In: Proceedings of EURASIP Workshop, February 15-17, 1990, Portugal

INVERSION IN TIME

SEBASTIAN THRUN · ALEXANDER LINDEN
Gesellschaft für Mathematik und Datenverarbeitung mbH
D–5205 St. Augustin, West Germany

Abstract

Inversion of multilayer synchronous networks is a method which tries to answer questions like "What kind of input will give a desired output?" or "Is it possible to get a desired output (under special input/output constraints)?".
We will describe two methods of inverting a connectionist network. Firstly, we extend inversion via backpropagation (Linden/Kindermann [4], Williams [11]) to recurrent (Elman [1], Jordan [3], Mozer [5], Williams/Zipser [10]), time-delayed (Waibel at al. [9]) and discrete versions of continuous networks (Pineda [7], Pearlmutter [6]). The result of inversion is an input vector. The corresponding output vector is equal to the target vector except a small remainder. The knowledge of those attractors *may help to understand the function and the generalization qualities of connectionist systems of this kind.*
Secondly, we introduce a new inversion method for proving the non-existence of an input combination under special constraints, e.g. in a subspace of the input space. This method works by iterative exclusion of invalid activation values. It might be a helpful way to judge the properties of a trained network.
We conclude with simulation results of three different tasks: XOR, morse signal decoding and handwritten digit recognition.

Keywords: connectionist systems, backpropagation, inversion, recurrent neural networks, digit recognition

1 Introduction

In most applications of connectionist networks the set of input/output patterns, the domain of a network, is too wide or partially unknown. Therefore it is impossible to train all patterns. The consequence is that the network is trained with a representative subset (test set) only, hoping for *good* results in a very intuitive manner. Unfortunately, the network has no concept of generalization at all, and so for judging the generalization behavior after the training period, one usually tests the quality of the network with a second finite subset of the input space, the test set. The choice of the training and the test sets are essential for the results one obtains. Furthermore, those results normally cannot give details on the whole generalization qualities.

The usual way of using a trained network can described by the question "Which output is produced by a given input pattern (sequence)?". In order to test in a systematic way the robustness of the trained network we ask the inverse question: "What is the set of input patterns (sequences), which will produce a given output?". In addition we can ask "Is it possible to produce a given output at all?". Answers to these questions may be helpful to understand the function and the generalization of a trained network.

We have developed two methods, one for each question posed above.

2 Unfolding in Time

To apply the inversion methods to recurrent neural networks, we have to convert such networks into equivalent multilayer networks. Then it is sufficient to derive our inversion algorithms for simple multilayer networks.

One can generally describe the synchronous networks which are to be examined in this paper, in the following way:

- A synchronous network can be understood as a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ with nodes $\mathcal{N} = \{1, ..., N\}$ and directed edges $\mathcal{E} = \{(j,i)_t \mid i,j \in \mathcal{N}, t \in \{0,1,...,T\}\}$. Here the tuple $(j,i)_t$ has the semantic interpretation of an edge from node i to node j with a (discrete) time delay of t.

- Every cycle in $\mathcal{G}$, if any, is not free of delays, which means that the sum over all time delays t round the cycle is greater than zero. This constraint is the essential condition for the functionality of a synchronized neural network.

- A weight, a real number $w_{jit} \in I\!R$, is associated to each edge $(j,i)_t \in \mathcal{E}$. During the *learning period*, the weights are adaptive, which is not interesting for inversion.

- The set of input nodes $\mathcal{I} \subseteq \mathcal{N}$ may be defined as the set of the nodes without predecessors. Usually the set of output nodes, denoted by $\mathcal{O} \subseteq \mathcal{N}$, is disjunct from the set of input nodes. It includes all nodes without any successor.

For unfolding the network in time (Rumelhart/McClelland [8], Mozer [5]), the graph $\mathcal{G}$ has to be transformed into its spatial representation. Therefore it is necessary to know the maximum length over all input sequences $\hat{T} \in I\!N \cup \{+\infty\}$. Note that in the case of $\hat{T} = +\infty$ the spatial representation will be of infinite length. This case is neither practicable, nor interesting.

To construct the unfolded network, we have to take three steps:

- First we have to reduce the set of edges to the relevant edges for computation:

$$\mathcal{E}' := \{(j,i)_t \in \mathcal{E} \mid t \leq \hat{T}\} \tag{1}$$

  It is easy to see that all *forgotten* edges are useless, because time delays greater than $\hat{T}$ are too slow to influence the result of a sequence of time length $\hat{T}$.

- Secondly, each node gets $\hat{T}$ counterparts, one for every relevant time step:

$$\hat{\mathcal{N}} := \left\{ n_t \mid n \in \mathcal{N}, t \in \{0, ..., \hat{T}\} \right\} \tag{2}$$

  Now we have one original network for every existing time step. This is the main idea of the unfolding procedure.

- Finally, the edges are converted into delayless edges:

$$\hat{\mathcal{E}} := \left\{ (j_{\tau+t}, i_\tau)_0 \mid (j,i)_t \in \mathcal{E}' \ \wedge \ \tau \in \{0, ..., \hat{T} - t - 1\} \right\} \tag{3}$$

  An edge $(j,i)_{\tilde{t}}$ with the time delay $\tilde{t}$ will be converted into several edges. Each of them connects the corresponding unit i_t in the network at time step t with the node $j_{t+\tilde{t}}$ in the network at time step $t + \tilde{t}$.

Since the new corresponding graph $\hat{\mathcal{G}} = (\hat{\mathcal{N}}, \hat{\mathcal{E}})$ is free of cycles and free of time delays, the inversion method by Linden/Kindermann [4] can be used. Now we can go so far as summarizing their inversion method.

3 Inversion via Backpropagation

One way of inverting a multilayer network is the usual gradient search for minimizing the Least-Mean-Square-Error-Function

$$E = \sum_{k \in \mathcal{O}} (t_k - a_k)^2 \tag{4}$$

with the given target $T = (t_1, ..., t_m)$ as desired output. With an arbitrary starting vector $I^{(0)}$ it is possible to approximate the desired output vector via error backpropagation step by step $(t = 0, 1, ...)$:

- Compute the network output $O^{(t)} = F(I^{(t)})$ as usual.
- Propagate for all $i \in \mathcal{I}$ the error signals

  $$\delta_i = -\frac{\partial E}{\partial a_i} \tag{5}$$

  back to the input layer. This is done by the usual backpropagation algorithm; the resulting values δ_i tell the input units how to change in order to decrease the total output error.
- Use the δ_i's to change the input vector in small steps:

  $$I_i^{(t+1)} = I_i^{(t)} + \gamma \cdot \delta_i^{(t)} \tag{6}$$

  with the stepwidth $\gamma > 0$.

As a remark, working with the activation gradient (equation (5)) leads to unbounded activations which violate the usual constraint that activities are bounded (by $[0, 1]$, for example). When using a pseudo net-input and its gradient

$$\delta_i = -\frac{\partial E}{\partial net_i} \tag{7}$$

instead of equation (5), the activations are always in the correct interval. In this case, the computation works slower.

Sometimes it is useful to approximate a certain input vector $\tilde{I}$ simultaneously. This is the case, if special qualities of the resulting input vector are wanted. Extending the error function (4) by an input-target term leads to

$$\tilde{E} = \left(\sum_{k \in \mathcal{O}} (t_k - a_k)^2 \right) + \mu \cdot \left(\sum_{j \in \mathcal{I}} (\tilde{I}_j - a_j)^2 \right). \tag{8}$$

In this equation μ expresses the *importance* of the constraint with respect to the input-target. The error propagation function in the input layer is changed as follows:

$$\delta_i = -\frac{\partial \tilde{E}}{\partial net_i} = -\frac{\partial E}{\partial net_i} - 2\mu \cdot (\tilde{I}_j - a_j) \tag{9}$$

The resulting inversion method searches for an input vector which approximates both a given output T and a *target-input* $\tilde{I}$ simultaneously. In order to avoid getting stuck in local minima, it is important to have a *good* choice of the starting vector $I_i^{(0)}$. In our applications we did not get difficulties with disadvantageous starting vectors.

4 Inversion with Validity Intervals

Until now we have described a method to invert a network with an approximation method over the input space. Unfortunately, this method cannot answer questions like "Is there any input which produces my given target in general?".

The idea of inverting the unfolded network with validity intervals is to associate an interval of valid output activity $[\alpha_i, \beta_i]$ with each unit $i \in \hat{\mathcal{N}}$. Let us consider an arbitrary unit i in the network. On the one hand the validity interval depends on the maximum valid input the unit can get from its predecessors $k \in P(i)$. But on the other hand it also depends on the validity intervals of every succeeding unit $j \in S(i)$, because it must be ensured that each of those units is able to get an input which produces an output in their own validity interval. On the whole, there are two directions to propagate constraints through the network. So the validity intervals can be decreased iteratively. We derive the algorithm in the appendix.

Since each sequence of intervals is monotonous and bounded by $[0, 1]$, it has to converge. There are two criteria for finishing the iteration: a) the accumulated changes are smaller than a small ϵ, no significant changes are expected here and b) a contradiction comes forward: $\alpha_i > \beta_i$. In this case the computation has shown that a valid possibility for covering the remaining free variables does not exist.

After *good* approximation, the user has to fix one or several input activities by setting the corresponding intervals to $[\alpha_i, \alpha_i]$. After doing so, the approximation algorithm has to be repeated. The essence of this method is the contradiction that may appear. In this way, we have got a possibility to show that a given target cannot be obtained with certain constraints. The tool of non-existence proofs is useful in the following considerations.

5 Inversion in Time

Using the method of *Unfolding in Time*, it is easy to apply the two procedures described above to recurrent and time-delayed networks. In several cases the length of an input sequence is not determined. Then it is necessary to ask questions like "What kind of input with length $\hat{T}$ will give a desired output?" or "What kind of input with minimal length will give a desired output?". If the length is fixed by $\hat{T}$, an unfolded network of length $\hat{T}$ can be constructed in a simple way. In the second case one can deal with a sequence of unfolded networks $\mathcal{G}^{(0)}, \mathcal{G}^{(1)}, \ldots$ with corresponding spatial lengths $0, 1, \ldots$. The method of validity intervals covers elements of this network sequence that are not able to give the desired output.

6 Simulation Results

In all our simulations we used the symmetric activation function instead of the usual one (compare equation (12)):

$$s[net_i] = \frac{1}{1 + e^{net_i}} - \tfrac{1}{2} \tag{10}$$

The inversion algorithms are not touched by this modification.

6.1 Inversion of an XOR-Network

In the first experiment we trained a backpropagation network with 2 input, 2 hidden and one output unit using the standard XOR patterns (see figure 1). After training, we set the output activity

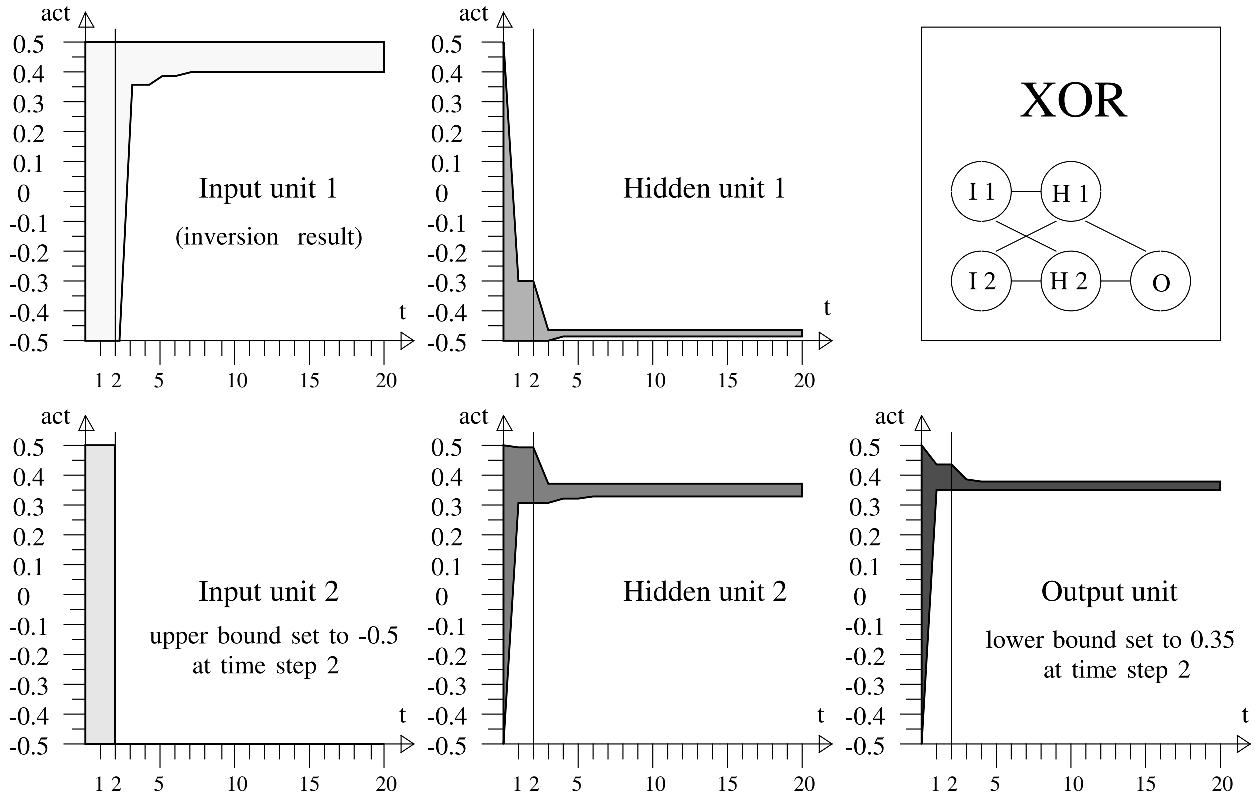

Figure 1: Inversion of an XOR-network with validity intervals. The lower bound of the output interval was set to 0.35. After two iterations all changes were done. Then the upper bound of the second input interval was set to -0.5. The algorithm finished after 18 further iterations.

interval to [0.35, 0.5] and applied the inversion with validity intervals algorithm to the network. In fact, the two binary input combinations (0|1) and (1|0) provide such an output, therefore the input intervals did not change fundamentally. After clamping a low level signal to the second input unit by setting their interval to $[-0.5, -0.5]$ (at time step 2), the interval of the other input unit went to [0.399, 0.5] (see also figure 1). So we could prove the intuitive statement: For getting a clearly positive output with a negative input (-0.5) the remaining input activation has to be low. Another interesting point was the resulting interval of the output unit: the maximum output activation was 0.379 instead of 0.5.

Tests with other combinations showed similar results. The completeness of the binary training set lead to a trained network without false attractors. Knowing this, inversion via backpropagation seemed to be less interesting in this context.

6.2 Morse-Decoding

A simple sequential approach is the decoding of morse signals. We constructed a network consisting of one input, five hidden, five context and 26 output units (Elman [1], see figure 2). One signal, either long or short, was fed into the net in one time step. Short signals were represented by low activation values ($\in [-0.5, -0.3]$) and long signals by high activation values ($\in [0.3, 0.5]$). The training set consisted of the whole alphabet (26 letters) and the networks' task was to classify a morse coded letter. The input length was between one and four signals and the output coding was a local coding (1 of 26). After 658 training epochs the network had "learned" all signals.

We were very interested in finding an input sequence with *false length*, e.g. an input pattern of length 1, 2 or 4, which produces the output "g" (see figure 2: the length of "g" is 3). After

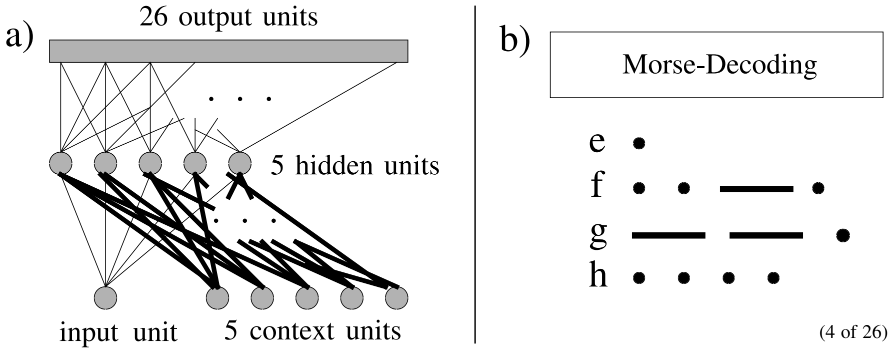

Figure 2: a) Network for Morse-decoding – the bold lines are both forward and recurrent lines and b) some example morse digits

several inversions via backpropagation with the four test letters "*e*", "*f*", "*g*" and "*h*" and with different starting patterns, we did not find a false attractor. So it seemed reasonable to assume that such false attractors do not exist. Now we applied inversion with validity intervals to the trained network. We could prove that there is no attractor of false length for those four target patterns at all.

6.3 Handwritten Digit Recognition

Finally, 490 handwritten digits were trained with a recurrent network similar to those proposed by Elman [1] and Jordan [3]. Every picture of a handwritten digit was coded by an (11×8)-pixel matrix. These digits were trained in 8 time steps with an 11 pixel window technique, hence the employed network consisted of 11 input, 20 hidden, 15 context and 10 output units (one output for each digit), with full connections without time delay from input to hidden, context to hidden and hidden to output layer and full connections back from hidden to context and from context to context layer with the time delay of one discrete time step. After the training procedure with the usual backpropagation, we tested the performance of the network with a (disjunct) test set, containing 140 further handwritten digits. 82 percent of the test patterns were classified correctly (that is with the error square sum less then 0.2). Keeping the generalization results in mind, we used inversion via backpropagation to compute an input vector which produced the output *five*. We got many spurious attractors – see figure 3.

After training, a *typical five* – taken from the training set – produced the correct output shown in the first pattern of figure 3. The inversion experiment examined here started with a *don't know pattern*, that means with all input activations clamped to 0.0. After about 700 approximation steps we got a very typical result of inversion via backpropagation: the third activation pattern produced the output *five*, except a very small remainder. Using an input-target of 0.0 for each input unit and setting μ to 0.1, after 135 further iterations we got an inversion result which was very similar to the zero matrix and also produced the desired output (fourth pattern). With an input-target of -0.5 the algorithm found an input pattern with (sub)maximal negative activations: the fifth pattern.

Usually the input set is defined as binary at low level pattern recognition. In our training set all input pixels were binary, too. For forcing a binary inversion solution, we used an expanded inversion energy function (see eqation (8)):

$$\tilde{E} = \left(\sum_{k \in \mathcal{O}} (t_k - a_k)^2 \right) + \mu \cdot \left(\sum_{j \in \mathcal{I}} (\tfrac{1}{2} - |a_j|)^2 \right). \tag{11}$$

This energy function pulls the input activation to the extremal (binary) values. Again we started with the first inversion result (figure 3, third pattern). After 175 further iterations we got the result

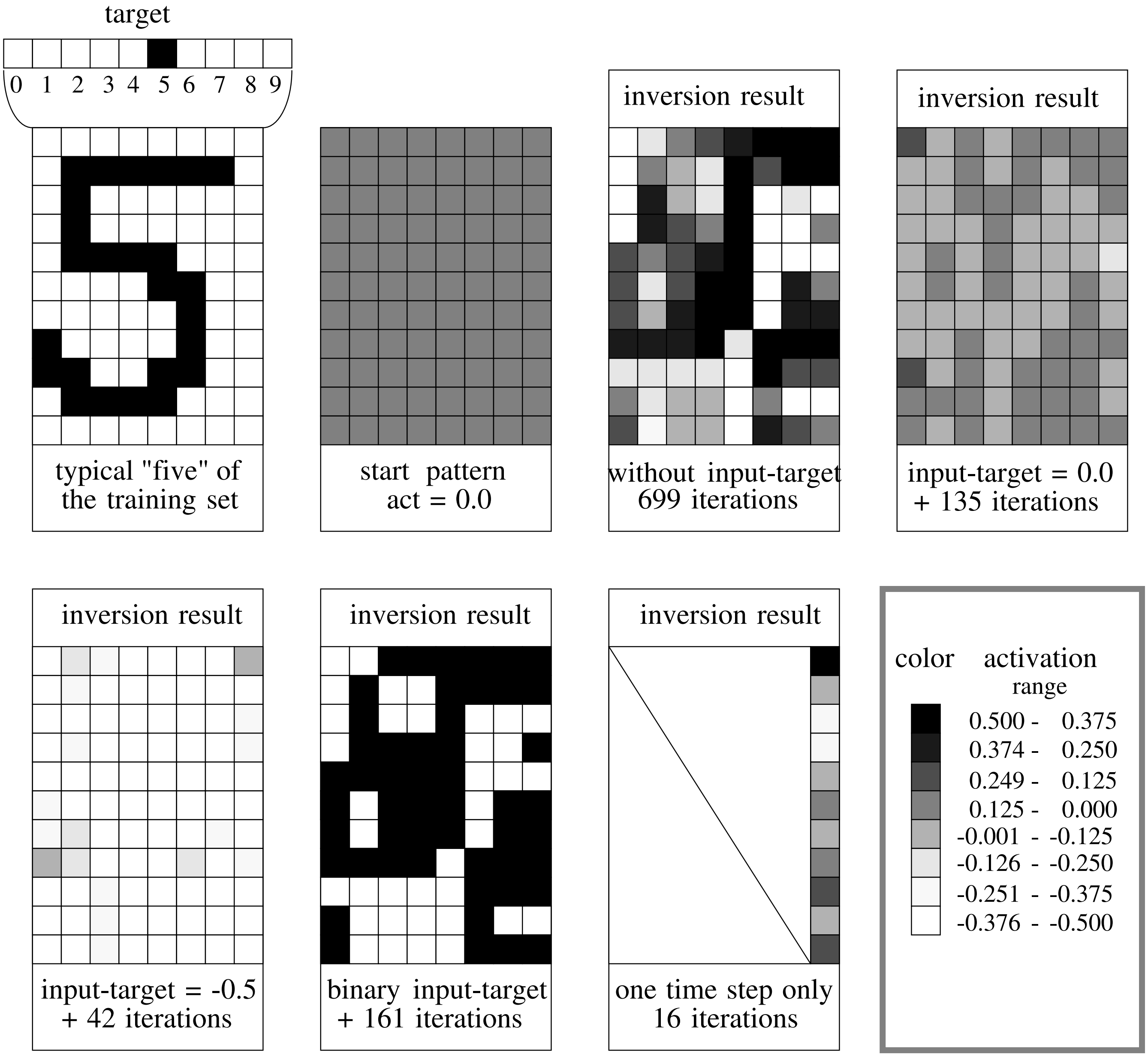

Figure 3: Some backpropagation inversion results at digit recognition

shown in the sixth pattern of figure 3. It does not look like a *five*, actually.

As you can see in figure 3, we wanted to know whether a given target *five* could be produced in only one step. This is not a relevant question for digit recognition, but we think it is an interesting point in many applications of recurrent networks. The seventh input pattern was mapped to the output *five*. Of course, we repeated these experiments with all ten defined targets and got similar results each time.

Our interpretation of this result reads: (usual) backpropagation with unstructured networks is not suitable to learn digit patterns at all. Although the generalization rate is not bad, there are lots of unknown (*random*) input vectors which act like a certain digit picture.

7 Outlook

We have applied the two different methods of inversion to three problems. The XOR-problem is a very simple binary problem with a complete training set. By using inversion with validity intervals we pointed out the clear classification qualities of the XOR-network directly. Next, we were interested in a sequential problem. We examind a recurrent network for morse signal recognition. Although the training set was (quasi) complete, we looked for spurious attractors with false time length at four different test letters, using inversion via backpropagation. We did not find any; in addition we proved by inversion with validity intervals that there are no spurious attractors for our test letters. At last we did some experiments on handwritten digit recognition using a recurrent architecture. Here the training set was incomplete. Inversion via backpropagation gave us some interesting examples for spurious attractors.

In this paper we have shown how to use the two inversion methods to get an impression of the generalization concept of a trained discrete recurrent network. On the same way one might apply our methods to continuous networks (Pearlmutter [6], Pineda [7]). In general, the method of inversion with validity intervals is more useful for smaller networks and binary tasks; inversion via backpropagation provides good results at large networks which have to learn fuzzy classifications. Currently we are on the point to improve the validity interval algorithm for special approaches like pattern recognition and/or sequence learning.

The next thing we are interested in is using the results of inversion via backpropagation for improving the training set. Inversion uncovers spurious attractors. In order to avoid them, they can be used as counterexamples. However, a kind of reinforcement is necessary to decide whether an inversion result is a positive or a negative example. First results show that the generalization performance can be improved by expanding the training set with those patterns purposefully.

Inverting a continuous temporal network with a predefined class of input functions is not possible yet. It might be interesting to look for a method to invert those kinds of networks with a good choice of the input functions' class.

8 Appendix

Derivation of the Inversion with Validity Intervals Algorithm

Since we do not see a possibility to compute the validity interval of a unit directly, we offer a simple local approximation algorithm by using the monotony of the logistic activation function:

$$a_j = s[net_j] = s\left[\sum_i w_{ji} a_i\right] = \frac{1}{1+e^{net_j}} \tag{12}$$

The monotony ensures that in its inverse function

$$s^{-1}[a_j] = net_j = -ln\left(\frac{1}{a_j} - 1\right) \tag{13}$$

the image of an interval is an interval, too.

The algorithm: Let us start with the **Initialization**:

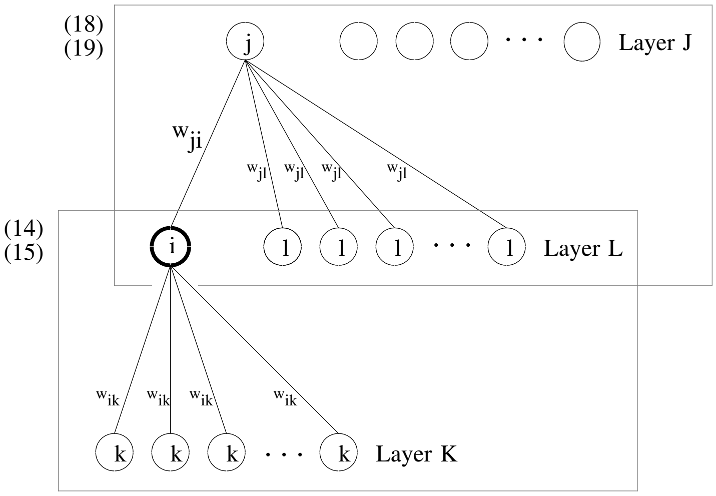

Figure 4: A lattice representation of (14), (15), (18) and (19).

At the beginning of approximation, there are two different kinds of units, which will be handled similarly: a) the units with a given activation target interval, i. e. the output units or certain input units and b) the other units, the *free* units. At the beginning of the algorithm each interval of a restricted unit will be initialized by its restriction interval – the free units' intervals will be set to the largest allowed interval $[0, 1]$.

Loop: Compute the new interval serially/in parallel for every unit $i \in \hat{\mathcal{N}}$:

1. The validity output interval of unit i depends on the *maximal* net-input. Every predecessor $k \in P(i)$ of unit i has got a validity interval itself. Therefore, for getting the maximal output range of unit i depending on all predecessors k, the (new) lower bound at the iteration step $(t+1)$ yields

$$\bar{\alpha}_i^{(t+1)} = s\left[\left(\sum_{k\in P(i)} m_\alpha(\alpha_k^{(t)}, \beta_k^{(t)}, w_{ik})\right) - \Theta_i\right] \tag{14}$$

and the corresponding upper bound yields

$$\bar{\beta}_i^{(t+1)} = s\left[\left(\sum_{k\in P(i)} m_\beta(\alpha_k^{(t)}, \beta_k^{(t)}, w_{ik})\right) - \Theta_i\right]. \tag{15}$$

The functions m_α and m_β are defined by

$$m_\alpha(\alpha_k^{(t)}, \beta_k^{(t)}, w_{ik}) = \begin{cases} \alpha_k^{(t)} \cdot w_{ik}, \text{ if } w_{ik} \geq 0 \\ \beta_k^{(t)} \cdot w_{ik}, \text{ otherwise} \end{cases} \tag{16}$$

$$m_\beta(\alpha_k^{(t)}, \beta_k^{(t)}, w_{ik}) = \begin{cases} \alpha_k^{(t)} \cdot w_{ik}, & \text{if } w_{ik} < 0 \\ \beta_k^{(t)} \cdot w_{ik}, & \text{otherwise.} \end{cases} \tag{17}$$

One should interpret the function m_α as the minimum under w_{ik} and m_β as the corresponding maximum. It is easy to see that $[\bar{\alpha}_i^{(t+1)}, \bar{\beta}_i^{(t+1)}]$ is the maximum activity range of a unit under the maximum possible input from all its predecessors.

2. Now we have to update the interval $[\alpha_i, \beta_i]$ depending on every succeeding unit $j \in S(i)$ and its predecessors $l \in P(j)\backslash\{i\}$.

   In order to derive the equations (18) and (19) let us assume the weight w_{ji} to be positive. The new lower bound $\hat{\alpha_{ij}}^{(t+1)}$ of unit i is its activation, it has to contribute to the input of unit j, assuming all other predecessors of unit j having their *optimal* activation. By *optimal* activation we mean the activation with the largest contribution to the input of unit j. The first case of equation (18) shows the resulting mathematics.

   The complete algorithm is as follows:
   Compute for every $j \in S(i)$ the (possible) new lower bounds:

$$\hat{\alpha}_{ij}^{(t+1)} = \begin{cases} \frac{s^{-1}\left[\alpha_j^{(t)}\right] - \left(\sum_{l \in P(j)\backslash\{i\}} m_\beta(\alpha_l^{(t)}, \beta_l^{(t)}, w_{jl})\right) - \Theta_j}{w_{ji}} & \text{if } w_{ji} > 0 \\ \frac{s^{-1}\left[\beta_j^{(t)}\right] - \left(\sum_{l \in P(j)\backslash\{i\}} m_\alpha(\alpha_l^{(t)}, \beta_l^{(t)}, w_{jl})\right) - \Theta_j}{w_{ji}} & \text{if } w_{ji} < 0 \\ 0, & \text{otherwise} \end{cases} \tag{18}$$

   and the (possible) new upper bounds analogously:

$$\hat{\beta}_{ij}^{(t+1)} = \begin{cases} \frac{s^{-1}\left[\beta_j^{(t)}\right] - \left(\sum_{l \in P(j)\backslash\{i\}} m_\alpha(\alpha_l^{(t)}, \beta_l^{(t)}, w_{jl})\right) - \Theta_j}{w_{ji}} & \text{if } w_{ji} > 0 \\ \frac{s^{-1}\left[\alpha_j^{(t)}\right] - \left(\sum_{l \in P(j)\backslash\{i\}} m_\beta(\alpha_l^{(t)}, \beta_l^{(t)}, w_{jl})\right) - \Theta_j}{w_{ji}} & \text{if } w_{ji} < 0 \\ 1, & \text{otherwise} \end{cases} \tag{19}$$

   (Notice that we define $\ln(0) := -\infty$.) These equations do nothing else than what is explained above: each $[\hat{\alpha}_i^{(t+1)}, \hat{b}_i^{(t+1)}]$ is the maximum activity range under the demanded input of the successor l.

3. Finally we have to choose the strongest limitations to construct the new validity interval of unit i:

$$\alpha_i^{(t+1)} = \max_{j \in S(i)} \left\{ \hat{\alpha}_{ij}^{(t+1)}, \bar{\alpha}_i^{(t+1)}, \alpha_i^{(t)} \right\} \tag{20}$$

$$\beta_i^{(t+1)} = \min_{k \in S(i)} \left\{ \hat{\beta}_{ij}^{(t+1)}, \bar{\beta}_i^{(t+1)}, \beta_i^{(t)} \right\} \tag{21}$$

   Of course, it is possible to involve special constraints, like "the activation of unit i always is equal to the activation of unit j" for two units i and j. This will be done here and in the equations above. The reader may find sensible constraints by him/herself.

References

[1] J. L. Elman, *Finding Structure in Time.* Technical Report CRL Technical Report 8801, Center for Research in Language, University of California, San Diego, 1988

[2] G. E. Hinton, *Connectionist Learning Procedures.* Technical Report CMU-CS-87-115, Pittsburgh, 1987

[3] M. I. Jordan, *Serial Order: A Parallel Distributed Processing Approach.* Technical Report ICS Report 8604, Institute for Cognitive Science, University of California, B1986

[4] A. Linden, J. Kindermann, *Inversion of Multilayer Nets.* Proceedings of the First International Joint Conference on Neural Networks, Washington, 1989

[5] M. C. Mozer, *A Focused Back/Propagation Algorithm for Temporal Pattern Recognition.* Technical Report CRG-TR-88-3, 1988

[6] B. A. Pearlmutter, *Learning State Space Trajectories in Recurrent Neural Networks.* Technical Report CMU-CS-88-191, 1988

[7] F. J. Pineda, *Generalization of backpropagation to recurrent neural networks.* Physical Review Letters, 59/19:2229-2232, 1987

[8] D. E. Rumelhart, J. L. McClelland, *Parallel Distributed Processing.* The MIT Press, University of California San Diego, 1986

[9] A. Waibel, H. Sawai, K. Shikano, *Modularity and Scaling in Large Phonemic Networks.* Technical Report TR-I-0034, ATR Interpreting Telephony Research Laboratories, 1988

[10] R. J. Williams, D. Zipser, *A Learning Algorithm for Continually Running Fully Recurrent Networks.* ICS Report 8805, 1988

[11] R. J. Williams, *Inverting a Connectionist Network Mapping by Backpropagation of Error*, Proceedings 8th Annual Conference of the Cognitive Science Society, Lawrence-Erlbaum 1986