Prediction of Dynamical Systems by Recurrent Neural Networks

Peter Trebatický*
Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Ilkovičova 3, Bratislava, Slovakia
trebaticky@fiit.stuba.sk

Abstract

Recurrent neural networks in general achieve better results in prediction of time series then feedforward networks. Echo state neural networks seem to be one alternative to them. I have shown on the task of text correction, that they achieve slightly better results compared to already known method based on Markov model. The major part of this work is focused on alternatives to recurrent neural networks training that are based on Kalman filtration modifications. I describe in detail the training by filters: Extended Kalman Filter, Unscented Kalman Filter (UKF), nprKF Filter and their joint versions UKFj and nprKFj. Contribution of this work is presentation of simpler equations for individual filters, because they are modified specifically for recurrent neural network training. Filter UKFj in context of recurrent neural networks was probably firstly described in my work. I compare individual filters with each other and also with gradient descent method Truncated Backpropagation Through Time (BPTT(h)). I show the results are consistently better when comparing recurrent neural networks trained by these advanced methods with BPTT(h). In the like manner, Extended Kalman Filter achieves worse results compared to the other filters, which on the other hand achieve comparable results with each other. I describe how to speed up their computation by utilizing the graphics card. My work is one of the first (if not the first) that focuses on recurrent neural network training utilizing the processor on graphics card. This paper represents my dissertation summary.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: LearningNeural nets; I.3 [**Computer Graphics**]: Miscellaneous

*Recommended by thesis supervisor: Prof. Jiří Pospíchal Defended at Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava on June 25, 2009.

Keywords

dynamical systems, recurrent neural networks, Kalman filter, prediction

1. Introduction

The ability to predict the behavior of a system is in general dependent on the knowledge of principles describing given occurence. If we are able to express those principles as a set of equations we can predict the behavior in the past for a given starting conditions. This is often impossible or will be very time- / money-wise consuming. In principle, we are able to predict dynamic system behavior in the future using observed data from the past. Many approaches exist how to achieve this.

In my dissertation I introduce the reader into basics of the terms: dynamical system, prediction, artificial recurrent neural networks. I especially address the description of the types of recurrent neural networks and methods for their training, and thus also how to achieve prediction using them. I describe in detail the Echo state neural networks and methods of neural networks training based on Kalman filtration. I mention in it the objectives of my dissertation that are in a large extent related to the field of prediction of dynamical systems by recurrent neural networks. I describe the methods that I chose in order to accomplish the given objectives and I show the results of realized experiments.

1.1 Dynamical Systems

Dynamical system in general consists of the set of states and of the relation that determines the next state based on past states of the system [11, 12, 2]. State is a vector which consists of every relevant variable of the system in such a way that the state vectors provide enough information to predict future states. The important fact is that the system's states are not measured directly. We observe only the output of the system which is determined by the current state, so it is possible to estimate the current state.

Dynamical systems could be described either in continuous or discrete time. We will consider only the discrete time dynamical systems in the following. Dynamical system is discrete when it works in discrete, usually equally spaced time intervals. It does not automatically mean that its outputs are discrete values. System's state usually changes non-linearly by time-variant function and the state transition can also be corrupted by noise. Similarly,

the measurement (output vector) from the system is in general non-linear, dependent on the current state according to non-linear time-variant function and can also be corrupted by noise.

The measurements from the dynamical system form a time series. That is why we can use the methods for time series prediction also for dynamical systems prediction.

1.2 Prediction

The aim of the time series prediction is, in short, to determine the continuation $x_{N+1}, x_{N+2}, \ldots$ on the basis of the known sequence $x_1, x_2, \ldots, x_N$ up to time N [15]. The sequence might be created e.g. by measuring the outputs of dynamical system working in discrete time, or by sampling the system working in continous time and can have either stochastic or deterministic origin. The standard approach to prediction stems in the effort to create a model that describes (generates) the observed sequence.

There exists a broad spectrum of classical methods for prediction, e.g. extrapolation of trend curve, exponential smoothing, Holt-Winters technique, Box-Jenkins methodology, general exponential smoothing; ARARMA, where the trend is filtered out by autoregressive model before application of ARMA model, etc. The mentioned methods are very useful when applicable. But their usage is limited by the need of many practical experiences, and with every one separately, which is very demanding, especially because each one comprises of different mathematical equations and thus different parameters.

Apart from the classical methods there exists so called structural modelling [4]. They have many things in common, but they differ in that the classical time series prediction tries to describe the observed data, as opposed to the structural modelling where the aim is to model the hidden dynamics that produced the observed data. That is why the classical methods are more suitable for short-term future data prediction, whereas the structural modelling provides long-term dynamical behavior [11].

The time series generated by non-linear dynamical systems are common in practice, that is why the non-linear models which are able to capture their dynamics are sought-after. It was shown that the feedforward neural networks with finite number of neurons is universal approximator [3]. These arguments provide the basic motivation to make use of neural networks for time series prediction, which is not a new idea.

1.3 Recurrent Neural Network

If the output of the network is not determined solely by the current network's input, but also by the history of inputs, it is necessary to seek such a structure of the network that will be able to create the state representation of the *time* context in data. It turns out the recurrent neural networks have this capability. It is possible to use also *Time delay neural network* – TDNN, but its capabilities are limited due to the fixed size of the time window. Recurrent network is any network which have certain subset of neurons, called recurrent, that store information about its activities from previous time steps. Values of the outputs of recurrent neurons from previous step are copied into so called context neurons and are appended to the current input vector. Neural network is thus augmented with *internal memory*.

Recurrent neural networks, in contrast to the classical feedforward neural networks, better handle inputs that have space-time structure, e.g. symbolic time series. It is possible to use them for example to process the sequences of words of languages generated by grammars, or sequences having chaotic character, etc., which is very useful e.g. in control of robotic systems.

Similarly as it is with feedforward neural networks, there are no connections within the same layer (technically, the connections are allowed but with nonzero delay). There exist various neural networks architectures, within which the Elman's is the most used one – it has one hidden layer which is recurrent.

Recurrent neural networks are trained for the next desired vector prediction for example by using *Real-time recurrent learning* method – RTRL [9]. Its essence is in the back propagation of errors where not only space but also time structure of data is reflected.

Another algorithm for recurrent neural network training is *Backpropagation through time* – BPTT [9]. It serves mainly for training to classify the sequences. The desired output is provided to the network at the end of each sequence. The neural network is unfolded in time, so it has as many hidden layers as is the number of inputs in one sequence. It is then trained the same as feedforward neural network with n hidden neurons, where n is the sequence length. The drawback of these *gradient descent* methods is for example the tendency to get stuck in some local minimum.

2. Thesis Objectives

It is possible to predict the dynamical systems using various approaches. In my research I have focused on the field of recurrent neural networks that in general achieve better results in prediction of time series then feedforward networks, because it is necessary to take also time context into account. Recurrent neural networks are still quite rarely used what is probably given by the fact that their training converges slowly to satisfactory results when we use classical gradient descent methods. That is why one of my work's objectives is to focus on the alternatives of gradient descent methods.

2.1 Echo State Networks Adaptation

One of this alternatives shows to be Echo state neural networks. Artificial neural networks with echo states represent a novel view on the recurrent neural network training. The basic idea is to use a large reservoir of *untrained* recurrently connected neurons that serves as a source of interesting signals, from which the desired output is composed. That is why I want to focus on researching of their capabilities in several experiments.

2.2 Kalman Filter Modifications

The more traditional alternative to gradient descent methods are methods of *training* of recurrent neural networks based on Kalman filtration. My objective is to research what results are achieved by recurrent neural networks trained by these advanced methods on several tasks concerning the next symbol prediction in the given sequence.

I plan to compare them with one another as well as with gradient descent methods especially from the point of

achieved prediction quality, what is of course the most important. The time aspect is also interesting, that is how long the training lasts from the point of overall duration, because these methods have various asymptotical complexity, as well as from the point of needed training cycles.

2.3 Graphics Processor Utilization

Recurrent neural network training is computationally intensive process. It is especially true for advanced methods using Kalman filtration, which in general achieve better prediction properties, but their time demands for discovering suitable parameter values and final training deter from more frequent usage. That is why I focused on researching the possibility to speed up the recurrent neural networks training, what will improve their practical application.

Recently, the suitable computational device starts to be a "common" graphics card that is in common personal computers. This device can nowadays be used as a coprocessor, because it is high-performance parallel computational device. That is why I want to focus on showing the suitability of graphics processor utilization for the training of various types of networks and algorithms for recurrent neural networks. I will emphasize the objective comparison of speed with the computation solely on classical processor. I plan to achieve this by using equivalent libraries and algorithms for graphics as well as classical processors. This will show speed up also from the practical view, that is by using comparable effort for implementing given algorithm.

The important criterion is that the proposed algorithms shall be sufficiently universal and usable also in the future, i.e. they shall not be tied to the particular graphics processor. As the manufacturers of graphics cards themselves support general purpose computations on them, especially through creation of libraries for them, it is a valid assumption that the hardware structure of these cards will remain nearly identical (they are called stream processors), not to mention the existing programming interfaces.

3. Kalman Filter Modifications

Recurrent neural network learning is a very difficult numerical problem which approaches very poorly and slowly to satisfactory results when being solved with the classic gradient optimisation methods on longer input sequences. That is the reason for searching for the more effective methods of recurrent network learning. The methods based on Kalman filtration serve as a better alternative to classic gradient descent methods.

The Kalman filtering problem lies in jointly solving the process and measurement equations of *linear* dynamical system for the unknown state in an optimal manner [8]. Kalman filter functions in two repeating steps (after initialisation)

1. *Time update*, also called prediction step – we compute the apriori estimation of the state and the error covariance.

2. *Measurement update*, also called correction step – we correct the state estimate obtained in the previous step according to the new measurement.

This concept of prediction–correction is a basic principle of Kalman filtration and is present in each type of filter mentioned further.

3.1 Extended Kalman Filter

When the model is *nonlinear*, which is the case of neural networks, we have to extend Kalman filter using linearisation procedure. Resulting filter is then called the Extended Kalman filter (EKF) [2].

Neural network is a nonlinear dynamical system that can be described by equations [2, 11]:

$$\mathbf{x}_{k-1} = \mathbf{x}_k + \mathbf{r}_k \quad (1)$$

$$\mathbf{y}_k = h(\mathbf{x}_k, \mathbf{u}_k, \mathbf{v}_{k-1}) + \mathbf{q}_k \quad (2)$$

The process equation expresses the state of neural network as a stationary process corrupted with the process noise $\mathbf{r}_k$, where the state of the network $\mathbf{x}$ consists of network weights. Measurement equation expresses the desired output of the network as a nonlinear function of the input vector $\mathbf{u}_k$, of the weight vector $\mathbf{x}_k$ and for recurrent networks also of the activations of recurrent neurons from previous step $\mathbf{v}_{k-1}$. This equation is augmented by the random measurement noise $\mathbf{q}_k$. The covariance matrix of the noise $\mathbf{r}_k$ is $\mathbf{R}_k = E[\mathbf{r}_k \mathbf{r}_k^T]$ and the covariance of the noise $\mathbf{q}_k$ is $\mathbf{Q}_k = E[\mathbf{q}_k \mathbf{q}_k^T]$.

The basic idea of the Extended Kalman filter lies in the linearisation of the measurement equation at each time step around the newest state estimate $\hat{\mathbf{x}}_k$. We use for this purpose just the first-order Taylor approximation of nonlinear equation [2].

It follows that the Extended Kalman filter can be directly used for prediction of non-linear dynamical system. The problem is that it is necessary to know the process as well as measurement equations, which is not always possible. But because we know these equations for recurrent neural network, it is beneficial to use them as a model of nonlinear dynamical system, which contains many parameters (connection weights). Extended Kalman filter estimates these parameters based on measurements.

We can express the neural network training as a problem of finding the state estimate $\mathbf{x}_k$ that minimalises the least-squares error, using all the previous measurements. We can express the solution of this problem as:

$$\mathbf{K}_k = \mathbf{P}_k \mathbf{H}_k^T [\mathbf{H}_k \mathbf{P}_k \mathbf{H}_k^T + \mathbf{R}_k]^{-1} \quad (3)$$

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_k + \mathbf{K}_k [\mathbf{y}_k - h(\hat{\mathbf{x}}_k, \mathbf{u}_k, \mathbf{v}_{k-1})] \quad (4)$$

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \mathbf{K}_k \mathbf{H}_k \mathbf{P}_k + \mathbf{Q}_k \quad (5)$$

where $\hat{\mathbf{x}}$ is a vector of all the weights, $h(\cdot)$ is a function returning a vector of actual outputs, $\mathbf{y}$ is a vector of desired outputs, $\mathbf{K}$ is the so called Kalman gain matrix, $\mathbf{P}$ is the error covariance matrix of the state and $\mathbf{H}$ is the measurement matrix (Jacobian). Matrix $\mathbf{H}$ contains partial derivatives of i's output with respect to j's weight. Its computation is realised either using the Backpropagation algorithm in forward neural network; or using either Backpropagation through time, Truncated backpropagation through time, or Real time recurrent learning in recurrent network.

3.2 Unscented Kalman Filter

The key element in the EKF method is the development of the covariance matrix, which is in every step appro-

ximated using the Taylor expansion of nonlinear function, which relates the network outputs to the weights. The first-order Taylor linearization provides an insufficiently accurate representation in many cases, and significant bias, or even convergence problems are commonly encountered due to the overly crude approximation [10].

The nonlinear generalisation of the Kalman filter called the Unscented Kalman filter (UKF) was also used with success in various applications including the recurrent neural network training [13]. The basic principle of the UKF is conceptually similar to the EKF, but the implementation is significantly different. No derivatives are needed, only function evaluations, as opposed to the Taylor approximation. Of essence is a fact that this method generally achieves better results than EKF.

The basic difference between the EKF and UKF stems from the manner in which Gaussian random variables (GRV) are represented for propagation through system dynamics. In the EKF, the state distribution is approximated by GRV, which are then propagated analytically through the first-order Taylor approximation of the nonlinear system. As already mentioned, this can introduce large errors in the true posterior mean and covariance of the transformed GRV.

The UKF uses so called *unscented transformation*, which is a relatively new method for calculating the statistics of a random variable which undergoes a nonlinear transformation [7]. A set of *sigma points* is chosen so that their sample mean and sample covariance are the true mean and true covariance, respectively. When propagated through the *true* nonlinear system, they capture the posterior mean and covariance accurately to the second order of Taylor series expansion for *any* nonlinearity. The EKF, in contrast, only achieves first-order accuracy.

This method resembles the Monte Carlo-type methods, where many samples are randomly chosen, which also are propagated through nonlinear transformation. On the other hand, the sigma points are not chosen randomly, but deterministically and moreover, the number of sigma points is low – $2n_x + 1$, where n_x is the dimension of random variable (vector's dimension) [7].

In [1] is an important statement about recurrent neural network training by nprKF, but it is valid for the UKF as well (which I can confirm on the basis of own experiments). It is necessary to repropagate the *recurrent* neural network several steps from the past with the new weights, which is similar to the derivatives computation with the BPTT(h). For that we obviously have to store the network inputs and the outputs of at least recurrent neurons from previous steps. The purpose of this is to adapt the recurrent neurons activations to a new weight vector.

3.3 Filter nprKF

The UKF uses sigma points and unscented transformation for the computation of statistics of a random variable that went through nonlinear transformation. The EKF uses the first-order Taylor expansion. The other method, or the other Kalman filter modification is the filter named nprKF [13].

As with the UKF, also with this filter it is not necessary to calculate derivatives, since the Taylor expansion is in this case replaced by the Stirling's formula for nonlinear function approximation at the interval, where the function f is approximated by the second-order terms [10]. This formula may be interpreted as a Taylor approximation with the derivatives replaced by central *divided differences* (hence the original filter's name – DD2). They derived this filter's equations for a general dynamical system, when the nonlinearity occurs in the state equation as well as in the measurement equation and even either process or measurement noise may impact nonlinearly.

Even though the nprKF authors do not explicitly mention the sigma points, they in fact also use them and even the same number of them ($2n_x + 1$). The analysis in [10] confirms this, since the state estimate is identical in both filters. But they differ in the covariance estimation. The nprKF provides slightly better estimate – the differences in covariance update in comparison with the UKF are in the fourth- and greater-order of the Taylor expansion. Covariance estimate with the UKF is therefore slightly less accurate and may sometimes even lead to the non-positive definite estimate [10]. As with the UKF, we need to repropagate the network several steps for every weight vector variation (sigma point).

3.4 Joint filters UKF and nprKF

As mentioned previously, when training recurrent neural networks, it is necessary to repropagate the network several steps from the past for every sigma point (i.e. with different weights).

The alternative approach lies in including the recurrent neurons activations in the state vector, when we simultaneously use the UKF or nprKF for weights estimation as well as the state estimation [13]. It is therefore necessary for thus formulated problem, which is called the Joint Unscented Kalman Filter (*UKFj*) or Joint nprKFj, respectively, to change the state equation and hence the UKF/nprKF equations. It is worth noting that this name is used also for the filter in which the system's state and inputs are concatenated into single state vector [2, 22]. This is used in the situation when we do not have the noise-free input into a system.

I have not found the UKFj filter in the literature specifically in the context of *recurrent* neural network training. I have created it on the basis of similarity with the nprKFj filter described in [13].

The joint filters on one hand increase the computation complexity by expanding the state vector $\mathbf{x}$, on the other hand it simplifies the computation of the network output for various weight vectors (sigma points). The difference in comparison with the basic versions of the filters lies only in the time update, since the function f has become nonlinear by including additional states. Specifically for the Elman's architecture it is therefore sufficient to set (apart from the weights) the hidden recurrent neurons outputs and then just to compute the outputs of neurons in the output layer. This interpretation is consistent with the usual measurement equation in dynamical systems, when the system output is a function of its state variables [13].

4. Neural Networks on GPU

Graphics processing units (GPU) in common graphics cards have recently evolved into the powerful resource for general purpose computing. They were originally used so-

lely for speed up of mathematical computation involved in graphics. New specialized algorithms were gradually added in order to achieve better visual effects, but because of growing complexity of these algorithms the possibility to program GPUs was eventually added.

Thus, the possibility to program them was originally motivated to achieve faster and more realistic graphics rendering. However, the computation power of this commonly available device was still frequently used for applications that had nothing to do with graphics. We can therefore say the graphics card can be used as a coprocessor to the classic central processing unit (CPU).

I show in this section how it is possible to speed up the computations in recurrent neural networks utilizing commonly available graphics cards. What is important, the proposed algorithms shall be sufficiently universal and they shall not be tied to the particular hardware. As both main manufacturers of graphics cards nVidia and ATI are using the term stream processor besides the term graphics card, it is evident they are built on the same principle. While this principle remains unchanged the described methods stay valid.

What is more, if we constrain ourselves to work only with matrices/vectors, then the used function calls will remain unchanged, because they adhere to the Basic Linear Algebra Subprograms (BLAS) standard [25]. So, if the principle of graphics cards for some reason changes, this interface will be preserved.

4.1 Forward Signal Propagation

It is possible to speed up also the forward propagation of signal through neural network on the graphics processor. There are two options how to approach this problem.

If the network is organized into layers, then we can express the input propagation through network as a gradual propagation of input through each layer. Utilization of matrix operations is beneficial especially because the libraries for graphics cards exist specifically for matrix operations. I have used CUBLAS library which is a part of CUDA [17].

In order to better utilize parallel processor in graphics card, it is beneficial to propagate several inputs at once through a layer in non-recurrent networks. We concatenate respective input vectors into the matrix $\mathbf{U}$ and compute output matrix $\mathbf{Y}$ ($\mathbf{W}_1$ is weight vector):

$$\mathbf{W}_1\mathbf{U} = \mathbf{A} \quad (6)$$

$$\mathbf{Y}_1 = f(\mathbf{A}) \quad (7)$$

It is possible to use parallel evaluation of elements in matrix $\mathbf{A}$ by application of (sigmoidal) function f to every matrix element and obtain thus matrix $\mathbf{Y}$. If the network has more layers, the similar computation is applied to the other layer, just the output from previous layer augmented with a row of ones (i.e. biases) will serve as input for next layer.

We can visualize neural network as a system of interconnected nodes, i.e. as a graph, where the oriented edge may exist between any node couple. In other words, neural network does not have to be organized into layers. And if the graph contains a cycle then it is recurrent network.

In this case we can modify a method for simulation of gravitational interaction of N bodies [24]. In this simulation, every body influences every other body through its gravitational force, whereas the resulting gravitational force affecting the body is a sum of individual forces. Here is the similarity with neural networks, because, in general, the neuron activation is a sum of outputs of every other neuron multiplied by the connection weight.

We will simulate network operation in time steps, where in each time step the input vector is presented to the network and new activations of neurons are computed using the neuron outputs from previous step. For the network with k layers, we obtain the network output in step $t+k$ given the input in step t.

4.2 Echo State Networks

Recurrent neural networks with echo states (ESN) represent a novel view on the recurrent neural network training. The basic idea is to use a large reservoir of recurrently connected neurons that serves as a source of interesting signals from which the desired output is composed. What differs the Echo-state networks from the classical training techniques is the fact that the weights of connections in the reservoir remain constant during the training process. Only the output weights are adapted what corresponds with the desired output "composition" [5]. This approach is closely related to the so-called architectural bias [14], when the untrained networks have also useful properties.

It is necessary to carry out several steps when training these networks.

The main part of Echo state networks is dynamical reservoir that typically contains hundreds, even thousands of neurons. As the neurons in dynamical reservoir are randomly interconnected, in principle, every neuron is connected with every other neuron, we obtain the response of neurons to input signal by using simulation described in the previous section.

When we have outputs of neurons in every time step, it is necessary to compute output weights. Linear combination of signals from reservoir in such a way as to be as close to the desired output as possible, can be obtained by this computation [6]:

$$\mathbf{R} = \mathbf{X}^T\mathbf{X} \quad (8)$$

$$\mathbf{P} = \mathbf{X}^T\mathbf{Y} \quad (9)$$

$$\mathbf{W}^{out} = \mathbf{R}^{-1}\mathbf{P} \quad (10)$$

where $\mathbf{X}$ is matrix with signals from reservoir and $\mathbf{Y}$ is matrix of desired outputs.

We can use any method to compute linear regression ("fitting" of signals from reservoir onto desired outputs). I chose this one, because inverse of matrix $\mathbf{R}$ can be obtained using Cholesky decomposition, computation of which was already studied on graphics processor [23].

5. Results

In this section, I show and comment the results of experiments that I conducted using described methods.

5.1 Text Correction Using ESN

I have used the Echo state network to correct corrupted symbol sequence in this experiment. The principle is to

create Probabilistic finite automaton (PST) from the states of dynamical reservoir, because then we are able to apply Viterbi's algorithm for text correction. The task of repairing corrupted text (sequence of symbols) means to find the most likely state sequence that generated the original text [18].

I have chosen as a training sequence the books of King James Bible, excluding the Genesis book which served as test sequence. This choice was inspired by the article [18]. The length of test sequence was $1.9 \cdot 10^5$ and the length of training sequence was $2.9 \cdot 10^6$. Alphabet includes 26 characters and blank space.

In order to test the prediction capabilities of Variable length Markov model, with which I compared results, I have created the Prediction suffix tree containing 1790 nodes. I have transformed it into probabilistic finite automaton with 13911 states.

I have used reservoir with 100 neurons for the Echo state network. 20% of all the possible connections were initialized to uniformly random floating point numbers from the interval $(-1, 1)$, the other connections had zero weights. I have scaled the matrix of internal connections to have spectral radius close to 1, specifically 0.998. The reason is to achieve longer short-term memory. Input weights were chosen randomly and uniformly from the interval of integers $\langle -5, 5 \rangle$.

During the creation of automaton from ESN I took every 50th state of reservoir as reactions to input training sequence and I created 10000 clusters by K-means method from thus obtained 58056 states. Each cluster corresponds to one state of automaton [19]. The next step is to find the probabilities of transition between these states. These are estimated based on relative frequency of transitions from the origin state's cluster i into the target state's cluster j. We sequentially present the training sequence as an input to the network, we monitor the states of reservoir and determine each state's cluster (i.e. which cluster's center has the shortest Euclid's distance to the state).

The final step is to determine the next symbol probability distribution for each automaton's state (cluster). This can be achieved by presenting the training sequence and observing which symbol caused the transition into the reservoir's state that corresponds to given automaton's state (cluster). We obtain thus for each automaton's state the relative frequencies of observed symbols. In majority of cases the single symbol has the markedly highest relative frequency, so it is sufficient to remember just this one symbol for each autoamton's state [20]. Now we can apply Viterbi's algorithm.

The first 1000 symbols from the testing set (book Genesis) was changed to another symbol with the probability of 0.1, i.e. was corrupted. I noted how many from the corrupted symbols were successfully corrected by Viterbi's algorithm, but also how many corrupted symbols were in the end, because this algorithm can "correct" also original symbols. That is caused by the fact that we have only a model of the "process" which generated the text of the Bible and it is therefore possible the different sequence seems more probable than the correct one.

I conducted this experiment 100 times – that is always different around 10% from 1000 symbols were corrupted and I computed the averages. Variable length Markov model in average corrected 81.16 symbols from 100.12 which is 81.1% and there were 29.38 still corrupted symbols. Automaton created from the ESN in average corrected 84.32 symbols from 99.84 which is 84.5% and there were 23.6 still corrupted symbols.

The example of text correction using ESN is in Table 1. It is an extract of 1000 symbols from the book Genesis which was used as a testing set. We can see the successful correction of heavilly corrupted text (e.g. the first two words), the unsuccessful correction (e.g. word "appear"), as well as corruption of correct word ("dry").

I have compared two different approaches to the text correction – Variable length Markov model and Echo state networks. Echo state networks achieved slightly better performance, especially from the point of lower number of corrupted symbols in resulting text. The main disadvantage of Echo state networks in this experiment is the computational complexity of K-means. It is possible to use more effective methods of vector quantization than K-means, or to revise the Viterbi algorithm idea in such a way that it will not be neccessary to explicitly create probabilistic automaton. It is worth noting that after creation of probabilistic automaton from ESN the text correction itself takes less time than by the automaton created from PST, because it has lower number of states.

5.2 Measuring predictive performance

The next symbol prediction procedure in general is the following: we present in every time step the first symbol in order, and the desired network's output is the next symbol in sequence order. I chose the Elman's network architecture in the experiments – i.e. the network with one hidden layer, which is recurrent.

The predictive performance was evaluated by means of a normalized negative log-likelihood (NNL), calculated over symbolic sequence from time step $t = 1$ to T [21]:

$$NNL = -\frac{1}{T} \sum_{t=1}^{T} \log_{|A|} p^{(t)}(s^{(t)}) \tag{11}$$

where the base of the logarithm $|A|$ is the number of symbols in the alphabet A and $p^{(t)}(s^{(t)})$ is the probability of predicting symbol $s^{(t)}$ in the time step t. If $NNL = 0$, then the network predicts next symbol with 100%, while $NNL \geq 1$ corresponds to a very inaccurate prediction (random guessing).

This measure could also be interpreted as how well is the given sequence compressible. Lower values of NNL mean the given sequence can be compressed better, on the other hand higher values represent worse compressible sequence.

5.3 Kalman Filter Modifications

I describe in this section one of the experiments that focused on the comparison of the performance of all the Kalman filter modifications, as well as the BPTT(h), in the Elman's recurrent neural network training task. The other experiments described in my dissertation confirm the conclusions from this experiment.

The training sequence in this experiment was generated by the Reber automaton, where the next symbol was cho-

Table 1: Example of text correction using ESN – differences are underlined.

Original text	...gathered together unto one place and let the dry land appear and it was so and god called the dry land earth...
Corrupted text	...gadhetedqqogethpr unno jne p_ace and eet jhe drf land appeam and it was so nnd god _alled the dry land earth...
Corrected text	...gathered together unto the place and let the day land a_peac and it was so and god called the day land earth...

sen with 50% probability from two candidate symbols (except when the last symbol follows). The sequence has length of one million symbols, but for our needs are sufficient the first 50000 symbols. Reber's grammar contains 6 symbols, which we encode using the *one-hot encoding* and we present them to the network as its input. This encoding means that a single input is reserved for each symbol, which we set to 1 and others to 0. The network output is similarly one-hot encoded. The advantage is that we are able to easily compute the NNL with this setup – it is sufficient to normalize the output vector (i.e. the sum of its elements equals 1) and the prediction probability of the desired symbol is then immediately the value of that part of the vector which is reserved for given symbol.

The prediction of symbols generated by the Reber automaton is a relatively simple task, since we are able to tell in which state we are solely from the last two symbols. The problem therefore lies mostly in the "lucky" prediction of one out of two possibilities, i.e. the neural network after training ought to be able to predict with which probability each symbol follows. This also results in the fact that we should not expect the NNL to be close to zero, quite contrary, it should certainly be less than 0.5, since we are not certain which symbol follows when there are two options.

The recurrent neural network with three hidden neurons, what is sufficient for this particular problem, was trained with each method. The training process was carried out by presenting the symbols to the network, while the weight update was performed in every time step. The NNL was computed for every 1000 symbols. I present the comparison of all the filters by this indicator in Figure 1. The graph also contains the curve representing the "ideal" NNL. We obtain it when we predict the next symbol with the probability precisely 0.5, except for the cases when symbol with the probability 1 follows.

We can clearly see the performance of each method (in the sense of convergence and final result). The weakest method has shown to be the BPTT(h) as expected. The Extended Kalman filter converges more quickly and the final result is better by approximately 0.01 in comparison with the BPTT(h). The dominance of the filters UKF, UKFj, nprKF and nprKFj is evident. The convergence of all these four filters is rapid as opposed to the BPTT(h), and very quick when compared with the EKF. The differences are minimal – it is impossible to choose the "winner" among them.

In further experiments with EKF and BPTT(h) I tried several number of hidden neurons and various values for parameter h. They showed that none of the changes to these parameters signifficantly altered the results and the EKF is consistently better than BPTT(h).

5.4 Extended Kalman Filter on GPU

The experiments showed the better performance of Kalman filter based methods as opposed to the classic gradient descent method. Better performance in a sense the lower number of training cycles are needed and especially in a sense of better predictive results, where the gradient descent method did not converge to such good results. The disadvantage is the greater asymptotic computational complexity, so the training cycle performed by them lasts longer. It is worth noting that after training is done, i.e. when the network is ready for usage, the evaluation of the network is then independent from the training method used, that is lasts the same amount of time. But during the phase of searching for suitable parameters the speed up of the training is important and allows to try several possibilities which adds to the overall quality of the resulting solution.

In this experiment I focused on as objective comparison of various implementations of the Extended Kalman filter as possible – implemented on graphics processor and implemented using automatically tuned library for the particular processor (CPU) – ATLAS [16]. This library is capable to utilize the multicore processor in order to eventually speed up the computation by parallel processing. ATLAS library was configured to support threading on CPU. The number of maximum threads was chosen to be 4 which is the number of cores on our test machine. I have used for this experiment processor Intel Core2 Quad CPU Q6600 2.4 GHz and graphics card nVidia GeForce 8800 GTX having 128 parallel processors.

As the majority of graphics processors still uses single precision floating point numbers, I focused on the impact of this aspect as well. I compared the speed with the implementation on processor using single precision floating point representation, and I also evaluated what impact has the potential loss of precision when compared with double precision on achieved quality of trained neural network.

We can express the equations of the Extended Kalman filter for one step of training for recurrent neural network as follows:

$$\mathbf{K}_k = \mathbf{P}_k\mathbf{H}_k^T[\mathbf{H}_k\mathbf{P}_k\mathbf{H}_k^T + \mathbf{R}_k]^{-1} \quad (12)$$

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_k + \mathbf{K}_k[\mathbf{y}_k - g(\hat{\mathbf{x}}_k, \mathbf{u}_k, \mathbf{v}_{k-1})] \quad (13)$$

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \mathbf{K}_k\mathbf{H}_k\mathbf{P}_k + \mathbf{Q}_k \quad (14)$$

where $\hat{\mathbf{x}}$ is a vector of all the weights, $g(\cdot)$ is a function returning a vector of actual outputs, $\mathbf{y}$ is a vector of desired outputs, $\mathbf{K}$ is the so called Kalman gain matrix, $\mathbf{P}$ is the error covariance matrix of the state and $\mathbf{H}$ is the measurement matrix (Jacobian). Matrix $\mathbf{H}$ contains partial derivatives of ith output with respect to jth weight. We used Truncated backpropagation through time for this purpose implemented on CPU.

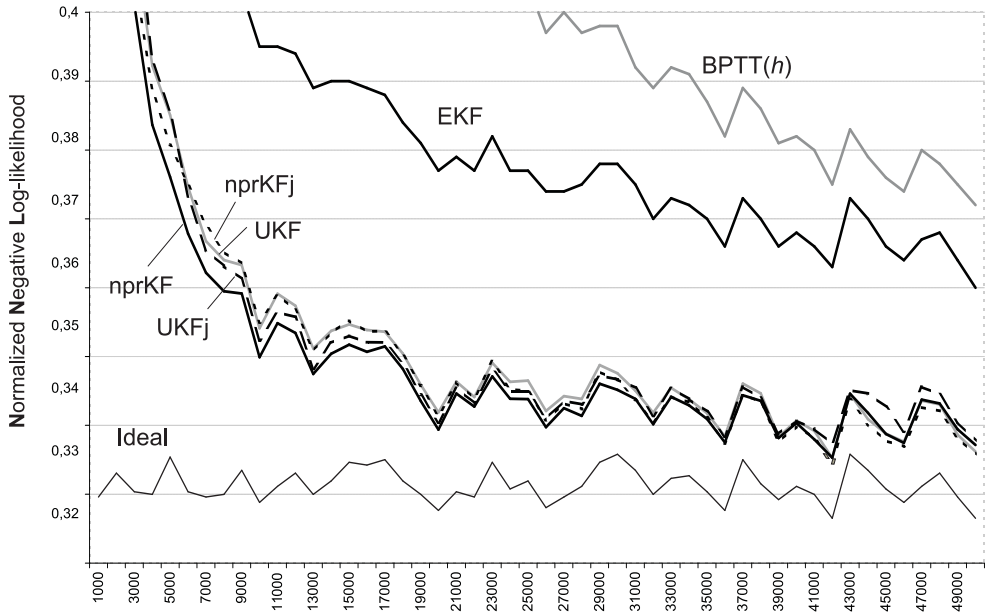

Figure 1: The NNL dependence of the next symbol prediction in sequence generated by Reber grammar on the number of symbols for various recurrent neural network training methods. The ideal value of NNL for this experiment is also displayed.

These equations contain only vector and matrix operations and represent the most computationally intensive part of the training. That is why I used the functions for matrix operations provided by the libraries CUBLAS for graphics processors and ATLAS for processor. The only thing that is not readily available is matrix inversion. It is a symmetric positive definite matrix, that is why I chose to use Cholesky factorization to compute its inverse. Cholesky factorization on GPU has already been studied [23], but because it is usually small matrix (its dimensions are equal to the number of network outputs) we can simply compute it on the CPU. This requires an additional transfer of data between GPU and CPU which is a costly operation, but we already have to transfer much bigger measurement matrix $\mathbf{H}$ and weight vector $\mathbf{x}$, so this should not have a big impact.

We will use the following abbreviations of corresponding implementations in this section:

EKF GPU EKF implemented using CUBLAS library

EKF ATLASf EKF implemented using ATLAS library with single precision functions and utilizing single thread

EKF ATLASft EKF implemented using ATLAS library with single precision functions and utilizing four threads

EKF ATLASd EKF implemented using ATLAS library with double precision functions and utilizing single thread

EKF ATLASdt EKF implemented using ATLAS library with double precision functions and utilizing four threads

We trained the Elman's recurrent neural network on the next symbol prediction. The predicted sequence is based on real data obtained by quantization of activity changes of laser in chaotic regime [21]. This sequence contains four symbols and its length is 10000. Training is performed on the first 8000 symbols, testing on remaining 2000.

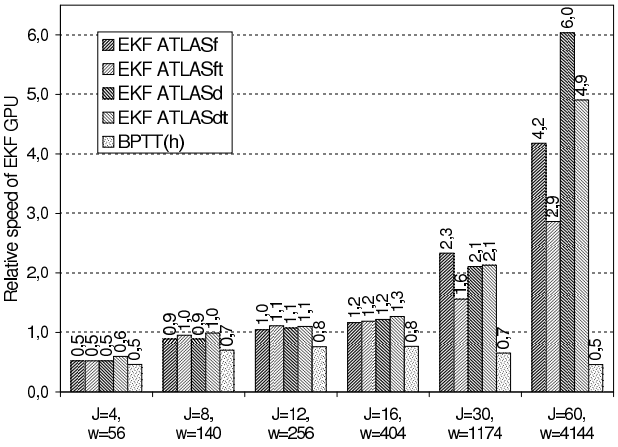

Figure 2: Speed comparison of EKF GPU relative to other implementations. J is number of hidden neurons and w the number of weights in recurrent neural network. The speedup is significant for networks with many weights even when compared to the threaded CPU version with single floating point precision. On the other hand it is not beneficial for small networks. Gradient descent method BPTT(h) is still faster but converges slowly and achieves worse results

We have performed 20 training cycles in order to compare the precision of various implementations in longer run, even though the Extended Kalman filter in this case achieves the best result on training set around the fifth cycle already.

I present the results for the elapsed time during training of neural networks with various numbers of hidden neurons by each method relative to EKF GPU in Fig. 2. From these results we can see that the implementation of EKF on GPU provides significant speedup for larger networks, but is not beneficial for smaller networks. This stems from

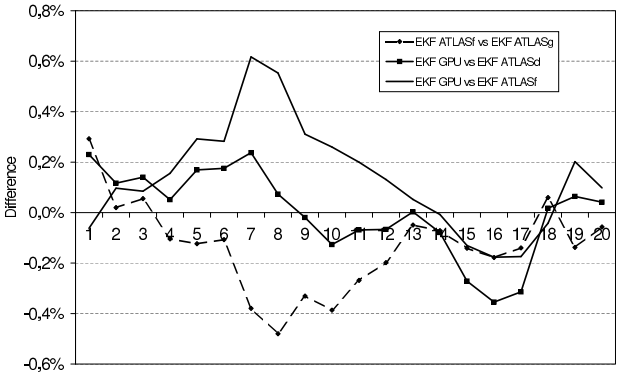

Figure 4: Difference between achieved results of used implementations of EKF in every training cycle as a percentage of valid NNL interval length (i.e. 1). Positive values mean better achievement of first implementation than second implementation. The differences are negligible, all within 0.6%**.**

the fact, that the overhead of copying data from system memory to GPU and of parallelization alone is not worth when there is not much to compute.

The most "fair" comparison of EKF GPU is with EKF ATLASft, because both utilize parallelization and work with single precision floating point arithmetic. EKF GPU achieves nearly 3 times speedup for largest network when compared to EKF ATLASft. The most significant acceleration by using computation with EKF GPU – 6 times on largest network – is achieved when compared to EKF ATLASd, which is probably most similar to existing implementations of EKF.

The achieved speedup of EKF GPU made it also more feasible to conduct thorough experiments with larger networks, as seen in Fig. 3. This also justifies the increasing of number of hidden neurons for this problem, as the best achieved result is by networks with 60 hidden neurons (NNL=0.1156 in training cycle 4).

In Fig. 2 we can further see that method BPTT(h) is consistently the fastest, whereas EKF ATLASd is generally the slowest one. The obvious question is if the achieved results are on the one hand worth the speed degradation when compared with BPTT(h) and on the other hand worth the significant speedup when compared with potentially more precise EKF ATLASd. The answer is in Fig. 3 which for each method shows the average results when used for training 10 randomly initialized networks. In this graph we can see the superior convergence and achieved result of EKF when compared with BPTT(h). We can also see the comparable results of various implementations of EKF, which is more obvious from Fig. 4. It means the used floating point arithmetic does not play a significant role in EKF performance in this experiment.

I experienced the numerical stability problem in my experiments when computing Cholesky factorization of matrix which became non positive definite. This was remedied by restarting the training process with different initial weights. Since the problem usually arose in the early training cycles, and the training is fast, the restarting did not cause significant delays.

6. Conclusion

In my research I have focused on the prediction of dynamical systems by recurrent neural networks. I have shown the suitability of usage of these networks trained by various methods in several experiments with symbolic sequences. However, all the methods, except text correction, are not limited to symbolic sequences.

The contributions of my thesis can be summarized in the following points:

- I have shown as first the possibility to use the Echo state neural networks on the task of text correction. This idea emerged naturally from the mutual paper [20], where we researched this problem for untrained networks. Echo state networks are also untrained, that is why the application of the same approaches on them was relatively straightforward. The method is more easily implementable than Markov models, with which I compared achieved results, but is more computationally demanding and the correction is not significantly better.

- Another contribution of this work is presentation of simpler equations for individual Kalman filters, because they are modified specifically for recurrent neural network training. The modification of equations is possible, because the dynamics of neural network can be expressed by less complex equations. Filters EKF, UKF and nprKF are expressed mostly for general dynamical system in the literature. That means the process and measurement noises can influence state vector, input vector, even time index and also non-linearly, e.g. by multiplication. On one hand it shows the broad applicability of individual filters, on the other hand the equations describing their function are complicated and are more numerous. It is caused by the fact that here are intertwined the fields of automation, artificial intelligence and mathematics, especially statistics.

  I have also described in detail, implemented and tested the effectiveness of relatively new filters Unscented Kalman filter (UKF), nprKF and their joint versions – UKFj and nprKFj. There is no need to compute the derivation when working with these filters, as opposed to the Extended Kalman filter (EKF). Beside the fact that the implementation is thus simplified, they provide more precise system's state estimate.

- Filter UKFj in context of recurrent neural networks training was probably firstly described in my work [26]. I claim it on the basis that I have not found the UKFj filter in the literature specifically in the context of *recurrent* neural network training. I have created it on the basis of similarity with the nprKFj filter described in [13].

- Another contribution of my thesis is direct comparison of all the filters particularly for the recurrent neural networks training and compared them with classic gradient descent algorithm BPTT(h). The results of the experiments with the task of next symbol predicion showed significantly better performance of filters UKF, nprKF, UKFj a nprKFj (which achieve comparable results with each other) when comparing with EKF. Significantly better results achieves also the EKF in comparison with BPTT(h).

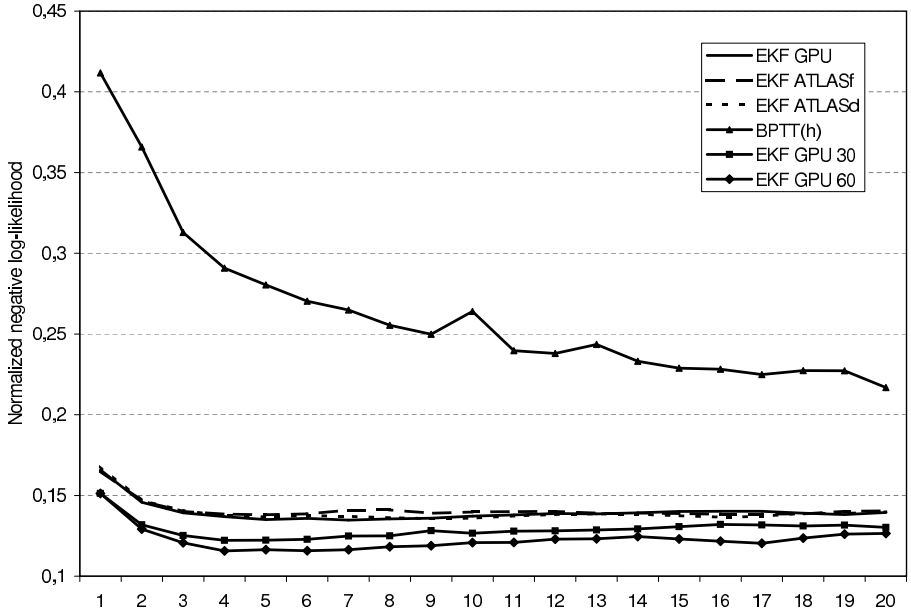

Figure 3: The NNL dependence on the training cycles for various implementations and methods used for recurrent network with 16 hidden neurons. The exception is *EKF GPU 30* and *EKF GPU 60* which is a result for 30 and 60 hidden neurons respectively. The results show that (a) EKF is superior to gradient descent method BPTT(h); (b) the various EKF implementations achieve comparable results (see Fig. 4); (c) the increasing of hidden neurons is beneficial for this problem and was made more feasible by achieved speedup.

It follows that methods based on Kalman filtration really are the better alternative to gradient descent methods in a sense of better achieved results [26, 27].

- Another contribution is description of my approach how to speed up computation utilizing graphics card. Even though the methods based on Kalman filtration achieve better results in comparison with gradient descent methods, they have greater asymptotical (regrettably, as well as non-asymptotical) complexity. However, even any multiple of speed boost of computation is beneficial in practical application. My work [27] is one of the first (if not the first) that copes with the *recurrent* neural networks training using processor on graphics card.

  The main idea why to use this particular computational device is that the graphics card which allows general purpose computing (i.e. not related to graphics) is in every newer computer, including portable ones. It thus provides speed up practically free of charge. That is in contrast with computations utilizing special coprocessors, or grid.

  Important fact is that my methods of implementation for graphical processor are not dependent on some particular hardware. They rely only on the preservation of the basic principle of current graphics cards. Such a change is not expected in near future, because they are based on stream processing of data. What is more, if we constrain ourselves to work only with matrices/vectors, then the used function calls will remain unchanged, because they adhere to the BLAS standard.

We can summarize that the application of my proposed methods can at least in mentioned tasks significantly improve or speed up the prediction.

Acknowledgements. This work was supported by Scientific Grant Agency of Slovak Republic under grants #1/0804/08 and #1/4053/07.

References

[1] Lee A. Feldkamp, Timothy M. Feldkamp, Danil V. Prokhorov. Neural Network Training with the nprKF. In *Proceedings of International Joint Conference on Neural Networks*, 1, pages 109–114, 2001.

[2] S. Haykin editor. *Kalman Filtering and Neural Networks*. New York: John Wiley & Sons, Inc., ISBN: 0-471-36998-5, 2002.

[3] S. Hornik, M. Stinchombe, H. White. Multilayer feedforward networks are universal approximators. In *Neural Networks*, 2, pages 359–366, 1989.

[4] C. Chatfield. *The Analysis of Time Series. An Introduction*. 6th edition, New York: Chapman & Hall/CRC, ISBN: 1-584-88317-0, 2003.

[5] H. Jaeger. Short Term Memory in Echo State Networks. *Technical report 152*, GMD – Forschungszentrum Informationstechnik GmbH, 2002.

[6] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. *Technical report 148*, GMD – German National Research Institute for Computer Science, 2001.

[7] Simon J. Julier, Jeffrey K. Uhlmann. A New Extension of the Kalman Filter to Nonlinear Systems. In *Proceedings of AeroSence: The 11th International Symposium on Aerospace/Defence Sensing, Simulation and Controls*, Mar. 2000.

[8] R. E. Kalman. New Approach to Linear Filtering and Prediction Problems. In *A Transactions of the ASME, Series D, Journal of Basic Engineering*, 82, pagesn35–45, 1960.

[9] P. Návrat et al. *Artificial Intelligence*. STU Press, Bratislava, 2001.(in Slovak)

[10] M. Nørgaard, N. K. Poulsen, O. Ravn. Advances in Derivative-Free State Estimation for Nonlinear Systems. *Technical report IMM-REP-1998-15*. Revised edition. Technical University of Denmark, 2000.

[11] G. S. Patel. Modeling Nonlinear Dynamics with Extended Kalman Filter Trained Recurrent Multilayer Perceptrons. *Diploma project*. McMaster University, 2000.

[12] Dynamical system. *Wikipedia, The Free Encyclopedia*, 6 Mar. 2006, `http://en.wikipedia.org/w/index.php?title=Dynamical\_system&oldid=42091498`.

[13] D. V. Prokhorov. Kalman Filter Training of Neural Networks: Methodology and Applications. *Techical paper*, Ford Research Laboratory, 2002.

[14] P. Tiňo, M. Čerňanský, Ľ. Beňušková. Markovian architectural bias of recurrent neural networks. In *IEEE Transactions on Neural Networks*, 15(1), pages 6–15, 2004.

[15] E.A. Wan. Time Series Prediction by Using a Connectionist Network with Internal Delay Lines. In A. Weigend, N. Gershenfeld editors, *Time Series Prediction. Forecasting the Future and Understanding the Past* SFI Studies in the Sciences of Complexity, Proc., XVII, Addison-Wesley, 1994.

[16] Automatically Tuned Linear Algebra Software (ATLAS). Available at `http://math-atlas.sourceforge.net`

[17] Compute Unified Device Architecture (CUDA). Available at `http://www.nvidia.com/object/cuda_home.html`

[18] D. Ron, Z. Singer, N. Tishby. The Power of Amnesia. In *Machine Learning*, 25, 1996.

[19] P. Tiňo, V. Vojtek. Extracting Stochastic Machines from Recurrent Neural Networks. In *IEEE Transactions on Neural Networks*, 2004, volume 15(1), pages 6–15.

[20] M. Čerňanský, M. Makula, P. Trebatický, P. Lacko, P.: ext Correction Using Approaches Based on Markovian Architectural Bias. In *TEANN 2007, Proceedings of the 10th International Conference on Engineering Applications of Neural Networks*. Publishing Centre Alexander Technological Educational Institute of Thessaloniki, ISBN 978-960-287-093-8, pages 221–228, 2007.

[21] M. Čerňanský, M. Makula, Ľ. Beňušková. Processing Symbolic Sequences by Recurrent Neural Networks Trained by Kalman Filter-Based Algorithms. Bratislava: Slovak University of Technology and Comenius University, 2003.

[22] E. A. Wan, A. T. Nelson. *Dual EKF Methods*, Department of Electrical and Computer Engineering, Oregon Graduate Institute, August 2000.

[23] J. Jung. Cholesky Decomposition and Linear Programming on a GPU. Scholarly paper. University of Maryland.

[24] H. Nguyen editor. GPU Gems 3. Addison-Wesley Professional, ISBN: 978-0321515261, 2007.

[25] J. Dongarra. Basic Linear Algebra Subprograms Technical Forum Standard. In *Int. J. of High Performance Applications and Supercomputing* **16**(1), pages 1–111, 2002.

[26] P. Trebatický. Recurrent neural network training with the Kalman filter-based techniques. In *Neural Network World*, Vol. 15, No. 5, (2005), pages 471–488, 2005.

[27] P. Trebatický, J. Pospíchal. Neural Network Training with Extended Kalman Filter Using Graphics Processing Unit. In *Lecture Notes in Computer Science*, volume 5164, *Artificial Neural Networks - ICANN 2008 Proceedings, Part II, 18th International Conference*, Prague, Springer Berlin/Heidelberg, pages 198–207, 2008.

Selected Papers by the Author

V. Kvasnička, P. Trebatický, J. Pospíchal, J. Kelemen. Mind, intelligence and life. STU Press, Bratislava, 2007. (in Slovak)

P. Trebatický. Recurrent neural network training with the Kalman filter-based techniques. In *Neural Network World*, Vol. 15, No. 5, pages 471-488, 2005.

P. Trebatický, J. Pospíchal. Neural Network Training with Extended Kalman Filter Using Graphics Processing Unit. In *Lecture Notes in Computer Science*, volume 5164, *Artificial Neural Networks – ICANN 2008 Proceedings, Part II, 18th International Conference*, Prague, Springer Berlin/Heidelberg, pages 198-207, 2008.

M. Čerňanský, M. Makula, P. Trebatický, P. Lacko. Text Correction Using Approaches Based on Markovian Architectural Bias. In *EANN 2007, Proceedings of the 10th International Conference on Engineering Applications of Neural Networks*, Thessaloniki, Greece, Publishing Centre Alexander Technological Educational Institute of Thessaloniki, pages 221–228, 2007.

P. Trebatický. Echo-state networks generalisation. In R. Matoušek P. Ošmera editors, *MENDEL 2005, 11th International Conference on Soft Computing*. Brno University of Technology, pages 151–156, 2005.

P. Trebatický. Recurrent Neural Network Training with the Kalman Filter Based Techniques. In *Proc. of Abstracts of 1st Slovak-Japanese Seminar on Intelligent Systems*, Herľany, Slovak Republic, organized by FEI TU Košice and Waseda University, Tokyo, Japan, 2005.

P. Trebatický. Echo-state networks generalisation. In J. Kelemen, V. Kvasnička, J. Pospíchal editors, *Cognition and Artificial Life V*. sv. 2. Opava, Silesian University pages 563–572, 2005. (in Slovak)

P. Trebatický. Kalman Filter modifications for neural networks training. In *Cognition and Artificial Life VII*. Opava, Silesian University, pages 349–355, 2007. (in Slovak)

P. Trebatický. Computation on graphics processor in artificial intelligence. In *Cognition and Artificial Life VIII*. Opava, Silesian University, pages 333–338, 2008. (in Slovak)

P. Trebatický. Recurrent Neural Network Training with the Extended Kalman Filter. In *Proceedings in IIT.SRC 2005*. STU Press, Bratislava, pages 57–64, 2005.
Awarded as the best conference paper among PhD students, IEEE CS section award

P. Trebatický. Prediction with Echo state Networks. In *Proceedings in IIT.SRC 2006*, STU Press, Bratislava, pages 79–86, 2006.

P. Trebatický. Kalman Filter Modifications for Neural Network Training. In *Proceedings in IIT.SRC 2007*, STU Press, Bratislava, pages 257–264, 2007.

P. Trebatický. Text correction by echo state neural networks. In *Cognition and Artificial Life VI. Opava, Silesian University, pages 393–399, 2006. (in Slovak)*

P. Trebatický. Recurrent neural networks training with the Extended Kalman filter. In CZ ACM Student 2004*, Praha, 2004. (in Slovak)*