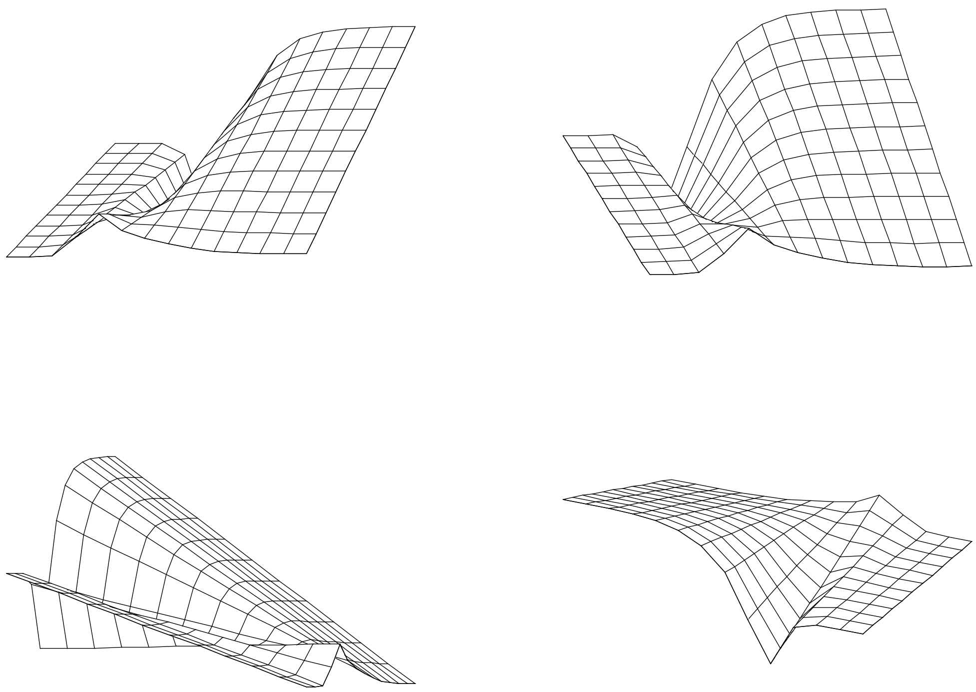

Figure 6.2: 3D graphs of $\langle f \rangle$ versus μ and w on $\mathcal{M}_W$

Refer to Figure 6.1 for explanation.

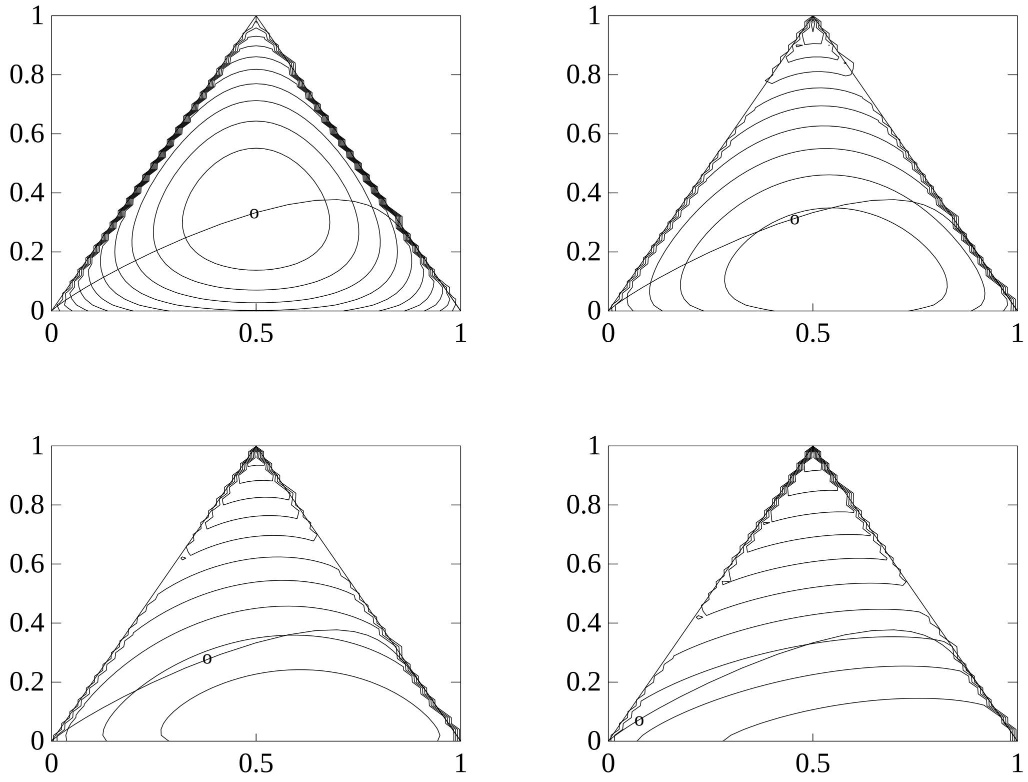

Figure 6.3: Contour of $\langle f \rangle$ on $\mathcal{M}$ for four different μ, superimposed with the position of $\mathcal{M}_W$ and w^1_μ

The four graphs are for $\mu \in \{0, 0.3, 1.0, 3.0\}$, respectively. The big triangle is $\mathcal{M} \cong \Delta^2$. The three corners of the triangle are, from lower right hand corner anti-clockwise, $[1, 0, 0]$, $[0, 1, 0]$ and $[0, 0, 1]$, respectively. The curve connecting $[1, 0, 0]$ and $[0, 0, 1]$ is $\mathcal{M}_W$. The small circle on $\mathcal{M}_W$ is the location of the local optimum of $\langle f \rangle$ corresponding to w^1_μ. The zigzagging at the boundary of the triangle is only an artifact of the finite precision of computation.

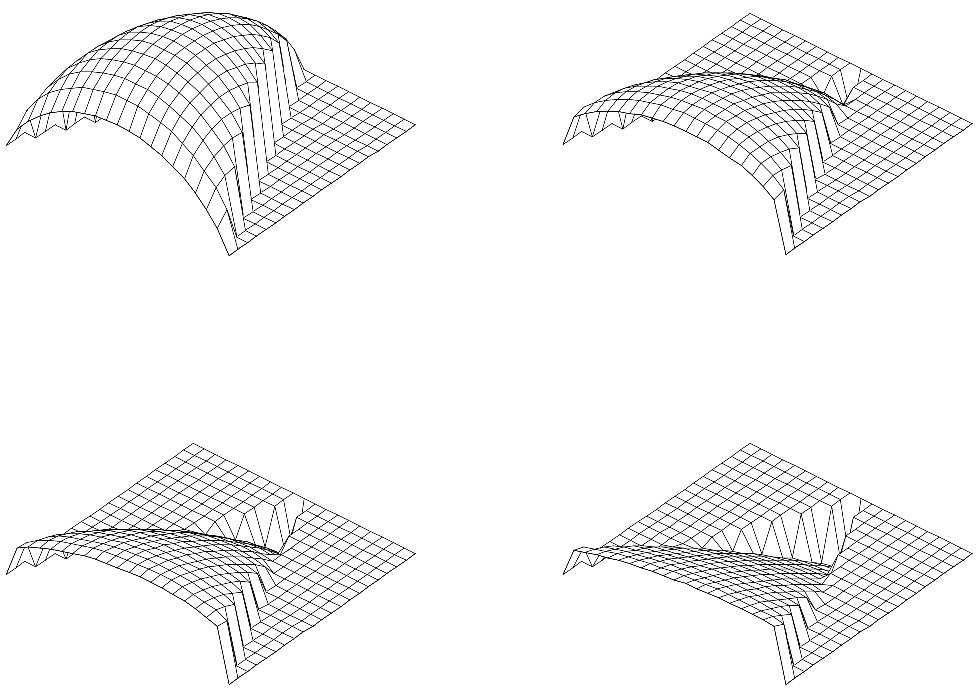

Figure 6.4: 3D graphs of $\langle f \rangle$ on $\mathcal{M}$ for four different μ

See Figure 6.3 for explanation. Note that the first subgraph is exactly the graph of entropy on Δ^2.

Conjecture 6.7.2 means that if the state space Y is represented by the state of neurons in a network, and if T is an at most 2-linear function on the neuronal states, then the submanifold $\mathcal{M}_W$ will be flat enough to guarantee the existence of w^*_μ which connects the global optima at $\mu = 0$ and $\mu = \infty$.

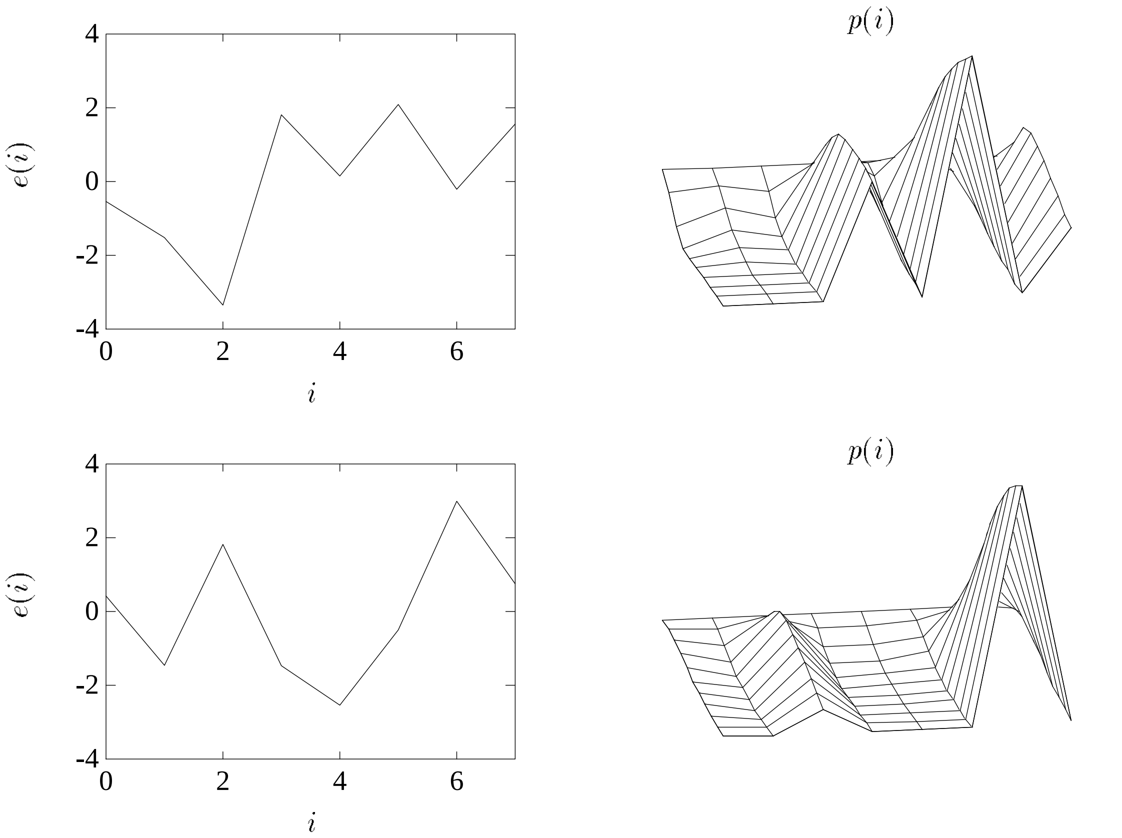

Figure 6.5: Learnt probability distributions $p(i)$ at different μ for a three-neuron network with two randomly generated evaluation functions $e(i)$

The symbol i represent the binary number corresponding to the neuronal state. $|Y| = 8$, $|W| = 6$. Note that in the second example it is quite clear that $p(6) \to 1$ while $p(i) \to 0, \forall i \neq 6$, evan though $p(2)$ increases initially.

Our numerical results indeed support this conjecture. We have performed deterministic simulations with e taken from a normal distribution. For $n = 3$, we have conducted 40 experiments. For $n = 4$, we have conducted 20 experiments. For $n = 5$, we have conducted 2 experiments (since it is quite slow at this stage). All of them agree perfectly with Conjecture 6.7.2. Samples of these results are shown in Figure 6.5, Figure 6.6 and Figure 6.7.

Summarising, we have shown that for a particular e as defined by Example 6.5.1, a SQB-NN with GF learning rule almost never converges to global optimum, while with SA learning rule it always converges to global optimum. For randomly generated e the SA learning rule also converges to global optimum reliably. Therefore it is very likely

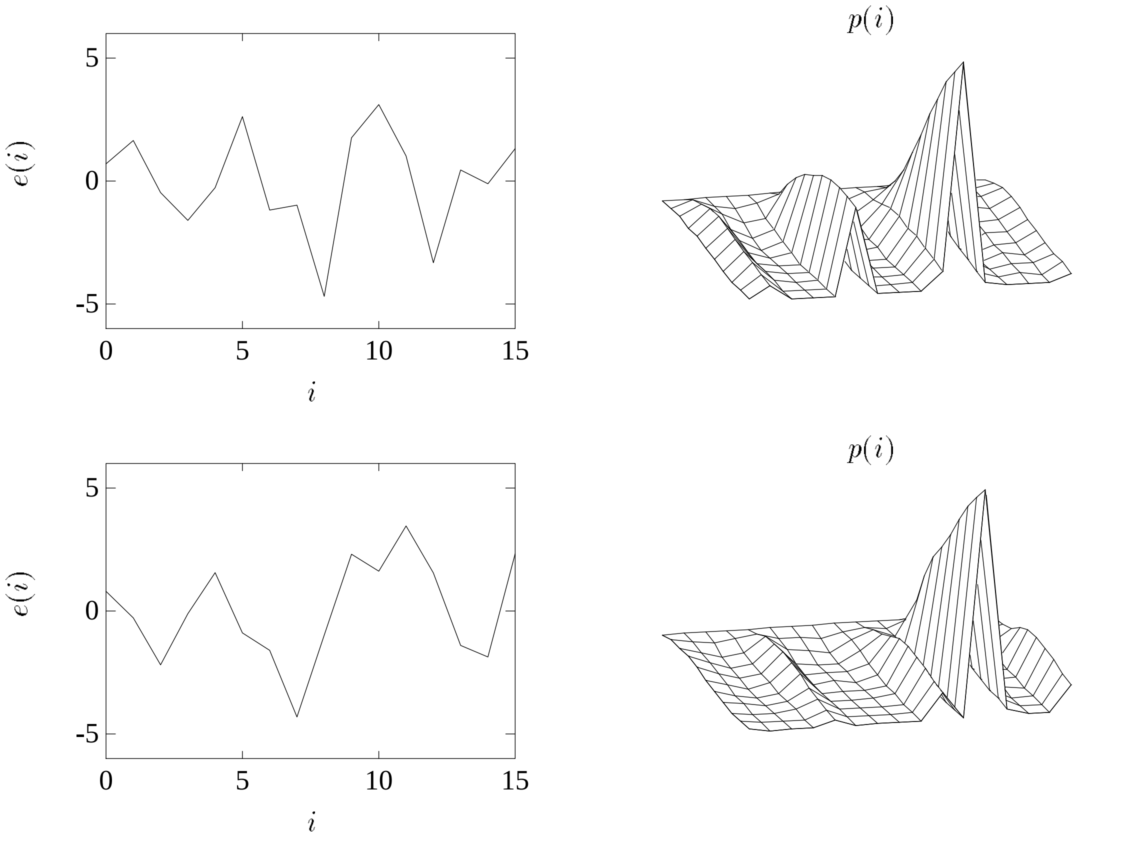

Figure 6.6: Learnt probability distributions $p(i)$ at different μ for a four-neuron network with two randomly generated evaluation functions $e(i)$

The symbol i represent the binary number corresponding to the neuronal state. $|Y| = 16$, $|W| = 10$.

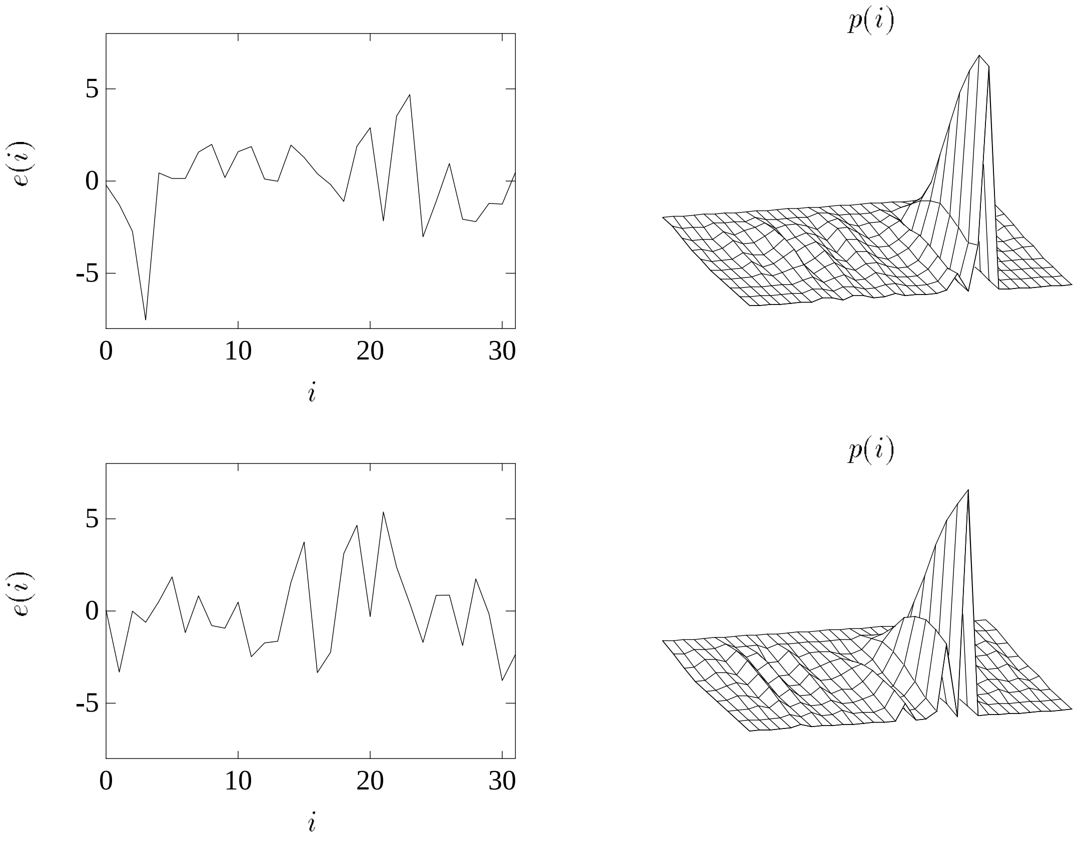

Figure 6.7: Learnt probability distributions $p(i)$ at different μ for a five-neuron network with two randomly generated evaluation functions $e(i)$

The symbol i represent the binary number corresponding to the neuronal state. $|Y| = 32$, $|W| = 15$.

that Conjecture 6.7.2 is true. In the next chapter will shall shall that this is also true for SQFB-NN with more than one layer. So the conclusion of Conjecture 6.7.2 may hold for weaker conditions (where hidden neurons are allowed).

Since our current method is both necessary and sufficient for all the experiments reported here and in Chapter 7, including all the cases where GF-ME learning rules do converge to local optimum, it is necessary for any successful future theory of learning to explain this phenomenon.

Remark 6.7.1 The SA learning rule at least does away with those undesirable features of the original BM learning rule [HS86] ("Ways in which the learning algorithm can fail"). This is certainly a more reasonable method than commonly used technique of "weight decay" [Hin89a], since the latter is equivalent to using the Euclidean norm on the weight space as a barrier function, which does not have a specific meaning. However, the qualitative similarity between Euclidean norm and the entropy at a gross level is high enough for the two methods to behave similarly in many circumstances, but weight decay would show its shortcoming in a long sequence of computations [Jay57a]. In general, Euclidean norm is good for linear spaces while entropy is good for simplexes.

Remark 6.7.2 Now we remark briefly on the role of hidden neurons. Any NN which needs hidden units must have local optima. The hidden unit must take different values to convey information; If it constantly takes the average value it effectively does not exist. Since there is no *a priori* preference between -1 and 1 for the hidden units, and no preference to which hidden unit should represent what feature, any local optimum is accompanied by a *permutation group* of other local optima with the same evaluation [Sus92].

Chapter 7

Implementation of SQFB Neural Networks

7.1 Introduction

We discuss the implementational issues concerning SQFB-NN which arise from the finite precision and finite speed of running average processes. In Section 7.2, an efficient implementation of Gibbs samplers on conventional computer is introduced, by which the computational time is bounded independent of the computational temperature. It is a straight forward computational method instead of the trial-and-error methods conventionally used. In Section 7.4, we discuss various aspects concerning finite speed of computation, including the speed of averaging, the speed of learning (rate for reducing weights) and cooling schedule (rate for reducing the temperature). In Section 7.5, the simulation results and more technical details are given. The results are also compared with previous results.

7.2 Simulation of Gibbs Samplers

At the heart of simulated annealing, the Boltzmann machine and many other statistical applicationsis a Gibbs sampler (GS) [GG84]. (See [Yor92] for a survey of the applications of GS.)

Let X be the state space, $e \in [X \to \mathbb{R})$ and $\tau \in GS(BD(e, \mu))$. Then for each $x^0 \in \mathcal{P}(X)$, it will produce a random process $[x^t : t \in \mathbb{N}]$ such that $x^{t+1} \in U(x^t)$, where $U(x^t)$ is in a certain neighbourhood of x^t, and that $p_{x^t} \to p := BD(e, \mu) : p \sim \exp(T)$, where $T := e\mu$. A sufficient and necessary condition for the stationary probability distribution to be the Boltzmann distribution is the *detailed balance* (eg. [Ami89]):

$$\text{(7.2.1)} \qquad \frac{p_{x^{t+1}|x^t}(\xi|\eta)}{p_{x^{t+1}|x^t}(\eta|\xi)} = \exp(T(\xi) - T(\eta)), \qquad \forall \xi, \eta \in X.$$

We shall only consider GS with $U(x) = \{y \in X : d(x, y) \le 1\}$, where $X = R^n$, $R = \{-1, +1\}$ and d is the Hamming distance. Such a GS can be described by repeatedly setting $i = \mathrm{rnd}(\mathbb{N}_n)$, and set $x_i^{t+1} = \pm 1$ with probability $p^{i(\pm)}(x^t)$. The two probabilities $p^{i(\pm)}$ can be further decomposed into $p^{i(+|+)}$, $p^{i(-|+)}$, $p^{i(+|-)}$, and $p^{i(-|-)}$. Since $p^{i(+|+)} + p^{i(-|+)} = 1$ and $p^{i(+|-)} + p^{i(-|-)} = 1$, there are only two independent parameters. The detailed balance gives

$$\frac{p^{i(+|-)}}{p^{i(-|+)}} = \exp(\Delta_i T),$$

so only one independent parameter left. There are two important GS in this one-parameter family: the Metropolis process [MRR+53] and the local Boltzmann process [AHS85]. The Metropolis process is defined by $\max\{p^{i(+|-)}, p^{i(-|+)}\} = 1$, or equivalently, $\forall a \in \{-1, 1\}$:

$$p^{i(a|-a)}(x) = \begin{cases} 1, & a\Delta_i T(x) \geq 0; \\ \exp(a\Delta_i T(x)), & a\Delta_i T(x) < 0 \end{cases} \tag{7.2.2}$$

The local Boltzmann process is defined by $\forall a \in \{-1, 1\}: \ p^{i(a|+)} = p^{i(a|-)}$, or equivalently,

$$p^{i(a)}(x) = 1/(1 + \exp(-a\Delta_i T(x)). \tag{7.2.3}$$

Process (7.2.3) is theoretically more appealing since it is the Boltzmann distribution of x_i conditional on all the other bits, and the probability is a continuous function of the $\Delta_i T$ independent of the previous value of x_i, but it is more time consuming. As $|\Delta T|$ increases, the ratio $p^{i(+|-)}/p^{i(-|+)}$ will increase exponentially, so that the duration in which no state transition occurs also increase exponentially. One way to compress such redundant computation is to raise the probability of transition between each pair of states without violating the detailed balance. The extreme along this line is just the Metropolis process, which is the fastest *local* method, in the sense that updating each bit requires only locally available information.

The computation can still be compressed if we are simulating the process on a digital computer with pseudo random number generator (PRNG), instead of using hardware simulation with physical noise. To avoid unnecessary complications, it is better to change to continuous time formulation. The continuous-time counterpart of (7.2.3) is the Glauber process [Gla63].

Denote $p_i := p^{i(-x_i)}(x)$ as in (7.2.3). Let the updating be made at interval δt, with $\Pr\left\{x_i^{t+\delta t} = -x_i^t | i\right\} = P_i(x)$. Let $t_0 = 0$ and $t_{k+1} := \sup\{t : x^\tau = x^{t_k}, \forall \tau \in [t_k, t]\}$. The time interval $[t_k, t_{k+1})$ is called a *phase*, the duration in which no bit has flipped. The *waiting time* of phase k is $\Delta t_k := t_{k+1} - t_k$. The Glauber process is defined as $\delta t \to 0$ with $P_i/\delta t \to p_i$.

Theorem 7.2.1 *Suppose $\delta t \to 0$ and $P_i/\delta t \to p_i$. Then Δt_k tends to a random number from an exponential distribution with mean $1/p$, where*

$$p := (1/n) \sum_i p_i. \tag{7.2.4}$$

Proof. Since there is $1/n$ probability of choosing i, as $\delta t \to 0$,

$$\Pr\{x^{t+\delta t} \neq x^{t_k} | x^t = x^{t_k}\}/\delta t = \sum_i \frac{P_i}{n\delta t} \to p, \qquad \forall t \geq t_k.$$

Let $t \geq t_k$. Denote $F_k(t) := \Pr\{t < t_{k+1}\}$. Then the above equation implies

$$\frac{F_k(t) - F_k(t+\delta t)}{F_k(t)\delta t} = \Pr\{t_{k+1} < t + \delta t | t < t_{k+1}\}/\delta t \to p, \qquad \forall t \geq t_k.$$

That is,

$$-\frac{1}{F_k(t)}\frac{dF_k(t)}{dt} = p, \qquad \forall t \geq t_k.$$

Since $F_k(t_k) = 1$, the solution is $F_k(t) = \exp(-p(t-t_k))$, $t \geq t_k$. In other words, $\forall \tau > 0 :\ \Pr\{\tau < \Delta t_k\} = \exp(-p\tau)$. □

Remark 7.2.1 Intuitively, P_i is the probability of flipping bit i when it is chosen, p_i is the probability of flipping i in unit time when it is chosen, p is the probability of flipping any bit in unit time.

It is obvious that the probability of unit i being flipped at the end of phase k is

$$\Pr\left\{x_i^{t_{k+1}} = -x_i^{t_k}\right\} = p_i / \sum_j p_j =: q_i. \tag{7.2.5}$$

In summary, the Glauber process can be described as

1. Set $t_0 = 0$. Set $x^0 = \mathrm{rnd}(X)$.
2. Wait time Δt_k taken from exponential distribution with mean $1/p$, where p is defined by (7.2.4) and (7.2.3).
3. Set $i = \mathrm{rnd}(\mathbb{N}_n, q)$, where q is defined by (7.2.5).
4. Set $t_{k+1} = \min\{t_k + \Delta t_k, t_{\mathrm{lim}}\}$. Set $x^{t_{k+1}} = x^{t_k}(i)$, where the notation $x(i)$ denote the vector formed by flipping x_i while keeping all the other x_j fixed.
5. Until $t_{k+1} = t_{\mathrm{lim}}$, goto step 2.

When the Gibbs sampler is implemented on a conventional computer, either as part of a simulation of stochastic neural networks, or as part of SA algorithm in its proper sense, all that is required is x^{t_k} and Δt_k. There is no need to actually wait for a time Δt_k. So our proposed simulation procedure is to compute the waiting time Δt_k explicitly, instead of wait for such a duration, and use the computed waiting time in the averaging process. This process stops when the computed accumulated time exceeds the prescribed stopping time t_{lim}. This method is locally computable except for the computation of $\sum_i p_i$. So it can also be used in simulations on parallel computers.

What remains to be decided is the stopping time, this obviously depends on both the precision required and on the stationary distribution itself. We found empirically that it appears to be a good choice to take $t_{\mathrm{lim}} = 10N\langle\Delta t\rangle$, where N is the number of neurons in the current layer and $\langle\Delta t\rangle$ is the average waiting time. In the experiments in §7.5, we have also tried to set t_{lim} to be much larger than that, this substantially slows down the computation without any discernible improvements. Smaller t_{lim} sometimes give unstable results. Since the time saving from this method over the original GS is enormous (we observed $\langle\Delta t\rangle > 100$ in many occasions), we have not conducted separate experiments to optimise such choices.

Definition 7.2.1 The process of approaching equilibrium is called *equilibration*.

In the simulation, an equilibration is the process allowing the network to relax to equilibrium, which is one call to the Gibbs sampler. The above choice of stopping time means that on average each neuron flips 10 times in one equilibration, or, there are, on average, $10n$ phases in one equilibration. By the above method the time spent at simulating the equilibration is independent of the temperature θ, while by the original GS it increases exponentially with $\mu = 1/\theta$ (§7.3.3).

Remark 7.2.2 The following are some lessons from our programming experience with this method: (1) It is necessary to have the waiting time taken from the exponential distribution, instead of set to the expected waiting time. Otherwise it is possible that the final states are correlated with the initial state. (2) When the time limit is exceeded, it is the state which exceeds this limit which is to be taken as the final state, not the state which is to succeed this state. (3) The accumulated time should be cut at the stopping time, no matter how long the computed time of the last phase is. (4) The initial $\langle \Delta t \rangle$ should be set to 2, since at infinite temperature there is 1/2 probability to flip any bit.

Remark 7.2.3 It seems prudent to point out here that the above analysis are based on the assumption that the random numbers are genuine. How the regularities in the pseudo-random number generator [Rip87] might affect the above results is an open question, although we have not found any discernible such effects in the experiments.

7.3 Encoder Problems and Ideal Learning

The learning rules of SQFB-NN are derived under the assumption that all the processes can be performed arbitrarily slowly. In this section we analyse the encoder problem, which provides a concrete example allowing us to discuss the speed of the these processes in the next section.

7.3.1 Scaling convention for evaluation and temperature

It is obvious that the temperature θ and the evaluation e can be multiplied by a same factor without altering the actual process. The function e can also be translated by any constant. We require that the learning process should not be affected by such transformations. Therefore it is necessary to represent e and θ in a unique normalised form. After considering several possibilities, we stipulate the following convention:

Definition 7.3.1 The *normalised evaluation* e is an affine transform of the original evaluation such that its mean and variance be

$$\langle e \rangle = 0, \qquad \sigma^2 := \left\langle (e - \langle e \rangle)^2 \right\rangle = \left\langle e^2 \right\rangle = n, \tag{7.3.1}$$

where n is the number of output units, at the start of learning (usually with the totally random mapping, ie., with zero weights).

Remark 7.3.1 This convention has the advantage that the critical temperature for the encoder problem will be independent of the size of the problem, and that the scaling can be obtained before any learning.

7.3.2 The class of encoder problems

We use as our primary example the well known *encoder problem* [AHS85, RHW86], which is described as the following: Let $R = \{-1, 1\}$, $n \in \mathbb{N}$. Define $R^n_1 := \{x \in R^n : |\{i : x_i > 0\}| = 1\}$. Let $X = Y \subseteq R^n$. Given a random variable $x \in X$ with an unknown distribution $p \in \mathcal{P}(X)$, and a parameterised stochastic mapping $y = \varphi_w(x) \in Y$ with conditional distribution $P_w(y|x)$, try to modify w so that y will be an approximation of x in either of the following two senses:

ME: It is required that $\langle d(x, y)\rangle \to 0$, where $d \in [Y^2 \xrightarrow{P} \mathbb{R}_+)$ and $d(\xi, \eta) = 0 \iff \xi = \eta$ [RHW86], or, equivalently, $\langle e\rangle \to \max$, where e is normalised from $-d$ with Definition 7.3.1

ML: It is required that $K(P', P) \to 0$ or $K(P, P') \to 0$, where K is the Kullback separation, P is the network distribution and P' is distribution corresponding to identity mapping [AHS85, LL88].

As we have seen in §6.6, these two criteria are related by

$$\frac{\partial K(P, P')}{\partial w} = \frac{\partial \langle f\rangle}{\partial w},$$

where $f = e\mu - \log P(y|x)$ and $P' = BD(e, \mu)$.

Remark 7.3.2 This formulation of encoder problem encompasses all the so-called "auto-association" problems in NN literature [Lip87]. The example in §6.5 is also an encoder problem, where the function d is stochastic. If d is the *Hamming distance* $d(x, y) := |\{i : x_i \neq y_i\}| = \|x - y\|_1/2$, the problem is called *Hamming distance encoder*.

Depending on the input distribution, two types of encoder problems may be used as a test problems. The *random code encoder problem* is defined by selection a "random" subset X of R^n as the input space. More precisely, we choose m codes from R^n with uniform distribution repeatably, and label them x^1 through x^m. Then the input distribution is uniform distribution mapped to $\{x^k : k \in \mathbb{N}_m\}$. The *unit code problem* is defined by setting the input as a random variable uniformly distributed, over R^n_1, set of codes which are "unit vectors".

7.3.3 Ideal learning

The learning process is said to be *ideal learning* if the distribution of output is identical to the equivalent environmental distribution, ie., it is the Boltzmann distribution associated with the evaluation function $P = BD(e, \mu)$. In the following we always denote $\theta = 1/\mu$.

Definition 7.3.2 Denote $\sigma^2 := \langle\langle e, e\rangle_x\rangle$. Define the *heat capacity* $C := -d\langle e\rangle/d\theta$. The maxima of C are called *peak heat capacity*, denoted C_c. The θ where C is maximal is called *critical temperature*, denoted θ_c. Denote $\mu_c = 1/\theta_c$.

Theorem 7.3.1 *For ideal learning, at any μ, it holds that*

$$T(y|x) = e(y|x)\mu, \qquad \sigma^2 = \frac{d\langle e\rangle}{d\mu}, \qquad C = \mu^2\sigma^2.$$

Proof. The first identity follows trivially from Theorem 5.2.1. The second identity follows Theorem 3.4.7, as

$$\frac{d\langle e\rangle}{d\mu} = \sum_x p_x \frac{dP_{y|x}}{d\mu} e_{y|x} = \sum_x p_x \left\langle \frac{dT_{y|x}}{d\mu}, e_{y|x} \right\rangle_x = \sum_x p_x \left\langle e_{y|x}, e_{y|x} \right\rangle_x$$

The third identity then follows from the fact that $\theta = 1/\mu$. □

Remark 7.3.3 The second and third identities in the above appeared in [AK89b], but they seem to have been well known in statistical mechanics (when there is no x) [KGV83, NS88].

7.3.4 Ideal learning for the Hamming distance encoder problem

We consider Hamming distance encoder here, which has some very nice features which makes it easy to be treated analytically. Later we shall see that other encoder problems are almost identical numerically, although being much more complicated analytically. Therefore Hamming distance encoder problem will serve as an ideal prototype for our later analysis.

Theorem 7.3.2 *Denote $F_k(x) := \{y \in R^n : d(x, y) = k\}$. Let e be normalised from $-d$, the Hamming distance on R^n. Then $\forall x \in R^n : |F_k(x)| = C_n^k$, and $\forall y \in F_k(x)$: $e(y|x) = x^T y = n - 2k =: e_k$*

Proof. $F_k(x)$ is the set of codes with exactly k bits different from x. Therefore $|F_k(x)| = C_n^k$. Since the evaluation e is normalised from $-d$, we can assume $e_k = c - ak$, where $c, a \in \mathbb{R}$ are constants. The normalising conditions

$$\sum_k C_n^k e_k = 0, \qquad \sum_k C_n^k e_k^2 = n \sum_k C_n^k,$$

and the identities (A.5.1) and (A.5.2) imply $c = n$ and $a = 2$. Since $d(x, y) = (n - x^T y)/2 = k$ for $y \in F_k(x)$, it follows that $e(y|x) = x^T y$. □

Lemma 7.3.3 *Denote $p := \exp(-2\mu)$. Then*

$$\frac{1-p}{1+p} = \tanh\mu, \qquad \frac{4np}{(1+p)^2} = 1 - \tanh^2\mu = \operatorname{sech}^2\mu.$$

Theorem 7.3.4 $P_k := \Pr\{y \in F_k(x)|x\}$. *Then for ideal learning on Hamming distance encoder,*

$$P_k = p^k/(1+p)^n, \tag{7.3.2}$$

$$\langle e\rangle = n\tanh\mu, \qquad \sigma^2 = n\operatorname{sech}^2\mu, \tag{7.3.3}$$

$$\langle T\rangle = n\mu\tanh\mu, \qquad C = n\mu^2\operatorname{sech}^2\mu. \tag{7.3.4}$$

Proof. It follows from $e_k = n - 2k$ that P_k is proportional to p^k. Summing P_k over k with coefficient C_n^k, using (A.5.1), gives (7.3.2). The identities in (7.3.3) then follow Definition 7.3.2, (7.3.2) and Lemma 7.3.3. The identities of (7.3.4) follow trivially from the above and Theorem 7.3.1. □

Theorem 7.3.5 *For the Hamming distance encoder in ideal learning, the heat capacity has a unique maximum at $\mu \tanh \mu = 1$. Its solution is $\mu_c = 1.19967864$, $\theta_c = 0.8335565$, and $C_c = n(\mu_c^2 - 1) = 0.439228839n$. The critical temperature is also determined by $\langle T \rangle = n$.*

Proof. Differentiating C with respect to μ, we get

$$\frac{dC}{d\mu} = 2n\mu \operatorname{sech}^2 \mu (1 - \mu \tanh \mu).$$

which equals zero if and only if $\mu \tanh \mu = 1$. The conclusion follows the fact that C tends to zero when μ either tends to zero or infinity. $\square$

Remark 7.3.4 These beautiful mathematical relations in the above two theorems also give some support to the normalisation convention we adopted.

For non-ideal learning, the *nominal heat capacity* $\sigma^2\mu^2$ is usually larger than the true heat capacity, especially if the network is trapped in some local optimum. We shall simply call it heat capacity, since the true heat capacity is hard to find out and never used here.

Definition 7.3.3 We call the graphs of $\langle e \rangle / n$, σ^2/n, $\langle T \rangle / n$ and C/n versus $\mu = 1/\theta$ the TE, TS, TT and TC graphs, respectively. Those of the same variables versus the numbers of presentations are called IE, IS, IT and IC graphs, respectively.

These curves are shown as solid curves in Figure 7.17. These graphs reveal important characteristics of the learning process. Intuitively, a good learning schedule should, on the one hand, follow the TE, TS, TT and TC graphs of the ideal learning as closely as possible so as to maintain quasi-equilibrium at all temperatures, while on the other, converge as fast as possible on the IE, IS, IT, IC graphs.

7.3.5 The blurred identity mapping encoder problem

The experiments on the Boltzmann machine (BM) reported in [AHS85] were conducted on the *blurred identity mapping encoder problem*: The network distribution $P = P_w$ is required to approximate, in the sense of $K(P', P)$, the target distribution P' which is formed by independently flipping each bit of x, with the probabilities

$$\log \frac{P_i'(-|-)}{P_i'(+|-)} = a, \qquad \log \frac{P_i'(+|+)}{P_i'(-|+)} = b.$$

In other words, given an input $x \in R^n$, the desired conditional distribution of output y is $P'(y|x) = \prod_i P_i'(y_i|x_i)$. The blurred identity mapping encoder problem with unit code input is called the *AHS encoder problem*.

Notation. For each $x \in R^n$, $k + j \le n$, denote

$$F_{kj}(x) := \{ y : \left| \{ i : y_i = 1 | x_i = -1 \} \right| = k, \left| \{ i : y_i = -1 | x_i = 1 \} \right| = j \}.$$

Lemma 7.3.6 *For the AHS encoder problem, $|F_{k0}| = |F_{k1}| = C_{n-1}^k$.*

Proof. The sets F_{kj} consist of codes y which is formed from x by flipping exactly k bits of $-$ to $+$ and j bits of $+$ to $-$. It have $C_{n-1}^k C_1^j$ members. Since $j \in \{0,1\}$, $C_1^j = 1$. □

Theorem 7.3.7 *For the AHS encoder problem, $P' = BD(e, \mu_0)$, where e is normalised according to Definition 7.3.1, and*

$$\mu_0 = \left(1 - \frac{1}{n}\right)\frac{a}{2} + \frac{1}{n}\frac{b}{2}.$$

Also, $\forall y \in F_{kj}(x):\ e(y|x) = e_{kj} = n - 2(kA + jB)$, where

$$A = \frac{na}{(n-1)a + b}, \qquad B = \frac{nb}{(n-1)a + b}.$$

Proof. Since P' is not degenerated, we can assume $P' = BD(e, \mu_0)$, where the free parameter μ_0 is determined by the normalization condition. By the independence of the blurring of the bits, it follows that

$$T'(y|x) = \sum_i T_i(y_i|x_i),$$

where T' is the tendency function for P' and

$$T_i(-|-) - T_i(+|-) = a, \qquad T_i(-|+) - T_i(+|-) = b.$$

Therefore all the $y \in F_{kj}(x)$ have identical $e(y|x)$ which can be denoted e_{kj}. Since P' is the Boltzmann distribution, we have $e_{kj}\mu_0 = c - ka - ib$, where μ_0 and c are to be determined. Let $A = \frac{a}{2}\theta_0$ and $B = \frac{b}{2}\theta_0$, where $\theta_0 = 1/\mu_0$. Then $e_{ki} = c\theta_0 - 2kA - 2iB$. From normalization convention

$$\sum_k C_{n-1}^k \left(e_{k0} + e_{k1}\right) = 0, \qquad \sum_k C_{n-1}^k \left(e_{k0}^2 + e_{k1}^2\right) = n2^n,$$

we get $c = (a/2)(n-1) + b/2$ and $(\theta_0/2)((n-1)a + b) = n$. So

$$A = \frac{na}{(n-1)a + b}, \qquad B = \frac{nb}{(n-1)a + b}.$$

□

Theorem 7.3.8 *Let $\mu \in \mathbb{R}_+$, $\theta = 1/\mu$. Define*

$$p := \exp(-2A\mu), \qquad q := \exp(-2B\mu). \tag{7.3.5}$$

Then, for the AHS encoder problem in ideal learning,

$$P_{kj} = \frac{p^k q^j}{(1+p)^{n-1}(1+q)}, \tag{7.3.6}$$

$$\langle e \rangle = (n-1)A\tanh(A\mu) + B\tanh(B\mu), \qquad \langle T \rangle = \langle e \rangle \mu. \tag{7.3.7}$$

$$\sigma^2 = (n-1)A^2\operatorname{sech}^2(A\mu) + B^2\operatorname{sech}^2(B\mu), \qquad C = \sigma^2\mu^2. \tag{7.3.8}$$

Proof. Since P_{kj} is proportional to $p^k q^j$, (7.3.6) follows straightforward calculation, applying (A.5.1). Let

$$u := \frac{2p}{1+p}A, \qquad v := \frac{2q}{1+q}B.$$

It follows by further calculation that

$$\langle e \rangle = n - (n-1)u - v,$$
$$\sigma^2 = (n-1)\frac{u^2}{p} + \frac{v^2}{q}.$$

These, together with Theorem 7.3.1, prove the theorem. □

Corollary 7.3.9 *If $a = b$ then $A = B = 1$, and Theorem 7.3.8 reduces Theorem 7.3.4, with $\mu_0 = a/2$ and $c = na/2$.*

The maximum of heat capacity of AHS encoder can not be analytically derived unless $a = b$, but it is quite easy to find out numerically. The corresponding parameters for the particular experiments on the encoder problems described in [AHS85] are summarised in Table 7.1.

problem	a	b	at θ_0: $1/\theta_0$	C/n	$\langle e \rangle /n$	ϵ	at θ_c: $1/\theta_c$	C_c/n
4-2-4	$\log\left(\frac{.95}{.05}\right)$	$\log\left(\frac{.85}{.15}\right)$	1.3210	0.4048	0.8672	1.20	1.1760	0.4121
8-3-8	$\log\left(\frac{.98}{.02}\right)$	$\log\left(\frac{.85}{.15}\right)$	1.8111	0.3077	0.9444	2.32	1.1586	0.4123
40-10-40	$\log\left(\frac{.9975}{.0025}\right)$	$\log\left(\frac{.9}{.1}\right)$	2.9471	0.0981	0.9932	7.90	1.1872	0.4323

Table 7.1: The parameters corresponding to the test results reported [AHS85] The parameters a and b are from pages 158, 160 and 162 of [AHS85] for the 4-2-4, 8-3-8, and 40-10-40 encoders, respectively. The column labeled ϵ is the total bit error of target (§7.5).

The TE, TS, TT and TC graphs for the Hamming distance encoder problem and the particular encoder problems as given in [AHS85], both shown in Figure 7.17 & 7.18, are almost indistinguishable.

7.4 Speed of Averaging, Learning and Annealing

The learning rules of SQFB-NN are derived under the assumption that all the processes can be performed arbitrarily slowly. In this section we discuss the speed of these processes by analysing the encoder problem. Many of the issues discussed are also relevant to simulated annealing not necessarily implemented on neural networks.

7.4.1 Running average schemes

In Chapter 6, the learning rules are derived based on mathematical expectations $\langle \cdot \rangle$, ie., ensemble averages, or random variables. In practice, only the random variables are directly available, and $\langle \cdot \rangle$ is only approximated by some running averaging process

$[\cdot]$. This is necessary for implementing any stochastic methods. In this subsection we temporarily deviate from the convention of using x, y to denote the input and output of the network used in the rest of the thesis.

Definition 7.4.1 (Running average) Let X be a linear space, and $T = \mathbb{R}_+$. Denote $T_t := x|_{\{\tau:\ \tau<t\}}$. Denote by $X_{T,\Omega}$ the space of stochastic processes valued in X. It is itself a linear space. A linear operator A on $X_{T,\Omega}$ is called a *running average* if it is *lower triangular*

$$\forall t \in T:\ x|_{T_t} = 0 \implies (Ax)|_{T_t} = 0,$$

and *asymptotically unbiased*

$$\forall a \in \mathbb{R}:\ x \to a \implies Ax \to a.$$

We use brackets to denote running average $[x] := Ax$. Our aim is to find running average processes which is reasonably accurate in some suitable sense, and which is reasonably simple to be used in each synapse of a neural network. Running averages as defined above are special types of linear signal filters [OS75, Pri81]. In particular, our aim can be expressed as "low-pass" filter. On the other hand, since the research on linear filters are predominately concerned with stationary signals, which obviously does not apply to the learning process, we have to reformulate what is needed here. The description here are by no means complete, even for first order filters. We give formulation for the continuous time model, which is slightly more convenient. Those for corresponding discrete time model have similar forms.

First order filter

The first order filter $y = [x]$ is defined by the differential equation

$$\partial_t y(t) = \rho(t)(x(t) - y(t)), \qquad y(0) = x(0),$$

which is equivalent to the following *first order running average*

$$y(t) = \int_0^t c(\tau)x(\tau)d\tau \Big/ \int_0^t c(\tau)d\tau,$$

where ρ, c and C are fixed functions related by $\rho(t) := c(t)/C(t)$, $c = K\exp(\int \rho dt)$, and $C(t) := \int_0^t c(\tau)d\tau$, where $K \in \mathbb{R}_+$ is an arbitrary constant which can in general be assumed to be unity. We call c the "weight" and ρ the "rate" of the running average.

Proposition 7.4.1 *If $x(t) = x$ for all $t \in (a, b)$, then*

$$y(b) - y(a) = \left(1 - \frac{C(a)}{C(b)}\right)(x - y(a)).$$

This formula remains valid if time is discrete, the integral is replaced by a sum, and the differential is replaced by a difference. This is very useful combined with the technique for simulating Gibbs samplers described in §7.2.

The running average defined by $c(t) = t^{s-1}$, ie. $\rho(t) = s/t$ is called *polynomial formula*, in which $\rho(t)$ depends on t. The only time invariant formula is the *exponential formula* $c(t) = e^{\rho t}$, which gives $\rho(t) = \rho_0$, where ρ_0 is a constant. However, it gives a nonzero weight to the single value $x(0)$, as if $x(t) = x(0)$, $\forall t < 0$.

Remark 7.4.1 The polynomial formula with $s = 1$ and the exponential formula have been used in learning methods in as early as [Sam59] and their merits were discussed in [Min61].

Second order filter

The second order filter $y = [x]$ is described by the differential equation

$$\partial_t u = \rho_1(x - u), \qquad \partial_t y = \rho_1(x - u) + \rho_2(x - y), \tag{7.4.1}$$

which, after eliminating u, is equivalent to

$$\partial_t^2 y + (\rho_1 + \rho_2)\partial_t y + \rho_1\rho_2 y = (\rho_1 + \rho_2)\partial_t x + \rho_1\rho_2 x. \tag{7.4.2}$$

The second order filter (7.4.1) requires an auxiliary variable u but gives better performance than the first order filter for a slowly varying process

$$\langle x\rangle = at + b \implies y - \langle x\rangle \to 0, (t \to \infty),$$

for any constants $a, b \in \mathbb{R}$.

Although this system has a better asymptotic properties than the first order running average, it is not as good for high frequency changes. We shall not conduct a systematic study on this issue. Instead, we use an analogy between these equations and that of the well-known *damped oscillator*

$$\partial_t^2 y + (\rho_1 + \rho_2)\partial_t y + \rho_1\rho_2 y = \rho_1\rho_2 x,$$

where y is the "displacement" of the oscillator and $\rho_1\rho_2 x$ is the "external force". Such an oscillator has a *characteristic frequency* $\omega = \sqrt{\rho_1\rho_2}$, and *gain factor* $Q = \sqrt{\rho_1\rho_2}/(\rho_1+\rho_2)$. The so-called *critical damping* (when the characteristic equation

$$\rho^2 + (\rho_1 + \rho_2)\rho + \rho_1\rho_2 = 0,$$

has two equal real roots) is generally regarded as the optimal choice between stability and accuracy, which is $Q = 1/2$, corresponding to $\rho_2 = \rho_1 = \rho$.

Choice of the averaging schemes

Although higher order filters may also be used in computation, they are much less biologically plausible. When ANN are viewed as models of BNN, these running averages may be realized by the density of certain chemical substances, or as electrical potentials. Higher order running averages are also much difficult to achieve technically, since we need one at each connection.

In our simulations, the following running average scheme are used. (Primed symbols denote auxiliary variables.)

1. The first order filter with $c(t) = \sqrt{t}$ and $\rho(t) = 3/2t$ is used to compute $[r_k]_{k-1}$ and $[r_k r_k^T]_{k-1}$, when r_{k-1} is fixed while r_k is equilibrating. That is, set

$$\Delta\,[r_k]_{k-1} := \frac{3}{2t}(r_k - [r_k]_{k-1}), \tag{7.4.3}$$

$$\Delta\,[r_k r_k^T]_{k-1} := \frac{3}{2t}(r_k r_k^T - [r_k r_k^T]_{k-1}), \tag{7.4.4}$$

   at the end of equilibration of layer k, where $t = 0$ when r_{k-1} is set and r_k starts equilibration.

2. The exponential formula with decay 0.00001 is used to compute $[\Delta t]$, the average waiting time, that is, set

$$\Delta[\Delta t] := 0.00001(\Delta t - [\Delta t]), \tag{7.4.5}$$

at the end of each phase throughout the learning process.

3. The averages $[e]$, $[T]$ and $\sigma^2 \approx [e, e]$ are obtained with the second order filter with $\rho_1 = \rho_2 = \rho$, where different ρ are used in different tests. That is, set

$$\Delta[e]' := \rho(e - [e]), \qquad \Delta[e] := \rho(2e - [e] - [e]'), \tag{7.4.6}$$
$$\Delta[T]' := \rho(T - [T]), \qquad \Delta[T] := \rho(2T - [T] - [T]'), \tag{7.4.7}$$

and

$$\begin{aligned} \Delta\,[e,e]' &:= \rho((e - [e])^2 - [e,e]), \\ \Delta\,[e,e] &:= \exp(\rho(2\log[e,e] - \log[e,e] - \log[e,e]')). \end{aligned} \tag{7.4.8}$$

at the end of each presentation, throughout the learning process. This is a multiplicative variant of the second order filter, since it is $\log[e,e]$ which needs to be accurate. In particular, $[e,e]$ should never be made negative.

Following are justifications of these choices:

Item 1 is based on the following consideration. What is desired here are ensemble averages, which can only be obtained in equilibrium. However, the running average is only computed in the *process of approaching equilibrium*, and is biased toward the initial distribution. Therefore we need a running average which puts more weight on later values. On the other hand, we need to average among a quite large sample to smooth out noise. Furthermore, since this is to be implemented at each synapse, higher order filters are computationally too expensive. The given formula offers an acceptable balance between these requirements.

Item 2 is based on the following considerations. The quantity $[\Delta t]$ is used to set the stopping time. Since we do not want them to be correlated to what have actually been computed, we need a very smooth average. (This can be afforded since it is carried out in the whole learning process.) It is also desirable for the average to be based on recent values. The exponential formula seems to satisfy these requirements. Incidentally, there is no penalty for given $[\Delta t_0]$ a nonzero weight.

Item 3 is based on the following considerations. These quantities are used to compute various speeds, as will be described below. They need to be fast and stable. They are computed for the whole network, so slightly more complicated scheme is affordable.

We have tried, based on intuition combined with random choice, many different combinations of running average schemes, until we found that the above gave as stable and fast results as acceptable. The experiments reported in following sections are based on these choices. Those other schemes we have tried include the following: (1) Exponential formula for $[rr]$ and $[r]$. They either gave too much bias toward initial value of $[rr]$ and $[r]$ (when $\rho \sim 1$), or too much variance (when $\rho \ll 1$). We also tried first order filter with $c(t) = 1$. It biased too much toward the initial values of rr and r. (2) Larger rate (such as 0.001) for computing $[\Delta t]$ are not as stable, while smaller rate (such as 10^{-7}) does not keep pace with the actual change of $\langle \Delta t \rangle$. We recognise that this rate may need to depend on the cooling speed, but we do not pursue this further since the current choice is quite satisfactory. (3) For $[e]$, $[T]$ and $[e, e]$, first order filters do not give as good results (either too slow or not stable enough). Also the additive formula for $[e, e]$ wrecks the computation since it is not guaranteed to be larger than 0. No attempts have been made at further optimising over these schemes, but it is certainly a rich subject of future research.

7.4.2 Time scale

From the analysis on ideal learning we can draw some conclusions on the time scale. It is readily seen that as $\theta \to 0$, $p \to 0$ with faster speed, so $P_k/p^k \to 1$. After the critical temperature, only P_0 and P_1 will have appreciable influence on $\langle e \rangle$, while $P_1/P_0 = p \to 0$. This means that the output will almost always be the correct code (the unique member of F^0), with those of F^1 occurring with frequency P_1. So further optimisation is mainly concerned with reducing P_1. This requires that the running average processes must be at least as slow as P_1, for otherwise they will not approximate the ensemble average. This is especially so as $\theta \to 0$, which results in $p/(\sigma^2/4n) \to 1$. The scaling factor

$$\phi := \sigma^2/n \in O(p) \tag{7.4.9}$$

is a good choice, since $\phi = 1$ for $\theta = \infty$.

Four typical runs of 4-4 Hamming distance encoders using uniform time scale are shown in Figure 7.1 & 7.2. The parameters α, β and γ will be explained in the following subsections and can be ignored for the moment. The sharp spikes in the TE, TS, and TC graphs show that the averaging frequency is too high. Comparing these graphs with the TT graph we see that these spikes and the jumps in the TT graph occur at corresponding position, when an "error" is made by the network. Therefore in these cases learning only occurs when an error is made. Also note that uniformly reducing the averaging frequency ρ_0 is not sufficient to eliminate the spikes, only to postpone them.

Three typical runs of 4-4 encoders with scaled speeds are given in Figure 7.3 & 7.4. Note that the parameters listed in the graph are not scaled. For example, in the graph $\rho = 0.1$ means that 0.1ϕ is actually used as ρ in the running average scheme.

Remark 7.4.2 The scale of the difference between (DATA4085) and (DATA4086), which use the same parameters, and that between (DATA4024) and (DATA4029), are typical among all the experiments. Also note that the performance are not very sensitive to the actual values of the parameters.

Remark 7.4.3 Although what we ultimately require is the performance of $\langle e \rangle$, it can be seen that the TS, TC and TT graphs can reveal much more additional information about the behaviour of the learning rule. None of these graphs would be available if simulated annealing were not used in the learning process. The same is true for IC graph. Although IE, IS and IT graphs are well defined even without SA, normally only the IE graph is given in the literature (eg., [AHS85, PA87], among many others).

7.4.3 Learning speeds

Generally speaking, learning speed refers to the speed of changing the weights. We first derive some auxiliary results of a general parameterised distribution.

Theorem 7.4.2 *Let $a, E \in \mathbb{R}^n$, $y \in (R^n)_\Omega$, $R = \{-1, 1\}$, and $e(y) := \sum_i E_i y_i$, $T(y) := \sum_i a_i y_i$. Then*

$$\langle e \rangle := \sum_i E_i \tanh a_i, \qquad \sigma^2 = \sum_i E_i^2 \operatorname{sech}^2 a_i,$$
$$\langle T \rangle := \sum_i a_i \tanh a_i, \qquad \log Z = \sum_i \log(2 \cosh a_i).$$

Furthermore, let $f(y) := e(y)\mu - \log P(y)$, then $\langle f \rangle = \langle e \rangle \mu - \langle T \rangle + \log Z$, and

$$\frac{\partial \langle f \rangle}{\partial a_i} = (E_i \mu - a_i) \operatorname{sech}^2 a_i.$$
$$\frac{\partial^2 \langle f \rangle}{\partial a_i \partial a_j} = -\delta_{ij} \left(2(E_i \mu - a_i) \tanh a_i + 1\right) \operatorname{sech}^2 a_i.$$

$$\frac{\partial \langle f \rangle}{\partial a_i} = 0 \implies \frac{\partial^2 \langle f \rangle}{\partial a_i^2} = -\operatorname{sech}^2 a_i = -\sigma_i^2 / E_i^2.$$

Proof. It follows

$$Z = \sum_y \prod_i \exp(a_i y_i) = \prod_i \sum_{y_i} \exp(a_i y_i) = \prod_i 2 \cosh(a_i),$$

that $P(y) = \prod_i P_i(y_i)$, where

$$P_i(y_i) := \exp(a_i y_i) / (2 \cosh(a_i)) = 1/(1 + \exp(-2 a_i y_i)).$$

So $\{y_i : i \in \mathbb{N}_n\}$ is statistically independent. It follows trivially that

$$\langle y_i \rangle = \tanh a_i, \qquad \sigma_{ij} := \langle y_i, y_j \rangle = \delta_{ij} \operatorname{sech}^2 a_i,$$
$$\frac{\partial \langle f \rangle}{\partial a_i} = \langle y_i, e(y)\mu - T(y) \rangle = (E_i \mu - a_i) \operatorname{sech}^2 a_i.$$

□

Now we consider one layer network with (normalised) Hamming distance evaluation function. In this case the ideal learning is possible.

Theorem 7.4.3 *Let $w \in \mathbb{R}^n$, $x, y \in (R^n)_\Omega$, $R = \{-1, 1\}$, $T(y|x) = \sum_i w_i x_i y_i$, $e(y|x) = \sum_i x_i y_i$, $f(y|x) := e(y|x)\mu - \log P(y|x)$. Then*

$$\frac{\partial \langle f \rangle}{\partial w_i} = (\mu - w_i) \operatorname{sech}^2 w_i. \tag{7.4.10}$$

When $w_i \approx \mu$,

$$\frac{\partial \langle f \rangle}{\partial w_i} \approx \phi(\mu - w_i). \tag{7.4.11}$$

Proof. Substituting $E_i = x_i$ and $a_i = w_i x_i$ into Theorem 7.4.2 and recalling that $y_i^2 = 1$, we get (7.4.10). Let $w_i = \mu$, then $\operatorname{sech}^2 w_i = \sigma^2 / n = \phi$. This proves (7.4.11) by continuity. □

Following [RHW86], we use a second order formula to update the synapses,

$$\Delta\delta w_i = \beta\left(\frac{\partial\langle f\rangle}{\partial w_i} - \delta w_i\right), \qquad \Delta w_i = \alpha\delta w_i, \tag{7.4.12}$$

where δw is an auxiliary variable. The continuous time counterpart is

$$\partial_t\delta w_i = \beta\left(\frac{\partial\langle f\rangle}{\partial w_i} - \delta w_i\right), \qquad \partial_t w_i = \alpha\delta w_i, \tag{7.4.13}$$

By eliminating δw (7.4.13) is transformed into the equation of a damped oscillator

$$\partial_t^2 w_i + \beta\partial_t w_i + \alpha\beta\phi w_i = \alpha\beta\phi\mu, \tag{7.4.14}$$

with frequency $\omega = \sqrt{\alpha\beta\phi}$ and gain factor $Q = \sqrt{\alpha\phi/\beta}$. The critically damped oscillator is $Q = 1/2$, or, equivalently, $\omega = 2\alpha\phi$ and $\beta = 4\alpha\phi$. As mentioned above, the characteristic frequency ω must be proportional to p. Let $\omega = \omega_0\phi$. Then these relations put together give

$$\alpha = \alpha_0 = \omega_0/2, \qquad \beta = \beta_0\phi = 2\omega_0\phi. \tag{7.4.15}$$

Intuitively (7.4.13) and (7.4.15) together is equivalent to

$$\delta w = \left[\frac{\partial\langle f\rangle}{\partial w}\right], \qquad \Delta w = \alpha\left[\frac{\partial\langle f\rangle}{\partial w}\right], \tag{7.4.16}$$

with one free parameter ω_0. In our simulations (§7.5) we also allow different relations between α and β, so there will be two free parameters α_0 and β_0.

Remark 7.4.4 If $f = e - \theta\log P$ is used in place of $f = e/\mu - \log P$ in the above, there will be a factor of θ difference between the gradients of free energy and the Kullback separation. The learning speed will be $\alpha = \omega_0\mu/2$ and $\beta = 2\omega$. However, this only concerns the scaling of δw, the speed $\partial_t w$ remain unchanged. Before we were aware of this, we have also tried to replace $e\mu - T$ with $e - \theta T$ or $(e\mu - T)\mu$ in the learning rule. Both gave worse results.

Theorem 7.4.4 *For critical damping (7.4.15), the differential equation (7.4.14) reduces to*

$$\partial_t^2 w_i + 2\omega\partial_t w_i + \omega^2 w_i = \omega^2\mu, \tag{7.4.17}$$

with solution $w_i = \mu + (C_{1i} + C_{2i}t)\exp(-\omega t)$, where C_{1i}, C_{2i} are arbitrary constants.

Notation. $\langle e\rangle_\mu := \left\langle\sum_y P_\mu(y|x)e(y|x)\right\rangle$, $\langle e\rangle_w := \left\langle\sum_y P_w(y|x)e(y|x)\right\rangle$, where $P_\mu \sim T_\mu = \mu\sum_i x_i y_i$, $P_w \sim T_w = \sum_i w_i x_i y_i$.

Theorem 7.4.5 *For critical damping (7.4.15), the differential equation (7.4.14) satisfies*

$$\frac{d\langle e\rangle_w}{dt}\frac{1}{\langle e\rangle_\mu - \langle e\rangle_w} \to \omega, \qquad (t\to\infty), \tag{7.4.18}$$

and $w \to \mu$.

Proof. It follows Theorem 7.4.4 that

$$\frac{\partial_t w_i}{\mu - w_i} \to \omega, \qquad (t \to \infty).$$

The asymptotic formula is proved by substituting this into

$$\frac{d\langle e\rangle_w}{dt} = \frac{\partial \langle e\rangle_w}{\partial w}\frac{dw}{dt}$$

and replacing the differential $\partial\langle e\rangle/\partial w$ with difference quotient $(\langle e\rangle_\mu - \langle e\rangle_w)/(\mu - w)$. □

This finally gives the formulae for the learning speed, by which $\langle e\rangle$ tends to an equilibrium value at rate $\exp(-\omega t)$. Compare this with [NS88] we see that $1/\epsilon = \omega$. Since many of the above relations are only asymptotically true for $p \to 0$, and since that learning must occur when p is reasonably large, the exact numerical coefficients might not be optimal. Simulation results for various combinations will be given in next section.

We call ω the *equilibration speed*, and ϵ the *equilibration time*. Following [NS88], we call $v := (\langle e\rangle_\mu - \langle e\rangle_w)/\sigma$ the *thermodynamic speed*.

7.4.4 Cooling schedule

There has been an intensive research on the cooling schedule of simulated annealing [KGV83, HCH91, SNH+88, MW87]. Although SA is usually described as reducing temperature, in the annealing process it is better to keep some other quantity as a constant, instead of the speed of temperature reduction [KGV83]. One might suggest such as constant heat dissipation (CHD), etc. The one we find most convincing is the *constant thermodynamic speed* (CTS) schedule [SNH+88]. For several technical reasons, it is better to describe cooling schedules in terms of $\mu = 1/\theta$, at least to avoid changing θ across zero in one step. We have experimented with the following three schedules,

(7.4.19) $$d\mu/dt = \gamma/\sigma^2 \qquad \text{(CHD schedule)},$$

(7.4.20) $$d\mu/dt = \gamma/\sigma \qquad \text{(CTS schedule)},$$

(7.4.21) $$d\mu/dt = \gamma.$$

It is to be noted that with our choice of the basic time scale proportional to ϕ, we must also set $\gamma = \gamma_0\phi$. We found that the CTS schedule is the best among them. The other two do not make as good balance of the cooling speed at the initial and final stage of learning.

Theorem 7.4.6 *Suppose that γ is small enough so that $\partial_t\langle e\rangle_w \approx \partial_t\langle e\rangle_\mu$, then for the CTS schedule $v \to \gamma/\omega$, for CHD schedule, $v \to \infty$, while for the schedule of (7.4.21), $v \to 0$.*

Proof. Applying Theorem 7.3.1 and Theorem 7.4.5 to the definition of v, we have

$$v = \frac{(\langle e\rangle_\mu - \langle e\rangle_w)}{\sigma} = \sigma(\langle e\rangle_\mu - \langle e\rangle_w)\sigma \Big/ \frac{d\langle e\rangle_\mu}{dt} = \frac{\sigma}{\omega}\frac{d\mu}{dt} = \begin{cases} \frac{\gamma}{\omega\sigma} & \text{(with (7.4.19))}, \\ \frac{\gamma}{\omega} & \text{(with (7.4.20))}, \\ \frac{\gamma\sigma}{\omega} & \text{(with (7.4.21))}. \end{cases}$$

□

So for CTS the thermodynamic speed is $v = \gamma/\omega = \gamma_0/\omega_0$, ie., the speed of annealing measured by the speed of learning.

Theorem 7.4.7 *The solution to $d\mu/dt = \gamma/\sigma$, $\gamma = \gamma_0\phi$, $\phi = \sigma^2/n$ and $\sigma^2 = n\,\mathrm{sech}^2\,\mu$ is*

$$\mu = \sinh^{-1}(\gamma_0 t/\sqrt{n} + c) \tag{7.4.22}$$

where c is an arbitrary constant.

Proof. The conditions imply that $d\mu/dt = (\gamma_0/\sqrt{n})\,\mathrm{sech}\,\mu$, whose solution is (7.4.22). □

Remark 7.4.5 It has been proved in [GG84] that the cooling schedule $\mu = \gamma\log(t+1)$, which is equivalent to $d\mu/dt = \gamma/(t+1)$ will guarantee global optimisation. The time t here is relative to the equilibrating speed of the Gibbs sampler, which does not have an exact correspondence here. If we interpret $\gamma = \gamma_0\phi$, ie. $\mu = \gamma_0\phi\log(t+1)$, then it is too slow to be put in practice. However, if we interpret the formula as $\gamma = \gamma_0$, so that $\mu = \gamma_0\log(t+1)$, it is asymptotically equivalent to the CTS schedule, since $\sinh^{-1} t \approx \log t$ as $t \to \infty$. Since there is no quantitative results on equilibration speed ω in [GG84], it is not known which of the two interpretation is correct. If the second interpretation is correct, then the CTS is both sufficient and necessary for good learning.

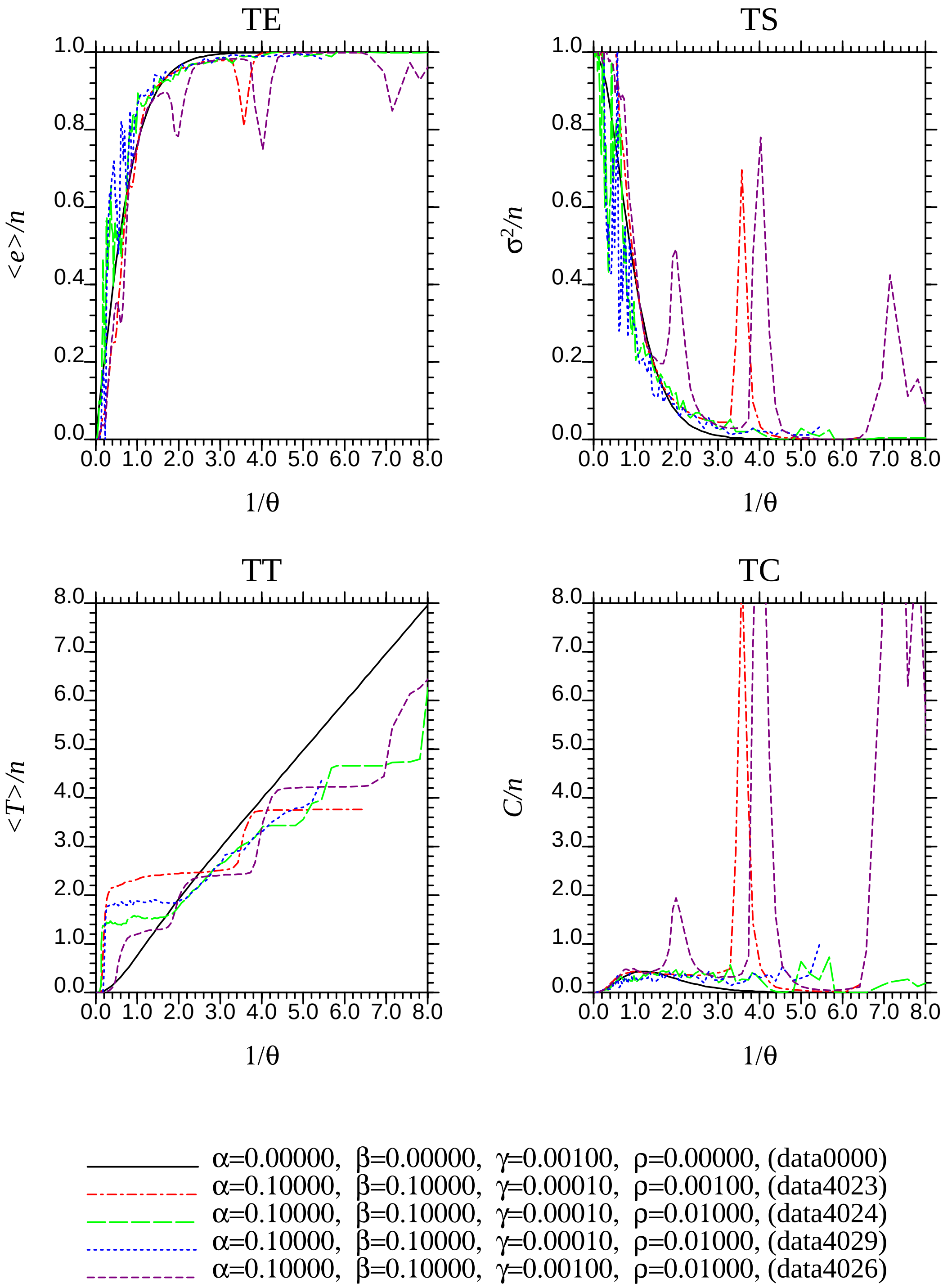

Figure 7.1: Four examples of 4-4 encoder with constant speed (a)

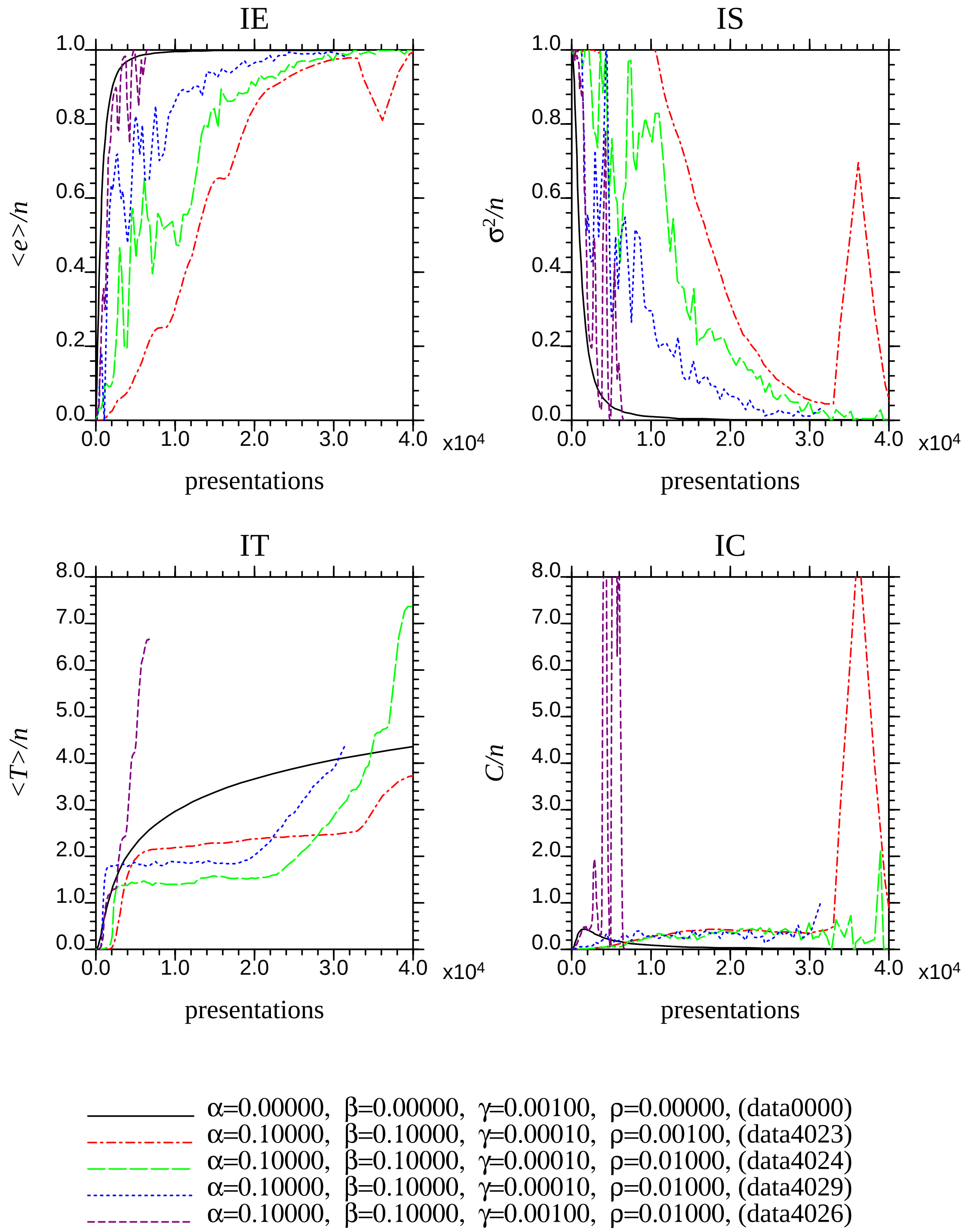

Figure 7.2: Four examples of 4-4 encoder with constant speed (b)

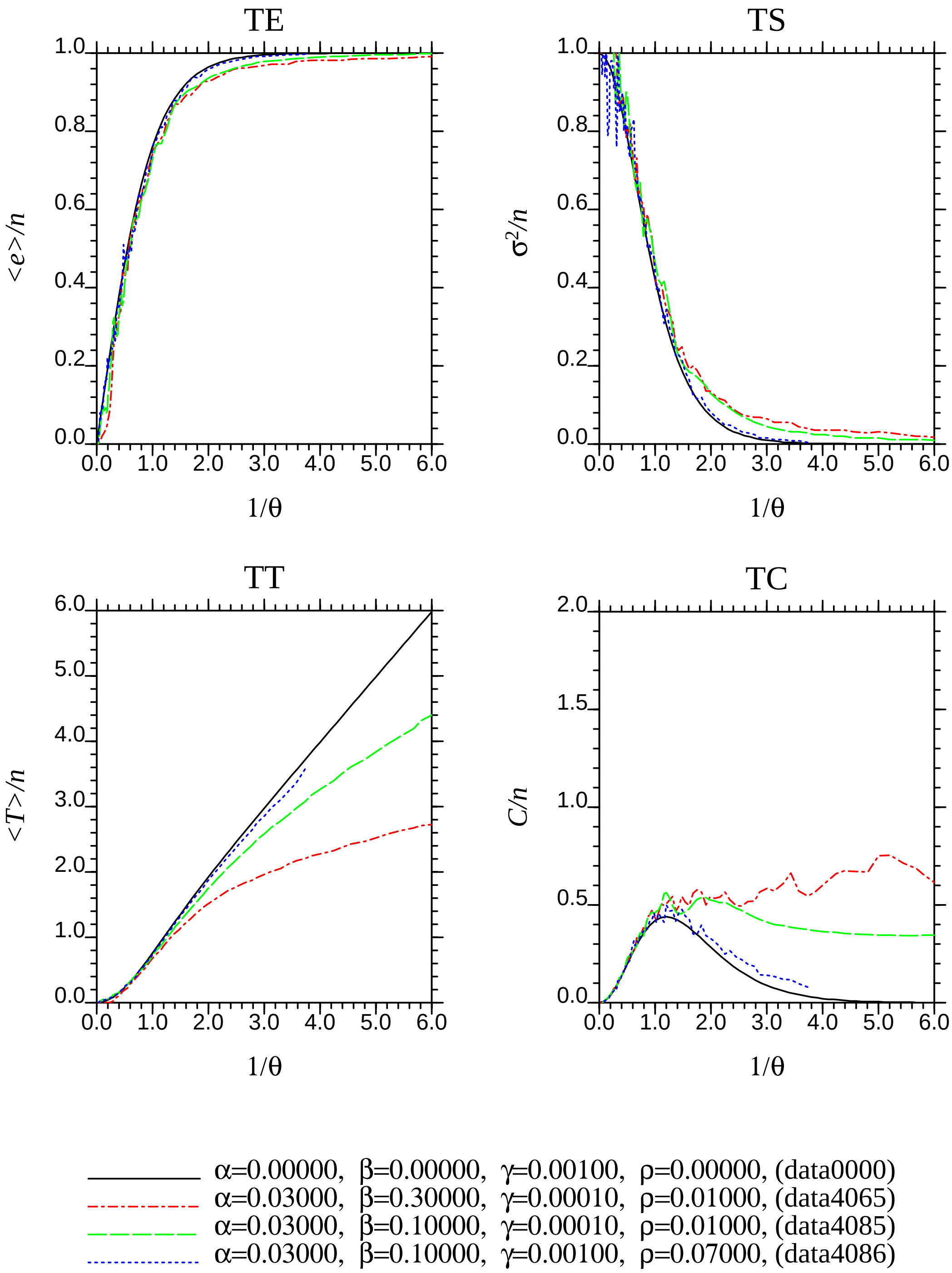

Figure 7.3: Three examples of 4-4 encoder with scaled speed (a)

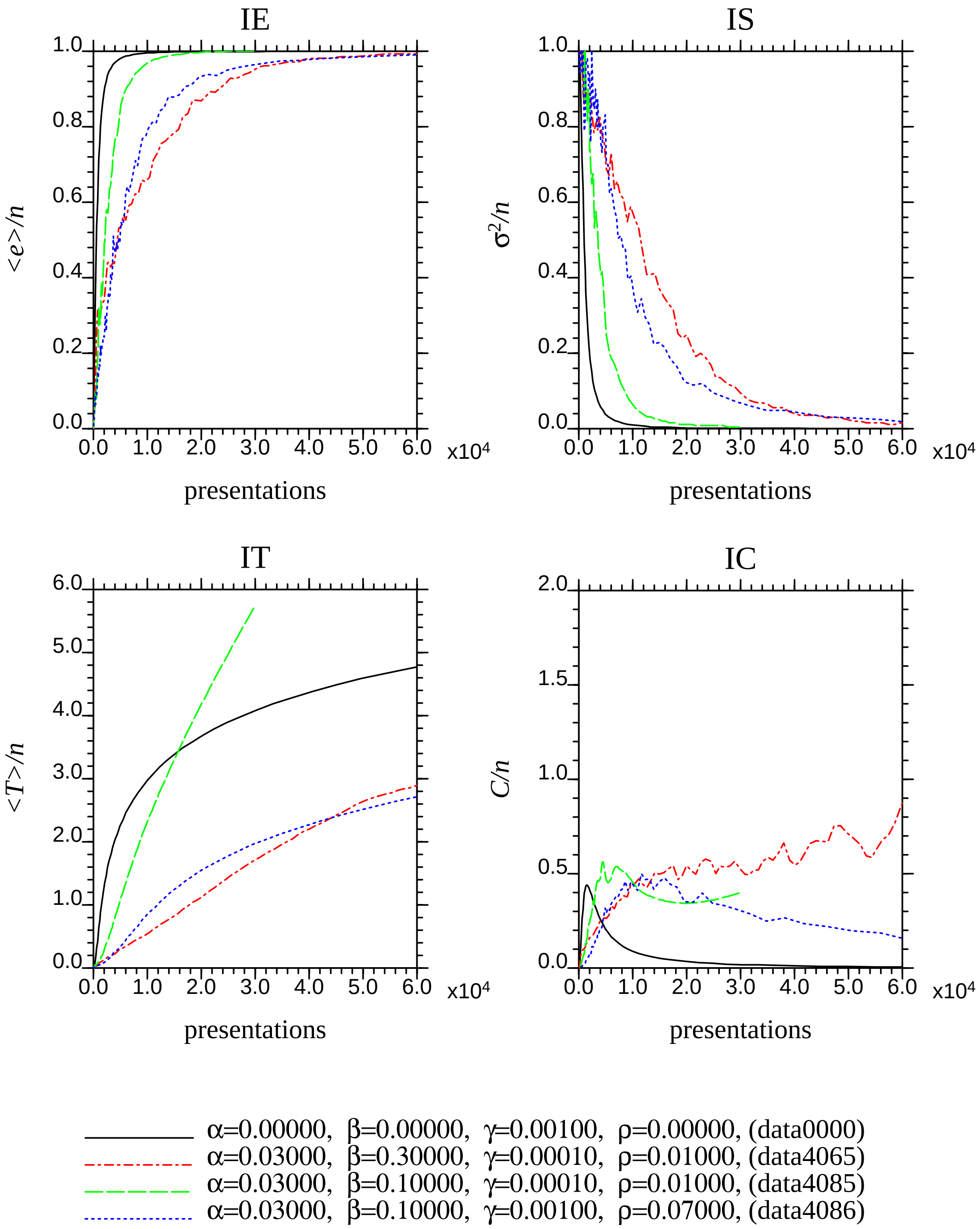

Figure 7.4: Three examples of 4-4 encoder with scaled speed (b)

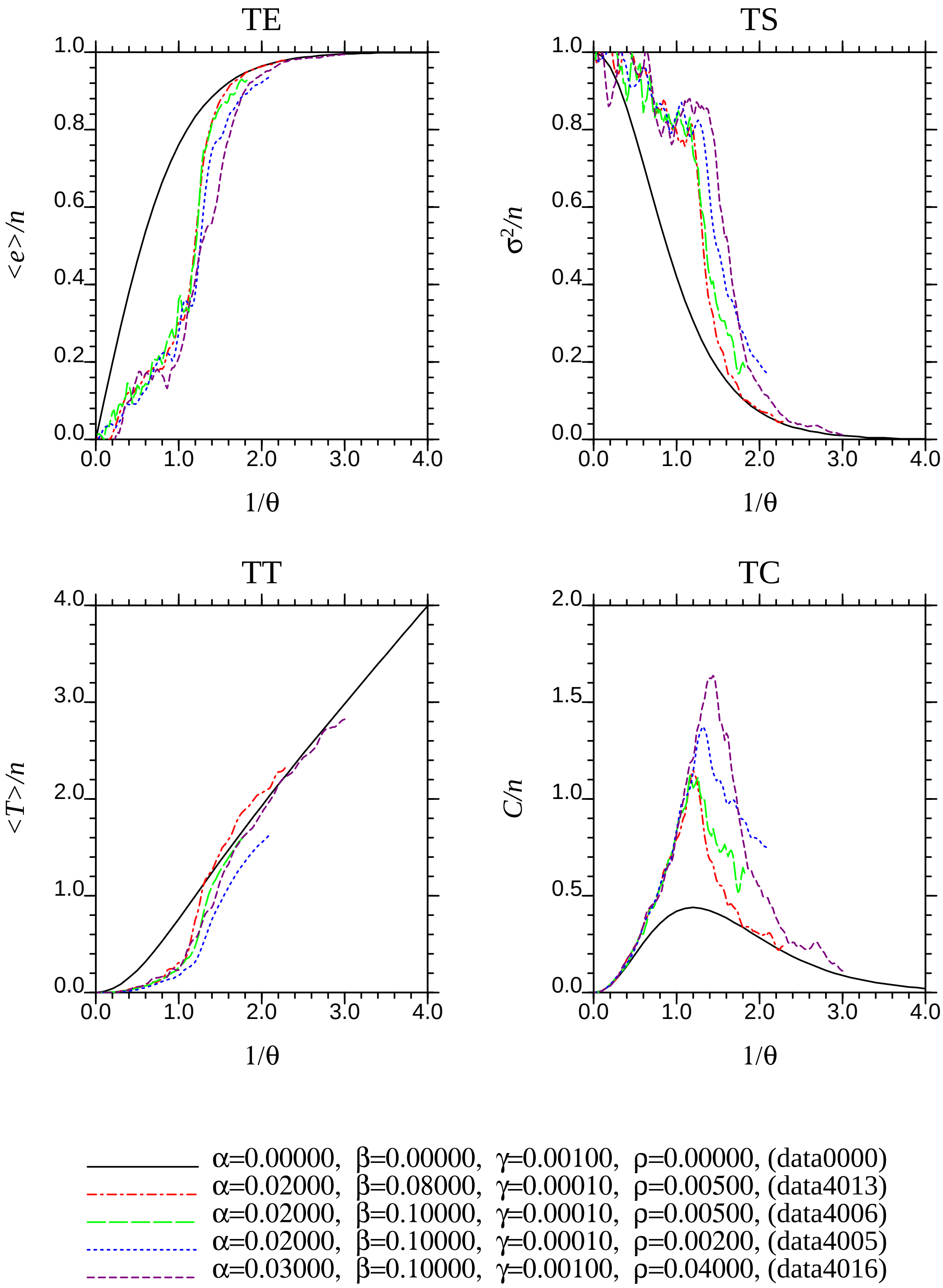

Figure 7.5: Four examples of 4-2-4 encoder (a)

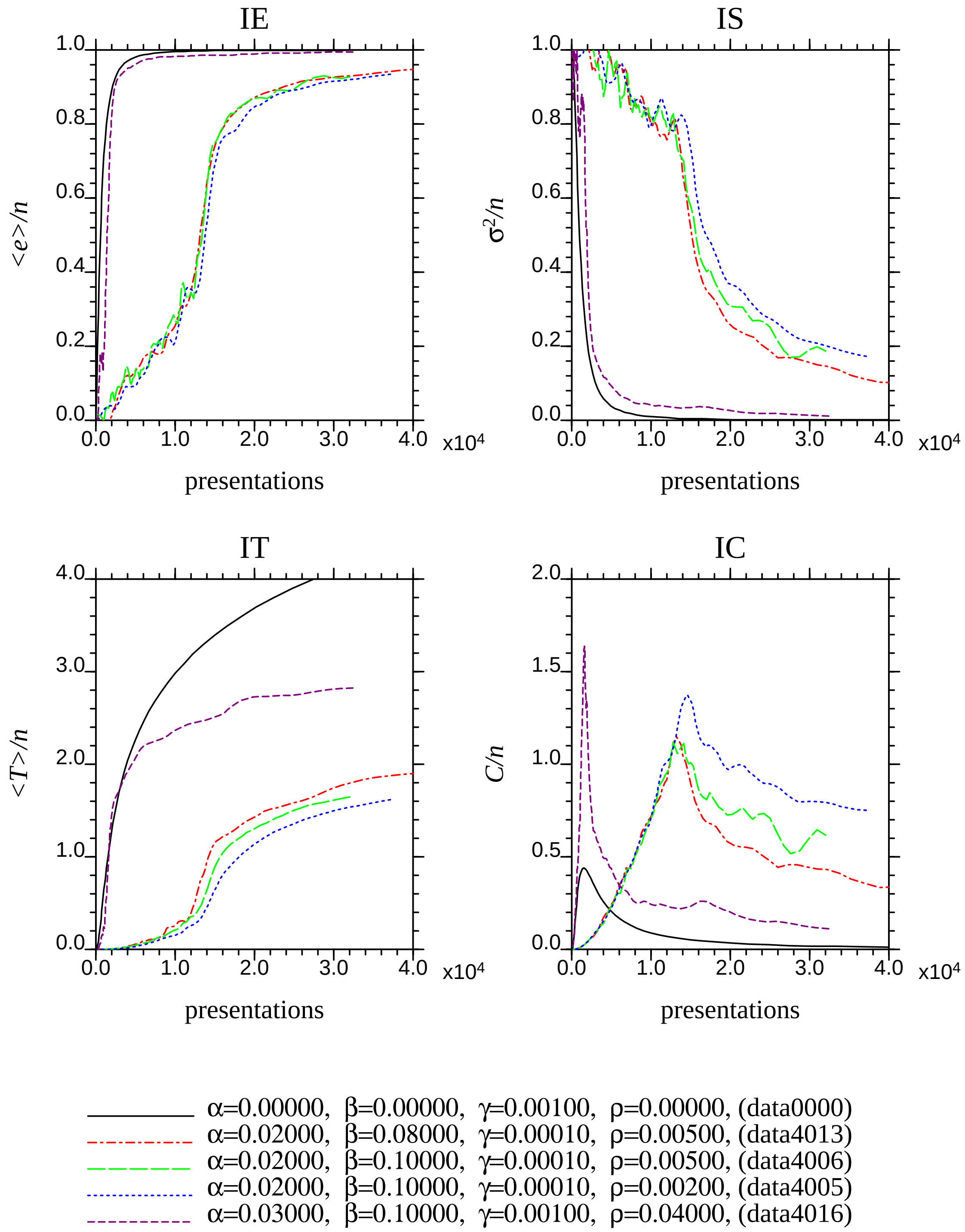

Figure 7.6: Four examples of 4-2-4 encoder (b)

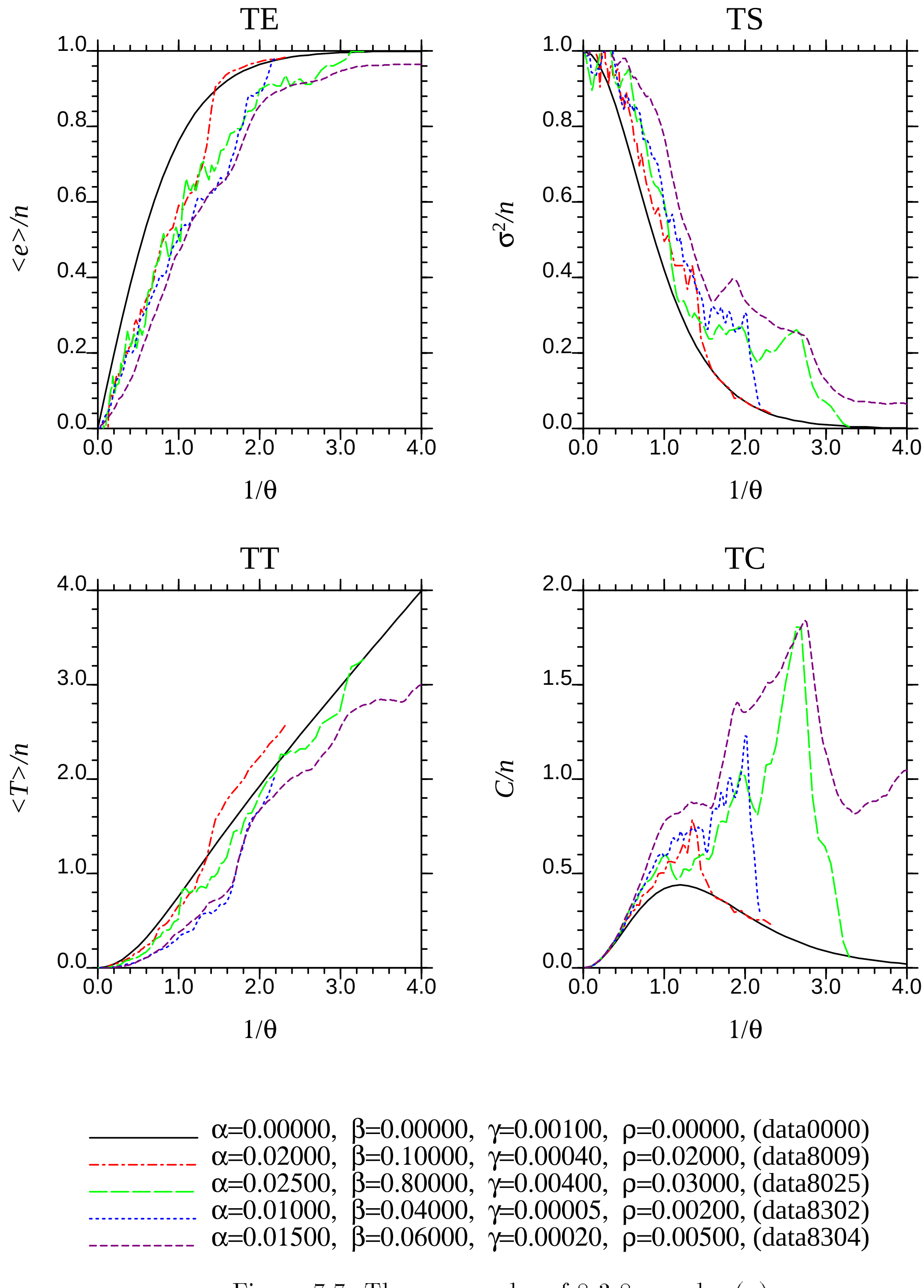

Figure 7.7: Three examples of 8-3-8 encoder (a)

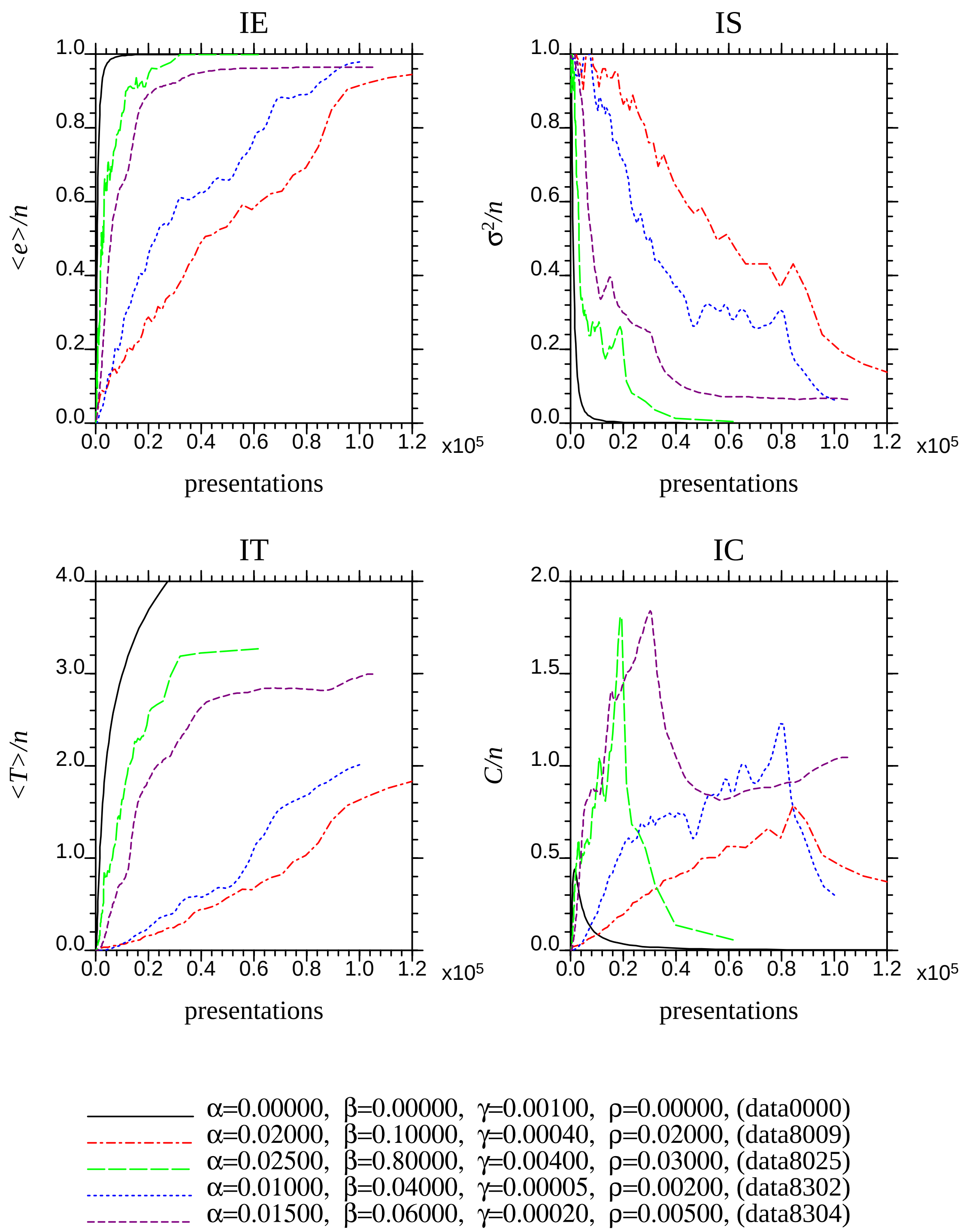

Figure 7.8: Three examples of 8-3-8 encoder (b)

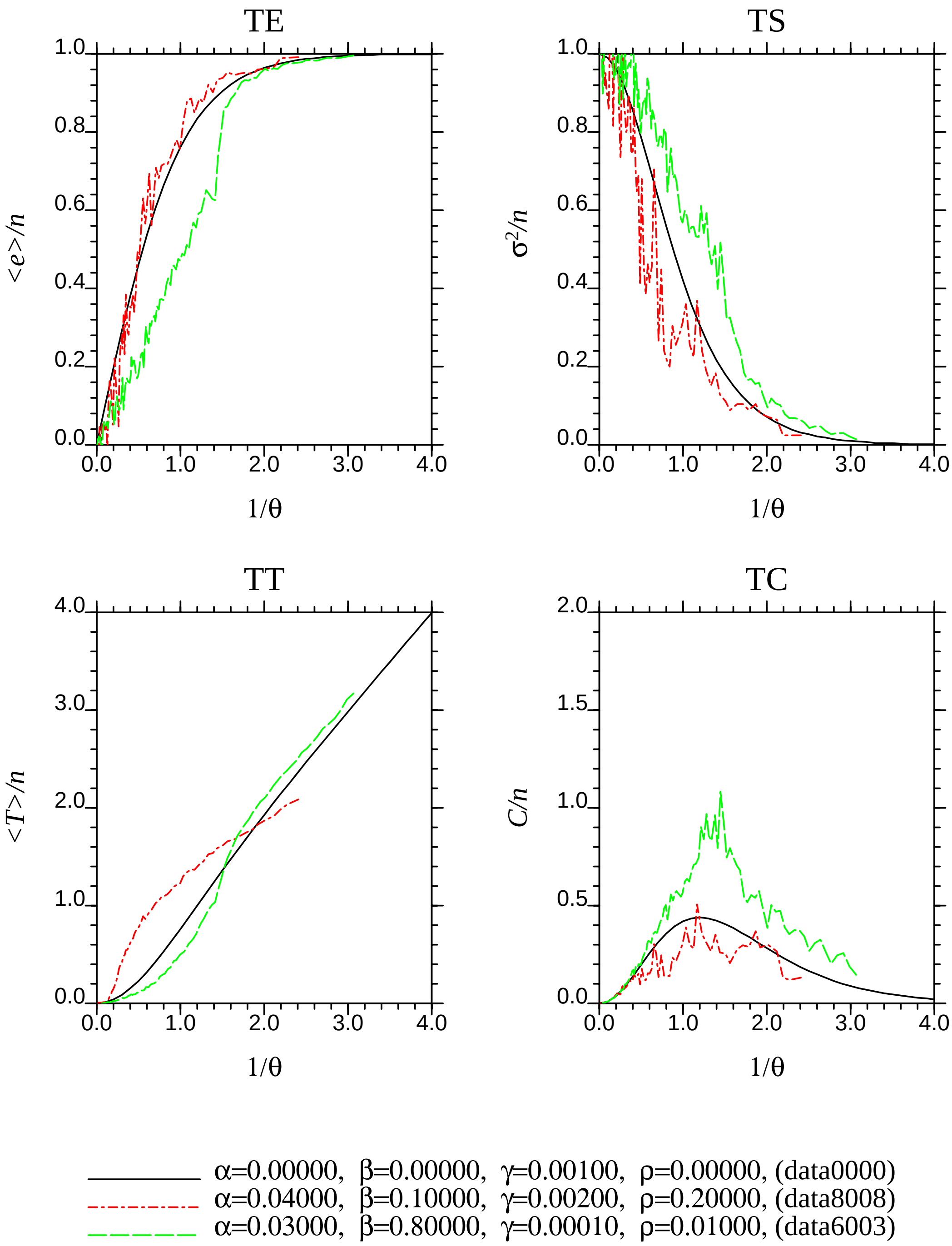

Figure 7.9: Two runs with redundant hidden units (a)

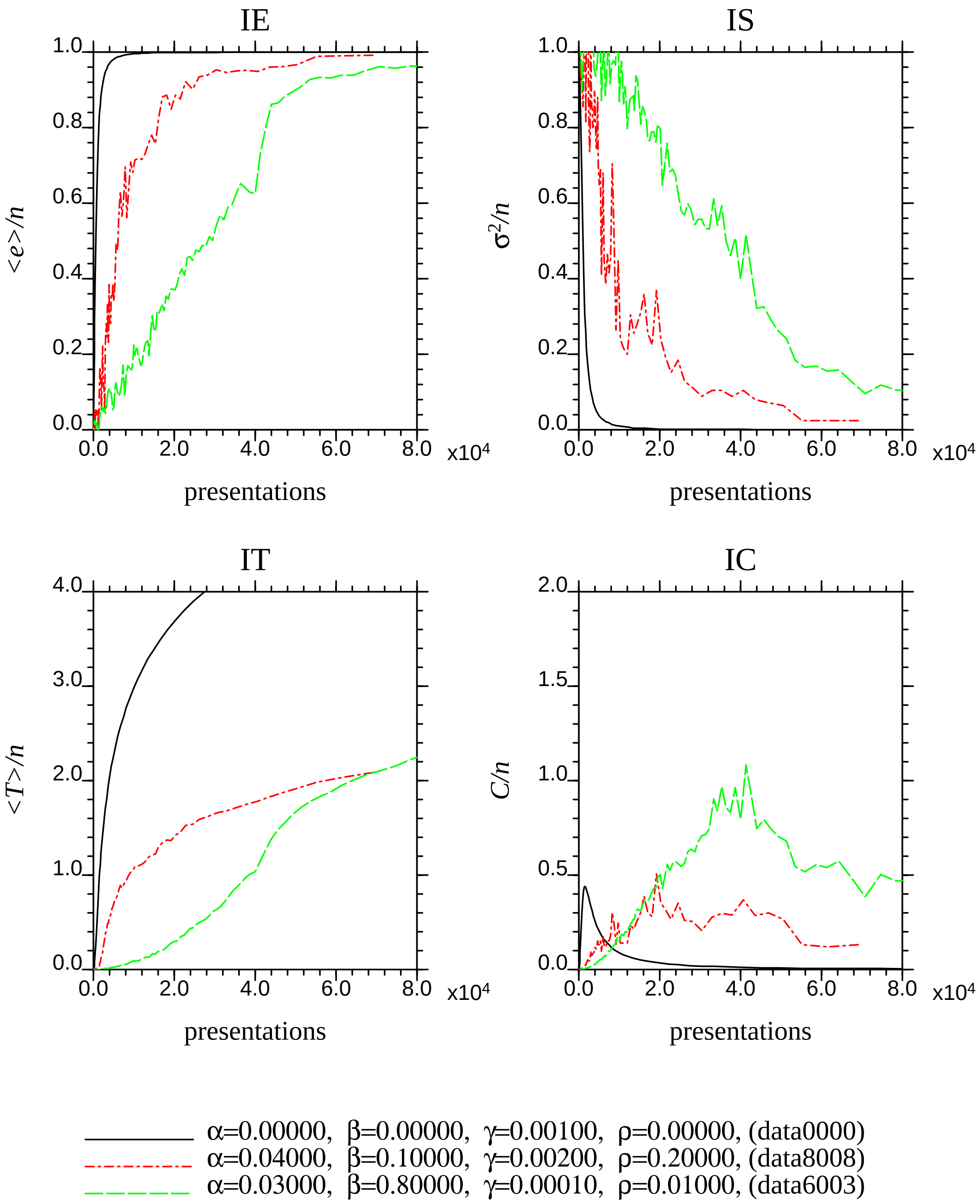

Figure 7.10: Two runs with redundant hidden units (b)

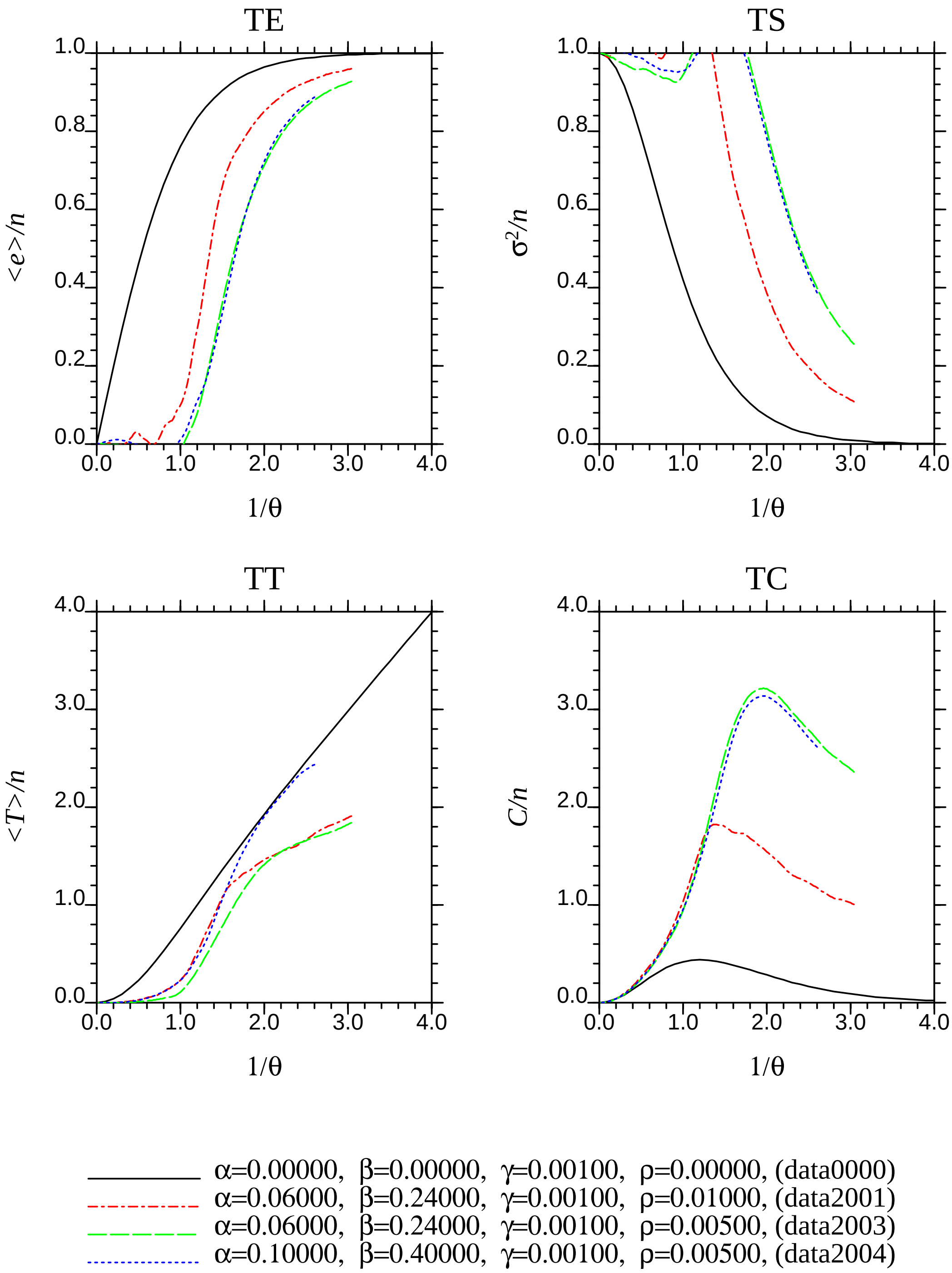

Figure 7.11: Three runs for the 2-2-2-2 encoder problem (a)

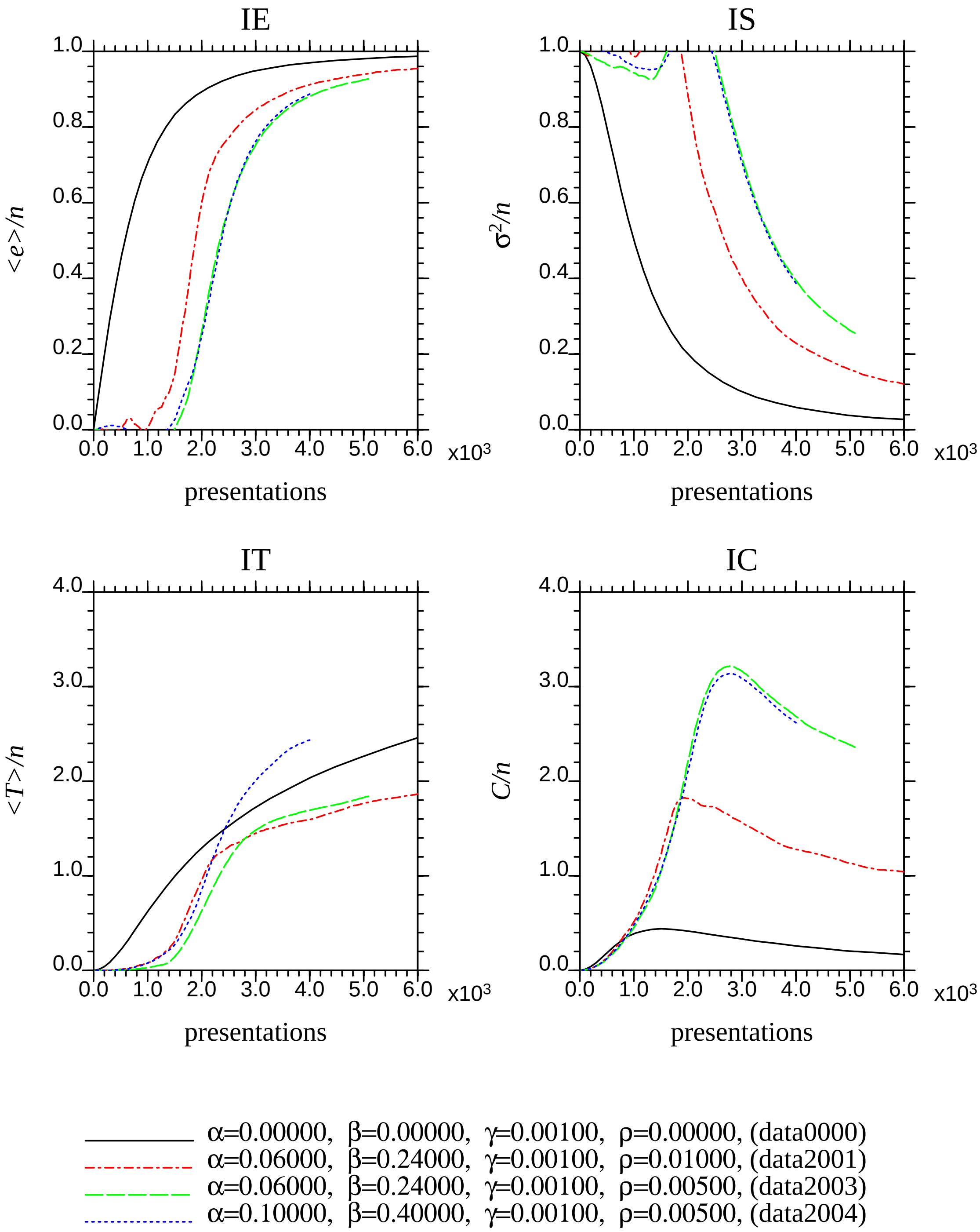

Figure 7.12: Three runs for the 2-2-2-2 encoder problem (b)

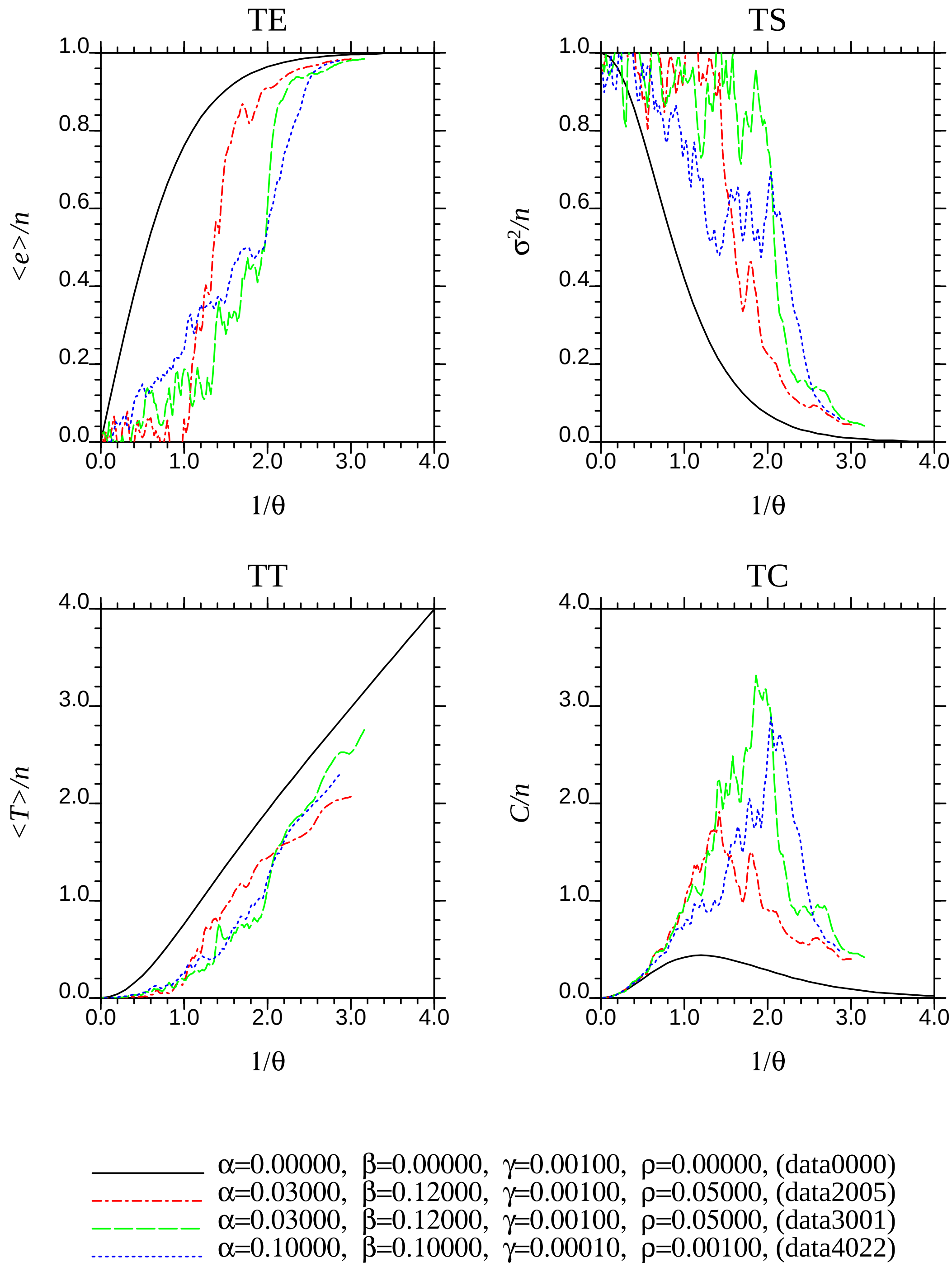

Figure 7.13: Three examples of the three layer encoder problem (a)

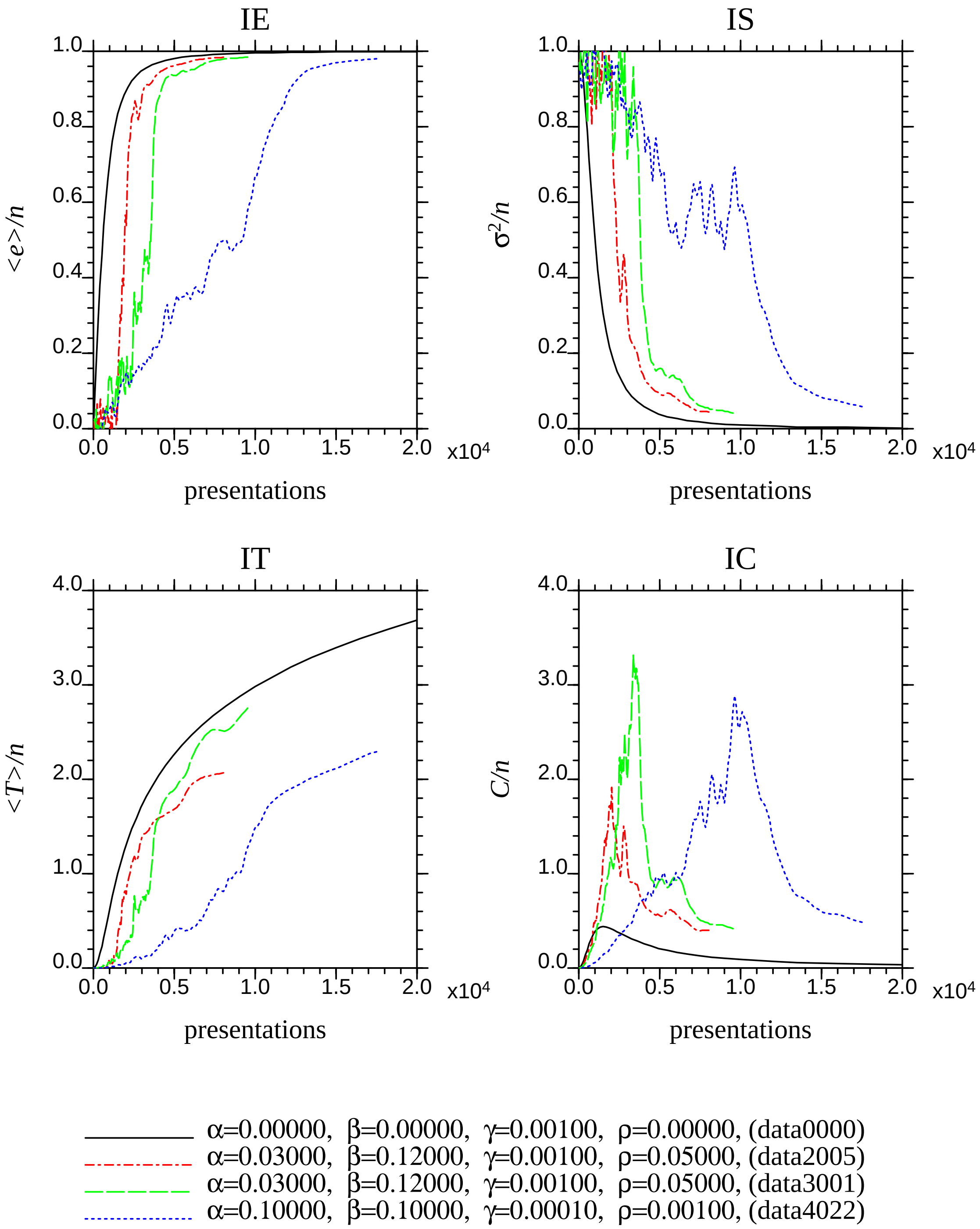

Figure 7.14: Three examples of the three layer encoder problem (b)

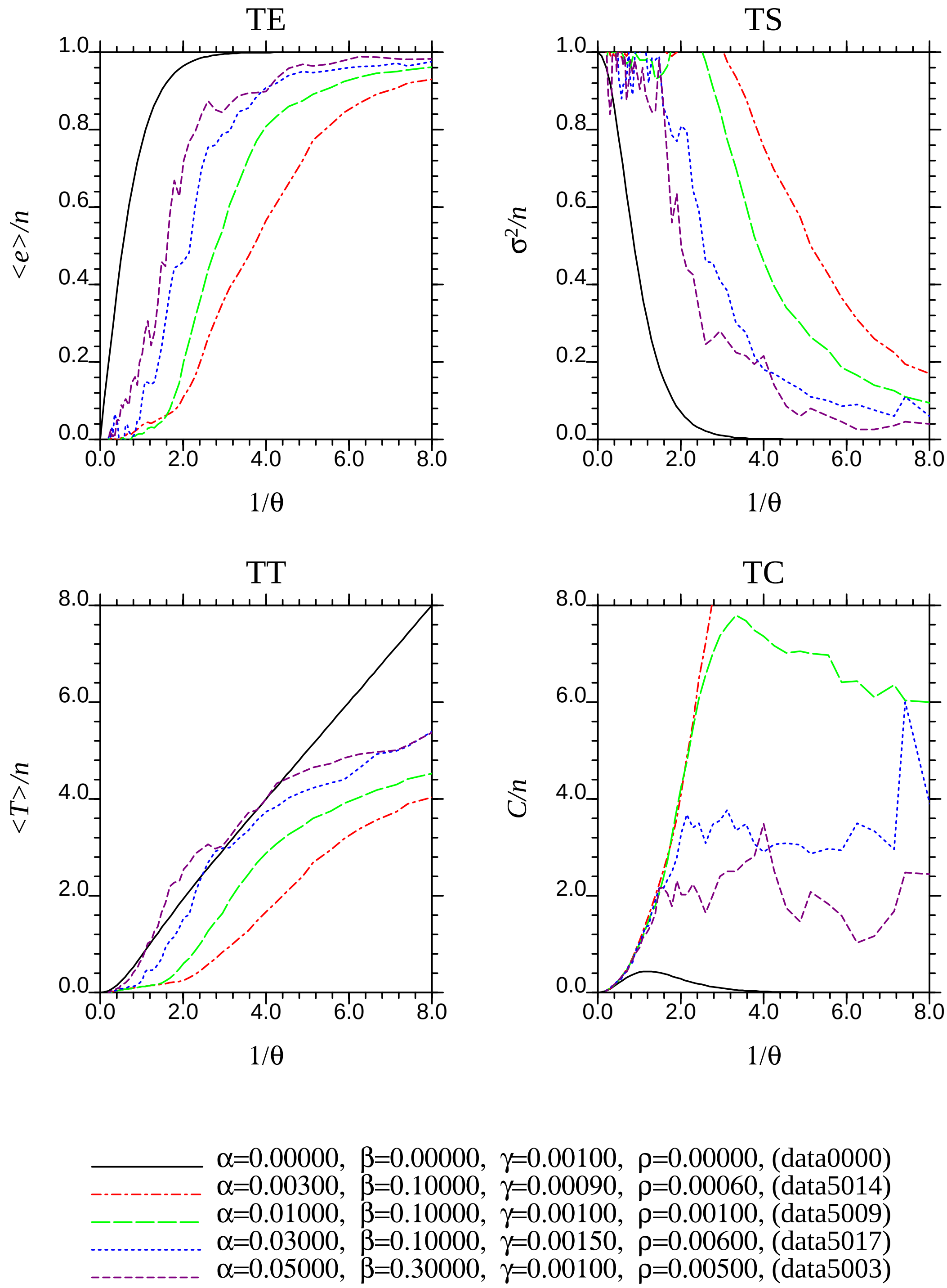

Figure 7.15: Four examples of the 5-3-5(8) random code problem (a)

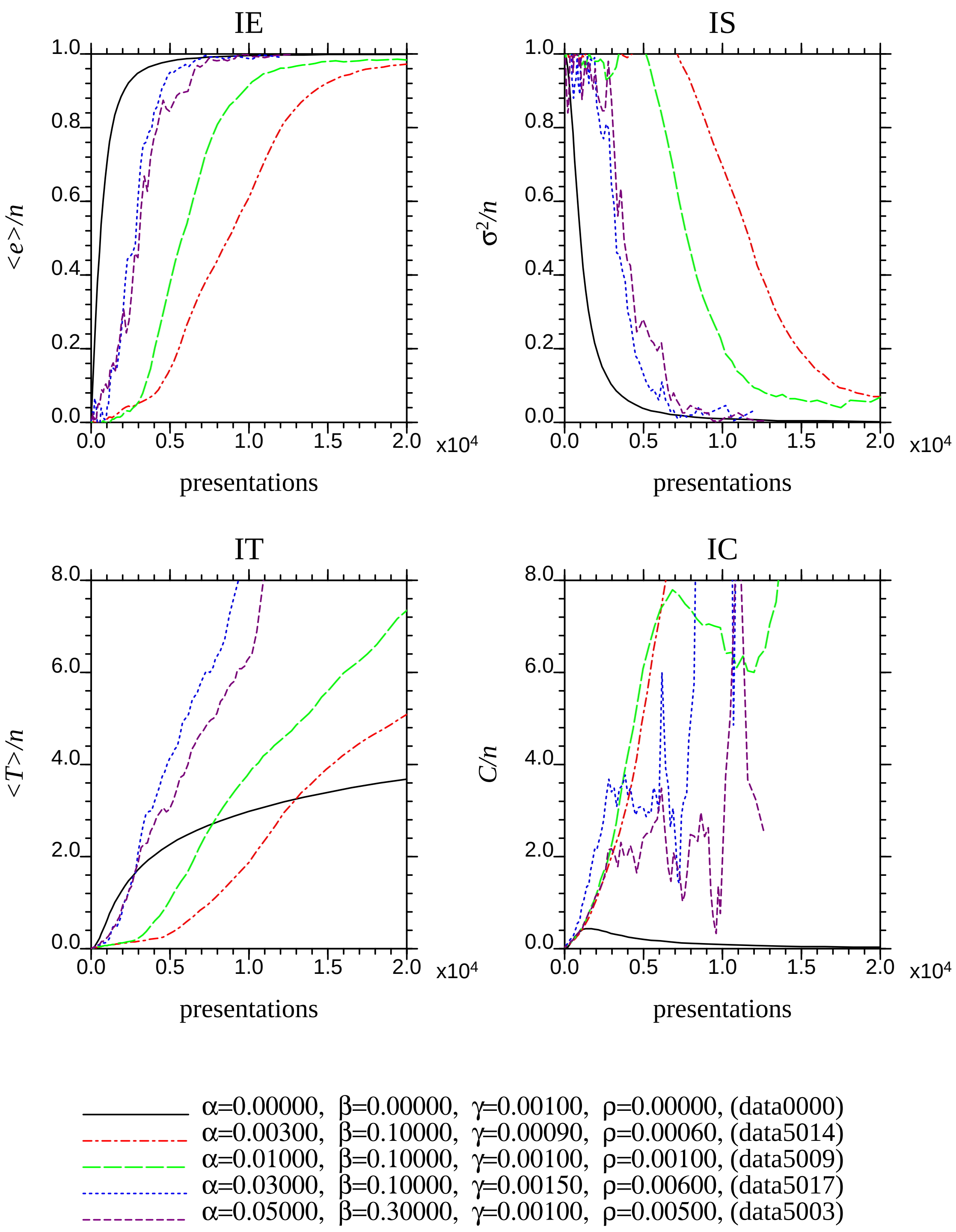

Figure 7.16: Four examples of the 5-3-5(8) random code problem (b)

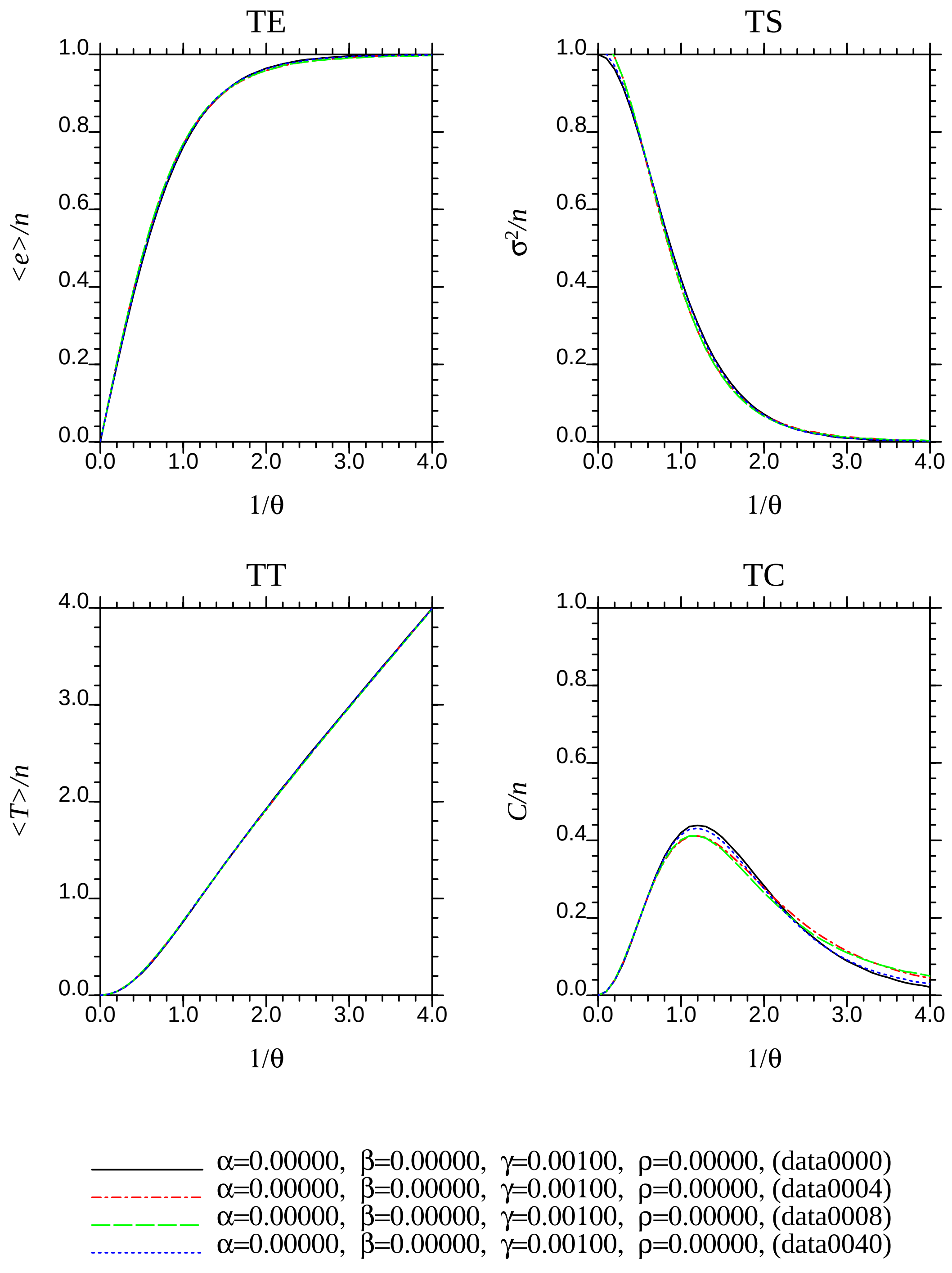

Figure 7.17: The ideal learning curve for the Hamming distance encoder and the encoders defined in [AHS85] (a)

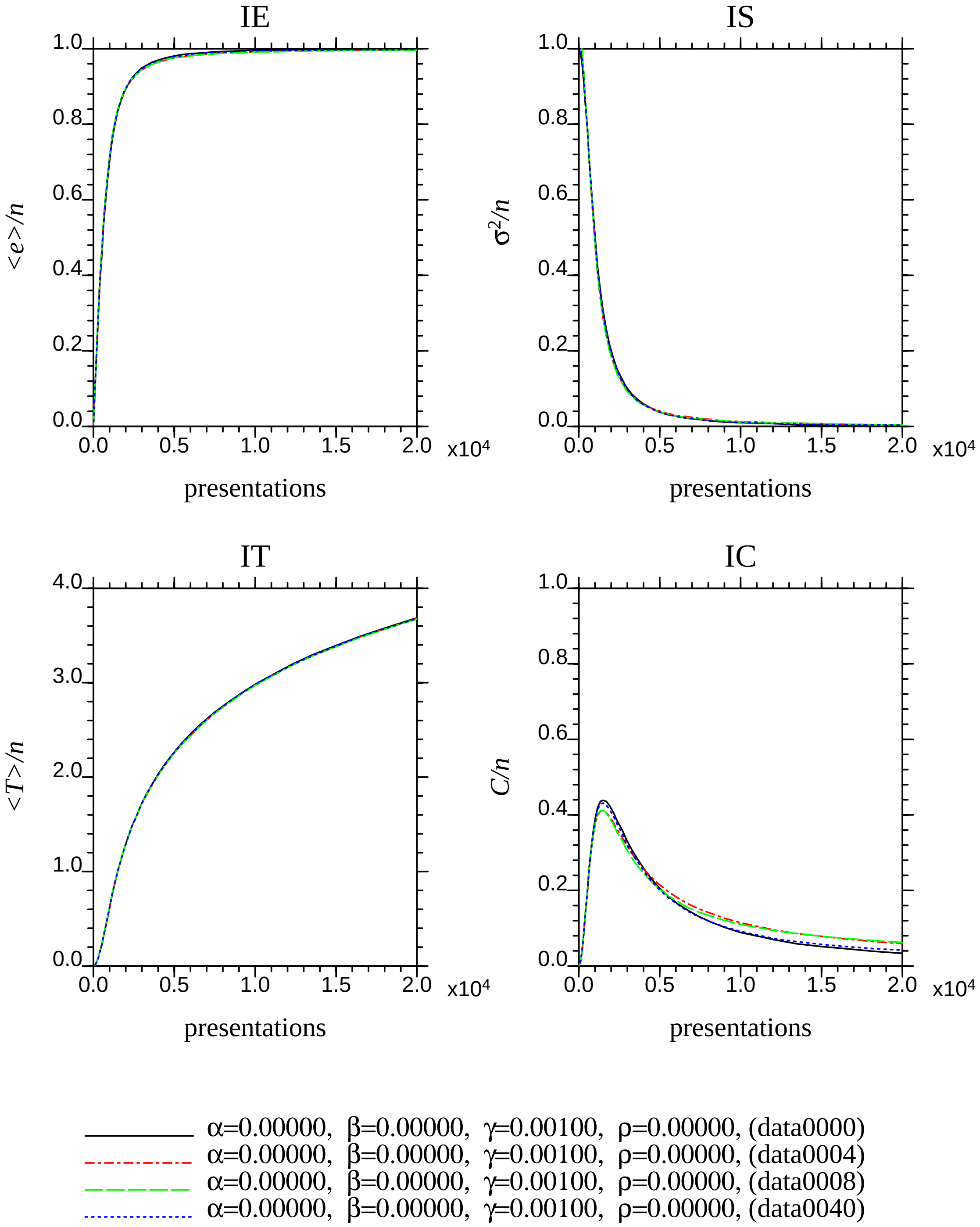

Figure 7.18: The ideal learning curve for the Hamming distance encoder and the encoders defined in [AHS85] (b)

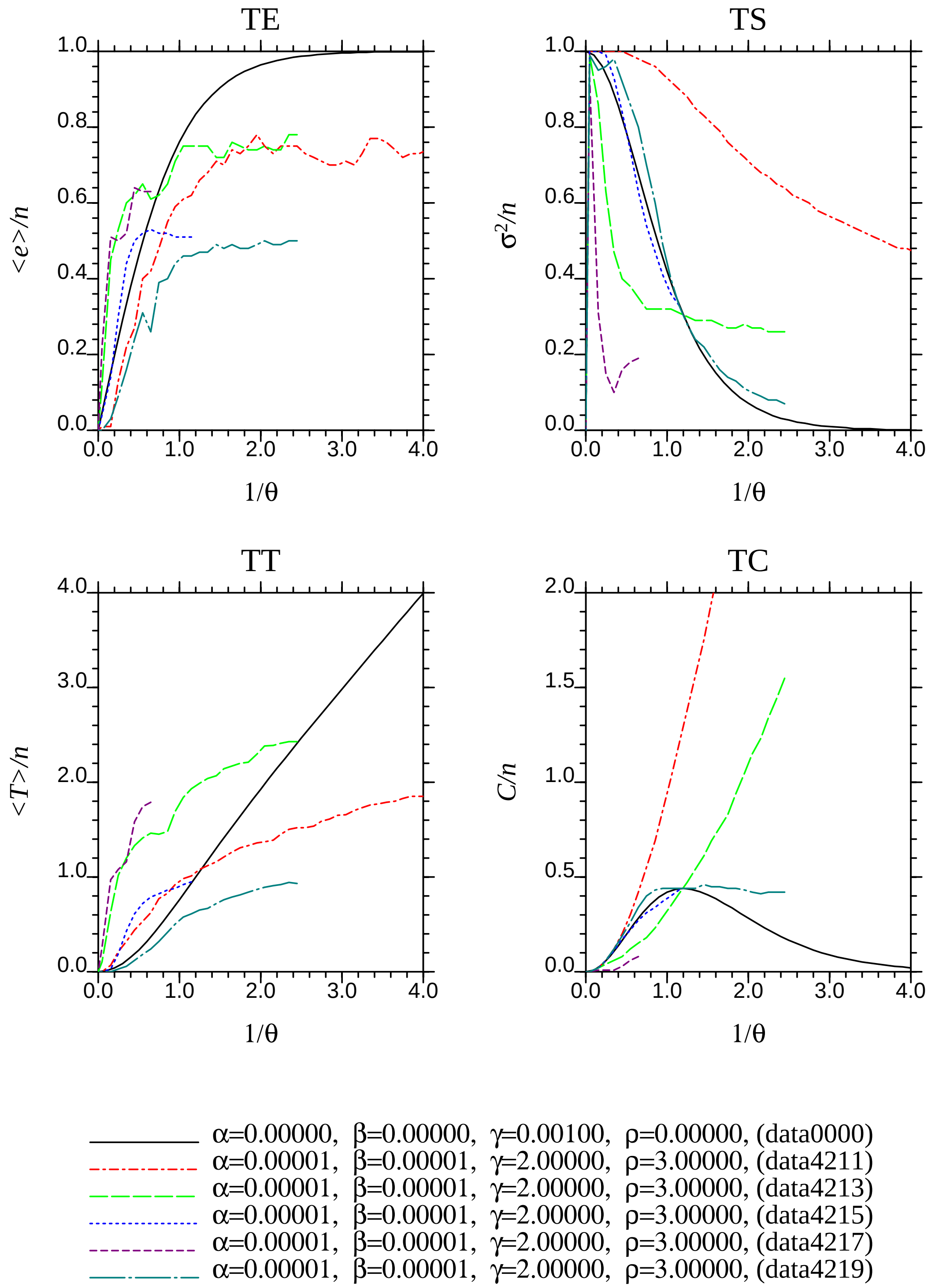

Figure 7.19: Four examples of the 4-2-4 encoder without simulated annealing (a)

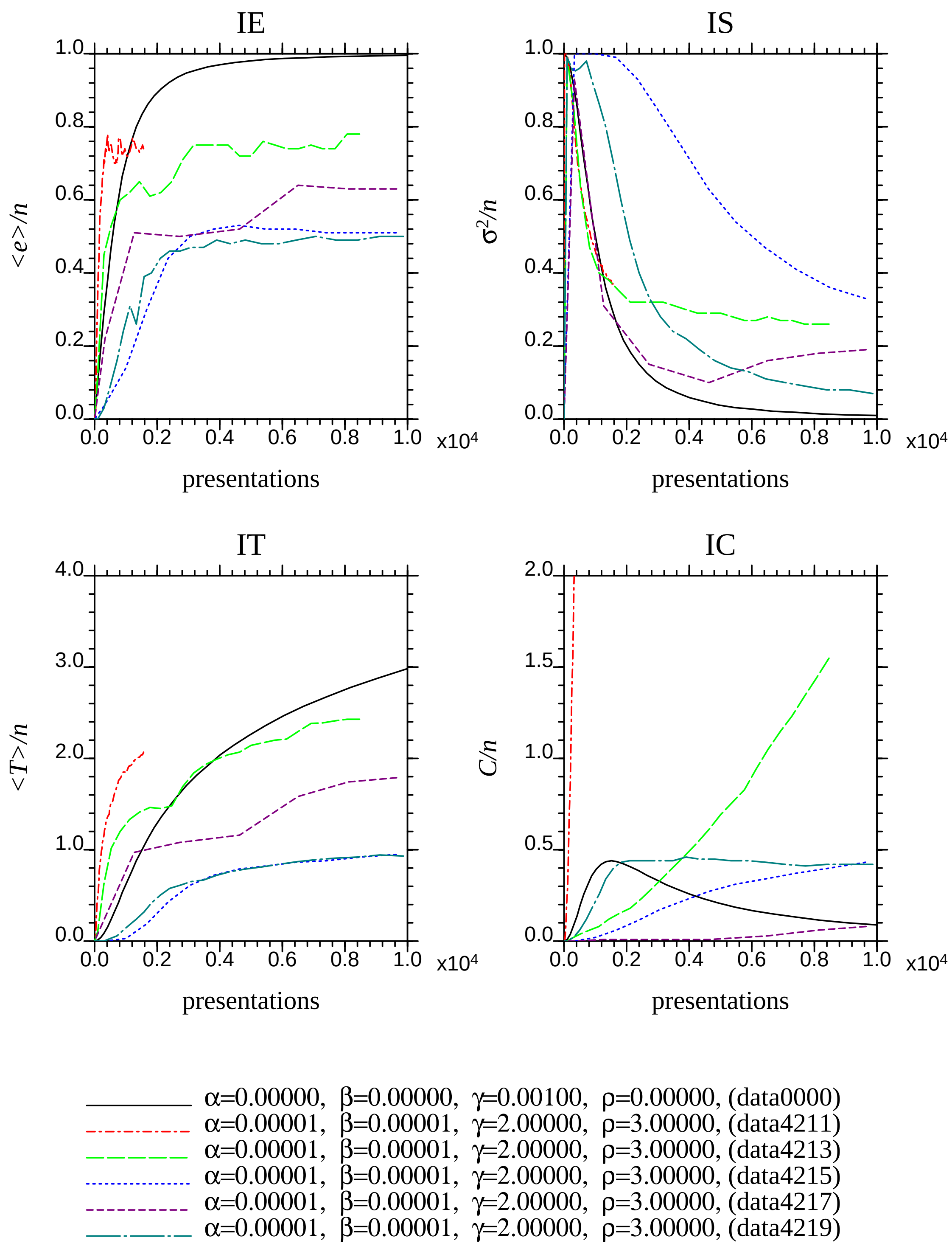

Figure 7.20: Four examples of the 4-2-4 encoder without simulated annealing (b)

7.5 Simulation Experiments

7.5.1 General considerations of the experiments

We have mainly tested the network on the encode problems since, on the one hand, they are representative of many combinatorial problems in that the energy distribution among all the micro states has a combinatorial nature, and on the other, it is easy to derive detailed theoretical results which can be compared with experiments.

All the experiments reported here are performed by writing FORTRAN 77 programs with NAG subroutines for random number generators (G05 routines). The computation is performed in double precision (real*8, integer*4). The whole set-up has been run both in the DOS environment on a transputer attached to a PC and in UNIX environment on a SUN Sparc station IPX. There are no discrepancies due to the difference in hardware.

We have tested both the unit code and the random code problems. It is not obvious which is easier as a learning task. For the random code problem, the codes are spread more evenly in the set R^n as measured by the Hamming distance. For the unit code problem, the Hamming distance between each pair of input codes is always two, but the codes are concentrated in a cone. For each test problem, the TE, TS, TT, TC, IE, IS, IT and IC graphs of several typical runs, and for the ideal learning, are produced. The cooling speed of ideal learning is set arbitrarily to $\gamma_0 = 0.001$ to facilitate comparison.

Remark 7.5.1 These graphs are produced according to the normalisation convention. Before settling down on this convention, we had tried on various other representations, especially those with θ or $\log\theta$ as the x-axis. We do not describe them here since the current representation is superior to them in all respects.

There are several different criteria of convergence. The *total bit error* ϵ, which is the total number of wrong bits in all the n codes after the network has converged, is

$$\epsilon = \frac{n^2}{2}(1 - \langle e\rangle_{\text{final}}).$$

At low temperature, the network is almost deterministic. If there is ϵ bit error in all the n codes, this will reduce $\langle e\rangle$ to $1 - 2\epsilon/n^2$. So it is certain that the network has learned completely when $\langle e\rangle > 1 - 2/n^2$. This usually happens after $\theta = \theta_c/2$, or, equivalently, $\mu = 2\mu_c$.

Another method to assess the quality of learning is to inspect the synapse directly, as was done in [AHS85]. We found empirically from examining sample runs of the experiments that if the learning will eventually be complete, all the n possible codes of the hidden layer are most likely to be used [1] when θ is around θ_c. After $\theta < \theta_c$, all the synapse increase in magnitude monotonically. It is very rare that learning occurs after $\theta < \theta_c$, although it does occur sometimes.

Yet another way to see that learning happens is to produce a matrix $[p_{ij}]$ where p_{ij} is the average of $\langle y_j\rangle$ when the input is the ith code. The ideal learning will result in $p_{ij} \to 2\delta_{ij} - 1$, ie. $[p_{ij}]$ will be a matrix whose diagonal elements are 1 while all the off-diagonal elements are -1. As soon as all the elements of this matrix has correct sign, it is reasonable to say that learning will be complete. This happens at about the same time the synapse has the correct sign.

[1] which means that the synapses will have correct sign of one member in a permutation group corresponding to ideal learning [Sus92].

7.5.2 The 4-2-4 encoder

We use the 4-2-4 encoder as our main test problem. About 200 test runs are done, with various combinations of the learning speed (α, β), the averaging speed (ρ), and the cooling speed (γ) — with both different formulae (§7.4.1) and different parameters. The following combination of parameters are found to be satisfactory, but not necessarily optimal: The reinforcement factor is $e/\theta - T$, the learning rates $\alpha = \alpha_0$ and $\beta = \beta_0\phi$, the cooling schedule $d\mu = \gamma/\sigma$ with $\gamma = \gamma_0\phi$, the running average rate $\rho = \rho_0\phi$, where $\phi = \sigma^2/n$. These are adopted in all the results reported here. They are observed to produce the most stable and reliable results among other choices at the same speed. The choice of α_0, β_0, γ_0 and ρ_0 shown in the captions of figures in this chapter are representatives of the empirically observed optimal range.

In Figure 7.5 & 7.6, the graphs for four typical runs of the 4-2-4 encoder problem are shown, with parameters as given in the figure caption. All of them converge to complete learning. ($\epsilon = 1$ corresponds to $1 - \langle e \rangle = .125$.) All the other runs with the same formulae and comparable or slower speeds converge to complete learning. For larger speeds, there are still some range of combinations of parameters which are almost always converging, but we have not done enough experiments to decide how reliable those parameters are.

7.5.3 The 8-3-8 encoder

To test that the network can be used for problems where $2^n \gg N^2$, we experimented with the 8-3-8 encoder problem. This is important since there are $O(2^n)$ possible outputs and there are $M \in O(N^2)$ weights.

The learning speed, cooling speed and averaging speed must be much slower to avoid stuck in local optima. We estimate that the cooling speed can not be faster than inversely proportional to n^3, which is too slow for conducting extensive experiments, so we have used somewhat faster learning speed which does not guarantee complete learning. Four runs of the encoder problems are given in Figure 7.7 & 7.8. Three of them (DATA8009, DATA8025, DATA8302) converge to complete learning. The other (DATA8304) converge to one bit error in eight codes. ($\epsilon = 1$ corresponds to $1 - \langle e \rangle = 2/8^2 = 0.031$.) Learning is actually complete at around the critical temperature, as for the 4-2-4 encoder.

Since the simulation is rather slow (half hour to several hours computing time), we only performed about 30 experiments. For those runs with speed no more than 10 times faster than those given in Figure 7.7 & 7.8, the final result is usually to learning correctly 5 to 7 out of 8 codes, with the remaining codes having one or two bits error. For runs with still faster speeds, the network will converge to local optima with $\epsilon = 8$, ie. average one bit error in each code.

We also tested encoder problems where there are redundancy in the hidden layer. The learning is faster. Two example, a 8-4-8 encoder and a 6-3-6 encoder, are given in Figure 7.9 & 7.10.

7.5.4 Three layer encoders

To test that the network can learn in the case of more than two layers, we tested the three layer encoder problem. In about 30 tests with 2-2-2-2, 3-2-2-3, and 4-2-2-4 encoders, the network always converge to global optima. The results of four runs of the 2-2-2-2 encoder are presented in Figure 7.11 & 7.12. Three examples, one for each of the 2-2-2-2, 3-2-2-3 and 4-2-2-4 encoder problems are compared in Figure 7.13 & 7.14.

Since we have simply dropped the shift entropy gradient in the SA learning rule (§3.5.3, §6.6), there is no theoretical guarantee that the learning rule would lead to global optimum on a multilayer network. These results on three layer encoders are important empirical support for this approach.

7.5.5 Random code encoders

The results are similar to those of the unit codes. As it is possible to have more codes than the number of output units, we have mainly used this to test the "5-3-5(8) encoder problem", which is an abbreviation of the encoder problem where the network has a 5-3-5 structure while there are eight codes from R^5 to be learned. The amount of information is the same as the 8-3-8 encoder, while the simulation is about five times faster (for our particular simulation program). One difference with the 8-3-8 encoder problem is that one bit error in all the five codes will reduce $\langle e \rangle$ by $2/(8 \times 5) = 0.05$. As with all the other tests, learning is actually complete at around the critical temperature. Four runs are shown in Figure 7.15 & 7.16. All of them converge to complete learning. Others with parameters in the same range also converge to complete learning. With larger speed (within 10 times, and with similar ratio between the parameters) the convergence is to some local optimum with ϵ being one or two.

7.6 Comparison with Results of Others

7.6.1 Different uses of neural networks

To compare our results to the results given in [AHS85], it is necessary to analyse what is actually learned in those experiments. The BM learning rule is a ML learning rule, while ours is a ME learning rule. They are equivalent[2] if we use the system-target interpretation of the ME learning rule, provided that the temperature is not zero in the ME learning, and that the target distribution is not degenerate in ML learning. The latter is satisfied by the experiments on BM in [AHS85] which blurs the target, but not in [PA87]. We explain the details as the following.

First we note that there are basically two ways of using neural networks as a computational tool [Gs88, Pin88], either in the learning process, as in [AHS85, PA87] , or in the dynamics, as in [AK89b, Hop84]. We have to distinguish between the "computational temperature" and the "learning temperature".

What is called "temperature" in the literature corresponds to computational temperature which appears in the SA1 method for accelerating the GS. The computational temperature of BM (different from our θ), is always reduced to a fixed positive value, which can be conveniently assumed to be 1, since there is no intrinsic scale for the energy and computational temperature for the BM. The SA1 used in the BM learning rule is only a way of arriving at the Boltzmann distribution at a faster speed, without it the Boltzmann distribution can also be reached eventually. In the examples given in [AHS85] the final computational temperature is 10 and the learning speed is 2. This network can be equivalently implemented using an annealing schedule where the computational temperature, the energy (corresponding to the connection weights) and the learning speed are scaled down by a factor of 10. We always set computational temperature to 1 in our experiments, ie., we do not use the SA1 method to accelerate the computation.

[2] More precisely, our learning rule is equivalent to the VBM learning rule [LL88].

There seems to be no comparable concept to our "learning temperature" in the literature. All the learning rules we know of are GF-rules, which corresponds to zero learning temperature SA learning rules, or, equivalently, infinite annealing speed SA learning rules.

7.6.2 Comparison of numerical results

The learning time of experiments on BM in [AHS85] are measured in the following form

$$C = (2n+1)E, \qquad E = 20T, \tag{7.6.1}$$

where C is the number of "learning cycles", E is the equilibration time, and T is the "unit time", which is the average time of probing all the neurons in the network once. Therefore the equilibration time E is comparable to that in our experiments. (In the SQFB-SA-ME learning rule, there is one equilibration in one presentation.)

The criteria used in [AHS85] to determine the duration of learning is not quite clear, but it basically involves examining the sign of the weights. We have observed quite reliably that the sign of weights become fixed at about $\mu = \mu_c$. Therefore we define the time it takes to arrive at $\mu = \mu_c$ from $\mu = 0$ as the learning time of our method to be compared with [AHS85]. This criterion itself is not directly applicable to the experiments in [AHS85] since, as analysed earlier, their experiments are set in fixed $\mu = \mu_0 \approx \mu_c$.

The results of [AHS85] are as follows. For 4-2-4 problems, the median run time is $110C = 990E$, the maximum run time is $1810C = 16290E$. For the 8-3-8 problems, the median run time is $1570C = 26700E$. For the 40-10-40 problem, the run time is $900C = 72900E$. Our corresponding results are as follows. For 4-2-4 problem, run time is about $1000E$. For 8-3-8 problem, the runtime is about $40000E$. Therefore, our general automatic learning rule performs comparably (to within a factor of two) with the hand-on learning rule in [AHS85] which is specially tailored to the particular encoder problems.

Since there are so many technical differences [3] between these results which are likely to contribute to the overall speed, it is best to say that the numerical results are comparable. This is impressive considering that the results of [AHS85] is achieved mainly because of human intervention, while with the SA learning rule everything is automatic.

7.6.3 What the original BM learning rule learns

The experiments in [AHS85] do not in any way support the BM learning rule studied there, since they actually implement a crude form of simulated annealing performed at an artificially selected fixed temperature.

The TE, TS, TT and TC graphs for the Hamming distance encoder problem and the particular encoder problems as given in [AHS85] are shown in Figure 7.17 & 7.18. By carefully examining these graphs and the parameters listed in Table 7.1, we can conclude that in these experiments

1. The parameter μ_0 is chosen sufficiently close to μ_c for learning to be reasonably fast. (see the TC graph.)

[3] Such as: batch learning vs. running average, use of SA in the Gibbs sampler, duration of equilibrating, updating step size, etc.

2. μ_0 are chosen sufficiently large so that the final results can be considered as complete learning for all input-output pairs. (see the TE graph.)

In other words, the choices of μ_0 were such that it yields best possible result. Such judicious choice of θ_0 is unlikely to be made randomly, but rather come after many trial-and-error experiments. However, the success reported in [AHS85] depends critically on these choices.

Remark 7.6.1 The experiments in [AHS85] on 40-10-40 problem employed another technique, apart from the crude form simulated annealing: The network was effectively exposed to a changing environment with two phases. In the first phase of learning, the network is only required to learn to produce output distribution unconditionally, ie., independent of the input. In the second phase of learning the network is required to correspond output with input. This can be likened to an "educational scheme" in which a child is taught useful skills without context, and only learns to apply them to actual situations later. Although this idea is very interesting to be explored and applied to various learning tasks, and may be indispensable for tasks complicated to a certain degree, it is clearly independent to the BM learning rule. We shall not take any more time to consider it further.

It is natural to ask how the algorithms will perform without simulated annealing. The BM learning rule without SA (a GF-ML learning rule) has been tested on the 4-2-4 encoder problems in [PA87]. The results are, as one can expect, very disappointing: The network saturates at about 1/4 correct for the 4-2-4 problems. (They consider 4-2-4 as the most difficult, and did not test 8-3-8.) This is so despite the fact that the learning rate is very slow. These results show that simulated annealing, even in the primitive form as in [AHS85], is quite indispensable in the learning process to overcome local optima.

Since in our experimental setting there is no such artificial attributes as batch learning, two phase cycle, clamped output unit, two phase equilibration, etc, it is not possible to directly verify their results. However, it is quite easy to perform GF-ME learning in our experimental setting by simply using e in place of $e\mu - T$. In about 50 tests with various parameters, it never converges to global optima. The results of five typical runs on the 4-2-4 problem are shown in Figure 7.19 & 7.20. The contrast to Figure 7.5 & 7.6 is striking. Furthermore, these local optima are generally insensitive to the learning parameters (within a factor of 10).

7.6.4 The SA-ML learning rule for SQFB-NN

The problem with the techniques employed in [AHS85] is that there is no efficient way to decide the correct amount of blurring, upon which the whole success depends. If the blurring is too small, the learning will be extremely slow, due to the time spent at local optima. If the blurring is too large, the equilibrium distribution might not be in the attracting basin of global optimum, and the final learning result may not be complete. Analytical results for these parameters are unlikely to be available for a general problem. The trial-and-error method is certainly impractical, since that requires solving the problem several times before actually obtaining its solution. Fortunately, it is quite straight forward to adapt the SA-ME learning rule and the BM learning rule (GF-ML) to a SA-ML learning rule for SQFB-NN.

Definition 7.6.1 (SA-ML learning rule for SQFB-NN) The SA-ML learning rule for SQFB-NN is derived from GF-ML learning rule for SQB-NN (Definition 5.2.4) by the following procedure:

1. Choose any energy function whose ground state corresponds to the strict target distribution at zero temperature. For example, the Hamming distance with the identity mapping, or any of the choices made in [AHS85].

2. Blur the target distribution to the Boltzmann distribution corresponding to a certain temperature.

3. Reduce temperature as learning proceeds according to a cooling schedule.

This will do away with the necessity of choosing the amount of blurring artificially.

Part III

Adaptive Computers

Chapter 8

Adaptive Markov Decision Process

8.1 Introduction

The theory of Markov decision processes (MDP), also known as dynamics programming, or theory of controlled Markov chains, has been studied extensively in the literature. In this chapter we generalise it to a theory of adaptive MDP (AMDP), in the sense that all the objects in the theory can be parameterised *continuously* and adapted incrementally.

The most important theoretical contribution is that we introduce the entropy principle and simulated annealing (SA) into MDP so that global optimum can be reached without an exhaustive combinatorially search in the state space and action space. From another point of view, this is a generalisation of SA to multistage problems. Our presentation will be similar to that of [Ros83, DY79], but substantially generalised to allow adaptive improvement.

8.2 The Mathematical Model

Definition 8.2.1 (Markov decision process) A *Markov decision process* (MDP) is a tuple $[X, A, E, T, \pi, \tau, r, \gamma]$, satisfying the following conditions

$$
\begin{aligned}
&E \subseteq S \times A, \qquad T = \mathbb{N}, \\
&\pi \in [T \to [X \xrightarrow{P} A) \cap E), \\
&\tau \in [T \to [E \xrightarrow{P} X)), \\
&r \in [T \to [E \xrightarrow{P} \mathbb{R})), \\
&\gamma \in [X \to [0, 1]),
\end{aligned}
$$

and that $\{\tau_k, \pi_k : k \in T\}$ is independent.

The sets X, A and E are called the *state*, *action*, *event spaces*, respectively. The set T is called the *time domain*. The (stochastic) functions π, τ, r and γ are called the *policy*, *transition*, *reward* and *discount functions*, respectively.

Remark 8.2.1 (1) T is considered as an ordered set. We do not simply use $\mathbb{N}$ in place of T in the definition since it is hoped to be generalised to $T = \mathbb{R}_+$ in the future.

(2) Unlike most conventional methods, in our formulation the discount γ is specified by the problem. The agent can introduce a further discount for the purpose of learning, which shall be discussed later.

For any $x \in X$, the *space of realizable actions* is defined as $A(x) = \{a : (x, a) \in E\}$. A state is called a *terminal state* if $\gamma(x) = 0$. Denote $X_T := \{\text{terminal states}\}$, $X_N := X \setminus X_T$.

Remark 8.2.2 (1) Some other equivalent definitions of the reward are possible. For example, it can be arranged that $r_t = r'_t(x_t, x_{t+1})$, or even $r_t = r''_t(x_{t+1})$. See, for example, [WB91]. (2) Terminal states are those states in which a game is finished and a new one is yet to start. In theory, we can consider the process as continuing infinitely after a terminal state, only that whatever happens is unobservable and does not count in the learning. (3) By adding an imaginary state $0 \in X$ we can assume that the MDP always starts in 0 and terminates in 0. All the different terminal states can be considered as identical. Therefore after discarding unreachable states the MC is irreducible if each state has a non-zero probability of transition into a terminal state.

In a rather informal sense, we call π the *agent*, $[\tau, r, \gamma]$ the *environment* and $[X, A, E, T]$ the *interface* between the agent and the environment.

Example 8.2.1 Consider the game of chess. The agent is the player under study. The environment is the opponent plus the referee (rules). The state space X is composed of possible board positions. The action space A consists of all the meaningful moves, while $A(x)$ consists of legal moves at position x. The event space E consists of possible position-move pairs. The policy π is the style of the player. The environmental transition τ is the rules and the style of the opponent, or more accurately, the "typical style" of possible opponents. The discount $\gamma = 1$ for non-terminal position and $\gamma = 0$ for terminal position. The reward r is 1 for a win, -1 for a lose, and 0 for a tie at the terminal position, and is 0 for any non-terminal position.

Given an arbitrary $x_0 \in X_\Omega$, the MDP realizes a stochastic process $[[x_t, a_t, r_t, \gamma_t] : t \in T]$

(8.2.1)
$$a_t = \pi_t(x_t), \qquad x_{t+1} = \tau_t(x_t, a_t), \qquad r_t = r(x_t, a_t), \qquad \gamma_t = \gamma(x_t).$$

Theorem 8.2.1 *The stochastic process $[[x_t, a_t, r_t] : t \in T]$ generated by (8.2.1) is a Markov chain.*

Proof. This is a direct consequence of the assumption that $\{\tau, \pi\}$ is independent. □

The sequence $[x_0, a_0, x_1, a_1, \ldots]$ is called a *history*. The portion of history between two terminal states is called a *mission*.

Assumption 8.2.2 To avoid technical complications we assume: (1) $E = S \times A$, so $A(x) = A$ for all $x \in X$. (2) Both X and A are finite. (3) $\{\tau_t : t \in T\}$ have identical distributions, ie., the distribution of τ is stationary.

Remark 8.2.3 The subscript t in the mappings π_t, τ_t, etc. is used not because their distributions are dependent on time, but because they are independent across different

t. For example, since $\{\tau_t\}$ are independent and have identical distributions, $\tau_t = \tau_{t+1}$ if and only if τ is deterministic. This might be one reason that the mathematical notion of stochastic mapping was not used in the NN literature, even though its intuitive counterpart has been assumed implicitly in most such studies.

Notation. Denote $F := [X \to \mathbb{R})$, $G := [E \to \mathbb{R})$, $P := [X \xrightarrow{P} A)$,

Let $m := |X|$ and $n := |A|$, then the $F \cong \mathbb{R}^m$,$G \cong \mathbb{R}^{mn}$ as finite dimensional commutative algebras with unit element $\mathbf{1}$.

Definition 8.2.3 (Adaptive Markov decision process) An *adaptive Markov decision process* (AMDP) is a MDP together with $[e, \delta]$, where $H := [E \to \mathbb{R}^N)$, $N \in \mathbb{N}$, $e \in [T \to H)$, and $\delta \in [E \times X \times \mathbb{R} \times P \times H | T \to P \times H)$. The operator δ is called a *development operator*, or an *adaptation operator*.

Given an arbitrary $x_0 \in X_\Omega$, the AMDP realizes a stochastic process $[[x_t, a_t, r_t, \gamma_t, \pi_t, e_t] : t \in T]$

$$a_t = \pi_t(x_t), \qquad x_{t+1} = \tau_t(x_t, a_t), \qquad r_t = r(x_t, a_t), \qquad \gamma_t = \gamma(x_t) \tag{8.2.2}$$

$$[\pi_{t+1}, e_{t+1}] = \delta(x_t, a_t, x_{t+1}, r_t, \pi_t, e_t). \tag{8.2.3}$$

Example 8.2.2 Continuing with Example 8.2.1. δ is the method by which the chess player improves skill, e is the evaluation function, ie., the function the player assigns a "goodness measure" to each board position and move. In the classical studies of "chess machines", one usually studies the policy π, which specifies "how to play chess". On the other hand, the adaptation operator δ specifies "how to improve the skill of playing chess". The study of π may be specific to chess, but that of δ is generic to a whole class of problems, which in our case is the class of discrete time finite state finite action Markov decision process.

When viewed as a black-box, at any fixed moment t, the "behaviour" of the agent can be described by the policy $\pi_t \in [X \xrightarrow{P} A)$, but the main property of the agent is the adaptation operator δ, which specifies how the agent should improve the policy based on experience. The objective of δ is to increase a certain "utility function" $u_t(\xi)$, to be defined §8.3, for all $\xi \in X$, as $t \to \infty$. The utility takes the fixed point form

$$u_t(\xi) = \langle r_t(\xi, \pi_t(\xi)) + \gamma(\xi) u_t(\tau_t(\xi, \pi_t(\xi))) \rangle \qquad \forall \xi \in X. \tag{8.2.4}$$

Remark 8.2.4 Some of the assumptions made here can be relaxed without any change of method, but the theory will become more complicated. For example, the environment can be assumed as "quasi-stationary", which means that τ may not be stationary, but the change in τ_t is not substantially faster than that of π_t. One of our test problem is of such type (§9.4.1). The state space X can be a compact subset of a Euclidean space, if some additional smoothness conditions are imposed on τ, so that the utility function is smooth enough to be approximated by a neural network. (Any function on a discrete set is always continuous.)

Proposition 8.2.2 *When $\gamma \equiv 0$, ie., the future is of no concern to the present, the AMDP can be implemented on a trainable information processor (TIP), such as SQFB-NN.*

Example 8.2.3 Most card games represent non-Markov decision processes because the transitions τ_t are not independent: if a card has been played it cannot be played again. If the player has the ability to remember the sequence of all the previously played cards, and taking this memory and the cards on the hand as the state, the game becomes Markov again. In general, this scheme does not work for practical problems: We do not make our daily decisions based on the memory of the details of all the past events, and it is impractical to write a computer program which saves computing time just by storing all the computations it has done in the past.

Notation. To avoid introducing excessive notations, we shall use the notation of random variables interchangeably with its value, whenever there is no danger of confusing. We shall sometimes omit reference to t if there is no danger of confusing. So the random process produced by AMDP is denoted $[x, a, r, \pi, e]$:

$$a = \pi(x), \qquad x_+ = \tau(x, a), \qquad r = r(x, a), \qquad \gamma = \gamma(x), \tag{8.2.5}$$

$$[\pi_+, e_+] = \delta(x, a, x_+, r, \pi, e). \tag{8.2.6}$$

The *composed transition function* $\tau_\pi \in [X \xrightarrow{P} X)$ is defined by $\tau_\pi(x) := \tau(x, \pi(x))$. The *composed reward function* $r_\pi \in [X \to \mathbb{R})$ is defined by $r_\pi(x) := \langle r(x, \pi(x))\rangle$.

Due to the independence of τ_k and π_k, it is more convenient to use the transition probability instead of transition mappings. Define

$$\begin{aligned}
&P_\pi \in \mathcal{P}(A|X): \ P_\pi(a|x) := \Pr\{\pi(x) = a\},\\
&P_\tau \in \mathcal{P}(Y|X, A): \ P_\tau(y|x, a) := \Pr\{\tau(x, a) = y\},\\
&P \in \mathcal{P}(Y|X): \ P(y|x) := \Pr\{\tau_\pi(x) = y\}.
\end{aligned}$$

Then

$$P(y|x) = \sum_a P_\pi(a|x) P_\tau(y|x, a).$$

$$r_\pi(x) = \sum_a P_\pi(a|x) \langle r(x, a)\rangle .$$

Example 8.2.4 As a further illustration, we consider a grossly simplified picture of how a mathematician works. The state space

$$X = \{x : x = \{[p, l] : p \in \text{propositions}, l \in \{\text{true,false,unknown}\}\}\}$$

is a set of "knowledge status" x, which in turn is a set of "knowledge atoms" $[p, l]$, where p is a formal proposition and l indicates it as either true, false or unknown. A state is therefore a list of knowledge atoms. An action is a proposal of formal calculation (in mind, on paper, using computers, checking references and discussing with colleagues, etc.) intended at adding, removing, or modifying a knowledge atom. The policy π, a mapping from a knowledge status to a proposal of calculation, is the "style" of the mathematician and is usually stochastic. The environmental transition τ is the formal calculation itself which can confirm a knowledge atom, refute one, modify one, or simply lead to nowhere (leaving the state unchanged). The discount γ specifies the importance of potential future results relative to immediate result. The reward r specifies how much the action improves the state, which is usually based on the consideration of significance and conciseness of result minus the cost of calculation. The adaptation operator δ is the manner in which the mathematician improves skill through experience.

Remark 8.2.5 In this example the state space and the action space are infinite, and an action picked "at random" is typically very costly and most likely to lead to nowhere. This is the reason why π, the style of attacking problems, is so important in mathematics. In the above picture we have omitted the further flexibility of inventing new notations and reorganising the systems, etc. The naive scheme of printing all the Gödel numbers and pick out the right ones does not work at all. A possible direct application of the ideas in this example is to use neural networks in computer algebra software where decisions are required for multiple choices (such as which integration rule to apply).

8.3 Utility Function and Value Function

Given τ, r and γ, for any policy π, define [1]

$$
\begin{aligned}
E_\pi \in [F \to F) :\ & E_\pi u(x) := \sum_y P(y|x)u(y), \\
A_\pi \in [F \to F) :\ & A_\pi u(x) := \gamma(x) E_\pi u(x), \\
R_\pi \in [F \to F) :\ & R_\pi(u)(x) := r_\pi(x) + A_\pi u(x), \\
Q \in [F \to G) :\ & Q(u)(x,a) := \langle r(x,a)\rangle + \gamma(x) \sum_y P_\tau(y|x,a)u(y), \\
U_\pi \in [G \to F) :\ & U_\pi q(x) := \sum_a P_\pi(a|x) q(x,a),
\end{aligned}
$$

and denote $R_* = \max_\pi R_\pi$.

It is obvious that $R_\pi = U_\pi Q$. It can be seen that E_π, A_π and U_π are linear operators, while R_π and Q are affine operators. Note that R_* is not an affine operator on F. Linear operators of finite dimension can be represented by matrices. The matrix of E_π is stochastic while that of A_π is non-negative.

The following auxiliary results are useful for discussing the utility function.

Theorem 8.3.1 *Let $A \in \mathbb{R}^{n\times n}$. $\forall b \in \mathbb{R}^n$: define $R_b \in [\mathbb{R}^n \to \mathbb{R}^n)$ by $\forall u \in \mathbb{R}^n$: $R_b(u) := Au + b$. Then $\forall k > 0$:*

$$R_b^k(0) = (I + A + \cdots + A^{k-1})b.$$

$$R_b^k(u) = A^k u + R_b^k(0), \qquad \forall u \in \mathbb{R}^n.$$

Proof. It is easily verified by induction on k. □

Theorem 8.3.2 *Let $A \in \mathbb{R}^{n\times n}$. $\forall b \in \mathbb{R}^n$: define $R_b \in [\mathbb{R}^n \to \mathbb{R}^n)$ by $\forall u \in \mathbb{R}^n$: $R_b(u) := Au + b$. Then the following are equivalent.*

(8.3.1) $\quad \forall b \in \mathbb{R}^n :\ \exists u_b \in \mathbb{R}^n :\ R_b^k(0) \to u_b,\ (k \to \infty)$

(8.3.2) $\quad \exists u_0 \in \mathbb{R}^n :\ \forall u \in \mathbb{R}^n :\ R_0^k(u) \to u_0,\ (k \to \infty)$

(8.3.3) $\quad \forall b \in \mathbb{R}^n :\ \exists u_b \in \mathbb{R}^n :\ \forall u \in \mathbb{R}^n :\ R_b^k(u) \to u_b,\ (k \to \infty)$

(8.3.4)

[1] We adhere to the convention that parenthesis for linear operators are omitted while those of nonlinear (including affine) transforms remain.

Further suppose that

$$\forall \lambda \in \sigma(A): \ |\lambda| \geq 1 \implies \lambda = 1.$$

then any of the above is also equivalent to

$$\forall b \in \mathbb{R}^n: \ \exists! u_b \in \mathbb{R}^n: \ R_b(u_b) = u_b \tag{8.3.5}$$

Furthermore, when any of the above conditions are met, $u_b = (I - A)^{-1} b$, *and the convergence is asymptotically of the rate* $\rho(A)^k$.

Proof. By the previous theorem it is easy to see that both (8.3.1) and (8.3.2) are equivalent to $\rho(A) < 1$, which implies (8.3.3) which in turn implies both (8.3.1) and (8.3.2). They also imply (8.3.5) trivially. Conversely, (8.3.5) implies that $1 \notin \sigma(A)$, which combined with the assumption implies $\rho(A) < 1$. □

Remark 8.3.1 An operator R_b satisfying the above conditions is known as a *contraction mapping* or simply a *contraction*. A theoretical upper bound for the convergence error is ([Wat89])

$$\|u - u_b\|_\infty \leq \|R_b(u) - u\|_\infty / (1 - \|A\|_\infty).$$

See [OR70] for further discussions. This is hardly of any use for us here, since the evaluation of maximum norm requires enumeration over X.

Definition 8.3.1 (Utility function) The function $u_\pi^k := R_\pi^k(0) \in F$ is called the *utility function of finite horizon* k under policy π. If $R_\pi^k(u)$ has a limit as $k \to \infty$ which is independent of u, then it is called the *utility function* under policy π, $u_\pi := \lim_{k\to\infty} u_\pi^k$.

Definition 8.3.2 (Action utility function) The *action utility function* is defined as

$$q_\pi(x, a) := Q(u_\pi)(x, a), \qquad \pi \in P, \ a \in A, x \in X.$$

Remark 8.3.2 Since the space F is finite dimensional the limit in the definition can be taken in any sense. By Theorem 8.3.2 this definition is equivalent to that based on fixed points of R_π. Note that the usual more intuitive definition of u_π as a weighted sum of the rewards [How60, Wit77, DY79, Ros83, Wat89, Sut90] are only applicable for uniform $\gamma < 1$.

Remark 8.3.3 The definition of action utility is inspired by the concept of "action values" in [Wat89, WD92]

$$Q_f(x, a) = Q(u_f)(x, a), \qquad f \in P, \ a \in A, x \in X,$$

$$Q_f(x, g) = R_g(u_f)(x), \qquad f, g \in P, \ x \in X.$$

Definition 8.3.3 (Value function) The function $v^k := R_*^k(0) \in F$ is called the *value function of finite horizon* k. If $R_*^k(u)$ has a limit as $k \to \infty$ which is independent of u, then it is called the *value function*, denoted $v := \lim_{k\to\infty} v^k$.

Proposition 8.3.3 *With the above definitions, the following properties hold* $\forall \pi \in P$:

$$u_\pi^0 = 0, \qquad u_\pi^{k+1} = R_\pi(u_\pi^k), \qquad R_\pi(u_\pi) = u_\pi,$$
$$v^0 = 0, \qquad v^{k+1} = R_*(v^k), \qquad R_*(v) = v.$$

Proposition 8.3.4 *With the above definitions, the following properties hold* $\forall \pi \in P$:

$$\forall u_1, u_2 \in F: \; u_1 \leq u_2 \implies R_\pi(u_1) \leq R_\pi(u_2), \; Q(u_1) \leq Q(u_2),$$
$$\forall q_1, q_2 \in G: \; q_1 \leq q_2 \implies U_\pi q_1 \leq U_\pi q_2,$$

Definition 8.3.4 The *maximum evaluation* is defined as $u_* := \sup_{\pi \in P} u_\pi$. The set of *optimal policy* is defined as

$$P_* := \{\pi \in P: \; u_\pi = u_*\}.$$

By this definition alone it is not clear whether there exists any optimal policy at all. But the following two remarkable theorems say that, provided the value function exists, the optimal policy always exists, and can be reached by local improvements.

Theorem 8.3.5 (Policy improvement) *Suppose* $\pi_1, \pi_2 \in P$ *satisfy* $R_{\pi_2}(u_{\pi_1}) \geq u_{\pi_1}$, *then* $u_{\pi_2} \geq u_{\pi_1}$. *If* $R_{\pi_2}(u_{\pi_1}) = u_{\pi_1}$, *then* $u_{\pi_2} = u_{\pi_1}$.

Theorem 8.3.6 (Optimality) *Suppose the value function* v *exists. Then* $v = u_*$, $P_* \neq \emptyset$, *and* $\forall \pi \in P$:

$$\pi \in P_* \iff R_*(u_\pi) = u_\pi \iff u_\pi = v \iff R_*(u_\pi) = v.$$

The proof of the above two theorems can be found in [Ros83].

Remark 8.3.4 The seemingly similar conditions $R_\pi(v) = u_\pi$ and $R_\pi(v) = v$ are *not* sufficient for π be optimal, since they are satisfied by any one step game, regardless of the policy.

8.4 Properties of Utility Function

The existence of the utility function depends on the spectral radius of A_π which further depends on γ. However, it is in general not practical to represent, not to say to find out the eigenvalues of such a large matrix by a computer (10^{120} dimensional for chess). In this section we generalise the concept of "lifetime", and formalise the intuition that if the lifetime is finite and the reward is bounded then the utility is well defined.

It is easy to see that if $r(x,a) = 1$ for all x and a, then $r_\pi = \mathbf{1}$. In this case $R^k_\pi(0)$ is monotonically increasing in k. Define the *virtual lifetime* $L_\pi(x) := u_\pi = \lim_{k\to\infty} R^k_\pi(0)(x)$ with $r_\pi = \mathbf{1}$, where the limit can be infinity. When $\gamma(x) \in \{0, 1\}$, $L_\pi(x)$ is lifetime in the normal sense ($\gamma(x) = 1$ means "live", $\gamma(x) = 0$ means "death"). It follows from Theorem 8.3.1 that

$$L_\pi = \sum_{k=0}^{\infty} A^k_\pi \mathbf{1} = (I - A_\pi)^{-1} \mathbf{1}.$$

Lemma 8.4.1 *Let* A *be a non-negative matrix and* r *be a vector. Denote by* x, y *their subscripts. Then*

$$|Ar(x)| = |\sum_y A_{xy} r_y| \leq \sum_y |A_{xy}||r_y| \leq \sum_y A_{xy} \|r\|_\infty = A\mathbf{1}(x) \|r\|_\infty.$$

Theorem 8.4.2 *For any reward function r and policy π, it holds that*

$$\|u_\pi^k\|_\infty \le \|r_\pi\|_\infty \|L_\pi\|_\infty ,$$

where both sides are allowed to be infinity.

Proof. Since A_π is non-negative, it follows that above lemma that

$$\begin{aligned}\|u_\pi^k\|_\infty &= \max_x \left| \sum_{l=0}^k A_\pi^l r_\pi(x) \right| \le \max_x \sum_{l=0}^k \left| A_\pi^l r_\pi(x) \right| \\ &\le \|r_\pi\|_\infty \max_x \sum_{l=0}^k A_\pi^l \mathbf{1}(x) \le \|r_\pi\|_\infty \|L_\pi\|_\infty .\end{aligned}$$

□

Theorem 8.4.3 (Existence of utility) *For any $p \in P$. The following three are equivalent*

(8.4.1) $\quad \|L_\pi\|_\infty < \infty,$

(8.4.2) $\quad \exists r \in F : \ \exists u_\pi \in F : \ \forall u \in F : \ R_\pi^k(u) \to u_\pi, \ (k \to \infty),$

(8.4.3) $\quad \forall r \in F : \ \exists u_\pi \in F : \ \forall u \in F : \ R_\pi^k(u) \to u_\pi, \ (k \to \infty).$

Furthermore, the above implies

$$\|u_\pi\|_\infty \le \|r_\pi\|_\infty \|L_\pi\|_\infty .$$

Proof. By the inequality

$$\begin{aligned}\|R_\pi^k(u)(x) - R_\pi^m(u)\|_\infty &\le \left\| \sum_{l=m+1}^k A_\pi^l r_\pi \right\|_\infty + \left\| A_\pi^m \left(A_\pi^{k-m} - I\right) u \right\|_\infty \\ &\le \sum_{l=m+1}^k \|A_\pi^l r_\pi\|_\infty + \left\| A_\pi^m \left(A_\pi^{k-m} - I\right) u \right\|_\infty \\ &\le \|r_\pi\|_\infty \left\| \sum_{l=m+1}^k A_\pi^l \mathbf{1} \right\|_\infty + \|A_\pi^m\|_\infty \left\| \left(A_\pi^{k-m} - I\right) u \right\|_\infty ,\end{aligned}$$

it is clear that $\|L_\pi\|_\infty < \infty$ implies (8.4.2) and (8.4.3), which are themselves equivalent to each other, due to Theorem 8.3.2. The converse is obviously true, if we take $r = \mathbf{1}$. The second part of the theorem follows immediately from Theorem 8.4.2. □

Theorem 8.4.4 *The following holds.*

$$\|L_\pi\|_\infty \le 1/\left(1 - \|A_\pi\|_\infty\right), \qquad \|A_\pi\|_\infty = \|\gamma\|_\infty .$$

Proof. Since $\rho(A_\pi) \le \gamma_0$, it follows that

$$\|L_\pi\|_\infty = \left\| \sum_{k=0}^\infty A_\pi^k \mathbf{1} \right\|_\infty \le \sum_{k=0}^\infty \|A_\pi\|_\infty^k \|\mathbf{1}\|_\infty = 1/\left(1 - \|A_\pi\|_\infty\right).$$

The second part of the theorem follows that $\|A\|_\infty = \max_i \sum_j |A_{ij}|$ for any matrix $A = [A_{ij}]$ and that E_π is stochastic. □

Corollary 8.4.5 *If $\|\gamma\|_\infty < 1$, then $\|L_\pi\|_\infty < \infty$, and u_π exists.*

Remark 8.4.1 The classical counterpart of this, where $\gamma \equiv \gamma_0 < 0$, is trivial, and is even not mentioned in most references. In many practical problems the condition $\|\gamma\|_\infty < 1$ does not hold, so the criterion with L_π is more general.

The following proposition on a special symmetric case reveals that the discount rate γ and the probability of termination contributes similarly to L_π.

Proposition 8.4.6 *Suppose $\exists \gamma_0 < 1$ such that $\forall x \in X_N : \ \gamma(x) = \gamma_0$. Further suppose $\exists p < 1$ such that*

$$\forall x \in X_N : \ \Pr\{\tau_\pi(x) \in X_N | x \in X_N\} = p.$$

Then $L_\pi(x) = 1/(1 - p\gamma_0)$, $\forall x \in X_N$.

Proof. Consider the associated problem with state space $\{X_N, X_T\}$. Then $\forall x \in X_N$: $L_\pi(x) = L_\pi(X_N)$, and it follows $\Pr\{\tau_\pi(x) = X_N | x = X_N\} = p$ that $L_\pi(X_N) = 1 + p\gamma_0 L_\pi(X_N)$. □

The state space X with transition mapping τ_π forms a Markov chain (MC). The concept of *closed set* and *recurrent state* of a MC are defined as in [CM65]. Closed set is also called *recurrent chain* in [How60] and *commutative set* in [Sen73b]. The following theorem gives a sufficient condition for the existence of utility function.

Theorem 8.4.7 *Suppose that each closed set of the MC (X, τ_π) has at least one terminal state. Then $\|L_\pi\|_\infty < \infty$.*

Proof. Suppose there is a state x such that $L_\pi(x) = \infty$. Then each state y with $P(x|y) > 0$ must also have $L_\pi(y) = \infty$. It follows that any state y in the same closed set as x must have $L_\pi(y) = \infty$. This is impossible since for any terminal state y it holds by definition $L_\pi(y) = 0$. The contradiction implies that there is no state with infinite lifetime. □

Remark 8.4.2 If we consider all the terminal states as identical, then the above MC has only one closed set, and all the non-terminal states are transient. The MC is therefore ergodic.

8.5 Equilibrium Policy and Simulated Annealing

8.5.1 The equilibrium policy

Now we consider the development operator δ, which specifies how to improve the policy. The objective of learning is to adjust π to increase u_π. Two most important methods in dynamic programming (DP) are based on the idea of approximating u_* by u_π and approximating v by v^k.

Policy Iteration Method (DP-PI)

1. Set $u^0(x) := 0$.

2. Set π^k to satisfy $R_{\pi^k}(u^k) = R_*(u^k)$.

3. Set $u^{k+1} = u_{\pi^k}$ by iteration
   (a) Set $u^{k,0} = u^k$.
   (b) Iterate $u^{k,l+1} := R_{\pi^k}(u^{k,l})$.
   (c) Set $u^{k+1} := \lim_{l\to\infty} u^{k,l}$.
4. Goto step 2.

Value Iteration Method (DP-VI)

1. Set $u^0(x) := 0$.
2. Set π^k to satisfy $R_{\pi^k}(u^k) = R_*(u^k)$.
3. Set $u^{k+1} := R_*(u^k)$.
4. Goto step 2.

Theorem 8.5.1 *With DQ-VI method, $u^k = u_{\pi^k} \geq u^{k-1}$. With DP-VI method, $u^k = v^k \geq u^{k-1}$. For both methods, $u^k > u^{k-1}$ for $u^k \neq u^*$, and $u^k = u_*$ for $k \geq |[X \to A)|$.*

Proof. It follows Theorem 8.3.5, that u^k is is increasing. It is also obvious that $\forall k : \exists \pi \in [X \to A) : u_\pi = u^k$. □

The value iteration method can also be written in the following more convenient form.

1. Set $u := 0$.
2. Set $\pi := \arg\max_\pi R_\pi(u)$.
3. Set $u := R_\pi(u)$.
4. Goto step 2.

These are well known results in DP (eg. [Ros83]). One of their distinctive features is that at each step the "optimal" action for current state under current utility function is chosen. Such methods have the following deficiencies: (1) The computation of R_π involves computing P_τ explicitly, which is impractical if X cannot be enumerated; (2) It requires that τ be stationary, which cannot be generalised to games of competition where τ depends on the policy of competing players and is therefore not stationary; (3) The computation of $\arg\max_\pi$ involves enumeration of A; (4) The solution π depends discontinuous on u. (5) They cannot be implemented on-line as AMDP.

In other words these methods assume complete information on each possible $[x, a, x_+]$. This is not achievable for many practical problems. The only practical method is usually to use a parameterised representation of the policy and evaluation of the utility, and to approximate the desired ones by modifying the parameters. It is often also desirable to implement such a method on-line, ie., to use the samples of $[x, a, x_+]$ which actually occurred instead of enumerating all the possibilities. This makes it neither desirable nor necessary to always choose the "optimal action" for each x: the agent would do better "exploring" as many different actions as possible so as to approximate the utility function more accurately, while continuing "exploiting" the evaluation of the utility, ie., choosing actions with as high evaluation as possible. The compromise between these two requirements can be achieved by assigning a value to the missing information (which might be gained by more experience), and choose the policy which maximise the compound evaluation (the sum of utility value and the information value). Its distribution is obviously $BD(u, \mu)$.

Let $\mu > 0$, define

$$\begin{aligned}
S_\pi \in F: \ & S_\pi(x) := \sum_a P_\pi(a|x) \log P_\pi(a|x), \\
R_{\pi,\mu} \in [F \to F]: \ & R_{\pi,\mu}(u)(x) := R_\pi(u)(x)\mu - S_\pi(x), \\
U_{\pi,\mu} \in [G \to F]: \ & U_{\pi,\mu}(q)(x) := U_\pi q(x)\mu - S_\pi(x), \\
Q_{\pi,\mu} \in [F \to G]: \ & Q_{\pi,\mu}(u)(x,a) = Q(u)(x,a)\mu - \log P_\pi(a|x).
\end{aligned}$$

Corollary 8.5.2 $R_{\pi,\mu} = U_\pi Q_{\pi,\mu} = U_{\pi,\mu} Q$.

Exact SA method (AMDP-VI-ESA)

1. Set $u := 0$. Set $\mu = 0$.
2. Set $\pi := \arg\max_\pi R_{\pi,\mu}(u)$.
3. Set $u := R_\pi(u)$.
4. Increase μ. Goto step 2.

It is obvious that as $\mu \to \infty$ the ESA method tends to the DP-VI method.

As soon as we do not require finding a deterministic policy, it is possible to incrementally improve the policy, as well as the evaluation of the utility functions.

Approximate SA method 1 (AMDP-VI-ASA1)

1. Set $e := 0 \in F$, Set $\mu = 0$. Set $\pi = \text{rnd}$.
2. Change π to increase $R_{\pi,\mu}(e)$.
3. Change e to approximate $R_\pi(e)$.
4. Increase μ. Goto step 2.

It is obvious that if steps 2 and 3 in AMDP-VI-ASA1 are exact, then $e = u$ and AMDP-ASA1 reduces to AMDP-ESA method.

Since $R_\pi = U_\pi Q$ involves taking averages over τ and π, it is a natural idea to use two evaluation functions e_1 and e_2 to approximate u_π and q, respectively. If two functions $e_1 \in F, e_2 \in G$ satisfies $e_1 = U_\pi e_2$, $e_2 = Q(e_1)$, then $e_1 = u_\pi$. Using $R_{\pi,\mu}(e_1)$ generalises those methods used in [BSA83, Sut90], while using $U_{\pi,\mu}(e_2)$ generalises the Q-learning method [Wat89].

Approximate SA method 2 (AMDP-ASA2)

1. Set $e_1 := 0$, $e_2 := 0$. Set $\mu = 0$. Set $\pi = \text{rnd}$.
2. Change π to increase $R_{\pi,\mu}(e_1)$ or to increase $U_{\pi,\mu}(e_2)$.
3. Change e_2 to approximate $Q(e_1)$.
4. Change e_1 to approximate $U_\pi e_2$.
5. Increase μ. Goto step 2.

It is obvious that if the step 3 is executed exactly, then AMDP-VI-ASA2 reduces to AMDP-VI-ASA1. If furthermore steps 2 and 4 are exact, then $e = u$ and it further reduces to AMDP-VI-ESA.

8.5.2 Incremental improvement

The policy π, the evaluations e_1 and e_2 can each be implemented by a trainable information processor (TIP). Here we study them without considering what type of TIPs are used. In the next chapter we shall study in detail how to implement them on neural networks.

In all the following considerations we shall assume that our objective function is defined for each $x \in X$. The total result is an average over all the states x with probability $p(x)$, the probability of that state occurring in a mission.

Modify the policy

Let $T_\pi(a|x)$ be the tendency associated with $P_\pi(a|x)$, which depends on some parameter w. It follows Theorem 3.4.7 that

$$\frac{\partial U_{\pi,\mu}(q)(x)}{\partial w} = \left\langle \frac{\partial T_\pi}{\partial w}, q\mu - T_\pi \right\rangle_x . \tag{8.5.1}$$

Therefore step 2 in ASA1 and ASA2 can be carried out by modifying w with

$$\Delta w_x = \alpha \frac{\partial U_{\pi,\mu}(q)(x)}{\partial w} = \alpha \left\langle \frac{\partial T_\pi}{\partial w}, q\mu - T_\pi \right\rangle_x ,$$

so that

$$\Delta w = \alpha \sum_x p(x) \left\langle \frac{\partial T_\pi}{\partial w}, q\mu - T_\pi \right\rangle_x .$$

Remark 8.5.1 The two different definitions of step 2 of AMDP-VI-ASA2 will give different results when $e_2 = Q(e_1)$ is not satisfied, depending on the exact implementation. It is not known which is better, and under what consideration. Several variants of these two methods have been empirically compared in an artificially designed game situation in [Lin92], but since there are many other factors different in that setting, the result is not conclusive for general problems. Note that e_2 introduces systematic error (bias) while $Q(e_1)$ introduces random error (variance). The computational demand on e_2, defined on $X \times A$, is much higher than that on e_1, defined on X. Our numerical experiments show that if the neural network implementing e_2 is not large enough, it is even worse than using e_1 alone (§9.6).

Modify the evaluation

The evaluations can be improved by increasing the performance measure. This requires a definition of goodness of the evaluation, which is by no means a trivial task. The first choice one would take is simply set

$$\epsilon := -\sum_x p(x)\frac{1}{2}(e(x) - u_\pi(x))^2 \tag{8.5.2}$$

Computing its gradient

$$\frac{\partial \epsilon}{\partial w} = \sum_x p(x)(u_\pi(x) - e(x))\frac{\partial e(x)}{\partial w} \tag{8.5.3}$$

requires the utility function u_π, which is of course not available. When the utility function u_π is well defined, it is always the case that $R_\pi(e)$ is a better estimate of u_π than e is. Substituting $R_\pi(e)$ in place of u_π in (8.5.3) gives

$$\frac{\partial \epsilon}{\partial w} = \sum_x p(x)(R_\pi(e)(x) - e(x))\frac{\partial e(x)}{\partial w}. \tag{8.5.4}$$

When there are estimations e_1, e_2 of u_π and q_π instead of a single e, we can use

$$\epsilon := \sum_x p(x)\left(\epsilon_1(x) + P_\pi(a|x)\epsilon_2(x,a)\right),$$

$$\epsilon_1(x) := -\frac{1}{2}\left(e_1(x) - U_\pi e_2(x)\right)^2, \qquad \epsilon_2(x,a) := -\frac{1}{2}\left(e_2(x,a) - Q(e_1)(x,a)\right)^2,$$

with

$$\frac{\partial \epsilon_1(x)}{\partial w} = \left(U_\pi e_2(x) - e_1(x)\right)\frac{\partial e_1(x)}{\partial w},$$

$$\frac{\partial \epsilon_2(x,a)}{\partial w} = \left(Q(e_1)(x,a) - e_2(x,a)\right)\frac{\partial e_2(x,a)}{\partial w},$$

Remark 8.5.2 It should be noted that $R_\pi(e)$, $U_\pi e_2$ and $Q(e_1)$ should be considered fixed in these equations, since they represents the teaching signal for e, e_1 and e_2. There is a detailed discussion of this issue in [Wer90b].

8.5.3 Relation between the discount and the temperature

The convergence of the utility function depends on $L_\pi < \infty$. For some problems, the condition $\gamma \le \gamma_0 < 1$ is not satisfied, so both L_π and u_π may approach infinity by changing π, and the convergence is only guaranteed for non-zero θ. This is more of a problem when the utility is monotonically related to L_π.

Example 8.5.1 In the "pole-balancing problem" [BSA83], the objective is to balance a "pole" on a "cart" for as long as possible by pushing the cart to left or right by fixed force in fixed intervals. Therefore $L_\pi \to \infty$ as $\pi \to \pi^*$, since the optimal policy can keep the pole balanced forever.

To get a generally applicable and robust algorithm, we use an "artificial" discount factor γ multiplied on top of whatever discount the problem itself might supply. More specifically, we set $\gamma := \gamma'\gamma''$, where γ' is an artificial discount while γ'' is the discount defined by problem.

It is ready to see that, when γ and μ are smaller, the convergences of both evaluation and policy are faster and more stable but with larger deviation from the "optimal policy". Therefore, to get good result in short time, it is better to increase γ and μ gradually. Intuitively, γ represent the confidence (of evaluation of next state), while θ is uncertainty (of choosing the next action).

There is currently no available theory for the optimal value of γ'. In the literature on DP and MDP γ' is either set to unity (for problems which guarantee $L_\pi < \infty$), or to some more or less arbitrarily chosen fixed value in $(0, 1)$ such as 0.9 (for problems which would lead to $L_\pi \to \infty$ for $\gamma = 1$).

We suggest that the optimal schedule for changing μ and γ' should be such that the deviation from the optimal policy effected by both of them should be comparable. We stipulate that the reduction of virtual lifetime caused by μ and γ' should be equal to each other for a certain problem. Consider the example in Proposition 8.4.6 with L_π as utility. By making an analogy with §7.3.3, we see that all of $\gamma' = \tanh\mu$, $\gamma' = 1 - e^{-2\mu}$, and $\gamma' = 1/(1 + e^{-2\mu})$ are all reasonable choices. They are asymptotically equal to each other. For definiteness, the formula $\gamma' = \tanh\mu$ will be used in all our experiments reported in the next chapter. In our other exploratory experiments other schemes have been used, such as $\gamma' = 1/(1+\theta) = 1 - 1/(1+\mu)$. No substantial difference is observed.

Remark 8.5.3 One advantage of our method is that equilibrium is always maintained throughout the learning process. This is only possible because we start from $\gamma' = 0$ and $\mu = 0$. The equilibrium solution is always unique, in the sense that there is only one maximum entropy distribution over the set of pure strategies. The initial evaluations and policy corresponds to $\gamma' = 0$ and $\mu = 0$. The value function and the optimal policy corresponds to $\gamma' = 1$ and $\mu = \infty$. The learning process follows a curve going through the initial and optimal policies.

Remark 8.5.4 In classical theory of dynamic programming, the *negative reward problems*, *positive reward problems* are usually studied separately, since for the former L_π decreases while for the latter L_π increases as $\pi \to \pi^*$. In our formulation the optimal solutions is defined as limits of equilibrium solutions there is no need to study them separately.

Our formulation also unifies different approaches in DP, such as *fixed horizon programming*, *discounted reward programming*, *total return programming* and *average gain programming* [How60, Bel57, Ros83], by the following theorem.

Theorem 8.5.3 *Let $E \in \mathbb{R}^{n\times n}$ with $\rho(E) \le 1$. Then $\forall r \in \mathbb{R}^n$:*

1. *The following holds whenever the limit of the denumerator exists:*

$$\lim_{h\to\infty} \left(\sum_{k=0}^{h} E^k r \Big/ \sum_{k=0}^{\infty} \left(1 - \frac{1}{h}\right)^k E^k r \right) = 1. \tag{8.5.5}$$

2. *The following holds whenever the sum at r.h.s exists:*

$$\lim_{\gamma\to 1^-}\sum_{k=0}^{\infty}\gamma^k E^k r = \sum_{k=0}^{\infty} E^k r. \tag{8.5.6}$$

3. *The following holds.*

$$\lim_{\gamma\to 1^-}(1-\gamma)\sum_{k=0}^{\infty}\gamma^k E^k r = \lim_{h\to\infty}\frac{1}{h}\sum_{k=0}^{h} E^k r. \tag{8.5.7}$$

Proof. Let $\lambda \in \mathbb{C}$ with $|\lambda| \le 1$. Then

$$\lim_{h\to\infty}\frac{\sum_{k=0}^{h}\lambda^k}{\sum_{k=0}^{\infty}\left(1-\frac{1}{h}\right)^k\lambda^k}$$

$$= \begin{cases} \lim_{h\to\infty}\frac{h}{h} = 1, & (\lambda = 1), \\ \lim_{h\to\infty}\left(\frac{1-\lambda^{h+1}}{1-\lambda}\Big/\frac{1}{1-\left(1-\frac{1}{h}\right)\lambda}\right) = & \begin{cases} 1, & (|\lambda|<1,), \\ \text{not exist}, & (\text{otherwise}). \end{cases} \end{cases}$$

$$\lim_{\gamma\to 1^-}\sum_{k=0}^{\infty}\gamma^k E^k r = \begin{cases} \infty, & (\lambda = 1), \\ \frac{1}{1-\lambda}, & (|\lambda|<1,), \\ \text{not exist}, & (\text{otherwise}). \end{cases}$$

$$\lim_{\gamma\to 1^-}(1-\gamma)\sum_{k=0}^{\infty}\gamma^k\lambda = \lim_{\gamma\to 1^-}\frac{1-\gamma}{1-\gamma\lambda} = \begin{cases} 1, & (\lambda = 1), \\ 0, & (\lambda \ne 1). \end{cases}$$

$$\lim_{h\to\infty}\frac{1}{h}\sum_{k=0}^{h}\lambda^k = \begin{cases} 1, & (\lambda = 1), \\ \lim_{h\to\infty}\frac{1}{h}\frac{1-\lambda^{h+1}}{1-\lambda} = 0 & (\lambda \ne 1). \end{cases}$$

The theorem is proved by applying these results to the eigenvalues of E. □

Therefore, finite horizon utility function with horizon h can be asymptotically approximated by a discounted utility function with discount $\gamma = 1 - 1/h$. Total return or the average gain utility functions can be approximated by discounted utility function (possibly normalised, see §9.5) with discount approaching unity.

8.6 Equilibrium Meta-States

In this section we study what happens when u, q and π can be changed at the same time.

Definition 8.6.1 The tuple $[u, q, \pi]$ is called a *meta-state*. Given μ, γ, define

$$\begin{aligned}
\mathcal{Q} \in [P \to G) &: \ \mathcal{Q}(\pi) := (QU_\pi)^\infty(0), \\
\mathcal{U} \in [P \to F) &: \ \mathcal{U}(\pi) := (U_\pi Q)^\infty(0), \\
\Psi \in [G \to P) &: \ \Psi(q) = \pi : \ P_\pi(\cdot|x) \sim \exp\left(q(x,\cdot)\mu\right), \\
\mathcal{D}_q \in [G \to G) &: \ \mathcal{D}_q(f) := QU_{\Psi(q)}(f), \\
\mathcal{P} \in [G \to G) &: \ \mathcal{P}(q) := \mathcal{D}_q^\infty(0), \\
\mathcal{V} \in [G \to G) &: \ \mathcal{V}(q) := \mathcal{D}_q(q).
\end{aligned}$$

By Theorem 8.3.2, $\mathcal{Q}$ and $\mathcal{U}$ are well defined. Most of these operators depend on the parameters μ and γ which are not explicit in the notation. Therefore they are strictly speaking families of operators.

Proposition 8.6.1

$$\mathcal{Q}(\pi) = Q\mathcal{U}(\pi), \tag{8.6.1}$$
$$\mathcal{U}(\pi) = U_\pi \mathcal{Q}(\pi), \tag{8.6.2}$$
$$\mathcal{P}(q) = \mathcal{Q}\Psi(q), \tag{8.6.3}$$
$$\mathcal{V}(q) = QU_{\Psi(q)}(q). \tag{8.6.4}$$

It can be easily verified that the policy iteration method and value iteration method correspond to the operators $\mathcal{P}$ and $\mathcal{V}$, respectively.

Definition 8.6.2 Given μ, γ, if

$$\pi = \Psi(q), \quad u = U_\pi q, \quad q = Q(u), \tag{8.6.5}$$

then $[u, q, \pi]$ is called an *equilibrium meta-state*, and u, q, π are called *equilibrium utility function*, *equilibrium action utility function* and *equilibrium policy*, respectively.

Theorem 8.6.2 *The following three are equivalent.*

$$q = \mathcal{V}(q), \tag{8.6.6}$$
$$q = \mathcal{P}(q), \tag{8.6.7}$$
$$\exists \pi = \Psi(q) : \ \exists u = U_\pi q : \ q = Q(u). \tag{8.6.8}$$

Proof. Since $\mathcal{V}(q) = QU_{\Psi(q)}(q)$, it is obvious that (8.6.6) and (8.6.8) are equivalent. Now suppose $q = \mathcal{P}(q)$. Denote $\pi := \Psi(q)$, $u := U_\pi q$, then

$$\mathcal{P}(q) = \left(QU_{\Psi(q)}\right)^\infty (0) = \left(QU_\pi\right)^\infty (0) = (QU_\pi)(q) = Q(u).$$

Conversely, suppose that (8.6.8) holds. Denote $q' := \mathcal{P}(q) = \left(QU_\pi\right)^\infty (0)$, then

$$(QU_\pi)(q) = q, \qquad (QU_\pi)(q') = q'.$$

So $q = q'$, since QU_π is a contraction (Theorem 8.3.2). □

This means that the fixed points of both value iteration and policy iteration coincide with the equilibrium meta-states. The remaining question is whether iteration by either of $\mathcal{V}$ and $\mathcal{P}$ does converge to fixed points. These questions are unresolved. Here we state two conjectures which may be compared with experiments.

Conjecture 8.6.1 The iteration of $\mathcal{P}$ and $\mathcal{V}$ converge to one of a finite number of fixed points, dependent on the temperature and discount. Formally, $\forall \mu, \gamma : \exists G^* \subset G : |G^*| < \infty, \forall q \in G : \exists q^* \in G^* :$

$$\mathcal{P}^k q \to q^*, \quad \mathcal{V}^k q \to q^*, \qquad (k \to \infty).$$

It will be shown in Chapter 9 that the methods discussed converge for testing problems which were considered hard in the past. So there is reason to expect that this holds for some general class of problems under some general conditions. One way to prove the convergence to (unique) fixed point is to show that the iteration operator is a contraction [OR70]. Unfortunately, it seems that neither $\mathcal{P}$ nor $\mathcal{V}$ are contractions globally. The following example shows that the norm of $\mathcal{V}$ may be larger than unity.

Example 8.6.1 Let $u \in F$, $q := Q(u)$, $q_i := q(x, a_i)$, $\pi := \Psi(q)$, $p_i := P_\pi(a_i|x)$ and $v := U_\pi q$. Then $p_i = \exp(q_i\mu)/\sum_j \exp(q_j\mu)$, $v = \sum_i p_i q_i$, and

$$\|\Delta q\|_\infty \leq \gamma \|\Delta u\|_\infty, \qquad \|\Delta v\|_\infty \leq \left\|\frac{\partial v}{\partial q}\right\|_1 \|\Delta q\|_\infty.$$

$$\frac{\partial v}{\partial q_i} = \mu p_i(q_i - v) + p_i. \tag{8.6.9}$$

Consider the case where there are only two values of q, ie. there are only two possible a_i. Without loss of generality, we can assume that $q_\pm = \pm 1$. Then by some tedious but straight forward derivation, we get

$$\frac{\partial v}{\partial q_\pm} = p_\pm((q_\pm - v)\mu + 1) = \frac{1}{2}(1 \pm \tanh\mu)(\mu(\pm 1 - \tanh\mu) + 1)$$
$$= \frac{1}{2}(1 \pm \tanh\mu \pm \mu(1 - \tanh^2\mu)).$$

It follows that $\|\partial v/\partial q\|_1$ has maximum value μ_c at $\mu = \mu_c \approx 1.2$, and tends to zero for $\mu \to 0$ and to one for $\mu \to \infty$. The choice of $\gamma = \tanh\mu$ leads to asymptotically $\gamma\left\|\frac{\partial v}{\partial q}\right\|_1 \to 1 + 2\mu\exp(-2\mu)$ as $\mu \to \infty$.

This example does not necessarily mean that the iteration will not converge to fixed points, since when there there are more than one fixed points, the iteration operator is always *not* a contraction near the boundary between the attractive basins of two fixed points. In fact, in our numerical simulations we have seldom encountered situations to suspect that there are attractors other than fixed points. As a precaution, if $\gamma\left\|\frac{\partial v}{\partial q}\right\|_1 < 1$ is maintained in the above example, the operator $\mathcal{V}$ will always be a contraction, and the system will be globally stable, although there will always be a distortion in the solution which favours immediate reward to rewards in distant future. In general, the analysis based on norm is worst case analysis, which almost always exaggerate the errors.

Proposition 8.6.3 *For $\gamma = 0$ there is a unique equilibrium meta-state* $[0, 0, \mathrm{rnd}]$.

Conjecture 8.6.2 There is a region $C \subseteq \mathbb{R}_+ \times [0,1] \subset \mathbb{R}^2$, $[0,0] \in \bar{C}$, $c^* := [\infty, 1] \in \bar{C}$, where $\bar{C}$ is the *completion* of C, such that as $[\mu, \gamma] \in \bar{C}$ moves continuously from $[0,0]$ to c^*, there is a corresponding branch of equilibrium meta-states $[u, q, \pi]$ moving continuously from $[0, 0, \mathrm{rnd}]$ to $[u^*, q^*, \pi^*]$, the optimal meta-state.

This would ensure that it is possible to decrease the temperature and increase the discount smoothly so as to maintain equilibrium at each intermediate point. The experiments in Chapter 9 even suggest that C can be chosen independent of problem, and it includes some "nice" curves such as $\gamma = \tanh\mu$.

Remark 8.6.1 There will be a further complication when parameterisation is involved. The convergence of neural network learning process in a non-stationary environment is generally unknown at the moment. The simulated annealing learning rule (§6.6) offers a possibility to combat the changes in the environment: the μ should not be increased to infinity, but remain bounded by a constant depending on the changing speed of the environment. In the extreme, if the environment is changing infinitely fast such that there is no statistical regularity, μ should be kept 0 for ever.

Chapter 9

Basic Adaptive Computers

9.1 Introduction

In this chapter we study basic adaptive computer (BAC) which is an implementation of MDP with neural networks. While the previous chapter is mathematical, the current one is computational. Recall the concepts of "enumerable set" and "combinatorially enumerable" from Chapter 1. Our basic assumption in this chapter is: (1) The state space X is a subset of $\mathbb{R}^n$ where n is enumerable; (2) The action space A is combinatorially enumerable; (3) The transitional mapping τ is continuous with respect to the state space (If the state space is a discrete subset of $\mathbb{R}^n$, the transitional probability is automatically continuous.); (4) The reward function R is bounded. The main improvement over previous methods is the ability to obtain global optimum in a combinatorial environment.

BAC can have two basic modules, decision module (D) and evaluation module (E1), and two optional modules, prediction module (P) and action evaluation module (E2). The modules (D) and (E) were present in [Sam59, BSA83], the modules (D), (E1) and (E2) appear in [Wat89], and the module (D), (E1) and (P) appear in [Sut90].

9.2 General Description of BAC

In this chapter our description will be implementation oriented, and not as formal as the previous chapter.

Assumption 9.2.1 To avoid technical difficulties, we assume that the feasible action set is the same for each state, so $E = X \times A$, thus we can identify A with $\mathbb{N}_n$.

Notation. Given MDP (8.2.1), for any random object f, define $\widehat{f} := \langle f \rangle_x$, $\widetilde{f} := \langle f \rangle_{x,a}$. Denote $q^\mu := q\mu - \log P_\pi$.

A *basic adaptive computer* (BAC) is a computing system $[\pi, e, \delta, \mu, \gamma'']$ with a modular structure where each *module* is no more complicated than a trainable information processor (TIP) and some fixed information processors (FIP), which, in an environment

$[\tau, r, \gamma']$, implements an adaptive Markov decision process (AMDP)

$$a = \pi(x), \qquad x_+ = \tau(x, a), \qquad r = r(x, a), \qquad e = e(x, a), \tag{9.2.1}$$
$$\gamma' = \gamma'(x), \qquad \gamma'' = \tanh\mu, \qquad \gamma = \gamma'\gamma'' \tag{9.2.2}$$
$$[\pi_+, e_+, \mu_+, \gamma_+] = \delta(x, a, x_+, r, \pi, e, \mu, \gamma). \tag{9.2.3}$$

where $e = [e_1, e_2]$ and

(9.2.4)
$$\Delta P_\pi : \ \max e_2^\mu := \langle e_2\mu - \log P_\pi \rangle_x , \qquad \Delta e : \ \max \epsilon := -\frac{1}{2}\left((\widehat{e}_2 - e_1)^2 + (\widetilde{r} + \widetilde{e}_{1+} - e_2)^2 \right).$$

A schematic diagram of BAC was given in Figure 1.1. Usually only one of e_1 and e_2 is implemented directly, the other is computed indirectly, by $e_2 = \widetilde{r} + \widetilde{e}_{1+}$ or $e_1 = \widehat{e}_2$. We shall only consider the structure with e_1 directly implemented.

A BAC can have two basic modules, *decision module* (D) and *evaluation module* (E1), and two optional modules, *prediction module* (P) and *action evaluation module* (E2). The module (D) implements the policy $\pi \in [X \to A)$, the module (E1) implements the evaluation $e_1 \in [F \to \mathbb{R})$ which approximates u_π, the module (E2) implements the evaluation $e_2 \in [G \to \mathbb{R})$ which approximates q_π, and the module (P) implements an approximation to $\tau \in [G \to X)$.

Denote the weights in (D) and (E) by w^D and w^E. The gradients are

$$\frac{\partial \widehat{q}^\mu}{\partial w^D} = \left\langle \frac{\partial T}{\partial w^D}, q^\mu \right\rangle_x , \tag{9.2.5}$$
$$\frac{\partial \epsilon}{\partial w^E} = (\widehat{q} - e)\frac{\partial e}{\partial w^E}. \tag{9.2.6}$$

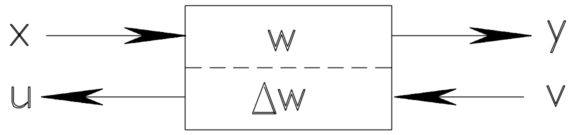

Figure 9.1: A neural network as a trainable information processor (TIP)

It works in the following cycle: (1) Receive input x; (2) Emit output y; (3) Receive auxiliary input v; (4) Update weight w, emit auxiliary output u.

The TIPs available to us are neural networks, which from now on can be viewed as "black-boxes" with desired properties (Figure 9.1). Following Chapter 4–7, we have at our disposal two general types of TIPs: SQFB-NN and DCFB-NN. A SQFB-NN $[\varphi, \delta]$ realises the process

$$y = \varphi_w(x), \qquad \Delta w = \delta_w(v)$$

such that

$$\langle \Delta w^T \rangle = \left\langle \left\langle \frac{\partial T}{\partial w}, v^\mu \right\rangle_x \right\rangle.$$

A DCFB-NN $[\varphi, \delta]$ realises the process

$$y = \varphi_w(x), \qquad \Delta w = \delta_w(v)$$

such that

$$\langle \Delta w^T \rangle = \left\langle v^T \frac{\partial y}{\partial w} \right\rangle .$$

Remark 9.2.1 It is possible to cascade these networks in the fashion of SQ-SQ, DC-SQ, or DC-DC, with handy learning rules. This will be useful when we try to construct AC other than BAC. It is also possible to cascade SQ-DC, but this is quite complicated and it best used in some more elaborate way (§10.4.2).

To implement NN in a modular structure such as AC means to specify the connection of x, y, u, and v with other parts of the system. The two modules (D) and (E) are specified as follows

1. Decision module. $x^D = x$, $y^D = a$, $v^D = q$, and

$$\left\langle (\Delta w^D)^T \right\rangle = \left\langle \left\langle \frac{\partial T}{\partial w^D}, q^\mu \right\rangle_x \right\rangle . \tag{9.2.7}$$

2. Evaluation module. $x^E = x$, $y^E = e$, $v^E = \widehat{q} - e$, and

$$\left\langle (\Delta w^E)^T \right\rangle = \left\langle (\widehat{q} - e) \frac{\partial e}{\partial w^E} \right\rangle . \tag{9.2.8}$$

Here we have omitted their internal states. In the actual implementation, weight updating should always follow the same input/output computation which caused this updating. More specifically, the TIPs should be described as

$$r = \zeta_w(x), \qquad y = \varphi_w(x), \qquad w_+ = \delta(v, w, r), \qquad u = \kappa(v, w, r). \tag{9.2.9}$$

The variable $\widehat{q}$ used in (9.2.8) can be approximated in many ways, such as $U_\pi(e_2)$, $R_\pi(e_1)$, $\bar{q}$ or $\widetilde{q}$, where $\bar{q}$ is defined as the average of $r + e_+$ over several samples of x_+ and $\widetilde{q}$ is the sample $r + e_+$. The ensemble average $R_\pi(e_1)$ is used in the experiment in the following sections. The "temporal difference" method (TD) [Sut88] can be used in conjunction with $\widetilde{q}$.

9.3 Implementing the Decision Module by DC-NN

When the action space A can be enumerated, the decision module can be constructed with DC-NN. This has the advantage that the convergence is faster than that based on SQ-NN because the statistics, not just samples, of probability distributions are used.

The decision module is composed of a *decision network* $[\varphi, \delta]$ and a *random action selector* ι. Given input x, it computes $T = \varphi_w(x)$, $a = \iota(T)$, such that $P_\iota \sim \exp T$. Then it receives auxiliary input $v = q(x, a)$, and modifies weight $w_+ = \delta(v, w)$. The policy is therefore $\pi_w := \iota(\varphi_w)$. This can be alternatively interpreted as having a neural network whose last layer is stochastic, with probability distribution restricted to the n unit vectors $\mathbf{e}_i$. See, for example, [YG90, Lin92].

Now we study what the auxiliary input $\widetilde{v}$ to the decision network should be. We want

$$\Delta w^T = \alpha \left\langle \widetilde{v}^T \frac{\partial T}{\partial w} \right\rangle_x = \alpha \frac{\widehat{\partial q^\mu}}{\partial w}.$$

Since

$$\frac{\widehat{\partial q^\mu}}{\partial w} = \left\langle \frac{\partial T}{\partial w}, q^\mu \right\rangle_x = \sum_{ij} P_i(\delta_{ij} - P_j)\frac{\partial T_j}{\partial w}(q_i^\mu), \tag{9.3.1}$$

we have the following three methods for providing $\widetilde{v}$.

$$\widetilde{v}_j = P_j(q_j^\mu - \widehat{q^\mu}) \tag{9.3.2}$$

$$\widetilde{v}_j = \widetilde{\delta}_j(\widetilde{q^\mu} - \widehat{q^\mu}) \tag{9.3.3}$$

$$\widetilde{v}_j = (\widetilde{\delta}_j - P_j)\widetilde{q^\mu} \tag{9.3.4}$$

Theorem 9.3.1 *Each of (9.3.2), (9.3.3) and (9.3.4) implies*

$$\left\langle \widetilde{v}^T \frac{\partial T}{\partial w} \right\rangle_x = \left\langle \frac{\widetilde{\partial T}}{\partial w}, \widetilde{q^\mu} \right\rangle_x . \tag{9.3.5}$$

Proof. Proof by direct calculation.

(9.3.2) $\Longrightarrow$

$$\left\langle \widetilde{v}^T \frac{\partial T}{\partial w} \right\rangle_x = \left\langle \sum_j P_j(q_j^\mu - \widehat{q^\mu})\frac{\partial T_j}{\partial w} \right\rangle_x = \left\langle \left\langle (\widetilde{q^\mu} - \widehat{q^\mu})\frac{\widetilde{\partial T}}{\partial w} \right\rangle_x \right\rangle_x = \left\langle \frac{\widetilde{\partial T}}{\partial w}, \widetilde{q^\mu} \right\rangle_x .$$

(9.3.3) $\Longrightarrow$

$$\left\langle \widetilde{v}^T \frac{\partial T}{\partial w} \right\rangle_x = \left\langle \sum_j \delta_j(\widetilde{q^\mu} - \widehat{q^\mu})\frac{\partial T_j}{\partial w} \right\rangle_x = \left\langle (\widetilde{q^\mu} - \widehat{q^\mu})\frac{\widetilde{\partial T}}{\partial w} \right\rangle_x = \left\langle \frac{\widetilde{\partial T}}{\partial w}, \widetilde{q^\mu} \right\rangle_x .$$

(9.3.4) $\Longrightarrow$

$$\left\langle \widetilde{v}^T \frac{\partial T}{\partial w} \right\rangle_x = \left\langle \sum_j (\widetilde{\delta}_j - P_j)\widetilde{q^\mu}\frac{\partial T_j}{\partial w} \right\rangle_x = \left\langle \widetilde{q^\mu}\left(\frac{\widetilde{\partial T}}{\partial w} - \frac{\widehat{\partial T}}{\partial w}\right) \right\rangle_x = \left\langle \frac{\widetilde{\partial T}}{\partial w}, \widetilde{q^\mu} \right\rangle_x .$$

□

Method (9.3.3) and (9.3.4) define v to be a random variable depending on the chosen action. They are therefore stochastic gradient descent on the free energy landscape $\widehat{q^\mu}$. So the learning rate α must be sufficiently small for the averaging to be meaningful. Method (9.3.2) is an accurate gradient descent on the free energy landscape. Its drawback is that it requires enumeration of evaluations of all the feasible choices in order to compute q_j^μ. Since our interest is mainly on how simulated annealing compare with direct gradient following methods, we mainly use Method (9.3.2) in the tests to be described in the following section.

We have also performed limited experiments with methods (9.3.4) and (9.3.3). The only noticeable difference is that they are slower than (9.3.2). They do not alter the fact whether the BAC converge to global optimum. In fact, as long as the speed is slow enough, all of them converge to global optimum. So the results to be described in the next section are representative of all these three variants.

9.4 Two Test Problems

We shall present numerical results on two test problems, which are chosen because on the one hand, they are simple enough to admit an intuitive definition of the optimality of solutions so that we can compare experiments with theoretical predictions, and on the other, they have features which represent a class of problems more general than those treated in the literature so far [BSA83, Sut90, Lin92, Wat89, WB91, Tes92]. Variations of these test problems can serve as new test problems for basic adaptive computers (BAC) with more modules. Both test problems can be of arbitrary sizes, but to avoid distraction from our main concern we describe them as though their sizes were fixed.

9.4.1 The "counting game"

This is a two persons zero-sum game. The state is described by two integers m_C and m_T, called the current number and the target number, respectively. At the start of each play, $m_C = 0$, and $m_T = \mathrm{rnd}\,\{7, 8, 9, 10\}$. One player is randomly chosen as the first to play. The two players alternatively choose an integer $a \in A = \{1, 2, 3\}$ and set the new current number to $m'_C = m_C + a$, until one player achieves $m'_C \geq m_T$ and wins the game.

The optimal strategy for this game as defined by the game theory is a deterministic strategy. It is best described by the set of strategic points $S(m_T) := m_T - 4\mathbb{N} = \{i = m_T - 4k, k \in \mathbb{N}\}$. The "optimal action" is to chose a so that $m'_C \in S(m_T)$. If player A has done so, player B may be prevented from getting into $S(m_T)$ until A wins.

Proposition 9.4.1

$$\forall m_T : \forall m_c \notin S(m_T) : \exists! a \in A : m_c + a \in S(m_T), \tag{9.4.1}$$

$$\forall m_c \in S(m_T) : \forall a \in A : m_c + a \notin S(m_T), \tag{9.4.2}$$

It can be seen that this strategy consists of a series of XOR problems. For example, if the $m_T = 9$ and $m_C = 4$, then $m'_C = 5 \in S(m_T)$ is the next strategic point. But when $m_T = 7$, the next strategic point for $m_C = 4$ is $m'_C = 7$. The choice $m'_C = 5$ will lend the opponent an easy win. The XOR structure of this game makes the simulated annealing necessary. This strategy can be explicitly expressed as

$$[0, 3, \{4, 5, 6\}, 7], \quad [0, \{1, 2, 3\}, 4, \{5, 6, 7\}, 8],$$
$$[0, 1, \{2, 3, 4\}, 5, \{6, 7, 8\}, 9], \quad [0, 2, \{3, 4, 5\}, 6, \{7, 8, 9\}, 10].$$

This means, for example, when the target number is 9, the first choice should be 1, and if the opponent has chosen 1, the agent should choose 2,3, or 4 with equal probability, and if the opponent has chosen any of 2, 3 or 4, the agent should choose 5, etc., (the third chain above).

Since our learning method use Boltzmann distribution with non-zero temperature, the equilibrium strategies will depend on the temperature. The optimal strategies of games theory corresponds to the zero temperature unit discount equilibrium strategy. The evaluation of all the states under different temperatures are shown in Figure 9.3 and Figure 9.4. The scaling of the evaluation is explained in §9.5.

There are at least two ways to put this game into the framework of Markov decision process. Either we can consider the two player as having the same mentality, or different. For the first assumption, the prediction module can be replaced by the decision module, by exchange the representation of the situation and evaluation of the two players after each ply, ie. half move, and set $-\gamma''$ as discount. More intuitively, to predict what the opponent will do, think what the player himself will do in that situation, and use the zero-sum property of the game. This is a quite old technique for testing such learning methods

[Sam59]. For the second assumption, a prediction module would be useful, although not necessary, for updating the evaluation. Under either assumption, the environment is not stationary. Therefore this game is slightly more general than our theory allows. Under the first assumption, the environmental change is not faster than the change in the adaptive computer. We call this *quasi stationary*. We do not give a formal definition of this concept here. The intuitive criterion is that the theory for MDP in stationary environment can be used without much discrepancy if equilibrium strategies are used.

9.4.2 The "moving point game"

This is a game of a single player. The state is represented by a rectangular array of possibly marked squares and a moving point. At the start of the game, all the squares are unmarked, and a point randomly placed in a square. The player moves the point along any of the four directions into a neighbouring square or out of the array. Each visited square is marked. If the player tries to move the point into a visited square or move out of the array, the game is over. The total number of the squares visited is the score it has achieved.

We tested the BAC with a 4×4 array. It is difficult to anticipate the best strategy for this game, as there are about 10^4 or more possible states and about 10^6 or more possible histories [1]. We had no idea before the test what strategy the BAC would use. The strategy the BAC invented turned out to be surprisingly simple. If a deterministic enumerative method is used, the game would be much more difficult than a maze, since all the visited squares becomes walls. This game is substantially more difficult than the maze problem played by "Dyna" in [Sut90].

9.5 Scaling and Speed

There have been numerous tests and applications of connectionist models reported in the literature. The one big problem facing some one who tries to compare these results is that all the different tests use measures of performance particular to the test problems, rather than the methods. Therefore, comparison of results for two methods is only possible if they have been tested on exactly the same problem. Even so the comparison does not necessarily reveal anything significant outside the particular test problems they are based on. One such arbitrariness occur in the measure of performance. Various measures which are only meaningful to the (artificial) problems have been used, such as surviving time, error rate, words recognised, just to mention a few (eg., [BSA83, Sut90, Lin92, WB91, Tes92]).

Example 9.5.1 A chess player should learn exactly the same strategy in the following two settings: (1) win= 1, lose= -1, tie=0; (2) win= 100, lose= 0, tie=50. The absolute value should only be interesting if the player is also required to perform activities outside chess, but then the scores are relative again in the larger context.

For methods which uses gradients, this also affects the definition of step lengths in various updating formulae. It is desirable to set a normalising convention which is only depends on the learning methods instead of the particular learning problems. This would facilitate the comparison of different learning methods, or different implementation of

[1] This is a very rough estimation. Note that $2^{16} = 6.5 \times 10^4$, $16! = 2 \times 10^{13}$.

the same learning method. It should also be a basic requirement for general learning method to be independent of the normalisations as in Example 9.5.1. This trivially obvious *requirement* is not at all a trivial *task*. Here we shall define a normalising convention which is almost invariant across a wide range of problems.

Notation. Denote $m := |X|$, $n := |A|$, $\nu = \log n$. Define $\langle q \rangle := \langle \langle q \rangle\, x \rangle$, $\sigma_q^2 := \langle \langle q, q \rangle_x \rangle$, $T := \langle \langle T \rangle_x \rangle$, $C := \sigma_q^2 \mu^2$.

We make the following requirements for the normalisation convention: (1) The normalised utility function should be an affine transform of the original one: (2) The normalised utility function should be invariant if the reward is scaled by a constant; (3) The following quantities should be near unity: $\left.(\langle q \rangle / \nu)\right|_{\mu=\infty}$, $\left.(\sigma_q^2/\nu)\right|_{\mu=0}$, $\left.(T/\nu\mu)\right|_{\mu=\infty}$, C_c/ν and μ_c.

Definition 9.5.1 Let q be the action utility function, the the *normalised action utility function* q' is defined as $q' = aq + b$, where a, b are constants, such that for $\pi = \mathrm{rnd}$:

$$\langle q' \rangle = 0, \qquad \sigma_{q'}^2 := \nu,$$

In other words,

$$\sum_x p(x) \sum_a q'(x,a)/n = 0, \qquad \sum_x p(x) \sum_a q'(x,a)^2/n = \nu,$$

where $p(x)$ is the probability of state x occurring under current policy. The *normalised utility function* is defined as $u' = au + b$.

In our actual implementations of the adaptive computer, and in all the tables and graphs in this chapter, the above normalisation convention is applied.

Remark 9.5.1 These normalised learning curves are much less impressive than those showing performance based on measure specific to the game, but they convey much more information about the *learning methods*. In particular, they allow comparison of the performance of the same method on quite different types of games.

9.5.1 Equilibrium strategy of the counting game

The equilibrium of the counting game can be computed numerically, at least for small $|X|$ and $|A|$. This provides one example to test the normalisation convention discussed earlier. We did not compute the ideal learning curves for the moving point game due to its huge state space.

Since the equilibrium strategies are described by the difference $m_T - m_C$, they can be illustrated by the corresponding evaluations for the equilibrium strategies under different temperature. We have test various limits of target number range (up to 400) and various sizes of action space (from 2 to target number). The given convention always yield the critical parameters in a surprisingly uniform way. Some of these results are shown in Table 9.1 and Figure 9.2. The results with the other parameters are almost identical. These normalisation conventions will be used in all the tests. The ideal strategy of the counting game is shown in Figure 9.3 and Figure 9.4.

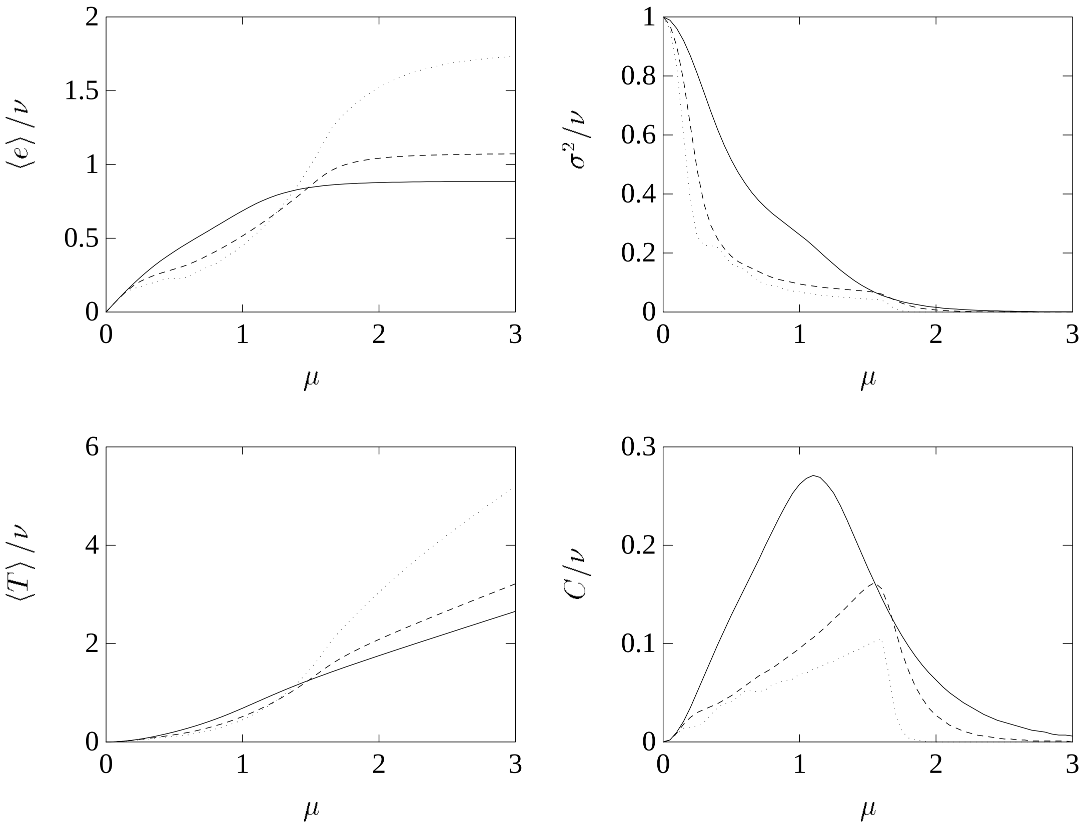

Figure 9.2: The ideal learning curves for the counting game.

Solid curve: $m = 10$, $n = 3$; dashed curve: $m = 50$, $n = 5$; dotted curve: $m = 400$, $n = 30$.

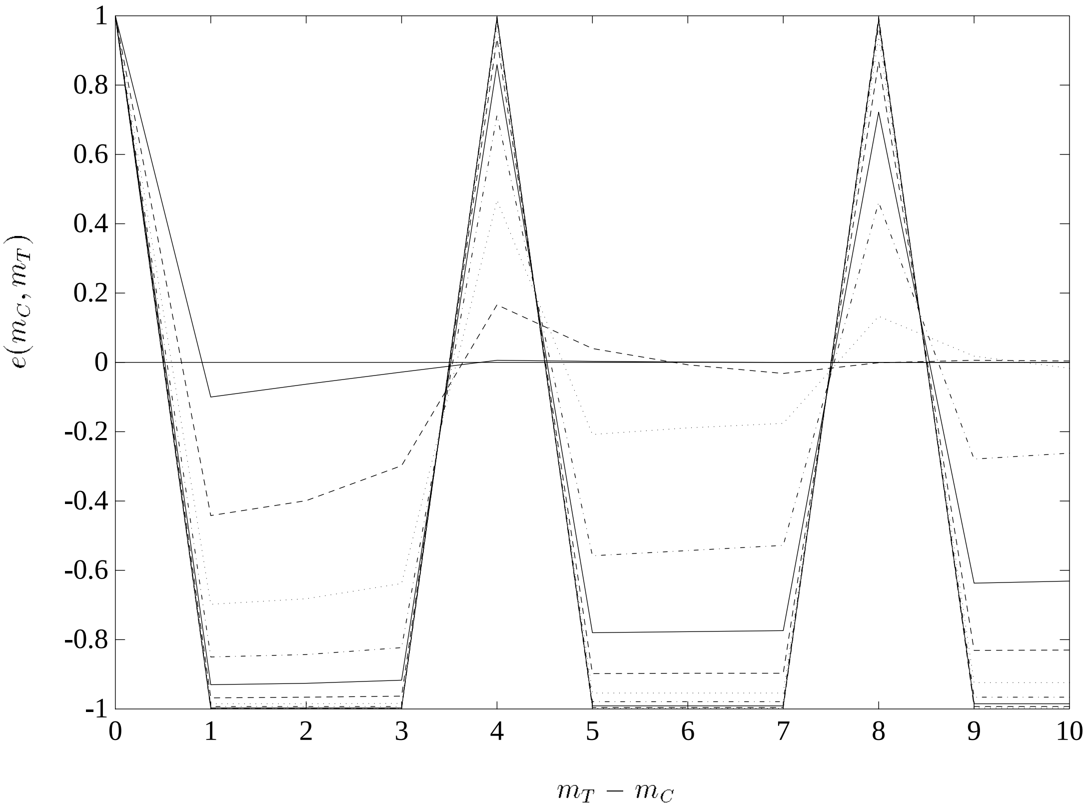

Figure 9.3: Normalised evaluation corresponding to equilibrium strategies of the counting game for $m_T \in \{7, 8, 9, 10\}$, $n = 3$

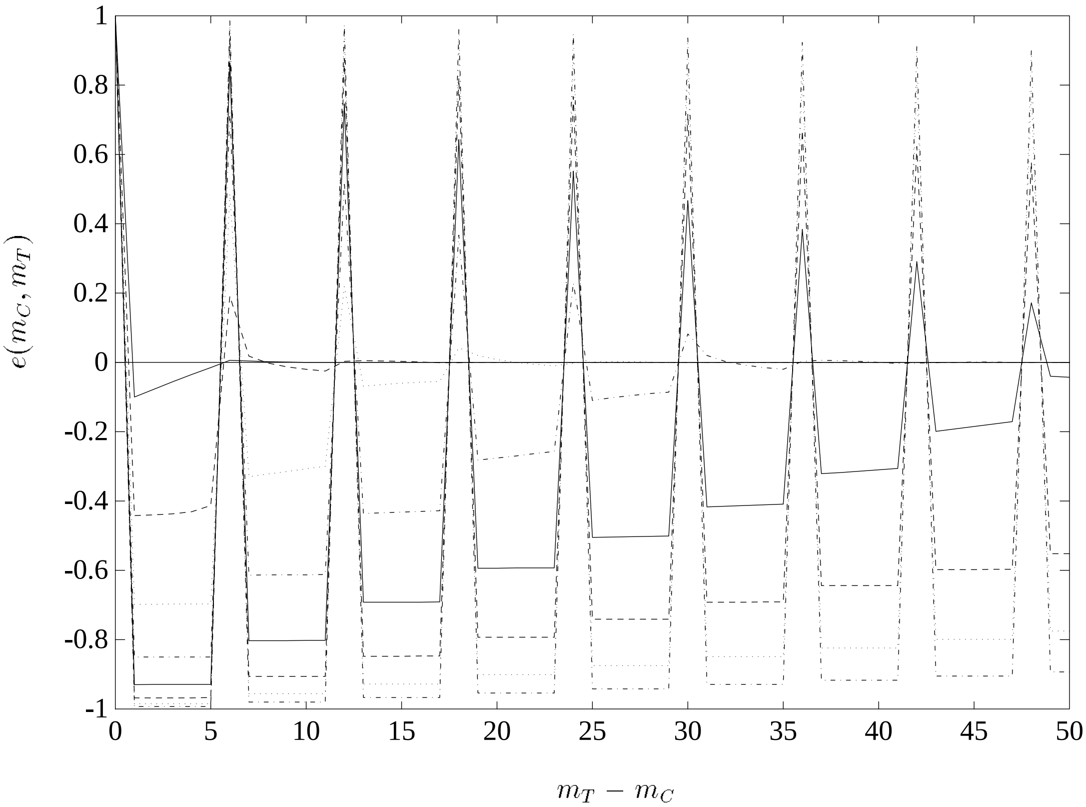

Figure 9.4: Normalised evaluation corresponding to equilibrium strategies of the counting game for $m_T \in \{46, \ldots, 50\}$, $n = 5$

$m = 10$					
n	$(\langle q\rangle/\nu)\vert_{\mu=\infty}$	$(\sigma_q^2/\nu)\vert_{\mu=0}$	$(T/\nu\mu)\vert_{\mu=\infty}$	C_c/ν	μ_c
3	0.885	1.000	0.885	0.271	1.100
4	0.903	1.000	0.903	0.300	1.000
5	0.912	1.000	0.912	0.304	0.950
6	0.982	1.000	0.982	0.338	0.900
$m = 50$					
n	$(\langle q\rangle/\nu)\vert_{\mu=\infty}$	$(\sigma_q^2/\nu)\vert_{\mu=0}$	$(T/\nu\mu)\vert_{\mu=\infty}$	C_c/ν	μ_c
5	1.072	1.000	1.072	0.162	1.550
10	1.202	1.000	1.202	0.213	1.200
15	1.178	1.000	1.178	0.281	1.050
20	1.097	1.000	1.097	0.333	0.950
25	1.004	1.000	1.004	0.368	0.950
30	1.020	1.000	1.020	0.348	0.950
$m = 400$					
n	$(\langle q\rangle/\nu)\vert_{\mu=\infty}$	$(\sigma_q^2/\nu)\vert_{\mu=0}$	$(T/\nu\mu)\vert_{\mu=\infty}$	C_c/ν	μ_c
30	1.733	1.000	1.733	0.105	1.600
100	1.356	1.000	1.356	0.262	1.050
170	1.001	1.000	1.001	0.449	1.000

Table 9.1: Normalised critical parameters in ideal learning for the counting game.

9.5.2 Speed of learning and averaging schemes

The various speed α, β, γ, and ρ take the same form as in §7.3.1, except that ϕ should also depend on the expected lifetime of a play. This is because in most problems the start and end of a play have distinctive characteristics, and learning methods should be gentle enough to actually experience several full circles of the game before changing behaviour too much. For example, in the "pole balancing problem" [BSA83], the only non-zero reward comes after a play terminates, while the occurrence of such events are pushed to more distant future as the policy improves.

We use

$$\phi = \frac{\sigma_q^2}{\nu L} \tag{9.5.1}$$

as the scaling on α, β, γ, and ρ, where L is expected (un-discounted) lifetime of a play. It is obvious that this reduce to (7.4.9) for $L = 1$. With reference to Proposition 8.4.6, we have the following two expressions

$$1 - 1/L = \gamma'' p, \qquad L = 1 + p\gamma'' L, \tag{9.5.2}$$

which give the following two methods of updating L:

(9.5.3)
$$\left[1 - \frac{1}{L}\right]_+ = \left[1 - \frac{1}{L}\right] + \rho\left(\gamma'' - \left[1 - \frac{1}{L}\right]\right), \qquad [L]_+ = [L] + \rho\,(1 + \gamma''[L] - [L]),$$

We mainly used the first method in our experiments.

9.6 Set-up of Experiments

The most important feature of our numerical experiments is that we use exactly the same programs to solve the two different problems which are of quite different nature. As far as we know, this has not been done before.

The main body of the simulation is written as FORTRAN77 subroutines and implemented on both a IBM 3081 mainframe machine with VMS and a SUN Sparc station under UNIX. The computation is performed with double precision arithmetic (real*8, integer*4), with calls to the NAG Fortran library for random number generators subroutines (G05 routines). A run of the simulation consists of a sequence of trials (plays), each of which consists of a sequence of actions (moves).

The simulation programs can be divided into three parts: (1) BAC structure and main control programs, (2) NN subroutines, and (3) game subroutines. The only information passing from the games subroutines are state x, action a, reward r, and discount γ', where x is a real vector, a is an integer, r is a real number and γ' is a logical variable (since in our test problems $\gamma' = 1$ for non-terminal states). The main control programs define the BAC structure, and is used for transmitting information among the modules and the games subroutines, and for printing out intermediate and final results.

The simulation programs can use two, three or four modules ((D), (E1) plus (E2) and/or (P)). Our main experiments are performed with two modules only. Here we describe the other two modules briefly. The (P) module is quite independent of all the other modules, since it models the environment which in our test are either stationary or quasi-stationary. The task of (P) is to predict x_+ given x, a, so it can be constructed by SQ-NN, (or DC-NN with a random selector if the states are enumerable,) with either ME or ML learning rules. We only tested implementation with DC-NN, with ML learning rule (almost identical to the implementation of (D)). In the tests it learns much faster than both (D) and (E). After it converges it behaves almost identical to the environment, so that the other modules using (P) acts as if connected to the environment directly.

The effect of (E2) is much more interesting. It was hoped that approximating $q(x,a)$ directly instead of averaging over samples of $r(x,a) + u(x_+)$ would provide better evaluation for the (D) module. However, in our tests this is never the case. The approximation e_2 is always slower and less accurate than e_1.

For these reasons in the next section we shall concentrate on the BAC structure with (D) and (E). We use the DCF-NNs to implement both (D) and (E) for its simplicity. We did not experiment on combining SQ-NN with BAC to solve problems with combinatorial action space, due to the estimation that any test problem of this kind would require huge amount of computing time. Small test problems would not tell much about the method.

9.7 Numerical Simulations

9.7.1 The "counting game"

Description of the test

We do not distinguish between row and column vectors here. The state is represented by a vector $x \in X = [0,1]^{22}$, such that $x = [\mathbf{e}_{m_C+1}, \mathbf{e}_{m_T+1}]$, where $\mathbf{e}_k$ is the kth unit vector. The action is represented by $a \in A = \{1,2,3\}$. The reward is $r = 1$ for the winner and is $r = -1$ for the loser at the end of the game, and it is $r = 0$ at all the intermediate steps.

The two players are simulated together by assuming that they have same mentality.

The BAC consists of a decision module and an evaluation module, each composed of a simple-layered DCF-NN, with one hidden layer of eight neurons. That means they have the structure of 22-8-3 and 22-8-1, respectively. The hidden layer size is chosen so that the simulation speed is reasonably high. Larger or smaller size (within a factor of 2) do not have appreciable effect. Too large a network makes the simulation unnecessarily slow. We also tested simple-layered networks with two hidden layers of eight neurons, with no discernible difference in the performance, except that simulation is slower. These choices are by no means optimal.

Experiments leading to SA learning rule

We first tested this problem before developing the theory on SA learning rules, using the GF learning rule which is almost identical to that of [Sut90]. To our surprise the BAC almost never converged to the desired global optimum, despite the fact that the global optimum is representable (ie., if the weights are hand set to somewhere near the global optimum, it converges very fast automatically). Several techniques most frequently cited in the literature were tested, including "momentum term" and random initial weights [RHW86], "weight decay" [HS86] and added noise [AHS85], none of them completely overcome this problem.

The mechanism by which the network converges to local optima is the following: since the network is fully connected, whenever one event happens, the evaluation and policy of any other events will be changed as well. Since the problem contains several XOR sub-problems, it often happens that before an "action" has had a chance to demonstrate its "usefulness", its probability has already been reduced to virtually zero. The evaluation module will then evaluate each state as if the option of taking such an action does not exists. Then the decision module will happily converge to the deterministic policy, since $P_i(q_i - \widehat{q}) = 0$ can be zero even when $q_i - \widehat{q}$ is large. For example, without SA the BAC always converges to a strategy with $[0, 3, 4, 5, \{6, 7, 8\}, 9]$ in place of $[0, 1, \{2, 3, 4\}, 5, \{6, 7, 8\}, 9]$. It is pointless, as a general learning rule, to solve this problem by increasing the number of neurons while restricting the connectivity of the network, since these techniques simply work by creating a network so large which remembers everything in tabular form.

Therefore it became clear that to prevent the BAC from converging to local optima it is necessary to restraint the BAC from becoming too deterministic. The last two of the above four techniques, weight decay and added noise, can be adapted to improve the learning quality. However, we found that it is not enough to use a fixed amount of decay or noise, the only effective way is to reduce them slowly in the learning process. This dramatically improves the performance, so that it became more common for the system for converge to global optimum than local optima. Weight decay is the most efficient among these four techniques.

On the other hand, we found that the learned evaluation function and the policy were still quite skewed. These techniques did not achieve globally uniform results. We then tried a quite simple substitute to the weight decay: using $P_j(q_j - \widehat{q}) + \theta(\frac{1}{n} - P_j)$ in place of $P_j(q_j - \widehat{q})$, which gave still better results. Then we tried another substitute, using $P_j(q_j - \widehat{q}) + \theta(\widehat{T} - T_j)$, which further improved the learning performance. This is effectively the SA learning rule.

We have seen in §7.5 that if θ is set constantly to zero, the SQFB-NN (including SQB-

NN) always converge to local optima for some simple problems. This also happens to the BAC without SA: the policy converges to such that at some state it will happen that the probability of choosing the optimal action is virtually zero, so that this alternative action becomes "invisible" to the evaluation network. On the other hand, as long as the speed $\theta \to 0$ is slow enough, convergence to global optimum is always achieved. This shows that simulated annealing is a necessity for such a construction to work.

Results and analysis of the SA learning rule

The learning process can be assessed by the accuracy of the evaluation of different states. The BAC learns to evaluate the end-game states very fast. The strategic points preceding the target number then become its objective as well. The learning process shows a pattern of "reversion" of evaluation propagating back from the end-game numbers to the starting numbers, as the BAC "realises" which numbers are strategic for the different targets. Whenever such a reversion occurs, the BAC seems to have lost a large amount of its learned skills. But after a brief period, a new pattern of behavior emerges in which a new strategic point is found. This quickly enhances the learned skills, so that everything is better than prior to the reversion. When the last strategic point—the one nearest to the starting point—is found, the behavior and the evaluation converges very quickly to that corresponding to the optimal strategy. A typical trial history after 2000 trials in a typical run is shown in Figure 9.5.

1st player	–	2	–	6	–	10
2nd player	0	–	4	–	7	–
Evaluation	−.98	.98	−.97	.99	−1.00	1.00
probability of playing each move						
move 1	.01	.32	.01	.36	.01	–
move 2	.99	.37	.99	.33	.00	–
move 3	.00	.31	.00	.32	.99	–

Figure 9.5: History of a typical trial of the "counting game" after 2000 trials in a typical run.
Top three rows show the current player, the current number and the evaluation of the current state by the current player. Bottom three rows show the probability of next move.

After convergence, the BAC can play the best move if it is in a winning position. It can also play the best possible defending move if it is in a losing position, namely, using a mixed strategy. It is obvious that the network size is not large enough to memorize all the possible states (If the target number is either 7, 8, 9 or 10, then there are 33 possible states). The fact that the BAC chooses the optimal moves with almost one hundred percent probability means that it must have made some generalizations.

Other possibilities

We have also tested the same game with different sizes where the target ranges from 3 up to 20. Convergence to global optimum is always achieved with SA. When the target is larger than 4 or 5, various local optima are encountered if SA is not used.

If the representation of the game is changed so that the target number and the current number are represented using the same vector, the number of hidden units required is much larger, as they have the new task to compare the order of two numbers. If there are enough number of hidden units to represent each possible pairing of the current number and the target number, then the BAC learns comfortably. If the number of hidden units is not large enough, the problem becomes much harder. In principle, the BAC should invent some digital representation like quaternary or binary representations which would enable it to deal with any large numbers. However, our test experiments do not show any sign of this. One possible reason is that the digital representation is only "economic" for dealing with sufficiently large numbers.

9.7.2 The "moving point game"

Description of the test

The state is represented as a vector $x = [x_M, x_C] \in X = \{0,1\}^{32}$, where $x_M \in \{0,1\}^{16}$ records the visited squares (0: unmarked, 1: marked), and $x_C \in \{0,1\}^{16}$ records the position of the current square. The action is represented by $a \in A = \{1,2,3,4\}$, corresponding to the four directions. The reward $r = 1$ at each step.

In the test, both the decision module and the evaluation module use simple-layered DCF-NN with one hidden layer of 40 neurons. That is, they have 32-40-4 and 32-40-1 structures, respectively. We also tested simple-layered networks with two layers of hidden units, and found no obvious advantage.

Results and analysis

A temporal sampling of a typical run of the moving point game is given in Figure 9.6. A "cross section" of the same run just after 4000 plays is shown in Figure 9.7. Figure 9.8 and Figure 9.9 show another run, with a faster learning speed.

In the first few trials of each run, the BAC usually scores average between 2 and 3, because it often chooses to go an invalid direction. It apparently learned the rules after about 800 trials, but not the best strategy. It discovered a good strategy after about 2000 trials, which can be expressed as "first go to the edge, then circle along the edges, then go to the centre". This strategy enables the BAC to score around 13 to 14 in each trial in average after convergence.

It should be noted that the BAC does not learn the evaluation quite as well as it learns the good strategy. The evaluations near the end of each game are especially inaccurate. The quality of the learnt policy can be explained if one notices that this strategy is essentially a local one. It is only necessary to know the immediate neighbours to carry out this strategy. The evaluation, however, needs global information such as connectedness. Therefore in such situations the adaptive methods are especially preferable to analytical methods.

9.7.3 Discussion of the simulation results

These test results are quite remarkable in the following aspects.

(1) The rules of the games are never explicitly told to the BAC. Our simulation program consists of subroutines which are separately written for the DCF-NN, the structure

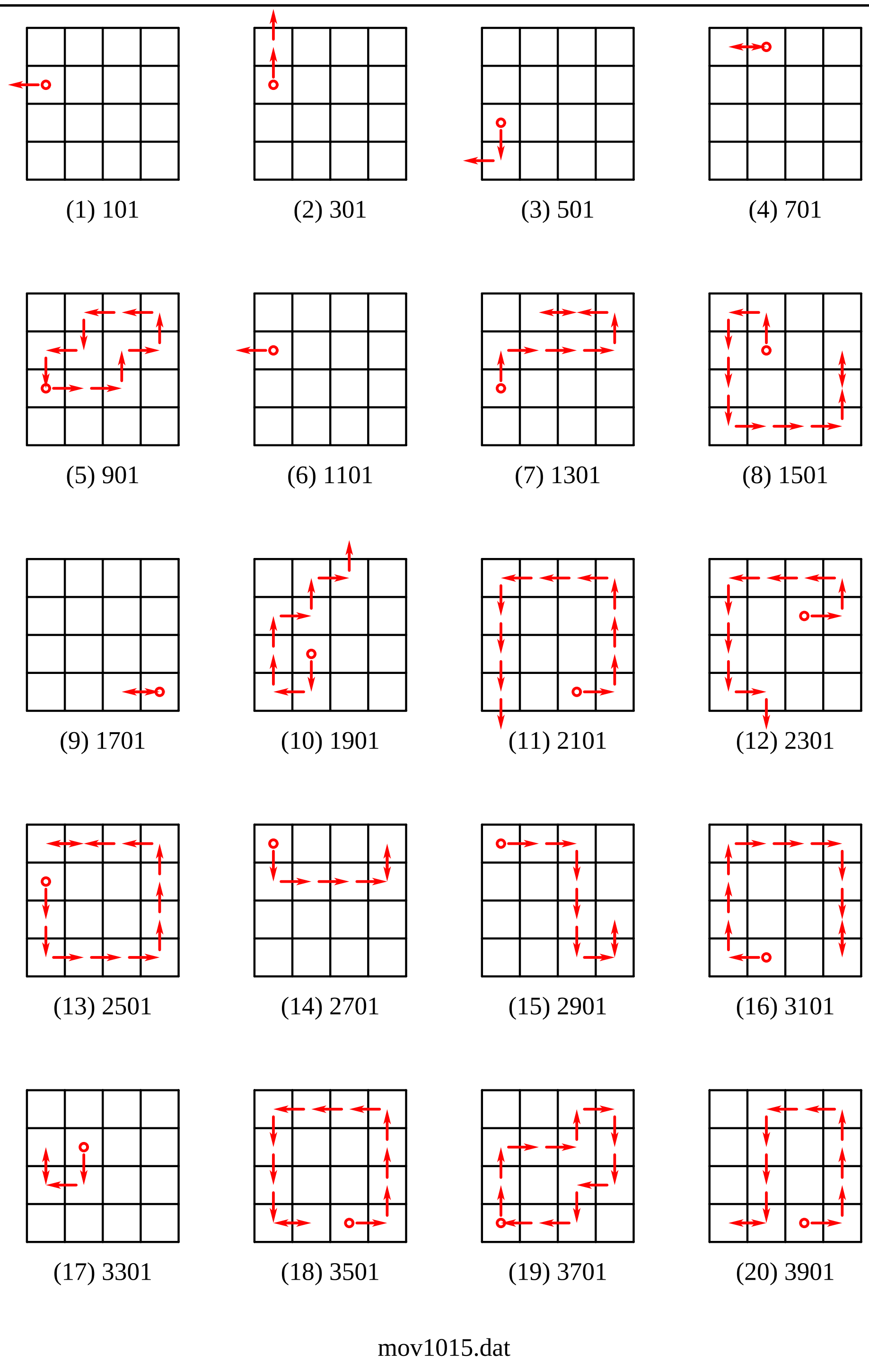

Figure 9.6: Samples from the history of a run of BAC on the moving point game

The number under each graph is the number of games played. Speeds: $\alpha = 0.1$, $\beta = 0.4$, $\gamma = 0.002$, $\rho = 0.1$.

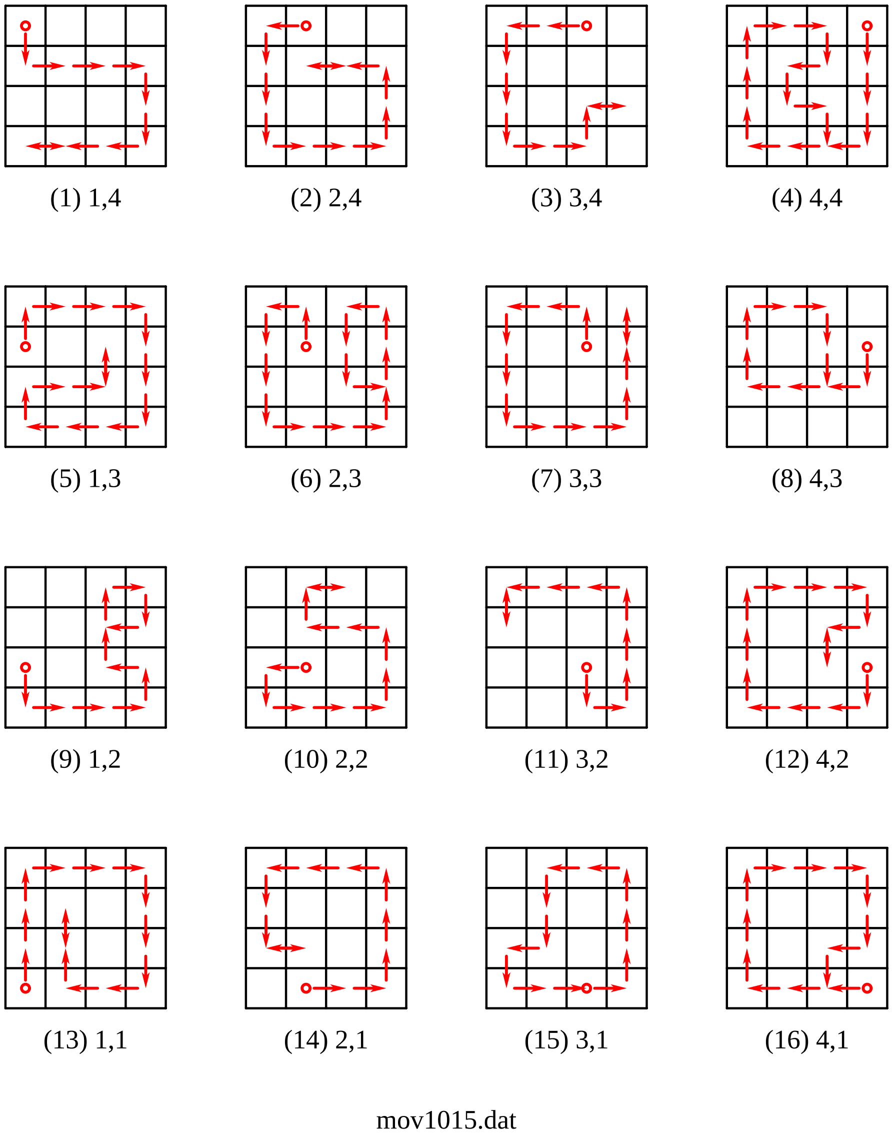

Figure 9.7: A "cross section" of the same run as Figure 9.6, after 4000 plays.

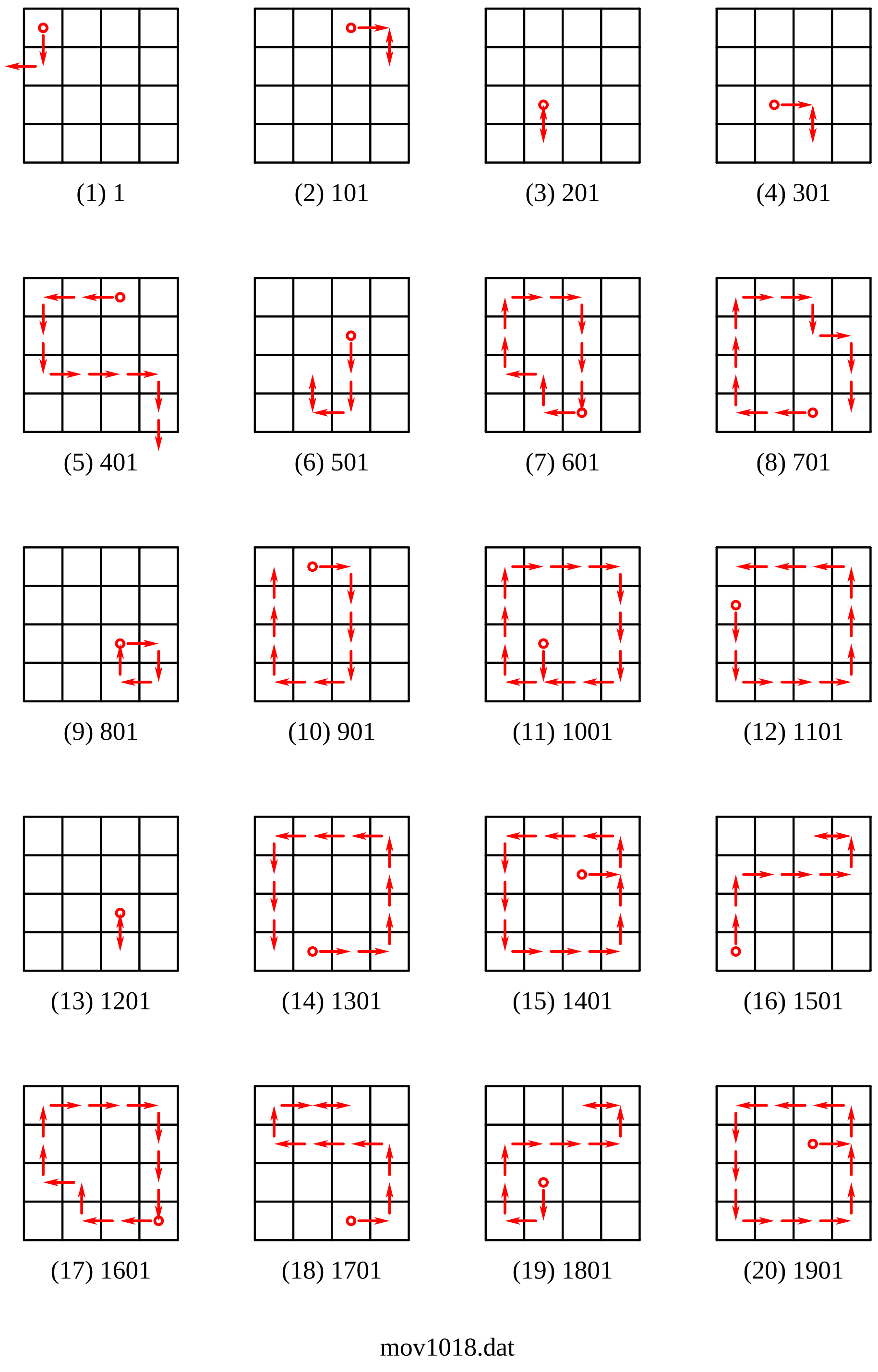

Figure 9.8: Samples from the history of a run of BAC on the moving point game

The number under each graph is the number of games played. Speeds: $\alpha = 0.1$, $\beta = 0.4$, $\gamma = 0.005$, $\rho = 0.1$.

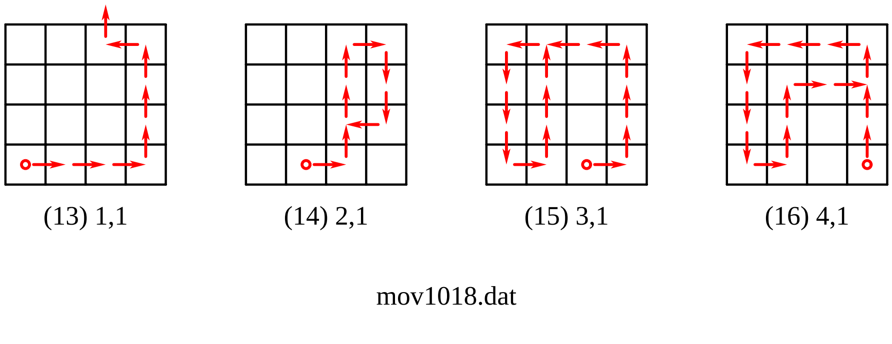

Figure 9.9: A "cross section" of the same run as Figure 9.8, after 2000 plays.

of BAC, and the games it is to solve. The only restriction on how the subroutines for the games should be written is that it should be able to give a starting state, that it should give a new state and an reward after being informed of the action taken, and that it should be able to tell when the game is over.

(2) There are no spatial relations presented to the BAC, as demonstrated in the following metaphor due to [Wer90c]. The moving point game as "seen" by the BAC can be described as this: There is a display board with 32 randomly located lights, and a control board with 4 randomly located buttons. At the start, two lights on the display board are on. At each step, the BAC is required to "press a button". Then one light will turn off and two other lights will turn on. This process continues until at some point the game is declared over and the number of "on" lights on the display board in the score. Without the help of spatial concepts, it is very difficult for a human player to discover the underlying rule of the game, (ie., which new light will be on after press each button, and when the game will be over,) not to say to play the game. The counting game can be described similarly, except that there are two players.

(4) The environment need not be stationary, as demonstrated in the counting game, although this goes beyond the formal theory. In principle, this can also be used to simulate chess. Our current implementation of the simulation on a serial machine is too slow to carry it out.

(5) The moving point game is an extension of the maze problem played by Dyna in [Sut90]. The new factor is that there are combinatorially enumerable states and there are numerous local optima for the decision module.

The combination of these properties seems have not been achieved by previous methods.

Chapter 10

Conclusions, Discussions, and Suggestions

10.1 Summary of Results

In this thesis we have developed a general mathematical framework of adaptive information processing, homogeneous semilinear neural networks and adaptive computers. The technical results obtained are summarised as follows.

1. The theory of information geometry is applied to parameterised Markov chains. The difficulty of computing shift entropy gradients on a Markov chain by back propagation is identified. Formal definitions of the Monte Carlo principle, the Entropy Principle, a formal definition of simulated annealing and Gibbs sampler are given in the framework of information geometry, their interrelationships are characterised. These results are generally applicable to stochastic adaptive computations. The concept of trainable information processor (TIP) is also formulated which is an abstraction of neural network models.

2. The class of homogeneous semilinear neural networks, with structures, dynamics and learning rules, are formally defined, which unifies and generalises the classical neural networks including backpropagation neural networks, Boltzmann machines, (discrete and continuous) Hopfield net, "belief networks", as well as logical circuits. This unified formulation will be helpful for the hardware implementation of NN.

3. A general class of NN structure which guarantees stability, the FB-structure, is defined and studied. This structure generalises most of the known H.S.NN structures which is proved stable. Examples are given which show that it cannot be further generalised without compromising stability.

4. The relation between the two most commonly used NN dynamics, DC and SQ, are clarified. Proof of the equivalence of different formulations of DCB-NN are given.

5. Numerous neural network learning rules are classified by several simple criteria, showing how a learning rule given for one type of NN can be applied to another.

6. The ME learning rules for SQFB-NN are defined, which are biologically plausible in a much stronger sense than any previously known learning rules.

7. Analysis and simulations show that gradient-following (GF), which underlies all the previously known general learning rules, is almost certain to be trapped in local optima for many quite simple and practical problems.

8. The SA-ME learning rule is proposed for SQFB-NN. It is analysed in the framework of information theory. Simulations show that it converges to global optima in all the cases where GF learning rules converge to local optima.

9. All the learning parameters are defined with reference to a universal normalisation convention, independent of the problem to be solved. This allows numerical simulations to be compared with quantitative predictions, as has been done for SQFB-NN on several encoder problems. Experiments show that the simplifications made in the theory do not have undesirable effects.

10. An efficient method of implementing Gibbs samplers on conventional computers is given. When simulating a Boltzmann distribution with fixed energy and changing temperature, the computation time is independent of temperature.

11. A general formulation of adaptive Markov decision process (AMDP) is given, which unifies and generalises the formulations of discounted reward programming, total return programming and average gain programming in the theory of dynamic programming and Markov decision theory. The concept of equilibrium solution under a given temperature and discount generalises the classical concept of optimal solution, and takes it as a limit.

12. A general computational model, the basic adaptive computer (BAC), which implements AMDP by TIPs, is given. It is a modular structure where each module can be implemented by a NN as a black box. Several alternative implementations are described.

13. Normalisation conventions are given for the BAC, by which the learning parameters are independent of the problems to be solved. This also makes the simulation results on SQFB-NN for optimal parameters to be applicable to BAC.

14. Simulations of BAC applied to games of conflict, to games with local optimum, and to games with combinatorially enumerable state space are conducted. The results conform theoretical predictions. The implementation of BAC can solve such different problems without any change (except size).

Throughout the thesis we have tried to provide a general methodology to put up a system which satisfies these constraints at the same time. In the following sections we shall elaborate on this methodology and its implications, as well as open questions and suggested future researches.

10.2 Open Questions and Future Work in Combinatorial Optimisation and Adaptive Information Processing

In a sense any computation is optimisation. In practice, however, optimisation problems come in many different forms requiring fundamentally different treatment. In this thesis we have concentrated on the class of combinatorial optimisation problems. Such problems have enumerable amount of information embodied in a combinatorially enumerable amount of data.

The simulated annealing (SA) method is the first computational method which is formally based on the amount of information instead of amount of data, in the sense that the only reasonable measure of information is entropy, and that to maximise the linear combination of entropy and evaluation results in the Boltzmann distribution. However, it still ignores the mutual information between different instances of the same problem, such as different distance matrices of the TSP. By introducing SA into NN learning rules, we have effectively derived a generic optimisation method which can utilise such information. By introducing SA to AMDP, we have extended SA to multistage problems. Several open questions arise from this approach.

One such problem is that for a parameterised Markov chain, it is only practical to use the local entropy gradient and the transferred entropy gradient, but not the shift entropy gradient (§3.5.3). There may be three prospects for this problem: (1) It may happen that some practical exact method will be discovered for computing the shift entropy gradient; (2) It may happen that some new formulation of the Entropy Principle will avoid the need for computing shift entropy gradient; or (3) A continuous effort is devoted to find approximate methods. From our numerical experiments, it seems that the second possibility is more likely than the others.

Another open question concerns the relation between $\mathcal{M}$ and $\mathcal{M}_W$. The convergence of simulated annealing to global optimum is only proved for $\mathcal{M}$, while the network can only implement $\mathcal{M}_W$ (§6.6). In simulation experiments, we have never encountered convergence to local optima with such methods, so the question is whether it is possible to prove that SA also guarantees convergence to global optima in $\mathcal{M}_W$.

It is also interesting to know what are the approximation properties of SQFB-NN, apart from those transferred from DCF-NN (§6.3).

The stochastic methods studied in this thesis do not fit into the tradition of statistical methods. Instead of processing random numbers in order to get deterministic statistics, we get random numbers as output. The statistics are only used in the analysis of the methods, not as the results of the methods. More figuratively, instead of computing the statistics of gamblers, the computers gamble themselves. The fact that we are forced to adopt stochastic methods for combinatorial problems will have a profound impact on the way we view the world. Leibniz had hoped that it would be possible to devise a "universal calculus" by which any scholastic dispute could be resolved by calculation. It may turn out that any such calculus will need the mechanism of flipping a coin so as to be practical (to avoid the "combinatorial explosion").

10.3 Open Questions and Future Work in the Theory of Neural Networks

In this thesis a mathematical theory of H.S.NN is developed, by which most of the well-known neural networks are generalised into two general classes of NN, namely, DCFB-NN and SQFB-NN. Although it seems unlikely that a general theory on SCFB-NN would be possible, it is possible to include DCFB-NN and SQFB-NN into a one-parameter family, using the formulation given in [Sej81] (§4.4).

One of the major obstacles so far to the development of neural network hardware have been the great variety of neural networks which has appeared in the literature. Therefore a thorough study on Sejnowski class of NN may hold the key to a unified theory on NN, without which the massive production of hardware NN is not very promising. This study might also provide a linkage to recurrent neural networks, the study of which has been totally omitted in this thesis.

One direction for future investigation is the stability conditions for NN. It would be a major breakthrough if a general sufficient and necessary condition were to be found, but small extensions to the FB-structure is clearly possible in the near future.

Our formulation of learning rules unifies and generalises many of the most important learning rules discussed in [Hin89a], but it still leaves many variations and parameters to choose from. For lack of time, we have only tested certain variant of learning rules on certain variants of NN. It would be of immense value if a thorough investigation is done on this subject. Unlike previous investigations, with our convention on the scaling (§7.3.1), the comparison would be much more meaningful and universally applicable.

Several learning rules for SQFB-NN were studied in this thesis. It is noticeable that the only known ME learning rules with practical implementation (§6.4, §6.6) use cross entropy (CE) while the only known ML learning rules with practical implementation ([AHS85], §5.2, §7.6.4) use reverse cross entropy (RCE). The ME learning rule with RCE has been studied in [Nea92] while the ML learning rule with CE has been studied in [LL88]. No non-enumerative implementations were given in both references. Therefore it remains an interesting open question as whether such learning rules are possible.

There have been many results on the issue of what a NN can represent, but it seems that little is known about what a NN can learn. This is apparently due to the prevailing of GF learning rule for which it is indeed not possible to have general learning theorems. The approximation properties of NN have not been used explicitly here , since it is only what can be learned is of interest for a NN as a TIP. It would be very interesting to investigate what a NN can learn with the SA learning rule.

The tanh function, and equivalently the logistic function, has been singled out as the most important activation function, due to its close relationship with entropy. This enables us to give information theoretical interpretations of many results of NN. Since the form of entropy is unique [Sha48], it is most likely that the tanh function is also unique in this regard. It is unclear how much of this holds for the formulations from the so-called Cauchy machine [Szu86].

Although we do not see at the moment any need for non-homogeneous NN from an engineering point of view, (since we have relied on the structure of AC which is outside NN to carry out many of the functions associated with inhomogeneous NN,) there may well be some nice use of certain kind of inhomogeneous NN. Of course nothing would be gained if we merely pretend that the whole AC is a large inhomogeneous NN.

It was pointed out in [Gen89] that there are three different ways of using a NN. We have concentrated on the second method, namely, learning. A study on recurrent NN and temporal ordering might reveal shortcuts for implementing several modules of AC on one NN.

10.4 Open Questions and Future Work in the Theory of Adaptive Computers

10.4.1 Results on BAC and open questions

The theory and methods studied are quite general in the sense that they describe computational systems which are non-linear, multi-dimensional, stochastic, and adaptive. The first two properties are usually required by most numerical methods, while the last two are the emphasis of this thesis. Significant aspects of our methods are

1. Global optimum is achieved;
2. Combinatorially enumerable state space and action spaces are allowed;
3. Multistage processes are allowed;
4. Learning parameters are specified in a manner independent of problem instances to be solved.

Unlike conventional approach to AI where various *models* demonstrating supposed "aspects of intelligence" were built, our approach requires that different *modules* are built whose merits can only manifest themselves in the ability of a large model composed of such modules. Important open theoretical questions concerning AC are the stability and optimality problems, which will become more prominent when more modules and subsystems are added to AC.

The following restrictions have been made in this research, their removal would constitute subjects of future investigation.

1. The BAC only interact with the environment at discrete time steps. For continuous output, some recurrent networks may be needed.
2. The action space is discrete, although allowed to be combinatorially enumerable. To produce continuous actions a control subsystem should be used in addition to the decision system.
3. The environment is assumed Markovian so everything the AC needs to know is contained in the instant input. If this is not so a memory subsystems would be necessary, and if the capacity of memory is limited the memory system should be adaptive.
4. The AC does not control its input. If it has a sensory device which can be actively controlled, but which cannot give complete input at any single moment, then it needs an adaptive sensory control system.
5. The environmental transition function is assumed to be stationary in the theory, although our experiments allows τ to change with the same speed of π. How much the theoretical constraint can be relaxed is unclear at the moment.

10.4.2 General Adaptive Computers

As we have used the term "basic adaptive computer", it seems appropriate to describe what should be a "general adaptive computer". The BACs are based on the theory of MDP. They each have two to four modules: (D), (E), and optionally, (P) and (E2), each implementing an approximation to a function defined in the theory, namely, π, u_π, τ, and q_π. Since the ultimate purpose of this system is to produce a good policy π, we call the decision module the *main module*. There are many ways that this might be generalised, each adding to the AC an additional *subsystem* containing two to four analogous modules: the main module, the evaluation modules, and the prediction module. The main module maintains a function which is the main purpose of this subsystem. The evaluation modules maintain instantaneous evaluations of the main module. The prediction module maintains an approximation of the environment of the *subsystem*, including other parts within the AC.

In the following subsections we briefly describe four directions such extensions can be made: (1) continuum actions space, (2) non-Markovian situations, (3) incomplete perception, and (4) continuous time.

10.4.3 Control subsystem for continuum action space

Neural networks can be used to solve both decision and control problems. A *decision problem* is a global optimisation problem over time on a finite set, called *action space*. A *control problem* is a differentiable optimisation problem over time on a *continuum* (connected compact set), called *control space*. Control methods are fundamentally inadequate to solve global optimisation problems with multiple local optimum (See [BS50] for an interesting example). For a one-step problem, a decision problem reduces to an integer optimisation problem, while a control problem reduces to a differentiable optimisation problem. Since it seems likely that general SC-NN will remain intractable, the control and decision methods must be combined to solve more general problems.

Example 10.4.1 (*global* optimisation problem on a *continuum*) Given a device D which can minimise over a set of no more than 100 elements, and a device C which can find a local minimum of a differentiable function over an interval of length 1 by gradient-following , define a function

$$f(x) := \sin x + (x/50)^2, \qquad x \in X = [-50, 50),$$

how to minimise f over X? Define $X_D := \{-50, \cdots, 49\}$, and $X_C := [0, 1)$. The problem can be solved by defining an auxiliary function

$$g(i) := \min_{\epsilon \in X_C} f(i + \epsilon), \qquad \forall i \in X_D.$$

Now we can use D to find $i = \arg\min_{i \in X_D} g(i)$, use C to find $\epsilon = \arg\min_{\epsilon \in X_C} f(i + \epsilon)$, and set $x = i + \epsilon$.

This shows that a more general AC which can produce actions from a continuum would require two subsystems, the *decision subsystem* and the *control subsystem*. The main issue for future research would be what the auxiliary input and output to each subsystem should be. It is unlikely that simply cascade a DC-NN after a SQ-NN would accomplish this task.

There is a reason why we assign the term "basic" to an AC with only a decision system, instead of an AC with only a control system: Every adaptive method depends on the geometry of its parameter space. Only the information geometry is intrinsic, in the sense that it is independent of the geometry of the environment once the dimension is fixed. In an intuitive sense, the decision subsystem is the "core" of a AC, and information geometry (of a combinatorial dimension) is the "geometry of metal processes".

Example 10.4.2 In Example 10.4.1, the geometry of D is on a 99-simplex, while the geometry of C is on $[0, 1)$, which clearly depends on the problem. Any permutation of points in X_D gives an essentially equivalent problem, while permutation of points in X_C may easily make the problem unsolvable by any computational method.

10.4.4 Adaptive memory for non-Markovian environment

Another direction of generalisation is when the environment is non-Markovian In a non-Markovian environment, the input to the AC at time t is the situation $s_t \in S$, while the *state of the world* $x_t \in X$ depends on the whole history $h_t := [s_0, a_0, s_1, \ldots, s_t]$. The reward r_t and the new situation s_{t+1} depends on $[x_t, a_t]$, rather than $[s_t, a_t]$ alone. This requires that the AC has a *memory system* if it is to be compatible with the environment..

If the memory system can remember the history h_t and transform a triple $[h_t, a_t, s_{t+1}]$ into h_{t+1}, then the memory of h_t will uniquely determine the state of world x_t, and the whole process will therefore be Markovian. However, this requirement is, except in some trivial occasions, neither realizable nor desirable,

Example 10.4.3 Consider a block maze problem. The history h_t can be represented in the form of a sequence of symbols representing "left turn", "right turn", "forward one block", etc., while the state of the world x_t can be represented as a map of explored routes together with a moving symbol for "you are here". The second form is more compact for storage, since repeated visit to the same block generates no extra data, and is much better for decision making.

For these reasons, the form of the memory m_t should not be pre-specified, and an adaptive "memoriser" is needed in addition to a "store". The *memoriser* is a trainable information processor which combines old memory with new data to form new memory, $M : [m_t, a_t, s_{t+1}] \to m_{t+1}$. The *store* S is a device such as a RAM [1] which can recall m_t at time $t + 1$.

Since the task of the memoriser is to capture whichever feature in its input which might be useful to *other modules using the memory*, a tentative evaluation function for the memoriser is the sum of the performance functions of all the other modules which use the memory. The auxiliary input of the memoriser is therefore the sum of the auxiliary outputs of those modules using the memory. However, there is a major difficulty for this method: The memoriser will tend to produce "pleasant memories", ie. to forget whatever event which might cause trouble in the future, since this is encouraged by the auxiliary outputs of other modules, There are several possible solutions to this. One is to require that the memory to contain all those information which might cause the decision module to make different decisions, not only those which makes the decision module to make "good decision". Another is to explicitly require the memory of "unpleasant events".

[1] This does not reduce its biological plausibility. The function of a RAM can be assumed by a recurrent neural network.

10.4.5 Sensory control of incomplete perception

In some problems the "situation" cannot be perceived directly, so that the "perception" is not Markovian, even if the situation is Markovian. In addition the sensory can be actively controlled [WB91]. With an adequate active *sensory control subsystem* and an adaptive model of the situation of the outside world, a "perceived situation" formed from a sequence of "perceptions" should approximate the real situation.

This should not be treated simply as a non-Markov decision process, although they are closely related, since the sensory action is quite different from the action which influence the outside world. They have the same property that their knowledge of environment is itself determined by their output, so the same problem of "ostrich behaviour", called "perception aliasing" in [WB91], is also the most important issue for sensory control. The difference is that if the memorise discards some information, it cannot be recovered later, while the sensory system can always choose to look again at the same feature of situation.

The solution to this problem that nature provides seems to be to weigh unpleasant information with more importance. More specifically, the evaluation of sensory control seems to be exactly the negative of the evaluation of decision. (Suppose a deer spots a lion in distance. What direction should it direct its attention to, and what direction should it move to?)

10.4.6 Continuous time models

In the physical world time is continuous (not considering quantum effects). A tentative idea for constructing an AC which can act in continuous time is to cascade the back-propagation through time (BTT) learning in recurrent networks [Wer90a], which can generate aperiodic output given a stationary input, and is useful for filling the gap between two consecutive time steps. As remarked previously, BTT is inefficient for learning when there is a long delay in the reward, so they will only be useful for filling the gaps between a discrete time steps.

As with the control module, which is also to be cascaded after the decision module, the most important issue will be how to apportion the discrete actions and the continuous movements. In the control literature, this is view as "how to assign intermediate goals".

10.5 Intelligence and Artificial Intelligence

The philosophical implications of viewing computation as information processing rather than data processing has a strong implications on the debate over artificial intelligence (AI) [Sea80, Pen90].

10.5.1 A tentative definition of intelligence

There seems to be no existing definition of intelligence which can encompass all the special features one usually assigns to this concept [Min87]. We suggest a tangible definition which may serve as a working definition. This definition was partly inspired by the observation made in [Min87] that intelligence is something which slips away as soon as we know a procedure to do it.

Definition 10.5.1 The ability of a system to adapt its behaviour in a sufficiently complex environment so that its performance can surpass any deterministic discrete non-adaptive system of similar complexity is called *intelligence*.

(1) General knowledge, solutions to problems, etc., are not intelligence, since the criterion is not based on the amount of solved problems, but on the ability to solve unsolved problems. In terms of AMDP, intelligence is not a property of the operator π, but the operator δ.

(2) Most of what is known as "artificial intelligence", especially the so-called expert systems, can be more descriptively called "artificial skills" [2]. This applies to anything which we call (non-adaptive) data processing.

(3) Whether a system is intelligent is irrelevant to the description of system.

(4) Intelligence is relative to the environment. If the environment has no structure at all, nothing can be intelligent in it. Furthermore, a system may be intelligent in one environment but not in another.

Remark 10.5.1 For a system to be intelligent, it must satisfy the following two conditions: (1) It must be stochastic, so that the amount of data expressible would far exceeds the amount of mutual information between its behaviour and its performance; (2) It must have the ability to accumulate such information, other than raw data, so that the actual amount data used to store this information is not prohibitive. Neither a deterministic system nor a totally random system can be intelligent. The former has no flexibility while the later has no direction. Neither of them can learn.

10.5.2 The Turing Test

So far the only widely accepted criterion of artificial intelligence is the famous Turing test [Tur50].

Definition 10.5.2 A *Turing Test* (TT) is the following procedure: An interrogator from a certain cultural background is to determine the identity of a typical human adult from the same background and a suitably programmed and updated digital computer, on the basis of a teletype conversation for a fixed length of duration with both subjects, both of them trying to be identified as human. A *restricted Turing Test* (RTT) is a procedure which is identical to TT except that the topics of conversation is restricted to a fixed domain dependent on the machine.

Turing's own guess [Tur50] was that at the end of this century the best machine would be powerful enough for an ordinary interrogator not to be able to derive 0.12 bit of mutual information between five minute's conversation and the correct identification. $(1 + 0.3 \log_2 0.3 + 0.7 \log_2 0.7 \approx 0.12.)$

We can convince ourselves to some extent that TT is a proper test of the statement that "the machine has human-like intelligence and have been brought up in the same cultural background as the other candidate".

[2] This in no way diminishes their usefulness. It is only that we define intelligence not as the skills to do something, but as the ability to acquire these skills by themselves.

10.5.3 Turing machine and its generalisations

The following definitions are not formal, but can be easily formalised.

Definition 10.5.3 A *Turing machine* (TM) [Tur36] is a device with finite internal states, which can perform the following operations on an infinitely long tape of binary digits (bits): read a bit, write a zero, write a one, move one bit to the left, move one bit to the right, and stop, as well as changing internal states along these operations, according to a specific set of instructions and the internal state. The binary number on the tape when it starts is its input, that when it stops is its output.

A TM can be represented by $T(m) = n$ where m is the input, and n is the output. All the possible instructions of TMs can be coded by binary numbers. Such a code is called an *algorithm*. The Turing machine whose algorithm is k is denoted T_k. A *universal Turing machine* (UTM) is a TM U such that $U(k, m) = [k, n]$, where $n = T_k(m)$, for all k and m.

Definition 10.5.4 (1) A *stochastic Turing machine* (STM) is a modification of TM with the additional operation of writing a zero or one according to the result of flipping a coin (or any other random experiment). (2) An *extended Turing machine* (ETM) is a modification of TM which has a memory of previous computations. For each instance of computation, it can be represented as $T_k(l, m) = [l', n]$. The next time an input is supplied, l' will be the new l. (3) The definition of *stochastic extended Turing machine* (SETM) is the combination of those of STM and ETM. (4) All these definitions can be generalised to continuous state, continuous input-output machines (hence defining correspondingly CM, SCM, ECM and SECM).

An ETM can also be represented as $U(k, l, m) = [k, l', n]$. We call T_k the *central TM*, l the *inforbase*[3], and k the *meta-algorithm* of the ETM. At each fixed moment, we call l the *instant inforbase*, the TM $T_{[k,l]}$ such that $T_{[k,l]}(m) = n$ the *instant TM*, and $[k, l]$ the *instant algorithm*. Since CMs are well simulated by TMs by numerical analysis, we tacitly assume that they are identical.

10.5.4 Can digital computers have intelligence?

This question can be d??entangle into the following presumably easier questions (with some "pseudo-mathematical notations" [4]).

1. (philosophy): Is our definition of intelligence the right one?
2. (mathematics): Is SETM enough for intelligence ($SETM \cap I \neq \emptyset$)?
3. (mathematics): Can ETM approximate SETM in some sense ($\overline{ETM} = SETM$)?
4. (physics): Is there any natural phenomenon which is intrinsically random ($SETM \neq ETM$)?

[3] This term is coined because we have not found any existing word suitable for this concept. The words "database" or "experience" implies something accumulating in volume, not in refinement. The word "software" implies some outside programmer for responsibility of changing it.

[4] It is "pseudo" since the exact mappings and topologies have yet to be defined

5. (mathematics) Can CMs be modelled by TMs ($\overline{TM} = CM$, etc. This includes four different questions.)?
6. (biology): Is the brain a SECM ($B \subseteq SECM$)?
7. How far can we go toward AI in the framework of SECM with technological constraints (maybe simulated in ETM)?

For the philosophical question refer to [Dre79], which discusses the failure of symbolic AI [5]. It should be remarked that the final conclusions there (that computers cannot have intelligence) should not be read literally since our terminology is obviously quite different.

We conjecture that there are SETM which is intelligent, and that the space of ETM is dense in the space of SETM in a suitable sense. The first statement in the conjecture is a formalisation of ideas in [Tur50], where the necessity of "random element" and "child machine" for passing the TT is discussed. The second statement of the conjecture means that SETM can be (approximately) implemented on TM as a virtual machine (For stochastic approximations refer to [Rip87, Knu81]). In this thesis we have tacitly assumed that the use of *pseudo-random number generator* (PRNG) in the simulations produces no substantial difference from that of RNG. Turing has conjectured that SETM=ETM, by using, for example, the digit of π as a RNG [Tur50] [6].

For the physics question we refer to numerous texts on quantum mechanics. A more quite description can be found in [Pen89].

For the biological question, it seems a common sense that the brain is a SECM, where the collection of genes is the meta-algorithm while the current personality, mind, mentality, or the like, is the *instant SCM*. Although the behaviour is *stochastically* determined by the instant SCM, the instant SCM itself will change with new experience.

The last question in the list is really a task, which is all we are interested in. Our research is a step in this direction, as the BAC is clearly a SETM. It demonstrated that it is possible to construct artificial systems that is both flexible and can direct the flexibility (Remark 10.5.1).

Remark 10.5.2 Many of the current discussions on AI ([Sea80] and commentaries, [Pen89, Slo92], and [Pen90] and commentaries.) use the term "mechanical" as an equivalent term for "physical", without specifying whether it includes flipping a coin. The term "algorithmic" is also frequently used without specifying whether previous results are allowed to be used as part of the input, or, in a different but equivalent point of view, as part of the algorithm. Many of such debates can be resolved by using the concepts introduced here.

10.5.5 Answers to some possible objections

The most challenging objections to our definition of intelligence come from the line that the set of intelligent systems might be empty.

One possible objection is that the simulation of SETM on a ETM is deterministic so that there is no fundamental difference between stochastic and deterministic computations. The answer is that the whole system is deterministic, but the part without PRNG

[5] The word "failure" does not mean that AI failed to put the results of human intelligence into machines, but that it failed to put any intelligence into machines.

[6] Note that π can only be computed by a ETM, not a TM since the latter will never stop [Pen89].

is approximately stochastic. In order to avoid *cycling of the PRNG*, it is necessary that the internal memory of the PRNG should not have a limited size. Therefore PRNG can not be regarded as an negligible part of the system.

Another possible objection follows Shannon's coding theorem [Sha48], which says that any information source can be coded in arbitrary precision so as to use an amount of data almost as small as the amount of information. There is even a general deterministic algorithm for doing this, namely the Huffmann coding [Abr63]. This seems to imply that a symbolic expert system consisting of rules and counter-rules can do as good as a stochastic system to simulate the *reaction* of a human adult, which is all that is needed for passing TT. The problem is that such a coding is in the form of "rules-and-exceptions" [7], and its computation requires the enumeration of all the possibilities starting from the least probable option. So if the set of all the possible alternatives is not enumerable, such a scheme can not even start. Such systems are not adaptive, either, since there are critical states in which a slight change in the probabilities will results in the reversal of the rôles of rules and exceptions.

There is a famous metaphor used in the "strong-AI debate" arguing against the TT as a test of intelligence, called the "Chinese room problem" [Sea80]: An English speaker knowing nothing about the Chinese language is locked in a room with formal rules written in English for handling Chinese characters written on slips passed through a window. The question is, if the input-output relation of these slips is such that it would be interpreted by a Chinese speaker outside the room as "intelligent conversation" originated from a Chinese speaker, does this imply that the person in the room or the whole system understands Chinese? This metaphor generated intense debate simply because its premise is vacuous. The only reason that we ascribe intelligence to an agent who understands a natural language such as Chinese or English, but not to an agent who only understands a formal language such as Fortran or Lisp, is that natural languages cannot be completely described by a formal grammar within the resource of this universe. In this sense, the "Chinese room problem" is more absurd than the following example: Suppose we can print the whole Encyclopaedia Britannica on one page of ordinary English text, does this page alone constitute an encyclopaedia? Obviously there are infinitely many reasons that it is, and infinitely many reasons that it is not, for the sole reason that it is impossible to be realized unless we assume the page is so large to make the question meaningless.

10.6 Algorithms and Computations

There are two "paradigm shifts" in our approach to computation. To distill them we have to somehow go beyond what have been studied so far in the thesis.

10.6.1 Pan-Algorithms

The term "algorithm" seems to have been used by researchers in almost every field since the emergence of computer. By examining the concept of trainable information processors, we see that there are different levels of algorithms, some of them are used to change the others. We call them *pan-algorithms* to distinguish from TM algorithm.

[7] The code for "rules" are shorter than that for "exceptions". More precisely, the length of each code is almost proportional to its entropy.

Here we have taken a broad sense of pan-algorithm, allowing it to be both continuous and stochastic. We give examples of zeroth, first, second and third order pan-algorithms in philosophy, sciences, engineering, and everyday life.

A *zeroth order pan-algorithm* is a sequence of pre-programmed output without dependence on any input. The following examples are instances of zeroth order pan-algorithms: Leibniz's monad, clock, open-loop (feedforward) controller, plant (approximately), etc.

A *first order pan-algorithm* is a set of rules which specifies outputs in relation to inputs. Its examples include: Newtonian atom, TM algorithm, thermostat, feedback control, associative psychological model, general scientific result (in "if-then" form), neural network dynamics, lower animal (which only "reacts" to environment in an innate manner), evolution of plant, etc.

A *second order pan-algorithm* is a set of rules for changing free parameters in a first order pan-algorithm with respect to the feedbacks from the environments. Its examples include: neural network learning rule, adaptive control, developmental psychological model, software engineering, meta-algorithm, scientific method, higher animal (which can learn to react), evolution of lower animal, etc.

A *third order pan-algorithm* is a set of rules for changing free parameters in a second order pan-algorithm with respect to the feedbacks from the environments. Its examples include: evolution of brain of higher animal, study of learning method, scientific methodology (study of scientific methods), genetic algorithms applied to improving neural network learning rule, etc.

Although this list obviously can go on for ever in theory, higher order pan-algorithms do not have direct relevance, if we take into account the computational demands of these pan-algorithms. Consider a simple environment with only ten states and ten actions. If each pan-algorithm is fully adjustable, then from zeroth to third order pan-algorithms will have about 10, 10^{10}, $10^{10^{10}}$, and $10^{10^{10^{10}}}$ options, respectively. Since it is impossible to enumerate 10^{200} in our universe, higher order pan-algorithms are only practical under special conditions. The most common of such conditions is that a $(k-1)$th order pan-algorithms is parameterised, leaving a "small" (ie., not exponential) number of free parameters to be processes by a kth order pan-algorithm. This is exactly the property which makes the higher order pan-algorithms listed above both practical and useful. The typical time scale of first, second and third order pan-algorithms in these examples are seconds, years, and millions of years. In this thesis we tried to construct a theory of a subclass of second order pan-algorithms (NN learning rules and AMDP) on parameterised first order pan-algorithms (NN dynamics and MDP).

Therefore, whenever we contemplate higher order pan-algorithms, we must make sure that the environment allows the data to be compressed, and that in the parameterisation of lower order pan-algorithms the interdependent data are compressed and irrelevant data removed (to a certain extent), while the necessary information are preserved. The debate on whether algorithm is enough for intelligence becomes pointless if it is not clear whether pan-algorithm are considered and which order of pan-algorithms are allowed.

10.6.2 Types of computation

Another "paradigm shift" the neural network research brought about is the distinction of the following three types of computational problems:

The *Type A computation* is characterised by the following problem: Given an input and a criterion for output, how much resources (time, storage) it takes to get the output?

The *Type B computation* is characterised by the following problem: Given an input and an evaluation of output operator on the set of possible outputs, and a certain amount of resources, how good the output can be?

The *Type C computation* is characterised by the following problem: Given a problem class, how fast can a solution method's performance on this class improve by solving samples from this class?

Type A computation problems are in general ill-defined, since there is no TM algorithm to decide whether an arbitrary TM algorithm will ever stop. Even when it is well defined, its solution is often useless. For example, the answer to the TSP problem is that it needs exponential amount of resources. Type B computation problems are well defined. There can always be an average performance measure for Type B computation, provided that the maximal and minimal evaluations are both finite. The most obvious example to transforming the Type A computation into Type B computation is the common practice in software by which a "process" is terminated if it exceeds a pre-allocated amount of resource.

It is obvious that what is usually required is actually Type B computations, yet the conventional computational theory are usually expressed in terms of Type A computations [GJ79], despite the fact that it is both unnatural and ill-posed. This seems to be mainly due to the fact that the only previously known computational model is the Turing machine and its equivalents, which is defined in terms of Type A computation. Traditional numerical analysis, in contrast to theoretical computer science, has put greater emphasis on Type B computations, by preferring iterative solutions to direct solutions. However, this has often be considered as a technique. What neural network brought us is a computational paradigm in which it is both desirable and natural to use Type B computations and Type C computations.

It has been a quite widely accepted common sense that the main reason computers are not as "smart" as brains is that it is not as fast or as large as brain in the parallel sense [Boc91]. We suggest that the main difference is that brain computation only uses relevant information, instead of data. The ability to extract information from data is not *a priori* guaranteed for an arbitrary system, but can only exist as the result of elaborated construction.

10.7 Final Remarks

We have solved several theoretical problems, opened some engineering possibilities and reformulated some philosophical questions. Now it is all too easy, as have always been the case when a philosophical subject became available for scientific investigation, for one with an optimistic mind, to suppose that there will not be any principal obstacles in the field. Such assumption will not be granted even if, which we do not claim, that all the previously known obstacles have been removed.

One of the main issues discussed in this thesis which may be expected to come up at many unexpected places is the combinatorial complexity. Many things may seem possible "in principle", ie. out of context, but not in practice when the constraints of technology, or even the whole universe is considered.

Early research of AI has generated substantial criticism by claiming too much generality than what is actually generalisable, in particular, by claiming that passing a RTT can be generalised to passing a TT (See [Dre79] for detailed discussions). In order to restrict our research in the field of science rather than philosophy, here we discuss what is not achieved by our methods, in light of the fact that many theories we have superseded did claim to have overcome some of such limitations. We have discussed (§10.4.2) what BAC can not do but may be done by some general AC by simply adding more modules. The following are what our methodology (adding more subsystems and modules) might cover, with increasing difficulty, but their technical difficulties might be of completely different nature.

1. The AC cannot process natural image as human eyes can do. Natural images have amount of data far exceeding the capacity of any data processing system, although the amount of *useful information*, ie., mutual information between the image and the performance of the system, is quite limited. An image processing module in AC would require more than a stand-alone model of the human vision system, for which a cascade of DCFB-NN and SQFB-NN may be a good model.

2. The AC will not have consciousness, until it has a substantial internal model of the external world (§10.4.4, §10.4.5) which can evolve without external input and can be updated at any moment with any incomplete external input, such that most decisions of the whole system are based on this model instead of the instant input.

3. It will not have self-consciousness and personal value. This will only happen if it has a robotic body, and it has a model of the physical condition of the body in relation, but separate, to the model of the external world.

4. It cannot process natural language. Natural language is a social phenomenon in a species where elaborate cooperation between individuals are important. It also requires special processing subsystems in the brain. When a later AC can have natural language processing capability, the research would almost certain require theoretical tools quite different from those used up to now, even if the BAC is still a centre piece.

5. It will not have the ability of introspective consciousness, which will only happen if it has a separate model of the main decision system. This may be a consequence of the language processing ability.

6. It cannot do logical deduction and mathematics beyond an elementary level. This is the consequence of the fact that data processing is at the boundary of information processing. Therefore even if later versions of AC can be more and more logical, it can never be totally logical. This does not preclude its use in, for example, specialised computer algebra software.

7. As a consequence, it cannot pass the TT, and so it will not be "intelligent" in the sense of TT. It is certainly possible to pass certain RTT, but that does not imply anything significant.

Nearly half a century has passed since Turing announced his optimistic prediction on artificial intelligence. Instead of having produced a "child machine" in twenty years and subject it to "education" in twenty years, as he predicted, we are not even sure whether we have had a glimpse on the principles necessary for producing even a "chicken machine". The bright side of the picture, which is my hope, is that we are beginning to be able to conduct scientific research in this field for some time without being disturbed by philosophical difficulties.

Appendix A

Preliminaries, Terminology and Notation

A.1 General Terminology and Conventions

This appendix is intended to clarify the terminology and nomenclature which have different usages by different authors, and to introduce some new notations. Only default notations used throughout the thesis are listed here, special notations are defined in the text where necessary. We introduce some new mathematical notations, most notably those on mappings, which will be used extensively.

Basic concept from pure mathematics can be found in [Die60]. Most results on matrices which we shall use can be found in [Cia89, Sen73b, CM65, Sen73a]. Our reference to probability and stochastic processes are [Kol56a, CM65, GS82, GS74].

The *slanted typeface* generally denote the first or the most formal appearance of a technical term. This is usually where it is defined, or the standard reference (cited at the beginning of the chapter or the section) is referred to for definition. It may also be the case that this term is specific to some research field our theory is to be applied, so the special typeface only acts as a reminder that this term has a strict definition in that application, although there is no need to get into the detail for our theory.

Whenever possible, the text is divided into three groups: (1) definitions, notations, terminologies, and assumptions, (2) theorems, lemmas, propositions, and corollaries, (3) proofs, remarks, examples, and conjectures. Often material in the first group is not specially labelled as such, but can be easily recognised by the appearance of the slanted typeface and the colon which is used for definition (see below).

We use quite a few abbreviations. Two abbreviations are in general not related, no matter how they look similar, unless they are generated by some explicitly stated rules. (This only happen to our classification of neural networks.)

A.2 Logic and Sets

Most notations in this section are standard. We shall use the the following basic nomenclature and symbols of mathematical logic and set theory: $\in$, $\ni$, $\subseteq$, $\supseteq$, $\subset$, $\supset$, $\cup$, $\cap$, $\setminus$, $\emptyset$, $\{\ldots\}$, $\forall$, $\exists$, $\Longrightarrow$, $\Longleftarrow$, $\Longleftrightarrow$, $\neg$, $\wedge$, $\vee$ and their properties. The last four notations in the above list represent logical "not", "and", and "or", respectively.

Precedence of operation in an expression in always represented by the parentheses ( ). Any ordered collection of objects, such as ordered pair, vector, matrix, and sequence, is always expressed with the brackets [ ]. Any un-ordered collection of objects is always represented by the set notation { }. The colon : represents, and can usually be read as, "such that".

Let p and q be *properties* (propositions), a and b be *primary entities*, and A and B be *sets*. The notation $p :\iff q$ and $q \iff: p$ both denote that "p is *defined* as q", or "q is *denoted* as p". Similar definition apply to $a := b$ and $b =: a$ [1].

The following notational conventions are used,

$$\begin{aligned}
(p,\ q) &:\iff (p \wedge q), \\
(p,\ \text{or } q) &:\iff (p \vee q), \\
(p(a),\ \forall a \in A) &:\iff (\forall a \in A :\ p(a)), \\
(\forall a \in A :\ p(a) \iff q(a)) &:\iff (\forall a \in A :\ (p(a) \iff q(a))).
\end{aligned}$$

Similar convention applies if "$\forall$" is replaced by "$\exists$" and " $\iff$ " by " $\Longleftarrow$ " and " $\Longrightarrow$ ".

The notation $\exists!$ denote "exists uniquely"

$$(\exists! x \in A :\ p(x)) :\iff (\exists x \in A :\ p(x),\ \forall y \in A :\ p(y) \implies y = x).$$

Let A be a set. Then $|A|$ denote the *cardinality* of A, ie. the number of members of A, which can be ∞. The subset of A composed of *members* having property p is denoted $\{a \in A :\ p(a)\}$. When there is no danger of confusion, we shall identify the notation of a single-member set with that of the member [2].

The notations $\mathbb{C}$, $\mathbb{R}$, $\mathbb{Z}$ and $\mathbb{N}$ denote the set of *complex numbers*, the set of *real numbers*, the set of *integers*, and the set of *natural numbers*. Let $a, b \in \mathbb{R} \cup \{\pm\infty\}$. Then *interval* $[a, b) := \{x \in \mathbb{R} :\ a \le x < b\}$. Other intervals are defined similarly. This notation also apply to *rectangles* in higher dimensions, with obvious definition. Let $A \subseteq \mathbb{R}$. Then $A_+ := A \cap (0, \infty)$, and $A_- := A \cap (-\infty, 0)$. $\mathbb{N} = \mathbb{Z}_+ \cup 0$.

$$\mathbb{N}_{m,n} := \mathbb{N} \cap [m, n], \qquad \mathbb{N}_n := \mathbb{N}_{1,n} = \{1, \ldots, n\}.$$

The set $\Delta^{n-1} := \{p = (p_1, \ldots, p_n) \in {\mathbb{R}^n}_+ :\ \sum_i p_i = 1\}$ is called the *standard simplex*.

The *Cartesian product* of a *finite family* of sets $\{A_i :\ i \in \mathbb{N}_n\}$ is $\prod_{i \in \mathbb{N}_n} A_i := \{[a_1, \ldots, a_n] :\ a_i \in A_i,\ \forall i \in \mathbb{N}_n\}$. When all the A_i are identical, this gives $A^n := \prod_{i \in \mathbb{N}_n} A$.

A.3 Relations, Orders, Mappings

Many notations in this section are new. Let A and B be two sets. The set of all *relations* between A and B is

$$(A \sim B) := \{f :\ f \subseteq A \times B\}.$$

The *inverse* of a relation $f \in (A \sim B)$ is $f^{-1} := \{[y, x] :\ [x, y] \in f\}$. A relation between A and A is also called a relation on A. Let $R \in (A \sim B)$. Then $[a, b] \in R \iff: aRb$.

[1] It has not been decided whether to use such notations as sentences or phrases in the grammatical sense, so there may be some awkward but perfectly unambiguous sentences involving them.

[2] It seems to me that there is a simple variation to the ZFC set theory which legalises this notation, but this is not our concern here.

There are several subsets of $(A \sim B)$ which are of particular interest. They are represented by replacing the symbols"(", "$\sim$", and ")" in the notation $(A \sim B)$, according to the following rules.

1. If "$\sim$" is substituted by "$\to$", then the *image* is *unique*, ie.,

$$[x, y] \in f,\ [x, z] \in f \implies y = z.$$

2. If "$\sim$" is substituted by "$\leftarrow$", then the *pre-image* is unique, ie.,

$$[x, z] \in f,\ [y, z] \in f \implies x = y.$$

3. If "(" is substituted by "[", then the image *exists*, ie., $\forall x \in A :\ \exists y \in B :\ [x, y] \in f$.

4. If ")" is substituted by "]", then the pre-image exists, ie., $\forall y \in B :\ \exists x \in A :$ $[x, y] \in f$.

Following are some of the most important examples of these notations: The set $[A \to B)$ consists of *mappings* from A to B, ie.,

$$f :\ A \to B \iff f \in [A \to B).$$

The set $[A \to B]$ consists of *surjections* from A to B. The set $[A \leftrightarrow B)$ consists of *injections* from A to B. The set $[A \leftrightarrow B]$ consists of *bijections* from A to B. As usual when $f \in (A \to B)$, $[a, b] \in f \iff: b = f(a)$. If A, B are linear spaces then $L[A \to B)$ is the linear space of linear operators from A to B. The notation $L^k[A \to B)$ denote the space of k-linear mappings from A to B. If A and B are differentiable manifolds then $C^k[A \to B)$ is the differentiable manifold of kth order differentiable mappings from A to B. We also use C^1_2 to denote the functions which is first order differentiable with respect to the second argument.

If $f \in [A \to B)$ and $X \subseteq A$, then the *restriction* of f to X is $f|_X := f \cap (X \times B)$. $\forall f, g \in [A \to B) : f \underset{X}{=} g \ :\iff\ f|_X = g|_Y$.

Denote $f(A) := \{f(x) :\ x \in A\}$. Whenever $\exists f \in [A \leftrightarrow B)$ it is allowed to *identify* A with $f(A) \subseteq B$. When both A and B have structures such identification are only performed if the structure are preserved by f and f^{-1}.

The *Cartesian product* of an *indexed family of sets* $\{A_i :\ i \in I\}$, where I can be any set, is $\prod_{i \in I} A_i := [I \to A)$. This definition implies the previous one.

We also define $[A|B \to C) := [B \to [A \to C))$, which can be naturally identified with either $[A \times B \to C)$ or $[B \times A \to C)$, and hence to $[B|A \to C)$, as long as there is no danger of confusion in the order of the *operands* (*arguments*).

When defining a mapping, we use the notation $f \in [A \to B) :\ f(x) := y$, instead of the more conventional $f :\ x \in A \to y \in B$. This new notation is much more flexible and clearer, especially when the expression of y is complicated. For example, when f is implicitly defined by a property p, the definition looks like

$$f \in [A \to B) :\ f(x) := y :\ p(x, y),$$

and the suggested reading is "f is defined as a mapping from A to B such that for all x in A, $f(x)$ is the unique member y in B for which the property $p(x, y)$ holds".

In line with these notations, we also write an optimisation problem as $\max f : C$, instead of the conventional

$$\begin{cases} \text{maximize} & f, \\ \text{subject to} & C, \end{cases}$$

where f is the objective function, and C is the condition. The solution of the above problem is denoted as $x : \max f(x) : C(X)$.

Let $R \subseteq A^2$ be a relation. The following is a reminder of standard terminology on the type of R. For all $a, b, c \in A$:

$$\begin{array}{l} aRa \quad \text{(reflexive)}, \qquad \neg aRa \quad \text{(irreflexive)}, \\ aRbRc \implies aRc \quad \text{(transitive)}, \qquad aRb \vee bRa \vee a = b \quad \text{(total)}, \\ aRb \implies bRa \quad \text{(symmetric)}, \qquad aRb \implies \neg bRa \quad \text{(asymmetric)}, \\ aRbRa \implies a = b \quad \text{(antisymmetric)}. \end{array}$$

Definition A.3.1 (1) A *pseudo order* [3] $\precsim$ is a reflexive, transitive relation. (2) A *partial order* $\preceq$ is a pseudo order which is antisymmetric. (3) A *strict partial order* $\prec$ is an irreflexive, transitive, antisymmetric, relation; or, equivalently, a transitive, asymmetric relation. (4) A *quasi order* [4] $\lesssim$ is pseudo order which is total. (5) An *order* $\leq$ is a quasi order which is antisymmetric; or, equivalently, a partial order which is total. (6) A *strict order* $<$ is an irreflexive, transitive, antisymmetric, total relation; or, equivalently, a transitive, asymmetric, total relation. (7) An *equivalence relation* $\sim$ is a reflexive, transitive, symmetric relation.

For a given equivalence relation $\sim$, the *equivalence class* of $a \in A$ is $[a]_\sim := \{b \in A : b \sim a\}$, which we call a *cluster*. The *cluster set* is denoted $A_\sim$. A *partition* P of A is a set of subsets of A:

$$A = \cup P, \qquad \forall X, Y \in P : X \cap Y \neq \emptyset \implies X = Y.$$

The following propositions will be important for us. They are straight coward to verify.

Proposition A.3.1 *A set P is a partition of A if and only if it is the set of clusters of an equivalence relation $\sim$ on A.*

Proposition A.3.2 *There is a one-one correspondence between partial orders and strict partial orders, and between orders and strict orders.*

$$a \preceq b \iff a \prec b \vee a = b, \qquad a \leq b \iff a < b \vee a = b.$$

Proposition A.3.3 *Each pseudo order $\precsim$ induces an associated equivalence relation $\sim$ and an associated strict partial order $\prec$:*

$$a \sim b :\iff a \precsim b \wedge b \precsim a, \qquad a \prec b :\iff a \precsim b \wedge \neg b \precsim a.$$

It is also determined by them,

$$a \preceq b \iff a \prec b \vee a \sim b.$$

[3] Not standard name.
[4] Not standard name.

Proposition A.3.4 *Suppose that $|A| < \infty$. Then each pseudo order $\precsim$ induces an associated partial order $\preceq$, and each quasi order $\lesssim$ induces an associated order $\leq$, on the cluster set $A_\sim$*

$$[a]_\sim \preceq [b]_\sim :\iff a \precsim b \qquad [a]_\sim \leq [b]_\sim :\iff a \lesssim b.$$

Proposition A.3.5 *Each pseudo order $\precsim$ extends to at least one quasi order $\lesssim$. Each partial order $\preceq$ extends to at least one order $\leq$. Each strict partial order $\prec$ extends to at least one strict order $<$.*

Proposition A.3.6 *The intersection of a family of pseudo orders is still a pseudo order.*

Let R be an arbitrary relation on A. Then the *minimum pseudo order* containing R is defined as the intersection of all pseudo orders containing R.

Notation. An order $\leq$ is represented in the form of $[\ldots]$. An equivalence $\sim$ is represented by its cluster set $\{\{\ldots\},\ldots\}$. A partial order $\preceq$ can be represented by a set of *ordered segments* $\{[\ldots],\ldots\}$, which is not unique. Therefore, a quasi order can be represented as an order of clusters $[\{\ldots\},\ldots]$. A pseudo order can be represented as a set of ordered segments of clusters $\{[\{\ldots\},\ldots],\ldots\}$, which is not unique.

A.4 Linear Algebra and Analysis

A space $(X, +, \circ, \cdot)$ is called an *associative commutative algebra* (abbrv. algebra) if $(X, +, \cdot)$ is a *linear space* and $(X, +, \circ)$ is a *ring* where the multiplication $\circ$ is *associative*, *commutative*, and it is *distributive* with respect to the addition $+$. A *normed algebra* is an algebra in which the linear space is *normed* while the *norm* is *sub-multiplicative* for the ring. We shall only use the most simple properties of normed algebra. Examples of normed algebras include $\mathbb{C}$, $\mathbb{R}$, $\mathbb{R}^{n\times n}$, and $[A \to \mathbb{R})$. The notation $\simeq$ is generally used to denote isomorphism, be it in the sense of linear space, algebra, or manifold, as will be specified when used. We often identify $\mathbb{R}^n$ with $[\mathbb{N}_n \to \mathbb{R})$, except that $\mathbb{R}^n$ is only regarded as a linear space while $[\mathbb{N}_n \to \mathbb{R})$ is regarded as an algebra.

Generally, we use lower case letters for *vectors* and functions, upper case letters for matrices and *operators*. The notation x^n either denotes x with superscript n, or x to the power of n, depending on the context. In *matrix notation*, we shall write each $x \in \mathbb{R}^n$ as a *column vector*, while $x \in \mathbb{R}^{n*}$, the *linear dual* of $\mathbb{R}^n$, as a *row vector*.

Let $A \in \mathbb{R}^{m\times n}$. The ijth *component* of matrix A is denoted A_{ij}. For the matrix A_k which already has a subscript, the notation $(A_k)_{ij}$ is used. The *transpose* of A is denoted A^T. We also use the notation $A = [a_{11}, \ldots, a_{1n}; \ldots; a_{m1}, \ldots, a_{mn}]$, where $a_{ij} := (A)_{ij}$, apart from the conventional notation..

The ith *unit vectors* of $\mathbb{R}^n$ is denoted $\mathbf{e}_i$, which has a 1 at the ith position and 0 elsewhere. The *unit function* is $\mathbf{1} \in [A \to \mathbb{R}) : \mathbf{1}(x) = 1,\ \forall x$. This notation is also used for the vector $\mathbf{1} := [1, \ldots, 1]^T$. The *identity matrix* of $\mathbb{R}^{m\times n}$ is denoted I_{mn}, or simply I. The *zero matrix* of $\mathbb{R}^{m\times n}$ is denoted 0_{mn}, or simply 0. The *Kronecker* δ is $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ if $i \neq j$. The identity matrix can be represented as $I = [\delta_{ij}]$.

The *exchange tensor* $\mathbb{R}^n$ is defined as the unique *fourth order tensor* $E \in \mathbb{R}^{n\times n\times n\times n}$ such that $v^T E x = x v^T$ where $x, v \in \mathbb{R}^n$. Using subscripts, this is interpreted as $E_{ijkl} =$

$\delta_{ik}\delta_{jl}$, and

$$(v^T E x)_{jk} = \sum_{il} v_i E_{ijkl} x_l = \sum_{il} v_i \delta_{ik}\delta_{jl} x_l = v_k x_j = (x v^T)_{jk}.$$

Partial derivative of $f \in C[\mathbb{R}^m \to \mathbb{R}^n)$ is denoted $\partial f \in L[\mathbb{R}^m \to \mathbb{R}^n) \cong \mathbb{R}^{n\times m}$, apart from conventional notations. We adopt the convention that the derivative with respect to a column vector is a raw vector, and vice versa. This has the advantage that formally

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y}\frac{\partial y}{\partial x}, \qquad \frac{\partial y}{\partial A} = Ex, \qquad \frac{\partial y}{\partial x} = A.$$

where f can be any object, x and y are vectors, A is a matrix and $y = Ax$. These are interpreted as

$$\left(\frac{\partial f}{\partial x}\right)_i = \frac{\partial f}{\partial x_i} = \sum_j \frac{\partial f}{\partial y_j}\frac{\partial y_j}{\partial x_i} = \sum_j \left(\frac{\partial f}{\partial y}\right)_j \left(\frac{\partial y}{\partial x}\right)_{ji} = \left(\frac{\partial f}{\partial y}\frac{\partial y}{\partial x}\right)_i .$$

$$\left(\frac{\partial y}{\partial A}\right)_{ijk} = \frac{\partial y_i}{\partial A_{kj}} = \frac{\partial \sum_l A_{il} x_l}{\partial A_{kj}} = \sum_l \delta_{ik}\delta_{jl} x_l = (Ex)_{ijk}.$$

We shall not use more elaborated properties of tensors, and the above usage can always be verified by writing the subscripts explicitly.

Each $f \in [\mathbb{R} \to \mathbb{R})$ corresponds uniquely to a function $f^* \in [\mathbb{R} \to \mathbb{R})^n : \ f_i := f$, the so-called *vector of functions* and further to $f^{**} \in [\mathbb{R}^n \to \mathbb{R}^n) : \ (f(x))_i := f((x)_i)$, the so-called *function of vectors*. We shall identify f^* and f^{**} with f, thus $[\mathbb{R} \to \mathbb{R}) \subseteq [\mathbb{R} \to \mathbb{R})^n \subseteq [\mathbb{R}^n \to \mathbb{R}^n)$. A member of $[\mathbb{R} \to \mathbb{R})^n$ is called a *diagonal function*. If $f \in [\mathbb{R} \to \mathbb{R})$ is differentiable, then $f \in [\mathbb{R} \to \mathbb{R})^n$ has a derivative $\partial f \in \mathbb{R}^{n\times n}$ which is a *diagonal matrix*.

Let $A \in \mathbb{R}^{n\times n}$. The notations $\sigma(A)$ and $\rho(A)$ denotes the *spectrum* and *spectral radius* of A,

$$\sigma(A) := \{\lambda \in \mathbb{C} : \ \exists x \in \mathbb{R}^n \setminus 0 : \ Ax = \lambda x\}, \qquad \rho(A) := \max\{|\lambda| : \ \lambda \in \sigma(A)\}.$$

Let $p \in [1, \infty]$. The *l_p-norm* $\|\cdot\|_p$ on $\mathbb{R}^n$ (norm on a linear space) is defined by

$$\|x\|_p := \left(\sum_i |x_i|^p\right)^{1/p}, \qquad \|x\|_\infty := \lim_{p\to\infty} \|x\|_p = \max_i |x_i|.$$

The corresponding *induced matrix norm* (norm on an algebra) on $\mathbb{R}^n$ is defined by

$$\|A\|_p := \sup_{x\neq 0} \frac{\|Ax\|}{\|x\|} = \max_{\|x\|=1} \frac{\|Ax\|}{\|x\|}.$$

The most important special cases are

$$\|A\|_1 = \max_j \sum_i |A_{ij}|, \qquad \|A\|_\infty = \max_i \sum_j |A_{ij}|, \qquad \|A\|_2 = \rho\left(A^T A\right)^{1/2}.$$

For any induced matrix norm $\|\cdot\|$,

$$\rho(A) = \lim_{k\to\infty} \left\|A^k\right\|^{1/k} = \inf\{\|A\| : \ \|\cdot\| \text{ is an induced norm}\},$$

$$\left\|A^k\right\| \to 0 \iff \rho(A) < 1 \iff \forall x \in \mathbb{R}^n : \ \left\|A^k x\right\| \to 0 \iff \exists (I - A)^{-1} = \sum_{k=0}^{\infty} A^k.$$

A matrix $A \in \mathbb{R}^{n\times n}$ is called positive (non-negative) if all its elements are positive (non-negative). It is called stochastic if it is non-negative and each of its row sum to unity. It follows that each non-negative matrix can always be represented as the product of a non-negative diagonal matrix and a stochastic matrix. This representation is unique if and only if no row is identically zero. The spectral radius of each stochastic matrix is 1, with $\mathbf{1}$ as corresponding eigenvector.

A.5 Miscellaneous

Since this research spans several areas with drastically different terminology, it is worthwhile to give a list of the terms which have technical meanings here but which may be regarded as a plain language word by researchers in some related research fields.

The following terms are used with their technical meanings in pure mathematics: *function*, *map*, *mapping*, *operator*, *manifold*, *distribution* (probability), *approximation*, *spectrum*, *continuum*. The following terms are used with their technical meanings in applied mathematics: *information*, *data*, *game*, *policy*, *strategy*, *utility*, *optimization*.

The following terms also have specific meaning in this thesis. By *evaluation* we mean an approximation of utility. By *decision problem* we mean discrete optimization over time. By *control problem* we mean differentiable optimization over time. The term *adaptive* always have the meaning of changing over a continuum instead of a discrete set, even for decision theory.

The term “measure” is used in ordinary sense, instead of that of measure theory.

Definition A.5.1 A *problem class* is a tuple $[Q, A, \epsilon]$, Q is a set of input, A is a set of output, and $\epsilon \in [Q \times A \to \mathbb{R})$ is a rule for evaluation input-output pairs. Solving a problem is producing a method $f \in [Q \to A)$ with as high an evaluation as possible. A *problem instance* is a given input.

A *fixed solution* is a $f \in [Q \to A)$. An *adaptive solution* is a method which accumulates information of previous input-output-evaluations, and produces different fixed solutions each time.

A *run* of a solution method is the whole process of starting from scratch until giving acceptable answers. A *presentation* is the process of producing a single output for a single input. It is also called a *trial*. A solution method is called a *generic method* if each run corresponds to a problem class and each presentation corresponds to an arbitrary problem instance in the class.

Example A.5.1 Consider the *travelling salesman problem* (TSP). An example of problem class is TSP20, the set of all 20-city TSP where the input is the *distance matrix* and the output is an integer matrix representing a *tour*. A problem instance is a given distance matrix in TSP20. If a solution method accumulates information about all its past experience in TSP20, then it is generic. If it start from scratch for each distance matrix, it is not generic.

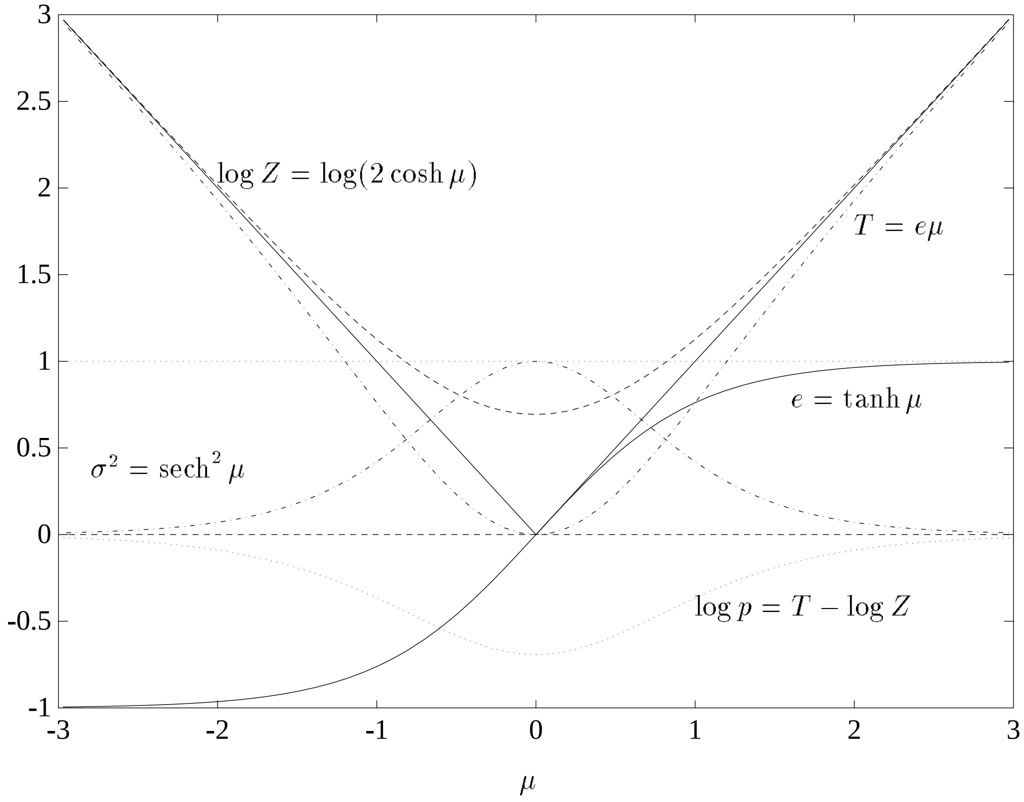

Figure A.1: Graphs of frequently used elementary functions.

Lemma A.5.1 *For any $n \in \mathbb{N}$ and $p \in \mathbb{R}$,*

$$\sum_k C_n^k p^k = (1+p)^n, \qquad \sum_k C_n^k = 2^n, \tag{A.5.1}$$

$$\sum_k C_n^k k = n2^{n-1}, \qquad \sum_k C_n^k k^2 = n(n+1)2^{n-2}. \tag{A.5.2}$$

Corollary A.5.2 *For any $n \in \mathbb{N}$ and $a, c \in \mathbb{R}$,*

$$2^{-n} \sum_k C_n^k (c - ak) = c - an/2, 2^{-n} \sum_k C_n^k (n-2k)^2 = n.$$

Lemma A.5.3 *Let $|A| < \infty$, $n \in \mathbb{N}$. Then*

$$\sum_{x \in A^n} \prod_i f_i(x_i) = \prod_i \sum_{x_i \in A} f_i(x_i). \tag{A.5.3}$$

The graphs of some frequently used elementary functions are collected in Figure A.1.

Appendix B

Glossary of Acronyms

AC — adaptive computer
AI — artificial intelligence
AIP — adaptive information processor
AM — associative memory
AMDP — adaptive Markov decision process
ASA1 — approximate simulated annealing with one evaluation module.
ASA2 — approximate simulated annealing with two evaluation modules.
ANN — artificial neural network.
B- — zero diagonal symmetric feedback type (network, connection, property, structure)
BAC — basic adaptive computer
BD — Boltzmann distribution
BF-NN — feedback-feedforward neural network (generally not stable)
BM — Boltzmann machine
BNN — biological neural network
BP — backpropagation
BTT — backpropagation through time
C- — continuous type (network, activation rule, dynamics)
CAP — credit assignment problem
CE — cross entropy
CHD — constant heat dissipation
CL — competitive learning
CM — continuous machine (generalisation of TM)
CMC — controlled Markov chain
CNS — central nervous system (of brain)
CPU — central processing unit (of computer)
CTS — constant thermodynamic speed
D- — deterministic type (network, activation rule, dynamics)
DC- — D- and C-
DCB- — D- and C- and B-
DCB-NN — deterministic continuous feedback neural network (Hopfield net)
DCF- — D- and C- and F-
DCF-NN — deterministic continuous feedforward neural network (multilayer perceptron, backpropagation network)

DCFB- — D- and C- and FB-, generalisation of DCF- and DCB-
DCFB-NN — generalisation of both DCF-NN and DCB-NN
DP — dynamics programming
DPN — double precision numbers
DQ- — D- and Q-
DQB- — D- and Q- and B-
DQB-NN — deterministic quantised feedback neural network
DQF-NN — deterministic quantised feedforward neural network
DQFB-NN — deterministic quantised feedforward-feedback neural network (generalisation of both DQF-NN and DQB-NN)
ECM — extended continuous machine (generalisation of ETM and CM)
EP — entropy principle.
ESA — exact simulated annealing (method)
ETM — extended Turing machine (generalisation of TM)
F- — feedforward type (network, connection, property, structure)
FB- — feedforward-feedback type (network, property, structure)
FIP — fixed information processor
GA — genetic algorithms
GAC — general adaptive computer
GF — gradient-following (learning rules)
GS — Gibbs sampler
H. — homogeneous (neural network)
H.S.NN — homogeneous semilinear neural network
IC — iteration vs heat capacity diagram
IE — iteration vs average evaluation diagram
IG — information geometry
IS — iteration vs standard deviation of evaluation diagram
IT — iteration vs average tendency diagram
LA — learning automata
MAXENT — maximum entropy
MC — Markov chain
MDP — Markov decision process
ME — maximum evaluation
MF — mean field
ML — maximum likelihood
NN — neural network
NP-complete — a class of combinatorial problems generally considered hard to solve
P-coordinate — coordinate of a point on an information manifold which is proportional to probability distribution
PDP — parallel distributed processing
PGS — parameterised Gibbs sampler
PI — policy iteration (method)
PIP — programmable information processor
PRN — pseudo-random number
PRNG — pseudo-random number generator
PSA — parameterised simulated annealing
Q- — quantised type (network, activation rule, dynamics)
Q-learning — a reinforcement learning method developed by Watkins

Q.NN — quasilinear neural network
RAM — random access memory (of a digital computer)
RCE — reverse cross entropy
RL — reinforcement learning
RN — random number
RNG — random number generator
RTT — restricted Turing test
S- — stochastic type (network, activation rule, dynamics)
S. — semilinear (network)
S.NN — semilinear neural network
SA — simulated annealing
SA0 — simulated annealing to zero temperature
SA1 — simulated annealing to non-zero temperature
SC-NN — stochastic continuous neural network (generally impossible to be implemented)
SCAP — structural credit assignment problem
SCFB-NN — stochastic continuous feedforward-feedback neural network (generally impossible to be implemented)
SCM — stochastic continuous machine (generalisation of STM and CM)
SECM — stochastic extended continuous machine (generalisation of SCM, ECM. SETM)
SETM — stochastic extended Turing machine
SL — supervised learning
SPN — single procession number
SQ- — S- and Q-
SQB — S- and Q- and B-
SQB-NN — stochastic quantised feedback neural network (Boltzmann machine)
SQF-NN — stochastic quantised feedforward neural network (belief network, certain type of stochastic learning automata)
SQFB- — S- and Q- and FB-
SQFB-NN — stochastic quantised feedforward-feedback neural network (multilayer Boltzmann machine, generalisation of SQF-NN and SQB-NN)
STM — stochastic Turing machine (generalisation of TM)
TC — (inverse) temperature vs heat capacity diagram
TCAP — temporal credit assignment problem
T-coordinate — coordinate of a point on an information manifold which is affine-equivalent to tendency function
TD — temporal difference (method)
TE — (inverse) temperature vs average evaluation diagram
TIP — trainable information processor
TM — Turing machine
TNN — a general class of H.S.NN with tanh activation function
TS — (inverse) temperature vs standard deviation of evaluation diagram
TSP — travelling-salesman problem
TSP20 — the class of travelling-salesman problem with 20 cities
TT — Turing test
— (inverse) temperature vs average tendency diagram
UL — unsupervised learning

UTM — universal Turing machine
ESA — exact simulated annealing.
VI — value iteration (method).
VBM — variant Boltzmann machine

Bibliography

[AA87] J. Alspector and R. B Allen. A neuromorphic VLSI learning system. In P. Loseleben, editor, *Advanced Research in VLSI: Proceedings of the 1987 Stanford Conference*. MIT Press, 1987.

[Abr63] N. Abramson. *Information Theory and Coding.* McGraw-Hill, New York, 1963.

[Ada75] R. A. Adams. *Sobolev Spaces.* Academic Press, New York, 1975.

[AH89] S. Amari and T. S. Han. Statistical inference under multi-terminal rate restrictions: A differential geometric approach. *IEEE Trans. Inform. Theory*, 35(2):217–227, 1989.

[AHM+88] B. Andersen, K. H. Hoffman, K. Mosgaard, J. D. Nulton, J. M. Pedersen, and P. Salamon. On lumped models for thermodynamic properties of simulated annealing problems. *J. de Phys.*, 49:1485–1492, 1988.

[AHS85] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cog. Sci.*, 9:147–169, 1985.

[AK89a] E. Aarts and J. Korst. Boltzmann machine as a model for parallel annealing. *Algorithmica*, 6:437–465, 1989.

[AK89b] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines.* J. Wiley & Sons, Chichester, 1989.

[AKN92] S. Amari, K. Kurata, and H. Nagaoka. Information geometry of Boltzmann machines. *IEEE Trans. Neural Networks*, 3(2):260–271, 1992.

[Alm87] L. B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Proc. IEEE Intl. Conf. on Neural Networks*, pages 609–618, 1987.

[Alm89a] L. B. Almeida. Back propagation in non-feedforward networks. In I. Aleksander, editor, *Neural Computing Architecture: The design of brain-like machines.* North Oxford Academic Publishers, 1989.

[Alm89b] L. B. Almeida. Back propagation in perceptrons with feedback. In Eckmiller, R. and C. van der Malsburg [EvdM89].

[Ami89] D. J. Amit. *Modeling Brain Function: The World of Attractor Neural networks.* Cambridge University Press, Cambridge, 1989.

[And83] J. A. Anderson. Cognitive and psychological computation with neural models. In *IEEE Trans. Sys. Man and Cyber.* [Vem88], pages 799–815.

[And89] C. W. Anderson. Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine*, pages 31–36, April 1989.

[BA85] A. G. Barto and P. Anandan. Pattern-recognizing stochastic learning automata. *IEEE Trans. Sys. Man and Cyber.*, SMC-15(3):360–375, 1985.

[Bar90] A. G. Barto. Connectionist learning for control: An overview. In Miller, III et al. [MSW90], chapter 1.

[BD62] R. E. Bellman and S. E. Dreyfus. *Applied Dynamic Programming.* Princeton University Press, Princeton, NJ, 1962.

[Bel57] R. E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

[BGMT89] R. C. Brower, R. Giles, K. J. M. Moriarty, and P. Tamayo. Combining renormalization group and multigrid methods. *J. Comp. Phys.*, 80:472–479, 1989.

[BJ90] R. Beale and T. Jackson. *Neural Computing: An Introduction.* IOP Publishing, 1990.

[BL91] E. K. Blum and L. K. Li. Approximation theory and feedforward networks. *Neural Networks*, 4:511–515, 1991.

[BNCR86] O. E. Barndorff-Nielsen, D. R. Cox, and N. Reid. The role of differential geometry in statistical theory. *Intern. Stat. Rev.*, 54(1):83–96, 1986.

[Boc91] P. Bock. Building the ultimate machine — the emergence of artificial cognition. *Lecture Notes in Computer Science*, 496:472–481, 1991.

[BR92] A. L. Blum and R. L. Rivest. Training a 3-node neural network is NP-complete. *Neural Networks*, 5:117–127, 1992.

[BS50] H. W. Bode and C. E. Shannon. A simplified derivative of linear least square smoothing and prediction theory. *Proc. IRE*, 38:417–425, 1950. reprinted in [ET73].

[BSA83] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuron-like elements that can solve difficult learning control problems. *IEEE Trans. Sys. Man and Cyber.*, 13:834–846, 1983.

[BSW90] A. G. Barto, R. S. Sutton, and C. J. C. H. Watkins. Learning and sequential decision making. In Gabriel and Moore [GM90].

[BW88] E. B. Baum and F. Wilczek. Supervised learning of probability distributions by neural networks. In D. Z. Anderson, editor, *Neural Information Processing Systems.* Amer. Inst. Phys., New York, 1988.

[Cia89] P. G. Ciarlet. *Introduction to Numerical Linear Algebra and Optimization.* Cambridge University Press, Cambridge, 1989.

[CM59] H. Chernoff and L. E. Moses. *Elementary Decision Theory.* J. Wiley & Sons, New York, 1959.

[CM65] D. R. Cox and H. D. Miller. *The Theory of Stochastic Processes.* Methuen, 1965.

[Cun85] Y. Le Cun. Une procedure d'apprentissage pour reseau a sequil assymetrique (a learning procedure for asymmetric threshold network). *Proceedings of Cognitiva*, 85:599–604, 1985.

[Dav82] P. C. W. Davis. *The Accidental Universe.* Cambridge University Press, Cambridge, 1982.

[Dav87] L. Davis, editor. *Genetic Algorithms and Simulated Annealing.* Pitman, London, 1987.

[Day92] P. Dayan. The convergence of TD(λ) for general λ. *Machine Learning*, 8(3/4), 1992.

[dBM90] D. E. Van den Bout and T. K. Miller, III. Graph partitioning using annealed neural networks. *IEEE Trans. Neural Networks*, 1(2):192–203, 1990.

[Der70] C. Derman. *Finite State Markovian Decision Process.* Academic Press, New York, 1970.

[Die60] J. A. Dieudonné. *Foundations of Modern Analysis.* Academic Press, New York, 1960.

[DL76] S. E. Dreyfus and A. Law. *The Art and Theory of Dynamic Programming.* Academic Press, New York, 1976.

[Dre79] H. L. Dreyfus. *What Computers Can't Do: Revised edition.* Harper Colophon Books, New York, 1979.

[dRMT92] J. del R. Millán and C. Torras. A reinforcement connectionist approach to robot path finding in non-maze-like environments. *Machine Learning*, 8(3/4):363–395, 1992.

[DY79] E. B. Dynkin and A. A. Yushkevich. *Controlled Markov Processes.* Springer-Verlag, New York, 1979.

[ET73] A. Ephremides and J. B. Thomas. *Random Processes: Multiplicity theory and Canonical Decomposition.* Bench Mark Papers in Electrical Engineering and Computer Sciences. Dowden, Hutchinson &Ross, Stroudsburg, Pennsylvania, 1973.

[EvdM89] R. Eckmiller and C. van der Malsburg. *Neural Computers.* Proc. NATO Adv. Res. Workshop on Neural Computers. Springer-Verlag, Berlin, 1989.

[Fah88] S. E. Fahlman. Fast-learning variations on back-propagation: An empirical study. In D. S. Touretzky, G. E. Hinton, and Sejnowski, editors, *Connectionist Models Summer School: 1988 Proceedings*, pages 38–51, San Mateo, CA, 1988. Morgan Kaufmann.

[FF63] E. A. Feigenbaum and J. Feldman, editors. *Computers and Thought.* McGraw-Hill, New York, 1963.

[Fle87] R. Fletcher. *Practical Optimization.* J. Wiley & Sons, Chichester, 1987.

[FS90] J. A. Franklin and O. G. Selfridge. Some new directions for adaptive control theory in robotics. In Miller, III et al. [MSW90].

[Fun89] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.

[Gen89] C. T. Genís. Exploring three possibilities in network design: Spontaneous node activity, node plasticity and temporal coding. In Eckmiller, R. and C. van der Malsburg [EvdM89], pages 301–310.

[GG84] S. Geman and D. Geman. Stochastic relaxation, Gibbs distribution, and the Beyessian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 6:721–741, 1984.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[Gla63] R. J. Glauber. Time dependent statistics of the Ising model. *J. Math. Phys.*, 4:294–307, 1963.

[GM90] M. Gabriel and J. W. Moore, editors. *Learning and computational neuroscience: Foundations of adaptive networks.* MIT Press, Cambridge, MA, 1990.

[Gro86] S. Grossberg. *The Adaptive Brain I: Cognition, Learning, Reinforcement, and Rhythm, and The Adaptive Brain II: Vision, Speech, Language, and Motor Control.* Elsevier/North-Holland, Amsterdam, 1986.

[Gro88] S. Grossberg. Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural Networks*, 1:17–61, 1988.

[GS74] I. I. Gihman and A. V. Skorohod. *The Theory of Stochastic Processes*, volume 1. Springer-Verlag, Berlin, 1974.

[GS82] G. Grimmett and D. Stirzaker. *Probability and Random Processes.* Clarendon Press, Oxford, 1982.

[Gs88] C. T. Gení s. Relaxation and neural learning: Points of convergence and divergence. *J. Parallel & Distributed Computing*, 6:217–244, 1988.

[HCH91] K. H. Hoffman, M. Christoph, and M. Hanf. Optimizing simulated annealing. In *Parallel Computation from Nature*, volume 496 of *Lecture Notes in Computer Sciences*, 1991.

[Heb49] D. O. Hebb. *The Organization of Behavior.* J. Wiley & Sons, New York, 1949.

[HH64] J. M. Hammersley and D. C. Handscomb. *Monte Carlo Methods.* Methuen, London, 1964.

[Hin89a] G. E. Hinton. Connectionist learning procedures. *Artif. Intell.*, 40(1):185–234, 1989.

[Hin89b] G. E. Hinton. Deterministic Boltzmann machine learning performs steepest descent in weight space. *Neural Computation*, 1:143–150, 1989.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems.* Univ. of Michigan Press, Ann Arbor, 1975.

[Hop82] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. Nat. Acad. Sci. USA*, 79:2554–2558, 1982.

[Hop84] J. J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. Nat. Acad. Sci. USA*, 81:3088–3092, 1984.

[Hop89] J. J Hopfield. Learning algorithms and probability distributions in feed-forward and feed-back networks. *Proc. Nat. Acad. Sci. USA*, 84:8429–8433, 1989.

[Hor91] K. Hornik. approximation capability of multilayer feedforward networks. *Neural Networks*, 4:251–257, 1991.

[How60] R. A. Howard. *Dynamic Programming and Markov Processes.* MIT Press, New York, 1960.

[HS86] G. E. Hinton and T. J. Sejnowski. Learning and relearning in Boltzmann machines. In Rumelhart and McClelland [RM86], chapter 7.

[HSW89] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–368, 1989.

[HSW90] K. Hornik, M. Stinchcombe, and H. White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 3:551–560, 1990.

[HT86] J. J. Hopfield and D. W. Tank. Collective computation with continuous variables. In E. Bienenstock, F. Fogelman Soulie, and G. Weisbuch, editors, *Disordered Systems and Biological Organization.* Springer-Verlag, Berlin, 1986.

[Jay57a] E. T. Jaynes. Information theory and statistical mechanics. *Physics Review*, 106(4):620–630, 1957.

[Jay57b] E. T. Jaynes. Information theory and statistical mechanics. *Physics Review*, 108(2):171–190, 1957.

[KA89] J. Korst and E. Aarts. Combinatorial optimization on a Boltzmann machine. *J. Parallel & Distributed Computing*, 6:331–357, 1989.

[KGV83] C. Kirkpatrick, D. Gelat, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[Khi57] A. I. Khinchin. *Mathematical Foundation of Information Theory.* Dover Publications, New York, 1957.

[Knu81] D. E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, Reading (Mass.), 2 edition, 1981.

[Koh77] T. Kohonen. *Associative Memory: A System Theoretic Approach.* Springer-Verlag, Berlin, 1977.

[Koh84] T. Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, Berlin, 1984.

[Kol56a] A. N. Kolmogorov. *Foundations of the Theory of Probability.* Chelsea Publishing, New York, 1956. English translation, with added bibliography.

[Kol56b] A. N. Kolmogorov. On the Shannon theory of information transmission in the case of continuous signals. *IEEE Trans. Inform. Theory*, IT-2:102–108, 1956.

[Kul59] S. Kullback. *Information Theory and Statistics.* J. Wiley & Sons, New York, 1959.

[Lak81] S. Lakshmivarahan. *Learning Algorithms: Theory and Applications.* Springer-Verlag, New York, 1981.

[Lin92] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3/4):293–321, 1992.

[Lip87] R. P. Lippman. An introduction to computing with neural nets. In *IEEE ASSP magazine* [Vem88], pages 4–22.

[LL80] L. D. Landau and E. M. Lifshitz. *Statistical Physics.* Pergamon, Oxford, 1980.

[LL88] C.-Y. Liou and S.-L. Lin. The other variant boltzmann machine. In IJCNN, editor, *IJCNN Intl. Joint Conf. on Neural Networks*, volume 1, 1988.

[LTS90] E. Levine, N. Tishby, and S. Solla. A statistical approach to learning and generalization in layered neural networks. *Proc. IEEE*, 78(10):1568–1574, 1990.

[Mel92] R. Melzark. Phantom limbs. *Sci. Amer.*, pages 90–96, April 1992.

[MHP90] A. J. Maren, C. T. Harston, and R. M. Pap. *Handbook of Neural Computing Applications.* Academic Press, London, 1990.

[Min61] M. L. Minsky. Steps toward artificial intelligence. *Proc. IRE*, 49:8–30, 1961. Reprinted in [FF63].

[Min87] M. L. Minsky. *The Society of Mind.* William Heinemann, London, 1987.

[MP43] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[MP69] M. L. Minsky and S. A. Papert. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, 1969.

[MP88] M. L. Minsky and S. A. Papert. *Perceptrons: Expanded Edition.* MIT Press, Cambridge, 1988.

[MR86] J. L. McClelland and D. E. Rumelhart, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2. MIT Press, Cambridge, MA, 1986.

[MRR+53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of state calculations by fast computing machines. *J. Chem. Phys.*, 21:1087, 1953.

[MSW90] W. Thomas Miller, III, R. S. Sutton, and P. J. Werbos, editors. *Neural Networks for Control.* MIT Press, Cambridge, MA, 1990.

[MW87] I. Morgenstern and D. Würtz. Simulated annealing for "spin-glass-like" optimization problems. *Z. Phys. B*, 67:397–403, 1987.

[Nea92] R. M. Neal. Connectionist learning of belief networks. *Artif. Intell.*, 56:71–113, 1992.

[NL77] K. S. Narendra and S. Lakshmivarahan. Learning automata—a critique. *J. of Cyber. & Inf. Sci.*, 1:53–65, 1977.

[NS72] A. Newell and H. A. Simon. *Human Problem Solving.* Prentice Hall, Englewood Cliffs, NJ, 1972.

[NS88] J. D. Nulton and P. Salamon. Statistical mechanics of combinatorial optimization. *Phys. Rev.*, A37:1351–1356, 1988.

[NT74] K. S. Narendra and M. A. L Thathatcher. Learning automata—a survey. *IEEE Trans. Sys. Man and Cyber.*, 4:323–334, 1974.

[NT89] K. S. Narendra and M. A. L Thathatcher. *Learning Automata: An Introduction.* Prentice Hall, Englewood Cliffs, NJ, 1989.

[OR70] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations of Several Variables.* Academic Press, New York, 1970.

[OS75] A. V. Oppenheim and R. W. Schafer. *Digital Signal Processing.* Prentice Hall, Englewood Cliffs, NJ, 1975.

[PA87] C. Peterson and J. R. Anderson. A mean field algorithm for neural networks. *Complex Systems*, 1:995–1019, 1987.

[Par85] D. B. Parker. Learning logic. Technical Report TR-47, Massachusetts Institute of Technology, 1985.

[Pea87] J. Pearl. Evidential reasoning using stochastic simulation of causal models. *Artif. Intell.*, 32:254–257, 1987.

[Pen89] R. Penrose. *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics.* Oxford University Press, Oxford, 1989.

[Pen90] R. Penrose. Prècis of *the emperor's new mind: concerning computers, minds, and the laws of physics*. *Behav. Brain Sci.*, 1990.

[PH89] C. Peterson and E. Hartman. Exploration of the mean field theory learning algorithms. *Neural Networks*, 2:475–494, 1989.

[Pin87] F. J. Pineda. Generalization of backpropagation to recurrent neural networks. *Physics Review Letters*, 59:2229–2232, 1987.

[Pin88] F. J. Pineda. Dynamics and architecture for neural computation. *J. of Complexity*, 4:216–245, 1988.

[PM47] W. Pitts and W. S. McCulloch. How we know universals: The perception of auditory and visual forms. *Bulletin of Mathematical Biophysics*, 9:127–147, 1947.

[Pri81] M. B. Priestley. *Spectral Analysis and Time Series*. Academic Press, London, 1981.

[RHM86] D. E. Rumelhart, G. E. Hinton, and J. L. McClelland. A general framework for parallel distributed processing. In Rumelhart and McClelland [RM86], chapter 4.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In Rumelhart and McClelland [RM86], chapter 8.

[Rip87] B. D. Ripley. *Stochastic Simulations*. J. Wiley & Sons, Chichester, 1987.

[RIV91] A. K. Rigler, J. M. Irvine, and T. P. Vogl. Rescaling of variables in back propagation learning. *Neural Networks*, 4:225–229, 1991.

[RM86] D. E. Rumelhart and J. L. McClelland, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, Cambridge, MA, 1986.

[Ros58] F. Rosenblatt. The perceptron, a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[Ros59] R. Rosenblatt. *Principles of Neurodynamics*. Spartan Books, New York, 1959.

[Ros83] S. M. Ross. *Introduction to Stochastic Dynamic Programming*. Academic Press, New York, 1983.

[Sam59] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 3:210–229, 1959. Reprinted in [FF63, 71–105].

[Sam91] T. Samad. Back propagation with expected source values. *Neural Networks*, 4:615–618, 1991.

[SB90] R. S. Sutton and A. G. Barto. Time-derivative models of Pavlov reinforcement. In Gabriel and Moore [GM90].

[Sea80] J. R. Searle. Minds, brains and science. *Behav. Brain Sci.*, 3(3), 1980.

[Sej81] T. J. Sejnowski. Skeleton filters in the brain. In G. E. Hinton and J. R. Anderson, editors, *Parallel Models of Associative Memory*. Morgan Kaufmann, San Mateo, CA, 1981.

[Sen73a] E. Seneta. *Non-Negative Matrices: An Introduction to Theory and Applications*. George Allen & Unwin, London, 1973.

[Sen73b] E. Seneta. *Non-Negative Matrices and Markov Chains*. Springer-Verlag, Berlin, 1973.

[Sha48] C. E. Shannon. A mathematical theory of communication,I & II. *Bell Syst. Tech. J.*, 27:379–423, 623–656, 1948.

[Sha50] C. Shannon. Programming a digital computer for playing chess. *Phil. Mag.*, 41:356–375, 1950.

[Shv88] H. Shvaytser. Even simple neural nets cannot be trained reliably with a polynomial number of examples. In IJCNN, editor, *IJCNN Intl. Joint Conf. on Neural Networks*, volume 2, pages 141–146, 1988.

[Sin92] S. P. Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8(3/4):323–339, 1992.

[Slo92] A. Sloman. The emperor's real mind: Review of Roger Penrose's The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics. *Artif. Intell.*, 56:355–396, 1992.

[SNH+88] P. Salamon, J. D. Nulton, J. R. Harland, J. Pedersen, G. Ruppeiner, and L. Liao. Simulated annealing with constant thermodynamic speed. *Computer Phys. Commun.*, 49:423–428, 1988.

[SSF92] J. A. Scales, M. L. Smith, and T. L. Fisher. Global optimization methods for multimodal inverse problems. *J. Comp. Phys.*, 103(2):258–268, 1992.

[Sus92] H. J. Susmann. Uniqueness of the weights for minimal feedforward nets with a given input-output map. *Neural Networks*, 5:589–593, 1992.

[Sut84] R. S. Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, Dept. of Computer and Information Science, University of Massachusetts, Amherst, MA, 1984.

[Sut88] R. S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 1988.

[Sut90] R. S. Sutton. First results with dyna, an integrated architecture for learning, planning and reacting. In Miller, III et al. [MSW90], chapter 8.

[Szu86] H. Szu. Fast simulated annealing. In J. Denker, editor, *Neural Networks for Computing*, volume 151 of *AIP Conference Proceedings*, pages 420–425, Snowbird, UT, 1986. Amer. Inst. Phys.

[Tes92] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3/4):257–277, 1992.

[TH86] D. W. Tank and J. J. Hopfield. Simple "neural" optimization networks: An A/D converter, signal decision circuit, and a linear programming circuit. *IEEE Trans. Circ. Sys.*, 33(5):533–541, 1986.

[TTL84] Y. Tikochinsky, N. Tishby, and R. D. Levine. Alternative approach to maximum entropy inference. *Physics Review, A*, 30:2638–2644, 1984.

[Tur36] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proc. London Math. Soc.*, 42:230–265, 1936.

[Tur50] A. M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, October 1950. Reprinted in [FF63, pp. 17–36].

[Tur53] A. M. Turing. Digital computers applied to games. In B. V. Bowden, editor, *Faster than thought: A Symposium on Digital Computing Machines*, chapter 25, pages 286–310. Pitman, London, 1953.

[Vem88] V. Vemuri. Artificial neural networks: An introduction. In V. Vemuri, editor, *Artificial Neural Networks: Theoretical Concepts*, pages 1–12. IEEE, 1988.

[vNM47] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, Princeton, NJ., 1947.

[Was90] P. D. Wasserman. *NeuralSource: The Bibliographic Guide to Artificial Neural Networks.* Van Nostrand Reinhold, New York, 1990.

[Wat89] C. J. C. H. Watkins. *Learning with delayed rewards.* PhD thesis, Psychology Dept, Cambridge University, 1989.

[WB91] S. D. Whitehead and D. H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7:45–84, 1991.

[WD92] C. J. C. H. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8(3/4):279–292, 1992.

[Wer74] P. J. Werbos. *Beyond Regression: New tools for prediction and analysis in the behavioral science.* PhD thesis, Harvard University Committee on Applied Mathematics, 1974.

[Wer90a] P. J. Werbos. Backpropagation through time: What it is and how to do it. In *IEEE Proceedings*, pages 1550–1560, 1990.

[Wer90b] P. J. Werbos. Consistency of HDP applied to a simple reinforcement learning-problem. *Neural Networks*, 3(2):179–189, 1990.

[Wer90c] P. J. Werbos. A menu of designs for reinforcement learning over time. In Miller, III et al. [MSW90], chapter 3, pages 67–95.

[Wie48] N. Wienner. *Cybernetics.* Wiley, New York, 48.

[Wil87] R. J. Williams. Reinforcement-learning connectionist systems. Technical Report NU-CCS-87-3, College of Computer Science, Northeastern University, Boston, MA, 1987.

[Wil88] R. J. Williams. On the use of back propagation in associative reinforcement learning. In *Proc. IEEE Intl. Conf. on Neural Networks*, 1988.

[Wil90] R. J. Williams. Adaptive state representation and estimation using recurrent connectionist networks. In Miller, III et al. [MSW90], chapter 4.

[Wil92] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3/4):229–256, 1992.

[Wit77] I. H. Witten. An adaptive optimal controller for discrete-time Markov environment. *Information and Control*, 34:286–295, 1977.

[WRS89] T. Ward, P. A. S. Ralson, and K. E. Stoll. Intelligent control of machine and processes. *Computers & Ind. Engng.*, 19(1-4):205–209, 1989.

[YG90] E. Yair and A. Gersho. The Boltzmann perceptron network: A soft classifier. *Neural Networks*, pages 203–231, 1990.

[Yor92] J. York. Use of the gibbs samplers in expert systems. *Artif. Intell.*, 56:115–130, 1992. And Addendum, 397-398.