Handling Time-Warped Sequences with Neural Networks[*]

Claudia Ulbricht

Austrian Research Institute for Artificial Intelligence
Schottengasse 3, A–1010 Vienna, Austria
claudia@ai.univie.ac.at

Abstract

Being able to deal with time-warped sequences is crucial for a large number of tasks autonomous agents can be faced with in real-world environments, where robustness concerning natural temporal variability is required, and similar sequences of events should automatically be treated in a similar way. Such tasks can easily be dealt with by natural animals, but equipping an animat with this capability is rather difficult. The presented experiments show how this problem can be solved with a neural network by ensuring slow state changes. An animat equipped with such a network not only adapts to the environment by learning from a number of examples, but also generalizes to yet unseen time-warped sequences.

1 Introduction

For numerous tasks, autonomous agents have to be able to deal with time-warped sequences of events. Sequential patterns of variable length are common in real-world environments, and the number of available training examples are usually relatively small. Animats should not only be able to adapt to the given environment, but also automatically treat similar sequences of events in a similar way. They should be robust concerning natural temporal variability. Therefore, models capable of generalizing to time-warped patterns are needed. The neural network presented in this paper is shown to fulfil this requirement. Since it uses

[*]This is a revised version of the paper: C. Ulbricht, Handling Time-Warped Sequences with Neural Networks, From Animals to Animats 4, Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, September 9th-13th, 1996, Cape Cod, Massachusetts, P. Maes, M.J. Mataric, J.-A. Meyer, J. Pollack and S.W. Wilson, Bradford, 1996.

the input patterns for state formation, it is called the input state network. This model of temporal processing solves the given task by employing slowly changing states.

The task is described in Section 2. A brief overview of sequence processing with neural networks is given in Section 3. The two network models used in the experiments are presented in Section 4. In Section 5, the setup of the experiments is described, and the results can be found in Section 6.

2 The Task of the Agent

The Agent in its Environment: The task of the autonomous agent is to turn to the appropriate direction after having perceived a certain sequence of events. This can, for instance, be an animat which is situated in a maze and has to learn that the target (i.e. some reward like food) is to the left or right, depending on the characteristics of the aisle which it has passed. For instance, after having perceived a sequence like AAPD or AAPPD, the target is to the right, while it is to the left after sequences like XXPD or XXPPD. A and X represent arbitrary sources of sensory information, P denotes an aisle without any specific information, and D a decision point at an intersection. The input P models the pause between the input information, which is relevant to the decision, and the point in time when the decision is actually required.

Since passing some source (like A or X) takes some time, it is perceived several times in a row. The idea of repeating the sensory input is to simulate a real-world environment where a single appearance of an input is treated as noise. An input is not worth being noticed by the model unless it is present for some time and can be sampled several times in a row. The exact number of samples may vary, which may result in sequences like AAAPPPD or AAAAAPPD. These sequences are similar in that the overall order of sequence elements A,P, and D is maintained. Therefore, a new type of notation is introduced, where [X*] represents a sequence of an arbitrary number of sequence elements X. The sequences AAAPPPD and AAAAAPPD can thus be written as:

[A*] [P*] D.

The target of the autonomous agent is to the right after one of the sequences [A*][P*]D, [B*][P*]D, and [C*][P*]D, while it is to the left after one of the sequences [X*][P*]D, [Y*][P*]D, and [Z*][P*]D. The agent at such an intersection where a decision is required is depicted in Fig. 1.

The agent can perceive A, B, C, X, Y, Z, P, D, L, or R. It is trained to predict the location of the goal at the correct point in time. The goal is hidden so that it cannot be perceived before passing the intersection. The outputs (L or R) denote turns to the left and right respectively. The pause P is used to tune the time span between information input and required output. It is needed for designing an environment where the distance between the relevant information

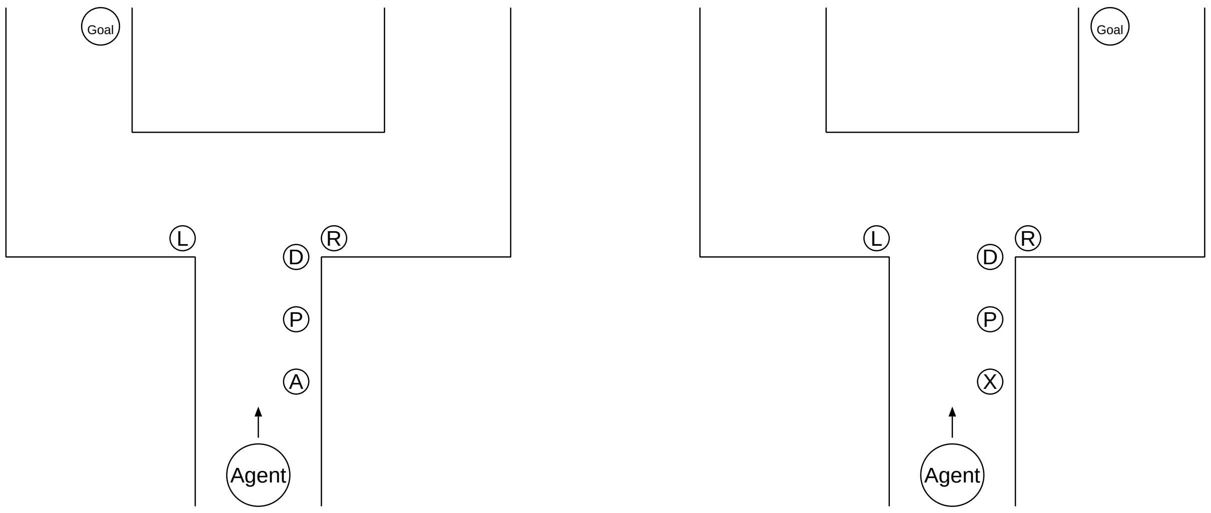

Figure 1: *The agent in its environment*

and the required reaction is variable. The input pattern D denotes a decision point where the agent has to turn left or right.
A typical input sequence consisting of three subsequences looks as follows:

AAAAPPPPPPPDLLLLLLLL
ZZZZZPPPPDRR
CCCCCCCCCPPDLLL .

In the concise format, this can be written as:

[A*][P*]D[L*][Z*][P*]D[R*][C*][P*]D[L*] .

This is the sequence of prototypical elements. The actual number of repeated samples varies dependent on the path followed by the agent and on the distance between the agent and the object. This variation is simulated by randomly determining how long each input is perceived.

The requirements of the task: The given environment requires a model which can memorize information for several time steps. The relevant criterion is the **length of the time period** between the information input and the point in time when this information is needed to produce the correct output. This time period can also be regarded as the **time lag** between an event and the desired response of the system. Long time lags, as they can be present in the given task, lead to **long input-output dependencies**.

The given task provides another challenge. It requires not only dealing with long dependencies, but also with **dependencies of variable length**. Since the sequence elements are samples taken at discrete intervals of time from a continuous environment, the sampling rate has no correspondence to any feature of the real-world environment. In such types of environment, subsequences which are slightly temporally displaced should be recognized as being similar and treated similarly unless this difference is actually relevant to the task, which it usually is not. In other words, **time-warped input patterns** should still be recognized. Designing models which are robust in respect to natural temporal variability is also the aim of the work presented in [Port and Anderson, 1989].

To summarize, the characteristics of realistic event sequences require a model capable of

- coping with dependencies of several time steps, and
- generalizing to time-warped patterns.

3 Neural Network Solutions

A sequence consists of sequence elements which are ordered in time. Usually, there is a single sequence element x per time step t:

$$\ldots, x(t-1), x(t), x(t+1), \ldots$$

Selecting an appropriate neural network is already difficult for static tasks, but when temporal tasks are to be handled, model selection becomes much more difficult. A task which involves only short dependencies can be handled more easily than one which involves long ones. Moreover, sequences should be treated similarly when they are stretched or squeezed with respect to the temporal dimension. Handling such time-warped sequences is not trivial, because they are not automatically recognized as being similar.

Even though most neural network research has focused on processing single patterns, a large number of approaches to handling sequences have been developed and tested. Sequence processing requires a method for saving information for subsequent time steps. Detailed overviews of neural networks for handling temporal aspects are given, for instance, in [Ulbricht *et al.*, 1992], [Mozer, 1993], [Rohwer, 1994], and [Chappelier and Grumbach, 1994].

Here the following four methods are distinguished and briefly addressed:

1. Layer delay without feedback
2. Layer delay with feedback
3. Unit delay without feedback
4. Unit delay with feedback

These methods can be employed in different layers of the neural network. They can also be associated with different weights and delays of different length.

Since they can be combined in various ways, the number of resulting combinations and thus actual network models, which can be used for handling sequences, are quite large.

The presented categorization scheme for sequence-handling methods is, in some parts, comparable to other taxonomies found in literature. The two categorization schemes provided by [Catfolis, 1994] and [Chappelier and Grumbach, 1994] refer mainly to the methods for sequence handling, whereas the scheme presented in [Mozer, 1993] covers also the trainability of network components.

1. **Layer Delay without Feedback**

   The most straightforward approach is to use an **input window** which holds a restricted small number of past sequence elements. Then this part of the time series is analyzed before the window is shifted further in time by one or more elements. The same result can be achieved by delaying the contents of the input layer several times in a row. A network that contains no connections with time delays which feed information back so that loops emerge is called a **non-recurrent network**.

   The complete mapping of a feedforward network from the input $\mathbf{i}(t)$ to the output $\mathbf{o}_1(t)$ can be written as:

   $$\mathbf{o}_1(t) = \mathbf{F}_1(\mathbf{i}(t)), \tag{1}$$

   where $\mathbf{F}_1$ is the mapping of a neural network with a hidden layer, which is also called a "multi-layer perceptron."

   The output $\mathbf{o}_2(t)$ of a network with an input window is

   $$\mathbf{o}_2(t) = \mathbf{F}_2\left(x(t-1), x(t-2), \ldots, x(t-p)\right), \tag{2}$$

   where p is the window size, and $x(t-1), x(t-2), \ldots, x(t-p)$ are the elements of the input sequence. This is comparable to auto-regressive models (AR-models) of order p, as they are treated in [Box and Jenkins, 1970]. Apart from the non-linear mapping, the two models are identical.

2. **Layer Delay with Feedback**

   Layers can be delayed and fed back to previous layers in the updating order. The resulting **cycles** in such **recurrent networks** allow the output of a unit to return to the same unit at a later point in time. Hidden and output layer feedback are the most common forms of layer delay with feedback.

   If recurrent networks are updated like feedforward networks with a single update cycle per time step, they keep the general characteristics of feedforward networks. Models of this kind are also called "simple recurrent networks," "recurrent feedforward networks," or "partially recurrent networks" [Hertz *et al.*, 1991].

The context provided by the feedback loop can also be regarded as the **state** the network is in. Therefore, the layer keeping the delayed information is also referred to as the "state layer." This idea of state is comparable to that in state automata. There, the new state is solely dependent on the previous state and the last input. The state subsumes the input history. This is related to the Markov property, which ensures that all the information which is relevant to the future is contained in the present state.

In a simple recurrent network with hidden or output layer feedback, the network output is a function of the input $\mathbf{i}(t)$ and the state $\mathbf{s}_1(t)$ of the network:

$$\mathbf{o}_3(t) = \mathbf{F}_3\left(\mathbf{i}(t), \mathbf{s}_1(t)\right). \tag{3}$$

Typically, the input consists of a single sequence element:

$$\mathbf{o}_3(t) = \mathbf{F}_3\left(x(t-1), \mathbf{s}_1(t)\right). \tag{4}$$

The next state $\mathbf{s}_1(t)$ is dependent on the input $\mathbf{i}(t-1)$ and on the most recent state $\mathbf{s}_1(t-1)$:

$$\mathbf{s}_1(t) = \mathbf{g}_1\left(\mathbf{i}(t-1), \mathbf{s}_1(t-1)\right). \tag{5}$$

Here $\mathbf{g}_1$ stands for some unknown function. When replacing $\mathbf{s}_1(t)$ in Equation 3 several times, it can be seen that the output $\mathbf{o}_3(t)$ is dependent on several $(n+1)$ inputs $\mathbf{i}(t)$, $\mathbf{i}(t-1)$, ..., $\mathbf{i}(t-n)$:

$$\mathbf{o}_3(t) = \mathbf{g}_2\left(\mathbf{i}(t), \mathbf{i}(t-1), \ldots, \mathbf{i}(t-n), \mathbf{s}_1(t-n)\right), \tag{6}$$

where $\mathbf{g}_2$ is another function which is not further specified. Equations 3 and 5 represent the two mappings which are part of this type of network:

(a) the feedforward mapping of all the available information ($\mathbf{i}(t)$ and $\mathbf{s}_1(t)$) to the output $\mathbf{o}_3(t)$, and

(b) the next-state function modeling the mapping from the past state $\mathbf{s}_1(t$–$1)$ and additional information to the current state $\mathbf{s}_1(t)$.

In addition to the input, two different entities influencing the output can be distinguished:

(a) the weights of the feedforward connections, and

(b) the activations of the state units.

These play different roles. Typically, the unit activations of the state layer change with each time step, while the weights are adjusted at a much lower pace. The weights represent the long-term memory holding all the acquired knowledge. The state is needed for processing the preceding sequence elements. It provides the context for the new input.

3. **Unit Delay without Feedback**

   The units themselves can also have temporal properties. A unit without feedback can be created by delaying information within the unit for a limited number of time steps. However, this approach is not very common.

4. **Unit Delay with Feedback**

   If the activation of a unit is influenced by its own preceding activation, this can be modeled by a feedback loop, which can be referred to as "**self-recurrent feedback loop**." In contrast to layer feedback, one can speak of "**self-recurrent unit feedback**." The self-recurrent feedback loops carry weights like the other connections in the network. A unit with a feedback loop is depicted in Fig. 2. It receives its own preceding output in addition to the input coming from other units.

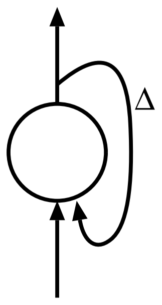

Figure 2: *Unit delay with feedback*

The activation of a unit $z_k(t)$ with feedback (the k-th unit in layer z) may be determined by adding its own preceding activation to the value coming in from another layer (for instance, from the input layer):

$$z_k(t) = \sigma\left(\sum_{j=1}^{J} i_j(t)\, w_{jk}\right) + z_k(t-1). \qquad (7)$$

where the w_{jk} are the weights linking units j and k.

Model Selection: All discussed networks can be used for processing sequences. They take into account the order of the sequence elements, and—if designed appropriately—can deal with long input-output dependencies. However, not all are well suited to coping with dependencies of variable length. They do not automatically treat patterns which are similar but warped in time similarly. Sequence elements arriving earlier or later than usual distract these systems. Such distractions might be due to missing or superfluous sequence elements, or simply due to changes in speed. This problem is called the **temporal invariance problem**.

Since non-recurrent networks parallelize temporal information, they do not automatically generalize to time-warped patterns. Recurrent networks are more flexible concerning the temporal dimension. However, most networks—including simple recurrent networks with layer feedback—are rigid in the sense that the lags between the sequence elements have to be of exactly the same length to be treated similarly. One of the reasons is that the state changes completely with each time step. Since the values returned by the feedback loops differ from the original values, consecutive states are not similar. In other words, states close in time have little in common and thus they cannot be treated similarly.

One solution to the temporal invariance problem is to represent time explicitly. For instance, in [Mannes, 1992], a model based on the ART network processes sequences by treating the order of sequence elements and timing information separately. However, if time should be handled implicitly, another solution has to be found.

Slowly Changing States: The solution to the temporal invariance problem proposed here is to use network states which move slowly through state space. The problem with models which do not produce such **sluggish states** is that the network's walk through state space is not automatically smooth. Subsequent states may lie far apart. This is due to the mapping from one state to the next which does not assure that subsequent states are similar. Thus, subsequent states are related, but not similar to each other. This causes the problem that input, which occurs a single time step earlier or later than usual, leads to a completely different output. This is usually not wanted when the input is sampled at time lags which are independent of the actual sequence. This problem can be solved by forcing the next-state function, which models the relationship between consecutive states, to perform a nearly identical mapping. This aspect is also mentioned in [Jordan, 1986]:

State vectors at nearby points in time must be similar.

The resulting "**continuity property**" is necessary in order to obtain slowly varying states. This requirement is usually not taken into account. The **requirement of state continuity** ensures a slowly changing state. On the other hand, the state must not remain stable, because two consecutive states have to differ with respect to the most recent sequence element. Therefore, consecutive states in a network are forced to be similar, but distinguishable. This way, warped inputs can be handled similarly *or* differently, according to what is required for the given task. Such sluggish states can be obtained by using a **nearly auto-associative next-state function**. This can be achieved, for instance, by inserting **self-recurrent feedback links** from state units to themselves.

Memories modeled with self-recurrent feedback loops represent just one potential approach to obtaining similar subsequent states. Other nearly auto-associative next-state functions may also be applicable. The important point is that the next-state function has to be auto-associative.

The strength of employing sluggish states is that such a model automatically **generalizes to time-warped patterns**. It is also well suited to handling **noisy input**, because its state is quite robust. In other words, the state is not quickly distracted by single outliers. A problem, though, which cannot be solved by such sluggish units, is that recent events have more influence on the state than events in the distant past. The memory decays at a slower rate than when there are no self-recurrent feedback loops, but it still decays. Arbitrary time lags could be bridged if there were a way to obtain temporary variable bindings. This area might be another chance for neural network research to learn from biological models.

Self-recurrent Feedback: Due to additional self-recurrent feedback loops, the state vector activations change more slowly. Thus, time-warped and noisy input is handled automatically. Such a network, with a state wandering slowly through the state space, has the intrinsic capability of handling temporal dependencies in spite of time-shifted and otherwise disrupted input. In the taxonomy presented in [Mozer, 1993], such memory layers with self-recurrent feedback loops are referred to as "exponential trace memories" because they decay exponentially with time. The resulting memory contents represent an exponentially weighted average of the input sequence. The speed of decay is dependent on the weights of the self-recurrent feedback loops.

In contrast to pure copies of layers, such memories with self-recurrent feedback loops are of low resolution. Global aspects can be detected more easily and dependencies spanning long time periods can be recognized. The resulting representation is a **reduced description** [Mozer, 1992] of the sequence. This type of smoothing is also comparable to low-pass filtering and sampling at a lower rate.

The requirement of state continuity is met, for instance, in the network described in [Jordan, 1986]. There, an exponentially weighted average of past network *outputs* is produced. This way, all arbitrarily distant past outputs have some influence on the state. However, the strength of this influence decreases quickly with time. The continuity of the state is achieved by feeding back the activation of each state unit to itself. The previous output unit activations are simply added to these state activations with weights of 1:

$$\mathbf{s}_2(t) = \mu_1 \, \mathbf{s}_2(t-1) + \mathbf{o}_4(t-1), \tag{8}$$

where $\mathbf{o}_4(t-1)$ is the result of mapping the input to the output, the output of the network at time $t-1$, and μ_1 is a real-valued constant ($0 < \mu_1 < 1$) modeling the weight, which modifies the feedback of the state units to themselves. This value can be tuned to make the state more or less flexible. As a result, adjacent states are more or less similar. When setting the first state equal to the initial network output ($\mathbf{s}_2(1) = \mathbf{o}_4(0)$), the state can also be described by the following formula:

$$\mathbf{s}_2(t) = \sum_{r=1}^{t} \mu_1^{r-1} \mathbf{o}_4(t-r). \tag{9}$$

This shows that the current state is a function of all past network outputs. The similarity of adjacent states is the result of averaging past states. It is difficult to determine the appropriate value for μ_1, though. Therefore, [Jordan, 1986] proposes to let the system learn the appropriate "next-state function."

Input State: The novel neural network presented in Section 4 employs self-recurrent feedback for creating the state of the network. This way, the state is directly influenced by the original inputs. The network described in [Jordan, 1986] uses the network outputs to produce the state, but here the inputs are forwarded to the state layer. Another difference is that the incoming values (in this case $\mathbf{i}(t-1)$) are weighted by $1 - \mu_2$, as shown in the following equation:

$$\mathbf{s}_3(t) = \mu_2 \, \mathbf{s}_3(t-1) + (1-\mu_2) \, \mathbf{i}(t-1). \tag{10}$$

This ensures that the sum never exceeds 1 and permits the weights to be interpreted as the degree of influence. They can thus be turned into statements in percent.

4 Tested Neural Network Models

As a result of the discussion in the preceding section, two different network models were selected for being tested in this study:

1. a simple recurrent network with hidden layer feedback, and
2. the input state network.

1. The Simple Recurrent Network with Hidden Layer Feedback

A simple recurrent network with hidden layer feedback, like the one presented in [Elman, 1990], is depicted in Fig. 3. While the dashed arrows denote 1:1-copies of layers, the full arrows represent m:n-connections, where all m units are linked with all n units in the other layer.

The output of such a network is dependent on the input and the preceding hidden layer activations:

$$\mathbf{o}_5(t) = \mathbf{F}_4\left(\mathbf{i}(t), \mathbf{h}_1(t-1)\right). \tag{11}$$

This equation does not show an important property of $\mathbf{h}_1(t-1)$, namely that it typically represents past inputs in a format which is relevant with

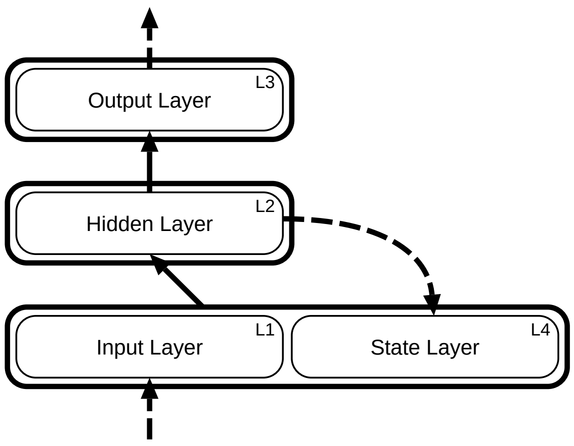

Figure 3: *Recurrent network with hidden layer feedback*

respect to the task of the model. Therefore, hidden layer feedback often leads to good results.

The activation of a unit s_{4_j} is

$$s_{4_j} = (1 - \mu_3) h_{1_j}(t) + \mu_3 s_{4_j}(t-1), \tag{12}$$

where h_{1_j} is a unit j in the hidden layer $h_1(t)$. Written in the recurrent form, the activation of a unit k in this layer is:

$$h_{1_k}(t) = \sigma(\sum_{j=1}^{J} i_j(t)\, v_{jk} + \sum_{l=1}^{L} h_{1_l}(t-1)\, u_{lk}), \tag{13}$$

where the weight v_{jk} leads from the unit j in the input layer to the unit k in the hidden layer, and the weight u_{lk} is the weight of the self-recurrent feedback loop linking unit l and unit k in the hidden layer.

2. The Input State Network

The input state network resembles a typical multi-layer perceptron trained by backpropagation. It also has two layers of weights connecting three layers of units. However, the original input is only used to form the state. Due to the self-recurrent feedback loops, subsequent states tend to be similar. The contents of the state layer deal as input to the conventional multi-layer perceptron. Since the preceding state and the new input are combined to form the state of the network, the first layer of the multi-layer perceptron is called the "state layer," and the network is called the "input state network." The network is depicted in Fig. 4. It is close to the network models described in [Mozer and Soukup, 1991] and [Mozer, 1994], but it differs in several respects. One important difference is that, in these networks, the weights of the connections leading to the state layer are trainable. It is also different from the network presented in [Mozer, 1992], where the units in the *hidden* layer are equipped with temporal properties.

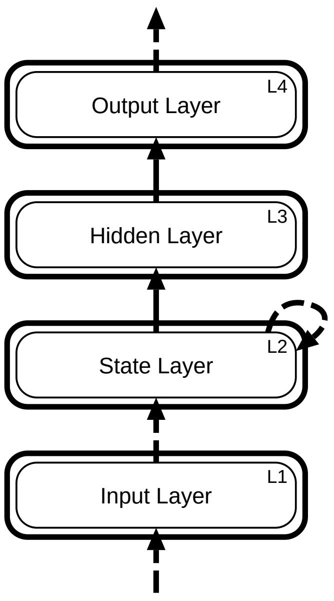

Figure 4: *The input state network*

Like in a network with hidden layer feedback, the new state $\mathbf{s}_5(t)$ is a function of the network input $\mathbf{i}(t)$ and the previous state $\mathbf{s}_5(t-1)$:

$$\mathbf{s}_5(t) = \mathbf{g}_3\left(\mathbf{i}(t), \mathbf{s}_5(t-1)\right), \tag{14}$$

where $\mathbf{g}_3$ is some function which is not further specified at this point. The output $\mathbf{o}_6(t)$ is simply a function of the state:

$$\mathbf{o}_6(t) = \mathbf{F}_5\left(\mathbf{s}_5(t)\right), \tag{15}$$

where $\mathbf{F}_5$ is the mapping produced by a multi-layer perceptron. Recent inputs are implicitly included, because they contribute to $\mathbf{s}_5(t)$:

$$\mathbf{o}_6(t) = \mathbf{F}_5\left(\mathbf{g}_3\left(\mathbf{i}(t), \mathbf{s}_5(t-1)\right)\right). \tag{16}$$

The activation of a unit in the state layer $s_{5_j}(t)$ is determined as follows:

$$s_{5_j}(t) = (1-\mu_{4_j})\ i_j(t) + \mu_{4_j}\ s_{5_j}(t-1), \tag{17}$$

where the $i_j(t)$ are the unit activations in the input layer, and the μ_{4_j} are the weights of the self-recurrent feedback loops of the state units $s_{5_j}(t)$.

State Formation: In the input state network, each single unit in the state layer is given a different **flexibility**. The weights of the self-recurrent feedback loops μ_{4_j} range from zero to close to one, with fewer low and more high weights. Such a distribution is obtained by using the following formula for determining the weight μ_{4_j} of the feedback loop of the j-th unit:

$$\mu_{4_j} = \begin{cases} 0 & \text{if } j = 1 \\ 1 - \sqrt{\frac{j-1}{J}} & \text{otherwise,} \end{cases} \tag{18}$$

where J is the total number of units in the state layer, and j is the number of the unit in the state layer starting with 1. The connections from the input to the state layer are weighted by $(1 - \mu_{4_j})$, so that the resulting activations range again from 0 to 1. This is displayed in Fig. 5 with $J = 20$. The result is a collection of state units of different flexibility, which form a set of short-term memories of different length. This collection of different memories is comparable to that in multi-recurrent networks, which are described in [Ulbricht, 1994]. Due to the flexible unit without any self-recurrent feedback loop (i.e. the first one with a feedback loop of zero weight), current input is quickly recognized. This is important, for instance, for input D, which demands a quick reaction.

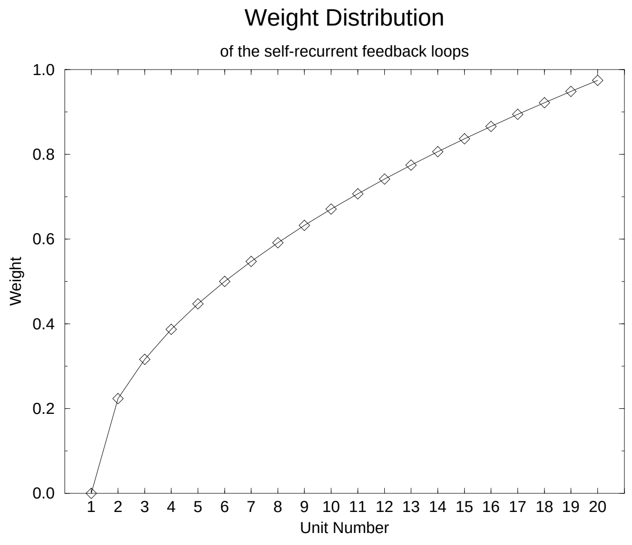

Figure 5: *Weight distribution in the state layer*

Preprocessing: Since each unit $s_j(t)$ in the state layer has different temporal characteristics due to the different weights μ_{4_j} of the self-recurrent feedback loops, it is not reasonable to encode the input using single active units as, for instance, the vector (1000000000) for input A. This way, different sequence elements would be weighted by separate weights. In order to subject different inputs to various types of state units with different weights μ_{4_j}, a distributed input representation is used. It is obtained by mapping the vectors with N single active units $a_n(t)$ via a set of weights b_{nm} and a sigmoid function to the M units $c_m(t)$:

$$c_m(t) = \sigma \sum_{n=1}^{N} a_n(t)\; b_{nm}, \tag{19}$$

where the weights b_{nm} are taken from a randomly selected set of weights. The layers are not fully connected, but only 80% of the connections are actually used. The sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{(-d \cdot x)}}, \tag{20}$$

with a d of 20 is used to obtain the input distribution. Due to the sigmoid function, the activations $c_m(t)$ lie again in the range [0,1].

The employed technique of preprocessing the input provides a method for obtaining a distributed representation. There is evidence that in natural systems, information is also represented in a distributed format. As can, for instance, be shown in experiments with rats, the hippocampus seems to use an "ensemble code" for location information [Wilson and McNaughton, 1994].

Properties: In the input state network, the self-recurrent feedback loops are used to fulfil the continuity requirement described in Section 3. The strength of this network model is that the actual input is delayed and combined with the new input at the next time step. Thus the original input information can be memorized. For the task described here, this solution is appropriate, because there is more information in the original input than in the hidden or output layer activations. This is due to the fact that preceding outputs are not relevant. For numerous other applications, though, self-recurrent feedback alone is too weak. In some cases, additional hidden and output layer feedback may be appropriate, or other techniques like layer delay or layer feedback may be required. Which type of delay is appropriate is dependent on the given task, but keeping some of the original input information is very likely to be useful.

5 Experiment Setup

The two networks presented in the preceding section are used for the experiments. The input state network has 20 input units, 5 hidden units, and 2 output units, plus bias units in the input layer and the hidden layer. The simple recurrent network has the same number of units, plus 5 state units for the past hidden unit activations.

Training: The networks are trained to perform the autonomous agent task described in Section 2. While the mapping functions (Equations 11 and 15) are trained by error-backpropagation, the next-state functions (Equations 12 and 14) are fixed. In order to evaluate the results, the mean square error (MSE) is determined:

$$\text{MSE} = \frac{1}{N}\sum_{n=1}^{N}\left(y_a(n) - y_d(n)\right)^2, \tag{21}$$

where $y_a(n)$ is the n-th network output, and $y_d(n)$ the n-th target output.

The subsequences used for training the agent to select the correct action at the intersection are selected in random order. For testing, a fixed set of 44 subsequences is used. During training, the agent slowly adapts to its environment. The weights of the randomly initialized neural network are adjusted based on the presented sequences of sensory input. This adjustment belongs to the type which is denoted "adaptation by learning" by [McFarland, 1991].

Testing the Generalization Capability: Since the generalization over the temporal dimension is to be investigated, the length of the pause between the relevant input and the required decision is set to a fixed value during training by putting a fixed number of inputs P in a row. For the experiments, the pause length was set to three time steps. In the end, the network generalizes over the temporal dimension *even though* the time span has not been varied during training.

In order to solve this task, the networks have to use information which was obtained several time steps back in time. Such a task can easily be handled with an input window or with time delays of appropriate length, if the length of the time lags does not vary. However, they cannot generalize over the temporal dimension, as required for solving the given task. If the input arrives a little earlier or later—for instance, after 2 or 4 time steps—these networks would not automatically treat the input similarly.

6 Results

Network with Hidden Layer Feedback: The tested networks with hidden layer feedback did not learn to solve the task. The results obtained with one of them are displayed in Figures 6 and 7. The network produced a similar output at all decision points. The activation of one output unit was close to one, and that of the other close to zero. For all tests with varying pause length, the MSE is similar, because 50% of the decisions were correct, and 50% were wrong.

Several variants of networks with hidden layer feedback—including networks with self-recurrent feedback loops in the state layer which holds the preceding hidden layer activations—were trained on the given task, but the result was the same. However, this does not imply that the underlying network model is inappropriate, the failure may be due to the weakness of the training procedure.

Input State Network: The experiment with the input state network was repeated several times. The variance was rather small. A typical result is visualized in Fig. 8. The mean square error is low in the neighborhood of the pause length used during training. In this case, the results for a pause length of 4 and 5 even happen to be slightly lower than for a pause length of 3.

A higher mean square error is not automatically equal to an incorrect output. Therefore, the percentage of correct decisions in several test sets with varied pause length is shown in Fig. 9. For pauses shorter than 2 and longer than 6, the network makes some mistakes, but for a pause length from 2 to 6, the network still comes up with a correct response in all the test problems. These results indicate that the network successfully learned to handle the given task. It is indeed capable of generalizing to time-warped sequential patterns.

7 Conclusion

Neither window networks nor simple recurrent networks with layer feedback are applicable to the given problem which is based on handling time-warped sequences. This is not only due to the network models, but also to the employed training algorithms. Such tasks are common in real-world environments and coping with them is easy for natural animals, but many neural networks cannot deal with them. The experiments have shown how the input state network can be trained to solve such a task. In this model, the original inputs are used to form the state. A single mechanism is employed: self-recurrent unit feedback. The resulting slowly changing states have been shown to enable the animat to deal with time-warped forms of event sequences. This method is well suited to modeling short-term memories of different length within a single system. Moreover, it supports generalization over the temporal dimension, which is crucial for autonomous agents faced with environments containing time-warped sequences of events. The robustness concerning natural temporal variability is an intrinsic property of this model of temporal processing. Even if the network is always trained with a sequence of fixed length, it treats time-warped forms of this sequence similarly. If time-warped versions are part of the training set, this task is even acquired much more easily. To conclude, models employing slowly changing states are useful for handling realistic sequences of sensory events. Thus, they are important for autonomous agents when their tasks involve the treatment of time-warped sequences.

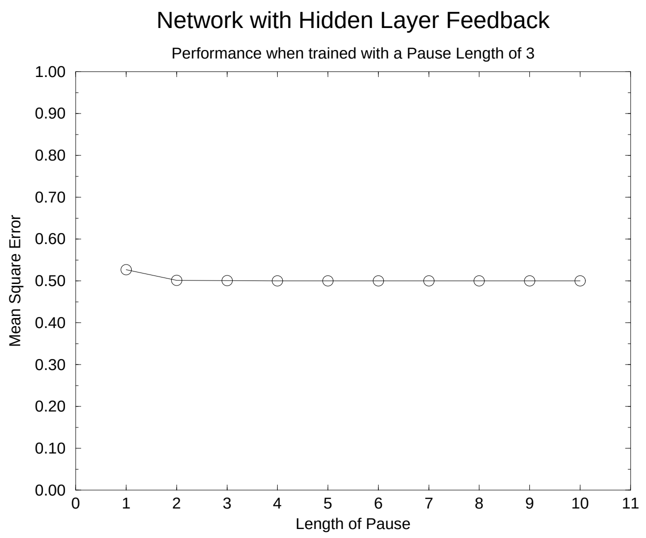

Figure 6: *Error of a network with hidden layer feedback for test sets with different pause length*

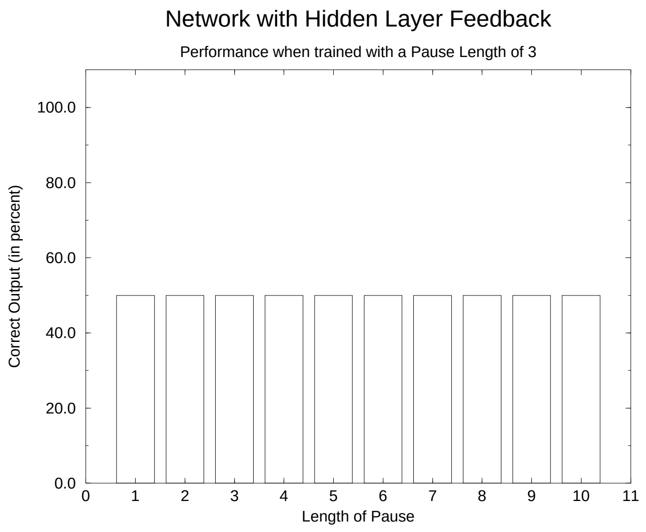

Figure 7: *Performance of a network with hidden layer feedback for test sets with different pause length*

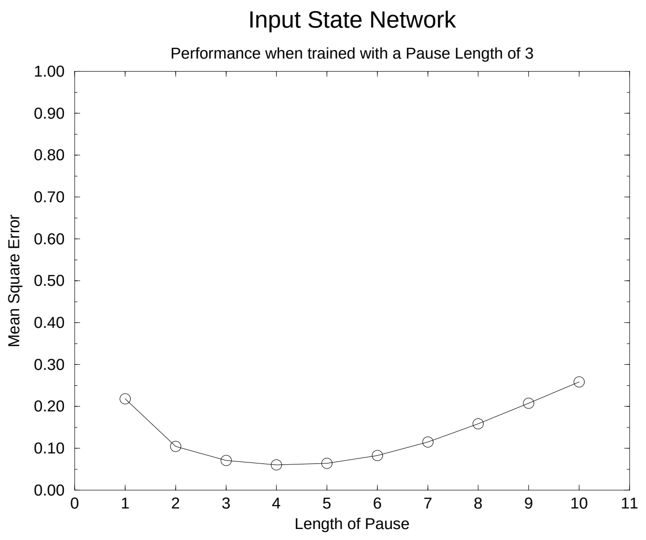

Figure 8: *Error of the input state network for test sets with different pause length*

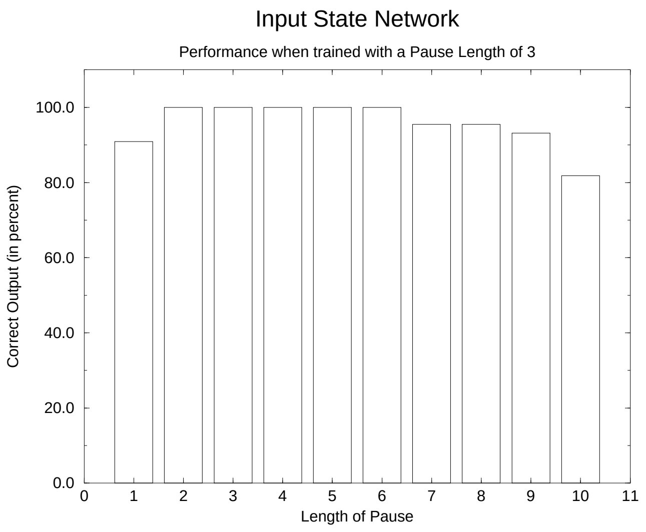

Figure 9: *Performance of the input state network for test sets with different pause length*

Acknowledgements

Thanks for valuable comments are due to Erich Prem and Georg Dorffner. Thanks to Robert Trappl for making this research possible.

The Austrian Research Institute for Artificial Intelligence is supported by the Austrian Federal Ministry of Science, Transport, and the Arts.

References

[Box and Jenkins, 1970] G.E. Box and G.M. Jenkins. *Time Series Analysis.* Holden-Day, San Francisco, 1970.

[Catfolis, 1994] T. Catfolis. Mapping a Complex Temporal Problem into a Combination of Static and Dynamic Neural Networks. *SIGART Bulletin, Vol. 5, No. 3*, 1994.

[Chappelier and Grumbach, 1994] J.-C. Chappelier and A. Grumbach. Time in Neural Networks. *SIGART Bulletin, Vol. 5, No. 3*, pages 3–11, 1994.

[Elman, 1990] J.L. Elman. Finding Structure in Time. *Cognitive Science*, 14:179–211, 1990.

[Hertz *et al.*, 1991] J. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley Publishing Company, 1991.

[Jordan, 1986] M.I. Jordan. Attractor Dynamics and Parallelism in a Connectionist Sequential Machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 531–546. Erlbaum, Hillsdale, NJ, 1986.

[Mannes, 1992] C. Mannes. A Neural Network Model of Spatio-Temporal Pattern Recognition, Recall and Timing. In *Proceedings of the IJCNN International Joint Conference on Neural Networks, Baltimore, IEEE*, pages 109–114, 1992.

[McFarland, 1991] D. McFarland. What It Means For Robot Behaviour To Be Adaptive. In J.-A. Meyer and S.W. Wilson, editors, *From Animals to Animats.* A Bradford Book, MIT Press, Cambridge, MA, 1991.

[Mozer and Soukup, 1991] M.C. Mozer and T. Soukup. Connectionist Music Composition Based on Melodic and Stylistic Constraints. In R.P. Lippmann et al., editors, *Advances in Neural Information Processing 3*, pages 789–796. Morgan Kaufmann, San Mateo, 1991.

[Mozer, 1992] M.C. Mozer. Induction of Multiscale Temporal Structure. In J.E. Moody, editor, *Neural Information Processing Systems 4, Morgan Kaufmann, San Mateo, CA*, 1992.

[Mozer, 1993] M.C. Mozer. Neural Net Architectures for Temporal Sequence Processing. In A. Weigend and N. Gershenfeld, editors, *Predicting the Future and Understanding the Past*. Addison-Wesley Publishing, Redwood City, CA, 1993.

[Mozer, 1994] M.C. Mozer. Neural Network Music Composition by Prediction: Exploring the Benefits of Psychoacoustic Constraints and Multiscale Processing. *Connection Science*, 1994.

[Port and Anderson, 1989] R. Port and S. Anderson. Recognition of Melody Fragments in Continuously Performed Music. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, pages 820–827. Lawrence Erlbaum, Hillsdale, NJ, 1989.

[Rohwer, 1994] R. Rohwer. The Time Dimension of Neural Network Models. *SIGART Bulletin, Vol. 5, No. 3*, pages 36–44, 1994.

[Ulbricht *et al.*, 1992] C. Ulbricht, G. Dorffner, S. Canu, D. Guillemyn, G. Marijuán, J. Olarte, C. Rodriguez, and I. Martin. Mechanisms for Handling Sequences with Neural Networks. In C.H. Dagli et al., editors, *Intelligent Engineering Systems through Artificial Neural Networks, ANNIE'92*, volume 2, pages 273–278. ASME Press, New York, 1992.

[Ulbricht, 1994] C. Ulbricht. Multi-recurrent Networks for Traffic Forecasting. In *Proceedings of the AAAI'94 Conference, Seattle, Washington*, volume II, pages 883–888, 1994.

[Ulbricht, 1996] C. Ulbricht. Handling Time-Warped Sequences with Neural Networks. In *From Animals to Animats 4, Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, September 9th-13th, 1996, Cape Cod, Massachusetts*. Bradford, 1996.

[Wilson and McNaughton, 1994] M.A. Wilson and B.L. McNaughton. Reactivation of Hippocampal Ensemble Memories During Sleep. *Science*, 265, 1994. 29. July 1994.