A General Feed-Forward Algorithm for Gradient Descent in Connectionist Networks

Sebastian Thrun · Frank Śmieja

GMD – German National Research Center for Computer Science
D – 5205 Sankt Augustin, Postfach 1240, West Germany
e-mail: st@gmdzi.uucp

A General Feed-Forward Algorithm for Gradient Descent in Connectionist Networks

Sebastian Thrun · Frank Śmieja

GMD – German National Research Center for Computer Science
D – 5205 Sankt Augustin, Postfach 1240, West Germany
e-mail: st@gmdzi.uucp

Abstract

An extended feed-forward algorithm for recurrent connectionist networks is presented. This algorithm, which works locally in time, is derived both for discrete-in-time networks and for continuous networks. Several standard gradient descent algorithms for connectionist networks (e.g. [48], [30], [28] [15], [34]), especially the backpropagation algorithm [36], are mathematically derived as a special case of this general algorithm. The learning algorithm presented in this paper is a superset of gradient descent learning algorithms for multilayer networks, recurrent networks and time-delay networks that allows any combinations of their components.

In addition, the paper presents feed-forward approximation procedures for initial activations and external input values. The former one is used for optimizing starting values of the so-called context nodes, the latter one turned out to be very useful for finding spurious input attractors of a trained connectionist network.

Finally, we compare time, processor and space complexities of this algorithm with backpropagation for an unfolded-in-time network and present some simulation results.

Keywords: gradient descent, backpropagation, recurrent connectionist networks, inversion

This research was supported by grant ITR 8800 L7 from the German Federal Ministry for Scientific Research and Technology (BMFT).

Contents

1 Introduction **3**

2 Formal Definition of a Recurrent Connectionist Network **5**
2.1 The Graph 5
2.2 Activations and Weights 5
2.3 Activation Functions 5
2.4 The Squashing Function 7
2.5 Restrictions for Discrete Networks 9

3 Gradient Descent **10**
3.1 The Error Measure 10
3.2 The Learning Algorithm 11
3.3 Computability of the Gradients (β^k_{ijn}) 14
3.4 The Pattern Set 15

4 Online Learning **16**

5 Teacher Forcing **18**

6 Special Cases of the Gradient Descent Algorithm **19**
6.1 Non-recurrent Multilayer Networks 19
6.2 Fully Connected Recurrent Connectionist Networks 20
6.3 Further Special Cases 21

7 Adaption of External Quantities by Means of Gradient Descent **22**
7.1 Adaption of Initial Activations 22
7.2 "Inversion": Adaption of the External Input 23

8 Complexity **26**

9 Simulation Results **29**
9.1 Continuous Trajectory Learning 29
9.2 Pattern Classifying and Inversion with a Discrete-in-Time Network 29

10 Appendix: The General Gradient Descent Algorithm **37**

1 Introduction

Since the renewed introduction of the paradigm of connectionist networks into computer science in the '80s, the backpropagation algorithm [36] has been gaining in importance. This algorithm is nothing other than an approximation procedure of adaptive internal parameters (weights, thresholds etc.) for generating a desired input-output mapping. Starting from a randomly selected function in the form of randomly selected initial values for the adaptive parameters, the current difference between actual and desired mapping is measured with a differentiable error measure; the internal parameters are adapted step by step according to the method of *steepest descent* with respect to this error measure, until the error gets smaller than some small remainder ε. This technique, which has long been known in mathematics, gained in importance with its application to connectionist networks whose structure was motivated by models of human neural networks.

First subjects of investigation were multilayer structures with no recurrencies and no time contexts, which were mainly used for classification and filtering. These multilayer networks filter or classify the externally given input pattern by propagating it layer by layer through the network in the form of activations, each layer processing the output of the preceding layer in a non-linear way. The output of the last layer is then compared with a given target and the gradient of the error function with respect to the output activations, the error gradient, is computed. For gradient descent it is now sufficient to transmit the error only to the last layer, the output layer, directly. It is then propagated back layer by layer through the network in the opposite direction. In this way each layer modifies its parameters in small steps with respect to the error gradient. The name *backpropagation* originates from this inversion of the propagation direction [36].

Some years later, after the mathematical possibilities and limits of the backpropagation algorithm in multilayer feed-forward connectionist networks had been quite well studied, attention was directed more and more to *recurrent networks*. In this case, the network is used for classifying or filtering entire sequences of input patterns by "storing" information by links provided with time-delay values, i.e. information from the past is considered in current classification. These links violate the unidirectional structure of the above multilayer networks and so they are called "recurrent" links.

For applying the backpropagation algorithm to these recurrent networks, too, first a method presented in [36] was taken up. The recurrent network is mapped to an equivalent purely unidirectional network by generating a separate copy of the network for each time step. The connections within these subnetworks consist of all non-recurrent and therefore unidirectional links of the original network while subnetworks are interconnected unidirectionally by the formerly recurrent links. This technique, also referred to as *unfolding in time* [36], allows the application of the backpropagation algorithm to recurrent networks, but the whole past of a sequence till the analysis of the error has to be stored which increases the required storage linearly to the sequence length.

This generally undesired effect was avoided by a modified mathematical approach, which propagated specific auxiliary gradients together with the activations through time, i.e. purely forward [32, 48, 26, 34]. Although these networks do not show an inversion of the propagation direction any more, the term "backpropagation" sometimes was retained.

In [32, 48] a *feed-forward* gradient descent algorithm for fully recurrent connectionist networks is presented, which avoids the effect of linearly storage costs mentioned above by skipping the backward-phase. Unfortunately, in their derivation each link has a fixed time delay of one time unit, which makes the integration of conventional non-delayed multilayer structures into a connectionist network difficult. In addition, asynchronous recurrent networks which do not work synchronously at fixed time steps only, but which work (more or less) continuously in time have gained in interest [30, 28, 15, 13]. Since current digital computers do not allow a continuous propagation of activations through time, implementations of such algorithms are very similar to those of synchronous algorithms.

In this paper, we present a general gradient descent algorithm for connectionist networks which does not require a backward phase and which works locally in time. In addition to recurrent structures of any type, conventional multilayer structures can also be integrated into the network. The algorithm is applicable to synchronous discrete-in-time networks as well as to continuous-in-time networks so that the algorithm presented here can be regarded as a generalization of many backpropagation and gradient descent algorithms used so far in this area.

To date many modifications of the standard backpropagation algorithm for accelerating of convergence or improving the generalization behavior have been investigated [9, 12, 18, 23, 36]. We do not consider these modifications here but concentrate on the strict gradient descent algorithm, although many of those could be easily included into our algorithm.

We begin with a formal definition of a connectionist network in the sense of the algorithm which we require later for a proof of computability. Two different activation functions, one discrete in time and one continuous, are presented and gradient descent is derived for both. Following a short computability proof and the description of the techniques of "*online learning*" and "*teacher forcing*", the unidirectional multilayer algorithm developed by [36] and the recurrent algorithm [48] are explicitly derived as examples of gradient descent algorithms for recurrent connectionist networks.

Subsequently, a *feed-forward* gradient descent method is derived for adapting the initial activations with respect to the error function. The initial activations the nodes receive before the first propagation are an important quantity which influence the subsequent computation. Since optimal initial activations are usually unknown a priori, they are often specified by uniform *don't-know* values. But they can be optimized by gradient descent as well.

In addition, a *feed-forward* inversion algorithm is presented. This algorithm is used for finding input sequences for a given output target for an already adapted network [47, 24, 22, 42].

Having described all this, we proceed by comparing complexities of serial and parallel implementations of the feed-forward algorithm with the *unfolding-in-time* method. Since we will not describe this method formally in this paper, see [36] for further details.

Lastly, we present some simulation results, namely continuous trajectory learning and pattern classifiaction and inversion with a discrete-in-time network and discuss them shortly.

2 Formal Definition of a Recurrent Connectionist Network

2.1 The Graph

Definition:
A *connectionist network* is a *directed graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with *node set* $\mathcal{V}$ and *set of links* $\mathcal{E}$ (*edge set*). The node set is finite (in its generalization, countable is sufficient): $\mathcal{V} \equiv \{v_1, v_2, ..., v_N\}$ for a $N \in I\!N$. Let $\mathcal{N} \equiv \{1, 2, ..., N\}$ be the index set of the nodes. Without loss of generality, set of links (edges) $\mathcal{E}$ is a subset of $\{\langle v_i, v_j \rangle_n |\ v_i, v_j \in \mathcal{V} \wedge n \in \mathcal{S}\}$ for a $\mathcal{S} \overset{finite}{\subset} I\!R_0^+$. $\mathcal{S}$ is the *set of (time-) delays* of the network and $\hat{n} \equiv \max \mathcal{S}$ the *maximum delay* of all links, a network-specific constant.

A subset $\mathcal{O} \in \mathcal{N}$ of the node indices indicates the set of *output nodes*: $\{v_k |\ k \in \mathcal{O}\}$ whose function is discussed in section 3.1.

2.2 Activations and Weights

Each node $v_i \in \mathcal{V}$ is assigned a real, bounded *activation value (activity)* $x_i(t)$. This activation changes in time according to fixed rules, and is labeled by a *time index* t which is an element of a fixed working interval $[t_0, t_1]$; t_0 is the *start time* and t_1 the *stop time*. In addition, each link $\langle v_i, v_j \rangle_n \in \mathcal{E}$ has associated a *weight* $w_{ijn} \in I\!R$ which, in general, does not change in the interval $[t_0, t_1]$ (otherwise it can also be provided with a time index: $w_{ijn}(t)$).

Note that

- In the following formulas, in the case of partial connectivity, all links which are not available in $\mathcal{E}$ are regarded as links with a fixed weight 0.0 to avoid time-consuming and confusing case differentiations in summations of weight expressions.
- A *threshold (bias) value* which occurs in many activation functions in the literature is realized by adding a special node v_* to the node set. This special node always has the maximum activation (e.g. 1.0) and is a predecessor of all other nodes $v_i \in \mathcal{V} \backslash \{v_*\}$. The weight $\langle v_i, v_* \rangle_0$ is often written as θ_i.

2.3 Activation Functions

In this context, we distinguish *discrete-in-time* and *continuous-in-time* connectionist networks. We name the former ones *discrete* and the latter ones *continuous*.

a) Discrete Networks:

Discrete networks are characterized by an activation function which is discrete-in-time: activations are defined only at the discrete points of time $t_0, t_0 + 1, t_0 + 2, ..., t_1$. (Hence it follows that $t_1 - t_0$ is an integer.)

Due to the connectivity between two nodes, the following holds: $\mathcal{S} \subseteq \{0, 1, ..., \hat{n}\}$, i.e. between each node pair (i, j), there are at most $\hat{n} + 1$ links which are indexed with the delay values $0, 1, ..., \hat{n}$. The link $\langle v_i, v_j \rangle_n$ is in the direction node v_j to node v_i; n is defined as the *(time-) delay* of the link $\langle v_i, v_j \rangle_n$.

In the discrete case, this paper is based on the following activation function from which many of the activation functions quoted in literature can be derived:

$$x_i(t) \equiv \sigma \left(\sum_{j \in \mathcal{N}} \sum_{n=0}^{\min\{t-t_0, \hat{n}\}} w_{ijn} x_j(t-n) \right) + I_i(t) \quad \forall i \in \mathcal{N} \tag{1}$$

where

- σ is a *squashing function*, which is primarily used for bounding and which is discussed in more detail in Chapter 2.5.
- $I_i(t)$ is the *external input* which is fed into the network at the node i and at time t.
- For the set of discrete delays $\mathcal{S}$, the following holds: $\mathcal{S} \subseteq \{0, 1, ..., \hat{n}\}$ for a fixed $\hat{n} \in I\!N_0$. That means nothing else but that only integer values up to $\hat{n}$ may be delays. Therefore

$$\{n \in \mathcal{S} |\ n \leq t - t_0\} \subseteq \{0, 1, ..., \min\{t - t_0, \hat{n}\}\}. \tag{2}$$

- In addition, as in the case of the continuous activation function (see below),

$$net_i(t) \equiv \sum_{j \in \mathcal{N}} \sum_{n=0}^{\min\{t-t_0, \hat{n}\}} w_{ijn} x_j(t-n) \tag{3}$$

  is defined as the *net-input* (network-internal input) of the node i at time t.

- The *initial activations* $\tilde{x}_i$ are also external quantities which usually have an effect on the activation trajectory which is similar to that of the external input. They initialize the activations of the network: $\forall i \in \mathcal{N}$

$$x_i(t_0) \equiv \tilde{x}_i \tag{4}$$

b) Continuous Networks:

Continuous networks are characterized by a system of coupled differential equations for activation computation: $\forall i \in \mathcal{N}, t \in]t_0, t_1]$

$$\begin{aligned} x_i(t_0) &\equiv \tilde{x}_i \\ T_i \frac{dx_i}{dt}(t) &\equiv -x_i(t) + \sigma \left(\sum_{j \in \mathcal{N}} \sum_{\substack{n \in \mathcal{S} \\ n \leq t - t_0}} w_{ijn} x_j(t-n) \right) + I_i(t) \end{aligned} \tag{5}$$

where

- T_i is an adaptive time constant which controlles the integration speed. A gradient descent algorithm is also derived for these integration constants.
- $\mathcal{S} \overset{finite}{\subset} \mathbb{R}_0^+$, a finite subset of non-negative real numbers, is the *set of delay values* in network $\mathcal{G}$.

Concerning the activation computation on a digital computer, it should be noted that continuous activation trajectories cannot be computed in principle if the differential equations are not solved. Therefore we confine ourselves to regard the gradient in small intervals $[t, t+\Delta t]$ (with $\Delta t > 0$) as constant and thus to compute the subsequent actions in discrete time steps as well. The resulting forward-propagatable activation functions are as follows:

$$
\begin{aligned}
T_i \frac{\Delta x_i}{\Delta t}(t) &= -x_i(t) + \sigma\left(\sum_{j\in\mathcal{N}} \sum_{\substack{n\in\mathcal{S} \\ n\le t-t_0}} w_{ijn} x_j(t-n)\right) + I_i(t) \\
\iff x_i(t+\Delta t) &= \left(1 - \frac{\Delta t}{T_i}\right) x_i(t) + \frac{\Delta t}{T_i}\sigma\left(\sum_{j\in\mathcal{N}} \sum_{\substack{n\in\mathcal{S} \\ n\le t-t_0}} w_{ijn} x_j(t-n)\right) + \frac{\Delta t}{T_i} I_i(t) \\
\iff x_i(t+\Delta t) &= \left(1 - \frac{\Delta t}{T_i}\right) x_i(t) + \frac{\Delta t}{T_i}\sigma\left(\sum_{j\in\mathcal{N}} \sum_{n=0,\Delta t,\ldots}^{\min\{t-t_0,\hat{n}\}} w_{ijn} x_j(t-n)\right) + \frac{\Delta t}{T_i} I_i(t) \\
&\qquad \forall i \in \mathcal{N}, t \in [t_0, t_1 - \Delta t] \qquad (6)
\end{aligned}
$$

for some $\Delta t > 0$.
(For $\Delta t \longrightarrow 0$, the discrete activation functions approximate the underlying differential equations.)

From equation (6) it follows that $t_1 - t_0$ must be an integer multiple of Δt. Moreover, all delays $n \in \mathcal{S}$ should be an integer multiple of Δt.
A comparison of the two activation equations (1) and (6) shows that they are very similar. An advantage of the continuous network presented here is however that, after learning, the "fineness" of the time resolution Δt can be regulated.

2.4 The Squashing Function

Especially in the early times of the most recent investigation phase of connectionist networks, several different *squashing functions* were studied. The task of the squashing function is the bounding of the net-inputs and therefore the bounding of the activations:

$$\sigma : \mathbb{R} \longrightarrow]x^-, x^+[\qquad (7)$$

Here the *minimum activation* is x^-, the *maximum activation* x^+. In general, the squashing function is (strictly) monotonous, (continuous and) differentiable since differentiability is required for gradient descent algorithms.

In this context, we focus our attention to the prototype of sigmodial, semilinear squashing functions, the so-called *logistic function*:

$$\sigma(net_i) \equiv \frac{b}{1 + e^{-c \cdot net_i}} - a \tag{8}$$

The real constants $a, b, c \in I\!R$ define symmetry point, range and steepness of the squashing function:

- symmetry point of σ is $(0|\frac{b}{2} - a)$ and
- the range interval $]x^-, x^+[$ is $]\min\{-a, b-a\}, \max\{-a, b-a\}[$.
- The steepest point, the symmetry point, has the steepness (derivation) $\frac{1}{4}bc$.

We obtain the following derivation of the squashing function with respect to net_i:

$$\begin{aligned}
\sigma'(net_i) &= \frac{\partial \sigma(net_i)}{\partial net_i} \\
&= bc \frac{1}{1 + e^{-c \cdot net_i}} \left(1 - \frac{1}{1 + e^{-c \cdot net_i}}\right) \\
&= \frac{c}{b} \left(\sigma(net_i) + a\right) \left(b - a - \sigma(net_i)\right) \qquad (9)
\end{aligned}$$

Usually, we use the parameters

$$\begin{aligned}
a &= 1 \\
b &= 2 \\
c &= 1
\end{aligned}$$

This is the origin-symmetrical squashing function with activations in the interval $]-1, 1[$ and whose steepest point has the steepness $\frac{1}{2}$.
Squashing function and derivation are then:

$$\sigma(net_i) = \frac{2}{1 + e^{-net_i}} - 1 \tag{10}$$

$$\sigma'(net_i) = \tfrac{1}{2}(\sigma(net_i) + 1) \cdot (1 - \sigma(net_i)) \tag{11}$$

The use of other continuous, eventually non-monotonous or unbounded squashing functions have no influence on the algorithm and its derivation.

If a specific squashing function is assigned to each node in form of specific parameters a_i, b_i, c_i $(\forall i \in \mathcal{N})$ based on the squashing function (8), these parameters can also be used for minimizing the error function and can be adapted. This however will not be discussed in this paper.

2.5 Restrictions for Discrete Networks

The activation function for **discrete networks** (1) shows that following restrictions have to be made in the case of discrete or discontinuous networks:

Condition $(*)$:
There is no cycle $\{\langle v_{i_2}, v_{i_1}\rangle_0, \langle v_{i_3}, v_{i_2}\rangle_0, ..., \langle v_{i_c}, v_{i_{c-1}}\rangle_0, \langle v_{i_1}, v_{i_c}\rangle_0\} \neq \emptyset$ *in* $\mathcal{G}$.

In general, this restriction is required if the computation of the activation of a node at time t uses activations of other nodes at the same time. Without $(*)$, there is the danger of a computational deadlock: Each node of a subset of $\mathcal{N}$can compute its own activation only if some other nodes from the same subset have already computed their activations.

Theorem: The condition $(*)$ is equivalent to the following statement: $(**)$:
It is possible to find a new numbering of the node set $\mathcal{V}$ *for which the following holds:* $\forall i, j \in \mathcal{N}, i \leq j : \langle v_i, v_j\rangle_0 \notin \mathcal{E}$.

Proof:

"$\Longleftarrow$" Trivial.

"$\Longrightarrow$" (constructive) proof: Remove step by step, from the set of nodes $\mathcal{V}$, a node v to which no link $\langle v, u\rangle_0$ leads from a node u from (the remainder of) $\mathcal{V}$. At any time, however, such a node v has to be available since otherwise there is a cycle as described in $(*)$. Since there are only finitely many nodes in $\mathcal{V}$, the algorithm terminates with an empty set $\mathcal{V}$.
The new numbering is obtained by indexing the n-th node taken with the index n. The following holds for two nodes v_i and v_j of the new numbering: From $i < j$, it follows that node v_i was taken before node v_j. At the time of taking v_i, v_j was therefore still available in $\mathcal{V}$; hence $\langle v_i, v_j\rangle_0 \notin \mathcal{E}$.

The indexing of the new node set also determines the computation sequence of the activation values: as mentioned above, a node can only evaluate its activation if all its predecessor activations have been evaluated.

3 Gradient Descent

3.1 The Error Measure

The adaption of the weights by means of a gradient descent algorithm requires a *continuous error measure* E, which measures the present output error of the connectionist network. The derivation of this error measure indicates for the parameters which are to be adapted – namely the weights (w_{ijn}) and, in the continuous case, also the time constant (T_i) – the direction in which an infinitesimally small change has to be made in order to reduce the error.

Let $\zeta_i(t)$ be the error functional which describes the error at the output node $i \in \mathcal{O}$ at time t. In the discrete case the *general error function* to be minimized is as follows:

$$E \equiv \sum_{t=t_0}^{t_1} \sum_{k \in \mathcal{O}} \zeta_k(t) \tag{12}$$

and in the continuous case:

$$E \equiv \int_{t_0}^{t_1} \sum_{k \in \mathcal{O}} \zeta_k(t) dt \tag{13}$$

Note that the error function is of double summary character. First it sums up over time and, second, it sums up over all output units.

The momentary gradient $e_k(t)$ of this error function with respect to an activation x_k at time t is defined as:

$$e_k(t) \equiv \frac{\delta E}{\delta x_k}(t) = \frac{\partial \zeta_k}{\partial x_k}(t) \tag{14}$$

Often, the weighted (multiplied by $\frac{1}{2}$) *quadratic deviation* from a given trajectory is chosen as the error. If t_k represents this desired trajectory for the activation x_k of output node v_k for each $k \in \mathcal{O}$, we obtain the following error measure:

$$E = \sum_{t=t_0}^{t_1} \sum_{k \in \mathcal{O}} s_k(t) \tfrac{1}{2} (t_k(t) - x_k(t))^2 \tag{15}$$

$$\text{resp.} \quad E = \int_{t_0}^{t_1} \sum_{k \in \mathcal{O}} s_k(t) \tfrac{1}{2} (t_k(t) - x_k(t))^2 dt \tag{16}$$

t_k is also called a *target trajectory* of the output activation x_i. The *weighting* of the error $s_k(t)$ *in time* allows a weighting of the error depending on the time of its occurrence. Mostly, we select $s_i(t) \equiv 1$ (In [15] s is defined as $s_i(t) = \lambda^t$ for a $\lambda > 0$).

If E measures the quadratic deviation from a trajectory, the following holds:

$$e_k(t) = (x_k - t_k)(t) \tag{17}$$

$e_k(t)$ is here also the momentary error which is immediately measurable at any time.

3.2 The Learning Algorithm

The algorithm presented here goes together with the activation propagation forward in time. This requires additional considerations of some auxiliary quantities in activation computation. These quantities and their propagation rules are described in the following.

The fundamental approach of the gradient descent algorithm is to modify step by step each weight in the opposite direction of the error gradient (with respect to this weight). E.g., if the gradient is positive, a sufficient small reduction of the weight is likely to reduce the overall error – the same applies to the reciprocal case. In the discrete case, the updating rule is therefore:

$$\begin{aligned} \Delta w_{ijn} &\equiv -\eta \frac{\partial E}{\partial w_{ijn}} \\ &\overset{(12),(14)}{=} -\eta \sum_{t=t_0}^{t_1} \sum_{k \in \mathcal{O}} e_k(t) \frac{\partial x_k}{\partial w_{ijn}}(t) \end{aligned} \tag{18}$$

where $\eta > 0$ is a *stepwidth* or *learning rate*, respectively. The *weight modification term* Δw_{ijn} is added to the weight w_{ijn} at weight updating.

If we define

$$\beta_{ijn}^k(t) \equiv \frac{\partial x_k}{\partial w_{ijn}}(t), \tag{19}$$

we obtain for the weight modification Δw_{ijn} and for the computation of β_{ijn}^k in the discrete case:

$$\Delta w_{ijn} = -\eta \sum_{t=t_0}^{t_1} \sum_{k \in \mathcal{O}} e_k(t) \beta_{ijn}^k(t) \tag{20}$$

$$\begin{aligned}
\beta_{ijn}^k(t_0) &= 0 \\
\beta_{ijn}^k(t) &= \frac{\partial x_k}{\partial w_{ijn}}(t) \\
&= \frac{\partial x_k}{\partial net_k}(t) \frac{\partial}{\partial w_{ijn}} \left(\sum_{l \in \mathcal{N}} \sum_{m=0}^{\min\{t-t_0, \hat{n}\}} w_{klm} x_l(t-m) \right) \\
&= \sigma'(net_k(t)) \cdot \left(\delta_{ik} x_j(t-n) + \sum_{l \in \mathcal{N}} \sum_{m=0}^{\min\{t-t_0, \hat{n}\}} w_{klm} \beta_{ijn}^l(t-m) \right) \\
&\quad \forall \ t \in \{t_0+1, t_0+2, ..., t_1\}
\end{aligned} \tag{21}$$

The adaption of the weights in the continuous case is by analogy:

$$\begin{aligned}
\Delta w_{ijn} &\equiv -\eta \frac{\partial E}{\partial w_{ijn}} \\
&\overset{(13),(14)}{=} -\eta \int_{t_0}^{t_1} \sum_{k \in \mathcal{O}} e_k(t) \frac{\partial x_k}{\partial w_{ijn}}(t) dt \\
&= -\eta \int_{t_0}^{t_1} \sum_{k \in \mathcal{O}} e_k(t) \beta_{ijn}^k(t) dt
\end{aligned} \tag{22}$$

$$\beta_{ijn}^k(t) \equiv \frac{\partial x_k}{\partial w_{ijn}}(t) \tag{23}$$

The propagation rule for continuous networks is also derived by analogy:

$$\begin{aligned}
\beta_{ijn}^k(t_0) &= 0 \\
\frac{d\beta_{ijn}^k}{dt}(t) &= \frac{d}{dt} \frac{\partial x_k}{\partial w_{ijn}}(t) \\
&\overset{(5)}{=} \frac{\partial}{\partial w_{ijn}} \frac{1}{T_k} \left(-x_k(t) + \sigma\left(net_k(t)\right) + I_k(t) \right) \\
&= \frac{1}{T_k} \left(-\beta_{ijn}^k(t) + \sigma'\left(net_k(t)\right) \left[\delta_{ik} x_j(t-n) + \sum_{l \in \mathcal{N}} \sum_{\substack{m \in \mathcal{S} \\ m \le t-t_0}} w_{klm} \beta_{ijn}^l(t-m) \right] \right) \\
&\quad \forall \ t \in [t_0, t_1]
\end{aligned} \tag{24}$$

This leads to the following discretized (cf. (6)) version:

$$\Delta w_{ijn} = -\eta \sum_{t=t_0}^{t_1} \sum_{k \in \mathcal{O}} e_k(t) \beta_{ijn}^k(t) \Delta t \tag{25}$$

$$\beta_{ijn}^k(t_0) = 0$$

$$\beta_{ijn}^k(t+\Delta t) = \left(1 - \frac{\Delta t}{T_k}\right) \beta_{ijn}^k(t) + \frac{\Delta t}{T_k} \sigma'(net_k(t)) \left(\delta_{ik} x_j(t-n) + \sum_{l \in \mathcal{N}} \sum_{\substack{m \in \mathcal{S} \\ m \leq t - t_0}} w_{klm} \beta_{ijn}^l(t-m) \right)$$
$$\forall \ t \in \{t_0, t_0 + \Delta t, ..., t_1 - 2\Delta t, t_1 - \Delta t\} \tag{26}$$

In the continuous activation function, the time constants (T_i) are also adaptive. Using

$$\begin{aligned} \Delta T_i &= -\eta \frac{\partial E}{\partial T_i} \\ &\overset{(13),(14)}{=} -\eta \int_{t_0}^{t_1} \sum_{k \in \mathcal{O}} e_k(t) \cdot \frac{\partial x_k}{\partial T_i}(t) dt \\ &= -\eta \int_{t_0}^{t_1} \sum_{k \in \mathcal{O}} e_k(t) \alpha_i^k(t) dt \end{aligned} \tag{27}$$

$$\alpha_i^k(t) = \frac{\partial x_k}{\partial T_i}(t) \tag{28}$$

we obtain the following differential propagation equation:

$$\alpha_i^k(t_0) = 0$$

$$\begin{aligned} \frac{d\alpha_i^k}{dt}(t) &= \frac{\partial}{\partial T_i} \frac{1}{T_k} \left(-x_k(t) + \sigma(net_k(t)) + I_k(t)\right) \\ &= -\frac{\delta_{ik}}{T_k^2} \left(-x_k(t) + \sigma(net_k(t)) + I_k(t)\right) \\ &\quad + \frac{1}{T_k} \left(-\alpha_i^k(t) + \sigma'(net_k(t)) \sum_{l \in \mathcal{N}} \sum_{\substack{m \in \mathcal{S} \\ m \leq t - t_0}} w_{klm} \alpha_i^l(t-m) \right) \end{aligned} \tag{29}$$

and thus the discretized propagation rule:

$$\Delta T_i = -\eta \sum_{t=t_0}^{t_1} e_k(t)\alpha_i^k(t)\Delta t \tag{30}$$

$$\alpha_i^k(t_0) = 0$$

$$\alpha_i^k(t+\Delta t) = \left(1-\frac{\Delta t}{T_k}\right)\alpha_i^k(t) + \frac{\Delta t}{T_k}\sigma'(net_k(t))\left(\sum_{l\in\mathcal{N}}\sum_{\substack{m\in S\\ m\le t-t_0}} w_{klm}\alpha_i^l(t-m)\right)$$

$$-\frac{\delta_{ik}}{T_k^2}\left(-x_k(t)+\sigma(net_k(t))+I_k(t)\right) \tag{31}$$

In both cases, the simultaneous forward propagation of the activations together with the gradient (β_{ijn}^k) at the time t_1 allows an adaption of the weights according to the error gradient without the need either for storing old activations (beyond the time window $\hat{n}$) or for backpropagation of an error signal backwards in time.

It is still to be shown that (β_{ijn}^k) can indeed be computed.

3.3 Computability of the Gradients (β_{ijn}^k)

We will show only the computability of the most complicated gradients (β_{ijn}^k). The computability of the gradients (α_{ijn}^k) can be shown by analogy.

In the discrete case, the computation of a certain β_{ijn}^k at time t requires certain other β_{ijn}^k from the same time step. (cf. equation (21) in contrast to equation (26)). Therefore, the computability of the gradient (β_{ijn}^k) is to be proved (only in the discrete case):

Since we know the initial values of (β_{ijn}^k), it is sufficient to show by induction over the time t that, with the knowledge of all (β_{ijn}^k) until the time $t-1$ (induction assumption), all $(\beta_{ijn}^k)(t)$ can be computed. It is in particular sufficient to know the (β_{ijn}^k) in the period of time $[t-\min\{t-t_0,\hat{n}\},t]$ so that the required storage capacity is does not depend on the length of the period $[t_0,t_1]$ – it depends only on the set of delays:

$$[\max\{t_0,t-\hat{n}\},t] = [t-\min\{t-t_0,\hat{n}\},t] \subseteq \underbrace{[t-\hat{n},t]}_{\text{time window}} \tag{32}$$

In (21), it is obvious that the critical case of computation only occurs for non-delayed links since all $(\beta_{ijn}^k)(t')$ with $t'<t$ have already been computed according to the induction assumption.
For simplification, we assume that the nodes in $\mathcal{V}$ have already been numbered in the same way as in $(**)$. This means that $\forall k,l\in\mathcal{N}$ with $k\le l$ holds: $\langle v_k,v_l\rangle_0\notin\mathcal{E}$. Due to the

above, (21) can be reduced to:

$$\beta^k_{ijn}(t) = \tag{33}$$

$$\sigma'(net_k(t)) \cdot \left[\delta_{ik} x_j(t-n) + \underbrace{\sum_{l<k} w_{kl0} \beta^l_{ijn}(t)}_{(***)} + \sum_{l \in \mathcal{N}} \sum_{m=1}^{\min\{t-t_0, \hat{n}\}} w_{klm} \beta^l_{ijn}(t-m) \right]$$

The critical term $(***)$ shows that only $\beta^l_{ijn}(t)$ with $l < k$ is required for computing $\beta^k_{ijn}(t)$. Therefore, the computation sequence at each time point t is evident:

$$(\beta^1_{ijn}(t)), (\beta^2_{ijn}(t)), ..., (\beta^N_{ijn}(t)) \tag{34}$$

□

3.4 The Pattern Set

So far we derived an adaption algorithm for approximating one pattern (sequence), specified by the input values $(I_i(t))$ and the output errors $(\zeta_i(t))$, which represent the desired mapping. This might be sufficient if, for instance, we use a continouos network for learning to reproduce one single trajectory (c.f. chapter 9.1), but usually a connectionist network is trained with more than one pattern, or pattern sequence in the case of recurrent networks, respectively. E.g. this is the case at classification tasks, where several input patterns are to be classified (c.f. chapter 9.2).

If we define

$$\mathcal{P} \equiv \{ < (I^p_i(t))_{i \in \mathcal{N}, t \in \{t_0, ..., t_1\}}, (t^p_k(t))_{k \in \mathcal{O}, t \in \{t_0, ..., t_1\}} \mid p = 1, 2, ..., P \} \tag{35}$$

as the set of patterns we simply have to extend the energy functions (12) and (13) by the sum

$$\sum_{p \in \mathcal{P}} .$$

Since this would cause the notation becoming much more confusing, although the mathematics would not change fundamentaly, we dropped this sum in the corresponding equations, namely (12), (13), (15), (16), (18), (20), (22), (25), (27), (30), and in the sequel, and left this extension to the reader. Note, that the algorithm in the appendix does sum up over all patterns.

4 Online Learning

Some applications show considerable reductions of learning times if the adaptive parameters – weights (w_{ijn}) and time constants (T_i) – are not only modified after the stop time t_1 but after each individual time step in $[t_0, t_1]$. This method is called *online learning*.

The error function E can be decomposed into disjunct terms:

$$E = \sum_{t=t_0}^{t_1} E_t \tag{36}$$

where E_t is the error restricted to time t:

$$E_t \equiv \sum_{k \in \mathcal{O}} \zeta_k(t) \tag{37}$$

For the continuous case, the following holds by analogy:

$$E = \int_{t_0}^{t_1} E_t dt \tag{38}$$

$$E_t \equiv \sum_{k \in \mathcal{O}} \zeta_k(t) \tag{39}$$

and therefore for the discrete version of (38):

$$E = \sum_{t=t_0}^{t_1} E_t \cdot \Delta t \tag{40}$$

At any time t, the weights are adapted with respect to the error function E_t:

- In the discrete case:

$$\begin{aligned} \Delta w_{ijn}(t) &= -\eta \frac{\partial E_t}{\partial w_{ijn}} \\ &= -\eta \sum_{k \in \mathcal{O}} e_k(t) \beta_{ijn}^k(t) \end{aligned} \tag{41}$$

- In the continuous case:

$$\begin{aligned} \frac{dw_{ijn}}{dt}(t) &= -\eta \frac{\partial E_t}{\partial w_{ijn}} \\ &= -\eta \sum_{k \in \mathcal{O}} e_k(t) \beta_{ijn}^k(t) \end{aligned} \tag{42}$$

$$\begin{aligned} \frac{dT_i}{dt}(t) &= -\eta \frac{\partial E_t}{\partial T_i} \\ &= -\eta \sum_{k \in \mathcal{O}} e_k(t) \alpha_i^k(t) \end{aligned} \tag{43}$$

- and therefore in the discrete version of the continuous case:

$$\Delta w_{ijn}(t) = -\eta \cdot \Delta t \sum_{k \in \mathcal{O}} e_k(t) \beta_{ijn}^k(t) \tag{44}$$

$$\Delta T_i(t) = -\eta \cdot \Delta t \sum_{k \in \mathcal{O}} e_k(t) \alpha_i^k(t) \tag{45}$$

The gradients (β_{ijn}^k) and (α_i^k) are computed as described above in the equations (21), (26) and (31).

Anyway, it is important to mention that online learning is not an exact gradient descent procedure, but a heuristic which often increases the convergence speed.

5 Teacher Forcing

In combination with *online learning* often the technique of *teacher forcing* is used [26, 48, 28, 13].

Imagine a recurrent network working through the time. One may argue that a desired subsequent state cannot be correctly computed if the current state already shows an error, since, in this case, even correctly adapted parameters will produce a wrong state. This happens because the last state influences the present state by the recurrent links.

Teacher forcing means to set the activations of the output units *after each online adaption* equal to a value which generates an error of 0.
For the usual error function (15) or (16) with a given trajectory $(t_k(t))$, it follows that, *after* each weight update, each output activation $x_k(t)$ is set equal to $t_k(t)$ for all $k \in \mathcal{O}$:

$$x_k(t) \longleftarrow t_k(t) \tag{46}$$

$$\text{or} \quad x_k(t) \longleftarrow x_k(t) - e_k(t) \tag{47}$$

The latter is only applicable if the error according to equation (15) or (16) is defined as a squared deviation of a given trajectory.

6 Special Cases of the Gradient Descent Algorithm

6.1 Non-recurrent Multilayer Networks

We concentrate on the "classical" backpropagation networks introduced in [36]. We obtain them by postulating:

- $\hat{n} = 0$ – The network works without time context,
- for simplification: $w_{ij} = w_{ij0}$,
- for simplification, too, let the accounts be numbered according to $(**)$, i.e. $\forall i, j \in \mathcal{N}$ with $i \leq j$: $\langle v_i, v_j \rangle_0 \notin \mathcal{E}$,
- due to the missing time context sensitivity, the time index (t) is dropped and
- no output node v_k (with $k \in \mathcal{O}$) has a successor[1]: $\forall l \in \mathcal{N} : w_{lk} = 0$.

For the weight gradients, we obtain according to (20) and (21)[2]:

$$\begin{aligned}
\Delta w_{ij} &= -\eta \sum_{k \in \mathcal{O}} e_k \beta_{ij}^k \\
&= -\eta \sum_{k \in \mathcal{O}} e_k \sigma'(net_k) \left[\delta_{ik} x_j + \sum_{i \leq l < k} w_{kl} \beta_{ij}^l \right] \\
&= -\eta \left[\mathbb{1}_{\mathcal{O}}(i) \cdot e_i \sigma'(net_i) x_j + \mathbb{1}_{\mathcal{N} \setminus \mathcal{O}}(i) \sum_{k \in \mathcal{O}} \sum_{i \leq l < k} e_k \sigma'(net_k) w_{kl} \beta_{ij}^l \right] \qquad (48)
\end{aligned}$$

Concerning the last equation, it is to be noted that in the case of $i \in \mathcal{O}$ for all $l \neq i$ the following holds: $\beta_{ij}^l = 0$, since the weight w_{ij} has effect only on the activation x_i which, however, cannot influence other activations since leaving links are missing. For the case $i = l$, it holds that $w_{kl} = w_{ki} = 0$.

For the equivalence proof, we consider

$$\begin{aligned}
\delta_i &= \frac{\partial E}{\partial net_i} \quad \forall i \in \mathcal{N} \text{ and} \\
\Delta w_{ij} &= -\delta_i x_j \quad \forall \langle v_i, v_j \rangle \in \mathcal{E} \qquad (49)
\end{aligned}$$

taken from [36][3]. We distinguish two cases:

[1] Of course, this restriction is not required, but it is used both in [36]. and the major part of published literature.

[2] Here $\mathbb{1}$ identifies the *membership function*:

$$\mathbb{1}_A(x) = \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{otherwise} \end{cases}$$

[3] This δ has nothing to do with the Kronecker delta.

1. $i \in \mathcal{O}$ is output node:

$$\begin{aligned}
\overset{(48)}{\Longrightarrow} \Delta w_{ij} &= -\eta \sum_{k \in \mathcal{O}} e_k \beta_{ij}^k \\
&= -\eta e_i \sigma'(net_i) x_j \\
&= -\eta \frac{\partial E}{\partial x_i} \frac{\partial x_i}{\partial net_i} x_j \\
&= \delta_i x_j \qquad (50)
\end{aligned}$$

2. $i \notin \mathcal{O}$ is hidden or input node:

$$\overset{(48)}{\Longrightarrow} -\eta \sum_{k \in \mathcal{O}} e_k \beta_{ij}^k = -\eta \sum_{k \in \mathcal{O}} \sum_{i \leq l < k} e_k \sigma'(net_k) w_{kl} \beta_{ij}^l \qquad (51)$$

It remains to show that both sides of (51) are equivalent, equivalent to $-\eta \delta_i x_j$:

$$\begin{aligned}
-\eta \sum_{k \in \mathcal{O}} e_k \beta_{ij}^k &= -\eta \sum_{k \in \mathcal{O}} \frac{\partial E}{\partial x_k} \frac{\partial x_k}{\partial w_{ij}} \\
&= -\eta \left(\sum_{k \in \mathcal{O}} \frac{\partial E}{\partial x_k} \frac{\partial x_k}{\partial net_i} \right) \frac{\partial net_i}{\partial w_{ij}} \\
&= -\eta \delta_i x_j \qquad (52)
\end{aligned}$$

$$\begin{aligned}
-\eta \sum_{k \in \mathcal{O}} \sum_{i \leq l < k} e_k \sigma'(net_k) w_{kl} \beta_{ij}^l &= -\eta \sum_{k \in \mathcal{O}} \frac{\partial E}{\partial x_k} \frac{\partial x_k}{\partial net_k} \sum_{i \leq l < k} \frac{\partial net_k}{\partial x_l} \frac{\partial x_l}{\partial w_{ij}} \\
&= -\eta \sum_{k \in \mathcal{O}} \frac{\partial E}{\partial net_k} \frac{\partial net_k}{\partial w_{ij}} \\
&= -\eta \frac{\partial net_i}{\partial w_{ij}} \frac{\partial E}{\partial net_i} \\
&= -\eta \delta_i x_j \qquad (53)
\end{aligned}$$

□

6.2 Fully Connected Recurrent Connectionist Networks

A feed-forward algorithm for fully connected recurrent connectionist networks [48, 49, 32] can be derived from this algorithm if we assume the following restrictions:

- $\hat{n} = 1$,

- $\forall i,j \in \mathcal{N} : \langle v_i, v_j \rangle_0 \notin \mathcal{E}$ and
- for simplification: $w_{ij} = w_{ij1}$ and $\beta_{ij}^k = \beta_{ij1}^k$.

Thus we obtain from (21):

$$\begin{aligned} \beta_{ij}^k(t_0) &= 0 \\ \beta_{ij}^k(t) &= \sigma'(net_k(t)) \cdot \left[\delta_{ik} x_j(t-1) + \sum_{l \in \mathcal{N}} w_{kl} \beta_{ij}^l(t-1) \right] \qquad (54) \\ & \forall \ t \in \{t_0+1, t_0+2, ..., t_1\} \end{aligned}$$

This is equivalent to the algorithm presented in [48].

6.3 Further Special Cases

Several special cases used in literature can easily be derived from our general approach. Let us mention the following:

- In [28], the author experiments with analog networks which represent special cases of our continuous networks. His activation function

$$T_i \frac{dx_i}{dt}(t) = -x_i(t) + \sigma(net_i(t)) + I_i(t) \qquad (55)$$

  is obtained by defining the link delay set $\mathcal{S} = \{0\}$.

- Recurrent networks introduced in [19, 14, 26] are easy to generate by setting the time constant $\hat{n}$ equal to 1 (hence $\mathcal{S} = \{0, 1\}$) and by dropping the links which are not required.

- Time-delay networks [46] can be derived by full analogy to the above non-recurrent networks due to their missing recurrencies.

7 Adaption of External Quantities by Means of Gradient Descent

7.1 Adaption of Initial Activations

Upon the adaption of the initial activations ($\tilde{x}_i$), they are modified step by step to minimize the error. This avoids unfortunate initial configurations which do not allow a good reduction of the error.

Let the modification of an initial activation $\forall i \in \mathcal{N}$ be defined as follows:

$$\begin{aligned}
\Delta\tilde{x}_i &= -\eta\frac{\partial E}{\partial\tilde{x}_i} \\
&= -\eta\frac{\partial}{\partial\tilde{x}_i}\sum_{t=t_0}^{t_1}\sum_{k\in\mathcal{O}}\zeta_k(t) \\
&= -\eta\sum_{t=t_0}^{t_1}\sum_{k\in\mathcal{O}}e_k(t)\frac{\partial x_k(t)}{\partial\tilde{x}_i} \\
&= -\eta\sum_{t=t_0}^{t_1}\sum_{k\in\mathcal{O}}e_k(t)\gamma_i^k(t), \qquad (56)
\end{aligned}$$

If we define

$$\gamma_i^k(t) \equiv \frac{\partial x_k(t)}{\partial\tilde{x}_i} = \frac{\partial x_k(t)}{\partial x_i(t_0)}, \qquad (57)$$

the following holds:

$$\begin{aligned}
\gamma_i^k(t_0) &= \delta_{ik} = \begin{cases} 1, \text{ if } i = k \\ 0, \text{ otherwise} \end{cases} \\
\gamma_i^k(t) &= \frac{\partial x_k(t)}{\partial\tilde{x}_i} \\
&= \frac{\partial}{\partial\tilde{x}_i}\sigma\left(\sum_{l\in\mathcal{N}}\sum_{m=0}^{\min\{t-t_0,\hat{n}\}}w_{klm}x_l(t-m)\right) \\
&= \sigma'(net_k(t))\left(\sum_{l\in\mathcal{N}}\sum_{m=0}^{\min\{t-t_0,\hat{n}\}}w_{klm}\gamma_i^l(t-m)\right) \qquad (58) \\
&\quad \forall\ t \in \{t_0+1, t_0+2, ..., t_1\}
\end{aligned}$$

By analogy to (β_{ijn}^k), the (γ_i^k) are propagated forward in time. Of course, online adaption or teacher forcing is not possible in this case since the initial activations are used only at the beginning of an activation propagation phase, at t_0.

In the case of continuous activation functions, the initial activations are adapted analogously so that the mathematical derivation can be left to the reader.

$$\begin{aligned}
\Delta\tilde{x}_i &= -\eta\Delta t \sum_{t=t_0}^{t_1} \sum_{k\in\mathcal{O}} e_k(t)\gamma_i^k(t) \\
\gamma_i^k(t_0) &= \delta_{ik} = \begin{cases} 1, \text{ if } i=k \\ 0, \text{ otherwise} \end{cases} \\
\frac{\gamma_i^k}{dt}(t) &= \frac{1}{T_k}\left(-\gamma_i^k(t) + \sigma'(net_k(t))\left[\sum_{l\in\mathcal{N}} \sum_{\substack{m\in\mathcal{S} \\ m\le t-t_0}} w_{klm}\gamma_i^l(t-m)\right]\right) \qquad (59) \\
\gamma_i^k(t+\Delta t) &= \left(1-\frac{\Delta t}{T_k}\right)\gamma_i^k(t) + \frac{\Delta t}{T_k}\sigma'(net_k(t))\left(\sum_{l\in\mathcal{N}} \sum_{\substack{m\in\mathcal{S} \\ m\le t-t_0}} w_{klm}\gamma_i^l(t-m)\right) \\
&\qquad \forall t \in \{t_0, t_0+\Delta t, ..., t_1-2\Delta t, t_1-\Delta t\} \qquad (60)
\end{aligned}$$

7.2 "Inversion": Adaption of the External Input

"Inversion" [24, 22, 42] turned out to be a useful tool for finding spurious input attractors of a trained connectionist network. Inversion is the gradient search of an external input $(I_i(t))$, which is mapped by the network to a specific given target trajectory $(\hat{t}_i(t))$ $(\forall i \in \mathcal{O}, t \in [t_0, t_1])$ except a small remainder.

Let $\mathcal{I} \subseteq \mathcal{N}$ be the set of indices of the input nodes – that are those nodes which are expected to have a real input $(I_i(t) \neq 0)$ at some time t. In the case of an *approximate inversion* of the connectionist network, we try to approximate with this external input $(I_i(t))$ the given output $(\hat{t}_i(t))$ (e.g. in general: partial continuous output conditions) as well as possible[4]. For this reason we define a new error function, the least mean square error with respect to this given target $(\hat{t}_i(t))$:

$$E_{inv} = \sum_{t=t_0}^{t_1} \sum_{k\in\mathcal{O}} s_k(t)\tfrac{1}{2}(\hat{t}_k(t) - x_k(t))^2 \qquad (61)$$

$$\text{resp.} \quad E_{inv} = \int_{t_0}^{t_1} \sum_{k\in\mathcal{O}} s_k(t)\tfrac{1}{2}(\hat{t}_k(t) - x_k(t))^2 dt \qquad (62)$$

We start with a suitable[5] initial input $(I_i(t))$ and modify it step by step by changing each particular input value $I_i(t)$ in the opposite direction of the (new) error gradient with respect to this input $I_i(t)$, i.e. $\forall s \in \{t_0, t_0+1, ..., t_1\}$:

$$\Delta I_i(s) = -\eta\frac{\partial E_{inv}}{\partial I_i(s)}$$

[4] It is always possible to approximate a specific *input-target*, a direct target vector for the external input $(I_i(t))$ simultaneously, see also [24, 42] and the simulation results.

[5] This term is not clear, but one can use a random vector as well.

$$= -\eta \sum_{t=s}^{t_1} \sum_{k \in \mathcal{O}} e_k(t) \frac{\partial x_k(t)}{\partial I_i(s)} \tag{63}$$

Define

$$\begin{aligned} \xi_{is}^k(t) &\equiv \frac{\partial x_k(t)}{\partial I_i(s)} \\ &= \sigma'(net_k(t)) \left(\sum_{l \in \mathcal{N}} \sum_{m=0}^{\min\{t-s, \hat{n}\}} w_{klm} \frac{\partial x_l(t-m)}{\partial I_i(s)} \right) + \underbrace{\frac{\partial I_k(t)}{\partial I_i(s)}}_{=\delta_{ik} \cdot \delta_{st}}, \end{aligned} \tag{64}$$

then the following holds for ξ_{is}^k:

$$\begin{aligned} \xi_{is}^k(s) &= \sigma'(net_k(s)) \left(\sum_{l \in \mathcal{N}} w_{kl0} \xi_{is}^l(s) \right) + \delta_{ik} \\ &\overset{\text{if } 0 \notin \mathcal{S}}{=} \delta_{ik} = \begin{cases} 1, \text{ if } i = k \\ 0, \text{ otherwise} \end{cases} \\ \xi_{is}^k(t) &= \frac{\partial x_k(t)}{\partial I_i(s)} \\ &= \sigma'(net_k(t)) \left(\sum_{l \in \mathcal{N}} \sum_{m=0}^{\min\{t-s, \hat{n}\}} w_{klm} \xi_{is}^l(t-m) \right) \\ &\quad \forall \ t \in \{s+1, s+2, ..., t_1\} \end{aligned} \tag{65}$$

These inversion values can also be computed during the forward propagation of the activations.

Since in this gradient descent procedure the input activations are not bounded any more, it is appropriate to bound them after applying the updating rule (63) by hand (c.f. chapter 2.4):

$$I_i(s) \leftarrow \max\{I_i(s), \overbrace{\min\{-a, b-a\}}^{\text{lower bound}}\} \tag{66}$$

$$I_i(s) \leftarrow \min\{I_i(s), \underbrace{\max\{-a, b-a\}}_{\text{upper bound}}\} \tag{67}$$

The derivation of the inversion rule for continuous connectionist networks and its discretized version is again left to the reader. We summarize the results here:

$$\Delta I_i(s) = -\eta \Delta t \sum_{t=s}^{t_1} \sum_{k \in \mathcal{O}} e_k(t) \xi_{is}^k(t) \tag{68}$$

$$\xi_{is}^k(s) = \frac{\partial x_k(s)}{\partial I_i(s)} \overset{(6)}{=} 0$$

$$\frac{d\xi_{is}^k}{dt}(t) = \frac{1}{T_k}\left[-\xi_{is}^k(t) + \sigma'(net_k(t))\left(\sum_{l\in\mathcal{N}}\sum_{\substack{m\in\mathcal{S}\\ m\leq t-s}} w_{klm}\xi_{is}^l(t-m)\right) + \delta_{ik}\delta_{st}\right] \quad (69)$$

$$\begin{aligned}\xi_{is}^k(t+\Delta t) &= \left(1-\frac{\Delta t}{T_k}\right)\xi_{is}^k(t) + \frac{\Delta t}{T_k}\sigma'(net_k(t))\left(\sum_{l\in\mathcal{N}}\sum_{\substack{m\in\mathcal{S}\\ m\leq t-s}} w_{klm}\xi_{is}^l(t-m)\right)\\ &+ \frac{\Delta t}{T_k}\delta_{ik}\delta_{st}\end{aligned}$$

$$\forall t \in \{s, s+\Delta t, ..., t_1 - 2\Delta t, t_1 - \Delta t\} \quad (70)$$

8 Complexity

Since there is only very poor knowledge about convergence speed of gradient descent in connectionist networks, we consider only the time complexity of one parameter estimation step. It is a well-known fact that the success of the training process depends often on the randomly selected initial parameters, namely the weights and biases. Therefore it is impossible to give a simple bound for the learning complexity of a connectionist network as it is presented here.

It is interesting to compare time and space complexities of the feed-forward algorithm and the corresponding backpropagation algorithm of the unfolded network. Let a network $\mathcal{G}$ with $\mathcal{N}$ nodes be given, among them let O nodes be output nodes: $O = |\mathcal{O}|$. The link set constitutes a subset of $\{\langle v_i, v_j \rangle_n |\ v_i, v_j \in \mathcal{V} \wedge n \in \mathcal{S}\}$ which satisfies the condition $(*)$. In the worst case, there are $\mathbf{O}(N^2 S)$ links if we define $S = |\mathcal{S}|$ as the number of the delay indices of each link.

Let the number of forward steps be T:

$$T = \begin{cases} t_1 - t_0 & \text{discrete activation function} \\ \frac{t_1 - t_0}{\Delta t} & \text{discrete version of the continuous activation function} \end{cases} \tag{71}$$

Let the set of presented pattern sequences of one training cycle be $\mathcal{P}$ and $P = |\mathcal{P}|$ the number of pattern sequences.

In general, complexity is of interest in two cases:

1. Sequential simulation:
   (a) The connectionist network is simulated on *one sequential processor*. The processor cost is thus $\mathbf{O}(1)$. The main cost of the internal computation loop (over time t, cf. appendix) is undoubtedly the computation of (β^k_{ijn}). All β^k_{ijn} at one time step t can be computed sequentially in $\mathbf{O}(ON^3S^2)$ (cf. equations (20), (21), (25), (26)). This yields an overall amount of time required for a learning cycle of $\mathbf{O}(PTON^3S^2)$. Storage cost is also restricted by the required storage for (β^k_{ijn}), this is $\mathbf{O}(ON^2S)$.
   (b) If we use *online learning* for adapting the internal network parameters, the complexities are the same. This case is more interesting at *unfolded-in-time* networks.
2. The connectionist network is operated in a *maximally parallel way.* Here two aspects are important for bounding the complexity: On the one hand the availability of links without delay (i.e. $0 \in \mathcal{S}$) influences the computation complexity considerably. On the other hand, for *batch learning*, the presentation of the various patterns $p \in \mathcal{P}$ can be parallelized in addition. Therefore, we distinguish three cases:
   (a) $0 \in \mathcal{S}$: In section 3.3, we have shown that the computation of (β^k_{ijn}) is possible if, for a fixed $l \in \mathcal{O}$, the (β^l_{ijn}) are not computed before all $(\beta^{l'}_{ijn})$ have been

	feed-forward			**unfolded in time**		
Complexity:	time	processor	storage	time	processor	storage
serial	$PTON^3S^2$	1	ON^2S	PTN^2S	1	N^2S+TN
serial *online*	$PTON^3S^2$	1	ON^2S	PT^2N^2S	1	N^2ST
parallel $0 \in \mathcal{S}$	$PTONS$ $PTO\log(NS)$	N^2S N^3S^2	ON^2S ON^2S	PTN	NS	N^2S+TN
parallel $0 \notin \mathcal{S}$	$PTNS$ $PT\log(NS)$	ON^2S ON^3S^2	ON^2S ON^2S	PT	N^2S	N^2S+TN
batch learning	$T\log(NS)\log P$	PON^3S^2	ON^2S	$T\log P$	PN^2S	N^2S+TN
hardware impl.	PT	N		PT	N	

Figure 1: Complexity comparison of recurrent connectionist network learning procedures: *feed-forward gradient descent* and *backpropagation with an unfolded-in-time network.*

computed for all $l' \in \mathcal{O}$ with $l' < l$. The same applies to the gradients (α_i^k), (γ_i^k) etc. Therefore the computation cost for the inner loop (we mentioned above: via t, cf. appendix) in $\mathbf{O}(ONS)$ – this yields $\mathbf{O}(PTONS)$ for a learning cycle – using $\mathbf{O}(N^2S)$ processors with sequential processing of the summation loop in the computation of (β_{ijn}^k). If this summation is also parallelized, we obtain a time complexity of $\mathbf{O}(O\log(NS))$ using $\mathbf{O}(N^3S^2)$ processors. Here $\mathbf{O}(\log(NS))$ time is needed to sum up all single summands in the sum of formula (21) or (26), respectively, if the summation is hard-wired. The required capacity is again $\mathbf{O}(N^2S)$.

(b) $0 \notin \mathcal{S}$: Now, all (β_{ijn}^k) can be computed in parallel. Using $\mathbf{O}(ON^2S)$ processors, we can thus reduce a time complexity of $\mathbf{O}(NS)$ of the inner loop (via t) and of $\mathbf{O}(PTNS)$ for a learning cycle.
If we parallelize in addition the computation of the inner sum of (β_{ijn}^k), we obtain a time cost of $\mathbf{O}(PT\log(NS))$ in the case of $\mathbf{O}(ON^3S^2)$ processors. The required space is also $\mathbf{O}(N^2S)$.

(c) *Batch learning*: Using parallel computation of all P pattern sequences, we can also reduce the time complexity of a learning cycle to $\mathbf{O}(\max\{T\log NS, \log P\})$ which requires however $\mathbf{O}(PON^3S^2)$ processors. The $\log P$ time steps are required for taking the mean of the weights.
(It is to be noted that by means of *pipelining* the computation cost of the inner loop (via t) in the case of *batch learning* can also be kept at $0 \in \mathcal{S}$ in $\mathbf{O}(\log N)$ in the case of a sufficient amount of time of $T > 2N$ and if $\mathbf{O}(N^4S)$ processors are used. The storage cost, however, increases to $\mathbf{O}(N^3(S+N))$ since further N past values of (β_{ijn}^k) have to be stored.)

These results are compared with learning methods which require a real back-computation in time in strict extension of the backpropagation algorithm (*unfolding-in-time*, c.f. [36, 42]).

1. Sequential simulation:

   (a) In the case of a subsequent computation using $\mathbf{O}(1)$ processors, the computation cost is $\mathbf{O}(PTN^2S)$, since $\mathbf{O}(N^2S)$ link gradients have to computed. Required

capacity is $\mathbf{O}(N^2S + TN)$ since, on the one hand, all link weights have to be stored ($\mathbf{O}(N^2S)$) and, on the other, all activations of the past have to be stored ($\mathbf{O}(TN)$) in the case of each pattern sequence $p \in \mathcal{P}$.

(b) *Online-learning* is rather expensive at this learning technique. At each time step the whole backpropagation has to take place. Therefore the time complexity grows to $\mathbf{O}(PT^2N^2S)$. Since after adapting the weights (after each time step) the old weight matrix is still used for backpropagating the error gradient through the time, the storage complexity grows to $\mathbf{O}(N^2ST+TN) = \mathbf{O}(N^2ST)$.

2. In the maximally parallel computation, the above three cases have to be distinguished again:

   (a) $0 \in \mathcal{S}$: This means that each node activation and gradient of the *unfolded in time* network has to be computed separately. Time complexity is therefore $\mathbf{O}(TPN)$ if $\mathbf{O}(NS)$ processors are used. Space complexity is $\mathbf{O}(N^2S + TN)$.

   (b) $0 \notin \mathcal{S}$: Now, we can compute all node activations and gradients of a time layer in parallel which results in a time complexity of $\mathbf{O}(PT)$ in the case of a processor complexity of $\mathbf{O}(N^2S)$ processors. Space complexity continues to be equal to $\mathbf{O}(N^2S + TN)$.

   (c) *Batch learning*: Even for *unfolded in time* networks, it is possible, as in our *feed-forward* algorithm, to parallelize the use of pattern sequences so that we can finally obtain a time complexity of $\mathbf{O}(T \log P)$ at a processor cost of $\mathbf{O}(PN^2S)$.

The result is summarized in table 1. Though the feed-forward algorithm described in this paper needs $\mathbf{O}(ONS)$ more resources – of time or processors – the storage capacity does not depend on T. Therefore one should prefer this algorithm if the time T is large and the storage capacity restricted. Moreover, online-learning in time (e.g. at endless target trajectories) is much more efficient with this feed-forward algorithm.

All these complexities are derived for common binary processors. Those processors have a bounded degree and are not able to add only a fixed number of summands at one time step. Due to a *connectionist hardware*, it is interesting to derive complexities for a future analogous hardware-implementation of this algorithm. Then the time complexity will be about $\mathbf{O}(PT)$ with $\mathbf{O}(N)$ processors. The same holds for the unfolded network, which is to be trained with backpropagation. Since these complexities are much smaller (c.f. Fig. 1), the expense is the huge amount of links, which are to be implemented.

9 Simulation Results

9.1 Continuous Trajectory Learning

Following the examples from [28] we used a two-node continuous network with a sampling rate $\tau = 0.01$ for learning a circular trajectory. It is known that a fully recurrent network of at least two units is able to oscillate – and the simplest oscillation is a circular oscillation. With a stepwidht $\eta_w = 0.8$ for the weights w_{ij0} and $\eta_T = 0.7$ for the time constants T_i and an error weighting function $s_k(t) = \lambda^t$ with $\lambda = 0.6$ (c.f. equation (16)) we teached the network to generate a circular trajectory (by online-learning and teacher forcing) with the target equations:

$$\begin{aligned} t_0(t) &= \sin(2t) * 0.8 \\ t_1(t) &= \cos(2t) * 0.8 \end{aligned} \tag{72}$$

Since the squashing function paramters were set to the values in (10) and (11), namely $a = 1$, $b = 2$ and $c = 1$ and since no external input was given, the activation interval was $]-1, 1[$. In order to avoid the extremal activation ranges we bounded the target values by $[-0.8, 0.8]$.

The learning process is plotted in Fig. 2. Each 10 time steps the osclillator performed by the instantaneous network configuration is plotted.

Next we trained a seven-node network to generate an "eight"-trajectory. The learning parameters were: $\eta_w = 1.0$, $\eta_T = 0.7$ and $\lambda = 0.6$. The training results are plotted in Fig. 3.

9.2 Pattern Classifying and Inversion with a Discrete-in-Time Network

We trained a special recurrent network [14] as shown in Fig. 4 to the task of recognizing handwritten digits (cf. Fig. 5). After successful training we tested the generalization behavior. The decision criterion was hard: only if the output error E was less then 0.2, the pattern was classified correctly. With this classification bound, we obtained a generalization rate of about 82%.

For finding spurious attractors, approximative inversion was applied. Some results are shown in Fig. 6. As you can see there, it is possible to approximate a specific so-called *input target* [42] and the desired output target simultaneously by using an extended energy function (Fig. 7). Moreover, it is possible to find binary spurious input attractors, which are also classified as a digit (Fig. 8). Lastly, we found a single 11-pixel input vector, which was classified as a digit, although the usual digits consisted of eight (intead of one) 11-pixel input vectors (Fig. 9).

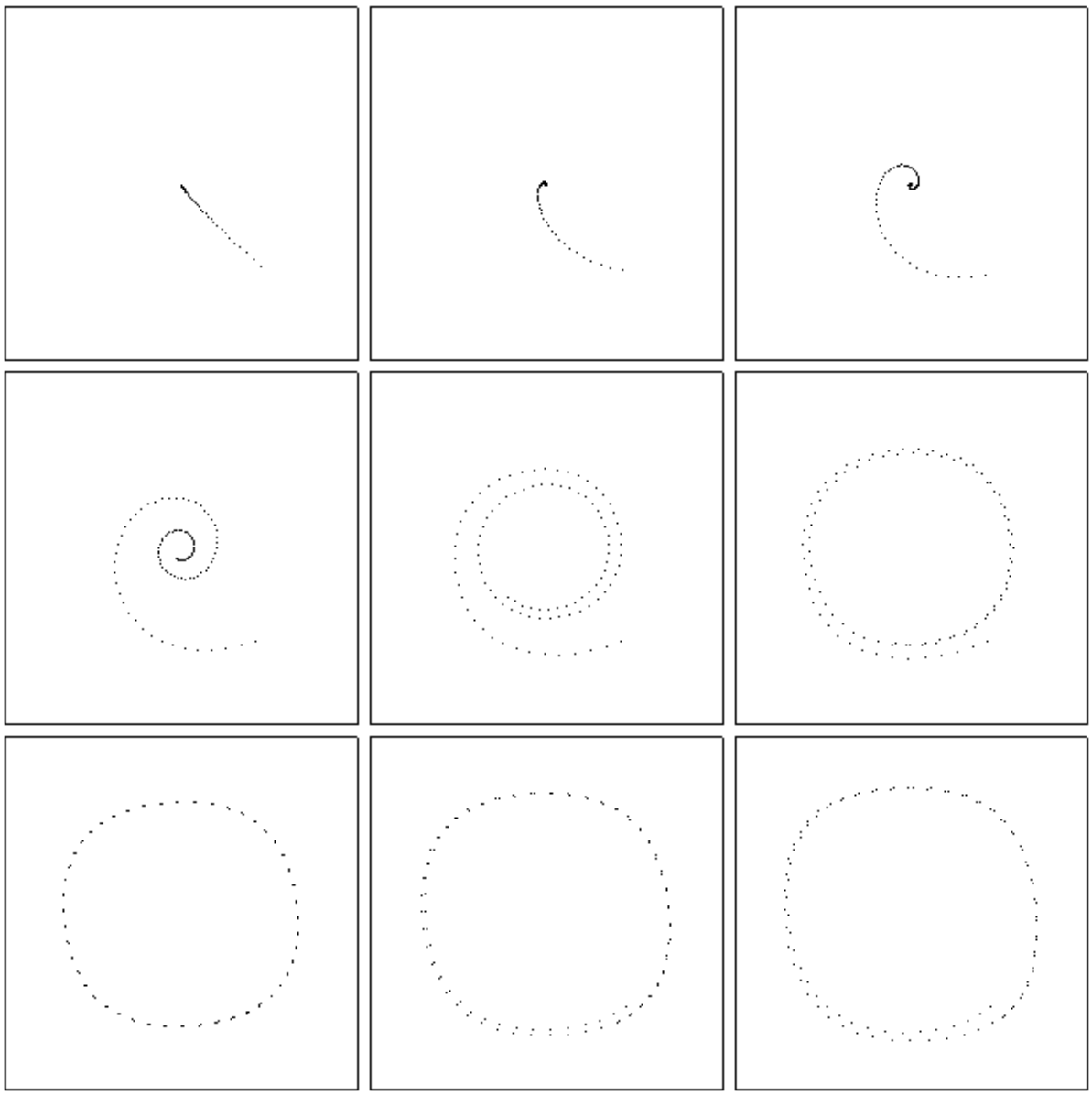

Figure 2: Learning of a circular trajecory. The activation of the first unit is plotted in horizontal, the activation ot the second unit in vertical direction. After each ten time steps a "snapshot" of the current oscillator is shown.

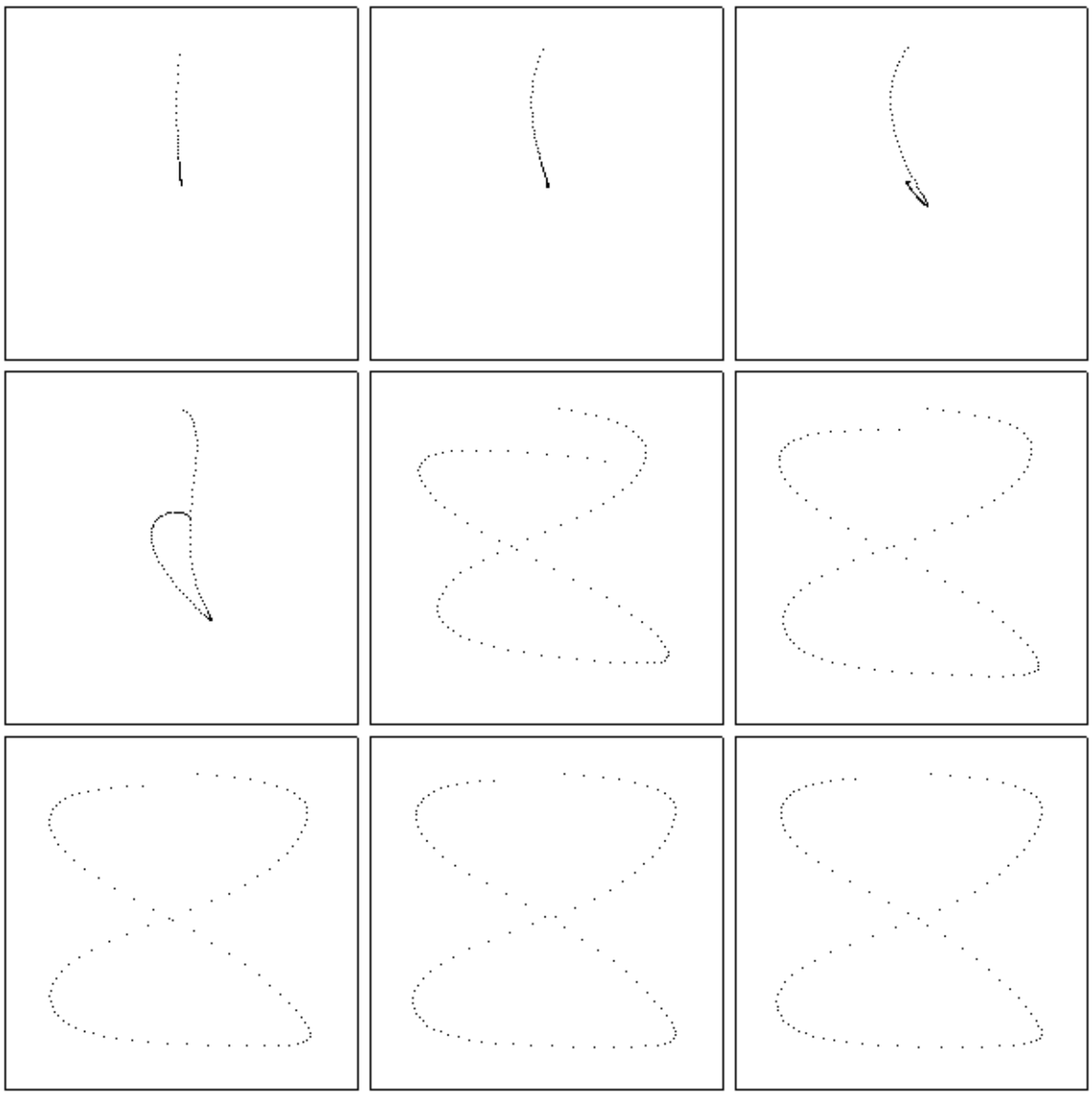

Figure 3: Learning of a trajectory "eight". Since learning was more complicated and thus slower than in the previous example, only each twentieth time step the network output was plotted.

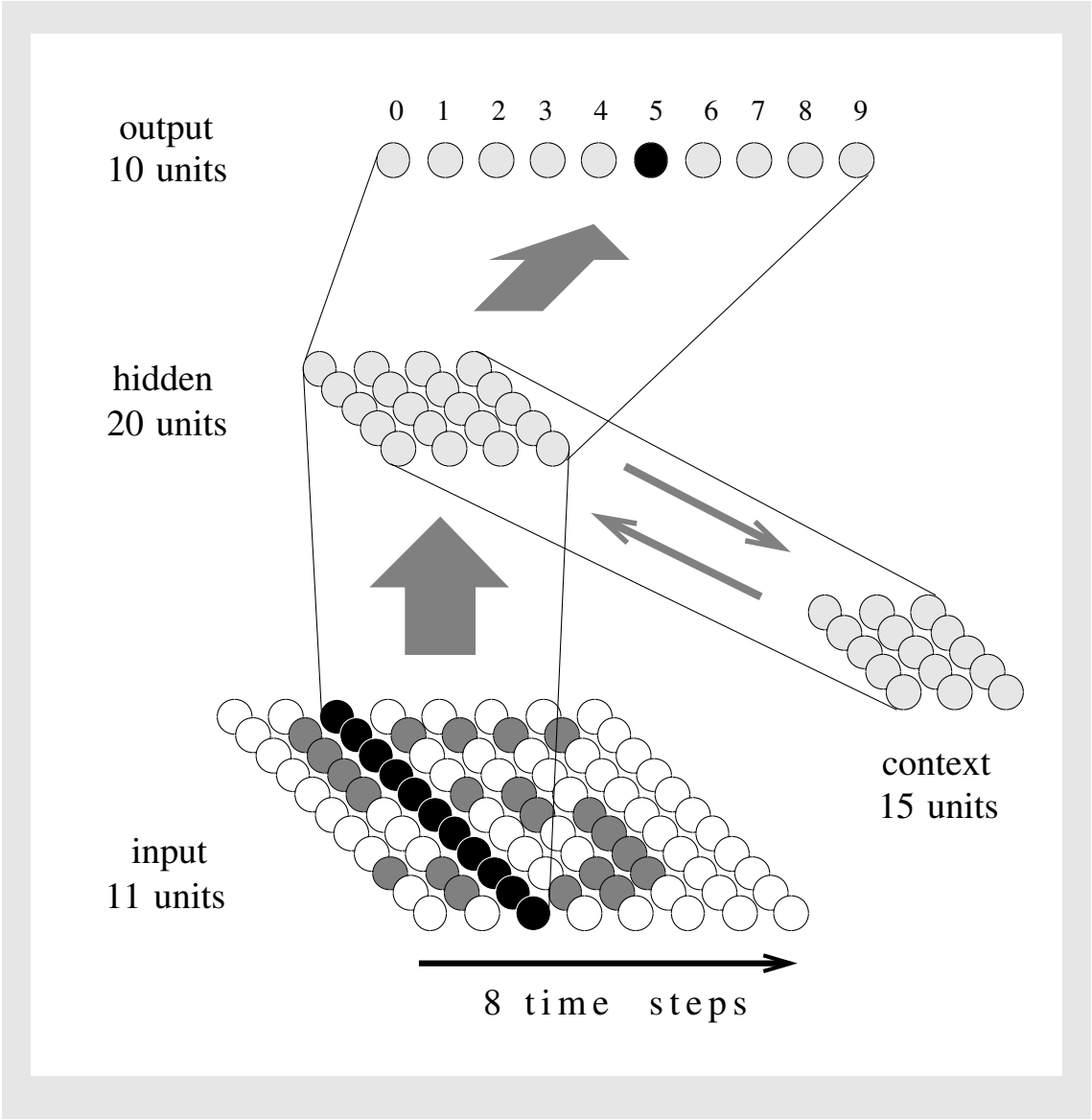

Figure 4: A recurrent connectionist network for digit recognition. The input window contains 11 input units which scan the 8x11 input matrix in 8 time steps. Each of the input units is connected with each of the 20 hidden units. These hidden units are fully connected to the 10 output and the 15 context units. In opposition to all other links, the recurrent nodes from the hidden layer to the context layer had a delay of 1. The context layer itself is again fully connected to the hidden layer.

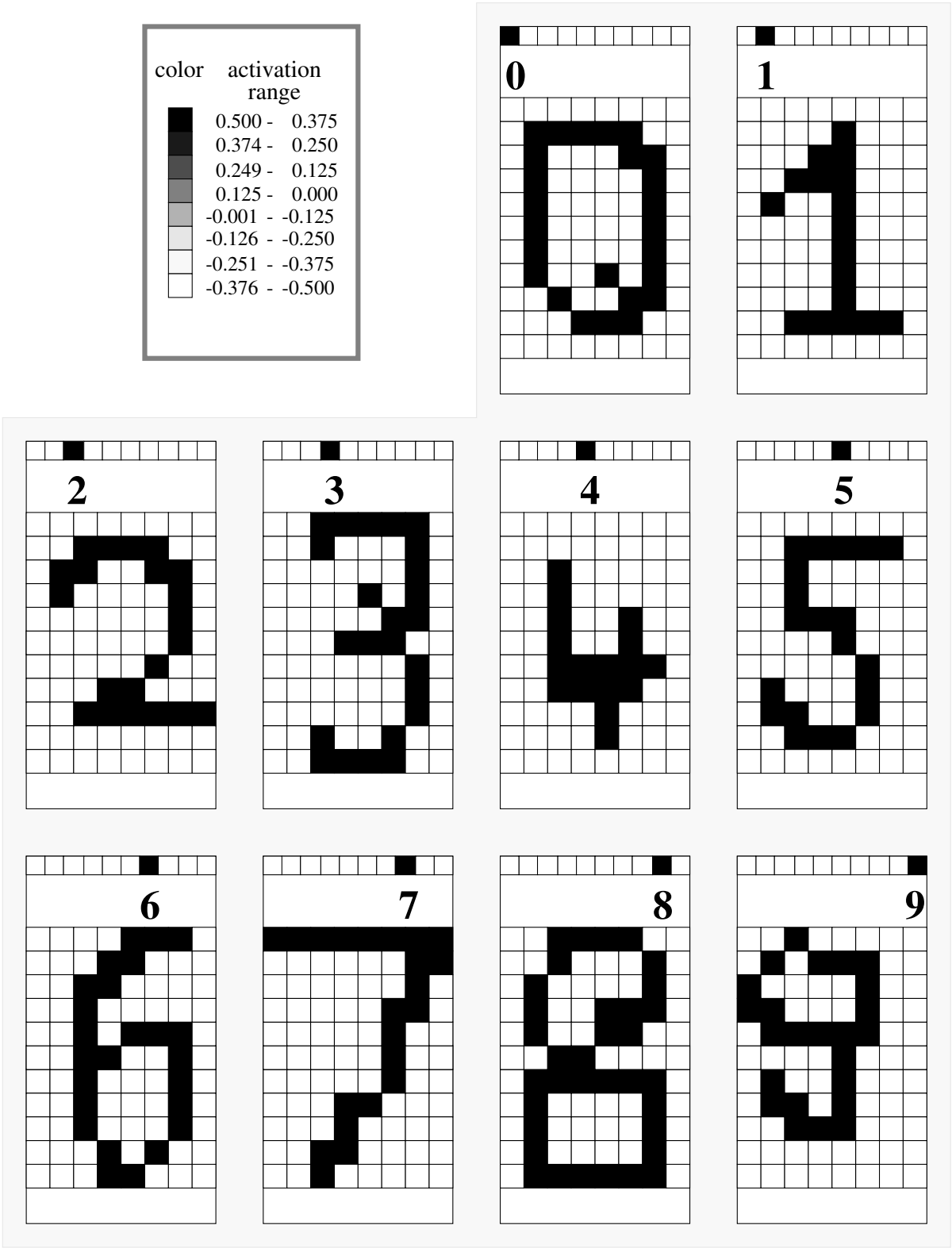

Figure 5: Example patterns from the training set. This training set consists of 490 training patterns – the test set of 140.

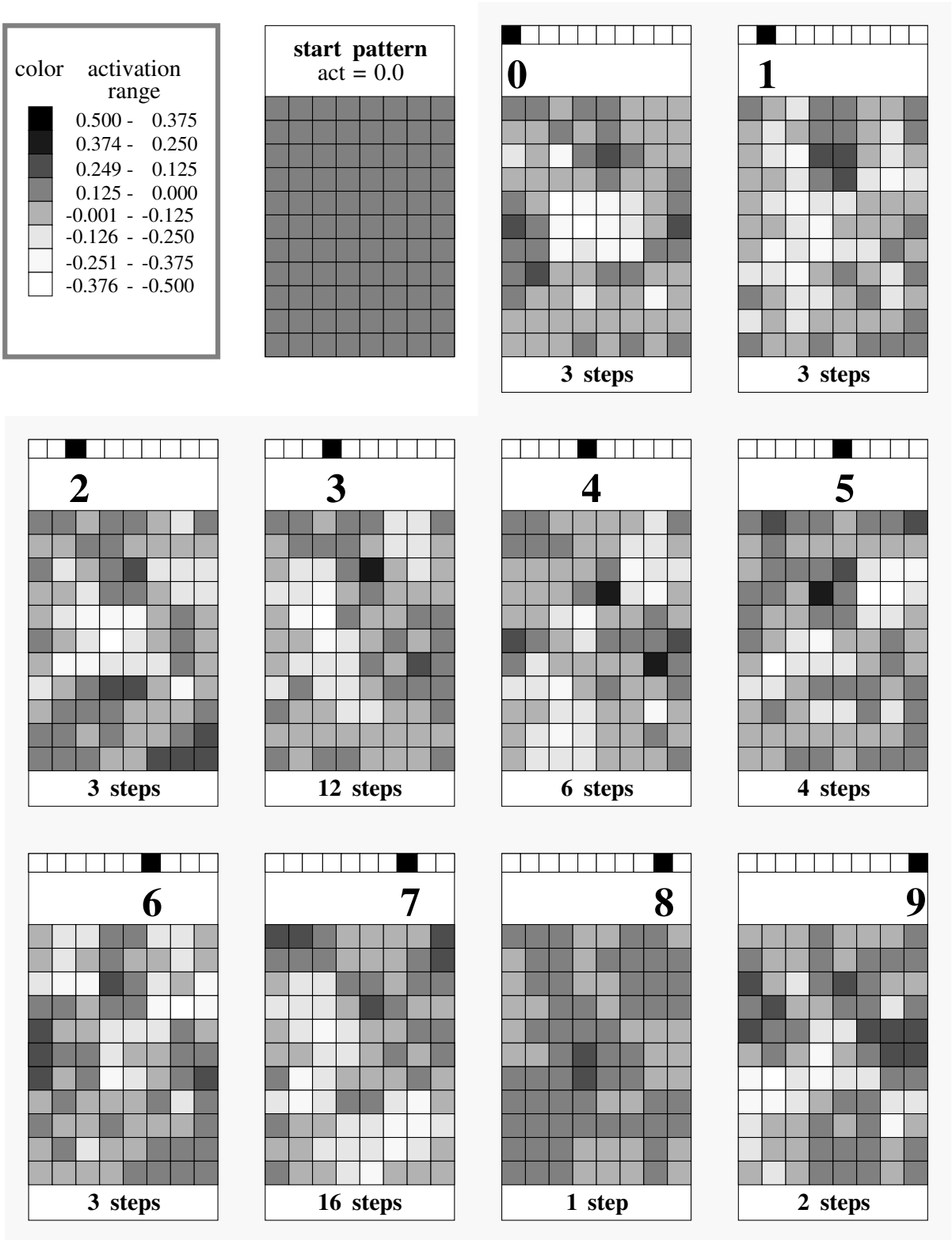

Figure 6: Inversion results: With the start pattern in the upper left corner as initial vector the inversion found several spurious attractors, which are classified by the trained network as digits.

$$E_{inv} = \sum_{t=0}^{7} \sum_{k \in O} (\hat{t}_k(t) - x_k(t))^2 + \mu \sum_{t=0}^{7} \sum_{j \in I} (\hat{I}_j(t) - x_j(t))^2$$

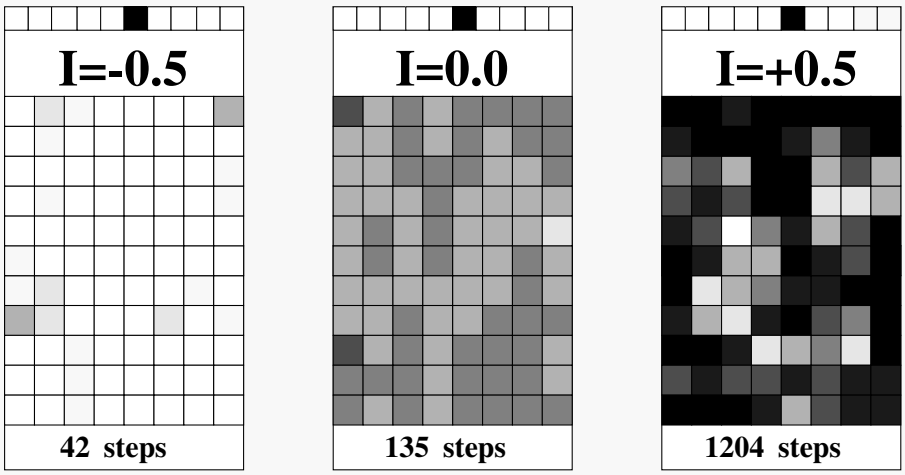

Figure 7: Inversion with an additional constraint for the input units: The output target ($\hat{t}_k(t)$) and the so-called *input target* ($\hat{I}_j(t)$) are approximated simultaneously. Here μ is a fixed factor > 0.

$$E_{inv} = \sum_{t=0}^{7} \sum_{k \in O} (\hat{t}_k(t) - x_k(t))^2 + \mu \sum_{t=0}^{7} \sum_{j \in I} (\tfrac{1}{2} - |x_j(t)|)^2$$

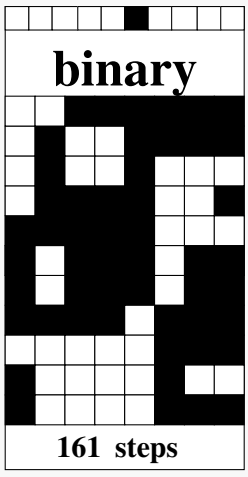

Figure 8: Inversion with a *binary input target*: The input activations become binary.

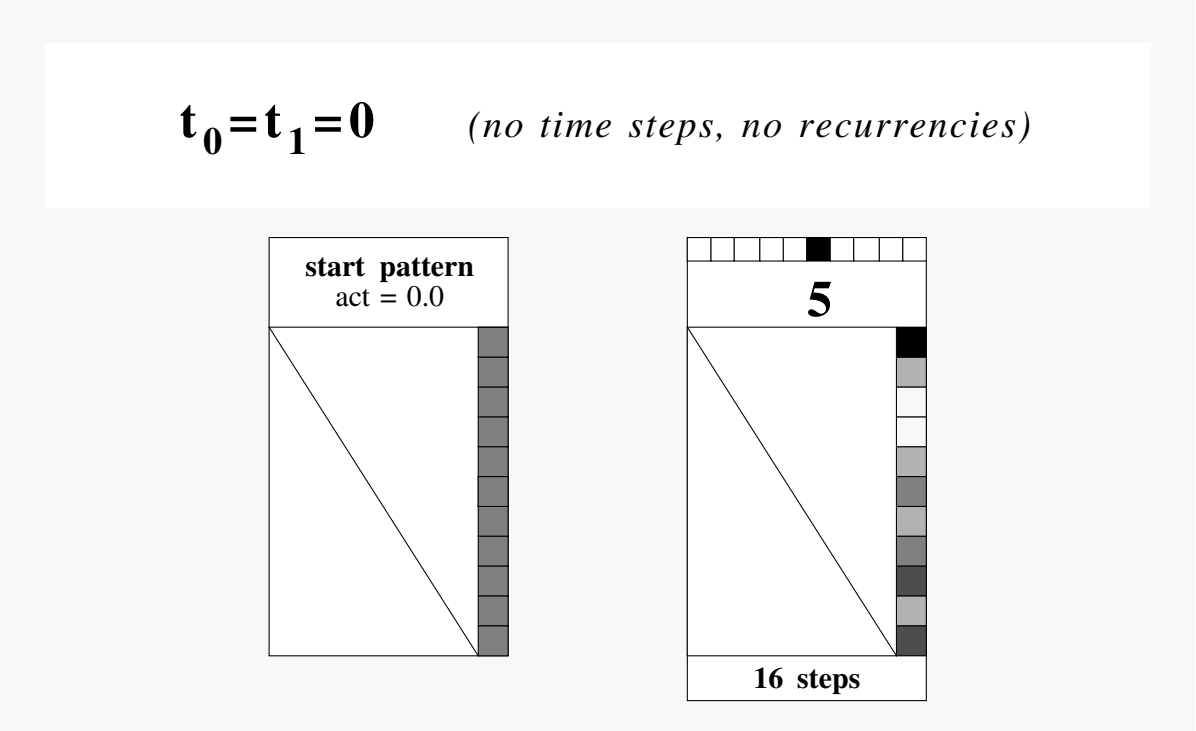

Figure 9: Inversion over one time point only: Starting with the left activation pattern, the network classifies the right input vector as the digit "five", although only one 11-pixel input vector (instead of the eight) was presented.

10 Appendix: The General Gradient Descent Algorithm

Since a backpropagation network is often trained with more than one pattern sequence, we define $\mathcal{P}$ as *set of pattern sequences* with $P = |\mathcal{P}|$ (c.f. chapter 3.4). These pattern sequences are to be numbered as follows: $\mathcal{P} = \{p_1, p_2, ..., p_P\}$. The index p on the input variable $I_i(t)$ and the corresponding error $e_k(t)$ labels the actual pattern sequence $p \in \mathcal{P}$. This definition finally yields the following algorithm in Fig. 10.

```
initialize w_ijn, T_i, x̃_i random   ∀i,j ∈ 𝒩 ,k ∈ 𝒪, n ∈ 𝒮

repeat
   Δw_ijn := 0  ∀i ∈ 𝒩;
   ΔT_i := 0  ∀i ∈ 𝒩;
   Δx̃_i := 0  ∀i ∈ 𝒩;

   for p := 1 to P by 1 do
      E := 0;
      x_i(t_0) := x̃_i  ∀i ∈ 𝒩;
      β^k_ijn(t_0) := 0  ∀i,j ∈ 𝒩 ,k ∈ 𝒪, n ∈ 𝒮;
      α^k_i(t_0) := 0  ∀i ∈ 𝒩, k ∈ 𝒪;
      γ^k_i(t_0) := δ_ik  ∀i ∈ 𝒩, k ∈ 𝒪;

      for t := t_0 + Δt to t_1 by Δt do
         compute I^p_i(t)  ∀i ∈ 𝒩;
         compute net_i(t)  ∀i ∈ 𝒩;                          with (3)
         compute x_i(t)  ∀i ∈ 𝒩;                            with (1) or (6)
         compute e^p_k(t)  ∀k ∈ 𝒪;
         compute β^k_ijn(t)  ∀i,j ∈ 𝒩, k ∈ 𝒪, n ∈ 𝒮;         with (21) or (26)
         compute α^k_i(t)  ∀i ∈ 𝒩, k ∈ 𝒪;                    with (31)
         compute γ^k_i(t)  ∀i ∈ 𝒩, k ∈ 𝒪;                    with (58)
         E := E + E_t · Δt;
         Δx̃_i := Δx̃_i − η Σ_{k∈𝒪} e^p_k(t) · γ^k_i(t)  ∀i ∈ 𝒩;
            Δw_ijn := Δw_ijn − η Σ_{k∈𝒪} e^p_k(t) · β^k_ijn(t)
                      ∀i,j ∈ 𝒩, k ∈ 𝒪, n ∈ 𝒮;
            ΔT_i := ΔT_i − η Σ_{k∈𝒪} e^p_k(t) · α^k_i(t)  ∀i ∈ 𝒩;

         if online-learning then
            w_ijn := w_ijn + Δw_ijn  ∀i,j ∈ 𝒩, n ∈ 𝒮;
            Δw_ijn := 0  ∀i,j ∈ 𝒩, k ∈ 𝒪, n ∈ 𝒮;
            T_i := T_i + ΔT_i  ;ΔT_i := 0  ∀i ∈ 𝒩;
         endif

         if teacher-forcing then
            x_k(t) = x_k(t) − e^p_k(t)  ∀k ∈ 𝒪;              if (15) or (16) holds
         endif
      endfor t
   endfor p

   if not online-learning then
      w_ijn := w_ijn + Δw_ijn  ∀i,j ∈ 𝒩, k ∈ 𝒪, n ∈ 𝒮;
      T_i := T_i + ΔT_i  ∀i ∈ 𝒩;
   endif

   x̃_i = x̃_i + Δx̃_i  ∀i ∈ 𝒩;
until E < ε
```

Figure 10: The general feed-forward algorithm for gradient descent in connectionist networks.

Acknowledgements

We wish to thank KNUT MÖLLER, ALEXANDER LINDEN and JÖRG KINDERMANN for their helpful comments and URSULA BERNHARD for her excellent translational work.

References

[1] R. Allen. Reference in connectionist language users. Poster at Connectionism in Perspective, Zürich Oct. 1988, 1988.

[2] R. B. Allen. Several studies on natural language and back-propagation. In *Proceedings of the International Conference on Neural Networks*, San Diego, CA, 1987.

[3] R. B. Allen. Connectionist state machines. Technical Report ARA-88-300, Bellcore, Morristown, NJ, 1988.

[4] R. B. Allen. Sequential connectionist networks for answering simple questions about a microworld. In *Proceedings of the Cognitive Science Society*, Montreal, 1988.

[5] R. B. Allen. Adaptive training for connectionist state machines. In *ACM Computer Science Conference*, Louisville, 1989.

[6] L. B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Proceedings of ICNN-87, Vol. II, San Diego*, pages 609–618, 1987.

[7] L. B. Almeida. Backpropagation in perceptrons with feedback. In R. Eckmiller and C. von der Malsburg, editors, *Neural Computers*, Berlin, 1988. Springer-Verlag.

[8] D. J. Amit. Neural networks counting chimes. *Proc. Natl. Acad. Sci. USA*, 85:2141–2145, 1988.

[9] C. W. Anderson. Learning and problem solving with multilayer connectionist systems. Technical Report COINS TR 86-50, Dept. of Computer and Information Science, University of Massachusetts, Amherst, MA, 1986.

[10] A. F. Atiya. Learning on a general network. In D. Anderson, editor, *Proceedings of IEEE Conference on Neural Information Processing, Denver, November 8-12*. IEEE, AIP, 1987.

[11] J. Barhen and S. Gulati. "chaotic relaxation" in concurrently asynchronous neurodynamics. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE, IEEE TAB Neural Network Committee.

[12] Y. Chauvin. A back-propagation algorithm with optimal use of hidden units. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, pages 519–526, San Mateo, CA, 1989. IEEE, Morgan Kaufmann.

[13] K. Doya and S. Yoshizawa. Memorizing oscillatory patterns in the analog neuron network. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE, IEEE TAB Neural Network Committee.

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

[15] M. Gherrity. A learning algorithm for analog, fully recurrent neural networks. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE, IEEE TAB Neural Network Committee.

[16] M. Gori, Y. Bengio, and R. De Mori. Bps: A learning algorithm for capturing the dynamic nature of speech. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE, IEEE TAB Neural Network Committee.

[17] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE, IEEE TAB Neural Network Committee.

[18] R. A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.

[19] M. I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Conference on Cognitive Science*, 1986.

[20] M. I. Jordan. Serial order: A parallel distributed processing approach. Technical Report ICS Report 8604, Institute for Cognitive Science, University of California, 1986.

[21] M. I. Jordan. Generic constraints on unspecified target constraints. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE, IEEE TAB Neural Network Committee.

[22] J. Kindermann and J. Linden. Inversion of neural networks by gradient descent. *Parallel Computing*, 1990.

[23] A. Linden. Untersuchung von Backpropagation in konnektionistischen Systemen. Diplomarbeit, Universität Bonn, Bonn, 1990.

[24] A. Linden and J. Kindermann. Inversion of multilayer nets. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE.

[25] L. Massone. Target-switching experiments with a sequential network. Technical report, Dept. of Brain and Cognitive Sciences, MIT, Cambridge, MA, 1989. submitted to Neural Computation.

[26] M. C. Mozer. A focused backpropagation algorithm for temporal pattern recognition. Technical Report CRG-TR-88-3, Depts. of Psychology and Computer Science, University of Toronto, Toronto, Jun 1988.

[27] S. J. Nowlan. Gain variation in recurrent error propagation networks. *Complex Systems*, 2:305–320, 1988.

[28] B. Pearlmutter. Learning state space trajectories in recurrent neural networks. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proceedings og the 1988 Connectionist Models Summer School*, Palo Alto, 1988. Morgan Kaufmann.

[29] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. Technical Report CMU-CS-88-191, Carnegie Mellon University, 1988.

[30] F. J. Pineda. Generalization of backpropagation to recurrent neural networks. *Physical Review Letters*, 59(19):2229–2232, 1987.

[31] D. Psaltis and N. Farhat. Optical information processing based on an associative memory model of neural nets with thresholding and feedback. *Optics Letters*, 10, 1985.

[32] A. J. Robinson. *Dynamic Error Propagation Networks*. PhD thesis, Cambridge University Engineering Dept., Cambridge, UK, February 1989.

[33] A. J. Robinson and F. Fallside. A dynamic connectionist model for phoneme recognition. Technical report, Cambridge University Engineering Dept., Cambridge UK, 1988.

[34] R. Rohwer and B. Forrest. Training time-dependence in neural networks. In M. Caudill and C. Butler, editors, *Proceedings of IEEE Conference on Neural Networks*, San Diego, 1987. IEEE.

[35] R. Rohwer and S. Renals. Training recurrent networks. In *Proceedings of NEURO-88*, Paris, 1988. ENST.

[36] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing. Vol. I + II*. MIT Press, 1986.

[37] D. Servan-Schreiber, A. Cleeremans, and J. McClelland. Encoding sequential structure in simple recurrent networks. Technical Report CMU-CS-88-183, Carnegie Mellon University, CS Dept., Pittsburgh PA, 1988.

[38] S. A. Shamma. Neural networks for speech processing and recognition. In M. Caudill and C. Butler, editors, *First International Conference on Neural Networks*, San Diego, 1987. IEEE.

[39] G. Shumaker, G. Gindi, E. Mjolsness, and P. Anandan. Stickville: A neural net for object recognition via graph matching. Technical Report 8908, Center for Systems Science, New Haven, April 1989.

[40] D. C. Tam and D. H. the late Perkel. A model for temporal correlation of biological neural spike trains. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE, IEEE TAB Neural Network Committee.

[41] D. W. Tank and J. J. Hopfield. Neural computation by concentrating information in time. *Proc. Natl. Acad. Sci. USA*, 84:1896–1900, 1987.

[42] S. Thrun and A. Linden. Inversion in time. In *Proceedings of the EURASIP Workshop on Neural Networks*, Sesimbra, Portugal, February 1990. EURASIP.

[43] S. Thrun, K. Möller, and A. Linden. Adaptive look ahead planning. In *Proceedings OEGAI 90*, 1990.

[44] T. Uchiyama, K. Shimohara, and Y. Tokunaga. A modified leaky integrator network for temporal pattern processing. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE, IEEE TAB Neural Network Committee.

[45] J. Utans, G. Gindi, E. Mjolsness, and P. Anandan. Neural networks for object recognition within compositional hierarchies: Initial experiments. Technical Report 8903, Center for Systems Science, New Haven, February 1989.

[46] A. H. Waibel, H. Sawai, and K. Shikano. Modularity and scaling in large phonemic networks. Technical Report TR-I-0034, ATR Interpreting Telephony Research Laboratories, 1988.

[47] R. J. Williams. Inverting a connectionist network mapping by backpropagation of error. In *8th Annual Conference of the Cognitive Science Society*, Hillsdale, NJ, 1986. Lawrence Erlbaum.

[48] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. Technical Report ICS Report 8805, Institute for Cognitive Science, University of California, San Diego, CA, 1988.

[49] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

[50] L. Wolf. Recurrent nets for the storage of cyclic sequences. In *Proceedings of IEEE Conference on Neural Networks*, San Diego, 1988. IEEE.