Artificial Neural Networks

- Multilayer Perceptrons (MLPs)

- Radial Basis Function Networks (RBFNs)

- Recurrent Neural Networks (RNNs)

Multilayer Perceptrons (MLPs)

- Appropriate Problems for Neural Network Learning

. Instances are represented by many attribute-value pairs.
. The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.
. The training examples may contain errors.
. Long training times are acceptable.
. Fast evaluation of learned target function may be required.
. The ability of humans to understand the learned target function is not important.

- Classification of Artificial Neural Networks (ANNs)

Recall \ Learning	*FEEDBACK RECALL*	*FEEDFORWARD RECALL*
UNSUPERVISED LEARNING	• Additive Grossberg (AG) • Shunting Grossberg (SG) • Binary Adaptive Resonance Theory (ART1) • Analog Adaptive Resonance Theory (ART2) • Discrete Hopfield (DH) • Continuous Hopfield (CH) • Discrete Bidirectional Associative Memory (BAM) • Temporal Associative Memory (TAM) • Adaptive Bidirectional Associative Memory (ABAM)	• Learning Matrix (LM) • Driver-Reinforcement Learning (DR) • Linear Associative Memory (LAM) • Optimal Linear Associative Memory (OLAM) • Sparse Distributed Associative Memory (SDM) • Fuzzy Associative Memory (FAM) • Learning Vector Quantization (LVQ) • Counterpropagation (CPN)
SUPERVISED LEARNING	• Brain-State-in-a-Box (BSB) • Fuzzy Cognitive Map (FCM)	• Perceptron • Adaline & Madaline • Backpropagation (BP) • Boltzmann Machine (BM) • Cauchy Machine (CM) • Adaptive Heuristic Critic (AHC) • Associative Reward Penalty (ARP) • Avalanche Matched Filter (AMF)

TABLE 5-1. Taxonomy of ANS paradigms showing the models that belong to each class. See text for a description of the taxonomy.

- Structure of MLPs

. Multiple layers of Perceptrons are connected.

. Three types of layers: input, hidden, and output layers.
Usually, the input layer is not included when we count
the number of layers.

. The direction of connection is from the input to output layers
(feedforward connection).

. Except the input layer, each unit in the layer is described by

$$n_j = \sum_{i=0}^{n} w_{ij} x_i \text{ and}$$

$$o_j = \sigma(n_j) = \frac{1}{1+e^{-n_j}}$$

where

w_{ij} = the connection between the ith input to the jth unit,

x_i = the ith input, and

o_j = the output of the jth unit.

Note that

$$\frac{\partial \sigma(n_j)}{\partial n_j} = \sigma_j(n_j)(1-\sigma(n_j)).$$

- Expressive Capabilities of MLPs

. Boolean functions:

Every boolean function can be represented by network with single hidden layer but might require exponential (in number of inputs) hidden units.

. Continuous functions:

Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer (Cybenko 1989; Hornik et al., 1989)

- Backpropagation algorithm

. Error function for multiple outputs:

$$E(\underline{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in Outputs} (t_{kd} - o_{kd})^2$$

where D represents the sample index set and $Outputs$ represents the output index set.

. Backpropagation algorithm:

For each training sample d, weights are updated (on-line mode), that is,

$$E_d(\underline{w}) = \frac{1}{2} \sum_{k \in Ouptputs} (t_k - o_k)^2.$$

. two-layer network

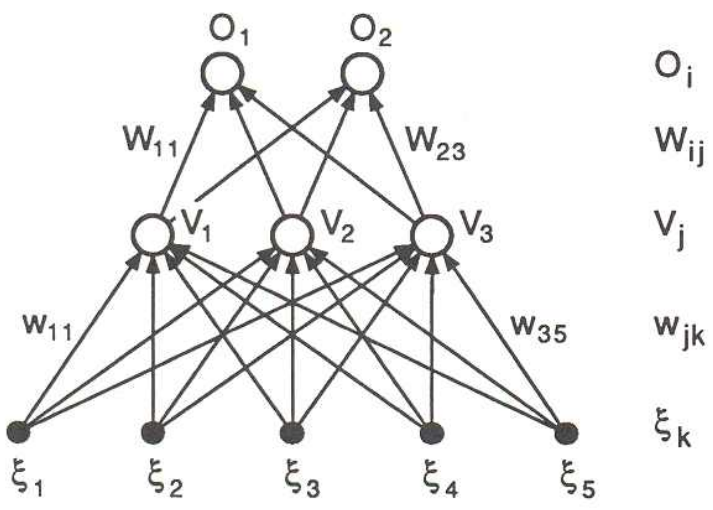

FIGURE 6.1 A two layer feed-forward network, showing the notation for units and weights.

. weight update between hidden and output layers

$$v_{kj}^{\neq w} = v_{kj}^{old} + \eta \Delta v_{kj} \quad \text{and}$$

$$\Delta v_{kj} = -\frac{\partial E_d}{\partial v_{kj}} = (t_k - o_k) o_k (1 - o_k) h_j = \delta_k h_j$$

where $\delta_k = (t_k - o_k) o_k (1 - o_k)$.

. weight update between input and hidden layers

$$w_{ji}^{\neq w} = w_{ji}^{old} + \eta \Delta w_{ji} \quad \text{and}$$

$$\Delta w_{ji} = -\frac{\partial E_d}{\partial w_{ji}} = h_j (1 - h_j) \sum_{k \in Outputs} v_{jk} \delta_k x_i = \delta_j' x_i$$

where $\delta_j' = h_j (1 - h_j) \sum_{k \in Outputs} v_{jk} \delta_k$.

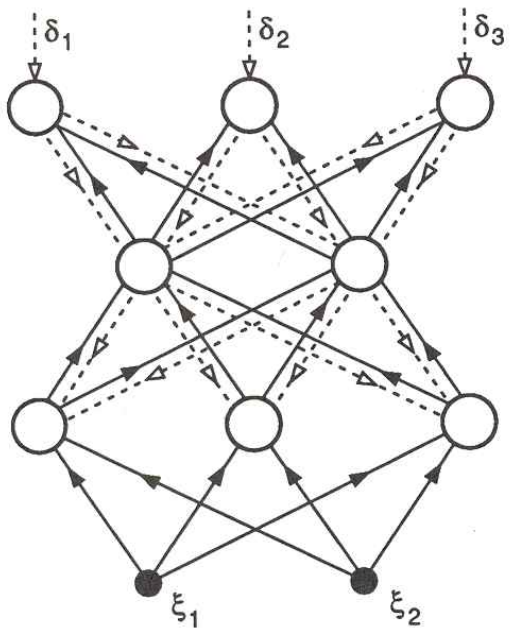

FIGURE 6.2 Back-propagation in a three-layer network. The solid lines show the forward propagation of signals and the dashed lines show the backward propagation of errors (δ's).

- Convergence of Backpropagation

. stochastic gradient descent
. convergence to the global minimum (or even to some local minimum) is not guaranteed. -> usually, small learning rate is applied.
. add momentum term for faster convergence.
. initialize weights near zero -> initial networks near-linear.
. increasingly non-linear functions possible as training progresses

- Backpropgation with momentum term

. A momentum term can be included in the gradient descent algorithm for the fast convergence.

. The weight update term is changed as follows:

$$\Delta w_{ij}(n) = \eta \delta_j x_i + \alpha \Delta w_{ij}(n-1)$$

where α represents the momentum constant.

. Let the time index $t = 0, 1, \cdots, n$. Then,

$$\Delta w_{ij}(n) = \eta \sum_{t=0}^{n} \alpha^{n-t} \delta_j(t) x_i(t) = -\eta \sum_{t=0}^{n} \alpha^{n-t} \frac{\partial E_d}{\partial w_{ij}}.$$

1) for convergence, $|\alpha| < 1$.

2) it tends to accelerate descent in steady downhill directions since $\Delta w_{ij}(n) \approx -\frac{\eta}{1-\alpha} \frac{\partial E_d}{\partial w_{ij}}$.

3) when $\frac{\partial E_d}{\partial w_{ij}}$ has opposite signs on consecutive iterations, $\Delta w_{ij}(n)$ shrinks (stabilizing effect).

. Effect of Momentum Term

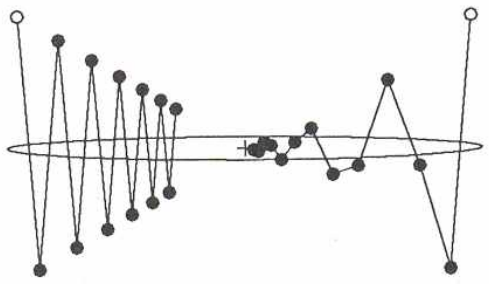

FIGURE 6.3 Gradient descent on the simple quadratic surface of Fig. 5.10. Both trajectories are for 12 steps with $\eta = 0.0476$, the best value in the absence of momentum. On the left there is no momentum ($\alpha = 0$), while $\alpha = 0.5$ on the right.

- Backpropagation with conjugate gradient

. In the CGA, information on Q, that is, the input correlation matrix is required.

. For the on-line mode, an alternative choice of α_k and β_k is needed without the knowledge of Q.

. Determination of α_k:

1) Assuming $E[\alpha_k]$ is unimodal
(single minimum in the neighborhood of current $\underline{w}_k$),

find three points η_1, η_2, and η_3 such that
$E[\eta_1] \geqq E[\eta_3] \geqq E[\eta_2]$ for $\eta_1 < \eta_2 < \eta_3$.

2) α_k is determined between η_2 and η_3.

. Determination of β_k:

Apply Fletcher-Reeves formula

$$\beta_k = \frac{\underline{g}_{k+1}^T \underline{g}_{k+1}}{\underline{g}_k^T \underline{g}_k}.$$

Conjugate Gradient Algorithm for MLP

Step 1. Set $k=0$, initialize $\underline{w}_0$, and compute $\nabla_k = \frac{\partial E}{\partial \underline{w}}|_{\underline{w}=\underline{w}_k}$.

Step 2. Set $\underline{g}_k = \nabla_k$ and $\underline{d}_k = -\underline{g}_k$.

Step 3. Find α_k that minimizes $E[\underline{w}_k + \alpha_k \underline{d}_k]$.

Step 4. Update the weight parameters: $\underline{w}_{k+1} = \underline{w}_k + \alpha_k \underline{d}_k$.

Step 5. Set $\underline{g}_{k+1} = \nabla_{k+1}$ and $\beta_k = \frac{\underline{g}_{k+1}^T \underline{g}_{k+1}}{\underline{g}_k^T \underline{g}_k}$.

Step 6. Set $\underline{d}_{k+1} = -\underline{g}_{k+1} + \beta_k \underline{d}_k$.

Step 7. If satisfied stop. Otherwise, $k \leftarrow k+1$ and go to Step 3.

. Comparison with the momentum term:

1) weight update rule with momentum term

$$\underline{w}_{k+1} = \underline{w}_k + \Delta \underline{w}_k = \underline{w}_k - \eta \nabla E(\underline{w}_k) + \alpha \Delta \underline{w}_{k-1}$$

2) weight update rule with conjugate gradient:

$$\underline{w}_{k+1} = \underline{w}_k + \Delta \underline{w}_k = \underline{w}_k - \alpha_k \nabla E(\underline{w}_k) + \alpha_k \beta_k \Delta \underline{w}_{k-1}$$

* similar effect with the momentum term
* same computational complexity
* in practice, better solution of parameters

- overfitting in MLPs

. Split samples into training and validation sets

. Let M and N represent the number of parameters and the number of samples respectively. Then, overfitting may occur when $N < 30M$. (Amari, 1996)

. Let r be split ratio between training and validation sets. Then, the optimal split ratio for learning is

$$r_{opt} = 1 - \frac{\sqrt{2M-1}-1}{2(M-1)} \approx 1 - \frac{1}{\sqrt{2M}} \quad \text{(for large } M\text{)}.$$

For example, if $M = 100$, then $r_{opt} = 0.07$, that is, 93% of the samples are alloted to training samples. If $N > 30M$, exhaustive learning is satisfactory.

. overfitting effect on function approximation

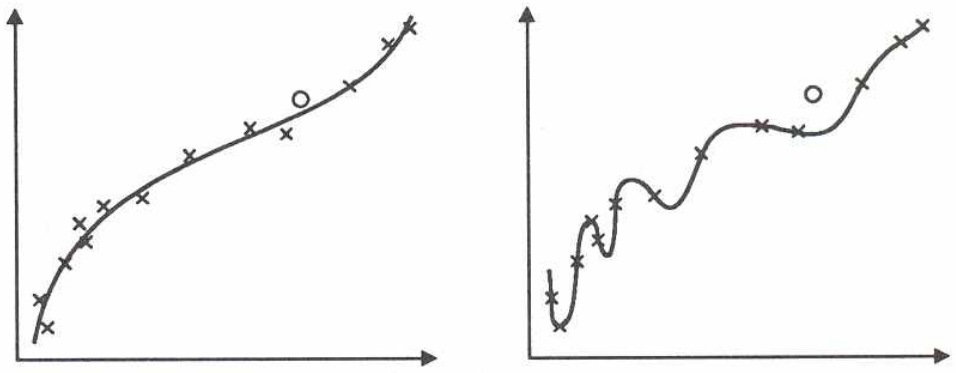

FIGURE 6.13 (a) A good fit to noisy data. (b) Overfitting of the same data: the fit is perfect on the "training set" (x's), but is likely to be poor on a "test set" represented by the circle.

- Complexity regularization

. Add the complexity term to the error measure and minimize the total risk.

. The total risk is described by

$$R(\underline{w}) = E(\underline{w}) + \lambda E_c(\underline{w})$$

where $E(\underline{w})$ represents the error measure,

$E_c(\underline{w})$ represents the complexity term, and

λ represents the regularization parameter.

. weight decay method (Hinton, 1989)

The complexity term is described by

$$E_c(\underline{w}) = \| \underline{w} \|^2.$$

This term is related to the complexity of the hypothesis space.

. weight elimination method (Weigend, 1991)

The complexity term can be bounded as follows:

$$E_c(\underline{w}) = \sum_{i \in C} \frac{(w_i/w_0)^2}{1+(w_i/w_0)^2}$$

where C represents all the synaptic connections.

If $|w_i| \ll w_0$, the ith synaptic weight tends to be eliminated.

If $|w_i| \gg w_0$, the ith synaptic weight has major influence to the network.

- Applications of MLPs

Backpropagation Applications	
Application	**Reference**
Image Processing	Troxel, Rogers & Kabrisky, 1988 Dayhoff & Dayhoff, 1988 Moya, Fogler & Hostetler, 1988 Fogler, Williams & Hostetler, 1988 Lehar, 1988a Roberts, 1988 Weiland, Leighton & Jacyna, 1988 Cottrell & Willen, 1988 Hurlbert, 1988 Castelaz, 1988 Glover, 1988a & 1988b Modorikawa, 1988 Wright, 1988 Scaletter & Zee, 1988 Cottrell, Munrop & Zipser, 1987 Dodd, 1987 Yang & Guest, 1987 Kuczewski, 1987
Speech Processing	Ricotti, Ragazzini & Martinelli, 1988 Robinson & Fallside, 1988 Tishby, 1988 Anderson, Merril & Port, 1988 Bourlard & Wellekens, 1987 & 1988 Kammerer & Kuppu, 1988 Landauer, Kamm & Singhal, 1987 Treurniet, et al., 1988 Burr, 1988a & 1988b Tenorio, Tom & Schwartz, 1988 Lippmann, 1987 & 1988 Rosenberg & Sejnowski, 1986 Sejnowski & Rosenberg, 1987 Elman & Zipser, 1987 Luse, et al., 1988 Watson, 1987 Waibel, et al., 1987
Temporal Processing	Shimohara, Uchiyama & Tokinuga, 1988 Graupe & Uth, 1988 Elman, 1988 Lewis, 1988 Robinson & Fallside, 1988 Tam, Perkel & Tucker, 1988
Prediction/Optimization	Werbos, 1974 & 1988 Castelaz, 1988 Moody & Denker, 1988 Madey & Denton, 1988 Smith, 1988 Dutta & Shekha, 1988 Lapedes & Farber, 1988a & 1988b Tishby, 1988 Karsai, et al., 1988 Quian & Sejnowski, 1988 Baum, 1986a & 1986b

TABLE 5-21.

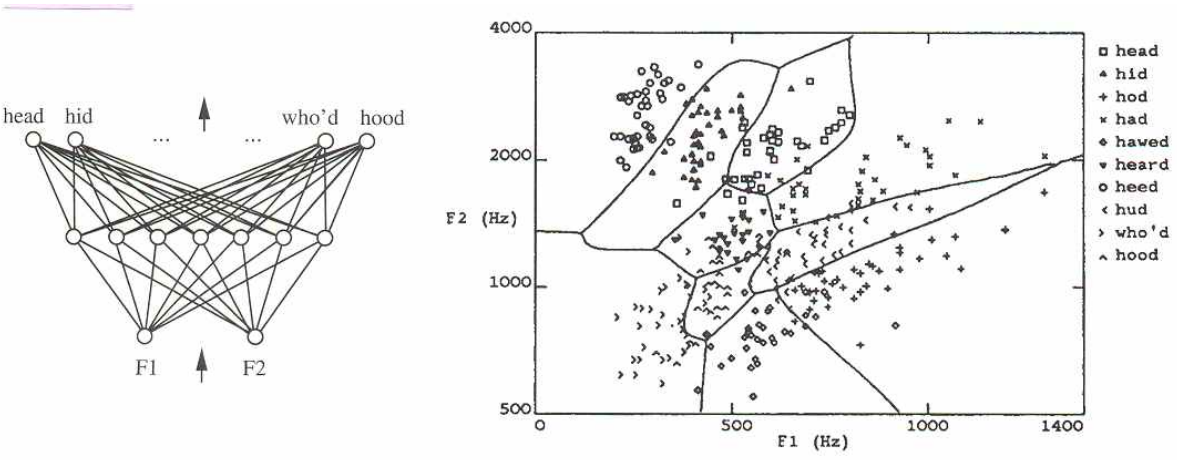

FIGURE 4.5
Decision regions of a multilayer feedforward network. The network shown here was trained to recognize 1 of 10 vowel sounds occurring in the context "h_d" (e.g., "had," "hid"). The network input consists of two parameters, F1 and F2, obtained from a spectral analysis of the sound. The 10 network outputs correspond to the 10 possible vowel sounds. The network prediction is the output whose value is highest. The plot on the right illustrates the highly nonlinear decision surface represented by the learned network. Points shown on the plot are test examples distinct from the examples used to train the network. (Reprinted by permission from Haung and Lippmann (1988).)

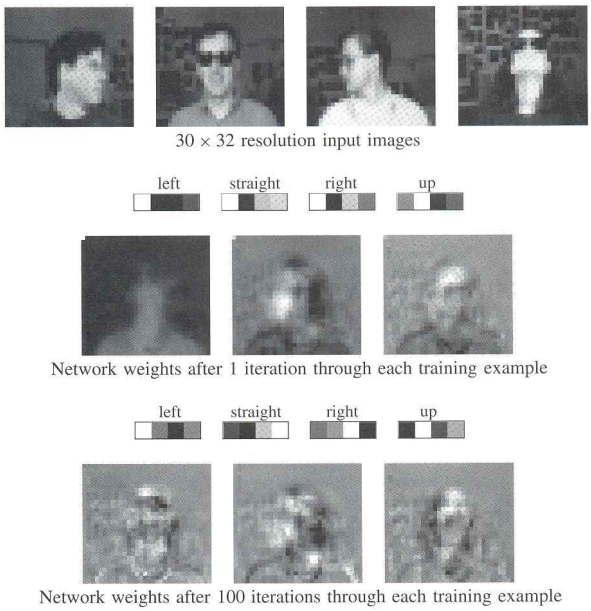

FIGURE 4.10
Learning an artificial neural network to recognize face pose. Here a $960 \times 3 \times 4$ network is trained on grey-level images of faces (see top), to predict whether a person is looking to their left, right, ahead, or up. After training on 260 such images, the network achieves an accuracy of 90% over a separate test set. The learned network weights are shown after one weight-tuning iteration through the training examples and after 100 iterations. Each output unit (left, straight, right, up) has four weights, shown by dark (negative) and light (positive) blocks. The leftmost block corresponds to the weight w_0, which determines the unit threshold, and the three blocks to the right correspond to weights on inputs from the three hidden units. The weights from the image pixels into each hidden unit are also shown, with each weight plotted in the position of the corresponding image pixel.

Radial Basis Function Networks (RBFNs)

- Structure of RBFNs

. Three layers: input, hidden, and output layers

. The hidden layer is composed of kernel functions.

. The output of network is described by

$$\phi(\underline{x}) = \sum_{i=1}^{m} w_i \psi(\underline{x}, \underline{p}_i)$$

where $\underline{p}_i$ represents the parameter vector for the ith kernel function.

- Expressive Capabilities of RBFNs

. Any bounded continuous function $f(\underline{x})$ can be approximated by $\phi(\underline{x})$ provided that

$$\int_{-\infty}^{+\infty}(\frac{1}{a})^n\psi(\underline{x},\underline{p}_a)d\underline{x}= C(>0) \quad \text{and}$$

$$\lim_{a\to\infty}(\frac{1}{a})^n\psi(\underline{x},\underline{p}_a) = C\delta(\underline{x})$$

where a represents the kernel width and $\underline{p}_a$ represents the shape parameter. (Lee and Kil, 1991)

That is, for any $\epsilon > 0$, there exists $\phi(\underline{x})$ such that $|f(\underline{x})-\phi(\underline{x})|<\epsilon$.

. Examples of kernel functions

$$\psi(x,a) = e^{-x^2/a^2}$$

$$\psi(x,a) = Sinc(\frac{x}{a}) = \frac{a}{\pi x}\sin(\frac{\pi x}{a})$$

$$\psi(x,a) = Rect(\frac{x}{a}) = \begin{cases}1 & \text{if } |x|<a/2\\ 0 & otherwise\end{cases}$$

- Learning algorithms

A. Parzen window approach (of estimating probability density functions)

. The volume of d-dimensional hypercube:

$$V_n = h_n^d$$

where h_n is an edge of hypercube.

. An window function is defined by

$$\psi(\underline{x}) = \begin{cases} 1 & \text{if } |x_i| \leq 1/2,\ i = 1, \cdots, d \\ 0 & otherwise \end{cases}$$

a unit hypercube centered at the origin.

. an estimate of density function:

$$p_n(\underline{x}) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{V_n}\psi\left(\frac{\underline{x} - \underline{x}_i}{h_n}\right) \quad \cdots \quad (1)$$

where n represents the number of samples and $\underline{x}_i$ represents the ith sample. This estimate converges to $p(\underline{x})$ if

$$\lim_{n\to\infty} E[p_n(\underline{x})] = p(\underline{x}) \quad \text{and} \quad \lim_{n\to\infty} Var(p_n(\underline{x})) = 0.$$

. From the convergence condition, we can deduce the following conditions of (1):

$$\max_{\underline{x}} \psi(\underline{x}) < \infty, \quad \lim_{\|\underline{x}\|\to\infty} \psi(\underline{x}) \prod_{i=1}^{d} x_i = 0,$$

$$\lim_{n\to\infty} V_n = 0 \quad \text{and} \quad \lim_{n\to\infty} n V_n = \infty.$$

That is, V_n should be changed as follows:

$$V_n = \frac{V_1}{\sqrt{n}} \quad \text{or} \quad V_n = \frac{V_1}{\log n}. \quad \cdots \quad (2)$$

. The condition of $\lim_{n\to\infty} E[p_n(\underline{x})] = p(\underline{x})$:

$$\begin{aligned}
\lim_{n\to\infty} E[p_n(\underline{x})] &= \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} E[\frac{1}{V_n}\psi(\frac{\underline{x}-\underline{x}_i}{h_n})] \\
&= \lim_{n\to\infty} \int \frac{1}{V_n}\psi(\frac{\underline{x}-\underline{v}}{h_n})p(\underline{v})d\underline{v} \\
&= \int \delta(\underline{x}-\underline{v})p(\underline{v})d\underline{v} \\
&= p(\underline{x})
\end{aligned}$$

. The condition of $\lim_{n\to\infty} Var(p_n(\underline{x})) = 0$:

$$\begin{aligned}
Var(p_n(\underline{x})) &= E[p_n^2(\underline{x})] - E^2[p_n(\underline{x})] \\
&= E[(\sum_{i=1}^{n} \frac{1}{n V_n}\psi(\frac{\underline{x}-\underline{x}_i}{h_n}))^2] - E^2[p_n(\underline{x})] \\
&= \sum_{i=1}^{n} E[\frac{1}{n^2 V_n^2}\psi^2(\frac{\underline{x}-\underline{x}_i}{h_n})] - E^2[p_n(\underline{x})] \\
&= \frac{1}{n V_n}\int \frac{1}{V_n}\psi^2(\frac{\underline{x}-\underline{v}}{h_n})p(\underline{v})d\underline{v} - E^2[p_n(\underline{x})] \\
&\leqq \frac{\max(\psi)E[p_n(\underline{x})]}{n V_n}
\end{aligned}$$

Therefore, if V_n satisfies the condition of (2),

$$\lim_{n\to\infty} Var(p_n(\underline{x})) = 0.$$

B. Learning strategies of estimating continuos functions

Case 1: fixed centers selected at random (Lowe, 1989)

Estimation function is represented by

$$\phi(\underline{x}) = \sum_{i=1}^{m} w_i \psi_i(\underline{x})$$

where

$$\psi_i(\underline{x}) = e^{-m \| \underline{x} - \underline{t}_i \|^2 / d_{\max}^2},$$

$\underline{t}_i$ = the ith chosen center, and

$d_{\max}$ = the maximum distance between chosen centers.

The output weights are determined as follows:

Let

$$\underline{w} = [w_1, w_2, \cdots, w_m]^T,$$

$$\underline{\psi}_k = [\psi_1(\underline{x}_k), \psi_2(\underline{x}_k), \cdots, \psi_m(\underline{x}_k)]^T,$$

$R = E[\underline{\psi}_k \underline{\psi}_k^T]$, and $P = E[d_k \underline{\psi}_k]$.

Then,

$$\underline{w}^* = R^{-1} P$$

The distribution of $\underline{t}_i$ is important to the performance of RBFN.

From this point of view, this method does not give the stable performance.

Case 2: k-means clustering algorithm to select kernel centers (Moody and Darken, 1989)

Estimation function is represented by

$$\phi(\underline{x}) = \sum_{i=1}^{m} w_i \psi_i(\underline{x})$$

where

$$\psi_i(\underline{x}) = e^{-\| \underline{x} - \underline{t}_i \|^2 / \sigma_i^2},$$

$\underline{t}_i$ = the ith chosen center, and

σ_i = the ith kernel width.

Phase 1: k-means clustering algorithm to select kernel centers

Step 1. Let $\{\underline{t}_k(n)\}_{k=1}^{m}$ denotes the kernel centers at the nth iteration. Choose $\underline{t}_k(0)$ randomly.

Step 2. Draw a sample vector $\underline{x}$ from a sample set S with a certain probability.

Step 3. Find the index of the best matching:

$$k(\underline{x}) = \arg\min_k \| \underline{x}(n) - \underline{t}_k(n) \| \quad \text{for } k = 1, \cdots, m.$$

Step 4. Update the kernel centers:

$$\underline{t}_k(n+1) = \begin{cases} \underline{t}_k(n) + \eta[\underline{x}(n) - \underline{t}_k(n)] & \text{if } k(\underline{x}) = k \\ \underline{t}_k(n) & otherwise \end{cases}$$

Step 5. If there is no noticeable change in $\underline{t}_k$s, stop. Otherwise, $n \leftarrow n+1$ and go to Step 2.

Phase 2: Determine kernel widths: for the ith kernel function

$$\psi_i(\underline{x}) = e^{-\|\underline{x}-\underline{t}_i\|^2/\sigma_i^2},$$

choose the kernel width such as

$$\sigma_i \propto \min_{j \neq i} \| \underline{t}_i - \underline{t}_j \| \quad \text{for } j = 1, \cdots, m.$$

Phase 3: Determine the output weights:

$$\underline{w}^* = R^{-1}P$$

In this method, kernel centers are distributed according to the distribution of samples. However, it does not consider the output of the target function.

cf. Applying genetic algorithm (GA) for selecting the kernel centers (Billings, 1995):

GA is the global search algorithm and it tends to find the optimal distribution of kernel centers. However, this method requires exhaustive search of kernel centers.

Case 3: Supervised selection of kernel centers (Lee and Kil, 1991; Kil, 1993)

Estimation function is represented by

$$\phi(\underline{x}) = \sum_{i=1}^{m} w_i \psi_i(\underline{x})$$

where

$$\psi_i(\underline{x}) = e^{-\| \underline{x} - \underline{t}_i \|^2 / \sigma_i^2},$$

$\underline{t}_i$ = the ith chosen center, and

σ_i = the ith kernel width.

Learning algorithm of RBFN

Phase 1: recruiting the necessary number of kernel functions

Step 1. Construct a RBFN with a small number of kernel functions (1 or 2), that is, $i\ (kernel\ index) = 1$ or 2.

Step 2. For each sample, do the following procedure:

(1) if $\underline{x}_k$ generates big error (or maximum error for $k = 1, \cdots, n$),

$i \leftarrow i+1$, $\underline{t}_i = \underline{x}_k$, and

$\sigma_i \propto \min_{j \neq i} \| \underline{t}_i - \underline{t}_j \|$ for $j = 1, \cdots, i$

(2) update output weights: weight parameters are update to incorporate the given sample $(\underline{x}_k, y_k)$.

(3) if satisfied, go to the next phase of learning.
Otherwise, go to Step 2.

Phase 2: fine tuning of parameters of RBFN

Here, we assume that the weight vector $\underline{w}$ is near the optimal weight vector $\underline{w}^*$. Each output sample can be represented by

$$y_k = \hat{y_k}(\underline{w}) + n_k$$

where n_k represents Gaussian noise. An estimator of output $\hat{y_k}$ can be approximated as

$$\hat{y_k}(\underline{w}) \approx \hat{y_k}(\underline{w}_0) + [\frac{\partial \hat{y_k}}{\partial \underline{w}}|_{\underline{w}=\underline{w}_0}]^T(\underline{w} - \underline{w}_0).$$

Let

$$\begin{aligned}\tilde{y_k} &= y_k - (\hat{y_k}(\underline{w}_0) - [\frac{\partial \hat{y_k}}{\partial \underline{w}}|_{\underline{w}=\underline{w}_0}]^T \underline{w}_0) \\ &= [\frac{\partial \hat{y_k}}{\partial \underline{w}}|_{\underline{w}=\underline{w}_0}]^T \underline{w} + n_k \\ &= \underline{h}_k^T \underline{w} + n_k\end{aligned}$$

where

$$\underline{h}_k = \frac{\partial \hat{y_k}}{\partial \underline{w}}|_{\underline{w}=\underline{w}_0}.$$

Then, we can apply the RLS algorithm for the above equation.

The update rule for weight vector:

$$\begin{aligned}\underline{w}_{k+1} &= \underline{w}_k + \underline{a}_k(\widetilde{y_k} - \underline{h}_k^T \underline{w}_k) \\ &= \underline{w}_k + \underline{a}_k(y_k - (\widehat{y_k}(\underline{w}_0) + \underline{h}_k^T(\underline{w}_k - \underline{w}_0))) \\ &= \underline{w}_k + \underline{a}_k(y_k - \widehat{y_k}(\underline{w}_k))\end{aligned}$$

$$\underline{a}_k = B_k \underline{h}_k$$

$$B_k = B_{k-1} - \frac{(B_{k-1}\underline{h}_k)(B_{k-1}\underline{h}_k)^T}{1 + \underline{h}_k^T B_{k-1} \underline{h}_k}$$

- Optimal structure of RBFNs

. Regularization methods: the complexity term is included in the error function and one of the supervised learning algorithms is applied. eg. weight decay or weight elimination method.

. Optimization methods: learning problem is reformulated as the optimization problem. eg. support vector machine (SVM) with kernel functions.

. Model selection methods: the confidence interval term indicating the bound on the difference between the true and empirical risks is derived and applied to the optimization of regression models. eg. AIC, BIC, SEB, MCIC, etc.

- Applications of RBFNs

. probability estimation
. pattern recognition
. function approximation
. nonlinear time series prediction
. dynamic reconstruction of chaotic processes

Recurrent Neural Networks (RNNs)

- Hopfield Networks
- Simulated Annealing
- Boltzmann Machine
- Types of RNNs

Hopfield Networks (HNs)

- Associative Memory

. Store a set of p patterns

$\xi_i^\mu, \mu = 1, \cdots, p$ (pattern index), $i = 1, \cdots, n$ (unit index)

in such a way that when presented with a new pattern ζ_i, the network responds by producing whichever one of stored patterns most closely resembles ζ_i.

. The model:

The activation value of the ith unit:

$$S_i = sgn\left(\sum_{j=1}^{n} w_{ij} S_j - \theta_i\right),$$

where

$$sgn(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & otherwise \end{cases}$$

w_{ij}: the connection weight from the jth unit to the ith unit

θ_i: the bias term of the ith unit

In the Hopfield model,

$w_{ij} = w_{ji}$ and $w_{ii} = 0$

. Storing patterns

(1) stability condition for storing one pattern $\xi_i \in (-1, +1)$:

$$sgn\left(\sum_{j=1}^{n} w_{ij}\xi_j\right) = \xi_i \quad \forall i$$

If $w_{ij} \propto \xi_i\xi_j$ (Hebb's rule), the stability condition is satisfied since $\xi_j^2 = 1$. For convenience,

$$w_{ij} = \frac{1}{n}\xi_i\xi_j.$$

Then, the pattern ξ_i is referred to as an attractor and the pattern $-\xi_i$ (a reverse state) is also an attractor.

(2) storing many patterns:

Make w_{ij} as a superposition of terms; that is,

$$w_{ij} = \frac{1}{n}\sum_{\mu=1}^{p} \xi_i^\mu \xi_j^\mu,$$

where p represents the total number stored patterns.

The stability of a particular pattern ξ_i^ν:

$$sgn(h_i^\nu) = \xi_i^\nu \quad \forall i,$$

where

$$h_i^\nu = \sum_{j=1}^{n} w_{ij}\xi_j^\nu = \frac{1}{n}\sum_{j=1}^{n}\sum_{\mu=1}^{p} \xi_i^\mu \xi_j^\mu \xi_j^\nu$$

$$= \xi_i^\nu + \frac{1}{n}\sum_{j=1}^{n}\sum_{\mu\neq\nu}^{p} \xi_i^\mu \xi_j^\mu \xi_j^\nu \ (crosstalk\ term)$$

(3) storage capacity

Let $C_i^\nu = -\xi_i^\nu \cdot crosstalk\ term$

$$= -\xi_i^\nu \cdot \frac{1}{n}\sum_{j=1}^{n}\sum_{\mu\neq\nu}^{p}\xi_i^\mu\xi_j^\mu\xi_j^\nu$$

Then, if $C_i^\nu < 0$, the stability condition is satisfied

if $C_i^\nu > 1$, it changes the sign of h_i^ν and makes the ith bit of pattern ν unstable.

Assuming purely random patterns, with equal probability for $\xi_i^\mu = 1$ and $\xi_i^\mu = -1$, independently for each i and μ. Then, the probability P_{err} that any chosen bit is unstable is

$P_{err} = P(C_i^\nu > 1)$.

Here, C_i^ν is $1/n$ times the sum of about np independent random numbers, each of which is +1 or −1. Then, for large np,

$C_i^\nu \sim N(0,\sigma^2)$, where $\sigma^2 = \frac{p}{n}$ by CLT.

In this case,

$$P_{err} = \frac{1}{\sqrt{2\pi}\,\sigma}\int_1^\infty e^{-\frac{x^2}{2\sigma^2}}dx = \frac{1}{2}\left(1 - erf\left(\sqrt{\frac{n}{2p}}\right)\right),$$

where

$$erf(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-u^2}du.$$

Let $P_{err} < 0.01$. Then, $p_{\max} \approx 0.15n$ (maximum storage capacity).

Here, $p_{\max}$ is an upper bound since it only tells the initial stability.

- Energy Function

. Let us define the energy function H as

$$H = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} S_i S_j .$$

Then, it always decreases (or remains constant) as the system evolves according to its dynamical rule; that is, the attractors (or memorized patterns) are at the local minima of the energy function.

. Let S_i' be the new value of S_i for some particular unit i. Then,

$$S_i' = sgn\left(\sum_{j=1}^{n} w_{ij} S_j\right)$$

If $S_i' = S_i$, then $H' - H = 0$

If $S_i' = -S_i$, then

$$\begin{aligned} H' - H &= -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} S_i' S_j + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} S_i S_j \\ &= S_i \sum_{j=1}^{n} w_{ij} S_j < 0 \end{aligned}$$

This implies that the energy decreases every time an S_i changes.

. Spurious states

(1) The reverse states are also minima and have the same energy as the original patterns.

(2) Stable mixture states corresponding to linear combination of an odd number of patterns are also stable states.

- Optimization problems

. The Hopfield network is expected to find a configuration which minimizes an energy function.

. The optimization problems are transformed into minimizing the energy function or cost (objective) function.

. Relaxation methods for optimization for the convergence to the global optimum:
(1) simulated annealing (Kirkpatric et al., 1983)
(2) mean field annealing (Soukoulis et al., 1983)

Simulated Annealing

. At each temperature T, the solid is allowed to reach the thermal equilibrium, characterized by a probability of being in a state with energy E given by the Boltzmann distribution:

$$P(E) = \frac{1}{Z(T)} e^{-E/(k_B T)},$$

where $Z(T)$ represents a normalization factor (or partition function), k_B represents a Boltzmann constant, and $e^{-E/(k_B T)}$ represents a Boltzmann factor.

. The Boltzmann distribution is used to simulate the evolution to the thermal equilibrium of a solid for the temperature T.

- Simulated Annealing Algorithm (Metropolis, 1953)

. A sequence of Metropolis algorithm is evaluated at a sequence of (decreasing) temperature T. Then, the probability of configuration j given the configuration i in the next sequence is given by

$$P(j|i) = \begin{cases} 1 & \text{if } \Delta C_{ij} \leqq 0 \\ e^{(-\Delta C_{ij}/C)} & otherwise \end{cases},$$

where $\Delta C_{ij} = C(j) - C(i)$ (difference of cost functions)

(Metropolis criterion) In the equilibrium state,

$$P(i) = \frac{1}{Q(C)} e^{(-C(i)/C)},$$

where $Q(C)$ represents a normalization constant and C is a control parameter (temperature).

Simulate Annealing Procedure

```
begin
  initialize parameters; m = 0; (scheduling index)
  repeat
    repeat
      perturb (config i -> config j, ΔC_ij);
      if ΔC_ij ≦ 0, then accept;
      else if e^(-ΔC_ij/C) > Random[0,1), then accept;
      if accept, then update (config j);
    until equilibrium;
    C_(m+1) = f(C_m); m = m+1;
  until stop criterion = true (system is frozen);
end;
```

. This process can be described by inhomogeneous Markov chain. For the convergence to the global optimum, the following conditions (sufficient conditions) should be satisfied:

$\lim_{k\to\infty} C_k = 0$ and $C_k \geqq O\left(\frac{1}{\log k}\right)$

(Laarhoven and Aarts, "Simulated Annealing," 1987)

. For the fast simulated annealing, the Cauchy-Lorentz visiting distribution can be used:

$\frac{C_k}{C_k + (\Delta C_{ij})^2}$ is used instead of $e^{-\Delta C_{ij}/C_k}$,

where $C_k = T_0/k$.

(Szu and Hartley, Physics Letter A, 122:157, 1987)

- Boltzmann Annealing

. Optimization in non-convex cost functions

1. $g(\boldsymbol{x})$: probability density of state-space of D parameters $\boldsymbol{x} = \{x_1, x_2, \cdots, x_D\}$
2. $h(\boldsymbol{x})$: probability density for the acceptance of new cost function given the previous value
3. $T(k)$: schedule of annealing for the temperature T in annealing-time steps k

. The class of Gaussian-Markovian systems

$$g(\boldsymbol{x}) = \frac{1}{Z(T)} e^{(-E(\boldsymbol{x})/(k_B T))} = \frac{1}{(2\pi T)^{D/2}} e^{(-\Delta \boldsymbol{x}^2/(2T))},$$

where $\Delta x = x - x_0$ (the deviation of x from x_0)

. The acceptance probability: chances of obtaining a new state E_{k+1} relative to a previous state E_k

$$h(\boldsymbol{x}) = \frac{e^{(-E_{k+1}/T)}}{e^{(-E_{k+1}/T)} + e^{(-E_k/T)}} = \frac{1}{1+e^{(\Delta E/T)}},$$

where $\Delta E = E_{k+1} - E_k$.

. Given $g(\boldsymbol{x})$, it is sufficient to obtain a global minimum of $E(\boldsymbol{x})$ if T is selected to be not faster than $T(k) = T_0/\log k$.

$$g_k = \frac{1}{Z(T)} e^{(-E/(k_B T(k)))}$$

(H. Szu and R. Hartley, 1987)

. In order to statistically assure, any point in x space can be sampled infinitely often in annealing time:

$$\prod_{k=k_0}^{\infty}(1-g_k) = 0 \text{ and } \sum_{k=k_0}^{\infty} g_k = \infty.$$

Let $E = k_B T_0$. Then,

$$\sum_{k=k_0}^{\infty} g_k = \frac{1}{Z(T)} \sum_{k=k_0}^{\infty} e^{(-\log k)} = \frac{1}{Z(T)} \sum_{k=k_0}^{\infty} \frac{1}{k} = \infty.$$

- Fast Annealing

. The Cauchy distribution:

$$g_k(x) = \frac{T(k)}{(\Delta x^2 + T^2(k))^{(D+1)/2}},$$

where

$$T(k) = \frac{T_0}{k}.$$

. Statistical assurance for the convergence to the global optimum:

$$\prod_{k=k_0}^{\infty}(1-g_k) = 0 \quad \text{and} \quad \sum_{k=k_0}^{\infty} g_k \approx \frac{T_0}{\Delta x^{D+1}} \sum_{k=k_0}^{\infty} \frac{1}{k} = \infty.$$

Boltzmann Machine (BM)

- The model

. There are visible (input and output) units and hidden units.

. All connections are symmetric; that is, $w_{ij} = w_{ji}$

. The units are stochastic:

$$P(S_i = 1) = \frac{1}{1+e^{-2\beta h_i}}, \text{ where}$$

$$h_i = \sum_{j=1}^{n} w_{ij} S_j \text{ and } \beta = \frac{1}{T}.$$

$$P(S_i = -1) = 1 - P(S_i = 1).$$

That is, the stochastic version of Hopfield network.

. After equilibrium, the probability of finding the system in a particular state $\{S_i\}$ is determined by the Boltzmann-Gibbs distribution:

$P\{S_i\} = \frac{1}{Z} e^{-\beta H\{S_i\}}$, where

$H\{S_i\} = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} S_i S_j$ (energy function) and

$Z = \sum_{S_i} e^{-\beta H\{S_i\}}$ (partition function)

- Learning of Boltzmann machine

The probability of finding visible units in state α irrespective of state β in the freely running system is

$P_\alpha = \sum_\beta P_{\alpha\beta} = \frac{1}{Z}\sum_\beta e^{-\beta H_{\alpha\beta}}$, where

$Z = \sum_\alpha \sum_\beta e^{-\beta H_{\alpha\beta}}, \quad H_{\alpha\beta} = -\frac{1}{2}\sum_i \sum_j w_{ij} S_i^{\alpha\beta} S_j^{\alpha\beta}$

For a set of desired probabilities R_α for these states, the relative entropy (Kullback-Leibler divergence) is defined by

$E = D(R_\alpha \| P_\alpha) = \sum_\alpha R_\alpha \log \frac{R_\alpha}{P_\alpha}$.

$E \geqq 0$ and $E = 0$ if and only if $P_\alpha = R_\alpha \quad \forall \alpha$.

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = \eta \sum_{\alpha} \frac{R_\alpha}{P_\alpha} \frac{\partial P_\alpha}{\partial w_{ij}}, \quad \frac{\partial P_\alpha}{\partial w_{ij}} = \beta \left(\sum_{\beta} S_i^{\alpha\beta} S_j^{\alpha\beta} P_{\alpha\beta} - P_\alpha \langle S_i S_j \rangle \right),$$

where

$$\langle S_i S_j \rangle = \frac{1}{Z} \sum_{\lambda} \sum_{\mu} e^{-\beta H_{\lambda\mu}} S_i^{\lambda\mu} S_j^{\lambda\mu}.$$

Therefore,

$$\begin{aligned} \Delta w_{ij} &= \eta\beta \left(\sum_{\alpha} \sum_{\beta} R_\alpha P_{\beta|\alpha} S_i^{\alpha\beta} S_j^{\alpha\beta} - \langle S_i S_j \rangle \right) \\ &= \eta\beta \left(\overline{\langle S_i S_j \rangle}_{clamped} - \langle S_i S_j \rangle_{free} \right) \end{aligned}$$

Here, $\overline{\langle S_i S_j \rangle}_{clamped}$ implies that the value of $\langle S_i S_j \rangle$ when the visible units are clamped in state α, averaged over α's according to their probabilities R_α.

- Operation of Boltzmann machine

Step 1. Wait until the equilibrium state for some temperature $T > 0$.

Step 2. Measure the correlation $\langle S_i S_j \rangle$ by taking a time average of $S_i S_j$.

Step 3. The visible units are clamped in each of α for which $R_\alpha > 0$.

Step 4. Wait until the equilibrium state.

Step 5. Measure the correlation $\overline{\langle S_i S_j \rangle}_{clamped}$.

Step 6. Network weights are updated by $w_{ij}^{\text{new}} = w_{ij}^{\text{old}} + \Delta w_{ij}$, where $\Delta w_{ij} = \eta\beta \left(\overline{\langle S_i S_j \rangle}_{clamped} - \langle S_i S_j \rangle_{free} \right)$.

- Applications

. Pros: The BM learns probabilities rather than particular patterns.

. Cons: long training time, local minima problem.
  -> One remedy is using the simulated annealing
     (or fast simulated annealing).

. The BM can be applied to
  various pattern classification problems,
  learning vector quantization (LVQ), and
  various combinatorial optimization problems.

Types of Recurrent Neural Networks (RNNs)

. A popular way to recognize (or reproduce) sequences has been to use partially recurrent networks.
. In most cases, the feedback connections are fixed, and updating is synchronous with one update for all units at each time step (sequential networks).
. There are context units that receive feedback signals.
. Examples:
(Jordan, 1986; Stornetta, 1988; Mozer, 1989; Elman, 1990)

- Backpropagation through time (BTT)

. Synchronous dynamics and discrete time:
the output of unit i at time $t+1$ is determined by

$$V_i(t+1) = f(h_i(t)) = f\left(\sum_{j=1}^{n} w_{ij} V_j(t) + \xi_i(t)\right),$$

where $\xi_i(t)$ represents the external input of the ith unit at time t.

. A sequence of maximum length T = A feedforward net with $T-1$ layers -> For the learning of w_{ij}s', the back-propagation algorithm in MLP can be applied.
For long sequences, the approach becomes impractical due to duplication of units.

. Learning without duplicating units
(real-time recurrent learning (RTRL), Williams and Zipser, 1989)

The output of unit i at time t:

$$V_i(t) = f(h_i(t-1)) = f\left(\sum_{j=1}^{n} w_{ij} V_j(t-1) + \xi_i(t-1)\right) \quad \text{... (1)}$$

The error measure on unit k at time t:

$$E_k(t) = \begin{cases} \zeta_k(t) - V_k(t) & \text{if } \zeta_k(t) \text{ is the desired output at time } t \\ 0 & \text{otherwise} \end{cases}$$

The total cost function:

$$E = \sum_{t=0}^{T} E(t), \text{ where } E(t) = \frac{1}{2}\sum_{k} (E_k(t))^2$$

The gradient descent with respect to w_{pq} is determined by

$$\Delta w_{pq}(t) = -\eta \frac{\partial E(t)}{\partial w_{pq}} = \eta \sum_k E_k(t) \frac{\partial V_k(t)}{\partial w_{pq}} \quad \text{... (2)}$$

In the recurrent back-propagation algorithm,

$$\frac{\partial V_i(t)}{\partial w_{pq}} = f'(h_i(t-1)) \left(\delta_{ip} V_q(t-1) + \sum_j w_{ij} \frac{\partial V_j(t-1)}{\partial w_{pq}} \right).$$

Learning Algorithm:

Step 1. At each time step, update V_i according to (1).

Step 2. Update w_{pq} as $w_{pq}(t) + \Delta w_{pq}(t)$ using (2).

. Memory requirement: for N fully connected units, N^3 derivatives to maintain and updating each takes a time proportional to N or N^4 in all at each time step.

. If the learning rate η is sufficiently small, the real-time learning is possible. However, it may have stability problem.

. Applications:

sequence learning tasks -

sequence recognition (eg. speech recognition),

sequence reproduction (eg. learning a set of songs),

sequence (or temporal) association, etc.

Conclusion

- The analysis of data distribution is required before selecting the machine learning model.
- The given task (or machine learning problem) can be classified according to the class of learning models.
- The pros and cons of machine learning models should be investigated for the given task.
- The optimization of the structure of machine learning model is necessary to achieve the improved and stable performances (generalization capacity).