Stochastic Recurrent Networks Training by the Local Backward-Forward Algorithm

Athanasios Kehagias
Division of Applied Mathematics
Brown University
Providence, RI 02912

e-mail: st401843@brownvm.brown.edu

March 4, 1991

Abstract

We introduce *Stochastic Recurrent Networks*, which are collections of interconnected finite state units. At every discrete time step, each unit goes into a new state, following a probability law that is conditional on the state of neighboring units at the previous time step. A network of this type can *learn* a stochastic process, where "learning" means maximizing the probability Likelihood function of the model. A new learning (i.e. Likelihood maximization) algorithm is introduced, the *Local Backward-Forward Algorithm.* The new algorithm is based on the Baum Backward-Forward Algorithm (for Hidden Markov Models) and improves speed of learning substantially. Essentially, the local Backward-Forward Algorithm is a version of Baum's algorithm which estimates local transition probabilities rather than the global transition probability matrix. Using the local BF algorithm, we train SRN's that solve the 8-3-8 encoder problem and the phoneme modelling problem.

Contents

NOTATION

Given a finite set A, we denote the number of elements in A by $|A|$. E.g., for $A = \{a_1, ..., a_N\}$, $|A| = N$. The **alphabet** A of a stochastic process $\{Z^t\}_{t=1}^{\infty}$ is the set of all possible values that Z^t can take for any t. E.g. we could have a *binary* stochastic process $\{Z^t\}_{t=1}^{\infty}$ where Z^t equals either 0 or 1 for every t. In that case the alphabet is $A = \{0, 1\}$. Or, we could have a *vector-binary* process $\{Z^t\}_{t=1}^{\infty}$, where $Z^t = [Z_1^t ... Z_N^t]$ and Z_n^t is either 0 or 1 for every t, $n = 1, ..., N$. In that case the alphabet is $\{0,1\}^N \doteq \{[a_1 ... a_N] : a_n \in \{0,1\}, n = 1, ..., N\}$. In general, given a finite alphabet A, we define $A^N \doteq \{[a_1 ... a_N] : a_n \in A, n = 1, ..., N\}$.

We use capital letters X, Y, Z etc. for stochastic processes and small letters x, y, z for the values of the processes (characters of the alphabet). For instance we write $Prob(X^t = x)$ for the probability that X^t equals the character $x \in A$; we write $Prob(X^{t+1} = x^1, ..., X^{t+\tau} = x^\tau)$ for the probability that $X^{t+1} ... X^{t+\tau}$ equals $x^1 ... x^\tau$, $x^1, ..., x^\tau \in A^\tau$. Say $x = [x^1 ... x^m]$, $y = [y^1 ... y^n]$; then the **concatenation** of x, y is denoted by xy and defined by $xy \doteq [x^1 ... x^m y^1 ... y^n]$.

We will often consider probabilities that depend on the value of a certain parameter, say $\mathcal{P}$. To explicitly denote the dependence, we write: $Prob(X^{t+1} = x^1, ..., X^{t+\tau} = x^\tau; \mathcal{P})$.

We often consider a set $S = \{1, ..., N\}$ and vector $x = [x^1 ... x^N]$. Sometimes we write x_S instead of x. Similarly, for a set $R = \{r_1, ..., r_M\} \subset S$ we write x_R in place of $[x_{r_1} ... x_{r_M}]$. For example, suppose $S = \{1, 2, 3\}$, $R = \{1, 2\}$, we have $x_S = [x_1, x_2, x_3]$, $x_R = [x_1, x_2]$. Obviously, if $x_s \in A$ for $s \in S$, then $x_S \in A^{|S|}$.

On several occasions we use sums of the form $\sum_{n=1}^{N} w_n x_n$. We sometimes write these using **inner product** notation: $< w, x > \doteq \sum_{n=1}^{N} w_n x_n$. Finally, $Bin(m, n)$, where m is an integer, means the n-th digit of integer m written in binary notation.

1 Introduction

The learning and reproduction of temporal behavior have a special place in connectionism. A brief inspection of the connectionist literature reveals many cases where we apply neural networks with dynamic behavior to the learning of static input/output patterns (e.g. [Hop82, Hop85, Gro86a, Gro86b]). In such cases we are essentially trying to shape the equilibrium behavior of the network. Other times we use static networks (essentially nonlinear regressors) to learn temporal behavior (e.g. [W+89a, W+89b],). The use of recurrent networks for the learning of dynamically evolving temporal relationships is a relatively new but fast growing area of connectionism. For some examples see [Alm87, Alm 88, Alm89, BW88, BW89, Elm88, Gil90, Pin88, Pol87, R+88, Roh90, RR90, WZ88, Jo86, Sut88].

An important example of a stochastic neural network is the Boltzmann Machine [A+85]. In this case, a stochastic network is used to learn a deterministic input/output relationship. This is also true of other stochastic networks (see [Sun89, Ale89]). There is some work done on stochastic networks and, in particular, the modelling of probabilistic input/output relationships (see [Ama71, Ama72, Ama83, Lin88a, Lin88b, Pea86, Sol88]) but it is probably safe to say that stochastic networks are outside the mainstream of connectionist research. However, note also the work of Hinton and his group; they have produced a large number of papers dealing with stochastic networks. [Nea90, Now90a, Now90b, Now90c] are of particular relevance to our point of view, as will be explained later.

This is the first in a sequence of papers [Keh1, Keh2, Keh3] that develop a theory of **Stochastic Recurrent Networks** (henceforth **SRN**). We try to develop a general type of network that combines stochastic and dynamic behavior. In our framework, deterministic and/or static behavior appear as special cases. We look at SRN's from a computational point of view. We are not concerned with issues of biological plausibility. What we attempt here is to develop a model that implements parallel distributed computing and can be trained very quickly.

We will give a precise definition of SRN in the next section; informally, a SRN is *a collection of interconnected finite state units that change states synchronously, according to a stochastic mechanism*. For any particular unit, the stochastic change of state depends on the previous state of this unit and its *parents*. We only deal with finite state units; the theory for units with continuous valued states is exactly analogous but mathematically more involved and will be developed elsewhere. In developing the SRN theory we rely heavily on two points of view: (a) that of *Stochastic Control* [May82] and (b) that of *Hidden Markov Modelling* [L+83].

Stochastic Control deals mostly with continuous valued dynamical systems; the problems addressed within this context are very similar to connectionist problems such as prediction and classification. In addition, Stochastic Control is a mature discipline with a large toolkit of theoretical and applied methods. We will try to apply some of these methods to the study of our finite state SRN's.

The Hidden Markov Modelling point of view is used in developing a new fast training algorithm, the **local Backward-Forward** algorithm. This is a modification of the celebrated Backward Forward algorithm of Baum [Bau72,B+70,BE67, L+83]. Both of these algorithms are fast: convergence to a locally Maximum Likelihood solution takes place in very few iterations (on the order of ten). For discussions of the connections between connectionist models and HMM's see [Be+90a, Be+90b, W+89a, W+89b].

In [Keh2] we will prove the following equivalence results.

1. Every SRN has a HMM description (in terms of global transition and emission matrices)

2. For every HMM there is a SRN so that they both have the same input/output behavior.

3. Every connectionist network of the classical type (sigmoid units with linear combination inputs) can be considered as a SRN.

4. Every SRN can be implemented as a network of nonlinear (but not necessarily sigmoid) units with random input/output behavior.

Some of these facts are obvious (e.g. 1 and 3); others will be briefly discussed in this paper as appropriate. The complete story about equivalence of the three types of models (SRN, HMM and calssical connectionist) will be told in [Keh2], along with certain representation results. We mention these facts here to point out that our SRN model is a universal model which includes both HMM and connectionist models. In particular, we can say that SRN is a *local* implementation of Hidden Markov Modelling. It is local, as will become apparent in the next section, because we specify the model in terms of *local conditional transition probabilities*, instead of the traditional description of HMM's in terms of *global* transtion and emission probability matrices (see [L+83]).

In this paper we do the following: give a precise definition of the SRN model (Section 2), propose and discuss a basic learning task for stochastic processes (Section 3) and finally derive the Local Backward Forward Algorithm (Section 4). Then we present some training examples (Section 5) and conclude with a discussion section (Section 6).

There are also four appendices. In Appendix A we compare the new SRN model to the classical connectionist SRN model with sigmoidal units and show how we can build any SRN from a collection of nonlinear units with random input. In Appendix B we discuss the EM method [D+77]. The EM method is a general approach to learning/estimation problems; the local BF algorithm is a special case of EM. We show that Relaxation training (for Boltzmann Machines) is also a special case of EM. In Appendix C we consider SRN from the point of view of Hidden Markov Modelling. In Appendix D we take a very brief review of disciplines (other than connectionism) which consider objects very similar to SRN's and consider what can be learned from these alternative views. Generally speaking, the subject of recurrent, finite state networks is very popular and has been examined by many different reasearch communities outside of connectionism. These connections are explored in much greater detail in [Keh3].

Let us also mention briefly the content of the papers following in the SRN sequence [Keh1, Keh2, Keh3]. In [Keh1] we develop a general theory of estimation and classification of stochastic processes; our starting point is the basic model, learning task and training algorithm introduced here. In [Keh2] we prove certain theoretical results about representation power of SRN's and consistency (in the statistical sense) of Maximum Likelihood estimation. In [Keh3] we consider similarities of our model with models used in other research contexts and use these alternative viewpoints to motivate some extensions of our methods.

2 The SRN Model

(We assume the reader is familiar with the concepts of a random variable and a stochastic process.)

Here we develop a type of **Stochastic Recurrent Network** (**SRN**) which consists of a collection of interconnected units. This network receives an **input** stochastic process $\{U^t\}_{t=1}^{\infty}$ and generates a **hidden** stochastic process $\{X^t\}_{t=1}^{\infty}$ and an **output** stochastic process $\{Y^t\}_{t=1}^{\infty}$. In this paper all stochastic processes (input, hidden and output) are vectors with their components taking values in a finite **alphabet** A. Several authors have considered very similar models (see [Sun89, Nea90]) .

The SRN model introduced here is somewhat different from classical connectionist recurrent networks, in that it does not consist of units with sigmoid input/output response. The classical model is a special case of ours; on the other hand *any* SRN can be implemented as a classical connectionist model with nonlinear units. This point is discussed in Appendix A.

Let us first give an informal description of the mechanism that generates the hidden and output processes. (Later we will give a mathematically precise description.) Consider a network of interconnected units. For simplicity of presentation assume each unit is binary, that is, it can be either on or off. When the unit is off its state is 0; when the unit is on its state is 1. Following standard connectionist practice, the units are separated in three layers: input, hidden and output. The state of the network can be fully described by three vectors of 0's and 1's: one vector for each layer, one vector component for each unit. At a given time $t-1$ every unit of the network is in some state. At time t a new *epoch* starts, during which the units update their state as follows.

1. First the input units turn on or off with a probability that is independent of the other network units and completely determined by the external environment.

2. Then each of the hidden units receives as input the states of its "parents" (which can be input or hidden units, including that same unit, but *not* output units). Depending on the configuration of its parents, there is a certain probability that each hidden unit will turn on or off.

3. Finally, each of the output units receives as input the states of its parents (which can be input or hidden units, but *not* output units) and turns on or off with a configuration-dependent probability. At this point all units in the network have updated their state and epoch t is complete.

Let us now formulate this mechanism in a mathematically precise manner. Call U^t the M_i-long state vector of input units at time t, X^t the M_h-long state vector of hidden units at time t, Y^t the M_o-long state vector of output units at time t. The components of these vectors all come from the same finite set $A = \{0, 1, ..., K-1\}$. The sequences $\{U^t\}_{t=1}^{\infty}$, $\{X^t\}_{t=1}^{\infty}$ and $\{Y^t\}_{t=1}^{\infty}$ are the input, hidden and output stochastic processes, respectively.

We want a specification of a SRN which, together with the initial state X^0 and the input process $\{U^t\}_{t=1}^{\infty}$, will describe the hidden process $\{X^t\}_{t=1}^{\infty}$ and the output process $\{Y^t\}_{t=1}^{\infty}$. The processes $\{U^t\}_{t=1}^{\infty}$, $\{X^t\}_{t=1}^{\infty}$, $\{Y^t\}_{t=1}^{\infty}$ all take values in finite alphabets (A^{M_i}, A^{M_h}, A^{M_o} respectively). Hence they are fully described by their **probability functions**:

$$p_U(u^1...u^m) \doteq Prob(U^1 = u^1...U_m = u^m) \qquad \forall m, \forall u^1, ..., u^m \in A^{M_i},$$

$$p_X(x^1...x^m) \doteq Prob(X^1 = x^1...X_m = x^m) \qquad \forall m, \forall x^1, ..., x^m \in A^{M_h},$$

$$p_Y(y^1...y^m) \doteq Prob(Y_1 = y^1...Y_m = y^m) \qquad \forall m, \forall y^1, ..., y^m \in A^{M_o}.$$

The SRN will be specified in terms of a *directed graph* $\mathcal{G}$ and a set of *local conditional probabilities* $\mathcal{P}$. Thus, a stochastic recurrent network is a pair $(\mathcal{G}, \mathcal{P})$; given $(\mathcal{G}, \mathcal{P})$, p_U and an initial condition X^0, we can compute $p_X(x^1, ..., x^m)$, $p_Y(y^1, ..., y^m)$ for all m, x^1, ..., x^m, y^1, ..., y^m.

The directed graph $\mathcal{G}$ is itself a pair $\mathcal{G} = (S, \mathcal{N})$, where $S = \{s_1, ..., s_M\}$ is the collection of units (or nodes, to use the graph theoretic term). The unit set S is partitioned into three mutually exclusive sets: $S = S_i \cup S_h \cup S_o$, where S_i is the set of input units, S_h is the set of hidden units, S_o is the set of output units. We have $|S_i| = M_i$, $|S_h| = M_h$, $|S_o| = M_o$. Note that $M = M_i + M_h + M_o$.

Take any two units $r, s \in S$. If s reads the state of r before changing its own state then there is a directed edge from unit r to unit s. In such a case we say that r is a **parent** of s. A unit $s \in S$ can have none, one or many parents and even be a parent of itself. The set of s's parents is indicated by $N(s)$ and the class of all parent sets is denoted by $\mathcal{N} \doteq \{N(s), s \in S\}$.

$(S, \mathcal{N})$ is a complete description of the **topology** of the net. We assume the SRN topology satisfies the following restriction. The parent set of every unit can be partitioned as follows:

$$\forall s \in S_i \qquad N(s) = \emptyset,$$

$$\forall s \in S_h \qquad N(s) = N_i(s) \cup N_h(s) \text{ where } N_i(s) \subset S_i \text{ and } N_h(s) \subset S_h,$$

$$\forall s \in S_o \qquad N(s) = N_i(s) \cup N_h(s) \text{ where } N_i(s) \subset S_i \text{ and } N_h(s) \subset S_h.$$

In words, this means that input units receive no input, while hidden and output units receive input only from input and hidden units. This completes the description of the topology of the network.

The probabilistic state update mechanism is described by $\mathcal{P}$, which is the set of *local conditional probabilities*. As already described, the state update takes place synchronously and locally for every unit. Mathematically, this is reflected in the Markovian factorization of joint probabilities:

$$Prob(X^t = x^0 | X^{t-1} = x^{-1}, X^{t-2} = x^{-2}, ..., U^t = u^0, U^{t-1} = u^{-1}, U^{t-2} = u^{-2} ...) =$$
$$\prod_{s \in S_h} Prob(X_s^t = x_s^0 | X_{N_h(s)}^{t-1} = x_{N_h(s)}^{-1}, U_{N_i(s)}^t = u_{N_i(s)}^0). \tag{1}$$

Similarly,

$$Prob(Y^t = y^0 | ..., X^{t+1} = x^1, X^t = x^0, X^{t-1} = x^{-1}, ..., U^{t+1} = u^1, U^t = u^0, U^{t-1} = u^{-1}, ...) =$$
$$\prod_{s \in S_o} Prob(Y_s^t = y_s^0 | X_{N_h(s)}^t = x_{N_h(s)}^0, U_{N_i(s)}^t = u_{N_i(s)}^0). \tag{2}$$

Remark: Equation (1) says that, given all the past history of the hidden state and input, we gain no more information about the next hidden state , than if we knew just the immediate past. In addition, the local state of any unit depends only on the states of its parents. Similarly, equation (2) says that, given all the history of the hidden state and input (past, present and future) we gain no more information about the present output, than if we knew just the present hidden state and input. In addition, the local state of any unit depends only on the states of its parents.

Remark: Note that (2) is a stronger statement than

$$Prob(Y^t = y^0 | X^t = x^0, X^{t-1} = x^{-1}, ..., U^t = u^0, U^{t-1} = u^{-1}, ...) =$$
$$\prod_{s \in S_o} Prob(Y_s^t = y_s^0 | X_{N_h(s)}^t = x_{N_h(s)}^0, U_{N_i(s)}^t = u_{N_i(s)}^0). \tag{3}$$

Equation (3) says that knowledge of all the *past and present* of hidden state and input gives no more information on the value of the present output. But (2) says that even knowledge of *past, present and future* gives no more information.

The process $\{X^t\}_{t=1}^{\infty}$ is obviously Markov, not only in time, but also *locally* within the network. Therefore, if we define the **local conditional probabilities** for all $s \in S_h \cup S_o$, $a \in A$, $b \in A^{|N_h(s)|}$, $c \in A^{|N_i(s)|}$

$$p_s(a|b, c) \doteq Prob(X_s^t = a | X_{N_h(s)}^{t-1} = b, U_{N_i(s)}^t = c)$$

we can compute the probability $Prob(X^t|X^{t-1}, U^t)$ in terms of the local conditionals:

$$Prob(X^t = x^0|X^{t-1} = x^{-1}, X^{t-2} = x^{-2}, ..., U^t = u^0, U^{t-1} = u^{-1}, ...) =$$

$$\prod_{s\in S_h} p_s(x_s^0|x_{N_h(s)}^{-1}, u_{N_i(s)}^0).$$

Similarly we can compute

$$Prob(Y^t = y^0|..., X^{t+1} = x^1, X^t = x^0, X^{t-1} = x^{-1}, ..., U^{t+1} = u^1, U^t = u^0, U^{t-1} = u^{-1}, ...) =$$

$$\prod_{s\in S_o} p_s(y_s^0|x_{N_h(s)}^0, u_{N_i(s)}^0).$$

The set $\mathcal{P}$ is the set of all the local conditionals:

$$\mathcal{P} \doteq \left\{p_s(a|b,c), s \in S_h \cup S_o, a \in A, b \in A^{|N_h(s)|}, c \in A^{|N_o(s)|}\right\}.$$

Now suppose $((S,\mathcal{N}),\mathcal{P})$, p_U, $Prob(X^0)$ are known. We will first compute $p_X(x^1...x^m)$. We have

$$p_X(x^1...x^m) = Prob(X^1...X^m = x^1...x^m) =$$

$$\sum_{x^0\in A^{M_h}, u^1,...,u^m\in A^{M_i}} Prob(X^0...X^m = x^0...x^m, U^1...U^m = u^1...u^m) =$$

$$\sum_{x^0\in A^{M_h}, u^1,...,u^m\in A^{M_i}} \left(\prod_{t=1}^m \prod_{s\in S_h} p_s(x_s^t|x_{N_h(s)}^{t-1}, u_{N_i(s)}^t)\right) p_U(u^1...u^m) Prob(X^0 = x^0).$$

Similarly we can compute $p_Y(y^1...y^m)$:

$$p_Y(y^1...y^m) = Prob(Y^1...Y^m = y^1...y^m) =$$

$$\sum_{x^0,x^1,...,x^m\in A^{M_h}, u^1,...,u^m\in A^{M_i}} Prob(Y^1..Y^m = y^1..y^m, X^0..X^m = x^0..x^m, U^1..U^m = u^1..u^m) =$$

$$\sum_{x^0,x^1,...,x^m\in A^{M_h}, u^1,...,u^m\in A^{M_i}} \left(\prod_{t=1}^m \prod_{s\in S_h\cup S_o} p_s(x_s^t|x_{N_h(s)}^{t-1}, u_{N_i(s)}^t)\right) p_U(u^1...u^m) Prob(X^0 = x^0).$$

This completes the computation of $p_X(y^1...y^m)$ and $p_Y(y^1...y^m)$. This computation can be done for any m, $y^1, ..., y^m$, $x^1, ..., x^m$. Hence we have shown that $(\mathcal{G},\mathcal{P})$, p_U and $Prob(X^0)$ are sufficient to determine p_X, p_Y.

Remark: This model can be fully implemented by a network of nonlinear input/output units with additional white noise input:

$$X^t = f(X^{t-1}, U^t, V^t),$$

$$Y^t = g(X^t, U^t, W^t).$$

This is proven in Appendix A. Deterministic behavior can be obtained as a special case, when the noise has zero variance. There is a formal similarity of the equations above with the equations of a continuously valued stochastic control system. This formal similarity suggests that we apply stochastic control methods to solve connectionist problems such as prediction and classification. This is discussed briefly at the end of the next section and in more detail in [Keh1].

3 Learning a Stochastic Process

"Learning" a stochastic process $\{Y^t\}_{t=1}^{\infty}$, in the context of SRN, can mean several different things.

1. **Modelling** the process, i.e. building a SRN $(\mathcal{G}, \mathcal{P})$ such that its output $\{\hat{Y}^t\}_{t=1}^{\infty}$ is the "same" as the original process $\{Y^t\}_{t=1}^{\infty}$.

2. **Prediction**, i.e. building a network $(\mathcal{G}, \mathcal{P})$ which receives the values $..., Y^{t-1}, Y^t$ as input and produces Y^{t+1} as output.

3. **Classification**, i.e. building a network that receives the values $..., Y^{t-1}, Y^t$ as input and produces as output a guess as to which of several candidate models actually produced the observations.

Also, we need not be limited to a single, "output" process. Many times we are interested in learning an input/output relationship between two processes. Consider two processes $\{U^t\}_{t=1}^{\infty}$ and $\{Y^t\}_{t=1}^{\infty}$. Take the modelling problem: it consists in finding a network $(\mathcal{G}, \mathcal{P})$ that will take input $\{U^t\}_{t=1}^{\infty}$ and produce output $\{\hat{Y}^t\}_{t=1}^{\infty}$, which is "close" to $\{Y^t\}_{t=1}^{\infty}$. We can similarly redefine the prediction and classification problem. In fact we can consider the "output-only" problem as a special case of the input/output problem, assuming the input layer of the network to be empty. This will be clarified by the phoneme modelling example of Section 5.

In this paper we will only consider the modelling problem. This is useful in itself, as we will show by applications to the encoder problem, modelling of phonemes etc. (Section 5). But it is also useful as a stepping stone in solving the prediction (more generally, estimation) problem and the classification problem. However these two problems will be tackled in a future paper [Keh1].

In particular we will consider Maximum Likelihood modelling. Given a graph topology $\mathcal{G}$, an initial condition $X^0 = x^0$, a sample $u^1, ..., u^n$ from an input process $\{U^t\}_{t=1}^{\infty}$ and a sample $y^1, ..., y^n$ from an output process $\{Y^t\}_{t=1}^{\infty}$, we want to find a set of local conditionals $\hat{\mathcal{P}}$ such that

$$L(\mathcal{P}) \doteq Prob(Y^1 = y^1, ..., Y^n = y^n | X^0 = x^0, U^1 = u^1, ..., U^n = u^n; (\mathcal{G}, \mathcal{P}))$$

is maximized at $\mathcal{P} = \hat{\mathcal{P}}$. Note that we assume some fixed $\mathcal{G} = (S, \mathcal{N})$ on which $\mathcal{P}$ lives. We choose $\mathcal{P}$ by Maximum Likelihood training; the local BF algorithm provides a rather easy solution to this problem. The choice of $(S, \mathcal{N})$, on the other hand, is much harder; in [Keh2] we will consider this question in somewhat more detail, but there are essentially no easy answers.

This completes the mathematical specification of the modelling problem. The reader may want to skip to Section 5 where he can study a few examples of modelling problems. Before we finish this section, let us mention briefly how we intend to use modelling for prediction and classification (in [Keh1]) and how our methods are different from classical connectionist method.

Take for example the prediction problem. Prediction in connectionism is usually done by nonlinear regression. One builds a model that at time t receives as input past values $..., Y^{t-1}$, Y^t and produces $\hat{Y}^{t+1}$ as a prediction of the next value $\hat{Y}^{t+1}$. This is an essentially *static* (feedforward) estimator, very similar to FIR filters in Signal Processing, and is not necessarily the best solution to the prediction problem. In stochastic control [Aok87], Signal Processing

(IIR filters) as well as in Statistics [BD87], much improved estimates are obtained by the use of dynamical (feedback) estimators, most notably the Kalman filter [Kal60]. Probably the most attractive characteristic of the Kalman filter is its recursive nature. In the context of continuous valued stochastic processes, the recursive computation of Kalman filtering only applies to linear systems. However, as we will show in [Keh1], for discrete valued processes we can implement a recursive analog of Kalman filtering for any SRN.

Similarly, consider classification. In the connectionist literature classifiers typically perform a nonlinear regression on the observables $..., Y^{t-1}, Y^t$. Even when we are dealing with an essentially dynamic problem, such as speech, it is not unusual to use static networks ([W+89a, W+89b] - but see also [Rob88, Rob89, BW88, BW89] for dynamic networks). However, in Stochastic Control Theory, especially in System Identification and Adaptive Control dynamical methods of classification, are available, most notably the Lainiotis algorithm [Lai71]. In [Keh1] we use the Lainiotis algorithm for classification of discrete state SRN's. Note that the Lainiotis algorithm is (for dynamic processes) the analog of Nowlan's *competing experts* method [Now90b].

4 The Local Backward Forward Algorithm

In this section we develop the *Local Backward-Forward* algorithm (henceforth *local BF*) to solve the Maximum Likelihood learning problem for SRN's. This algorithm is a modification of the Baum Backward-Forward algorithm [Bau72, BE67] used in Speech Recogntion [L+83, Rab88]. It is an alternative to Back-Propagation [Alm87, Alm88, Alm89, AN89, Pin88] or Relaxation type methods (used in training Boltzmann Machines - e.g. see [A+85]) and its basic advantage is great speed of learning.

Recall the **Maximum Likelihood Learning** problem. We are given an initial condition x^0, input sample $u^1, u^2, ..., u^T$ and output sample $y^1, y^2, ..., y^T$ which will be considered fixed for the rest of the discussion. We are also given a fixed network topology $\mathcal{G} = (S, \mathcal{N})$. Now we want to select a set of local conditionals $\mathcal{P}$ such that the Likelihood function $L(\mathcal{P})$ is maximized; where $L(\mathcal{P})$ is defined to be:

$$L(\mathcal{P}) \doteq Prob(Y^1 = y^1, ..., Y^T = y^T | X^0 = x^0, U^1 = u^1, ..., U^T = u^T; (\mathcal{G}, \mathcal{P})).$$

To solve the Maximum Likelihood Learning problem, we will now develop the local BF algorithm. First define some useful quantities:

$$\Psi_{\mathcal{P}}(x^1, ..., x^T) \doteq$$

$$Prob(Y^1 = y^1, ..., Y^T = y^T, X^1 = x^1, ..., X^T = x^T | X^0 = x^0, U^1 = u^1, ..., U^T = u^T; (\mathcal{G}, \mathcal{P})).$$

Note that we have $\sum_{x^1..x^T \in A^{M_h}} \Psi_{\mathcal{P}}(x^1..x^T) = L(\mathcal{P})$. We also define:

$$\Phi(\mathcal{P}, \mathcal{Q}) \doteq \sum_{x^1...x^T \in A^{M_h}} \Psi_{\mathcal{P}}(x^1, ..., x^T) \log \Psi_{\mathcal{Q}}(x^1, ..., x^T). \tag{4}$$

Now we will prove the following theorem:

Theorem 1 *(Baum's Theorem -* [BE67]*) Suppose* $\Phi(\mathcal{P}, \mathcal{Q}) \geq \Phi(\mathcal{P}, \mathcal{P})$. *Then* $L(\mathcal{Q}) \geq L(\mathcal{P})$.

Proof:

$$\log \frac{L(\mathcal{Q})}{L(\mathcal{P})} = \log \sum_{x^1..x^T} \frac{\Psi_{\mathcal{Q}}(x^1..x^T)}{L(\mathcal{P})} = \log \sum_{x^1..x^T} \frac{\Psi_{\mathcal{P}}(x^1..x^T)}{L(\mathcal{P})} \cdot \frac{\Psi_{\mathcal{Q}}(x^1..x^T)}{\Psi_{\mathcal{P}}(x^1..x^T)} \tag{5}$$

Now, we have $\sum_{x^1..x^T} \Psi_{\mathcal{P}}(x^1..x^T)/L(\mathcal{P}) = 1$ so (5) is a convex combination; also $\log(.)$ is a concave function. Hence (by Jensen's Theorem):

$$\log \frac{L(\mathcal{Q})}{L(\mathcal{P})} \geq \sum_{x^1..x^T} \frac{\Psi_{\mathcal{P}}(x^1..x^T)}{L(\mathcal{P})} \log \frac{\Psi_{\mathcal{Q}}(x^1..x^T)}{\Psi_{\mathcal{P}}(x^1..x^T)} = \frac{1}{L(\mathcal{P})} \left(\Phi(\mathcal{P},\mathcal{Q}) - \Phi(\mathcal{P},\mathcal{P})\right) \geq 0. \tag{6}$$

This completes the proof. •

Baum's Theorem forms the basis of a class of ML algorithms (see [BE67, Bau72]). Choose $\mathcal{P}_0$ arbitrarily, then maximize $\Phi(\mathcal{P}_0, \mathcal{P})$ with respect to $\mathcal{P}$; call the maximizer $\mathcal{P}_1$. Obviously $\Phi(\mathcal{P}_0, \mathcal{P}_1) \geq \Phi(\mathcal{P}_0, \mathcal{P}_0)$, hence also $L(\mathcal{P}_1) \geq L(\mathcal{P}_0)$. Now maximize $\Phi(\mathcal{P}_1, \mathcal{P})$ with respect to $\mathcal{P}$; caal the maximizer $\mathcal{P}_2$. Obviously $\Phi(\mathcal{P}_1, \mathcal{P}_2) \geq \Phi(\mathcal{P}_1, \mathcal{P}_1)$, hence also $L(\mathcal{P}_2) \geq L(\mathcal{P}_1) \geq L(\mathcal{P}_0)$. Proceeding in this manner we get a sequence $\mathcal{P}_0, \mathcal{P}_1, ...$ such that

$$L(\mathcal{P}_0) \leq L(\mathcal{P}_1) \leq L(\mathcal{P}_2) \leq ....$$

This is an iterative, greedy algorithm: at every iteration the value of the Likelihood is increased. It has been proven [Wu83] that this procedure guarantees convergence to a local maximum. Convergence to the global maximum is not guaranteed. In all these respects the algorithm is similar to a steepest ascent procedure. However there is one important difference: at every step m we can explicitly compute the maximizer p_s^m (as we will show presently) and hence are not confined to a small step $|p_s^m - p_s^{m+1}|$.

We want to minimize the function $\Phi(\mathcal{P}, \mathcal{Q})$ with respect to $\mathcal{Q}$, where $\mathcal{P} = \{p_s, s \in S_h \cup S_o\}$, $\mathcal{Q} = \{q_s, s \in S_h \cup S_o\}$ are two sets of local conditionals. So we want to minimize $\Phi(\mathcal{P}, \mathcal{Q})$ with respect to $q_s(a|b,c)$, $s \in S$, $a \in A$, $b \in A^{|N_h(s)|}$, $c \in A^{|N_i(s)|}$. These are real variables in the range $[0,1]$; however they are not independent. They must satisfy:

$$\sum_{a \in A} q_s(a|b,c) = 1 \qquad \forall s \in S, b \in A^{|N_h(s)|}, c \in A^{|N_i(s)|}. \tag{7}$$

We incorporate these constraints to the Likelihood function by using Lagrange multipliers $\mu_s(b,c)$, $s \in S$, $b \in A^{|N_h(s)|}$, $c \in A^{|N_i(s)|}$:

$$\Phi^*(\mathcal{P},\mathcal{Q}) = \Phi(\mathcal{P},\mathcal{Q}) + \sum_{s \in S_h \cup S_o} \sum_{b \in A^{|N_h(s)|}, c \in A^{|N_i(s)|}} \mu_s(b,c) \sum_{a \in A} q_s(a|b,c).$$

Maximization of $\Phi(\mathcal{P}, \mathcal{Q})$ under the constraints (7) is equivalent to maximization of $\Phi^*(\mathcal{P}, \mathcal{Q})$ without constraints. To maximize Φ^* we apply the usual necessary condition that the partial derivatives with respect to the q's equal zero. The partial derivatives are given by:

$$\frac{\partial \Phi^*}{\partial q_s(a \mid b,c)} = \mu_s(a \mid b,c) +$$

$$\frac{\sum_t \sum_x Prob(Y^1..Y^T = y^1..y^T, X^1..X^T = x^1..x^T \mid U^1..U^T = u^1..u^T, X^0 = x^0; \mathcal{P}) \mathbf{1}_{a,b,c}(x_s^{t+1}, x_{N_h(s)}^t, u_{N_i(s)}^t)}{q_s(a \mid b,c)}. \tag{8}$$

Here the function $\mathbf{1}_{a,b,c}(x,z,u)$ equals 1 when $x=a$, $z=b$ and $u=c$; it is zero otherwise.

Suppose now that s is a hidden unit. Setting (8) equal to 0 we obtain that

$$q_s(a \mid b,c) = \frac{\sum_{t:u^{t+1}_{N_i(s)}=c} Prob(Y^1..Y^T = y^1..y^T, X^{t+1}_s = a, X^t_{N_h(s)} = b \mid U^1..U^T = u^1..u^T, X^0 = x^0)}{\sum_{t:u^{t+1}_{N_i(s)}=c} Prob(Y^1..Y^T = y^1..y^T, X^t_{N_h(s)} = b \mid U^1..U^T = u^1..u^T, X^0 = x^0)}$$

Remark: Note that the summation is taken only over these times t such that $u^t_{N_i(s)} = c$. Now we have the following reestimation iteration:

$$p^{m+1}_s(a \mid b,c) = \frac{\sum_{t:u^{t+1}_{N_i(s)}=c} Prob(Y^1..Y^T = y^1..y^T, X^{t+1}_s = a, X^t_{N_h(s)} = b \mid U^1..U^T = u^1..u^T, X^0 = x^0; \mathcal{P}^m)}{\sum_{t:u^{t+1}_{N_i(s)}=c} Prob(Y^1..Y^T = y^1..y^T, X^t_{N_h(s)} = b \mid U^1..U^T = u^1..u^T, X^0 = x^0; \mathcal{P}^m)} \tag{9}$$

We can compute the terms in the fraction entirely in terms of the p^m's. However to do so we must sum out all possible $X^1, ..., X^T$ configurations, the number of which grows exponentially with T. This renders direct computation impractical. To overcome this difficulty we modify the algorithm slightly. First, assume $U^1 = u^1, ..., U^T = u^T$, $X^0 = x^0$ are fixed for the rest of this discussion and define some auxiliary quantities. The **transition matrix** is defined for all $z, x \in A^{M_h}$, for $t = 1, 2, ..., T$:

$$P^m_{zx}(t) \doteq Prob(X^t = x \mid X^{t-1} = z, U^t = u^t; \mathcal{P}^m).$$

This can be computed in terms of the p^m's:

$$P^m_{zx}(t) = \prod_{s \in S_h} p^m_s(x_s \mid z_{N_h(s)}, u^t_{N_i(s)}).$$

The **emission matrix** is defined for all $x \in A^{M_h}$, $y \in A^{M_o}$, for $t = 1, 2, ..., T$:

$$Q^m_{xy}(t) \doteq Prob(Y^t = y \mid X^t = x, U^t = u^t; \mathcal{P}^m).$$

This can also be computed in terms of the p^m's:

$$Q^m_{xy}(t) = \prod_{s \in S_o} p_s(y_s \mid x_{N_h(s)}, u^t_{N_i(s)}).$$

Next we define the **forward** and **backward probabilities** for all $x \in A^{M_h}$, $t = 1, ..., T$

$$\alpha^m_t(x) \doteq Prob(Y^1...Y^t = y^1...y^t, X^t = x | U^1..U^T = u^1..u^T, X^0 = x^0; \mathcal{P}^m),$$

$$\beta^m_t(x) \doteq Prob(Y^{t+1} = y^{t+1}...Y^T = y^T | X^t = x, U^1..U^T = u^1..u^T, X^0 = x^0; \mathcal{P}^m).$$

The α's are called *forward probabilities* and the β's *backward probabilities*. Now we will obtain an evolution equation for the forward probabilities. This evolution equation in most cases is only *approximately true*. It is exactly true under the assumption that the input stochastic process

$\{U^t\}_{t=1}^{\infty}$ is a sequence of independent random variables. In that case the forward probabilities obey the forward evolution equation for all $x \in A^{M_h}$, $t = 0, ..., T-1$:

$$\alpha_{t+1}^m(x) = \sum_{z \in A^{M_h}} \alpha_t^m(z) P_{zx}^m(t+1) Q_{xy^{t+1}}^m(t+1)$$

with initial condition $\alpha_0^m(x^0) = 1$, $\alpha_1^m(x) = 0$ for all $x \neq x^0$. The backward probabilities satisfy the backward evolution equation: for all $x \in A^{M_h}$, $t = 1, ..., T-1$:

$$\beta_t^m(x) = \sum_{z \in A^{M_h}} P_{xz}^m(t+1) Q_{zy^{t+1}}^m(t+1) \beta_{t+1}^m(z)$$

with final condition $\beta_m^T(x) = 1$, $\forall x \in A^{M_h}$.

As already mentioned, the evolution equations hold true only if $U^1, U^2, ...$ are independent of each other (or, trivially, if there is no input process). In many cases we have no reason to expect the input process to be independent but we also have no information about the nature of correlation across time. In that case the assumption of independence is a natural one and in most cases we can expect the evolution equation to hold up to a reasonably good approximation.

Now, using the forward and backward probabilities we can write the following relationship:

$$Prob(Y^1..Y^T = y^1..y^T, X_s^{t+1} = a, X_{N_h(s)}^t = b | U^1..U^T = u^1..u^T, X^0 = x^0, \mathcal{P}^m) =$$

$$\sum_{x^t, x^{t+1} : x_s^{t+1} = a, x_{N_h(s)}^t = b} \alpha_t^m(x^t) P_{x^t x^{t+1}}^m(t+1) Q_{x^{t+1} y^{t+1}}^m(t+1) \beta_{t+1}^m(x^{t+1}).$$

Similarly, we have:

$$Prob(Y^1..Y^T = y^1..y^T, X_{N_h(s)}^t = b | U^1..U^T = u^1..u^T, X^0 = x^0, \mathcal{P}^m) =$$

$$\sum_{x^t : x_{N_h(s)}^t = b} \alpha_t^m(x^t) \beta_t^m(x^t).$$

Now we can rewrite the fraction in (9) as

$$p_s^{m+1}(a \mid b, c) = \frac{\sum_{t : u_{N_i(s)}^{t+1} = c} \sum_{x^t, x^{t+1} : x_s^{t+1} = a, x_{N_h(s)}^t = b} \alpha_t^m(x^t) P_{x^t x^{t+1}}^m(t+1) Q_{x^{t+1} y^{t+1}}^m(t+1) \beta_{t+1}^m(x^{t+1})}{\sum_{t : u_{N_i(s)}^{t+1} = c} \sum_{x^t : x_{N_h(s)}^t = b} \alpha_t^m(x^t) \beta_t^m(x^t)} \tag{10}$$

Once again, note that summation is *not* over all t, just the ones which have appropriate input: $u_{N_i(s)}^t = c$.

The reestimation formula (10) holds for all hidden units $s \in S_h$. If s is an output unit ($s \in S_o$), we can follow similar reasoning as before to get:

$$p_s^{m+1}(a \mid b, c) = \frac{\sum_{t : u_{N_i(s)}^t = c, y_s^t = a} \sum_{x^t : x_{N_h(s)}^t = b} \alpha_t^m(x^t) \beta_t^m(x^t)}{\sum_{t : u_{N_i(s)}^t = c} \sum_{x^t : x_{N_h(s)}^t = b} \alpha_t^m(x^t) \beta_t^m(x^t)} \tag{11}$$

Note that, once again, the summation is not over all t, just over the ones where the appropriate restriction of u^t, y^t equals c, a, respectively.

This completes the description of the local BF algorithm. Putting all the pieces together, we have:

The local BF algorithm:

Given a sample $y^1, y^2, ..., y^T$ and $u^1, u^2, ..., u^T$, and a network topology $\mathcal{G} = (S, \mathcal{N})$, we want to find a set of local conditionals $\hat{\mathcal{P}}$ that is compatible with $(S, \mathcal{N})$ and maximizes the Likelihood

$$L(\mathcal{P}) = Prob(Y^1 = y^1, ..., Y^T = y^T, U^1 = u^1, ..., U^T = u^T; (\mathcal{G}, \mathcal{P})).$$

We obtain $\hat{\mathcal{P}}$ as the limit of $\mathcal{P}^0, \mathcal{P}^1, ...$ which are obtained as follows:

1. Choose $\mathcal{P}^0$ arbitrarily.
2. ...
3. Given $\mathcal{P}^m$ compute $P^m(t)$, $Q^m(t)$, $t = 1, ..., T$ with:

$$P^m_{zx}(t) = \prod_{s \in S_h} p^m_s(x_s \mid z_{N_h(s)}, u^t_{N_i(s)}).$$

$$Q^m_{x,y}(t) = \prod_{s \in S_o} p_s(y_s \mid x_{N_h(s)}, u^t_{N_i(s)}).$$

Given $P^m(t), Q^m(t)$, $t = 1, 2, ..., T$, compute $\alpha^m(t)$, $\beta^m(t)$ $(t = 1, ..., T)$ with:

$$\alpha^m_1(x^0) = 1, \qquad \alpha^m_1(x) = 0 \quad \forall x \neq x^0.$$

$$\alpha^m_{t+1}(x) = \sum_{z \in A^{M_h}} \alpha^m_t(z) P^m_{zx}(t+1) Q^m_{xy^{t+1}}(t+1)$$

$$\beta^T_m(x) = 1 \qquad \forall x \in A^{M_h},$$

$$\beta^m_t(x) = \sum_{z \in A^{M_h}} P^m_{xz}(t+1) Q^m_{zy^{t+1}}(t+1) \beta^m_{t+1}(z)$$

4. Given α^m, β^m compute $\mathcal{P}^{m+1}$ by:

$$\forall s \in S_h \qquad p^{m+1}_s(a \mid b, c) =$$

$$\frac{\sum_{t: u^{t+1}_{N_i(s)} = c} \sum_{x^t, x^{t+1}: x^{t+1}_s = a, x^t_{N_h(s)} = b} \alpha^m_t(x^t) P^m_{x^t x^{t+1}}(t+1) Q^m_{x^{t+1} y^{t+1}}(t+1) \beta^m_{t+1}(x^{t+1})}{\sum_{t: u^{t+1}_{N_i(s)} = c} \sum_{x^t: x^t_{N_h(s)} = b} \alpha^m_t(x^t) \beta^m_t(x^t)}$$

$$\forall s \in S_o \qquad p^{m+1}_s(a \mid b, c) =$$

$$p^{m+1}_s(a \mid b, c) = \frac{\sum_{t: u^t_{N_i(s)} = c, y^t_s = a} \sum_{x^t: x^t_{N_h(s)} = b} \alpha^m_t(x^t) \beta^m_t(x^t)}{\sum_{t: u^t_{N_i(s)} = c} \sum_{x^t: x^t_{N_h(s)} = b} \alpha^m_t(x^t) \beta^m_t(x^t)}$$

We have proven rigorously that our algorithm will increase the Likelihood at every step. Convergence to a local maximum can also be rigorously proven [Wu83].

Now we will present a complementary point of view, where the algorithm is derived heuristically, from a general principle known as the *EM method* [D+77]. Suppose we did not know the actual values of the p_s's. However we could assume some arbitrary values $\mathcal{P}_0$ and using these proceed to compute various statistics of the network. In particular let us take some unit s (say it is a hidden unit) and, given the observed input and output, let us compute the expected number of times that unit s is state a and its parents were (one time epoch ago) in states b, c. The actual number of such configurations is:

$$\sum_{t=0}^{T-1} \mathbf{1}_{a,b,c}(X_s^{t+1}, X_{N_h(s)}^t, U_{N_i(s)}^{t+1}). \tag{12}$$

Now the expectation of this quantity, with respect to the probability measure $Prob(. \mid Y^1..Y^T = y^1..y^T, U^1..U^T = u^1..u^T, X^0 = x^0 \; ; \mathcal{P}^0)$ (the probability conditional on the input and output) is itself a probability:

$$E\left(\sum_{t=0}^{T-1} \mathbf{1}_{a,b,c}(X_s^{t+1}, X_{N_h(s)}^t, U_{N_i(s)}^{t+1})\right) = \tag{13}$$

$$\sum_{t:u_{N_i(s)}^{t+1}=c} Prob(X_s^{t+1} = a, X_{N_h(s)}^t = b | Y^1..Y^T = y^1..y^T, U^1..U^T = u^1..u^T, X^0 = x^0, \mathcal{P}^0)$$

Similarly we can compute the expected number of times the parents of s were in state bc. The actual number is:

$$\sum_{t=0}^{T-1} \mathbf{1}_{b,c}(X_{N_h(s)}^t, U_{N_i(s)}^{t+1}). \tag{14}$$

Now the expectation of this quantity, with respect to $P(. \mid Y^1..Y^T = y^1..y^T,\ U^1..U^T = u^1..u^T,\ X^0 = x^0; \mathcal{P}^0)$ (the probability conditional on the input and output) is in fact a probability:

$$E\left(\sum_{t=0}^{T-1} \mathbf{1}_{b,c}(X_{N_h(s)}^t, U_{N_i(s)}^{t+1})\right) =$$

$$\sum_{t:u_{N_i(s)}^{t}=c} Prob(X_{N_h(s)}^t = b | Y^1..Y^T = y^1..y^T, U^1..U^T = u^1..u^T, X^0 = x^0, \mathcal{P}^0) \tag{15}$$

Now, a reasonable re-estimate $p_s^1(a|b,c)$ would be the ratio of the two expectations in (13),(15):

$$p_s^1(a|b,c) = \frac{\sum_t Prob(X_s^{t+1} = a, X_{N_h(s)}^t = b | Y^1..Y^T = y^1..y^T, U^1..U^T = u^1..u^T, X^0 = x^0; \mathcal{P}^0)}{\sum_t Prob(X_{N_h(s)}^t = b | Y^1..Y^T = y^1..y^T, U^1..U^T = u^1..u^T, X^0 = x^0; \mathcal{P}^0)}$$

Multiplying both numerator and denominator by $Prob(Y^1..Y^T = y^1..y^T \mid U^1..U^T = u^1..u^T,$

$X^0 = x^0; \mathcal{P}^0)$ we get

$$p^1_s(a|b,c) = \frac{\sum_t Prob(Y^1..Y^T = y^1..y^T, X^{t+1}_s = a, X^t_{N_h(s)} = b | U^1..U^T = u^1..u^T, X^0 = x^0, \mathcal{P}^0)}{\sum_t Prob(Y^1..Y^T = y^1..y^T, X^t_{N_h(s)} = b | U^1..U^T = u^1..u^T, X^0 = x^0, \mathcal{P}^0)} \tag{16}$$

Using the same reasoning we can get the same reestimation formula for every $s \in S_h$ and a similar one from every output unit s ($s \in S_o$). Applying this reestimation for all $s \in S_h \cup S_o$ we get a new set of local conditionals, $\mathcal{P}^1$. Now we can use this new estimate $\mathcal{P}^1$ to recompute (16) and then compute another $\mathcal{P}^2$ estimate etc. However, the reestimate (16) is simply (9). The analogy works out in exactly the same way for the output units.

This heuristic discussion illuminates the mathematical derivation of the local BF. The heuristic is a special case of the EM method [D+77] which is used for ML estimation of parameters of random phenomena. The general idea is: assume the parameters to have some values, use these values to compute some statistics of the random phenomenon and use these statistics to recompute the parameters. We keep iterating until convergence is achieved. In many cases (e.g. BF, local BF) convergence can be proven. In Appendix B we show the Relaxation method used for Boltzmann Machine training is a special case of EM as well. Convergence can be proven when the training takes place under certain conditions (simulated annealing). Furthermore, both Relaxation and BF are greedy ascent methods. Similarly, Back Propagation, is a greedy *descent* method. (We want to *minimize* square error.) However, there is an important difference between Boltzmann Machine and Back Propagation learning and the local BF. BF has automatic step selection: given $\mathcal{P}^m$ we compute $\mathcal{P}^{m+1}$ by (10) and the difference $\|\mathcal{P}^{m+1} - \mathcal{P}^m\|$ need not be small. BP and Relaxation require that $\|\mathcal{P}^{m+1} - \mathcal{P}^m\|$ is small. This implies that they have to move slowly towards the optimum of the objective function. This is not the case for local BF, indeed there is a considerable speedup of convergence, as we will see in the next section.

Remark: The local BF algorithm is essentially a version of the BF algorithm that estimates local conditional probabilities, rather than the global transition matrix which is estimated by the Baum BF algorithm [L+83]. A point that is important to note is that the computational effort is of the order $O(T \cdot |A|^{2M_h})$. The linear rate in the sample length T is of course very satisfactory from the computational point of view. The M_h exponential does not look good at first sight, but recall that a network with M_h hidden units has $|A|^{M_h}$ states, so what we really have here is the usual result for HMM's: the computational effort is $O(T \cdot K^2)$, where K is the number of states. Of course this analysis of computational complexity does not take into account the fact that BF-type algorithms converge in fewer iterations than, say, Back Propagation.

Remark: Even though the local BF algorithm has been developed for application to recurrent networks, it can be used equally well for the training of static, feedforward networks. To see this, start with the observation that a recurrent network in $t = 1, ..., T$, unfolded in time, can be considered as a T-layer feedforward network (see [Keh90]). This feedforward network can be trained by the local BF algorithm. It is a special type of feedforward network, in that (a) every layer has the same number of units and (b) groups of units across layers are constrained to have the same local conditionals. Both of these constraints follow from the fact that the t-th layer is a time-shifted copy of the $(t-1)$-th layer. A more general type of feedforward network consists of several layers of probabilistic units, where each layer has an arbitrary number of units and each unit has a distinct local conditional. The local BF algorithm applies equally well to such a network with small modifications. The basic requirement for the algorithm to work is that the Ψ function in (4) can be written as a product of probabilities. (Why this is necessary becomes apparent from inspection of Baum's theorem and the maximization step - for a more general discussion see [BS68].) In a feedforward net, we will typically have a set of L input/output pairs: $(u_1, y_1), ..., (u_L, y_L)$. The Likelihood function will be a product of probabilities:

$$L(\mathcal{P}) = \prod_{l=1}^{L} Prob(Y^l = y^l | U^l = u^l), \tag{17}$$

because we can consider the sequence of input presentations as a sequence of independent random experiments. Starting from (17), we can obtain the local BF algorithm for feedforward networks in exactly the same way as before. This will be pursued in a later paper.
Remark: Another modification of the local BF algorithm is necessary when we do not have enough data to reliably estimate a large number of local conditionals (problem of overfitting). One way to deal with this, is by assuming certain units to have the same local conditionals. Obviously this limits the number of free parameters, hopefully to a level where we have enough data for reliable estimation. Say that units $s_1, s_2, ..., s_D$ have the same local conditionals (this implies they also have the same size of parent sets). Call the local conditional probability of this group p_σ (or q_σ etc.). Then, when computing the partial derivative of $Q(\mathcal{P}, \mathcal{Q})$ with respect to q_σ, as in (8), we have to add up the contribution of all sites s_d, $d = 1, ..., D$. This is the only change in the algorithm; the rest goes as previously and the details are omitted. This is the equivalent of "tied" transition probabilities discussed in [Jel83] and can result in substantial decrease of the number of parameters that need to be estimated.

5 Applications

In this section we will use the local BF algorithm to solve two problems: (a) the 8-3-8 encoder and (b) phoneme modelling. These are rather simple problems used to illustrate the most important features of the algorithm and also to introduce some practical tricks that improve its efficiency.
(a) 8-3-8 Encoder: This is a "static" problem discussed in [A+85] and elsewhere. We have a set of input/output pairs $(u^1, y^1), ..., (u^8, y^8)$ and we want to build a network that takes input u^n and produces output y^n for $n = 1, ..., 8$. The input/output pairs are the following 8-long binary vectors: $u^1 = y^1 = 10000000$, $u^2 = y^2 = 01000000$, ..., $u^8 = y^8 = 00000001$. The network has three hidden units. Hence the name "8-3-8 encoder".

The network consists of: 8 binary input units (call them $i_0, i_1, ..., i_7$), 3 binary hidden units (call them h_1, h_2, h_3) fully connected to each other and to all the input units and 8 binary output units (call them $o_0, o_1, ..., o_7$) fully connected to the hidden units. This completes the topology description. For specifying the local conditionals we will use the local BF algorithm.

However, we change the local BF algorithm somewhat so as to facilitate the learning process. Namely, we will only estimate the local conditionals of the hidden units; the local conditionals

Fig.1 about here

Figure 1: Plot of Input vs. Time

of the output units will *not* be estimated but "hand-picked" as follows. For $n = 0, ..., 7$:

$$p_{o_n}(1|x_1^0 x_2^0 x_3^0) = \begin{cases} 1 & \text{iff } x_1^0 + 2x_2^0 + 4x_3^0 = n \\ 0 & \text{else} \end{cases}$$

In words, the n-th output unit turns on if n is equal to the binary number that corresponds to the activation pattern of the hidden units.

So the learning problem is to find a set of local conditionals for the hidden units: $\{p_{h_n}(x \mid u_1^0 u_2^0 ... u_7^0),\ n = 1, 2, 3,\ x, u_0^0, ..., u_7^0 \in \{0, 1\}\ \}$.

We want to teach a static input/output relationship to a recurrent network. We need to convert the static problem to a dynamic one. We do so by inventing an appropriate time evolving input/output sequence. There are many ways to achieve this. We use the following input/output sequence. The input sequence is $u^1, ..., u^{200}$ and the output sequence is $y^1, ..., y^{200}$; we take $u^t = y^t = 10000000$ for $t = 1, ..., 25$, $u^t = y^t = 01000000$ for $t = 26, ..., 50$ etc. In other words, for every type of constant input we want the SRN to settle down to the appropriate deterministic steady state. It is not easy to plot an 8-long binary vector vs. time; instead we plot in Fig.1 the number of the active output unit (there is only one active output at any given time t).

We can see immediately that there is at least one set of hidden local conditionals that will realize the desired input/output behavior:

$$p_{h_n}(1|u_0^0 ... u_7^0) = \begin{cases} 1 & \text{iff for some } m \in \{0, .., 7\}\ u_m^0 = 1 \text{ and } Bin(m, n) = 1 \\ 0 & \text{else.} \end{cases} \tag{18}$$

It is easy to check that for any legal input (all but one input units are off!) the output activation pattern will be exactly the sane as the input one. To see this, take any legal input to the network - only one input unit will be on at any given time. Take this unit's number and write it out in binary notation. The corresponding binary number will reflect the activation pattern of the hidden units and this in conjunction with (18) will give the correct output. It is easy to check there are many other possible sets of local conditionals that will produce the desired behavior (the problem is underdetermined) but (18) is, in an obvious sense, the best solution. The question is whether the local BF will pick up this solution and how fast.

We initialize the hidden units for completely random output: $p_{h_n}(1|u_0^0...u_7^0) = .5$, $n = 0,1,...,7$, $\forall x, u_0^0,...,u_7^0 \in \{0,1\}$. After 5 iterations of the BF algorithm (seven minutes on a Sun/4) we arrive to the following solution:

$$p_{h_n}(1|u_0^0...u_7^0) = \begin{cases} 1-\epsilon & \text{iff for some } m \in \{0,...,7\}\ u_m^0 = 1 \text{ and } Bin(m,n) = 1 \\ \epsilon & \text{else.} \end{cases}$$

The ϵ varies for each $n = 1,2,3,\ x, u_0^0,...,u_7^0$ but is always less than .001. The SRN behaves almost deterministically and the solution is almost perfect solution. Of all the sets of conditional probabilities that solve the learning task, the local BF algorithm picks up the "best" solution very quickly.

It is clear that finding a good solution of the encoder problem is extremely easy using our model and algorithm. There are many possible sets of conditional probabilities that solve the learning task; in this sense the problem is underdetermined. What is remarkable is that the local BF algorithm picks up the best solution very fast.

(b) Phoneme Model: Now consider a dynamic problem. We will take a sample speech waveform and build a SRN model that will reproduce this waveform.

This task is essentially the same as Hidden Markov Modelling of a phoneme, much in the spirit of [L+83]. See Appendix C for a discussion of the connection between SRN and HMM. However, there are some important differences from [L+83]. First of all we estimate *local* conditional probabilities, not *global* ones (namely, the probability transition matrix). Second, we model the raw speech waveform rather than the LPC coefficients or some such parametric model. Such raw signal models of phonemes are relatively rare - for a more extensive treatment of this problem see [Keh91]. In [Keh91] we develop *global* HMM's; here we will develop a local HMM, in the sense that we estimate the local conditionals. Once again, we solve this problem as an example of the local BF and do not make any conclusive claims about the practical usefulness of such models. We *do* believe they may have important practical applications and we pursue them further in [Keh91] and [Keh1]. Still, both [Keh91, Keh1] are rather theoretical - there is need for more applications-oriented work.

Consider a typical steady state speech waveform (Fig.2). This is the waveform corresponding to the steady state part of phoneme [ou], from the utterance "one". It is a continuous valued waveform; on the other hand our SRN's produce output from a finite alphabet. We have to "preprocess" the original phoneme waveform; in particular we quantize it to four levels to get the waveform of Fig.3, which still retains the characteristic [ou] shape. In Fig.4 we plot the original and quantized waveform together. Now we want to build a model that "reproduces" the waveform of Fig.3.

We have to be careful about the meaning of "reproduction". Before we proceed with the network description, let us describe our modelling objective.

Once we fix the network topology, we will choose the values of the local conditionals in such a way that Likelihood is maximized; this is a well defined criterion. However, a poor choice of topology may result in the Maximum Likelihood model for this specific topology not being very

Fig.2 about here

Figure 2: Steady State, Continuous Valued Speech Waveform of [ou]

Fig.3 about here

Figure 3: Steady State, Quantized Speech Waveform of [ou]

Fig.4 about here

Figure 4: Continuous and Quantized Speech Waveform of [ou]

good. Clearly, some topologies will be better than others; at least we intuitively expect bigger networks (more units) to do better. We need some absolute criterion of goodness of modelling, independent of topologies. There are several possible points of view. Here we will mention only two.

The first point of view is that the original speech waveform is a stochastic process, and so is the output of the SRN model. Hence we want to develop a measure of distance of stochastic processes and check that in our case the distance from original to model is small. This approach is explained in more detail in [Keh91] and [Keh2]. There we develop and discuss several appropriate distance measures. All of these measures depend on the joint probabilities (for the original and model processes) of finite length outputs. This approach is theoretically important and is used in [Keh91, Keh2] to prove consistency of ML estimation. However, it is not appropriate for practical application in the case of speech waveforms, because of the following reason. The important information about speech appears in fairly long term dependencies, e.g. in the fact that the waveform of Fig.3 is quasi-periodic with approximately the same pattern being repeated every sixteen or so time steps. Now, for these dependencies to appear in the joint probability distributions $Prob(X^{t+1}...X^{t+n})$, we must look at n big enough to capture the periodicity, e.g. $n = 16$. And to compute distances between two such 16-th order joint probabilities, we must compute their values for all possible values of 16-long quaternary strings $X^{t+1}...X^{t+16}$. But there is several billions of these! So this approach is not very practical for computation.

An alternative approach, which we will follow here, is to concentrate on the *shape* of the speech waveforms. For different phonemes we get characteristically different shapes; compare Fig. 3 (an [ou] phoneme) with Fig.5 (an [ah] phoneme). The visual *texture* of the two waveforms is strikingly different. This corresponds to different audio texture, because the visual appearance depends on the number and location of the *formants* (see [Gra76]) which can be used to discriminate between different phonemes. Therefore, in this example we will consider our model succesful if it produces an output processes which "looks similar" to the original. This will be called the **visual similarity criterion**. It is admittedly a vague criterion, but in practice it works quite well, in other words it is pretty clear when the visual texture of a waveform is

Fig.5 about here

Figure 5: Steady State, Quantized Speech Waveform of [ah]

captured. A further confirmation of the validity of our model will appear in [Keh1], where it is used succesfully as the starting point of our prediction and classification algorithms.

Let us now return to the description of our SRN model. We use a network with *no* input units (since there is no input process correlated with the speech waveform), four binary hidden units $(h_1, ..., h_4)$ (fully connected to each other) and one four-valued output unit (call it o_1), fully connected to the hidden units.

The local BF algorithm can be modified to accomodate the lack of input units; just drop the input dependence from local and global conditionals. It is more natural to use one four-valued output unit than two binary ones; this can also be easily accommodated in the algorithm. The fact that we use a four-valued alphabet causes no increase in computation, (no exponential explosion) because the output values are observed and fixed.

Just like in the previous example, we adopt a hybrid approach in choosing the local conditionals. We will once again "handpick" the emission matrix. In this case the emission matrix is the same as the local conditional $p_{o_1}(y|x_1^0...x_4^0)$. The reason for using handpicked emission matrix is that a judicious choice can force the local BF algorithm to search in the right region of the parameter space. Before we elaborate on the choice of the emission matrix consider again the waveform of Fig.3.

Looking at the quantized speech waveform, we realize that it is quasi-periodic, with one quasiperiod being about 16 time steps long. In particular, it is *approximately* composed of **prototype** elements like the one shown in Fig.6. By "approximately" we mean the following: take a sequence of deformations of the element in Fig.6 (e.g. dilations and compressions of time scale) and string them together. What we will get is a waveform like the one of Fig.7, that satisfies the criterion of visual similarity that we postulated in the previous paragraphs. So the question is: how to build a SRN that creates this type of output, with local time scale deformations.

We can think of every time step t (t=0,1,...,15) in one prototype as corresponding to one state of the SRN. Recall that every state of the SRN is a 4-long binary vector. The sixteen four-long binary vectors correspond to the sixteen states of the protoype. We can number each

Fig.6 about here

Figure 6: Prototype Waveform

Fig.7 about here

Figure 7: “Deformed Prototypes” Waveform

vector/state by the binary number that corresponds to this vector. There are sixteen states (0-15).: **0**, **1**, ..., **15**. [1] Suppose the SRN changes states as follows: if at time $t-1$ it is in state **t-1**, at time t it will go deterministically to state **t** (state **15** goes to state **0**). At the same time it produces output y^t. Therefore, for $s = 0, ..., 15$, $t = 1, 2, ...$ we have

$$Prob(Y^t = y^s | X^t) = \begin{cases} 1 & \text{iff } \exists k \in \{0, 1, 2, ...\} \text{ and } t = s \cdot k + 1 \\ 0 & \text{else} \end{cases} \tag{19}$$

In this way we have specified the global transition and emission probabilities.

With a little reflection it should be clear that, if we start with a prototype $y^0, ..., y^{15}$ from Fig.6, and the network we described above, we will get a model that produces perfect copies of the original prototype. Call this **Model 1**. Now, Model 1 satisfies the visual criterion, but it is a poor Maximum Likelihood model, because our sample also has sections which are slightly different from the first quasiperiod/prototype and these will be assigned (under this deterministic model) probability zero.

To rectify this weakness of the model, we relax the deterministic state transition hypothesis and allow for random transitions between states. Say we allow every state two options: either move on to the next state (with high probability) or loop back to itself (with low probability). This model will be called **Model 2**. It is still a pretty simple model, but will allow *every possible* state path and will reproduce deformed copies of the prototype element of Fig.6. This model has higher Likelihood than Model 1. However, it is conceivable that a more general transition model will do even better (higher Likelihood, better visual texture).

Now, both of the models discussed so far, are "handcrafted" models, in that we handpick the global emission and transition conditionals. Theoretically, we should do at least as well if we implement the model using local conditionals estimated by the Maximum Likelihood method. However, there are many practical considerations (e.g. will the local BF yield a globally maximizing set of local conditionals?). Hence we will take a slightly different, hybrid approach to obtain **Model 3**.

1. We retain the emission matrix already described, namely

$$Q = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T .$$

   Comparing this matrix with the prototype waveform of Fig. 6 we see that it implements (19). We can implement this global output matrix in terms of local conditionals because there is only one output unit, which is fully connected to all hidden units. So in this case, emission matrix and local conditional are identical. In other words, we have **handpicked** the emission local conditionals.

[1] What would happen if the prototype was not sixteen time steps long? Say it was twelve time steps long. If we use binary units we need four of them and the SRN will have sixteen states anyway. We have four superfluous states; we need to take them out out of the game. One way to do this is to have the extra states emit an extra output character that has never been observed. This guarantees that Maximum Likelihood estimation of the local conditionals will assign zero probability to these states.

2. It is not so obvious how to "handpick" the hidden local conditionals. First of all, it is not obvious how to choose local conditionals that implement our Model 2 transition matrix. Secondly, maybe there is a "better" (higher Likelihood) set of hidden local conditionals to use with our emission matrix. We will trust that the local BF algorithm will find this set of hidden local conditionals.

It is obvious that we have used a hybrid approach of estimation. We have a good guess for the emission matrix and so we use this guess, rather than estimating the emission matrix. We also have some idea of what the global transition matrix should look like, but do not know much about the local conditionals. So we let the local BF algorithm search the parameter space for the Maximum Likelihood local conditionals. Hopefully, our choice of emission matrix "forces" the BF algorithm to go quickly to the relevant regions of the parameter space.

Now let us test our method by using local BF to determine the parameters of Model 3. We initialize the hidden units as follows (completely randomly): $p_{h_n}(x_n^1|x_1^0..x_4^0) = .5$, $n = 1,2,3,4$, $\forall x_n^1, x_1^0, ..., x_4^0 \in \{0,1\}$. After 30 iterations of the BF algorithm (twenty minutes on a Sun/4) we get a convergent solution; the actual values of the local conditionals are not very illuminating, but the global transition matrix we get from them is what we expect to be a "good" solution: most state transitions are completely impossible, every state has high probability of going to the next state and a low probability of looping back into itself. This "good" global transition matrix was derived from local conditionals which the local BF algorithm discovered quickly, starting from random initial values. So it appears that the local BF does indeed converge to the appropriate region of the parameter space quickly, even when it starts far away from it. Of course, the judicious choice of the emission matrix has helped things considerably.

We do not list the transition matrix here; it is big and would be awkward to represent. At any rate, more important than the parameter values, is the degree to which our model reproduces faithfully the sample waveform. To test this we run the SRN model to produce a new output sample and we get the waveform of Fig.8. It is intuitively obvious that the model captures the general shape of the original sample and should be a very good model. The succesful use of this model for prediction and classification (see [Keh1]) will confirm our intuition.

6 Conclusions

We have developed a new SRN model and a Maximum Likelihood algorithm to train it. We have used these to model static as well as dynamic input/output relationships. We get accurate and easy to train models. Extensions to training of feedforward networks are easy to obtain.

We can prove that our SRN can model a very large class of stochastic processes and input/output relationships. This will be done in [Keh2] where we study the representation power of SRN models. We have presented two examples where we model an input/output relationship by an SRN. We showed that a static input/output relationship can be captured as the steady state of a SRN (encoder problem). We also built a SRN that reproduces a dynamic, continuously evolving stochastic process (speech waveform). Once we have a model of a stochastic process we can use it for more advanced learning tasks such as prediction and classification. This will be done in [Keh1].

Fig.8 about here

Figure 8: SRN Model Produced Waveform

Of all the various points of view on SRN (see Appendix D for a very brief review) two are particularly promising. Using the nonlinear input/output system approach puts us our model in the context of stochastic control; this is a mature discipline with a history of several decades of theory and applications. Many estimation and learning methods that have been developed in stochastic control can be applied to connectionist problems. We will pursue this direction in [Keh1]. The other interesting approach is to consider the SRN as an inhomogeneous Hidden Markov Model; this is briefly explained in Appendix C. Clearly, this has been our motivation in developing the local BF algorithm. The advantage of the HMM approach is the speed and ease of training. On the other hand, using the SRN formulation we enjoy the advantages of parallelism and parsimonious modelling.

The SRN model we developed is more general than the classical connectionist model with sigmoidal units. On the other hand, our model can be implemented as a connectionist network of stochastic nonlinear (but not sigmoidal) units (see Appendix A). So there is a certain equivalence between our SRN and more traditional connectionist models.

The local BF algorithm is a special case of the EM method. EM is a very pervasive heuristic; in Appendix B we point out that we can view Boltzmann Machine training as another special case of EM. However the BF algorithm has an additional characteristic which gives it a speed advantage: it self-selects optimal step size. This is not the case with relaxation learning, or with Back Propagation for that matter. Both of these are small step steepest ascent/descent algorithms. This explains their relative slowness.

In conclusion, we have developed a powerful and easily trainable model. We believe that the combination of HMM and Stochastic Control methods will help further development of SRN theory and application to many difficult problems of learning temporally evolving relationships.

Acknowledgement: I want to thank Nick Chater, Steve Nowlan and Ah Tsung Tsoi for useful suggestions on how to improve the notation and the organization of this paper.

References

[A+85] D.H. Ackley et al. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9, 1985.

[Ale89] I. Aleksander. The logic of connectionist systems. In I. Aleksander, editor, *Neural Computing Architectures*. MIT, 1989.

[Alm87] L.B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Proc. of 1st Int. Conf. on Neural Networks, S. Diego*. IEEE, 1987.

[Alm88] L.B. Almeida. Backpropagation in perceptrons with feedback. In R. Eckmiller and C.v.d. Malsburg, editors, *Neural Computers*. Springer, 1988.

[Alm89] L.B. Almeida. Backpropagation in nonfeedforward networks. In I. Aleksander, editor, *Neural Computing Architectures*. MIT Press, 1989.

[Ama71] S.I. Amari. Characteristics of randomly connected threshold elements and network systems. *Proc. of the IEEE*, 39:33–47, 1971.

[Ama72] S.I. Amari. Learning patterns and pattern sequences by self-organizing nets of threshold elements. *IEEE Trans. on Computers*, 21:1197–1206, 1972.

[Ama83] S.I. Amari. Field theory of self-organizing neural nets. *IEEE Trans. on Systems , Man and Cybernetics*, pages 741–748, 1983.

[AN89] L.B. Almeida and J.P. Neto. Recurrent backpropagation and Hopfield networks. In F. Fogelman-Soulie, editor, *Neurocomputing, Algorithms Architectures and Applications*. Springer, 1989.

[Aok87] M. Aoki. *State Space Modelling of Time Series*. Springer, Berlin, 1987.

[Bar89] A.G. Barto. Connectionist learning for control:an overview. Technical Report TR 89-89, COINS Dept., Un. of Mass. at Amherst, 1989.

[BA85] A.G. Barto and P. Anandan. Pattern recognizing stochastic learning automata. *IEEE Trans. on Systems , Man and Cybernetics*, SMC 15(3), June 1985.

[Bau72] L.E. Baum. An inequality and associated maximization technique in statistical estimation. *Inequalities*, 3:1–8, 1972.

[B+70] L.E. Baum et al. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov Chains. *Ann. of Math. Stat.*, 41(1):164–171, 1970.

[BE67] L.E. Baum and J.A. Eagon. An inequality with applications to statistical estimation for probabilistic functions of Markov Processes. *Ann. of Math. Stat.*, pages 36–363, 1967.

[Be+90a] Y. Bengio et al. A hybrid coder for Hidden Markov Models using a recurrent neural network. *Proceedings of the International conference on Acoustics, Speech and Signal Processing*, Albuquerque, NM, April 1990, pp.537-540.

[Be+90b] Y. Bengio et al. Global optimization of neural network-HMM hybrid. Technical Report TR-SOCS-90.22. McGill University, School of Computer Science. December 1990.

[Boo67] T. Booth. *Sequential Machines and Automata Theory.* Wiley, New York, 1967.

[BS68] L.E. Baum and G.R. Sell. Growth transformations for functions on manifolds. *Pac. J. of Math.*, 27(2):211–227, 1968.

[BW88] H. Bourlard and C.J. Wellekens. Links between Markov models and multilayer perceptrons. Technical Report, Phillips Research Lab, 1988.

[BW89] H. Bourlard and C.J. Wellekens. Speech dynamics and recurrent neural nets. In *Proc. of the ICASSP.* IEEE, 1989.

[BD87] P.J. Brockwell and R.A. Davis. *Time Series: Theory and Methods.* Springer, New York, 1987.

[D+77] A.P. Dempster et al. Maximum Likelihood estimation via the EM algorithm. *J. of the Stat. Roy. Soc. B*, 39(1):1–38, 1977.

[Dob65] R.L. Dobrushin. Existence of a phase transition in two-dimensional and three-dimensional ising models. *Th. of Prob. and its Appl.*, 10(2):193–213, 1965.

[Dob70] R. L. Dobrushin. Prescribing a system of random variables by conditional distributions. *Th. of Prob. and its Appl.*, 15(3):459–486, 1970.

[dL88] D. d'Humieres and P. Lallemand. Fluid dynamics with lattice gases. In R. Livi et al., editors, *Chaos and Complexity*, pages 279–301. World Scientific, Singapore, 1988.

[Dre90] S.E. Dreyfus. Artificial neural networks backpropagation and the Kelly-Bryson gradient procedure. *J. Guid. Cont. Dyn.*, 13(5):926–928, Sept.-Oct. 1990.

[Ell85] R. S. Ellis. *Entropy, Large Deviations and Statistical Mechanics.* Grundlehren der Math. Wissenschaften. Springer, New York, 1985.

[Elm88] J. L. Elman. Finding structure in time. Technical Report CRL TR 8801, Center for Research in Language, University of California at San Diego, 1988.

[GG84] S.Geman and D. Geman. Stochastic relaxation Gibbs distributions and the bayesian restoration of images. *IEEE PAMI*-9:721–741, 1984.

[Gid85] B. Gidas. Nonstationary Markov Chains and convergence of the annealing algorithm. *J. Stat. Physics*, 39:73–131, 1985.

[Gil90] C.L. Giles. Higher order recurrent networks and grammatical inference. In D. Touretzky, editor, *Advances in Neural Information Processing*, San Mateo California, 1990. Morgan Kauffman.

[Gra76] A.H. Gray, Jr. and J.D. Markel. Linear Prediction of Speech. Springer, Berlin, 1976.

[Gro86a] S. Grossberg. *The Adaptive brain:I Learning, reinforcement, motivation and rhythm.* Wiley, 1986.

[Gro86b] S. Grossberg. *The Adaptive brain:II Vision, Speech, Language and Motor Control.* Wiley, 1986.

[Hop82] J.J. Hopfield. Neural nets and physical systems with emergent collective computational properties. *Proc. Nat'l Acad. Sci. USA*, 1982.

[Hop85] J.J.Hopfield and D.W.Tank. Neural computation of decisions in optimization problems. *Biol. Cyb.*, 52, 1985.

[HS86] G. Hinton and T. Sejnowski. Learning and relearning in Boltzmann machines. In *Parallel Distributed Processing*, volume 1. MIT, 1986.

[Jel83] F. Jelinek et al. A Maximum Likelihood approach to continuous speech recognition. IEEE PAMI-5(2), pp.179-190.

[Jor86] M.I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proc. of the 8th Ann. Conf. of Cog. Sci. Soc.*, 1986.

[K$^+$69] R.E. Kalman et al. *Topics in Mathematical System Theory.* McGraw-Hill, New York, 1969.

[Kal60] R.E. Kalman. A new approach to linear filtering and prediction problems. *ASME J. Bas. Eng.*, pages 35–45, 1960.

[Keh90] A. Kehagias. Optimal control for training: the missing link between Hidden Markov Models and connectionist networks. *Math. Comp. Mod.*, Vol. 14, pages 284-289, 1990.

[Keh91] A. Kehagias. *Approximation and Estimation of Stochastic Processes by Hidden Markov Models.* PhD thesis, Division of Applied Mathematics, Brown Un., Providence, Rhode Island, May 1991.

[Keh1] A. Kehagias. Stochastic Recurrent Networks: Prediction and Classification of Time Series. Technical Report, Division of Applied Mathematics, Brown Un., Providence, Rhode Island.

[Keh2] A. Kehagias. Stochastic Recurrent Networks: Representation Properties. Technical Report, Division of Applied Mathematics, Brown Un., Providence, Rhode Island, – In preparation.

[Keh3] A. Kehagias. Stochastic Recurrent Networks: Unification and Extension. Technical Report, Division of Applied Mathematics, Brown Un., Providence, Rhode Island, – In preparation.

[Lai71] D.G. Lainiotis. Optimal adaptive estimation: Structure and pattern adaptation. *IEEE AC*, 16(2):160–170, 1971.

[L+89] J. L. Lebowitz et al. Probabilistic cellular automata: Some statistical mechanical considerations. Technical report, Dept. of Mathematics, Rutgers Un. New Brunswick NJ, 1989.

[Leb90] J.L. Lebowitz. Statistical mechanical porperties of probabilistic cellular automata. *J. Stat. Phys.*, 59(1/2):117–170, 1990.

[L+83] S.E. Levinson et al. An introduction to the application of the theory of probabilistic functions of a Markov Chain. *The Bell Sys. Tech. J.*, 62(4), April 1983.

[Lig85] T. Ligget. *Interacting Particle Systems.* New York. Springer, New York, 1985.

[Lin88a] R. Linsker. How to generate ordered maps by maximizing the mutual information between input and output signals. Technical Report RC 14624(Nr.65530), IBM Research Division - T.J. Watson Research Center, Yorktown Heights NY, 1988.

[Lin88b] R. Linsker. An application of the principle of maximum information preservation to linear systems. Technical Report RC 14195(Nr.63478), IBM Research Division - T.J. Watson Research Center, Yorktown Heights NY, 1988.

[M+53] N. Metropolis et al. Equations of state calculations by fast computing machines. *J. Chem. Physics*, 21(1087-1092), 1953.

[May82] P.S. Maybeck. *Stochastic Models Estimation and Control.* Mathematics in Science and Engineering. Academic, New York, 1982.

[NT89] K.S. Narendra and M.A.L. Thathachar. *Learning Automata: an Introduction.* Prentice-Hall, Englewood Cliffs New Jersey, 1989.

[Nea90] R.M. Neal. Learning Stochastic Feedforward Networks. Technical Report CRG-TR-90-7, Dept. of Computer Science, Un. of Toronto, November 1990.

[Now90a] S.J. Nowlan Maximum Likelihood Competition in RBF Networks. Technical Report CRG-TR-90-2, Dept. of Computer Science, Un. of Toronto, February 1990.

[Now90b] S.J. Nowlan. Competing Experts: An Experimental Investigation of Associative Mixture Models. Technical Report CRG-TR-90-5, Dept. of Computer Science, Un. of Toronto, September 1990.

[Now90c] S.J. Nowlan. Maximum Likelihood Competitive Learning. In *Advances in Neural Information Processing Systems 2*, ed. D. Touretzky. Morgan Kauffman, 1990.

[Pea86] B.A. Pearlmutter and G. Hinton. Learning state space trajectories in recurrent neural nets. In *Proc. of IJCNN*. IEEE, 1986. In J.S. Denker, editor, *Neural Networks for Computing*, pages 333–338. American Institute for Physics, 1986.

[Paz71] A. Paz. *An Introduction to Probabilistic Automata*. Academic, New York, 1971.

[Pin88] F. Pineda. Generalization of backpropagation to recurrent and higher order neural networks. In D. Anderson, editor, *Neural Information Processing Systems*, 1988.

[Pol87] J. B. Pollack. Cascaded back propagation on dynamic connectionist networks. In *Proc. of the 9th Ann. Conf. of the Cog. Sci. Soc.*, 1987.

[R+88] Riedel et al. Temporal sequences and chaos in neural networks. *Phys. Rev. A*, 36:1428, 1988.

[Rob88] Anthony J. Robinson. A dynamic connectionist model for phoneme recognition. Technical report, Cambridge University Engineering Department, Cambridge, England, 1988.

[Rob89] Anthony J. Robinson. *Dynamic Error Propagation Networks*. PhD thesis, Cambridge University Engineering Department, Cambridge, England, 1989.

[Roh90] R. Rohwer. The moving targets training algorithm. In *Proc. of the EURASIP*, 1990.

[RR90] S. Renals and R. Rohwer. A study of network dynamics. *J. Stat. Phys.*, In Press, 1990.

[Rue69] D. Ruelle. *Statistical Mechanics*. Mathematical Physics Monographs. Addison-Wesley, Reading Mass., 1969.

[Rab88] L.R. Rabiner. A tutorial on HMM and selected applications in speech recognition. *Proc. of the IEEE*, 1988.

[Sol88] S. Solla. Accelerated learning experiments in layered neural networks. *Complex Systems*, 2, 1988.

[Sun89] R. Sun. The discrete neuronal model and the probabilistic discrete neuronal model. In *Neural and Intelligent System Integration*, (ed. B. Soucek). Wiley, 1991.

[Son90] E. D. Sontag. Feedback stabilization using two-hidden-layer nets. Technical Report SYCON-90-11, Rutgers Center for Systems and Control, Rutgers Un. New Brunswick New York, 1990.

[Sut88] R.S. Sutton. Learning to predict by the methods of temporal difference. *Machine Learning*, 3:9–44, 1988.

[Vas69] L.N. Vasherstein. Markov processes over denumerable products of spaces describing large systems of automata. *Problemy Peredachi Informatsii*, 5(3):64–72, 1969.

[W+89a] A. Waibel et al. Modularity and scaling in large phonemic neural networks. *IEEE ASSP*, ASSP-37(12):1888–1898, December 1989.

[W+89b] A. Waibel et al. Phoneme recognition using time-delay neural networks. *IEEE ASSP*, 37(3):328–339, March 1989.

[WZ88] R.J. Williams and D. Zipser. A learning algorithm for continually running fully connected recurrent neural networks. Technical Report ICS-8805, Un. of California at San Diego, 1988.

[Wol86] S. Wolfram. *Theory and Application of Cellular Automata*. World Scientific, Singapore, 1986.

[Wu83] C.F.J. Wu. On the convergence properties of the EM algorithm. *The Annals of Probability*, 11(1):95–103, 1983.

A SRN's with polynomially nonlinear units

The local conditional formulation used in Section 2 does not conform fully to classical connectionist models. These are usually built from units with a nonlinear (sigmoid) input/output response function. That is, unit s receives a deterministic input Z_s^t and a white noise input V_s^t and produces output X_s^t. We have the following relationships:

$$Z_s^t = \sum_{r \in N_h(s)} w_{sr} X_r^{t-1} + \sum_{r \in N_i(s)} w_{sr} U_r^t.$$

$$X_s^t = f(Z_s^t - V_s^t).$$

where f is a nonlinear function, w_{sr} are connectivity weights and V_s^t is a random number uniformly distributed in some appropriate range. For deterministic response we take V_s^t to be a fixed number, usually called the **threshold** of the unit s.

The two models are not incompatible. Trivially, a recurrent network of sigmoidal units has local conditional probabilities (possibly concentrating all probability mass on one output, if the units have a deterministic input/output response) and can be specified in terms of a $(\mathcal{G}, \mathcal{P})$ pair. In the rest of this section we will show that, conversely, any $(\mathcal{G}, \mathcal{P})$ SRN can be built from units with nonlinear (but not necessarily sigmoidal) random input/output response function.

Consider for simplicity a SRN $(\mathcal{G}, \mathcal{P})$ where the units are binary (i.e. the alphabet $A = \{0, 1\}$). We will design a network of nonlinear units that has the same graph/topology $(S, \mathcal{N})$ and the same local conditional $\mathcal{P}$.

We retain the topology of $(S, \mathcal{N})$. We use input/output units $s \in S$ has deterministic input U_s^t, random input V_s^t and output X_s^t. To express the relationship between these quantities, use an intermediate vector Z^t with:

$$Z_s^t = \sum_{r \in N_h(s)} w_{sr} X_r^{t-1} + \sum_{r \in N_i(s)} w_{sr} U_r^t.$$

Then we have:

$$X_s^t = H(f_s(Z_s^t) - V_s^t).$$

and V_s^t is a random number uniformly distributed in the interval $[0, 1]$. Also, w_{sr} are connectivity weights, $H(.)$ is the Heaviside step function and f_s is a nonlinear function *specific* to unit s. That is, every unit has its own response function. We take all of these functions to be polynomials:

$$f_s(z) = \alpha_{s0} + \alpha_{s1} z + ... + \alpha_{sn_s} z^{n_s}.$$

Now the output X_s^t will be 0 if $f_s(Z_s^t)$ is less than V_s^t and 1 otherwise. Then we have the following probabilities:

$$Prob(X_s^t = 1 | X_{N_h(s)}^{t-1} = x, U_{N_i(s)}^t = u) = Prob(f_s(< w_s, xu >) > V_s^t) =$$

$$\alpha_{s0} + \alpha_{s1} < w_s, xu > + ... + \alpha_{sn_s} < w_s, xu >^{n_s} .$$

On the other hand we want $Prob(X_s^t = 1 | X_{N_h(s)}^{t-1} = x, U_{N_i(s)}^t = u)$ to be equal to the local conditional $p_s(1|x,u)$. So we must find α_{sr}, $s,r \in S$ such that for all $s \in S$, $\forall x \in A^{|N_h(s)|}$, $u \in A^{|N_i(s)|}$ we have

$$\alpha_{s0} + \alpha_{s1} < w_s, xu > + ... + \alpha_{sn_s} < w_s, xu >^{n_s} = p_s(1|x,u). \tag{20}$$

This is a linear equation in the α's. The following conditions are sufficient to guarantee the existence of a solution:

1. The degree n_s of the polynomial in (20) is large enough,
2. The matrix $[< w, xu >^n]_{xu \in A^{M_h+M_i}, 1 \leq n \leq n_s}$ is nonsingular.

If conditions **1**, **2** are satisfied, the solution to (20) is easy to compute.

We can choose n_s as large as we please; to ensure that the matrix is nonsingular, choose the connectivity weights as follows: $w_{sr} = 2^r$, for all $s \in S$, $r = 1, ..., n_s$. Then **1**, **2** are satisfied and we have obtained the desired values for the local conditionals $p_s(1|x,u)$. As $p_s(0|x,u)=1 - p_s(1|x,u)$, it follows that $p_s(0|x,u)$ also have the desired values. So we have reproduced the local conditionals exactly and we have built the specified network $(\mathcal{G}, \mathcal{P})$ from *polynomial* units. Note that we can write the evolution equations for the hidden and output vectors as

$$X^t = g(X^{t-1}, U^t, V^t), \tag{21}$$

$$Y^t = h(X^t, U^t, W^t). \tag{22}$$

This representation will be useful (we use it in [Keh1], [Keh3]) because of the analogy with control systems [May82]. The case of non-binary alphabet A is more complicated, but can be solved similarly. The details will be reported elsewhere [Keh2].

We have two network descriptions: one is in terms of local conditionals, the other in terms of nonlinear functions and connection weights. In either case, a certain list of numbers has to be remembered by every unit (we can think of the unit as a small processor with a limited amount of memory); these numbers are either the local conditionals or the polynomial coefficients. In a certain sense the really important quantities are the local conditionals, because these can be computed (via the local BF algorithm, see Section 4) much faster than the weights. However, once the local conditionals are known, several representations are possible. We have a choice of storing the full set of local conditionals, or a full set of α coefficients that represent exactly the local conditionals, or finally, a reduced set of coefficients that correspond to a lower order polynomial. A low order polynomial can be chosen that *approximates* the local conditionals. We can use a least squares approximation; the problem is linear in in the α coefficients and can be solved easily. We have to be careful that we do not get negative probability values as a result of the approximation. The robustness of Markov chains to small perturbations should minimize the degradation of performance because of approximation.

B EM and the Boltzmann Machine

The Boltzmann machine is an example of a recurrent stochastic network. In the classical paper [A+85] by Ackley, Hinton and Sejnowski it is described as follows: "*... The machine is composed* *of primitive computing elements called units that are connected to each other by bidirectional links. A unit is always in one of two states, on or off, and it adopts these states as a probabilistic function of the states of its neighboring units and the weights on its links to them. ... The units of a Boltzmann machine partition into two functional groups, a nonempty set of visible units and a ... set of hidden units ...* ". Ackley et al. proceed to set up the 8-3-8 encoder problem as follows: "*Two groups of visible units, designated as V_1 and V_2 represent two systems that want to communicate their states. ... Each enviromental vector in turn was clamped over the visible units ... If the energy gap between the on and off states of the k-th unit is ΔE_k then regardless of the previous state set s_k=1 with probability*

$$p_k = \frac{1}{1 + e^{-\Delta E_k / T}} \tag{23}$$

where T is a parameter that acts like temperature...". They proceed to describe a rather complicated parameter estimation scheme, which will be discussed presently.

Let us place their model in the framework of stochastic recurrent networks. Translating their terminology to ours, $V_1 = S_i$, $V_2 = S_o$, $H = S_h$ (where H is the set of hidden units). Our conditional probability function is in their case expressed by (23); this is a relatively simple form where ΔE_k is a function of the state of the neighbors and the weights (network parameters). To be more explicit, we rename their node k to s, with an input neighborhood $N_h(s) \cup N_i(s)$. Then the p_k in (23) is in our notation $p_s(1 \mid x_{N_h(s)}, u_{N_i(s)})$ and is given by

$$p_s(1 \mid x_{N_h(s)}, u_{N_i(s)}) = \frac{1}{1 + e^{(E(0, x_{N_h(s)}, u_{N_i(s)}) - E(1, x_{N_h(s)}, u_{N_i(s)}))/T}} \tag{24}$$

where $E(.)$ is an appropriate energy function. Thus (24) is the translation of (23) in our notation.

Now let us consider the connections between the classical training of the Boltzmann machine, the EM method and our local BF algorithm.

We have explained the general EM idea in Section 4. Let us see how the classical Boltzmann Machine estimation procedure, as presented in [A+85] fits in this scheme. This procedure is outlined by Ackley et al. as follows:

"...

1. *Estimation of p_{ij}: Each enviromental vector in turn was clamped over the visible units ... Statistics about how often pairs of units were both on together were gathered at equilibrium...*

2. *Estimation of p'_{ij}: The network was left completely unclamped and allowed to reach equilibrium ... Statistics about cooccurences were then gathered for as many annealings as were used to estimate p_{ij}.*

3. *Updating the weights: All weights in the network were incremented or decremented by fixed weight step, with the sign of the increment being determined by the sign of $p_{ij} - p'_{ij}$. ...*"

The EM character of this procedure should be obvious. Steps 1 and 2 correspond to the expectation phase (the expectations are computed by thermodynamic *relaxation* and collection of empirical statistics) and Step 3 corresponds to the optimization phase: Ackley et al. show in their Appendix that the gradient of their cost function with respect to weight w_{ij} is proprtional to $p_{ij} - p'_{ij}$. Therefore their update rule in Step 3 points to approximately the steepest descent direction.

So, BM training by Relaxation, as well as our local BF training are both special cases of EM method. Comparison of local BF training for the encoder problem (Section 5) with the corresponding section from [HS86] shows the clear superiority of local BF, especially in speed of learning. On the other hand Boltzmann training by simulated annealing is guaranteed to arrive to the global maximum of the cost function if a slow annealing schedule is used. No such claim can be made for local BF. However, considering the standard practice of speeding up annealing schedules, this point becomes somewhat moot.

C SRN: a Species of Hidden Markov Models

As we have already pointed out, the motivation for SRN training comes from the field of Hidden Markov Models (HMM) [BE67, L+83]. The local BF algorithm is a modification of the Baum BF algorithm. Another point that is quite obvious is that the SRN model is special case of a HMM; in this appendix we discuss this point in more detail.

A HMM involves two stochastic processes $\{X^t\}_{t=1}^{\infty}$ and $\{Y^t\}_{t=1}^{\infty}$. The hidden process $\{X^t\}_{t=1}^{\infty}$ is Markov, that is:

$$Prob(X^t|X^{t-1}, X^{t-2}, ...) = Prob(X^t|X^{t-1}). \tag{25}$$

The output process $\{Y^t\}_{t=1}^{\infty}$ depends "instantaneously" on the hidden process. That is:

$$Prob(Y^t|...X^{t+1}, X^t, X^{t-1}, ...) = Prob(Y^t|X^t). \tag{26}$$

On the other hand, we know that the hidden and output processes of a SRN satisfy:

$$Prob(X^t|X^{t-1}, X^{t-2}, ..., U^t, U^{t-1}, ...) = Prob(X^t|X^{t-1}, U^t). \tag{27}$$

$$Prob(Y^t|...X^{t+1}, X^t, X^{t-1}, ..., U^{t+1}, U^t, U^{t-1}, ...) = Prob(Y^t|X^t, U^t). \tag{28}$$

It is obvious that, if we consider the (given) input U^t as a parameter, (25),(26) is exactly the same as (27),(28). So SRN is a HMM. The only difference from traditional HMM's, is that here the processes are not time invariant but vary in time according to the value of input U^t at time t. For instance, say that the input $U^1 = u^1, U^2 = u^2, ...$ is fixed. Now the Markov Chain $\{X^t\}$ is not *homogeneous*, but *inhomogeneous*, since

$$Prob(X^t = x|X^{t-1} = z; U^t = u^t) = P_{zx}(t).$$

In other words the transition probability matrix is time varying. The same is true of the emission probability matrix:

$$Prob(Y^t = y|X^t = x; U^t = u^t) = Q_{xy}(t).$$

Note that, any pair of processes that satisfy the equations

$$X^t = g(X^{t-1}, U^t, V^t), \tag{29}$$

$$Y^t = h(X^t, U^t, W^t), \tag{30}$$

is a HMM. This and the discussion in Appendix A provide another point of view that places SRN as a special case of HMM. The representation above is also very common in the context of stochastic control [May82]. The methods of stochastic control and HMM are quite different, but both are well developed and it appears that a combination of methods from the two fields will provide new powerful techniques for the study of SRN such as described by (29),(30).

One last point with respect to modelling. As we have already pointed out, we can think of local BF as a HMM estimation algorithm that computes local transition probabilities, rather than the global transition matrix of the system. Now, remember that a SRN of M_h hidden binary units has 2^{M_h} states. So we can implement large HMM's (many states) with relatively small networks (few units). This allows powerful modelling. On the other hand, using a SRN with M_h units we *cannot* get every HMM with 2^{M_h} states. [2] The reason is that the global transition matrix P of a SRN cannot be of the most general form possible; it is constrained by the fact that every element $[P]_{xz}$ of the transition matrix has a special form ($[P]_{xz} = \prod_s p_s(x_s|z_{N_h(s)})$). Counting of parameters shows that there are fewer local conditionals $p_s(x_s|x_{N_h(s)})$ than global transition probabilities $[P]_{xz}$. In other words we have a limited number of variables at our disposal. This is not necessarily a bad thing, however. In fact it is a way to deal with the problem of overfitting (i.e. too many parameters must be estimated from too few datapoints). In statistical terminology, the local HMM is more *parsimonious* than the global HMM.

D Alternative Viewpoints

This appendix is a very brief tour of some alternative points of view on the object of our study, namely SRN's. Many disciplines, outside of Connectionism, have dealt with the same object (although calling it different names): a collection of randomly operating interconnected finite state units. We review very briefly the work done along these lines. We make no claim for completeness; we are considering here several very well developed research programs with extremely voluminous literature. Our aim is to just point out some of the connections. In [Keh3] we will try to synthesize them into a more coherent picture.

There is a certain interchangeability between our SRN model and the more classical connectionist network with nonlinear input/output units (see Appendix A); the Boltzmann Machine also fits in our formulation very naturally (see Appendix B). There is also a number of not-so-standard connectionist models which have even more obvious resemblances to ours: see [Pea86, Ale89, Gil90, Sun89, Nea90] etc. Another group of researchers study Learning Automata [BA85, NT85], an area which intersects with Connectionism but is, in fact, a discipline in its own right.

Another discipline with its own history and research program, which strongly reminds us of connectionism in general and SRN's in particular, is the study of Cellular Automata (CA) [Wol86]. In particular, probabilistic cellular automata are essentially identical to SRN's. The tasks which are usually performed by CA are, however, quite different from those of connectionist networks. An area of particular interest for CA researchers is hydrodynamics and chaos [dL88]

[2] We will prove in [Keh2] that we *can* get every HMM with 2^{M_h} states using SRN's with *more* than M_h binary hidden units.

which are studied at a microscopic level by looking at the evolution of small finite state units ("molecules" ?).

This leads to another group of researchers, most of them mathematical physicists, who try to develop statistical physics from microscopical foundations [Ell85, M+53, L+89, Leb90, Lig85, Rue69, Vas69]. Terms such as Ising model, simulated annealing etc. have entered connectionist mainstream from the work of this group. This work is strongly connected to the study of Markov Random Fields [Dob65, Dob70, Gid85]. And Markov Random Fields theory has in turn some very striking applications in Image Processing [GG84]. [GG84] is an oft-quoted paper in connectionism and this is not an accident: the convergence analysis of simulated annealing presented there applies equally well to Relaxation methods [Ack+83] used for Boltzmann Machine training. So we have come full circle back to connectionism.

In fact, all the objects we have mentioned above are special cases of finite state, probabilistic machines [Paz71, Boo67]; such machines are in turn a special case of *machines* in the algebraic sense, as considered by Kalman and Arbib in [Kal69].

Now, the abstract development of [Kal69] is impressively general, but when we look for specific examples of their theory we have a rather limited choice. We are basically looking at models of the form:

$$X^t = g(X^{t-1}, U^t, V^t), \tag{31}$$

$$Y^t = h(X^t, U^t, W^t). \tag{32}$$

Here X^t is a hidden process and Y^t is the output process. U^t is the "control" process and V^t, W^t are white noise that adds (optional) randomness to the system. As already pointed out in Appendix A, the SRN model presented here fits in the framework of (31), (32). Another set of models that fit in this framework, are control systems [May82]; however, practically all systems studied in control theory until very recently involved processes in continuous spaces (very often the state space R^n). Apart from that difference, control theory deals with many problems that are extremely similar to those of connectionism, e.g. producing a desired input/output relationship, predicting future signals, classifying ("identifying") signals etc. This suggests that we try to combine the points of view of control and connectionism. Some work already exists that combines Control and Connectionist points of view; see [Bar89, Dre90, Keh90, Son90]. There is still, however, ample scope for an application of the control theoretic point of view to connectionism. A strategy that seems particularly promising, is the use of stochastic control methods in the study of SRN's. This is pursued in [Keh1].