# An R implementation of a Recurrent Neural Network Trained by Extended Kalman Filter

**Bogdan OANCEA** (bogdan.oancea@faa.unibuc.ro)
University of Bucharest
**Tudorel ANDREI** (andrei.tudorel@csie.ase.ro)
The Bucharest University of Economic Studies
**Raluca Mariana DRAGOESCU** (dragoescuraluca@gmail.com)
Artifex University of Bucharest

**ABSTRACT**

*Nowadays there are several techniques used for forecasting with different performances and accuracies. One of the most performant techniques for time series prediction is neural networks. The accuracy of the predictions greatly depends on the network architecture and training method. In this paper we describe an R implementation of a recurrent neural network trained by the Extended Kalman Filter. For the implementation of the network we used the Matrix package that allows efficient vector-matrix and matrix-matrix operations. We tested the performance of our R implementation comparing it with a pure C++ implementation and we showed that R can achieve about 75% of the C++ programs. Considering the other advantages of R, our results recommend R as a serious alternative to classical programming languages for high performance implementations of neural networks.*

**Keywords:** *R, neural networks, Extended Kalman Filter*
**JEL Classification:** *C10, C88*

## 1. INTRODUCTION

Neural networks are one of the most used techniques for forecasting in almost every field of science. Stock market predictions, character recognition, medical diagnosis, image compression, loan applications, prediction of the business success or start-up failures, prediction in education or healthcare systems are only few areas where neural networks were successfully applied [Rojas 1996; Andrei 2015, Oancea 2013a; Oancea 2013b].

Currently there are several software packages written in different programming languages that implement neural network models trained by

various algorithms. Neural networks have been implemented even using hardware for better efficiency [Omondi, 2006].

A review of the most popular implementations of neural networks can be found in [Rojas 1996; Maren et al. 1990]. The choice of available architectures for neural networks is based on a number of factors such as the type of application, processing speed, accuracy.

In this paper we describe an R implementation of a recurrent neural network trained by the Extended Kalman Filter with the derivatives calculated by the Back Propagation Through Time method. Recurrent neural networks have been used successfully in forecasting of financial time series, because of their property to "memorize" the previous values [Emad Saad W., 1998].

R software system has currently available a number of packages that implement various models of neural networks. Neuralnet package [Günther, 2010] implements a multi-layer perceptron architecture and supervised learning algorithms that can be used for regression analysis. It possesses facility to set the activation function and error function by simply defining an R function. Neuralnet package allows the user to choose between backpropagation or resilient backpropagation methods optionally using weight backtracking (Riedmiller and Braun, 1993). Another R package that implements a neural network is nnet [Venables, 2002]. It implements a feed-forward neural network with only one hidden layer, and it is used for multinomial log-linear models. RSNNS [Christoph Bergmeir, 2012] is an R wrapper of the Stuttgart Neural Network Simulator that contains several standard implementations of neural networks. It contains a low-level interface that allows access to almost all of the functionality of SNNS and also a high-level interface that makes integrating most of the common neural network topologies (including recurrent neural networks, multilayer perceptron, self-organizing maps, and radial basis function networks) and training methods into R seamlessly.

In this paper we describe an R implementation of a recurrent neural network trained by Extended Kalman Filter with the output derivatives computed by Truncated Back Propagation Through Time which is the first R implementation of such a training method for a neural network to our knowledge. Although RSNNS contains an implementation of a recurrent neural network, it uses only backpropagation or some variants of it for training the network. Studies like [Saad, 1998] showed that EKF clearly outperforms other neural networks training methods for time series predictions.

The rest of the paper is organized as follows: section 2 contains a brief theoretical introduction for extended Kalman filters, section 3 describes how EKF is applied to recurrent neural network weights estimation, section 4

contains a description of the algorithms' implementation and we conclude the paper with some proposals for further developments.

## 2. EXTENDED KALMAN FILTER

Kalman filtering (KF) is a technique that produces estimates of unknown variables using a series of measurements containing statistical noise. It is a recursive technique that process new data as they arrive being suited for inline real time processing. KF provides an optimal estimator for the unknown variables – if the noise is Gaussian, KF is the best linear estimator, minimizing the mean square error of the estimated variables. KF is used to estimate the state of a system when the state vector cannot be directly measured but we can get measurements of the system at a constant rate. A comprehensive description of the Kalman Filter and its application can be found in [Grewal, 2015].

While the original Kalman Filter works only with linear equations, Extended Kalman Filter can estimate the unknown variables even if the function that describes the transition of the system state and measurement equations is nonlinear but differentiable.

A short description of the EKF is given below. Consider a nonlinear system described by the following 2 equations:

State transition equation: $x_k = f(x_{k-1}) + q_{k-1}$ (1)

Measurement equation: $z_k = h(x_k) + r_k$ (2)

where $x_k$ is a vector that describes the system state, $z_k$ is the observation vector (values obtained through a direct measurement of the system), $q_k$ is the process noise vector, $r_k$ is the measurement noise vector, $f(...)$ is the a function that gives the state transition of the system (a nonlinear function), $h(...)$ is the observation (nonlinear) function, $Q_k = \mathrm{E}[\mathrm{q_k} \cdot \mathrm{q_k^t}]$ is the process noise covariance matrix and $R_k = E[r_k \cdot r_k^t]$ is the measurement noise covariance matrix.

The Extended Kalman filter is a predictor-corrector approach of jointly solving these equations. The solution is described in [Jaeger, 2002; Haykin, 2001; Welch, 2006] and consist in three basic steps:

1. Initialization:

$x_k^a = x_0$ with error covariance matrix $P_0$

2. The *k*-th predictor step:

$x_k^p = f(x_{k-1}^a)$

$P_k^p = J_f(x_{k-1}^a) \cdot P_{k-1} \cdot J_f^t(x_{k-1}^a) + Q_{k-1}$

3. The *k*-th corrector step (i.e. measurement update)

$x_k^a = x_k^p + K_k \cdot (z_k - h(x_k^p)$

$$K_k = P_k^p \cdot J_h^t(x_k^p) \cdot (J_h(x_k^p) \cdot P_k^p \cdot J_h^t(x_k^p) + R_k)^{-1}$$
$$P_k = (I - K_k \cdot J_h(x_k^p) \cdot P_k^p$$

Here *a* stands for the actual value of the variable, *p* for the predicted value, $J_f$ is the Jacobian matrix of the $f()$ function and $J_h$ is the Jacobian matrix of $h()$ function and $K_k$ a matrix called the Kalman gain.

## 3. APPLYING EXTENDED KALMAN FILTER TO NETWORK WEIGHT ESTIMATION

Applying Extended Kalman Filter to estimate the weights of a neural network supposes to interpret the weight vector as a state of a dynamical system. If we note by $w$ the vector of the network weights, $x$ the vector of input data and $y$ the vector of network output we can write:

$y(n) = h(w, x(0)\ x(1), ...x(n)$ where *n* is the number of input data vectors.

We will assume that the network weights update process adds a small Gaussian uncorrelated noise at every step [Haykin, 2001; Jaeger, 2002], so we can write the updating process in the following form:

State transition equation: $w(n+1) = w(n) + q(n)$ (3)
Measurement equation: $y(n) = h_n(w(n)\ + r(n)$ (4)

that are similar to equations (1) and (2). We can now apply EKF procedure to estimate $w$. Because of the simple form of the system state transition equation, the EKF procedure can be written as:

$$K_n = P_n \cdot H_n^t \cdot (R_n + H_n P_n H_n^t)^{-1} \quad (5)$$
$$w_{n+1} = w_n + K_n \cdot \varepsilon_n \quad (6)$$
$$P_n = P_n - K_n \cdot H_n \cdot P_n + Q_n \quad (7)$$

where $\varepsilon_n$ is the difference between the desired ouptut and the actual network output. This procedure finds the weights values that minimizes the sum of the squared erros: $\sum_{i=1...n} \varepsilon_i \varepsilon_i^t$. Applying EKF procedure needs the value of the matrix H whose elements are the derivatives of the network output. This can be computed using Back Propagation Through Time or a version of it, Truncated Backpropagation Through Time that reduces the memory and computational demands. A description of these algorithms is given in [Werbos, 1990; Wiliams and Zipser, 1995].

A short description of the network training algorithm by EKF is given below:

1. Input $x_n$ is propagated and the network provides output $y_n$. The error is computed very simply: $\varepsilon_n = d_n - y_n$;
2. Compute the derivative matrix $H_n$ by Truncated Back Propagation Through Time algorithm;
3. Compute Kalman gain matrix $K_n$ based on $H_n$, $P_n$ and $R_n$ (equation 5);
4. Update network weights (equation 6);
5. Update error covariance matrix (equation 7).

We set the values of the parameters of EKF procedure according to some general recommendations given by [Saad et al., 1998, Haykin, 2001]:

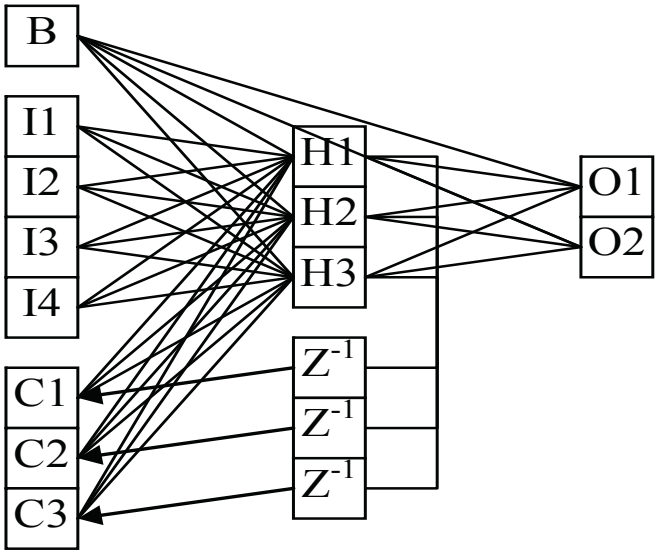
- the input and output data were scaled to zero mean, unit covariance;
- the weights were initialized with some small values given by a zero-mean normal distribution;
- $P_0 = \varepsilon^{-1} I$, where $\varepsilon$ is a small constant;
- $R_n = \eta^{-1} I$ where $\eta$ is a small constant;
- $Q_n = q_n I$ where $q_n$ ranges from 0 to 0.001.

## 4. RNN IMPLEMENTATION

We used Elman Simple Recurrent Network architecture to implement our RNN. This architecture is depicted in figure 1. Here we have a network with 4 input units (I1..I4), 2 output units (O1..O2), 3 hidden units (H1..H3), a bias unit (B), 3 context units (C1..C3 - they store the output values of the hidden units at the previous moment, t-1).

**An Elman Simple Recurrent Network**

*Figure 1*




The network was implemented in R that has several advantages over traditional languages like C++ or Java. We mention here only few of them:

- R can deal with data input/output operations easily;
- R uses the state-of-art matrix libraries (BLAS and LAPACK);
- R handles vectors and matrices very easy;
- R has advanced features for data visualization;
- There are specialized R packages that allow users to work with very large matrices (bigmemory, bigalgebra) and allow manipulation of dense as well as sparse matrices.

The data structures needed to implement the recurrent neural network and EKF training algorithm are vectors (for input, output, network weights and other temporary storage) and matrices for EKF implementation. We chose to use Matrix package to implement the data structures and algorithms needed for our neural network because it provides an extension of the basic matrix data type that allows efficient implementations of vector-matrix or matrix-matrix operations. Matrix package is now part of the standard R distribution [Bates and Maechler, 2016] and it is very easy to use. All these operations are straightforward to implement using the Matrix package. Before choosing Matrix package we run few performance tests for matrix-matrix multiplications and matrix inversion where we compared Matrix and MASS packages. All the

tests were run on a computer with an Intel Xeon E3 at 3.5 GHz processor and 16 Gb of RAM memory. The results of our experiments are presented in table 1 and showed that Matrix package outperforms MASS for large matrices.

Table 1. The time needed for matrix operations using Matrix and MASS packages (s)

| Matrix dimension | Matrix package | | MASS package | |
|---|---|---|---|---|
| | matrix-matrix multiplication | matrix inversion using solve() | matrix-matrix multiplication | matrix inversion using ginv() |
| 10 | 0.60 | 0.36 | 0.50 | 0.60 |
| 100 | 0.62 | 0.32 | 0.50 | 0.66 |
| 1000 | 0.70 | 0.69 | 0.80 | 3.29 |
| 5000 | 70 | 72 | 72 | 398 |

The pseudo code for EKF training is given below:

```
While not end of training {
    Forward propagation of the training data
    Compute the network error based on the network output
    Compute the derivatives matrix H using TBPTT
    Initialize P, Q and R matrices
    K = P * Transpose(H) * Inverse(H * P * Transpose(H) + R);
    P = P - K * H * P + Q;
    w = w + K * (D - O);
}
```

In our implementation we used a sigmoid activation function for the network units. The training algorithm is repeated for a predefined number of epochs. In a previous paper [Oancea et al. 2015] we showed that R could obtain performances similar to direct compiled programs written in C or C++ if a highly tuned linear algebra library is used. We compared our R implementation of the neural network with a pure C++ implementation and the results showed that R obtained about 75% of the running time compared with a C++ version. Both R and C++ implementations used the same BLAS and LAPACK libraries for matrix computations. We used a data series containing 3000 values of IBM stock exchange daily prices for our tests and we forecasted one value ahead using 20 past values. The data set was divided in a subset for training (75%) and another subset for testing (25%). The relative error of the prediction was about 5%.

## 5. CONCLUSIONS

In this paper we considered a recurrent neural network trained by Extended Kalman Filter. This type of neural network is suitable for time series forecasting because it could incorporate knowledge from the past values. Extended Kalman Filter was proved to be superior to other training algorithms for recurrent networks. We implemented our network in R and compared the performances of the network with a pure C++ implementation. We showed that even R is not a compiled language it can obtain very good performances. Combining this result with other well know advantages of R we can draw the conclusion that R is a serious candidate that can be used to solve such problems. For the future we intend to implement our model into an R package.

**References**


1. **Catalina Liliana Andrei, Bogdan Oancea, Monica Nedelcu, Ruxandra Diana Sinescu**, 2015, *Predicting Cardiovascular Diseases Prevalence Using Neural Networks*, Economic computation and economic cybernetics studies and research, 49(3) pp. 73-84.
2. **Douglas Bates** and **Martin Maechler**, 2016, *Sparse and Dense Matrix Classes and Methods*, CRAN archive, ***https://cran.r-project.org/web/packages/Matrix/Matrix.pdf.***
3. **Christoph Bergmeir, Jose M. Benitez**, 2012, *Neural Networks in R Using the Stuttgart Neural Network Simulator: RSNNS*. Journal of Statistical Software, 46(7), pp. 1-26.
4. **Frauke Günther** and **Stefan Fritsch**, 2010, *neuralnet: Training of Neural Networks*, The R Journal Vol. 2/1, June 2010.
5. **Mohinder S. Grewal, Angus P. Andrews**, 2015, *Kalman Filtering: Theory and Practice with MATLAB*, 4th Edition, Wiley-IEEE Press.
6. **Simon Haykin**, 2001, *Kalman Filtering and Neural Networks*, John Wiley and Sons.
7. **Herbert Jaeger**, 2002, *Tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the "echo state network" approach*, Technical Report GMD Report 159, German National Research Center for Information Technology.
8. **Alianna J. Maren, Craig T. Harston, Robert M. Pap**, 1990, *Handbook of Neural Computing Applications*, Academic Press Inc.
9. **Bogdan Oancea, Stefan Ciucu**, 2013a, *Time series forecasting using neural networks*, Proceedings of the Challenges of the Knowledge Society Conference, Bucharest, pp .1402-1408.
10. **Bogdan Oancea, Raluca Mariana Dragoescu, Stefan Ciucu**, 2013b, *Predicting students' results in higher education using neural networks*, International Conference on Applied Information and Communication Technologies (AICT2013), Jelgava, Latvia.
11. **Bogdan Oancea, Tudorel Andrei, Raluca Mariana Dragoescu**, 2015, *Accelerating R with high performance linear algebra libraries*, Romanian Statistical Review, no. 3, pp. 109-117.
12. **Amos R. Omondi, Jagath C. Rajapakse,** 2006, *FPGA Implementations of Neural Networks*, Springer.
13. **R. Rojas**, 1996, *Neural Networks*, Springer-Verlag, Berlin.

14. **Emad W. Saad, Danil V. Prokhorov, Donald C. Wunsch**, 1998, *Comparative Study. Of Stock Trend Prediction Using Time Delay, Recurrent and Probabilistic Neural Networks*, IEEE Transactions on Neural Networks, Vol. 9, No. 6, November 1998.
15. **Venables, W. N. & Ripley, B. D.**, 2002, *Modern Applied Statistics with S*. Fourth Edition. Springer, New York.
16. **Greg Welch**, and **Gary Bishop**, 2006, *An Introduction to the Kalman Filter*, UNC-Chapel Hill, TR 95-041.
17. **Paul J. Werbos**, 1990, *Backpropagation through time: what it does and how to do it*. Proceedings of the IEEE, Volume 78, Issue 10, pp. 1550–1560.
18. **Ronald J. Wiliams, David Zipser**, 1995, *Gradient-Based Learning Algorithms for Recurrent Networks and Their Complexity*, in Y. Chauwin & D. E. Rumelhart (Eds.), Back-propagation: theory, Architectures and Applications, Hillsdale NJ: Erlbaum.