

# Recurrent Neural Networks for Time Series Classification

Michael Hüsken<sup>a</sup> Peter Stagge<sup>a</sup>

<sup>a</sup>*Institut für Neuroinformatik  
Ruhr-Universität Bochum, 44780 Bochum, Germany  
{huesken,stagge}@neuroinformatik.ruhr-uni-bochum.de*

---

## Abstract

Recurrent neural networks (RNN) are a widely used tool for the prediction of time series. In this paper we use the dynamic behaviour of the RNN to categorize input sequences into different specified classes. These two tasks do not seem to have much in common. However, the prediction task strongly supports the development of a suitable internal structure, representing the main features of the input sequence, to solve the classification problem. Therefore, the speed and success of the training as well as the generalization ability of the trained RNN are significantly improved. The trained RNN provides good classification performance and enables the user to assess efficiently the degree of reliability of the classification result.

*Key words:* recurrent neural network; classification; time series; multitask learning; generalization

---

## 1 Introduction

Recurrent neural networks (RNNs) are dynamical systems that make efficient use of temporal information in the input sequence, both for classification [6,9,17] as well as for prediction [3,13]. This means that after training, interrelations between the current input and internal states are processed to produce the output and to represent the relevant past information in the internal states [5,10,15]. During a supervised learning process the target values constitute a second source of information. These target values specify the relevant interrelations in the input sequence.

For RNNs the input is typically a times series and the target is either a trivial sequence of constant values or another non trivial time series. For classification as carried out in the domain of language learning (i. e., labeling sentences as

grammatically correct or incorrect), the output is a constant class label. For prediction tasks the output consists of another time series (e.g., the shifted input).

These two tasks seem to be rather different as the target data represent different types of information. However, one can argue that those features that are necessary to predict a time series are also related to the features that are required to classify a time series. A good example is given in [2,5], where the learning of a grammar was carried out by predicting the next word of a sentence, although the induction of a grammar is more commonly formulated as a classification task. Analysis of the internal states has revealed that similar characteristic features are represented in both problem settings [5,6]. In the context of multitask learning it has turned out that the more information provided during training, the easier it is to identify the relevant features. The simultaneous consideration of different, but related tasks during the training process can also increase the generalization ability [1].

This leads to the basic assumption for the presented work: the relevant features that have to be represented in the internal states of an RNN for both prediction and classification are strongly related. Therefore, a classification task may be learned more efficiently if during training a prediction task is incorporated in the RNN via additional output units and target sequences. The seemingly unrelated tasks can be beneficially combined during learning because they are different but related as will be shown later.

The remaining paper is organized as follows. In the next section, we illustrate the basic methodology as well as the algorithms of our approach. In section 3, we present experimental results showing the benefit of additional learning tasks for the network training and for the classification ability. Our paper ends with a summary of the main results and concluding remarks.

## 2 The Methodology

The input series and the internal dynamics of an RNN determine the neurons' activations in the RNN. Therefore, after suitable training of the weights, the activations of some output neurons can be used to characterize or classify a presented input sequence. To improve the classification ability in terms of learning speed and generalization, we include a time series prediction task during training. This additional task helps to extract more representative features from the given input data. In this way the auxiliary task serves as an additional source of information for the learning process. We start this section with a brief introduction of the network structure, the error functions and a measure of the classification reliability, which can be computed easily and

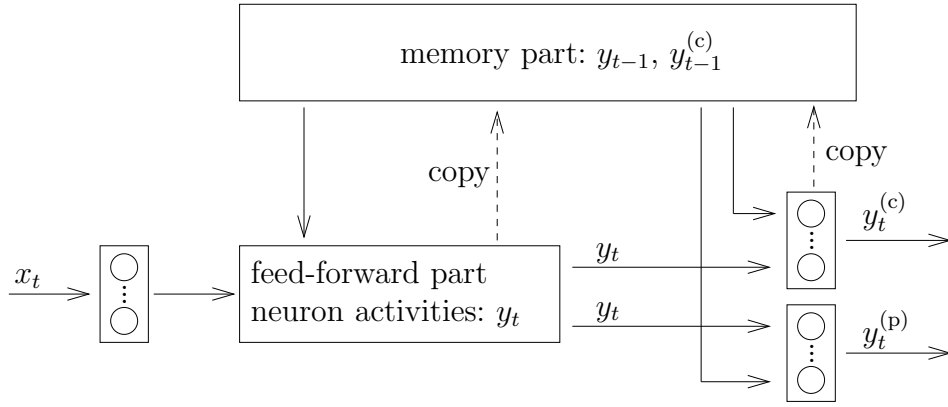


Fig. 1. Structure of the extended Elman-RNNs: the variables  $x_t$ ,  $y_t$ ,  $y_t^{(c)}$ , and  $y_t^{(p)}$  give the activations of the neurons at time step  $t$  (see section 2.2 for further explanations) and the arrows indicate the flow of information inside the network.

quickly in addition to the classification result.

## 2.1 Basic Strategy for Classification

The topology of the RNN we use is an extended Elman-Network [14], as depicted in figure 1. It consists of a feed-forward part and a memory part; the latter one stores the activations of the feed-forward neurons from the previous time step and serves as additional input for the feed-forward part.

To perform the classification, the time series is fed into the RNN and simultaneously the activation of the *classification neurons* is observed. It serves as classification output for which we use the “1-of- $N^{(c)}$ ”-encoding. This means that the number of classification neurons is equal to the number of classes  $N^{(c)}$  and the classification neuron with the highest output value determines the classification result.

The classification of a time series starts with the presentation of a warm-up sequence to the RNN in order to set the internal states correctly. Thereafter, the subsequent time series is fed into the network and the outputs  $y^{(c)}$  of the classification neurons are observed. In every time step, a winner-takes-all of these outputs is performed, i. e., the classification neuron with the highest activation yields the classification result of this single time step. After presenting the whole time series, the rates of wins  $h_i$  of all the classification neurons  $i$  ( $1 \leq i \leq N^{(c)}$ ) during the entire time series are evaluated. The neuron with the highest rate specifies the classification result. The classification error  $E^{(\text{class})}$  is given by the rate of misclassified input sequences in a data set.

The rates  $h_i$  not only yield the classification result but they also provide information about the reliability  $r$  of the classification result. If the values of

$h_i$  are more or less the same for all classification neurons, the classification result is quite questionable. On the other hand, if one value of  $h_i$  dominates over all others, the RNN is quite positive in its decision.<sup>1</sup> This idea can be quantified by measuring the maximum value of  $h_i$ . For normalization this value is mapped into the interval  $[0;1]$  and defined as reliability  $r$ :

$$r^{(\max)} = \frac{1}{N^{(c)} - 1} \left[ N^{(c)} \max_{i=1}^{N^{(c)}} \{h_i\} - 1 \right] \quad . \quad (1)$$

A more sophisticated definition of  $r$ , which also takes the distribution of the smaller rates into account, can be given by means of the entropy

$$H = - \sum_{i=1}^{N^{(c)}} h_i \log h_i \quad . \quad (2)$$

Again, the reliability follows after rescaling:

$$r^{(H)} = 1 - \frac{1}{\log N^{(c)}} H \quad . \quad (3)$$

The definitions (1) and (3) have in common, that  $r$  is close to 0, if all  $h_i$  are rather similar (i. e., the classification neurons present a highly ambiguous result) and that  $r$  becomes 1 in the case that  $h_i$  is equal to 1 for one classification neuron and 0 for all the others.

## 2.2 Training

To yield a proper classification result, the weights of the RNN have to be optimized with respect to some performance measure. Supervised learning (i. e., modifying the connection weights such that the correspondence between the network output and desired target data increases) provides a suitable way of adapting the weights and therefore the dynamic behaviour of the RNN. Instead of optimizing the classification error  $E^{(\text{class})}$  directly, the mean squared error

$$E^{(\text{MSE},c)} = \frac{1}{N \cdot T} \sum_{n=1}^N \sum_{t=1}^T \sum_{i=1}^{N^{(c)}} \left( \hat{y}_{nti}^{(c)} - y_{nti}^{(c)} \right)^2 \quad (4)$$

is minimized. Here,  $N$  denotes the number of time series in the training data set and  $T$  the number of time steps in every time series. The variables  $y_{nti}^{(c)}$  and  $\hat{y}_{nti}^{(c)}$  are defined as the output and target value of the  $i^{\text{th}}$  classification neuron at the  $t^{\text{th}}$  time step of the  $n^{\text{th}}$  time series.

---

<sup>1</sup> In section 3.3 we will see that this is only true for a well-trained RNN.

To perform the classification correctly, suitable dynamics representing the relevant characteristics of the time series has to be built up in the RNN. As indicated in section 1, we assume that the relevant features are comparable for related tasks. Moreover, such an additional task presents an additional source of information to build up a representation containing the main features of the problem at hand [1].

In our experiments the additional information stems from the target values of a prediction task of the time series to be classified. Therefore,  $N^{(p)}$  *prediction neurons* are added as network outputs, where  $N^{(p)}$  is equal to the dimension of the time series. These neurons are integrated into the feed-forward part of the RNN. In our setup their activations are not copied to the memory layer and thus they do not constitute states of the internal dynamics, see figure 1. However, they take part in the learning dynamic and therefore can guide learning.<sup>2</sup> There is no principal obstacle to include them in the memory layer, but we want to ensure that they do not take part in the classification process itself. To include the prediction task into the learning process, the performance measure (4) is modified such that the mean squares error is averaged over both, the classification and prediction neurons:

$$E^{(\text{MSE}, \text{c+p})} = \frac{1}{N \cdot T} \sum_{n=1}^N \sum_{t=1}^T \left[ \sum_{i=1}^{N^{(c)}} (\hat{y}_{nti}^{(c)} - y_{nti}^{(c)})^2 + \sum_{i=1}^{N^{(p)}} (\hat{y}_{nti}^{(p)} - y_{nti}^{(p)})^2 \right] \quad (5)$$

Here,  $y_{nti}^{(p)}$  and  $\hat{y}_{nti}^{(p)}$  denote the output and target values of the prediction neurons. Of course, one can think of weighting the two terms in (5) to tune the influence of the prediction task to the learning process. However, we restrict our investigations to the case of approximately equal influence of the prediction and classification task.

The training of the RNN is performed by means of the iRprop<sup>+</sup>-learning algorithm [7,8]. This method is an improved version of the well performing Rprop algorithm [12], a first order gradient-based learning scheme with adaptive step-sizes. In appendix A we give a short introduction to this algorithm. The gradients are calculated using *Back Propagation Through Time* [4]. However, in our opinion the results in this paper depend only quantitatively on the choice of the learning algorithm, but the main statements will hold for a much wider class of algorithms.

The generalization performance of the trained network is increased by means of cross-validation using two sets of data during training: the *training data*

---

<sup>2</sup> After the learning process these additional neurons and connections can be dropped again. In the case of missing input patterns in the input sequence, these additional elements can also be useful during the classification phase. One can think of using the prediction output in exchange of the missing input pattern to continue the RNN's dynamic.

*set* is used to adjust the connection weights. Those weights that produce the minimum  $E^{(\text{class})}$  on the *validation data set* constitute the final RNN configuration. In case of more than one network with the same classification error, the network with the smallest mean squared error is chosen. We utilize the *test data set*, a third data set that is not used during training, to determine the performance of the trained RNN.

### 3 Experimental Results

In this section, we evaluate the previously described methods by means of a sample classification task. After a short problem description, we present the results of our experiments. We show the strong effect of the additional prediction task on the training process as well as on the generalization ability of the trained RNN. Furthermore, we discuss the quality of the results of the reliability measure  $r$ .

#### 3.1 Trajectory Classification – Problem Description

The time series used in our experiments stem from sampling 2-dimensional ( $N^{(p)} = 2$ ) trajectories of  $N^{(c)} = 3$  different types (classes) [16]:

$$\begin{aligned}
 \text{class 1:} \quad & x_1(t) = \alpha \sin(t + \beta) |\sin(t)| & x_2(t) = \alpha \cos(t + \beta) |\sin(t)| \\
 \text{class 2:} \quad & x_1(t) = \alpha \sin(\frac{1}{2}t + \beta) \sin(\frac{3}{2}t) & x_2(t) = \alpha \cos(t + \beta) \sin(2t) \\
 \text{class 3:} \quad & x_1(t) = \alpha \sin(t + \beta) \sin(2t) & x_2(t) = \alpha \cos(t + \beta) \sin(2t)
 \end{aligned}$$

The value  $\alpha = 0.7$  is chosen to bound the input data  $x_i(t)$  between  $-0.7$  and  $0.7$  and  $\beta$  is randomly drawn for every time series from the interval  $[0, 2\pi)$ . Typical trajectories of the three classes are given in figure 2. To receive the time series, each trajectory is sampled with a step size of  $\Delta t = 2\pi/30$ , starting from a randomly chosen value  $t_0 \in [0, 2\pi)$ .

The training, validation and test data sets consist of  $N = 150$  trajectories, each data set with 50 trajectories of each class. Each trajectory is represented by  $T = 30$  pairs of coordinates. For each time series we use the preceding 30 time steps in the warm up phase to set the internal states of the RNN. To utilize the “1-of- $N^{(c)}$ ”-encoding, the target value of the classification neuron that corresponds to the time series’ class is set to  $\alpha$ ; the target values of the other classification neurons are set to  $-\alpha$ . As we have chosen a one-step prediction as the additional task, the target values of the prediction neurons are given by the coordinates of the trajectory at time  $t + \Delta t$ .

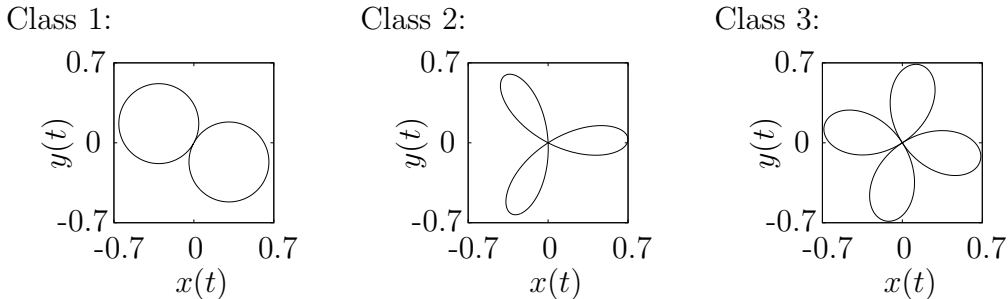


Fig. 2. Examples of members of the three classes. Other trajectories of the classes are given by arbitrary rotations of the depicted curves.

### 3.2 Additional Task Improves Learning

The topology of the RNN is kept fixed in our experiments. The feed-forward part contains 10 hidden neurons that are fully connected among each other. Furthermore, one memory layer with a time delay of one time step exists, which is fully connected to the feed-forward part (see figure 1). The activation function of all neurons is the hyperbolic tangent function. Before learning, all weights are initialized randomly with values from the interval  $[-0.1, 0.1]$ . One hundred independent runs have been performed for both training with and without the additional task.

#### 3.2.1 Speed of Learning

First, we focus on the influence of the additional prediction task on the learning speed. The evolution of the classification error  $E^{(\text{class})}$ , evaluated on the training data set, is given in figure 3 (a). In the runs with additional prediction task, the error decreases significantly faster<sup>3</sup> and the classification error after training is much smaller; on average the error is only one-third of the error after training without prediction task.

Due to the non-monotonic, but fluctuating, behaviour of the training error in every single run, the probability of finding a suitable network during the training process is of much more interest from the practical point of view, than the evolution of the mean training error. In figure 3 (b) the rate for finding a solution with  $E^{(\text{class})} \leq 5\%$ , as a function of the maximum number of learning cycles is given. The success of a run is shown on both training and test data set. In the latter case, the error is calculated on the RNN with the minimum validation error during training (see section 2.2). In 85% of the

<sup>3</sup> It is well known that the magnitude of the derivative of the error with respect to the weights depends on the number of output neurons; therefore the number of output neurons can increase the learning speed. As iRprop<sup>+</sup> only depends on the sign but not on the magnitude of this derivative, this explanation can be excluded.

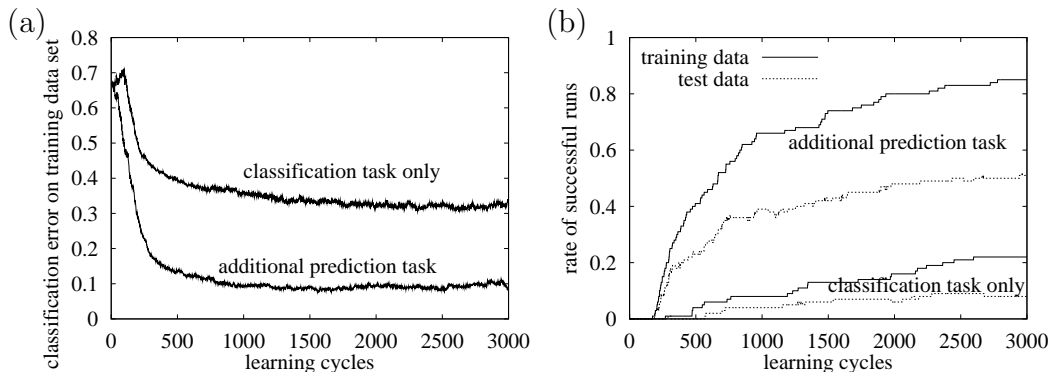


Fig. 3. The additional source of information improves the learning speed and the final error: (a) gives the evolution of  $E^{(\text{class})}$ , evaluated on the training data set and (b) shows the rate of successful runs (i. e., runs in which  $E^{(\text{class})}$ , evaluated on the training and test set, respectively, falls short of 5%).

Table 1

Average classification error  $E^{(\text{class})}$  on the validation and test data set at weight configurations with vanishing training error. The values in parentheses give the standard deviations.

	classification task only	additional prediction task
first occurrence	2.90% (2.34%)	2.32% (1.78%)
all points	2.98% (2.03%)	0.93% (1.29%)

runs that utilize the additional source of information,  $E^{(\text{class})}$  calculated on the training set reaches the 5% threshold and in 51% of the runs the test error reaches this threshold. In both cases this rate is more than four times higher compared to a training process that relies only on the classification task.

### 3.2.2 Generalization

The ability of the trained RNN to generalize, i. e., to correctly classify new and previously unseen time series, is at least as important as a fast training process. To investigate this, the classification error  $E^{(\text{class})}$  on the validation and test set is measured at those points where the classification error on the training set vanishes. Unlike in the last section here we take only the successful runs into account. One can argue that due to the faster learning the RNNs with the additional task give better results at the end of the training. Therefore, we evaluate the generalization error also on the first learning cycle where  $E^{(\text{class})}$  on the training set vanishes. The results are given in table 1.

At the first occurrence of an RNN with vanishing training error, training with the additional task has only slightly and statistically not significantly (Wilcoxon rank sum test:  $p \approx 0.485$ ) improved the generalization ability. In the succeeding learning cycles, training without the additional task yields

no further improvement, as can be seen from a comparison of the first and second row table 1. In contrast to this, continued training with the additional prediction task causes a highly significant (Wilcoxon rank sum test:  $p < 10^{-15}$ ) improvement of the generalization ability. This indicates, that the additional source of information during training causes a continued structuring towards higher generalization.

This result qualitatively coincides with the curves shown in figure 3 (b): the ratio of successful runs in terms of test error and training error is almost twice as high for training with the additional task (the ratio of these two values after the 3000<sup>th</sup> learning cycle is exactly 0.6) compared to training that relies only on the classification information (the ratio after 3000 learning cycles is approximately 0.36). Both results underline the basic assumption that the additional task supports learning but avoids overfitting the training data. It seems, as it supports a more suitable structuring of the RNN, which may coincide with the features known from multitask learning.

Summarizing, the results in this section show, that adding the prediction task during learning yields improvement in two respects: the error drops faster during training and the generalization ability of the found solution is increased. Both observations result in a significant reduction of the learning time necessary to find a suitable solution.

### 3.3 Reliability of classification result

As introduced in section 2.1, the RNNs not only give the class labels, but also an estimation of the degree of reliability of the classification result. Figure 4 shows the evolution of the average reliability during training with the additional prediction task, subdivided into correctly ( $r_+$ ) and incorrectly ( $r_-$ ) classified time series. As the curves of  $r^{(H)}$  and  $r^{(\max)}$  qualitatively behave the same, we will discuss them together. Three phases can be distinguished:

**Initial phase:** The reliability starts close to 0.8, which is mainly influenced by the initial configuration of the network; in most cases the weights are initialized such that the classification neurons yield a constant classification result, not depending on the input sequence. This assumption coincides with the initial classification error of  $E^{(\text{class})} \approx 2/3$ .

**Stagnation:** After a few number of learning cycles  $E^{(\text{MSE}, \text{c+p})}$  reaches a phase of constant error and the reliability drops to a very low level. In this phase, the RNN’s output mainly represents the mean of the target data. Therefore, the classification result is completely random and the average value of  $r$  can be calculated under the assumption of random results of the winner-takes-all-decision. These calculations strengthen the interpretation of this learning

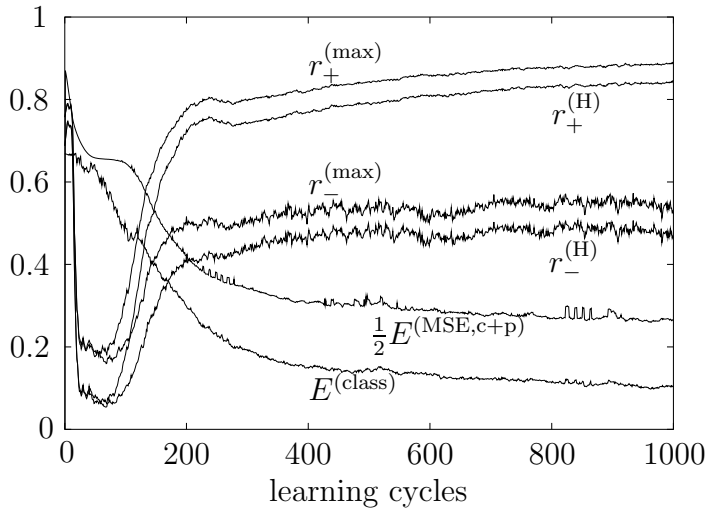


Fig. 4. Evolution of the reliability during training with additional prediction task, averaged over 100 runs. All curves are evaluated on the training data set. Most remarkable is the symmetry-breaking of  $r_+$  and  $r_-$  after about 70 learning cycles. The reliability of correctly classified sequences becomes much higher than the reliability of misclassified ones.

Table 2

Value of  $r^{(H)}$  and  $r^{(\max)}$  on the test data set, depending on the correctness of the classification result. The values are averaged over all networks with  $E^{(\text{class})} \leq 5\%$  on the validation data set. The standard deviation is given in parentheses.

	classification task only	additional prediction task
	correct / incorrect	correct / incorrect
$r^{(H)}$	0.904 (0.183) / 0.521 (0.235)	0.916 (0.182) / 0.487 (0.277)
$r^{(\max)}$	0.935 (0.196) / 0.552 (0.258)	0.943 (0.141) / 0.534 (0.278)

phase.

**Symmetry-breaking:** After about 70 learning cycles, the error strongly drops and the reliability  $r$  increases. Moreover, the symmetry of  $r_+$  and  $r_-$  breaks: the reliability of correct classifications increases much faster and stabilizes on a much higher level than the reliability of incorrect classification results.

Again we analyze closer the successful runs, i. e., RNNs yielding a validation error below 5% during learning. Table 2 compiles their reliability for classifying the patterns of the test data set. This result underlines the previously mentioned symmetry-breaking of the reliability for correctly and incorrectly classified inputs. This statement holds irrespectively of the definition of  $r$  ( $r^{(H)}$  or  $r^{(\max)}$ ) and of the choice of the training process (with or without prediction task). It should be mentioned, that the splitting is slightly larger for training with additional prediction task, which might be related to the higher generalization performance.

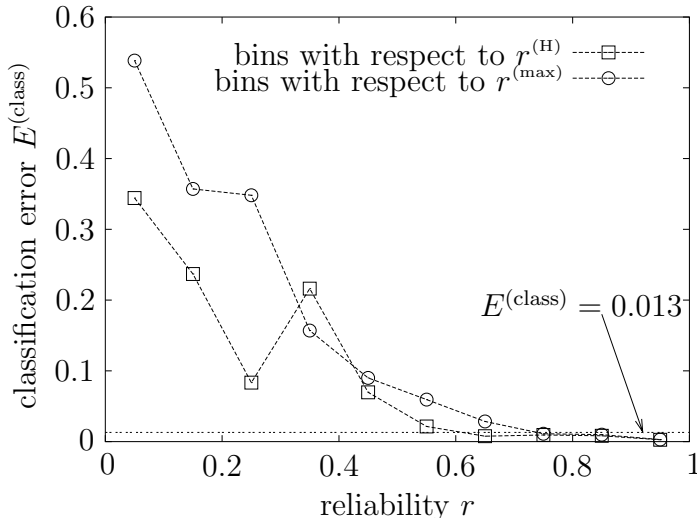


Fig. 5. Dependency of the classification error on the reliability. Only the networks with a validation error  $E^{(\text{class})} \leq 5\%$  are used for this investigation; the average classification error of these networks is  $E^{(\text{class})} = 0.013$ . The classification results of the training, validation and test data set are collected in bins of size 0.1 according to the reliability  $r^{(\text{max})}$  and  $r^{(\text{H})}$ . All classification results in one bin are averaged.

The previous results show, that misclassification coincides with a low reliability and vice versa. To make this clear figure 5 shows the classification error as a function of the reliability  $r$ . For very small values of the reliability the classification performance is not much better than a random labeling of the input sequences.<sup>4</sup> On the other hand,  $< 1\%$  of the input sequences are misclassified, if the reliability is  $> 0.7$ . As the reliability of the results is not equally distributed, this holds for more than 90% of the input sequences. The corresponding classification error is lower than the average error ( $E^{(\text{class})} = 1.3\%$ ) of the investigated RNNs. This reliability-threshold provides the possibility of rejecting or verifying ambiguous classification results. In the context of a combination of classifiers this can lead to an increase of the overall performance.

## 4 Discussion and Conclusion

We have shown how to use the dynamics of recurrent neural networks (RNNs) for the classification of time series. The internal dynamics were organized such that the output of the classification neurons represent the class of the time series fed into the network. The main issues of this article are the improvement of the learning process by an auxiliary training task and the additional utilization of the network's output for an estimation of the degree of reliability

<sup>4</sup> Actually, the classification results may only seem to be better than 66.7%, as they are averaged over reliabilities from intervals of size 0.1.

of the classification result.

The class labels of the training data can be used to organize the supervised training of the RNN. However, the probability of finding a suitable solution within a given amount of time is quite low. This situation changes significantly by teaching simultaneously an additional but related task. This additional task can help to organize the RNN such that the dynamics of the internal states represent important features of the task to solve. In our experiments the inclusion of a prediction task during learning strongly supports the learning process as well as the generalization ability of the resulting networks. This result indicates that the prediction and classification tasks rely on related information in the input sequences and that the inclusion of both tasks makes it easier to find a solution during the learning process. After training, the RNN provides a high classification rate.

Additionally, the system's output can be used to assess the reliability of the time series classification. We propose two measures for the reliability which provide an equally good basis to identify misleading classification results. The reliability measure allows the identification and possibly rejection of ambiguous classification results, which can be beneficially exploited in a multi-classifier-environment.

## A The iRprop<sup>+</sup>-Learning-Algorithm

The learning algorithm iRprop<sup>+</sup> [7,8] is an improved version of the well performing Rprop-algorithm [11,12], a gradient-based learning method with adaptive step size control. This algorithm has turned out to be faster and more robust in terms of the choice of its internal parameters compared to other learning algorithms.

Let  $w_{ij}$  be the weight of the connection from neuron  $j$  to neuron  $i$  and  $\Delta_{ij}$  be the step size of this weight, i. e., the amount of change of the weight  $w_{ij}$  in one learning cycle. Prior to the first learning cycle the step sizes are initialized equal to  $\Delta_0$ . The evolution of the step sizes as well as of the direction of the steps depends on the error  $E$  and the *sign* of the partial derivative of the error function  $E$  with respect to the weights  $w_{ij}$ ;  $E^{(t)}$  and  $\frac{\partial E}{\partial w_{ij}}^{(t)}$  denote the values of the error and the derivative at time step  $t$ . The step sizes are bounded by  $\Delta_{\min}$  and  $\Delta_{\max}$  and the adaptation velocity of the step sizes is determined by the parameters  $\eta^+$  and  $\eta^-$ . Algorithm 1 gives one learning cycle of iRprop<sup>+</sup> in pseudo-code. In our experiments, we used standard values for the five parameters:  $\Delta_0 = 0.01$ ,  $\Delta_{\min} = 0$ ,  $\Delta_{\max} = 50$ ,  $\eta^+ = 1.2$ , and  $\eta^- = 0.5$ .

```

for each  $w_{ij}$  do
  if  $\frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} > 0$  then
     $\Delta_{ij}^{(t)} := \min(\Delta_{ij}^{(t-1)} \cdot \eta^+, \Delta_{\max})$ 
     $\Delta w_{ij}^{(t)} := -\text{sign}\left(\frac{\partial E}{\partial w_{ij}}^{(t)}\right) \cdot \Delta_{ij}^{(t)}$ 
     $w_{ij}^{(t+1)} := w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$ 
  elseif  $\frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} < 0$  then
     $\Delta_{ij}^{(t)} := \max(\Delta_{ij}^{(t-1)} \cdot \eta^-, \Delta_{\min})$ 
    if  $E^{(t)} > E^{(t-1)}$  then  $w_{ij}^{(t+1)} := w_{ij}^{(t)} - \Delta w_{ij}^{(t-1)}$ 
     $\frac{\partial E}{\partial w_{ij}}^{(t)} := 0$ 
  elseif  $\frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} = 0$  then
     $\Delta w_{ij}^{(t)} := -\text{sign}\left(\frac{\partial E}{\partial w_{ij}}^{(t)}\right) \cdot \Delta_{ij}^{(t)}$ 
     $w_{ij}^{(t+1)} := w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$ 
  endif

```

Algorithm 1: Pseudo-code of one learning cycle of the iRprop<sup>+</sup>-algorithm.

## Acknowledgements

We would like to thank the two anonymous reviewers for their beneficial remarks and comments. For financial support we would like to thank the DFG, grant SFB 475, and the BMBF, grant LEONET, grant number 01 IB 802 C4.

## References

- [1] R. Caruana. Multitask learning. *Machine Learning*, 28:41–75, 1997.
- [2] A. Cleeremans, D. Servan-Schreiber, and J. L. McClelland. Finite state automata and simple recurrent networks. *Neural Computation*, 1:372–381, 1989.
- [3] J. T. Connor, D. Martin, and L. E. Atlas. Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks*, 5(2):240–254, 1994.
- [4] K. Doya. Recurrent networks: Supervised learning. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 796–800. The MIT Press, Cambridge, MA, 1998.

- [5] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.
- [6] C. L. Giles, C. B. Miller, D. Chen, G. Z. Sun, H. H. Chen, and Y. C. Lee. Extracting and learning an unknown grammar with recurrent neural networks. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 317–324, San Mateo, CA, 1992. Morgan Kaufmann Publishers.
- [7] C. Igel and M. Hüsken. Improving the Rprop learning algorithm. In H.-H. Bothe and R. Rojas, editors, *Proceedings of the Second International ICSC Symposium on Neural Computation (NC 2000)*, pages 115–121, Canada / Switzerland, 2000. ICSC Academic Press.
- [8] C. Igel and M. Hüsken. Empirical evaluation of the improved Rprop learning algorithm. *Neurocomputing*, 50(C):105–123, 2003.
- [9] S. Lawrence, C. L. Giles, and S. Fong. Natural language grammatical inference with recurrent neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 12(1):126–140, 2000.
- [10] C. W. Omlin and C. L. Giles. Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41–52, 1996.
- [11] M. Riedmiller. Advanced supervised learning in multi-layer perceptrons – from backpropagation to adaptive learning algorithms. *International Journal of Computer Standards and Interfaces*, 16(5):265–278, 1994.
- [12] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 586–591, New York, 1993. IEEE Press.
- [13] E. W. Saad, D. V. Prokhorov, and D. C. Wunsch. Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks. *IEEE Transactions on Neural Networks*, 9(6):1456–1470, 1998.
- [14] P. Stagge and B. Sendhoff. An extended Elman net for modeling time series. In W. Gerstner, A. Germond, M. Hasler, and J. Nicoud, editors, *Artificial Neural Networks (ICANN 97)*, volume 1327 of *Lecture Notes in Computer Science*, pages 427–432, Berlin, 1997. Springer Verlag.
- [15] P. Stagge and B. Sendhoff. Organisation of past states in recurrent neural networks: Implicit embedding. In M. Mohammadian, editor, *Computational Intelligence for Modelling, Control & Automation*, pages 21–27, Amsterdam, 1999. IOS Press.
- [16] G.-Z. Sun, H.-H. Chen, and Y.-C. Lee. Green’s function methods for fast on-line learning algorithm of recurrent neural networks. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing 4*, pages 333–340, San Mateo, CA, 1992. Morgan Kaufmann Publishers.

- [17] R. I. Watrous and G. M. Kuhn. Induction of finite-state automata using second-order recurrent neural networks. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 317–324, San Mateo, CA, 1992. Morgan Kaufmann Publishers.