Lattice Neural Networks for Incremental Learning

Daisuke Uragami[1], Hiroyuki Ohta[2] and Tatsuji Takahashi[3]
[1]School of Computer Science, Tokyo University of Technology, 1404-1 Katakuramachi, Hachioji City, Tokyo 192-0982, JAPAN
E-mail: dduragami@gmail.com
[2]Department of Physiology, National Defense Medical College, 3-2 Namiki, Tokorozawa, Saitama 359-8513, Japan.
[3]Division of Information System Design, School of Science and Technology, Tokyo Denki University, Hatoyama, Hiki, Saitama 350-0394, Japan.

Abstract
In incremental learning, it is necessary to conquer the dilemma of plasticity and stability. Because neural networks usually employ continuously distributed representation for state space, learning newly added data affects the existing memories. We apply a neural network with algebraic (lattice) structure to incremental learning, that has been proposed to model information processing in the dendrites of neurons. It has been proposed as a mathematical model of information processing in the dendrites of neurons. Because of the operation 'maximum' in lattice algebra weakening the continuously distributed representation, our proposed model succeeds in incremental learning.
Keywords: Stability-Plasticity Dilemma, Distributed Representation, Dendrite, Anticipation.

1 Introduction

There are two aspects of anticipation (Rosen, 1985; Dubois 1998) in learning. One is generalization. It is to presume a distribution or a function from existing data for predicting new data. Another aspect is incremental learning (Giraud-Carrier, 2000). In incremental learning, because the coordination of existing learnt data and newly added data is needed, learning mechanism must assume data addition.

First we explain what is incremental learning and why it is hard. Then we show how our model solves the problem.

1.1 Incremental Learning

Incremental learning is a task to learn a data set after learning another data set sequentially (online). In the learning of the second data set, the first data set is neither learnt nor referred. In ordinary neural networks, because of the continuously distributed representation (Rumelhart, Hinton and Williams, 1986), the incremental learning destroys the existing learning.

Figure 1 is a simple example exhibiting the difficulty of incremental learning. It shows the destruction of old learning by new learning. With the connection in the left, it outputs only 'A' from the input 'a'. Now the input 'a' and the output 'A' is the existing memory. Then, by learning the correspondence from the input 'b' to the output 'B', the connection becomes as in the middle. As the result, as in the right figure, the 'B' is outputted from 'a'. Such an aspect leads to the dilemma of stability or plasticity. One of the principal factors is that all the neurons in the state layer contribute to the output. It is that memory is distributed to the neurons.

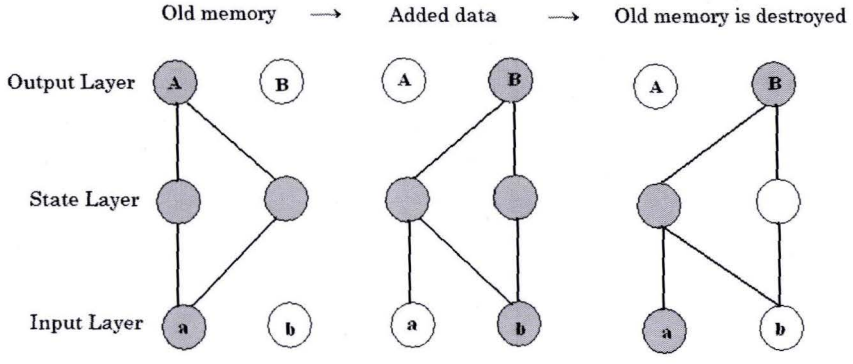

Figure 1: Destruction of memory by incremental learning.

As a solution to the dilemma, Ohta and Gunji (2006) proposed a model that employs winner-take-all method and negative reinforcement. Winner-take-all is an activation algorithm proposed in the study of self-organizing maps by Kohonen (2001). In this algorithm, only the most excited neuron in the state layer fires. As a result, distributed representation is weakened. On the other hand, negative reinforcement is a learning algorithm proposed by Chialvo and Bak (1999). In this algorithm, the teacher signal is not explicitly given. The depression is given in failure to the connections that

contributed to the firing. It is negative feedback; the strengths of active synapses get reduced if mistakes are made, otherwise no changes occur. By this algorithm, together with weight conservation and pre-synaptic inhibition, the unused firing pathways come to be used. As the result, existing memory can avoid getting overwritten. The model by Ohta and Gunji succeeds in incremental learning of time series data.

1.2 Lattice Neural Networks

In this study, we apply lattice neural networks (LNN) to incremental learning tasks. LNN proposed by Ritter and Urcid (2003) model the actions in the dendrites in neurons by lattice algebraic operations. As in the Figure 2, each neuron in the output layer has several dendrites. The dendrites can work as the state layer in an ordinary neural network. Each neuron in the input layer connects to all the dendrites. Every connection consists of a pair of excitatory and inhibitory synapses as in the left figure. (The neurons mediating inhibitory connection as in the right figure are here just omitted.) The operation between dendrites is the 'join' operation in lattice algebra. It is actually maximum operation, because, as we mention later, the lattice structure we treat is totally ordered, an interval of real numbers. Ritter and Urcid (2007) have applied their LNN to associative memory tasks by batch learning but not to online learning.

In this study, we modify LNN for online learning. We call the new model LNNI (Lattice Neural Networks for Incremental learning). In the model, the operation 'join' (denoted by '$\vee$') works instead of winner-take-all. To prevent the added data from overwriting, negative reinforcement and the plasticity of the dendrites are exploited.

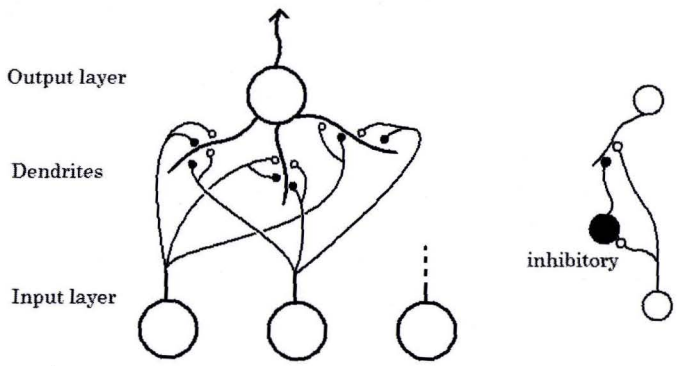

Figure 2: Lattice neural network that employs dendritic computing.

2 Model

2.1 Activation Algorithm

We explain about the firing algorithm of LNNI. It is, in other words, how to calculate the output from an input.

$$\boldsymbol{y} = f(\boldsymbol{x}\,;\, W) \tag{1}$$

As in eq.1, the function with the parameter W calculates the output $\boldsymbol{y} = (y_1, \cdots, y_j, \cdots, y_M)$ from the input $\boldsymbol{x} = (x_1, \cdots, x_i, \cdots, x_N)$. $W = [w_{ikj}^l]$ is the parameter representing the connection weights. It is a 4-dimensional matrix. There are four indices: $i = 1, \cdots, N$ is for the input nodes, $j = 1, \cdots, M$ is for the output nodes and $k = 1, \cdots, K$ is for the dendrites of each output nodes, and l is for discriminating excitory and inhibitory connection. N, M and K are respectively the number of the input nodes, output nodes and the dendrites of each output node. Although the number of the dendrites can be different in the respective output nodes, we assume that each neuron in the output layer uniformly has K dendrites for simplicity. If $l = 1$, the connection is excitory and if $l = 0$, it is inhibitory. The detail of the function f is as follows.

$$\varphi_{jk}(\boldsymbol{x}) = \Lambda_{i=1}^{N} \Lambda_{l\in\{1,0\}} r^l (x_i + w_{ikj}^l + \theta^l) \tag{2}$$

$$\tau_j(\boldsymbol{x}) = \mathrm{V}_{k=1}^{K} \varphi_{jk}(\boldsymbol{x}) \tag{3}$$

$$y_j = g(\tau_j(\boldsymbol{x})) = \begin{cases} 1 & \text{for } \tau_j(\boldsymbol{x}) > 0 \\ 0 & \text{for } \tau_j(\boldsymbol{x}) \leq 0 \end{cases} \tag{4}$$

$\varphi_{jk}(\boldsymbol{x})$ is the result of calculation at the k-th dendrite of the j-th output neuron. $\tau_j(\boldsymbol{x})$ is the maximum of the $\varphi_{jk}(\boldsymbol{x})$ at the j-th output neuron. If $\tau_j(\boldsymbol{x}) > 0$, the j-th output neuron fires. If not, it does not fire. $r^1 = 1$ and $r^0 = -1$. $\theta^1 = -0.5$ and $\theta^0 = -1.5$. θ^l is a constant just for alignment that makes easier to see the relation between input-output and w_{ikj}^l. As briefly mentioned earlier, in this study, the underlying lattice structure is an interval of real numbers that is a kind of chain and a totally ordered set. Therefore, 'V' and 'Λ' respectively denote maximum and minimum.

2.2 Examples

2.2.1 Example 1

Example 1 is the simplest experiment. There is only one input, output neuron and its dendrite ($N = M = K = 1$). $\tau(\boldsymbol{x})$ is the following.

$$\tau(\boldsymbol{x}) = (+1) \times (x + w^1 - 0.5) \wedge (-1) \times (x + w^0 - 1.5) \tag{5}$$

We set $w^1 = w^0 = 0$. y=1 if $0.5 < x < 1.5$. Otherwise, y=0. Actually, substituting 1 to x, we get y=1 as follows.

$$\begin{aligned} \tau(1) &= (+1) \times (1 + 0 - 0.5) \wedge (-1) \times (1 + 0 - 1.5) \\ &= (0.5) \wedge (0.5) \\ &= 0.5 \end{aligned} \tag{6}$$

$$y = g(\tau(1)) = g(0.5) = 1 \tag{7}$$

Similarly, substituting 1 to x, $\tau(0) = 0$ and $y = 0$.

$$\begin{aligned} \tau(0) &= (+1) \times (0 + 0 - 0.5) \wedge (-1) \times (0 + 0 - 1.5) \\ &= (-0.5) \wedge (1.5) \\ &= -0.5 \end{aligned} \tag{8}$$

$$y = g(\tau(0)) = f(-0.5) = 0 \tag{9}$$

In Figure 3 (left), the change in the input-output relation according to the value of w^1 and w^0. The interval bounded by the arrows is the range of x that makes τ positive, hence $y = 1$ in the interval.

2.2.2 Example 2

In the second example, we show that we can construct an XOR gate with two inputs and two dendrites ($N = 2$, $M = 1$, K=2). We set W and $\tau(\boldsymbol{x})$ as follows.

$$\begin{bmatrix} w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \end{bmatrix} = \begin{bmatrix} w_{11}^0 & w_{12}^0 \\ w_{21}^0 & w_{22}^0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{10}$$

$$\begin{aligned} \tau(x_1, x_2) = & \left\{ \begin{matrix} (+1) \times (x_1 - 0.5) \wedge (-1) \times (x_1 - 1.5) \\ \wedge (+1) \times (x_2 + 0.5) \wedge (-1) \times (x_2 - 0.5) \end{matrix} \right\} \\ & \vee \left\{ \begin{matrix} (+1) \times (x_1 + 0.5) \wedge (-1) \times (x_1 - 0.5) \\ \wedge (+1) \times (x_2 - 0.5) \wedge (-1) \times (x_2 - 1.5) \end{matrix} \right\} \end{aligned} \tag{11}$$

From the input (1, 0), 1 is outputted.

$$\begin{aligned} \tau(1,0) &= \{(+0.5) \wedge (+0.5) \wedge (+0.5) \wedge (+0.5)\} \\ &\quad \vee \{(+1.5) \wedge (-0.5) \wedge (-0.5) \wedge (+1.5)\} \\ &= \{+0.5\} \vee \{-0.5\} \\ &= +0.5 \end{aligned} \tag{12}$$

$$y = g(\tau(1,0)) = g(+0.5) = 1 \tag{13}$$

Similarly, the model outputs 0 from input (1, 1), 0 from input (0, 1) and 0 from input (0, 0). In the plot, the gray square regions are where $y = 1$. Each square is a region where the calculated value on a dendrite becomes larger than 0. It is how to adjust the connection weights between neurons.

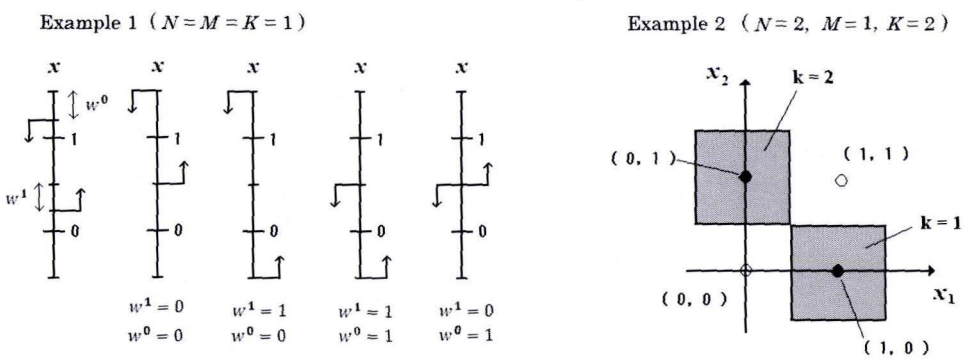

Figure 3: The activation regions.

2.3 Learning algorithm

Next we explain the learning algorithm of LNNI. Now the calculated value of the system from input $\boldsymbol{x} = (x_1, \cdots, x_i, \cdots, x_N)$ is $\boldsymbol{y} = (y_1, \cdots, y_j, \cdots, y_M)$. The correct answer is $\mathbf{y}' = (y'_1, \cdots, y'_j, \cdots, y'_M)$. Comparing $\boldsymbol{y}$ and $\boldsymbol{y}'$, the increase or decrease of W is determined as follows.

$$w^l_{ikj}(t+1) = w^l_{ikj}(t) + \Delta w^l_{ikj} \tag{14}$$

(L-1) For all j satisfying $y_j = 1$ and $y'_j = 0$
for all k satisfing $\varphi_{jk}(\boldsymbol{x}) > 0$
for all i

$$\Delta w^1_{ikj} = d^1_{\text{miss}}/\sigma_j < 0$$
$$\Delta w^0_{ikj} = d^0_{\text{miss}}/\sigma_j > 0$$

where σ_j is the number of k satisfied $\varphi_{jk}(\boldsymbol{x}) > 0$.

(L-2) For all j satisfying $y_j = 1$ and $y'_j = 1$
for all k satisfying $\varphi_{jk}(\boldsymbol{x}) > 0$
for all i satisfying $x_i = 1$

$$\Delta w^1_{ikj} = d^1_1/\sigma_j < 0$$
$$\Delta w^0_{ikj} = d^0_1/\sigma_j < 0$$

for all i satisfied $x_i = 0$

$$\Delta w^1_{ikj} = d^1_0/\sigma_j > 0$$
$$\Delta w^0_{ikj} = d^0_0/\sigma_j > 0$$

(L-3) If all $y_j = 0$, $K(t+1) = K(t) + 1$

The algorithm is divided into three cases. (L-1) means that if a firing of an output neuron was wrong, W is adjusted so that it uniformly closes the all related pathways. Here, the correct output itself is not given. (L-2) is that, if a firing of an output neuron was not wrong, W is adjusted so that the used pathway does not fire with the other inputs. (L-3) prescribes that, if none of output neurons fire, all the output neurons get one more dendrite added. If it is felt ad-hoc, a more natural method is to apply 'weight conservation' (Royer and Pare, 2003). However, our algorithm is sufficiently reasonable

by the plasticity of dendrites and in this way we can observe the learning process clearer. Note that the dendrite is uniformly added to all the output neurons. The addition is not restricted only to the output neuron corresponding to the correct output.

3 Simulation and Results

We have explained the firing and learning algorithm of LNNI. Hereafter we confirm that LNNI can execute incremental learning by simulation. The initial values of the parameters are the following. $w^1_{ikj}(t=0)=1$, $w^0_{ikj}(t=0)=0$. It is that the pathway is maximally open. In the course of learning, w^l_{ikj} can be larger than 1 or smaller or 0. Then w^l_{ikj} is set 1 or 0. They do not go beyond the interval from 0 to 1. The number of dendrites is one for each output neuron, in the initial setting ($K(t=0)=1$). The rates for learning are the following. $d^1_{\text{miss}}=-0.02, d^0_{\text{miss}}=0.02, d^1_1=-0.2, d^0_1=-0.2, d^1_0=0.1, d^0_0=0.05$. We execute three simulations. We cyclically input the samples. Comparing the system's output and the sample output, the weight W is adjusted. The initial setting described here is common to all the three simulations.

Figure 4 is the learning set used in all the experiments. One vertical column is a sample of input-output. Output is in the above and input is in the below. The number of inputs and outputs are both 5 ($N=5$, $M=5$), and there are 6 sets of inputs and outputs, discriminated by the index s. The characteristics of the input data set is that the inputs are partially overwrapped as in i=3 at s=2 and s=3. On the other hand, the characteristics of the output data set is that there are three in the row of j=1.

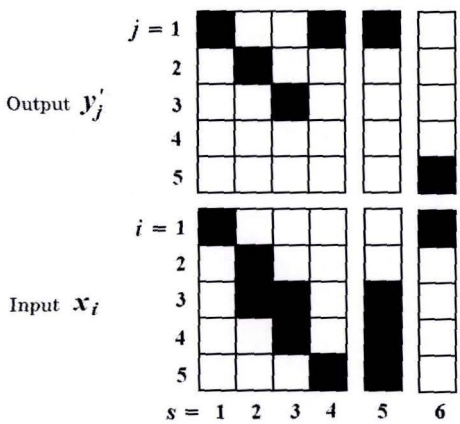

Figure 4: The learning set.

3.1 Experiment 1

This is how the learning goes. First the input samples from s=1 to s=4 is repeated 15 times, so it consumes 60 steps. Next the fifth sample (s=5) is iterated 40 times.

Figure 5 (top) shows the learning curve. The horizontal axis is time step and the vertical axis is the accuracy rate, the rate of the correct output by the system, among five samples. After t =20, it begins to correctly output to the sample of s=1, 2, 3. Before t=30, the second dendrite is generated. Although the correct rate becomes 0 once, but after the learning by the newly added second dendrite for s=4 sample, around t=50 it correctly responds to the sample s=1, 2, 3 and 4. At t=61, the incremental learning starts. Here again, by the third generation of dendrite, once the accuracy rate becomes 0, Then, because of the learning by the dendrite for the s=5 sample, finally the rate becomes 1.

Figure 6 (upper-left) shows how the weight W changes through the learning. It is how the dendrite is excited. The dendrite is of k=1 of the j=1 output neuron. The horizontal axis is time and the five plots correspond to the five inputs from i=1 to i=5. The white region is where it excites. At t=0, they get excited to all the inputs. Contrastingly, at t=100, they excite only to the input (1, 0, 0, 0, 0).

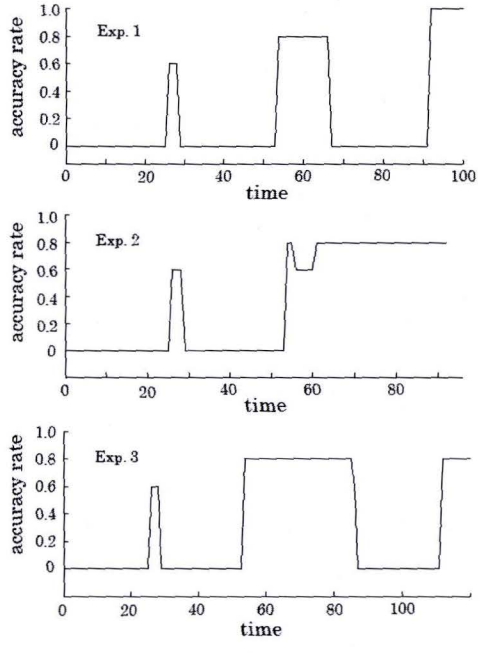

Figure 5: The learning curves.

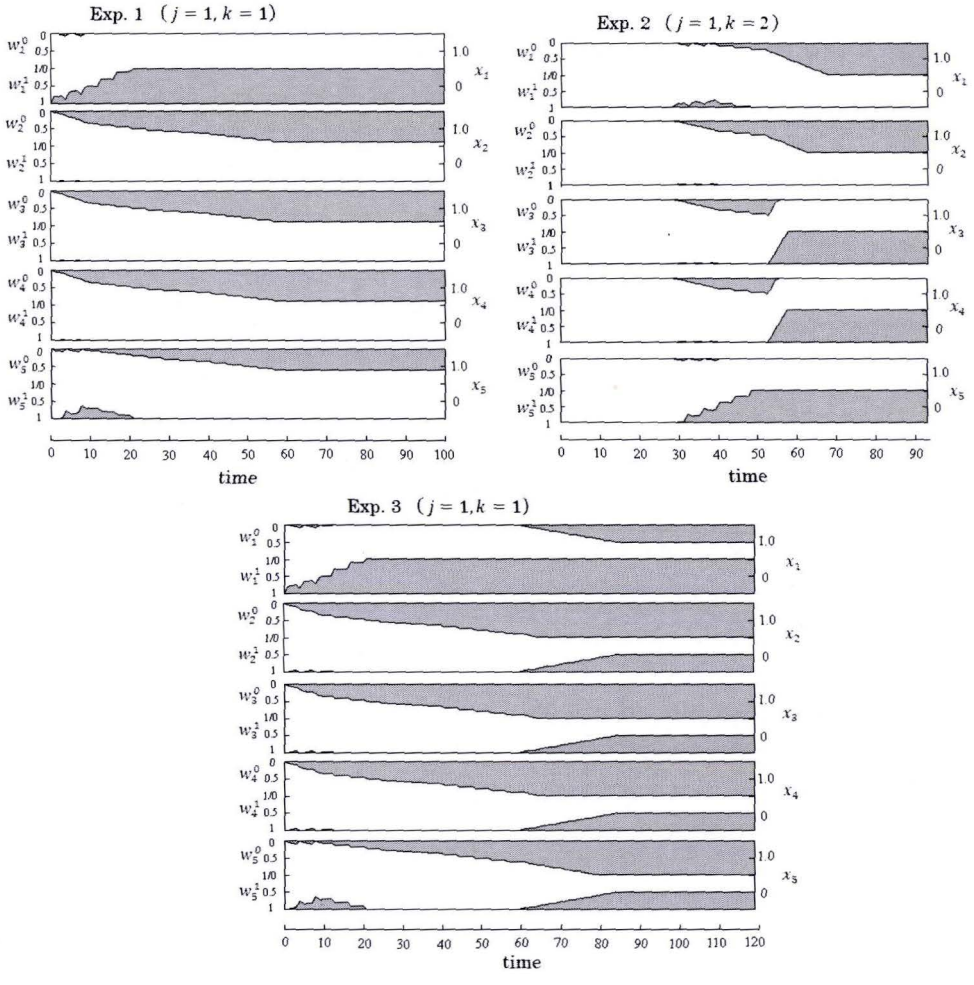

Figure 6: The evolution of the weight *W*.

3.2 Experiment 2

In the second experiment, the incremental learning beings earlier at t=53. The other conditions coincide with the one of the first experiment. It is when the system has learnt to correctly output to the first four samples. As the result, the existing learning result is overwritten. The correct rate does not get greater than 0.8 (Figure 5 middle).

We can see the overwriting by watching the time development of the weight W (Figure 6 upper right). In the center plot, for the i=3 input, before t=53, the result of learning is that it gets excited only to x_3=0. After t=53, the direction of the learning is changed so that it gets excited exclusively to x_3=1.

3.3 Experiment 3

In the third experiment, we confirm that the existing memory can be corrected by additional learning. First the input samples from s=1 to s=4 are repeated 15 times (60 steps). Next the sample s=6 is iterated. Note that the sample of s=6 has the same input as of s=1 but the output differs.

See Figure 6 (bottom). At the 60th step, the learning is finished so that excites only to the input s=1, (1, 0, 0, 0, 0). However, by additional learning, it excites to none of inputs. Afterwards, another dendrite is added and used for the input s=5. Figure 5 (bottom) shows the learning curve.

4 Discussion

The results by simulation can summarized as follows: (1) LNNI can execute incremental learning, (2) if the learning period is not long enough, it is overwritten by the new learning, and (3) the learning can be revised even if it has lasted long.

Now we review distributed representation. In distributed representation a concept is represented by a pattern of activity over a collection of neurons. Normally, the distributed representation is regarded as an antithesis to the grandmother cell representation.However, we consider that the problem is not in the dichotomy of distributed or not distributed. Although we normally believe that the representation is constructed only by the result of neuronal firing, we can choose another viewpoint to assume the potent input pattern on dendrite as implicit distributed representation. The operation 'join' (denoted by '$\vee$') in the proposed model extends the variability of possible input pattern. This 'implicit distributed representation' may be a principal subject of brain science.

5 Conclusion

In this article, we have reported that LNN can be refined to our LNNI for online learning, and that it can incrementally learn. It may be a touchstone to discuss distributed representation in the brain.

Acknowledgements

Preparation of this article was partially supported by the Cooperative Research Project Program of the Research Institute of Electrical Communication, Tohoku University.

References

Chialvo, D., Bak, P. (1999). Learning from mistakes. Neuroscience 90, pp.1137-1148.

Dubois, D. M. (1998). Introduction to Computing Anticipatory Systems. International Journal of Computing Anticipatory Systems 2, pp3-14.

Giraud-Carrier, C. (2000). A Note on the Utility of Incremental Learning. AI Communication 13, pp.215-223.

Kohonen, T. (2001). Self-organizing maps. New York: Springer.

Ohta, H., Gunji, Y.-P. (2006). Recurrent neural network architecture with pre-synaptic inhibition for incremental learning. Neural Networks 19, pp.1106-1119.

Ritter, G. X., Urcid G. (2003). Lattice algebra approach to single neuron computation. IEEE Trans on Neural Networks 14, No.2, pp.282-295.

Ritter, G. X., Urcid G, (2007). Learning in lattice neural networks that employ dendritic computing, Studies in Computational Intelligence 67, pp.25-44.

Rosen, R. (1985). Anticipatory Systems. Pergamon Press.

Royer, S., Pare, D. (2003). Conservation of total synaptic weight through balanced synaptic depression and potentiation. Nature 422, pp.518-522.

Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986). Learning internal representations by error propagation. Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations, Cambridge, MA, USA: MIT Press, pp.318-362.