Practical Recurrent Learning (PRL) in the Discrete Time Domain

Mohamad Faizal Bin Samsudin, Takeshi Hirose and Katsunari Shibata
Department of Electrical and Electronic Engineering,
Oita University, 700 Dannoharu, Oita 870-1192 Japan
Email: shibata@cc.oita-u.ac.jp

Abstract. One of the authors has proposed a simple learning algorithm for recurrent neural networks, which requires computational cost and memory capacity in practical order O(n^2)[1]. The algorithm was formulated in the continuous time domain, and it was shown that a sequential NAND problem was successfully learned by the algorithm. In this paper, the authors name the learning "Practical Recurrent Learning (PRL)", and the learning algorithm is simplified and converted in the discrete time domain for easy analysis. It is shown that sequential EXOR problem and 3-bit parity problem as non linearly-separable problems can be learned by PRL even though the learning performance is often quite inferior to BPTT that is one of the most popular learning algorithms for recurrent neural networks. Furthermore, the learning process is observed and the character of PRL is shown.

Keywords: Recurrent Neural Network (RNN), Supervised Learning, Practical Recurrent Learning (PRL), BPTT, Short-Term Memory

1 Introduction

When we think of the higher functions in humans, such as logical thinking, conversation, and so on, it is easily noticed that memory plays an important role in the functions. Accordingly, it is expected that the need for the RNN is going to grow drastically in the near future as the increase of the desire to the higher functions.

Conventionally, there are two popular learning algorithms for recurrent neural networks that have been proposed. One is BPTT (Back Propagation Through Time)[2] and the other one is RTRL[3] (Real Time Recurrent Learning). In BPTT, all the past states of the network are stored using $O(nT)$ of memory where n is the number of neurons and T is the present time step, and the learning is done by tracing back to the past using the memory. The order of the computational cost is $O(n^2T)$. The traced-back time step is often truncated at a constant number when T becomes large, but it is difficult to know the sufficient number of steps. On the other hand, in RTRL, the influence of each connection weight to the output of each neuron is kept in $O(n^3)$ of memory, and the order of the computation of the influence is as large as $O(n^4)$. BPTT is not practical in the meaning that the learning should be done with tracing back to the past. Even though the special hardware is developed, iteration of learning for the traceback is necessary. RTRL is not practical in the meaning that the required order $O(n^3)$ in the memory capacity and $O(n^4)$ in the computational cost are larger than $O(n^2)$ that is the order of the number of connections in a neural network. Even

though each connection has some memory, a memory on the connection should have $O(n)$ size, that means that the size of each memory should be larger according to the size of the neural network.

S. Hochreiter and J. Schmidhuber have proposed a special network architecture that has some memory cells. In each memory cell, there is a linear unit with a fixed weight self-connection that enables constant, non-vanishing error flow within the memory cell[4]. They used a variant RTRL and only $O(n^2)$ of computational cost is required. However, special structure is necessary and it cannot be applied to the general recurrent neural networks.

Therefore, a practical learning algorithm for the general recurrent neural networks that need $O(n^2)$ or less memory and $O(n^2)$ or less computational cost is strongly required. Then Practical Recurrent Learning (PRL) was proposed in the continuous time domain. In this paper, PRL is simplified and converted in the discrete time domain for easy analysis, and the learning performance is compared to BPTT.

2 Practical Recurrent Learning (PRL)

Here, PRL is explained using an Elman-type recurrent neural network as shown in Fig.1.

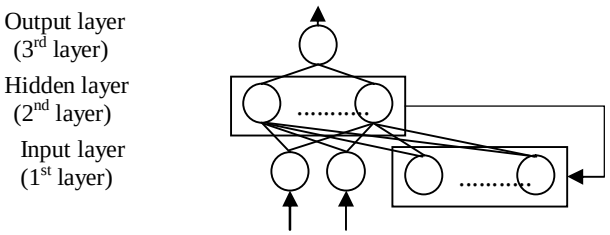

Fig.1 An Elman-type recurrent neural network

2.1 PRL in the continuous time domain[1]

This section describes roughly about PRL in the continuous time domain proposed in [4]. The forward calculation is the same as the conventional neural network that means that each hidden or output neurons calculate the weighted sum of the inputs and then non-linear function f is applied to get the output. Here, the sigmoid function whose value range is from -0.5 to 0.5 is used. In the output layer, the error signal is calculated by

$$\delta_j^{(3)} = Tr_j(t) - x_j^{(3)}(t) \tag{1}$$

where Tr : training signal, $x_j^{(3)}$:output of the output unit. Differing from the regular BP, the derivative of the output function $f'^{(3)}_j$ is not included. As well as the regular BP, the error signal in the hidden layer $\delta_i^{(2)}$ is calculated from the $\delta_j^{(3)}$ in the upper layer as described by the following equations.

$$\delta_i^{(2)} = \sum_j v_{ji} \delta_j^{(3)}(t) \tag{2}$$

$$\frac{d}{dt} v_{ji} = \left(w_{ji}(t)^{(3)} f'\left(S_j^{(3)}(t)\right) - v_{ji} \right) \left| \frac{d}{dt} x_j^{(3)}(t) \right| \tag{3}$$

where $w_{ji}^{(3)}$: connection weight (*i*th hidden unit - *j*th output unit) , $S_j^{(3)}$: the net value of the *j*th neuron in the output layer. *f*' is included in this equation on behalf that *f*' disappears in Eq. (1) in order to use *f*' when the output changed.

Then, in order to modify the value of weight without tracing back to the past, it is considered that the following information should be held.

(a) the latest outputs of pre-synaptic neurons,
(b) the outputs of pre-synaptic neuron that changes recently among all the inputs to the post-synaptic neuron,
(c) the outputs of the pre-synaptic neuron that caused the change of the post-synaptic neuron's output.

Corresponding to the (a),(b),(c), three variables *p(t), q(t), r(t)* that hold the past information in various ways are introduced and they are always modified according to the following differential equations.

$$\tau_j \frac{d}{dt} p_{ji}(t) = -p_{ji}(t) + x_i(t) f'\left(S_j(t)\right) \tag{4}$$

$$\frac{d}{dt} q_{ji}(t) = \left(x_i(t) f'\left(S_j(t)\right) - q_{ji}(t) \right) \sum_i \frac{d}{dt} x_i(t) \tag{5}$$

$$\frac{d}{dt} r_{ji}(t) = \left(x_i(t) f'\left(S_j(t)\right) - r_{ji}(t) \right) \left| \frac{d}{dt} x_j(t) \right| \tag{6}$$

Using the three variables, each connection weight is modified. The following equation is an example but the details can be seen in [1].

$$dw_{ji}(t) = \left(p_{ji}(t) + q_{ji}(t) + r_{ji}(t) \right) \delta_j(t) \tag{7}$$

Among the three variables, $r_{ji}(t)$ is considered to be a particularly important variable with respect to the learning of a problem that needs the past information before a long time lag. Fig.2 shows an example of the temporal change of the variable $r_{ji}(t)$ according to the input signal $x_i(t)$ and the output signal $x_j(t)$. As shown in Fig.2, it is the important character that $r_{ji}(t)$ holds the information about the output of the pre-synaptic neuron that caused the change of the post-synaptic neuron's output. This variable ignored the inputs while the output did not change. Accordingly the variable is expected to keep past and important information without tracing back to the past.

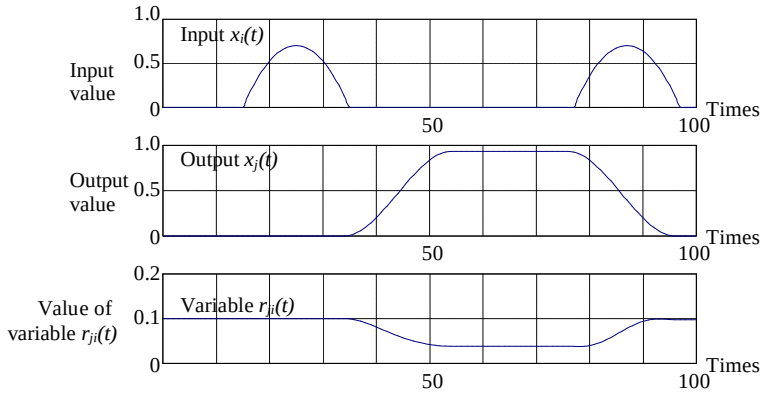

Fig.2 An example of the variable $r_{ji}(t)$ transition. From equation (11), variable $r_{ji}(t)$ integrates the value of input $x_i(t)$ when the output $x_j(t)$ changes, and holds the information of the previous state when the output does not change.

2.2 PRL in the discrete time domain

In order to make the analysis of PRL learning easy, PRL learning method in the discrete time domain is introduced here. The method of learning is similar to the conventional Back Propagation method in the meaning that each connection weight are modified according to the product of the propagated error signal δ of the post-synaptic(upper) neuron and the signal that represents the output x_i of the pre-synaptic(lower) neuron. Furthermore, to make the learning process become simple, conventional BP method is used for the learning of the connection weights between the hidden layer and the output layer and PRL learning method is used only between the input layer and the hidden layer.

In the output layer, the error signal $\delta_j^{(3)}$ is calculated as

$$\delta_j{}^{(3)} = \frac{\partial E(t)}{\partial S_j{}^{(3)}(t)} = \frac{\partial E(t)}{\partial x_j{}^{(3)}(t)} \cdot \frac{\partial x_j{}^{(3)}}{S_j{}^{(3)}} = \left(Tr_j(t) - x_j{}^{(3)}(t)\right) f'\left(S_j{}^{(3)}(t)\right) \quad . \tag{8}$$

Same as the conventional Back Propagation method, the modification of connection weights are calculated by

$$\Delta w_{ji}{}^{(3)} = \eta \delta_j{}^{(3)} x_i{}^{(2)} \, . \tag{9}$$

Each neuron in the hidden layer is trained by PRL and signal $\delta_j^{(2)}$ is calculated as

$$\delta_j{}^{(2)} = \sum_k \delta_k{}^{(3)} \cdot w_{kj}{}^{(3)}(t) \, . \tag{10}$$

From the equation above, *f'(t)* is not multiplied as the conventional BP method because *f'(t)* is included in the variable $r_{ji}(t)$ as shown in equation (11). Considering

that variable $r_{ji}(t)$ does not changed when the output does not changed, and integrates the input's value when the output changes, it is calculated as

$$r_{ji}(t) = r_{ji}(t-1)\left(1 - \left|\Delta x_j^{(2)}(t)\right|\right) + x_i^{(2)}(t) f'\left(S_j^{(2)}(t)\right)\left|\Delta x_j^{(2)}(t)\right| \tag{11}$$

where $\Delta x_j(t) = x_j(t) - x_j(t-1)$. Then, the modification of each connection weight in the hidden layer is calculated using only the variable r_{ji} by

$$\Delta w_{ji}(t)^{(2)} = \eta \delta_j^{(2)} r_{ji}(t) . \tag{12}$$

3 Simulation of EXOR and 3-bit parity problems

3.1 Simulation of EXOR problem

From the previous work[1], it was shown that a sequential NAND problem could be learned by PRL, but a sequential EXOR problem could not be learned. Here, sequential EXOR problem in a fix pattern order was tried to be learned by PRL in the discrete time domain. At first, the sequential EXOR logic function is explained. EXOR problem is a logical operation on two operands that results in a logical value of 1 if and only if exactly one of the operands has a value of 1 and the other has a value of 0. The network architecture used in this paper is the same as shown in Fig.1 besides it contains 1 output, 20 hidden units and 3 input signals. The input 1 is considered as a signal to distinguish the starting time of a pattern presentation and it is always 1 at t=0. As shown in Table 1, the value of 0 or 1 is entered to the input 2 at t=5 and the input 3 at t=15. At the other times, the signal is always 0. The training signal is given when t=time_lag (from the starting time to the time when the training signal was given) and the time_lag is set to 20 unless mentioned particularly. Parameter setup is shown in Table 2. As shown in Table 2, we used value 4.0 for initial connection weight for self-feedback connection to prevent the propagated error value from diverging or vanishing in BPTT method considering that the maximum derivative of output function is 0.25. All the valuables r are reset to 0 at t=0.

Table 1 The timing of inputs and training signal in the learning of one pattern

Time, t	0	1~4	5	6~14	15	16~time_lag	time_lag
Input 1	1	0	0	0	0	0	Training
Input 2	0	0	0 or 1	0	0	0	signal was
Input 3	0	0	0	0	0 or 1	0	given

Table 2 Parameter setup

Initial weight value for self-feedback	4.0
Initial weight value for the other feedback	0.0
Initial weight value (input layer-hidden layer)	Random number (1.0~1.0)
Initial weight value (hidden layer-output layer)	0.0
Termination condition	30000 iteration(1 pattern for 1 iteration)

3.1.1 Simulation result

Table 3 shows the simulation result when EXOR problems was tried. Successful learning is defined as the state that the difference from the difference from the training signal is less than 0.1 for the last 4 iterations before the end of the learning.

From the simulation results, it is shown that sequential EXOR problem as a non linearly-separable problem can be learned by PRL successfully as well as the case of BPTT. Moreover, we recognized that the learning performance for both learning methods has been improved when the learning rate for the feedback connections are smaller than the learning rate for the other connections in the network.

Table 3 Simulation result when the learning rates on the network were varied.

Learning rate	Learning rate for feedback connections	Success Rate PRL (/100times)	Success Rate BPTT (/100times)
1.0	1.0	1	6
	0.5	12	25
	0.1	96	95
	0.05	100	100
	0.01	100	100
0.5	0.5	42	27
	0.1	94	92
	0.05	100	98
	0.01	100	100
0.1	0.1	94	84
	0.05	94	82
	0.01	60	75

In addition, Table 4 summarizes the result of comparison for both methods when we exceeded time_lag to 100, but the timing of inputs is the same as shown in Table.1. In terms of learning ability, the conventional BPTT performs better than PRL even though the time for training by PRL is far smaller than BPTT.

Table 4 Simulation result when time lag is exceeded to 100

Time Lag	Success Rate PRL (/100times)	Success Rate BPTT (/100times)	Training time PRL (sec)	Training time BPTT(sec)
20	100	100	4	8
40	90	100	9	19
60	79	100	13	33
80	70	100	17	49
100	68	100	22	69

The sequential EXOR problem as a non linearly-separable problem can be learned successfully to some extent by the PRL in the discrete time domain rather than continuous counterpart. The reason of failure for the learning in the continuous time domain is not clear, but maybe the difficulty of setting the training signal. The value of the training signal was not given at a moment, but a shape of training signal for some duration was given in [1].

3.2 Simulation of 3-bit parity problems

This section presents the learning performances of the PRL in comparison to the BPTT in a sequential 3-bit parity problem in the random pattern order. In the 3-bit parity problem, the training signal is -0.4 when the number of signals whose value is 1 in 3 given inputs except for the input 1 is even, and the training signal is 0.4 when the number is odd. It is considered that the task is more difficult than EXOR because the number of inputs is larger and it might be difficult for the variable r_{ji} to keep the past information. We used the same network architecture (refer to Fig.1), but there are 4 input signals that are 1 input as a starter signal and 3 inputs that is used to calculate the parity signal. The input 1 is entered when t=0 and the value is always 1. Time_lag is set to 20 and the input 2, 3 and 4 are set to enter at t=5, 10 and 15 respectively and the value is chosen randomly from 0 and 1. Parameter setup was the same as in the previous section, but the termination condition is that the state with the squared error is less than 10^{-3} continues for 100 pattern of learning. Furthermore, the random pattern order is employed here to eliminate the possibility of memorizing the pattern order during the learning

3.2.1 Simulation Results

The result of simulation for the 3-bit parity problem in random pattern order is shown in Table 5. Even though no traceback is done in PRL learning, this 3-bit parity problem is learned by PRL to some extent. However, the BPTT outperforms PRL for success rates and average number of iterations.

Table 5 The comparison result of learning success rate and average number of iterations.

Random pattern order					
Learning rate	Learning rate for feedback connections	Learning success rate (/100times)		Average number of iterations	
		PRL	BPTT	PRL	BPTT
1	0.001	75	100	133003	17494
	0.003	81	100	119909	11393
	0.01	62	100	107270	7929
	0.03	26	96	119020	7710
	0.1	30	1	108248	6033

Here, we focused on the big difference on the average number of iterations to find the reason of inferiority. Firstly, the transition of the output neuron's output just after the learning process begins is observed as shown in Fig.3 for both methods. In order to make a comparison, the initial values of connection weights and pattern order are same for the both methods. Fig.3 shows that there is a big difference in the transition of output when the value of input 2 is 1 between both methods. The output seemed to increase drastically due to the presenting of the input 2 in the case of BPTT, but in the case of PRL, little change of the output is seen despite the presence of input 2. For example, the transition of the output in the case of pattern P5 and pattern P4 where a circle is put in Fig.3 show the difference between both methods.

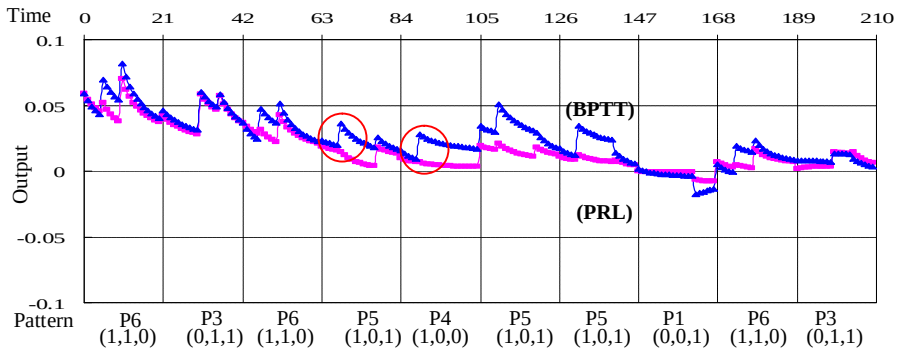

Fig.3. The transition of output in PRL and BPTT methods. The horizontal axis indicates time and the vertical axis indicates output's value. P3 indicates Pattern 3 for example.

Fig.4 shows the connection weights from hidden neurons to the output neuron and the output of each hidden neuron when input 2 is 1 at t=5 in the case of pattern P5. The initial value of connection weights from hidden neuron to output is set to 0. As shown in Fig.4, the sign of the connection weight between the hidden 20 and the output neuron is positive in the case of PRL while it is negative in the case of BPTT. As a result, the output is almost the same in the case of PRL while increases in BPTT.

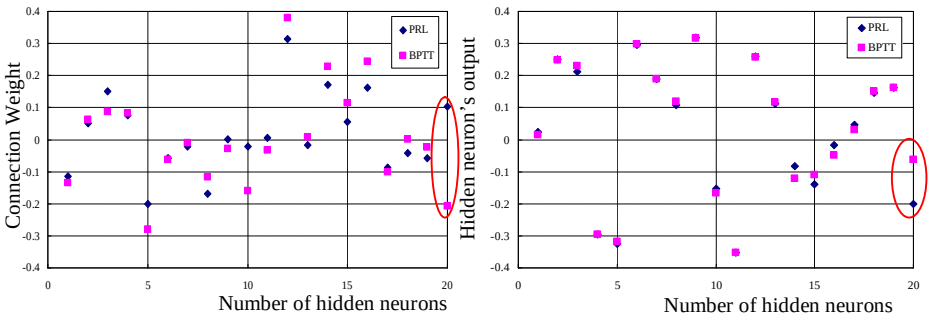

Fig.4. Connection weight from the hidden layer to the output layer and the output of hidden neurons when t=5.

Then, the change of the connection weight from input 1, 2, 3 and 4 to hidden 20 is observed. As shown in Fig.5, it is noticed that the transition of connection weights from input 1, 2, 3 and 4 to hidden 20 in the case of PRL is far smaller compared to the case of BPTT. Considering that the sign of the connection weight from hidden 20 to output are opposite between both methods, it is not a problem that the change of connection weights from the input 1 to the hidden 20 in PRL is also going to the opposite direction of the case in BPTT.

Then, the transition of variable *r* in one cycle for the pattern P5 is shown in Fig.6. The values are so small, but as expected, they changed at the time when the corresponding input is 1. Even though the values decreased a little bit when another input exists, they keep the information until the end phase of the learning process.

In order to compensate the small variable *r* and to promote the change of the connection weights from input 1, 2, 3, and 4 to hidden neurons, the learning rate for the connection is raised up. The result of simulation is shown in Table 6.

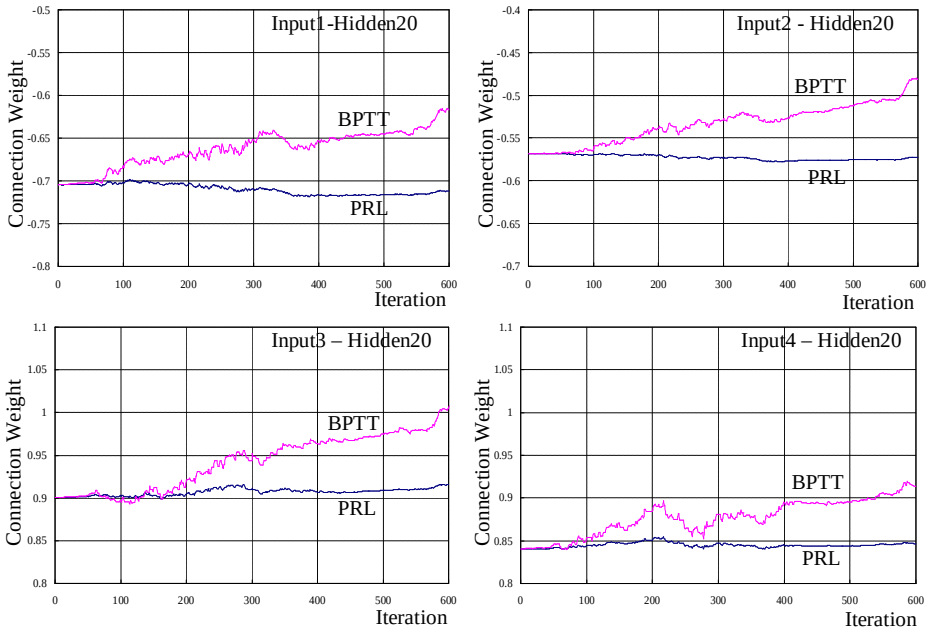

Fig.5. The change of the connection weight from input 1, 2, 3, and 4 to hidden20 for both methods at the early phase of learning process.

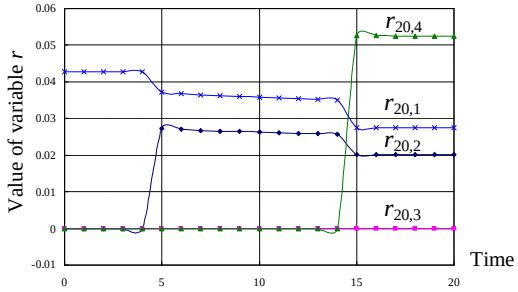

Fig.6. The value of variable *r* from input 1, 2, 3, and 4 to hidden 20 in the case of P5

Table 6 The comparison result of learning success rate and average number of iteration

Learning rate	Learning rate between input 1, 2, 3, 4 to hidden neurons	Learning rate for feedback connections	Learning success rate (/100times)		Average number of iterations	
			PRL	BPTT	PRL	BPTT
1	3	0.001	53	100	110107	22530
		0.003	59	100	96847	12943
		0.01	58	100	83112	7360
		0.03	47	100	86625	5720
		0.1	50	6	87693	4670

As shown in Table 6, even though the learning rate of input 1, 2, 3, and 4 to hidden neurons is set to be higher, BPTT still outperforms PRL in the viewpoints of both success rates and average number of iterations. Table 6 shows the characteristic of BPTT where the learning will become more successful when the learning rate is set to be small. However, the PRL does not have the same characteristic as BPTT because the learning success rate for PRL does not depend on the learning rate for the feedback connections. More experiments and analysis is required to examine whether the learning performance of the practical recurrent learning can be improved or not.

Conclusion

By formulating PRL learning method in the discrete time domain, it could be shown that sequential EXOR problem and 3-bit parity problem as non linearly-separable problem could be learned by PRL even thought PRL is practical as opposed to BPTT and RTRL with respect $O(n^2)$ of memory and $O(n^2)$ of computational cost. However the learning performance of PRL is inferior to BPTT. A big difference is seen in the weight transition between PRL and BPTT even though the variable r keeps the past information as expected. More additional analysis and experiment is needed to develop and improve the performance of this learning method.

Acknowledgment

A part of this research was supported by JSPS Grant in-Aid for Scientific Research #15300064 and #19300070.

References

[1] Katsunari Shibata, Yoichi Okabe, Koji Ito, “Simple Learning Algorithm for Recurrent Networks to Realize Short-Term Memories”, Proc. of IJCNN(Int’l Joint Conf. on Neural Network)’98, pp.2367-2372(1998).

[2] Rumelhart D.E., Hinton G.E. and Williams R.J.: "Learning internal representations by errorpropagating", Parallel Distributed Processing, Vol. 1, MIT Press, pp.318-362 (1986)

[3] Williams, R. J. and Zipser, D. , “A learning algorithm for continually running fully recurrent neural network”, Neural Computation, Vol.1, pp.270-280(1989)

[4] Sepp Hochreiter, Jurgen Schimidhuber, “Long Short Term Memory” Neural Computation, Vol 9, pp.1735-1780(1997)