ForeNet : Fourier Recurrent Networks for Time Series Prediction

Ying-Qian ZHANG and Lai-Wan CHAN

Department of Computer Science and Engineering
The Chinese University of Hong Kong
New Territories, Hong Kong
Email : {yqzhang, lwchan}@cse.cuhk.edu.hk
http://www.cse.cuhk.edu.hk/~lwchan

Abstract

Recurrent neural networks have been established as a general tool for fitting sequential input/output data. On the other hand, Fourier analysis is a useful tool for time series analysis. In this paper, these two fields are linked together to form a new interpretation to recurrent networks for time series prediction. Fourier analysis of a time series is applied to construct a complex-valued recurrent neural network. The proposed network is called Fourier Recurrent Network (ForeNet). We showed the proper parameter initialization and the learning algorithm for the complex weights in ForeNet. Experimental results show that ForeNet speeds up the learning, and the generalization performance is superior to traditional recurrent network.

1 Introduction

Time Delayed Recurrent networks are essentially feedforward networks with the addition of feedback connections. Due to the existence of these recurrent links, the recurrent neural network models preserve information through time and are more powerful than the static feedforward networks, especially in dynamic problems. They have been successfully applied to many areas, such as speech recognition [16, 14], grammar learning [8, 10] and the parsing problem [12]. Time series prediction [18, 1, 17, 7, 5] is also a major applicational area of recurrent networks. At present, recurrent networks are mostly formulated as nonlinear autoregression models [6] when applied to time series prediction problem. In this paper, we use a novel approach to interpret the recurrent networks. Here, we interprete the recurrent network as the decomposition of the time series into different frequency components and prediction is achieved using the reconstruction from the frequency components. Our recurrent network is named as *Fourier Recurrent networks (ForeNet)*.

The Fourier transform has proven to be a versatile tool that gives a better handle on the series data. A result of using the Fourier Analysis is that the parameters involved are complex number. Thus ForeNet is a complex recurrent network. At present, most commonly used neural network learning algorithms assumed real parameters. Previous works have been done to extend neural networks to cope with complex weights. The backpropagation algorithm for training a feedforward neural network with complex weights have been proposed [2] [9]. Georgious and Manolakos [13] used Complex Real Time Recurrent Learning algorithm to train a fully recurrent network. Using the complex parameters to construct neural network avoids the problem of the standstill in learning [15], and can also deal with dynamic time-sequential signal more stably and smoothly than the conventional recurrent networks [11].

In this paper, ForeNet was trained by Complex Real Time Recurrent Learning algorithm (CRTRL), which combines [5] and [13]. From our Fourier Analysis, we also provide a method of parameters initialization which reduces the initial network error and prevents the network from getting stuck with the initial weights. The experimental results show that ForeNet is computational efficient. It increases the rate of convergence and attains better generalization performance.

2 Fourier Analysis and its Recursive Form

2.1 Fourier Analysis of Time Series

The Fourier Analysis of a time series is to decompose the time series into a sum of sinusoidal components [3]. Consider a sequence of time series data $x(1), \cdots, x(t), \cdots, x(T)$. As the time is discrete, we use the Discrete Fourier Transform (DFT) to form

$$c_n = \sum_{t=1}^{T} x(t) \exp(-j\omega_n t) \qquad n = 1, 2, ..., T \tag{1}$$

and

$$x(t) = \frac{1}{T} \sum_{n=1}^{T} c_n \exp(j\omega_n t) \qquad t = 1, 2, ..., T \tag{2}$$

where $\omega_n = \frac{2\pi n}{T}$. Define

$$h_n(t) = \frac{1}{T} c_n \exp(j\omega_n t) \tag{3}$$

We have

$$\begin{aligned} x(t+1) &= \frac{1}{T}\sum_{n=1}^{T} c_n \exp(j\omega_n(t+1)) \\ &= \sum_{n=1}^{T} h_n(t)\exp(j\omega_n) \end{aligned} \tag{4}$$

for $t = 1, 2, ..., T-1$.

Suppose we append the series by an extra value $x(T+1)$ and the series becomes $x(1), \cdots, x(t), \cdots, x(T), x(T+1)$. The new $T+1$ points DFT is updated as follows

$$c'_n = \sum_{t=1}^{T+1} x(t)\exp(-j\omega'_n t) \tag{5}$$

for $n = 1, 2, ..., T+1$. If T is sufficiently large, we approximate ω'_n by ω_n and

$$c'_n \cong c_n + x(T+1)\exp(-j\omega_n(T+1)) \tag{6}$$

for $n = 1, 2, ...T$. The corresponding $h'_n(T+1)$ becomes

$$\begin{aligned} h'_n(T+1) &= \frac{1}{T+1} c'_n \exp(j\omega'_n(T+1)) \\ &\cong \frac{c_n \exp(j\omega_n(T+1)) + x(T+1)}{T+1} \\ &= \frac{T}{T+1} h_n(T)\exp(j\omega_n) + \frac{x(T+1)}{T+1} \end{aligned} \tag{7}$$

It suggests that the value of the function h'_n at time $T+1$ can be derived from its value at previous time step T and the current time series value.

3 Fourier Recurrent Neural Nets

3.1 Neural Networks Representation

We have demonstrated the recursive expression of the discrete Fourier Transformation in the previous section. In this section, we will make use of the recursive formulation to construct our recurrent neural networks for time series prediction.

Here our input to the recurrent network is $x(t)$ at each time step t. The network output and hidden units outputs are obtained as follows.

$$\begin{aligned} z(t) &= \sum_{i=1}^{N} v_i y_i(t) \\ y_i(t+1) &= f(\sum_{j=1}^{N} W_{ij} y_i(t) + u_i x(t+1)) \end{aligned} \tag{8}$$

where f is the activation function of the hidden units. W_{ij} is the recurrent weights. u_i and v_i are the weights connecting input to hidden units and hidden to output units. N is the number of hidden units in the network.

To predict the future time series $x(T+1)$ based on historical observations $x(1), x(2), ..., x(T)$, we input $x(t)$ to the network at each time step t. The network is expected to output the next step prediction $x(t+1)$, i.e., prediction of $x(T+1)$ can be obtained after we have inputted the whole sequence, $x(t), t = 1, 2, ..., T$ to the network. As recurrent networks are used, the hidden units can also serve as internal memories to store the intermediate states of the system.

Comparing equation (8) to equations (7) and (4), we can find a direct correspondence between them by putting $y_i(t) = h_n(t)$, $u_i = \frac{1}{T+1}$, $v_i = \exp(j\omega_n)$, $W_{ij} = \exp(j\omega_n)$ for $j = i$, $W_{ij} = 0$ for $j \neq i$, and f as a linear activation function. In this way, the output of the network would be $z(t) = x(t+1)$ which is the next step prediction of the time series, and this prediction is achieved by extrapolating equation (4) to $t = T$. Thus, in equation (7), the period of the time series T is a variable, which varies with the training.

One difference of the two forms is that the DFT representation requires a summation of T terms in equation (2), whereas in a recurrent network, the output unit is evaluated using the summation of N terms, where N corresponds to the number of hidden units. It is not practical to have a recurrent network with N hidden units where N is comparible to the length of the sequence, which is supposed to be long. Fortunately, in most time-series, the component frequencies are usually with low orders. High order frequencies usually are of small magnitude and they may correspond to the undesirable noise term. Thus, we can assume that the higher frequency terms are negligible and hence we use a small and fixed number of hidden units.

To summarize this, we rewrite the prediction equations (4) and (7) in the form

$$x(t+1) = \sum_{j=1}^{N} h_j(t) W_{jl} \tag{9}$$

$$h_j(t+1) = W_{ij} h_j(t) + \frac{1}{T+1} x(t+1) \tag{10}$$

There are two more implications of using the correspondence between the DFT and the recurrent network formulation.

- First, the weights of the recurrent networks are pre-defined constants and are evaluated from the input sequence. However, it has to be computed off-line. If we do not allow any off-line computation of the weights, we can apply the recursive equations to estimate the weights. In addition, in the computation, we have made a few approximations on the time-series. Some adjustments to the weights using the recursive equations may be required. Further, in case if the time series is non-stationary, the pre-defined weights cannot reflect the latest statistics of the time series. Thus, a learning algorithm to adapt the weights is still desirable. The pre-defined weights can be used as an option to initialize the weights.

- The second implication is the weights involved in the recurrent network are no longer real but complex. Traditional BPTT or RTRL learning algorithms are not applicable but we adopt the Complex Real Time Recurrent Learning algorithm as the learning algorithm [13].

3.2 Locally Recurrent Neural Network

According to the network representation, the hidden unit $h_j(t+1)$ is derived from its previous value $h_j(t)$. It suggests that there exit recurrent links in the hidden layer of neural networks and the recurrent links are diagonally connected only. On one hand, Fully Recurrent Network (FRN) is a very general architecture, which has a great freedom to build its own internal representations for encoding temporal information, and has shown a strong ability to time series learning. On the other hand, locally connected recurrent models have been suggested as an alternative to FRN because of its gain in complexity in both computational time and storage space [5]. In addition, it has been shown that a FRN with linear activation functions is inherently equivalent to some band diagonally connected recurrent networks [4]. Therefore, we adopt the locally connected recurrent models called ring-structured Neural Network (RRN) [5], which needs a much shorter training time and the performance is comparable to that of FRN. The architecture of RRN is shown in Figure 1. The inter-layer connections are all running forward and recurrent links exist in the hidden layer only.

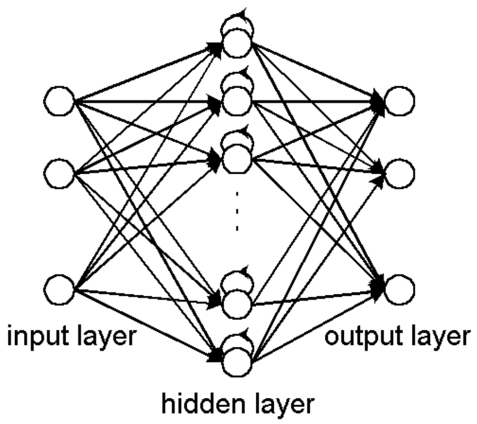

Figure 1: Locally Connected Recurrent Network

4 Learning Algorithm

We have proposed a recurrent network structure whose parameters are complex values and constructed to perform the predict task. This section shows an algorithm for adjusting the parameters of the network. The learning is based on a sample of time series input/output pairs $(x(t), x(t+1))$ for $t = 1, 2, ..., T-1$. The most widely used algorithm for training RNN is the Real Time Recurrent Learning (RTRL) algorithm proposed by Williams and Zipser [19]. However, RTRL is not suitable in our case to process complex parameters because it assumes all parameters are real-valued. Thus we are showing the learning algorithm for the complex-valued recurrent network. There are two steps in our learning process. The first one is the network initialization which is an optional step. The second is the training of the network using a modified complex RTRL algorithm.

4.1 Network Initialization

An initial guess for the network parameters can be derived by the links found in the previous section. Comparing the equations (9) and (10) with the equations (4) and (7), we have an initial estimation of the weights $W_{ij}(0)$, $W_{jl}(0)$ and the value of hidden units at initial time $h_n(t=0)$. The proposed initialization is the following

1. The weights between two hidden units, W_{ij}, can be initialized based upon the recursive equation (7). That is

$$W_{ij}(0) = \frac{1}{T'+1}\exp(j\omega'_n) \tag{11}$$

where different n is selected to generate different weight. Similarly, the weights between output units and hidden units, W_{jl}, are initially estimated by

$$W_{jl}(0) = \exp(j\omega'_n) \tag{12}$$

The weights between input and hidden units

$$W_{kj}(0) = \frac{1}{T'+1} \tag{13}$$

2. Since equation (10) is recursive, an initial estimate of the state of the hidden units should be given. Using the network inputs $x(t)(t = 1, ..., T'-1)$, we can get some rough estimate of $h_n(t=0)$ by discrete Fourier Transform.

$$\begin{aligned} h_n(t=0) &= c_n \exp(j\omega'_n t) \\ &= \frac{1}{T'}\sum_{\tau=1}^{T'} x(\tau)\exp(-j\omega'_t\tau) \end{aligned} \tag{14}$$

Where, T' is the estimated period of time series at the begining of training. Thus, an initial guess for the network parameters can be achieved. The advantage of such initialization is computational efficient. The proposed method may increase the rate of convergence while the generalization performance is comparable with other recurrent model. We will show its advantages through experiments in the next section.

4.2 Complex Real Time Recurrent Learning (CRTRL)

The Complex RTRL algorithm was proposed in [13] to train Fully Recurrent Networks. In this section, CRTRL is modified to suit our model. Real and complex numbers are denoted by lower and upper case letters. The real and imaginary parts of a complex number W is denoted by w_R and w_I respectively. Consider a RRN with p inputs, n hidden units and m output units. We use P, N and M to denote the set of input units, the set of hidden units and the set of output units respectively. All units are interconnected as described above.
The cost function is

$$e(t) = \frac{1}{2} \sum_{u_k \in M} |E_k(t)|^2 \tag{15}$$

where

$$\begin{aligned} E_k(t) &= D_k(t) - Z_k(t) \\ &= e_{Rk}(t) + j e_{Ik}(t) \end{aligned} \tag{16}$$

$E_k(t)$ is the error of the k^{th} output unit at time t. $D_k(t)$ denotes the k^{th} target and $Z_k(t)$ is the network output.

Forward Phase: For $j = 1, ..., N$

$$Y_j(t+1) = f(\sum_{u_i \in N} W_{ij} Y_j(t) + \sum_{u_k \in P} W_{kj} X_k(t+1)) \tag{17}$$

$$Z_j(t+1) = \sum_{u_l \in M} W_{jl} Y_j(t+1) \tag{18}$$

where
$Y_j(t+1)$: output of hidden unit j at time $t+1$;
W_{ij} : weights between hidden units;
W_{kj} : weights from input units to hidden units;
W_{jl} : weights from output units to hidden units;
$X_k(t+1)$: k^{th} input at time $t+1$.
f denotes the linear activation function. Expressing this equation with the real and imaginary parts of W_{ij}, W_{kj}, Y_j and X_k

$$\begin{aligned} y_{Rj}(t+1) = f(&\sum_{u_i \in N} (w_{Rij} y_{Rj}(t) - w_{Iij} y_{Ij}(t)) \\ &+ \sum_{u_k \in P} (w_{Rkj} x_{Rk}(t+1) \\ &- w_{Ikj} x_{Ik}(t+1))) \end{aligned} \tag{19}$$

Learning Phase: Since $e(t)$ is a real-valued function, we compute its gradient with respect to W_{ij}

$$\frac{\partial e(t)}{\partial W_{ij}} = \frac{\partial e(t)}{\partial w_{Rij}} + j \frac{\partial e(t)}{\partial w_{Iij}} \tag{20}$$

Now, differentiating (19), we get

$$\frac{\partial z_{Rk}(t+1)}{\partial w_{Rij}} = f'(y_{Rk}(t+1)) \frac{\partial y_{Rk}(t+1)}{\partial w_{Rij}} \tag{21}$$

The weights between the hidden layer and the output layer are non-recurrent in our model and can be updated easily by error backpropagation

$$\frac{\partial z_{Rk}(t+1)}{\partial w_{Rij}} = f'(y_{Rk}(t+1)) z_{Rl} \tag{22}$$

where, $u_k \in M$ and $u_l \in N$. The weights to the hidden units are recurrent and we can propagate the error terms one step back from the output layer to the hidden layer and update the weights according to equation

$$\begin{aligned} \frac{\partial z_{Rk}(t+1)}{\partial w_{Rij}} = f'(y_{Rk}(t+1)) (&\sum_{l=1}^{N} (w_{Rkl} \frac{\partial z_{Rl}(t)}{\partial w_{Rij}} \\ &- w_{Ikl} \frac{\partial z_{Il}(t)}{\partial w_{Rij}} + \delta_{ik} x_{Rj}(t)) \end{aligned} \tag{23}$$

where

$$\delta_{ik} = \begin{cases} 1 & \text{for } u_i = u_k \\ 0 & \text{for } u_i \neq u_k \end{cases} \tag{24}$$

In the similar way, we derive the recursive equation for $\frac{\partial z_{Rk}}{\partial w_{Iij}}$, $\frac{\partial z_{Ik}}{\partial w_{Rij}}$ and $\frac{\partial z_{Ik}}{\partial w_{Iij}}$. Finally, the weight update equation becomes

$$\Delta W_{ij}(t) = -\eta \frac{\partial e(t)}{\partial W_{ij}} + \alpha \Delta W_{ij}(t-1) \tag{25}$$

where, η is the learning rate, α is the momentum coefficient and $\Delta W_{ij}(t)$ is the change in W_{ij} at time t. The weight changes are calculated at each epoch along the way and take the full change as the time-sum of $\Delta W_{ij}(t)$ at the end of the training sequence.

5 Experiments and Discussion

To demonstrate the performance of the proposed training method both on convergence and generalization, ForeNet was applied to solve some prediction problems. The architecture of the recurrent model was a 1-10-1 network, i.e., the network had one input unit, one output unit and 10 hidden units. Its initial parameters were generated from equations (11),(12),(13) and (14). And the learning algorithm was the CRTRL. In order to show the advantage of the proposed initialization method, we compared the learning

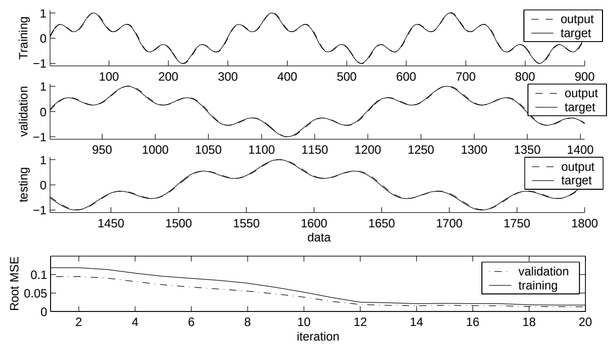

Figure 2: Training curves of ForeNet

results to a network initialized with random complex weights. Meanwhile, the learning performance of a ring-structure recurrent model (RRN), which was trained by RTRL, was also provided as a comparison.

5.1 Sine curve prediction

ForeNet was constructed to predict the next time step number in the following sinusoidal series:

$$y(t) = 0.7 \sin w_1 t + 0.3 \sin w_2 t \tag{26}$$

where, $w_1 = \frac{2\pi}{300}$ and $w_2 = \frac{2\pi}{60}$.
t is from 1 to 1800, and the network generates the output time series. The first 900 points were used as training data. The next 500 data were used for validation, whose performance was the stopping criteria of the training process. The remaining points were testing data. We used the root mean square error (RMSE) to monitor prediction performance of the network. Figure 2 shows the fitting and learning curves. It shows that after 20 training iterations the network converged and the prediction data and the target data were fitted well.

Table 1 is the comparison of the three networks. The results presented are the initial network RMSE, the RMSE of testing data, the number of iterations required and the training time taken. The number of iterations required by ForeNet initialized by the proposed method is smaller than the traditional training method using the ring-structure recurrent network. The proposed method needs less training time to achieve the convergence, and it has better generalization performance in terms of the root mean square error of testing set. It also shows that the proposed algorithm dramatically reduced the initial error. Compared to the Fourier recurrent network whose weights are randomly initialized, our method shows its advantage on the convergence speed.

Method	Initial RMSE	Testing RMSE	Number of iteration	Time (sec)
ForeNet	0.1186	0.0110	20	288
ForeNet with random initial weights	0.4151	0.0139	30	449
Ring RNN	0.3958	0.0211	83	977

Table 1: network performance on sine curve prediction

For such noise free periodic series prediction problem, theoretically, according to equation (9) and (10) the prediction function can be attained with infinite Fourier frequencies. However, only finite hidden nodes can be used in neural networks to represent the infinite frequencies. Although after the proper initialization of network parameters, the learned function is closed to the true function, the training is still necessary to modify the parameters in order to obtain the best approximation function.

5.2 The sinusoidal series with noise

We added to equation (26) at each time step a uniformly distributed random value σ in the interval $[-0.5, +0.5]$:

$$y(t) = 0.7 \sin w_1 t + 0.3 \sin w_2 t + \sigma \tag{27}$$

The whole series has 1800 data, 900 for training, 500 for validation and 400 for testing.
The results are shown in Table 2. According to the iterations required and the training time, the proposed method is obviously speed up the convergence rate. In terms of testing performance, the ForeNet outperforms both the ForeNet with random initialization and the RNN with traditional recurrent network training method.

Method	Initial RMSE	Testing RMSE	Number of iteration	Time (sec)
ForeNet	0.2079	0.0164	15	200
ForeNet with random initial weights	0.5341	0.0216	32	532
Ring RNN	0.5050	0.0396	94	1127

Table 2: netowrk performance on noisy sinusoidal series prediction problem

5.3 Mackey-Glass series prediction

We consider the chaotic time series prediction of the Mackey-Glass delay differential equation [Mackey and Glass, 1977]

$$\frac{\partial x(t)}{\partial t} = -bx(t) + a\frac{x(t-\tau)}{1 + x(t-\tau)^{10}} \tag{28}$$

It is chosen to show the performance of the recurrent networks with high dimensional systems, and it possesses many dynamic properties. ForeNet was applied to predict six steps time into the future. We choose delay $\tau = 17$, $a = 0.2$ and $b = 0.1$. The performance is shown in Table 3. The results again show that ForeNet dramatically reduced the number of iterations and the training time required. ForNet is capable of forecasting time series problem with more acceptable generalization performance than traditional training method.

Method	Initial RMSE	Testing RMSE	Number of iteration	Time (sec)
ForeNet	0.3380	0.1056	15	335
ForeNet with random initial weights	0.7284	0.0926	32	467
Ring RNN	0.9091	0.1458	135	977

Table 3: netowrk performance on Mackey-Glass prediction problem

6 Conclusions

In this paper we have introduced the Fourier recurrent networks (ForeNet) for time series prediction. We derived the link between Fourier analysis and recurrent networks. ForeNet has some advantages. Its method of parameters initialization helps to speed up the training and better generalization ability has been shown in experimental results.

Acknowledgement The authors would like to thank The Research Grants Council, HK for support.

References

[1] A. Back and A.C. Tsoi. FIR and IIR synapses, a new neural network architecture for time series modeling. *Neural Computation*, 3:375–385, 1991.

[2] N. Benvenuto and F. Piazza. On the complex backpropagation algorithm. *Signal Processing*, 40:967–969, 1992.

[3] Peter Bloomfield. *Fourier Analysis of Time Series: An Introduction.* John Wiley and Sons, Inc., Canada, 1976.

[4] Laiwan Chan. Connection reduction of the recurrent networks. In *Proceeding of ICONIP'95, Beijing*, volume 2, pages 813 – 816, 1995.

[5] Laiwan Chan and Fung-Yu Young. Ring-structured recurrent neural network. In *World Congress on Neural Networks 1993, Portland*, volume 4, pages 328–331, 1993.

[6] Jerome Connor, Les E. Atlas, and Douglas R. Martin. Recurrent networks and NARMA modeling. In John E. Moody, Steve J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 301–308. Morgan Kaufmann Publishers, Inc., 1992.

[7] J.T. Connor, R.D. Martin, and L.E. Atlas. Recurrent neural network and robust time series prediction. *IEEE Trans on Neural Networks*, 5(2):240–254, 1994.

[8] J.L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[9] George M. Georgiou and Cris Koutsougeras. Complex domain backpropagation. *IEEE Trans. on Circuits and Systems II : Analog and Digital Signal Processing*, 39(5):330–334, 1992.

[10] C.L. Giles, G.Z. Sun, H.H. Chen, Y.C. Lee, and D. Chen. Higher order recurrent networks and grammatical inference. In D.S.Touretzky, editor, *Advances in Neural Information Processing Systems, 2*, pages 380–387. Morgan Kaufmann Publishers, 1990.

[11] Akira Hirose and Hirofumi Onishi. Proposal of relative-minimization learning for behavior stabilization of complex-valued recurrent neural networks. *Neurocomputing*, 24:163–171, 1999.

[12] Edward Ho and Laiwan Chan. How to design a connectionist holistic parser ? *Neural Computation*, 11(8):1995–2016, 1999.

[13] George Kechriotis and Elias S. Manolakos. Training fully recurrent neural networks with complex weights. *IEEE Trans. on Circuits and Systems II : Analog and Digital Signal Processing*, 41(3):235–238, 1994.

[14] Tan Lee, P.C. Ching, and L.W. Chan. Recurrent neural networks for speech modeling and speech recognition. In *ICASSP-95, Proceedings of IEEE Int'l Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages 3319–3322, 1995.

[15] T. Nitta. An extension of the back-propagation algorithm to complex numbers. *Neural Networks*, 10:1391–1415, 1997.

[16] T. Robinson and F. Fallside. A recurrent error propagation network speech recognition system. *Computer Speech and Language*, pages 259–274, 1991.

[17] A.C. Tsoi and A. Back. Locally recurrent globally feedforward networks, a critical review of architectures. *IEEE Trans on Neural Networks*, 5(2):229–239, 1994.

[18] P.J. Werbos. *The roots of backpropagation : from ordered derivatives to neural networks and political forecasting.* John Wiley and Sons, Inc, 1994.

[19] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.