Fifth IFSA World Congress (1993), 1005-1008

RECURRENT NEURAL NETWORKS

—What Do They Learn and How?—

Yoshiki UCHIKAWA, Haruhiko TAKASE
Tatsumi WATANABE, and Kazutoshi GOUHARA*

School of Engineering, Nagoya University, Nagoya, 464-01 Japan
**School of Engineering, Chubu University, Kasugai, 487 Japan*

Supervised learning of recurrent neural networks (RNNs) is discussed. First, we review the present state of art, featuring their major properties in contrast of those of the multilayer neural networks. Then, we concisely describe one of the most practical learning algorithms, i.e. back-propagation through time.

Revising the basic formulation of the learning algorithms, we derive a general formula to solve for the exact solution(s) of the whole connection weights w of RNNs. On this basis we introduce a novel interpretation of the supervised learning. Namely, we define a multi-dimensional Euclidean space, by assigning the cost function $E(w)$ and every component of w to each coordinate axis.

Since $E=E(w)$ turns up as a hyper surface in this space, we refer to the surface as ***learning surface***. We see that topological features of the learning surface are *valleys* and *hills*. Finally, after explicating that the numerical procedures of learning are equivalent to descending slopes of the *learning surface* along the steepest gradient, we show that a minimal value of $E(w)$ is the intersection of curved valleys.

1. WHY NOW RECCURENT NEURAL NETWORKS?

Multilayer neural networks (MNNs)[1] have been increasingly investigated both in theory and application, and it is known that they are incompatible with time-variant transformation from the input to the output signals. The incompatibility results from the governing equations of MNNs which do not involve time explicitly as a variable.

In recent years, to overcome such incompatiblity, several attempts have been reported, in which time-variant outputs were treated. Doya and Yoshizawa[2] developed a neural network (NN) and named it Adaptive Neural Oscillator, which works as the temporal pattern memory in the motor nervous systems to realize periodic motions such as walking and swimming. Sato et al.[3] trained an NN, naming it APOLLON, to generate voice waveforms which include fluctuations necessary to retain naturalness of speech sound.

In these applications of NNs more elaborate topologies were used than that of MNNs. They include feedback connections between neurons as well as a neuron function allowing for time-variant neuron states. Currently there has been a growing interest in such NNs, called categorically recurrent neural networks (RNNs). One of the most noticeable features of RNNs is their compatibility with spatiotemporal patterns such as those related to speech synthesis, visual perception, and motion control of robots.

More generally, RNNs are categorized as an NN model which is featured by full connections between every neuron including a feedback loop to itself, as well as a neuron function governed by a nonlinear regular differential equation with respect to time. It is for this reason that the majority of NN models proposed so far can be categorized as more or less partial variations of RNNs. Due to such complexities, supervised learning (SL) of RNNs had not been developed until very recently, when different research groups introduced several prototypes of learning algorithms independently.

Most of these prototypes of SL algorithms (SLAs) were described as numerical procedures of solving simultaneous nonlinear differential equations for the unknown multiple connection weights. As the number of neurons increases, i.e. the number of connections increases, more and more numerical calculations are required.

It is mainly for this reason that no practical studies have ever been reported on large-scale numerical simulations for SL of RNNs. At the present state of art, no one can answer the fundamental question: "What can and what can not be realized by RNNs?" The investigations into SL for RNNs have only just begun in the history of the research on NNs which is more than a quarter of a century old.

2. HOW TO MAKE THEM LEARN?

2.1. How Neurons Behave?

Let us consider an RNN which consists of N neurons in total. At time t, an arbitrary neuron (number i) receives the external input signal $X_i(t)$. It is connected with the output $y_j(t)$ of every neuron (j) including itself through a weighted connection in terms of w_{ij}. Then it transmits the output signal $y_i(t)$ defined by the following equations:

$$\gamma_i \frac{dx_i(t)}{dt} = -x_i(t) + \sum_j^N w_{ij} y_j(t) + X_i(t), \quad (1)$$

$$y_i(t) = f(x_i(t)) \equiv 1/(1+e^{x_i(t)}), \quad (2)$$

where γ_i denotes the time constant of time-varying state $x_i(t)$ of neuron i. Let every connection weight w_{ij} be denoted by vector w of N^2-dimensions.

In principle, any one of the neurons can be either an input or an output neuron, or both. We consider the case where $n_{in}(\leq N)$ neurons are given the external input signals, and the output signals of $n_{out}(\leq N)$ neurons are externally extracted. We use vector representations to designate categorically the whole input and output signals as $x(t)$ and $y(t)$ of N-dimensions, as well as the external input and the desired output as $X(t)$ and $Y(t)$ of n_{in}- and n_{out}-dimensions, respectively.

2.2. What Is Supervised Learning?

Given a definite combination of the external input $X(t)$ and the desired output $Y(t)$, it is referred to hereafter as a "learning pattern" (LP). In principle, SL is a numerical process to minimize the difference between $y(t)$ and $Y(t)$ in terms of the cost function E through adjusting the connection weight w. There are several variations proposed concerning how to define E and how to adjust w.

We can classify them into two basic categories, judging whether w is adjusted off-line or on-line, as follows:

(A) Run an RNN within a certain period of time

(generally for the duration of a given LP), regarding w as time-invariant, then stop it, and adjust w off-line. (See fig.1.)

(B) Regarding w as time-variant, adjust it on-line, while running the RNN.

We can categorize into (A) the SLAs proposed by Pearlmutter[4] and Sato[5] independently, in which they employed the steepest descent method (SDM) to search for the minimum of the cost function. On the other hand, category (B) can be classified further into two sub-categories: i.e. (I) the one based on the steepest descent method, and (II) the other on the variation method. The former was proposed independently by Williams and Zipser[6], Doya and Yoshizawa[2], and Gherrity[7], while the latter by Sato[8].

In our previous papers[9,10], we investigated the above mentioned SLAs in detail, and found that category (A) is more practical than the other two. It is less intricate in the numerical procedures, and costs less computational time until convergence of SL and less memory space for simulations.

It is for this reason that we restrict our arguments to category (A) of SLAs in this paper. Note that the connection weight vector w is not a function of time t, but of the iteration number v of the numerical procedures of category (A).

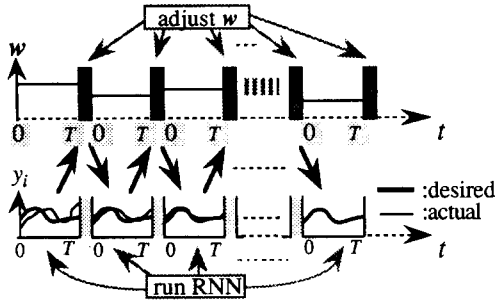

Fig.1 Schematic illustration of how the off-line SLA is conducted.

2.3. How to Formulate Learning?

For simplicity we treat tentatively the case where a single LP is given. Later, we will extend our arguments to more general cases of multiple LPs. Now, consider that the given LP extends along the time axis over t=0 to T.

As is the case in SDM for MNNs, we are concerned with the squared error between the actual output vector $\boldsymbol{y}(t)$ and the desired $\boldsymbol{Y}(t)$. For the time-variant LP of RNNs we have to integrate the error over t=0 to T to provide a cost function $E(\boldsymbol{w}(v))$ such as

$$E(\boldsymbol{w}(v)) = \int_0^T \sum_k^{n_{out}} \frac{1}{2}(y_k(t) - Y_k(t))^2 \, dt. \tag{3}$$

In SDM we move the weight vector $\boldsymbol{w}(v)$ along the gradient by fractional displacement $\Delta\boldsymbol{w}(v)$ per iteration of SLA expressed termwise as

$$\Delta w_{ij}(v) = -\eta \frac{\partial E(\boldsymbol{w}(v))}{\partial w_{ij}} \quad (\eta > 0), \tag{4}$$

where η is an arbitrary constant of small positive quantity. The fractional decrement ΔE of the cost function E against $\Delta\boldsymbol{w}(v)$ becomes

$$\Delta E_v = \sum_{i,j}^{N} \frac{\partial E}{\partial w_{ij}} \Delta w_{ij}(v) = -\eta \sum_{i,j}^{N} \left(\frac{\partial E}{\partial w_{ij}} \right)^2 \le 0. \tag{5}$$

It is not a simple task to derive a general formula for every component $\partial E/\partial w_{ij}$. As was described in [4] and [5], we introduce a Lagrange multiplier L_i to rewrite eq.3 into

$$E(\boldsymbol{w}(v)) = \int_0^T \left[\sum_k^{n_{out}} \frac{1}{2}(y_k(t) - Y_k(t))^2 - \sum_i^N L_i \left\{ \gamma_i \frac{dx_i}{dt} + x_i(t) - \sum_j^N w_{ij} y_j(t) - X_i(t) \right\} \right] dt. \tag{6}$$

Note that the formula in the braces on the right-hand side is always zero.

As a constraint for the Lagrange multiplier L_i let us presume that L_i is a function of t and satisfies the following equation:

$$\gamma_i \frac{L_i(t)}{dt} = L_i(t) - \sum_j^N L_j(t) w_{ij} \frac{df(x_i)}{dx_i} - \delta_{ik}(y_i - Y_i) \frac{df(x_i)}{dx_i}, \tag{7}$$

where δ_{ik} denotes Kronecker's delta. Fractional adjustment δw_{ij} results in fractional variation δx_i for every x_i and ultimately δE, for which using eq.6 we obtain

$$\delta E = \int_0^T \sum_{i,j}^N L_i(t) y_i(t) \delta w_{ij} dt - \left[\sum_i^N \tau_i L_i(t) \delta x_i \right]_0^T. \tag{8}$$

Provided the second term in eq.8 is equated to zero, we can derive $\partial E/\partial w_{ij}$. This is the case when any one of the following four pairs of boundary conditions is satisfied:

$$\delta x_i(0)=0 \quad \text{and} \quad \delta x_i(T)=0, \tag{9}$$

$$\delta x_i(0)=0 \quad \text{and} \quad L_i(T)=0, \tag{10}$$

$$L_i(0)=0 \quad \text{and} \quad \delta x_i(T)=0, \tag{11}$$

$$L_i(0)=0 \quad \text{and} \quad L_i(T)=0. \tag{12}$$

Then we can rewrite eq.8 as

$$\frac{\partial E}{\partial w_{ij}} = \int_0^T L_i(t) y_i(t) dt, \tag{13}$$

The differential equation for L_i in eq.7 is solvable when the neuron state $x_i(t)$ is known through t=0 to T. It is clear that the differential equations for x_i and L_i are of the first order. It is for these two reasons that we can only adopt eq.10 or 11 among the four.

2.4. How to Realize It?

An example of numerical procedures is illustrated in fig.2, where we selected eq.10 as the boundary conditions. First we calculate every neuron state $\boldsymbol{x}(t)$, or every output $\boldsymbol{y}(t)$ in the forward direction of time using eqs.1 and 2: i.e. $\boldsymbol{x}(0) \Rightarrow \boldsymbol{x}(T)$ and $\boldsymbol{y}(0) \Rightarrow \boldsymbol{y}(T)$. Then every Lagrange multiplier $(L_1(t), L_2(t), \cdots, L_N(t)) \equiv \boldsymbol{L}(t)$ in the backward direction of time using eq.7, i.e. $\boldsymbol{L}(T) \Rightarrow \boldsymbol{L}(0)$. Finally, fractional displacement $\Delta\boldsymbol{w}$ using eq.13. These steps of numerical calculations are iterated until we find a minimal value of the cost function $E(\boldsymbol{w}(v))$, in actuality, the one below the prespecified limit of error.

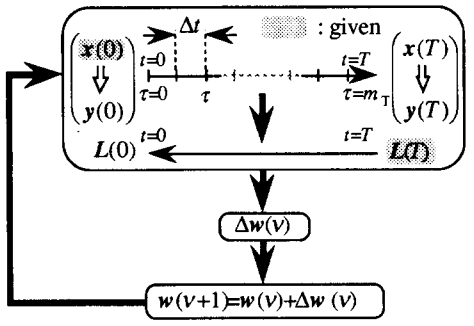

Fig.2. Flow diagram to illustrate the numerical procedures of BPTT under the discrete time regime when eq.10 is used as the boundary conditions.

The numerical procedures described above are called back-propagation through time (BPTT), since the mathematical formulation is similar to that of BP for MNNs. As we have seen, they are much more intricate than those of BP for MNNs so that we must face more seriously the same complications in application as we do in the case of MNNs. They are typically called local minima, slow learning, over-learning, etc.

Emphasis should be placed on the knowledge in functional analysis that the cost function E in eq.3 is the Lyapnov function of variable v to treat a nonlinear system under the discrete time regime. It implies that, if and only if N is unlimited, RNNs can approximate any dynamical systems to an arbitrarily high precision, and also that there exists an

optimum selection $\boldsymbol{w}_{opt}$ which realizes this approximation[11].

It is obvious that the above complications result from nonlinearity which exists between $\boldsymbol{w}$ and $E(\boldsymbol{w})$. For further arguments we introduce Euclidean space of (N^2+1)-dimensions where $E(\boldsymbol{w})$ and every component of $\boldsymbol{w}$ are assigned to each coordinate axis.

Imagine that we plot the cost function $E(\boldsymbol{w})$, and it turns up as a hyper surface in this space. So we can state that SL based on BPTT is equivalent to descending the slope of this hyper surface by a step of fractional displacement $\delta\boldsymbol{w}$ toward a minimal value (δE=0).

As was proposed in our previous papers as to MNNs[12,13], let us refer to Euclidean space E-$\boldsymbol{w}$ as ***learning space*** and hyper surface E=$E(\boldsymbol{w})$ as ***learning surface***, respectively. To avoid misleading arguments through analogy, we must keep in mind that solid geometry in 3-D is not always valid in the multi-dimensional space.

3. INNER WORLD OF RNN

3.1. Another Way to Formulate Learning

We discuss BPTT a little more precisely. To solve numerically the differential equation in eq.1, we must discretize the time interval t=0 to T into minute subintervals of m_T in total. Without impairing generality we can assume that the subinterval Δt is taken to be unit time length. Thus, we can rewrite eqs.1 and 2 into

$$ {}^{\tau+1}x_i = h_i\left(\boldsymbol{w}, {}^{\tau}\boldsymbol{x}, {}^{\tau}\boldsymbol{X}\right), \tag{14} $$

$$ {}^{\tau+1}y_i = f\left({}^{\tau+1}x_i\right), \tag{15} $$

where prior superscript τ denotes the discrete time from 0 to m_T-1, and h_i the first order difference equation derived from eq.1, such as

$$ h_i = \left(1-{}^{1}\!/_{\gamma_i}\right){}^{\tau}x_i + \sum_j w_{ij}\,{}^{\tau}y_j/\gamma_i + {}^{\tau}X_i/\gamma_i. \tag{16} $$

When we substitute eq.14 repeatedly into ${}^{\tau}\boldsymbol{x}$, decreasing τ one by one down to 0, we can derive

$$ \begin{aligned} {}^{\tau+1}y_i &= f\left(h_i\left(\boldsymbol{w}, {}^{\tau}\boldsymbol{x}, {}^{\tau}\boldsymbol{X}\right)\right) \\ &= f\left(h_i\left(\boldsymbol{w}, \boldsymbol{h}\left(\boldsymbol{w}, {}^{\tau-1}\boldsymbol{x}, {}^{\tau-1}\boldsymbol{X}\right), {}^{\tau}\boldsymbol{X}\right)\right) \\ &\vdots \\ &= f\left(h_i\left(\boldsymbol{w}, \boldsymbol{h}\left(\boldsymbol{w}, \cdots, \boldsymbol{h}\left(\boldsymbol{w}, \boldsymbol{h}\left(\boldsymbol{w}, {}^{0}\boldsymbol{x}, {}^{0}\boldsymbol{X}\right), {}^{1}\boldsymbol{X}\right), \cdots, {}^{\tau-1}\boldsymbol{X}\right), {}^{\tau}\boldsymbol{X}\right)\right) \\ &\equiv f\left({}^{\tau}H_i\left(\boldsymbol{w}, {}^{0}\boldsymbol{x}, {}^{\tau}\boldsymbol{\mathcal{X}}\right)\right) \equiv \Phi_k\left(\boldsymbol{w}, {}^{0}\boldsymbol{x}, {}^{\tau}\boldsymbol{\mathcal{X}}\right), \end{aligned} \tag{17} $$

where we used vector representations such as:

$$ \boldsymbol{h} \equiv \left(h_1, h_2, \cdots, h_N\right), \tag{18} $$

$$ {}^{\tau}\boldsymbol{\mathcal{X}} \equiv \left({}^{0}\boldsymbol{X}, {}^{2}\boldsymbol{X}, \cdots, {}^{\tau}\boldsymbol{X}\right). \tag{19} $$

Equation 17, derived first by the authors [12,13], is very significant, since it implies that the output of neuron i is a function of initial neuron state ${}^{0}\boldsymbol{x}$, weight $\boldsymbol{w}$ and extended sequence of the discretized external input ${}^{\tau}\boldsymbol{\mathcal{X}}$ from t=0 to τ.

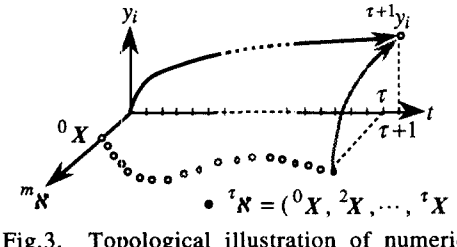

Fig.3. Topological illustration of numerical procedures of BPTT.

Thus, no matter how complex Γ may appear, eq.17 represents a nonlinear mapping from the extended external input ${}^{\tau}\boldsymbol{\mathcal{X}}$ to the output ${}^{\tau}\boldsymbol{y}$. The numerical algorithms are schematically illustrated in fig.3, in which the time-variant output $y_i(t)$ of neuron i is traced as a curved trajectory as a function of the extended external input ${}^{\tau}\boldsymbol{\mathcal{X}}$.

More generally, we can state that SL for RNNs is to trace trajectory $\boldsymbol{y}(t)$ of every external output, and to shape it into the desired $\boldsymbol{Y}(t)$ by adjusting weight $\boldsymbol{w}$.

3.2 Memory Surface

Let us consider the ideal case of perfect SL, i.e. $y_k(t)$=$Y_k(t)$ for every k. To remove the dependence on the initial states of neurons, we maintain them identical, say ${}^{0}\boldsymbol{x}$=$\boldsymbol{0}$ (zero vector). Then $\boldsymbol{w}$ must satisfy the nonlinear equation:

$$ {}^{\tau+1}Y_k = \Phi_k\left(\boldsymbol{w}, {}^{0}\boldsymbol{x}, {}^{\tau}\boldsymbol{\mathcal{X}}\right). \tag{20} $$

We can admit that there exists at least a solution of eq.20, which is denoted by ${}^{\tau}\boldsymbol{w}_k$. Noticing that eq.20 represents a constraint to decrease degrees of freedom by one in the N^2-dimensional topological space of $\boldsymbol{w}$, we can see that a set which contains every solution of eq.20, $\{{}^{\tau}\boldsymbol{w}_k\}$, constitutes a manifold of (N^2-1)-dimensions, i.e. a hyper surface* in the N^2-dimensional Euclidean space. Every point on the hyper surface enables the transformation from ${}^{\tau}\boldsymbol{\mathcal{X}}_k$ to ${}^{\tau+1}Y_k$, both of which are given to RNNs as the external signals.

In other words, when we consider the topological space Y-${}^{\tau}\boldsymbol{\mathcal{X}}$, it constitutes the external world of networks, while the topological space $\boldsymbol{w}$ constitutes the ***inner world***. A single hyper surface in the inner world corresponds to one point in the external world. Thus we may state that RNNs memorize an external input-output pair $({}^{\tau}\boldsymbol{\mathcal{X}}, {}^{\tau+1}Y_k)$ as a hyper surface $\{{}^{\tau}\boldsymbol{w}_k\}$ in the inner world. Therefore, we refer to this hyper surface in space $\boldsymbol{w}$ as ***memory surface*** [12,13].

For the LP from t=0 to T, BPTT is equivalent to tracing a trajectory of $({}^{\tau}\boldsymbol{\mathcal{X}}, {}^{\tau+1}Y_k)$ in the external world from τ=0 to m_T-1. This results in simultaneous nonlinear equations of class m_T, and it follows that multiple memory surfaces in total number m_T exist in space $\boldsymbol{w}$.

The above statement applies to each external output of neuron k=0 to n_{out}, so that we have $m_T \times n_{out}$ memory surfaces. Moreover, when we extend our arguments to multiple LPs of l in total, the number of memory surfaces amounts to $m_T \times n_{out} \times l$. In the ideal case where every memory surface intersects a common domain† (or domains), every point within the domain(s) provides a set of exact solutions $\{\boldsymbol{w}_{exact}\}$ of simultaneous nonlinear equations of class $m_T \times n_{out} \times l$.

* We are investigating whether or not a single set of solutions $\{{}^{\tau}\boldsymbol{w}_k\}$ of eq.19 represents separate multiple hyper surfaces just like double leaves of hyperboloids in three dimensions (3-D). Nevertheless, let us regard $\{{}^{\tau}\boldsymbol{w}_k\}$ tentatively, as if it is a single hyper surface.

† An intersection of multiple hyper surfaces is not necessarily a point but more generally a domain which is also a hyper surface of lower dimensions.

4. WHAT ABOUT LEARNING SPACE?

4.1. Learning Surface Looks Like What?

We have pointed out that existence of $\{\boldsymbol{w}_{opt}\}$ is admitted when an unlimited number of neurons are available. It is hardly possible to solve analytically a large class of simultaneous nonlinear equations as complex as shown in eq.17. It is for this reason that we employ SDM to search for a minimal value of the cost function $E(\boldsymbol{w}_{min})$.

Moreover, in practical applications where the total number of neurons is limited, it is quite uncertain whether or not there exists an exact solution. To clear this question, the notion of ***memory surface*** plays an essential role.

As we have demonstrated for SL of MNNs[12,13], let us decompose the cost function $E(\boldsymbol{w})$ in terms of LP λ, discrete time τ and output neuron k, to acquire a clear-cut overview of the learning space:

$$ E(\boldsymbol{w}) = \sum_{\lambda=1}^{l}\sum_{\tau=1}^{m_T}\sum_{k=1}^{n_{out}} {}^{\tau}_{\lambda}E_k(\boldsymbol{w}), \tag{21} $$

$$ {}^{\tau}_{\lambda}E_k(\boldsymbol{w}) = \frac{1}{2}\left({}^{\tau}_{\lambda}y_k(\boldsymbol{w}) - {}^{\tau}_{\lambda}Y_k\right)^2. \tag{22} $$

Note that a set of solutions $\{{}^{\tau-1}_{\lambda}\boldsymbol{w}_k\}$ for ${}^{\tau}_{\lambda}E_k(\boldsymbol{w})$=0, i.e. ${}^{\tau}_{\lambda}y_k\left({}^{\tau-1}_{\lambda}\boldsymbol{w}_k\right)$=${}^{\tau}_{\lambda}Y_k$, is equivalent to the memory surface. If $\boldsymbol{w}$=${}^{\tau-1}_{\lambda}\boldsymbol{w}_k$, gradient $\nabla {}^{\tau}_{\lambda}E_k(\boldsymbol{w})$ of the learning surface is always equal to 0. Thus, the first two terms in the series expansion of ${}^{\tau}_{\lambda}E_k(\boldsymbol{w})$ with respect to $(\boldsymbol{w}-{}^{\tau-1}_{\lambda}\boldsymbol{w}_k)$ are equated to 0, and we

obtain an approximation for ${}^{\tau}_{\lambda}E_k(w)$ around $w \approx {}^{\tau-1}_{\lambda}w_k$:

$$ {}^{\tau}_{\lambda}E_k(w) \cong \frac{1}{2}\left(w - {}^{\tau-1}_{\lambda}w_k\right)\nabla^2 {}^{\tau}_{\lambda}E_k\left({}^{\tau-1}_{\lambda}w_k\right)\left(w - {}^{\tau-1}_{\lambda}w_k\right)^{\mathrm{T}}, \quad (23) $$

where ∇^2 and superscript $^{\mathrm{T}}$ denote Hessian and transposition operator, respectively.

Equation 23 implies that, if we plot the termwise cost function in the learning space, it becomes a hyper surface of the second-order in the vicinity of the memory surface $\{ {}^{\tau-1}_{\lambda}w_k \}$. In addition, when we plot the locus of w which satisfies eq.23 for a positive definite value of ${}^{\tau}_{\lambda}E_k(w)$, or an equi-contour, it turns up as a hyper ellipse with each principal axis lying in the same direction as one of the eigenvectors of the Hessian matrix $\nabla^2 {}^{\tau}_{\lambda}E_k$. Thereby the length of each principal axis is inverse-proportional to the square root of the corresponding eigenvalue of the matrix.

It is fairly simple to show that every diagonal element of the Hessian matrix is positive, and also that its determinant is always 0. It follows immediately that one of the eigenvalues must be equal to 0. In other words, the corresponding principal axis of the hyper ellipse becomes infinite in length. This is a conspicuous feature of the termwise learning surface $E = {}^{\tau}_{\lambda}E_k(w)$ in the vicinity of the memory surface $\{ {}^{\tau-1}_{\lambda}w_k \}$.

In fact, we can visualize the termwise cost function in 3-D cross sections. When any two of the coordinate axes of 3-D coincide with the longest ellipse-axis and the E axis, it turns up as a very long ***valley*** along the longest axis of the hyper ellipse. In short, the bottom of the valley is another expression to paraphrase the memory surface. Due to limited space of the paper we only mention our observations that in most parts the valley is curved gently, and its bottom is barely inclined.

On the other hand, we can prove easily that every partial derivative $\partial {}^{\tau}_{\lambda}E_k(w)/\partial w_{ij}$ becomes zero in infinite regions of space w. Considering this together with fairly rapid saturation of the sigmoid function, we can understand that the termwise learning surface becomes flat in remote regions from the memory surface. This is the second feature of the learning surface which we may describe it as a ***hill*** by the analogy of geology.

4.2. Valleys and Hills Feature Learning Surface?

Having conducted various kinds of simulations, we discovered that the above two topological features are clearly observable, even when the termwise cost functions are multiply superposed.

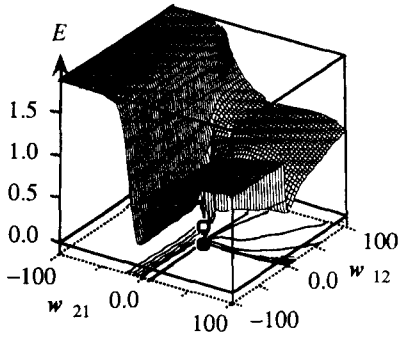

Fig.4. A typical example of simulation results showing ***valleys and hills*** of the learning surface with 2-D cross sections of all the 10 memory surfaces shown on the base plane. The open circle denotes $w_{\min}$, while the shaded w_{exact}.

Figure 4 illustrates a typical example of simple simulations for N=2, n_{in}=n_{out}=1, and m_T=5 of SL. All the memory surfaces of 10 in total are shown on the base plane as 2-D cross sections. As is clearly seen in this figure, the valleys of the learning surface are gently curved in most parts. This is a distinctive feature of the learning surface which we can utilize to accelerate SL. For details readers should refer to our previous papers[12,13].

The minimal value of $E(w_{\min})$ reached at ν=60000 is very close to zero so that the vector $w_{\min}$ is a good approximation of the intersection of all the memory surfaces w_{exact} indicated by a shaded circle. In this case we may conclude that SL has been completed almost perfectly.

5. HOW TO CLEAR LOCAL MINIMA?

In more complicated simulations of SL conducted using larger scale RNNs, the number of the termwise cost functions is fairly large. Obviously their superposition in the learning space complicates the overall structure of the resultant learning surface. Nevertheless, attempting to view the learning space perspectively through memory surfaces, we can acquire clear-cut insights into the complications commonly related to SL.

In some cases there exists apparently no common intersection of all the memory surfaces, but they come very close to each other in a certain region. The resultant learning surface may turn up there as a deep valley with a broad bottom like a geological basin, as is shown schematically in fig.5.

In case (a) each memory surface intersects every other and comes much closer to each other than in cases (b) and (c). It seems very reasonable to expect that the minimal in case (a) provides a good approximation of the minimum. As shown by (c), it happens very often that a particular memory surface never intersects the others and lies almost in parallel to one of them. The memory surfaces of this pair approach each other very gradually toward infinity. Once we are trapped in this valley, the bottom of which is usually inclined only barely, it is hardly possible to converge SLA. As is usually the case in applications where N is fixed, it depends mainly on initial selections of w(0) whether we reach ultimately (a), (b) or (c).

To solve this, we can reconstruct topological configuration of memory surfaces by attaching extra degrees of freedom to the learning space w. In other words, introducing an additional number of neurons, we can increase the dimensions of space w. We are now continuing our investigations into this possibility.

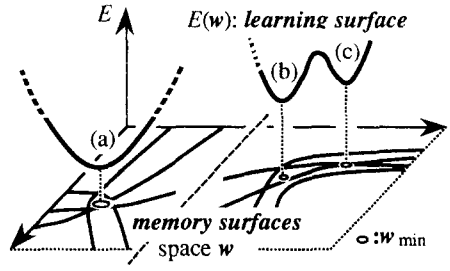

Fig.5. Schematic illustration of local minima.

REFERENCES

[1] Rumelhart DE, McClelland JL and the PDP Research Group: Parallel Distributed Processing. **1**, The MIT Press (1987)

[2] Doya K and Yoshizawa S: Adaptive neural oscillator using continuous-time back-propagation. *Neural Networks* **2** pp.375-385 (1989)

[3] Sato M, M Joe and Hirahara T: APOLONN brings us to the real world-learning nonlinear dynamics and fluctuations in nature. Proc. IJCNN'90-San Diego **1** pp.581-587 (1990)

[4] Pearlmutter BA: Learning state space trajectories in recurrent neural networks. *Neural Computation* **1** pp.263-269 (1989)

[5] Sato M: A learning algorithm to teach spatio-temporal patterns to recurrent neural networks. *Biol. Cybern.* **62** pp.259-263 (1990)

[6] Williams RJ and Zipser D: A learning algorithm for continually running fully recurrent neural networks. *Neural Computation* **1** pp.270-280 (1989)

[7] Gherrity M: A learning algorithm for analog, fully recurrent neural networks. Proc. IJCNN'89-Washington D.C. **1** pp.643-644 (1989)

[8] Sato M: A real time learning algorithm for recurrent analog neural networks. Biol.. Cybern. **62** pp.2229-2232 (1990)

[9] Watanabe T, Gouhara K and Uchikawa Y: Study of learning algorithms and shape of the learning surface for recurrent neural networks. *Systems and Computers in Japan* (1993) (in print)

[10] Gouhara K, Watanabe T and Uchikawa Y: Learning process of recurrent neural networks. Proc. IJCNN'91-Singapore **1** pp.746-751 (1991)

[11] Seidl D R and Lorentz R D: A structure by which a recurrent neural network can approximate a nonlinear dynamic system. Proc. IJCNN'91-Seattle **2** pp.709-714 (1991)

[12] Gouhara K, Uchikawa Y: Analysis for learning surface of hierarchical neural networks. *IEICE Techn. Rep.* **NC90**-43 (1990) (in Japanese)

[13] Gouhara K, Kanai N, Uchikawa Y: Valley shape of learning surface and learning rule. *IEICE Techn. Rep.* **NC90**-44 (1990) (in Japanese)