# MODIFIED KALMAN FILTER BASED METHOD FOR TRAINING STATE-RECURRENT MULTILAYER PERCEPTRONS

Deniz Erdogmus[1], Justin C. Sanchez[2], Jose C. Principe[1]

Computational NeuroEngineering Laboratory,
[1]Electrical & Computer Engineering Department
[2]Biomedical Engineering Department
University of Florida, Gainesville, FL 32611.
[deniz,justin,principe]@cnel.ufl.edu

**Abstract. Kalman filter based training algorithms for recurrent neural networks provide a clever alternative to the standard backpropagation in time. However, these algorithms do not take into account the optimization of the hidden state variables of the recurrent network. In addition, their formulation requires Jacobian evaluations over the entire network, adding to their computational complexity. In this paper, we propose a spatial-temporal extended Kalman filter algorithm for training recurrent neural network weights and internal states. This new formulation also reduces the computational complexity of Jacobian evaluations drastically by decoupling the gradients of each layer. Monte Carlo comparisons with backpropagation through time point out the robust and fast convergence of the algorithm.**

## INTRODUCTION

Neural network architectures have been successfully used in numerous applications ranging from adaptive signal processing to system control [1,2]. The celebrated backpropagation algorithm proposed by Rumelhart *et al.* [3] propelled the research efforts on neural network topologies and training algorithms. Backpropagation reduced the computational complexity of the steepest descent training using the mean-square error (MSE) criterion for feedforward multilayer perceptrons (MLP). Although the time information in signals can be brought in by utilizing time-delay neural networks (TDNN) (which can be trained efficiently using the standard static backpropagation algorithm) some applications require large memory depths. The only way to achieve this with a TDNN is to increase the length of the delay-line at the input, which introduces additional parameters. Large networks suffer from the bias-variance dilemma which results in poor generalization due to model overfitting [4]. To avoid this problem, we desire small errors together with a small number of weights.

The memory shortcomings of the TDNN have motivated the use of recurrent neural networks (RNN). The memory structure of the RNN is moved to the hidden layer where additional parameters are added as the square of the number of hidden nodes (as opposed to TDNN - #taps*#inputs*#hidden nodes). Since RNNs display gradient dependencies over time, the standard backpropagation algorithm used for MLPs cannot be used for optimizing RNN parameters. By unfolding the RNN to form an equivalent feedforward network with replicated weights spanning the time dependency, the algorithm called backpropagation through time (BPTT) is applicable [5]. BPTT is not without problems itself. RNNs trained with BPTT have difficulty learning time dependencies, since gradients tend to decay exponentially due to the composite derivatives of the nonlinearities [6]. Computational and storage requirements are also demanding, since activations, injected errors, and copies of the weights have to be computed and stored over a trajectory.

In order to avoid the difficulties associated with BPTT, Kalman filter-based learning algorithms have been proposed and investigated [7,8]. Singhal and Wu were the first to propose this state estimation approach to determine the optimal weights of an RNN [7]. They have demonstrated that in training feedforward MLPs with this approach, the number of iterations required for convergence can be decreased compared to the slow-converging backpropagation. One major drawback of this original formulation was the increased computational complexity per iteration; the evaluation of the Jacobian of the outputs with respect to all the weights is already necessary in backpropagation, however, the Kalman filter necessitates the inversion of matrices for the case of multiple outputs. Puskorius and Feldkamp [8] extended the Kalman filter formulation to the training of RNNs. They have successfully trained and utilized RNNs in various applications including the adaptive control of nonlinear dynamical systems [9]. Puskorius and Feldkamp aimed to reduce the computational complexity of the Kalman flter training algorithm for RNNs by disregarding "the interdependence of mutually exclusive groups of weights" [8]. Although these algorithms produce promising results in terms of training RNNs treated like transfer functions, *ignoring* the initial internal states of the hidden layer(s) as parameters to be optimized results in suboptimal learning of the weights.

In this paper, we modify the current Kalman filter approach used to train RNNs to take into account the optimization of this important parameter. This is achieved by extending the state vector of the Kalman filter to include the *optimal* hidden states as well as the weights of the RNN. Therefore, this new approach is called the spatial-temporal extended Kalman filter (STEKF) training for RNNs. The importance of the optimization of this hidden internal state comes from the fact that recurrent networks are not merely feedforward input-output mappings. For example, training for the RNN weights assuming zero initial hidden state would be suboptimal, in general.

## MODIFIED KALMAN FILTER FORMULATION

A state-recurrent neural network with a single hidden layer, as shown in Fig. 1, is represented by the dynamical equations in (1). These equations can easily be modified to account for multiple hidden layers by including the state update equations for each layer.

$$\begin{aligned} y_{k+1}^{1} &= s\,(W_k^1 x_k + W_k^f y_k^1 + b_k^1) \\ y_k &= W_k^2 y_k^1 + b_k^2 \end{aligned} \qquad (1)$$

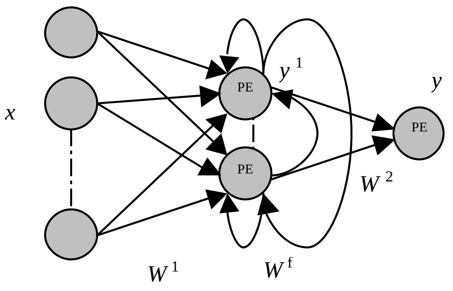


Figure 1. Fully-connected state-recurrent neural network

In (1), $W_k^1$, $W_k^2$, $W_k^f$, $b_k^1$, $b_k^2$ are the 1st layer, 2nd layer, and feedback weight matrices and the 1st layer, and 2nd layer bias vectors, respectively. The input vector sample (index $k$) to the network is $x_k$, and the internal state vector is $y_k^1$. The output of the RNN is $y_k$. Each processing element (PE) of the hidden layer contains the sigmoid-type nonlinearity denoted by $s$ (.). Our aim is to determine the optimal weights and the initial internal state, $y_0^1$, of this network for a given set of training data $\{(x_1, d_1),..., (x_N, d_N)\}$ that minimizes the MSE between the network output samples $y_k$ and the desired signal $d_k$.

In the STEKF formulation we assume that the optimal parameters (including the weights and the hidden states) to be estimated are the state variables of a hypothetical dynamical system. Suppose that we collect all the *optimal* weights of the RNN into a vector $w^*$. Since the optimal solution for the weights is fixed, we assume the dynamical relationship $w_{k+1}^* = I w_k^*$ between values of the optimal weights for consecutive time instances. On the other hand, the optimal internal state vector of the hidden layer is updated with the nonlinear equation $y_{k+1}^{1*} = s\,(W_k^{1*} x_k + W_k^{f*} y_k^{1*} + b_k^{1*})$, where all the weights assume their optimal values. Assuming that the desired signal is a noisy measurement of the RNN output evaluated using the optimal weights and the optimal initial internal state, the output can be written as $d_k = W_k^{2*} y_k^{1*} + b_k^{2*} + v_k$, where $v_k$ denotes the measurement noise (the approximation error at the optimal solution). These equations can be collected to give the following dynamical system

$$\text{Process Equation:} \quad \begin{cases} w^*_{k+1} = I w^*_k \\ y^{1*}_{k+1} = s\,(W^{1*}_k x_k + W^{f*}_k y^{1*}_k + b^{1*}_k) \end{cases} \tag{2}$$

$$\text{Output Mapping:} \quad d_k = W^{2*}_k y^{1*}_k + b^{2*}_k + v_k$$

Regarding the parameter optimization as a state estimation problem described by the equations in (2) allows us to use the extended Kalman filter to update the (optimal) weight estimates as well as the (optimal) hidden state. We summarize this update algorithm below.

*Algorithm:*

1. Initialize all the weights $W^1_k$, $W^2_k$, $W^f_k$, $b^1_k$, $b^2_k$ and the internal state $y^1_k$ to small random values. Initialize the Kalman gain matrix $K$ (possibly to all zeros). Initialize the optimal weight estimation error covariance matrix $P$ to diagonal matrix with relatively large values.
2. Select a schedule for annealing the covariance matrix $Q$ for the measurement noise $v_k$. Start from a large diagonal matrix and decrease the value as the iterations progress to a small value. Assume there exists a process noise in the process equations as well. Anneal the covariance matrix $R$ of this process noise similarly. (Experience showed that initializing $P$, $Q$, and $R$ to values of the same order of magnitude boosts convergence speed and helps avoid local minima.)
3. For each new training sample ($x_k$ , $d_k$) evaluate the Jacobian matrices of the process and output equations with respect to the state variables of (2) at the current estimates for the optimal weights and the internal state of the RNN. These matrices are given by $A$ and $C$ shown in (3). Here $s\,(w, y^1)$ denotes the second portion of the process equation and $h(w, y^1)$ denotes the output mapping in (2).

$$A = \begin{bmatrix} I & 0 \\ (\partial s\,(w, y^1)/\partial w)|_{w_k, y^1_k} & (\partial s\,(w, y^1)/\partial y^1)|_{w_k, y^1_k} \end{bmatrix}$$
$$C = \begin{bmatrix} (\partial h/\partial w)|_{w_k, y^1_k} & (\partial h/\partial y^1)|_{w_k, y^1_k} \end{bmatrix} \tag{3}$$

4. Evaluate the error for the new training sample, $e_k$, and update the Kalman gain matrix using the current Jacobian matrices.

$$\begin{aligned} e_k &= d_k - (W^2_k y^1_k + b^2_k) \\ P^- &= APA^T \\ K &= P^- C^T (CP^- C^T + Q)^{-1} \\ P &= (I - KC)P^- + R \end{aligned} \tag{4}$$

5. Update the estimates for the optimal weights and the internal state of the RNN using the Kalman gain matrix and the current output error (In (5)

we select the rows of $K$ corresponding to $w$ into $K^1$ and the rows corresponding to $y^1$ into $K^2$).

$$
\begin{aligned}
w_{k+1} &= w_k + K^1 e \\
y^1_{k+1} &= s\,(W^1_k x_k + W^f_k y^1_k + b^1_k) + K^2 e
\end{aligned}
\tag{5}
$$

6. Go to step 3 to process the next training sample.

Typically, Kalman filter training algorithms for RNNs utilize the first equation in (2), ignoring the optimization of the internal state vector. The reasons for this are two-fold:

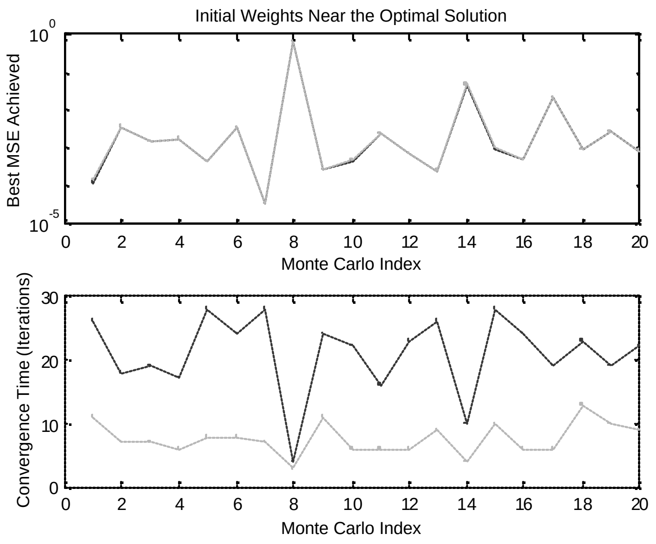
- First, there is a tendency to interpret the recurrent networks as straight input-output mappers (like a transfer function), in which case the initial state is usually set to zero.
- Second, using the traditional Kalman formulation, one would include the initial internal state of the hidden layer in the first equation of (2) along with the other weights. Once the estimate for the optimal weights and this initial state are updated, the entire training data set up to that point would have to be passed through the recursive network to evaluate the current RNN output with the new parameters. This would be computationally infeasible for long training sets.

Including the internal state estimation as a dynamic variable in the process equation of (2) solves both of these problems. In addition, the Jacobian evaluations of (3) are reduced to gradient calculations over each separate layer as opposed to the requirement of Jacobian evaluations over the entire network.

Generalizing the proposed algorithm to work for RNNs with an arbitrary number of hidden layers is easy. We simply add more equations to the process equations of (2). Eventually, there will be one equation for each hidden layer of PEs in the form $y^{l*}_{k+1} = s\,(W^{l-1*}_k x_k + W^{f_l*}_k y^{l-`*}_k + b^{l-1*}_k)$, where the weights associated with the corresponding layer are used.

## SIMULATIONS

To demonstrate the performance of the proposed STEKF training algorithm, we present the results of a Monte Carlo simulation performed on synthetic data. In order to guarantee that a global optimal solution that yields zero error exists, the training data is generated by a state-recurrent neural network, whose weights (including the initial internal state) are selected randomly. Once the RNN weights are fixed, a sequence of random vector inputs were passed through the network. This sequence of inputs and the generated network output were used as the input-output training pairs for another RNN with the same architecture. In the two sets of trainings performed, the weights of the RNN to be trained were initialized randomly either close or far to the true weights of the reference RNN that generated the training data. The internal states of the RNN being trained, however, were initialized to all zeros.



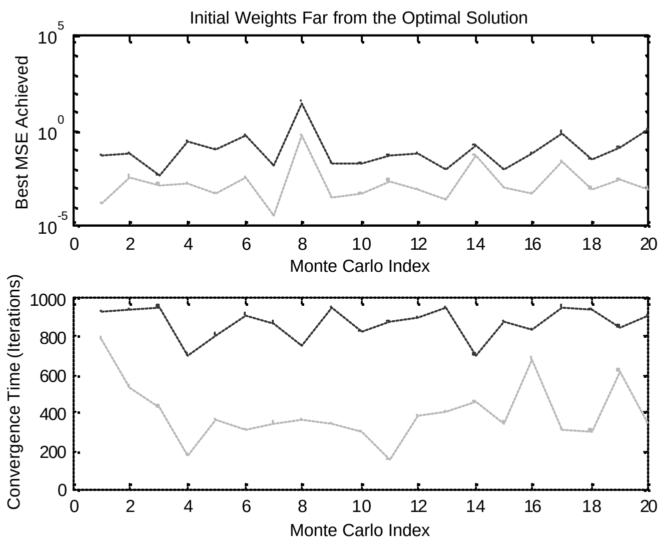
For both sets of Monte Carlo runs (20 runs for each case), the RNNs were trained on a sample-by-sample basis over 1000 training samples using the STEKF

Figure 2. Results of Monte Carlo runs with weights initialized close to the optimal solution a) Best normalized-MSE achieved by BPTT (dotted) and the Kalman filter (solid) b) Convergence time of BPTT (dotted) and the Kalman filter (solid).

approach explained above as well as the standard BPTT algorithm. In the BPTT training, the gradient backpropagation in time was truncated to one sample.

The results obtained from the Monte Carlo runs where the weights are initialized near their optimal values are summarized in Fig. 2. In the top subplot, the best normalized-MSE value obtained by both algorithms for each run is shown. Normalized-MSE is defined as the error energy divided by the energy of the desired signal for the complete training set. When computing this MSE using the current estimates of the optimal weights, we use the true initial internal state values used to generate the training data. This helps eliminate any additional MSE that would be caused by different initial conditions, thus gives a fair comparison between the two training algorithms. We observe from the presented MSE results that when initialized close to the optimal solution, both BPTT and STEKF produce equivalent solutions in terms of MSE. Nevertheless, STEKF converges to the optimal solution in fewer iterations (as already noticed by many researchers) as seen from the bottom subplot of Fig. 2. The convergence time was defined as the minimum iteration index such that the normalized-MSE is within 10% of its best-achieved value over the entire training procedure.

We also present the results obtained from the Monte Carlo runs where the weights are initialized far from their optimal values in Fig. 3. The top subplot, once again, shows the best normalized-MSE values obtained by the two algorithms. In this case, we observe the clear advantage of using the STEKF



algorithm. It is evident that this method is capable of avoiding local minima. Notice that on the average, the MSE of the solutions obtained by STEKF are equal

Figure 3. Results of Monte Carlo runs with weights initialized far from the optimal solution a) Best normalized-MSE achieved by BPTT (dotted) and the Kalman filter (solid) b) Convergence time of BPTT (dotted) and the Kalman filter (solid).

whether the weights are initialized close to or far from the optimal values. In the subplot of Fig. 3, we can see that the convergence of the Kalman filter is faster than that of BPTT.

To illustrate the structure of the learning curves, we present in Fig. 4, the normalized-MSE values for both BPTT and STEKF in a single 1000-sample training scenario. The Kalman filter clearly converges faster to a smaller MSE than the standard BPTT algorithm.

In terms of computational complexity of these algorithms, STEKF eliminates the need to compute the gradient of the RNN output with respect to the hidden layer(s)'s weights, as well as the need to consider the time-recurrency of the gradients. These are the main drawbacks of BPTT. On the other hand, additional matrix multiplications and an inversion (of a matrix the size of the output) are required by the Kalman filter.

Compared with the traditional Kalman filter approaches, the proposed algorithm exhibits the same advantages. Since only the weights of the RNN are included as in the first equation in (2) and the internal states are not considered in these traditional approaches, the Jacobian of the output mapping in (3) involves evaluating the gradient of the RNN output with respect to hidden layer and feedback weights. If one were to include the estimation of the internal states in the traditional formulation, that would necessitate considering the time-recurrence of these gradients.

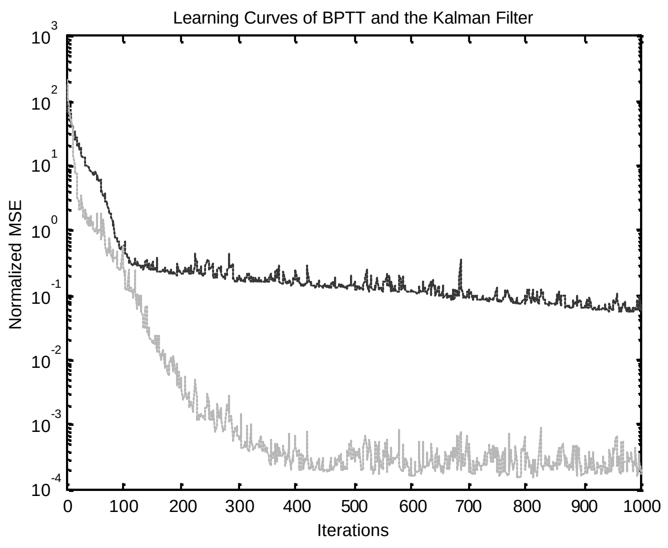


Figure 4. Learning Curves of BPTT (dotted) and the Kalman filter (solid) over 1000 samples, staring from an initial weight set far from the optimal solution.

## DISCUSSION

The most important issue in evaluating the performance of an algorithm is its computational complexity. Suppose a recurrent neural network with $L$ layers whose $l$th hidden layer has $N_l$ PEs is trained using the proposed algorithm. Assume that each hidden layer has full feedback to itself and only to itself.

TABLE 1. COMPUTATIONAL COMPLEXITY OF INDIVIDUAL EQUATIONS

| Eq | Scalar Multiplies | $s(.)$ | $s'(.)$ | Inversion |
|---|---|---|---|---|
| **(2)** | $\sum_{l=1}^{L-1} N_l (N_{l-1} + N_l) + N_{L-1}N_L$ | $\sum_{l=1}^{L-1} N_l$ | --- | --- |
| **(3)** | --- | --- | $\sum_{l=1}^{L-1} N_l(N_{l-1} + N_l + 1)$ | --- |
| **(4)** | $O(BW^2)$ | --- | --- | $N_L$ x $N_L$ matrix |
| **(5)** | $(W+B)N_L$ | --- | --- | --- |

We present, in Table 1, the number of operations required by each equation in the algorithm. In the table $W$ is the total number of weights of the RNN and $B$ is the total number of hidden PEs.

$$W = \sum_{l=1}^{L} N_l (N_{l-1} + 1) \quad , \quad B = \sum_{l=1}^{L-1} N_l \tag{6}$$

The most expensive computation, as we see in Table 1, is required by (4) in terms of scalar multiplications. The evaluation of PE nonlinearities and their derivatives constitute the second major computational load.

The advantage of this algorithm is that the derivative calculations are local to each PE, therefore backpropagation through layers or time is not an issue. The computations of the Jacobians is greatly simplified by decoupling the gradients of each layer. In addition, it becomes possible not only to optimize the RNN weights, but also to recursively update the internal states. This aspect of training recurrent neural networks has not been considered and it will lead to a smaller training MSE.

## CONCLUSIONS

Robust and fast training of recurrent neural networks has been a challenging problem for the last decade. Due to difficulties associated with the gradient evaluation in recurrent structures and the fading of magnitudes due to composite derivatives of the network nonlinearities, standard gradient-descent techniques have been unable to achieve this goal. In order to circumvent these problems linked to the backpropagation through time algorithm and its derivatives, researchers have proposed training algorithms based on the Kalman filter, formulating the training problem as one of state estimation. These algorithms, however, suffered from the computational complexity of the Jacobian evaluations. Although there have been solutions proposed to unravel these difficulties, for example by using a smaller Kalman filter for each hidden processing element and *disregarding* the interdependencies, recurrent neural network topologies did not become popular.

One other problem with the traditional recurrent neural network training algorithms has been their neglect to treat the hidden initial internal state as a parameter to be optimized. This understanding, which regarded recurrent neural networks as transfer functions, caused training algorithms to yield suboptimal solutions for the network weights.

In this paper, we proposed an extension to the traditional Kalman filter training approach by incorporating the internal state variable as an optimization parameter in the process equations. This modification results in optimal training of the weights of the recurrent networks as well as the initial hidden state variables. An added advantage of the proposed algorithm is the computational simplicity it introduces in the extended Kalman filter formulation. Incorporating the hidden state variables of the recurrent network into the state vector of the Kalman filter allows the decomposition of the network into its layers. Therefore, the Jacobian calculations can be carried out on each layer independently from all the other layers. This eliminates the backpropagation of the gradient through the layers. The recursive nature of the Kalman filter process equations take care of the time-recurrent nature of the gradients eliminating the requirement to backpropagate the gradients through time. Therefore, this new approach provides an unfolding of the recurrent network in time as well as layers.

Our comparisons with the standard backpropagation through time algorithm in Monte Carlo training runs performed on synthetic data provided evidence that this approach is a fast converging, robust way of training state-recurrent neural networks. Further work is necessary to test the performance of this training algorithm on difficult problems requiring a recurrent neural network solution. Additionally, the ideas behind the decoupled extended Kalman filter [8] and the unscented Kalman filter [10] could be used to improve the computational complexity of the proposed algorithm.

**Acknowledgments:** This work is partially supported by the grants NSF-ECS-9900394 and ONR-N00014-01-1-0405.

## REFERENCES

[1] S. Haykin, Neural Networks: A Comprehensive Foundation, 2nd Ed., Prentice Hall, Upper Saddle River, New Jersey, 1999.

[2] W.T. Miller III, R.S. Sutton, P.J. Werbos, (eds.), Neural Networks for Control, MIT Press, Cambridge, Massachusetts, 1990.

[3] D. Rumelhart, G. Hinton, R. Williams, “Learning Internal Representations by Error Back-propagation,” Nature, vol. 323, pp. 533-536, 1986.

[4] C. Bishop, Neural Networks for Pattern Recognition, Clarendon Press, Oxford, 1995.

[5] P.J. Werbos, “Backpropagation Through Time: What It Does and How to Do it,” Proceedings of the IEEE, vol. 78, no. 10, pp. 1550-1560, 1990.

[6] Y. Bengio, P. Simard, P. Frasconi, “Learning Long-term Dependencies with Gradient Descent is Difficult,” IEEE Transactions on Neural Networks, vol. 5, pp. 157-166, 1994.

[7] S. Singhal, L. Wu, “Training Multilayer Perceptrons with the Extended Kalman Algorithm,” Advances in Neural Information Processing Systems (NIPS’91), pp. 133-140, Denver, Colorado, 1988.

[8] G.V. Puskorius, L.A. Feldkamp, “Decoupled Extended Kalman Filter Training of Feedforward Layered Networks,” Proceedings of the International Joint Conference on Neural Networks (IJCNN’91), pp. 771-777, Seattle, Washington, 1991.

[9] G.V. Puskorius, L.A. Feldkamp, “Neurocontrol of Nonlinear Dynamical Systems with Kalman Filter Trained Recurrent Networks,” IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 279-297, 1994.

[10] S.J. Julier, J.K. Uhlmann, “A New Extension of the Kalman Filter to Nonlinear Systems,” Proceedings of AeroSense: The 11th International Symposium on Aerospace/Defense Sensing, Simulation and Controls, Orlando, Florida, 1997.