Artificial Neural Network : Training

Learning of neural networks: Topics

- Concept of learning
- Learning in
  - Single layer feed forward neural network
  - multilayer feed forward neural network
  - recurrent neural network
- Types of learning in neural networks

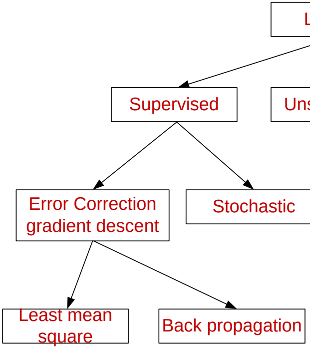
Concept of Learning

The concept of learning

- The learning is an important feature of human computational ability.
- Learning may be viewed as the change in behavior acquired due to practice or experience, and it lasts for relatively long time.
- As it occurs, the effective coupling between the neuron is modified.
- In case of artificial neural networks, it is a process of modifying neural network by updating its weights, biases and other parameters, if any.
- During the learning, the parameters of the networks are optimized and as a result process of curve fitting.
- It is then said that the network has passed through a learning phase.

Types of learning

- There are several learning techniques.
- A taxonomy of well known learning techniques are shown in the following.



Different learning techniques: Supervised learning

- **Supervised learning**

  In this learning, every input pattern that is used to train the network is associated with an output pattern.

- This is called ”training set of data”. Thus, in this form of learning, the input-output relationship of the training scenarios are available.

- Here, the output of a network is compared with the corresponding target value and the error is determined.

- It is then feed back to the network for updating the same. This results in an improvement.

- This type of training is called learning with the help of teacher.

Different learning techniques: Unsupervised learning

- **Unsupervised learning**

  If the target output is not available, then the error in prediction can not be determined and in such a situation, the system learns of its own by discovering and adapting to structural features in the input patterns.

- This type of training is called learning without a teacher.

Different learning techniques: Reinforced learning

- **Reinforced learning**

  In this techniques, although a teacher is available, it does not tell the expected answer, but only tells if the computed output is correct or incorrect. A reward is given for a correct answer computed and a penalty for a wrong answer. This information helps the network in its learning process.

- **Note :** Supervised and unsupervised learnings are the most popular forms of learning. Unsupervised learning is very common in biological systems.

  It is also important for artificial neural networks : training data are not always available for the intended application of the neural network.

Different learning techniques : Gradient descent learning

- **Gradient Descent learning :**
  This learning technique is based on the minimization of error E defined in terms of weights and the activation function of the network.
- Also, it is required that the activation function employed by the network is differentiable, as the weight update is dependent on the gradient of the error E.
- Thus, if ΔW_{ij} denoted the weight update of the link connecting the i-th and j-th neuron of the two neighboring layers then

  $$\Delta W_{ij} = \eta \frac{\partial E}{\partial W_{ij}}$$

  where η is the **learning rate parameter** and $\frac{\partial E}{\partial W_{ij}}$ is the **error gradient** with reference to the weight W_{ij}
- The least mean square and back propagation are two variations of this learning technique.

Different learning techniques : Stochastic learning

- **Stochastic learning**

  In this method, weights are adjusted in a probabilistic fashion. Simulated annealing is an example of such learning (proposed by Boltzmann and Cauch)

Different learning techniques: Hebbian learning

Hebbian learning

- This learning is based on correlative weight adjustment. This is, in fact, the learning technique inspired by biology.
- Here, the input-output pattern pairs (x_i, y_i) are associated with the weight matrix W. W is also known as the correlation matrix.
- This matrix is computed as follows.

  $$W = \sum_{i=1}^{n} X_i Y_i^T$$

  where Y_i^T is the transpose of the associated vector y_i

Different learning techniques : Competitive learning

- **Competitive learning**

  In this learning method, those neurons which responds strongly to input stimuli have their weights updated.

- When an input pattern is presented, all neurons in the layer complete and the winning neuron undergoes weight adjustment.
- This is why it is called a Winner-takes-all strategy.

In this course, we discuss a generalized approach of supervised learning to train different type of neural network architectures.

Training SLFFNNs

Single layer feed forward NN training

- We know that, several neurons are arranged in one layer with inputs and weights connect to every neuron.
- Learning in such a network occurs by adjusting the weights associated with the inputs so that the network can classify the input patterns.
- A single neuron in such a neural network is called perceptron.
- The algorithm to **train a perceptron** is stated below.
- Let there is a perceptron with $(n+1)$ inputs $x_0, x_1, x_2, \cdots, x_n$
- where $x_0 = 1$ is the bias input.
- Let f denotes the transfer function of the neuron. Suppose, $\bar{X}$ and $\bar{Y}$ denotes the input-output vectors as a training data set. $\bar{W}$ denotes the weight matrix.

With this input-output relationship pattern and configuration of a perceptron, the algorithm **Training Perceptron** to train the perceptron is stated in the following slide.

Single layer feed forward NN training

1. Initialize $\bar{W} = w_0, w_1, \cdots, w_n$ to some random weights.
2. For each input pattern $x \in \bar{X}$ do Here, $x = \{x_0, x_1, ...x_n\}$
   - Compute $I = \sum_{i=0}^{n} w_i x_i$
   - Compute observed output y

$$y = f(I) = \begin{cases} 1 & , \text{if } I > 0 \\ 0 & , \text{if } I \leq 0 \end{cases}$$

$$\bar{Y'} = \bar{Y'} + y$$ Add y to $\bar{Y'}$, which is initially empty

3. If the desired output $\bar{Y}$ matches the observed output $\bar{Y'}$ then output $\bar{W}$ and exit.
4. Otherwise, update the weight matrix $\bar{W}$ as follows :
   - For each output $y \in \bar{Y'}$ do
   - If the observed out y is 1 instead of 0, then $w_i = w_i - \alpha x_i$, $(i = 0, 1, 2, \cdots n)$
   - Else, if the observed out y is 0 instead of 1, then $w_i = w_i + \alpha x_i$, $(i = 0, 1, 2, \cdots n)$
5. Go to step 2.

Single layer feed forward NN training

In the above algorithm, α is the learning parameter and is a constant decided by some empirical studies.

Note :

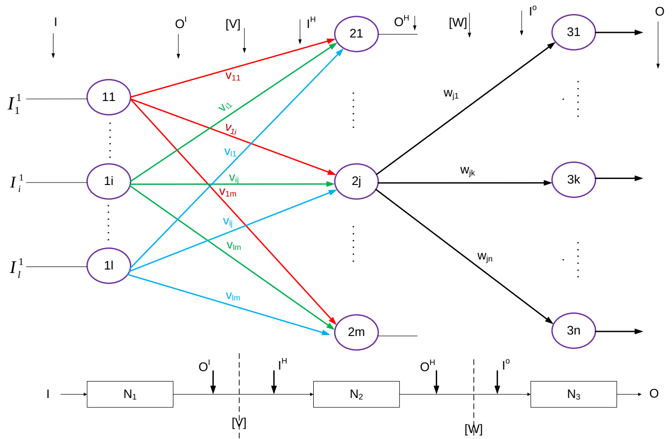
- The algorithm Training Perceptron is based on the supervised learning technique
- ADALINE : Adaptive Linear Network Element is also an alternative term to perceptron
- If there are 10 number of neutrons in the single layer feed forward neural network to be trained, then we have to iterate the algorithm for each perceptron in the network.

Training MLFFNNs

Training multilayer feed forward neural network

- Like single layer feed forward neural network, supervisory training methodology is followed to train a multilayer feed forward neural network.
- Before going to understand the training of such a neural network, we redefine some terms involved in it.
- A block digram and its configuration for a three layer multilayer FF NN of type $l - m - n$ is shown in the next slide.

Specifying a MLFFNN



Input layer	Hidden Layer	Output Layer
$\lvert N_1 \rvert = l$	$\lvert N_2 \rvert = m$	$\lvert N_3 \rvert = n$
Linear transfer function, ɵ	Log-Sigmoid transfer function	Tan-Sigmoid transfer function
$f_i^l = (I_i, \theta_i)$	$f_j^m = (I_j^H, \alpha_j) = \dfrac{1}{1+e^{-\alpha_j I_j^H}}$	$f_k^n = (I_k^o, \alpha_k) = \dfrac{e^{+\alpha_o I_o} - e^{-\alpha_o I_o}}{e^{+\alpha_o I_o} + e^{-\alpha_o I_o}}$

Specifying a MLFFNN

- For simplicity, we assume that all neurons in a particular layer follow same transfer function and different layers follows their respective transfer functions as shown in the configuration.
- Let us consider a specific neuron in each layer say i-th, j-th and k-th neurons in the input, hidden and output layer, respectively.
- Also, let us denote the weight between i-th neuron ($i = 1, 2, \cdots, l$) in input layer to j-th neuron ($j = 1, 2, \cdots, m$) in the hidden layer is denoted by v_{ij}.
- The weight matrix between the input to hidden layer say V is denoted as follows.

$$V = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1j} & \cdots & v_{1m} \\ v_{21} & v_{22} & \cdots & v_{2j} & \cdots & v_{2m} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ v_{i1} & v_{i2} & \cdots & v_{ij} & \cdots & v_{im} \\ v_{l1} & v_{l2} & \cdots & v_{lj} & \cdots & v_{lm} \end{bmatrix}$$

Specifying a MLFFNN

- Similarly, w_{jk} represents the connecting weights between $j-th$ neuron($j = 1, 2, \cdots, m$) in the hidden layer and k-th neuron ($k = 1, 2, \cdots n$) in the output layer as follows.

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1k} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2k} & \cdots & w_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{j1} & w_{j2} & \cdots & w_{jk} & \cdots & w_{jn} \\ w_{m1} & w_{m2} & \cdots & w_{mk} & \cdots & w_{mn} \end{bmatrix}$$

Learning a MLFFNN

Whole learning method consists of the following three computations:

1. **Input layer computation**
2. **Hidden layer computation**
3. **Output layer computation**

In our computation, we assume that $< T_0, T_I >$ be the training set of size $|T|$.

Input layer computation

- Let us consider an input training data at any instant be $I^I = [I_1^1, I_2^1, \cdots, I_i^1, I_l^1]$ where $I^I \in T_I$
- Consider the outputs of the neurons lying on input layer are the same with the corresponding inputs to neurons in hidden layer. That is,

$$O^I = I^I$$
$$[l \times 1] = [l \times 1] \qquad \text{[Output of the input layer]}$$

- The input of the j-th neuron in the hidden layer can be calculated as follows.

$$I_j^H = v_{1j}o_1^I + v_{2j}o_2^I +, \cdots, +v_{ij}o_j^I + \cdots + v_{lj}o_l^I$$

where $j = 1, 2, \cdots m$.

[Calculation of input of each node in the hidden layer]

- In the matrix representation form, we can write

$$I^H = V^T \cdot O^I$$
$$[m \times 1] = [m \times l]\ [l \times 1]$$

Hidden layer computation

- Let us consider any j-th neuron in the hidden layer.
- Since the output of the input layer's neurons are the input to the j-th neuron and the j-th neuron follows the log-sigmoid transfer function, we have

  $$O_j^H = \frac{1}{1+e^{-\alpha_H \cdot I_j^H}}$$

  where $j = 1, 2, \cdots, m$ and α_H is the constant co-efficient of the transfer function.

Hidden layer computation

Note that all output of the nodes in the hidden layer can be expressed as a one-dimensional column matrix.

$$O^H = \begin{bmatrix} \cdots \\ \cdots \\ \vdots \\ \frac{1}{1+e^{-\alpha_H \cdot I_j^H}} \\ \vdots \\ \cdots \\ \cdots \end{bmatrix}_{m\times 1}$$

Output layer computation

Let us calculate the input to any k-th node in the output layer. Since, output of all nodes in the hidden layer go to the k-th layer with weights $w_{1k}, w_{2k}, \cdots, w_{mk}$, we have

$$I_k^O = w_{1k} \cdot o_1^H + w_{2k} \cdot o_2^H + \cdots + w_{mk} \cdot o_m^H$$

where $k = 1, 2, \cdots, n$

In the matrix representation, we have

$$I^O = W^T \cdot O^H$$
$$[n \times 1] = [n \times m]\ [m \times 1]$$

Output layer computation

Now, we estimate the output of the k-th neuron in the output layer. We consider the tan-sigmoid transfer function.

$$O_k = \frac{e^{\alpha_o \cdot I_k^o} - e^{-\alpha_o \cdot I_k^o}}{e^{\alpha_o \cdot I_k^o} + e^{-\alpha_o \cdot I_k^o}}$$

for $k = 1, 2, \cdots, n$

Hence, the output of output layer's neurons can be represented as

$$O = \begin{bmatrix} \cdots \\ \cdots \\ \vdots \\ \frac{e^{\alpha_o \cdot I_k^o} - e^{-\alpha_o \cdot I_k^o}}{e^{\alpha_o \cdot I_k^o} + e^{-\alpha_o \cdot I_k^o}} \\ \vdots \\ \cdots \\ \cdots \end{bmatrix}_{n \times 1}$$

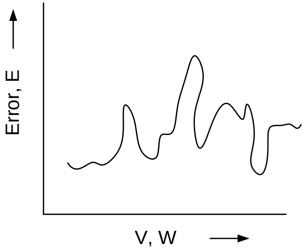
Back Propagation Algorithm

- The above discussion comprises how to calculate values of different parameters in $l - m - n$ multiple layer feed forward neural network.
- Next, we will discuss how to train such a neural network.
- We consider the most popular algorithm called Back-Propagation algorithm, which is a supervised learning.
- The principle of the **Back-Propagation algorithm** is based on the error-correction with **Steepest-descent method**.
- We first discuss the method of steepest descent followed by its use in the training algorithm.

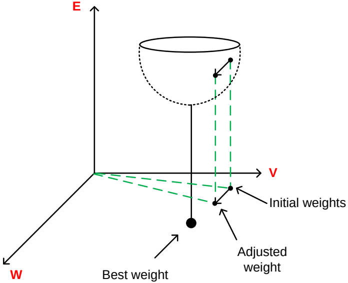
Method of Steepest Descent

- Supervised learning is, in fact, error-based learning.
- In other words, with reference to an external (teacher) signal (i.e. target output) it calculates error by comparing the target output and computed output.
- Based on the error signal, the neural network should modify its configuration, which includes synaptic connections, that is , the weight matrices.
- It should try to reach to a state, which yields minimum error.
- In other words, its searches for a suitable values of parameters minimizing error, given a training set.
- Note that, this problem turns out to be an optimization problem.

Method of Steepest Descent



(a) Searching for a minimum error



(b) Error surface with two parameters V and W

Method of Steepest Descent

- For simplicity, let us consider the connecting weights are the only design parameter.
- Suppose, V and W are the wights parameters to hidden and output layers, respectively.
- Thus, given a training set of size N, the error surface, E can be represented as

  $$E = \sum_{i=1}^{N} e^{i}\left(V, W, I_i\right)$$

  where I_i is the i-th input pattern in the training set and $e^{i}(...)$ denotes the error computation of the i-th input.
- Now, we will discuss the steepest descent method of computing error, given a changes in V and W matrices.

Method of Steepest Descent

- Suppose, A and B are two points on the error surface (see figure in Slide 30). The vector $\vec{AB}$ can be written as

$$\vec{AB} = (V_{i+1} - V_i) \cdot \bar{x} + (W_{i+1} - W_i) \cdot \bar{y} = \Delta V \cdot \bar{x} + \Delta W \cdot \bar{y}$$

The gradient of $\vec{AB}$ can be obtained as

$$e_{\vec{AB}} = \frac{\partial E}{\partial V} \cdot \bar{x} + \frac{\partial E}{\partial W} \cdot \bar{y}$$

Hence, the unit vector in the direction of gradient is

$$\bar{e}_{\vec{AB}} = \frac{1}{|e_{\vec{AB}}|} \left[\frac{\partial E}{\partial V} \cdot \bar{x} + \frac{\partial E}{\partial W} \cdot \bar{y} \right]$$

Method of Steepest Descent

- With this, we can alternatively represent the distance vector AB as

$$\vec{AB} = \eta \left[\frac{\partial E}{\partial V} \cdot \bar{x} + \frac{\partial E}{\partial W} \cdot \bar{y} \right]$$

  where $\eta = \frac{k}{|e_{\vec{AB}}|}$ and k is a constant

- So, comparing both, we have

$$\Delta V = \eta \frac{\partial E}{\partial V}$$
$$\Delta W = \eta \frac{\partial E}{\partial W}$$

  This is also called as **delta rule** and η is called **learning rate**.

Calculation of error in a neural network

- Let us consider any k-th neuron at the output layer. For an input pattern $I_i \in T_I$ (input in training) the target output T_{O_k} of the k-th neuron be T_{O_k}.
- Then, the error e_k of the k-th neuron is defined corresponding to the input I_i as

$$e_k = \frac{1}{2}\left(T_{O_k} - O_{O_k}\right)^2$$

where O_{O_k} denotes the observed output of the k-th neuron.

Calculation of error in a neural network

- For a training session with $I_i \in T_I$, the error in prediction considering all output neurons can be given as

$$e = \sum_{k=1}^{n} e_k = \frac{1}{2}\sum_{k=1}^{n} (T_{Ok} - O_{Ok})^2$$

where n denotes the number of neurons at the output layer.

- The total error in prediction for all output neurons can be determined considering all training session $< T_I, T_O >$ as

$$E = \sum_{\forall I_i \in T_I} e = \frac{1}{2}\sum_{\forall t \in < T_I, T_O >} \sum_{k=1}^{n} (T_{Ok} - O_{Ok})^2$$

Supervised learning : Back-propagation algorithm

- The back-propagation algorithm can be followed to train a neural network to set its topology, connecting weights, bias values and many other parameters.
- In this present discussion, we will only consider updating weights.
- Thus, we can write the error E corresponding to a particular training scenario T as a function of the variable V and W. That is

$$E = f(V, W, T)$$

- In BP algorithm, this error E is to be minimized using the gradient descent method. We know that according to the gradient descent method, the changes in weight value can be given as

$$\Delta V = -\eta \frac{\partial E}{\partial V} \tag{1}$$

and

$$\Delta W = -\eta \frac{\partial E}{\partial W} \tag{2}$$

Supervised learning : Back-propagation algorithm

- Note that $-ve$ sign is used to signify the fact that if $\frac{\partial E}{\partial V}$ (or $\frac{\partial E}{\partial W}$) > 0, then we have to decrease V and vice-versa.
- Let v_{ij} (and w_{jk}) denotes the weights connecting i-th neuron (at the input layer) to j-th neuron(at the hidden layer) and connecting j-th neuron (at the hidden layer) to k-th neuron (at the output layer).
- Also, let e_k denotes the error at the k-th neuron with observed output as $O_{O_k^o}$ and target output $T_{O_k^o}$ as per a sample intput $I \in T_I$.

Supervised learning : Back-propagation algorithm

- It follows logically therefore,

  $$e_k = \frac{1}{2}(T_{O_k^o} - O_{O_k^o})^2$$

  and the weight components should be updated according to equation (1) and (2) as follows,

  $$\bar{w_{jk}} = w_{jk} + \Delta w_{jk} \tag{3}$$

  where $\Delta w_{jk} = -\eta \frac{\partial e_k}{\partial w_{jk}}$
  and

  $$\bar{v_{ij}} = v_{ij} + \Delta v_{ij} \tag{4}$$

  where $\Delta v_{ij} = -\eta \frac{\partial e_k}{\partial v_{ij}}$
- Here, v_{ij} and w_{ij} denotes the previous weights and $\bar{v_{ij}}$ and $\bar{w_{ij}}$ denote the updated weights.
- Now we will learn the calculation $\bar{w_{ij}}$ and $\bar{v_{ij}}$, which is as follows.

Calculation of $\bar{w_{jk}}$

We can calculate $\frac{\partial e_k}{\partial w_{jk}}$ using the chain rule of differentiation as stated below.

$$\frac{\partial e_k}{\partial w_{jk}} = \frac{\partial e_k}{\partial O_{O_k^o}} \cdot \frac{\partial O_{O_k^o}}{\partial I_k^o} \cdot \frac{\partial I_k^o}{\partial w_{jk}} \tag{5}$$

Now, we have

$$e_k = \frac{1}{2}(T_{O_k^o} - O_{O_k^o})^2 \tag{6}$$

$$O_{O_k^o} = \frac{e^{\theta_o I_k^o} - e^{-\theta_o I_k^o}}{e^{\theta_o I_k^o} + e^{-\theta_o I_k^o}} \tag{7}$$

$$I_k^o = w_{1k} \cdot O_1^H + w_{2k} \cdot O_2^H + \cdots + w_{jk} \cdot O_j^H + \cdots w_{mk} \cdot O_m^H \tag{8}$$

Calculation of $\bar{w_{jk}}$

Thus,

$$\frac{\partial e_k}{\partial O_{O_k^o}} = -(T_{O_k^o} - O_{O_k^o}) \tag{9}$$

$$\frac{\partial O_{o_k^o}}{\partial I_k^o} = \theta_o(1 + O_{O_k^o})(1 - O_{O_k^o}) \tag{10}$$

and

$$\frac{\partial I_k^o}{\partial w_{ij}} = O_j^H \tag{11}$$

Calculation of $\bar{w_{jk}}$

Substituting the value of $\frac{\partial e_k}{\partial O_{O_k^o}}$, $\frac{\partial O_{O_k^o}}{\partial I_k^o}$ and $\frac{\partial I_k^o}{\partial w_{jk}}$

we have

$$\frac{\partial e_k}{\partial w_{jk}} = -(T_{O_k^o} - O_{O_k^o}) \cdot \theta_o(1 + O_{O_k^o})(1 - O_{O_k^o}) \cdot O_j^H \quad (12)$$

Again, substituting the value of $\frac{\partial E_k}{\partial w_{jk}}$ from Eq. (12) in Eq.(3), we have

$$\Delta w_{jk} = \eta \cdot \theta_o(T_{O_k^o} - O_{O_k^o}) \cdot (1 + O_{O_k^o})(1 - O_{O_k^o}) \cdot O_j^H \quad (13)$$

Therefore, the updated value of w_{jk} can be obtained using Eq. (3)

$$\bar{w_{jk}} = w_{jk} + \Delta w_{jk} = \eta \cdot \theta_o(T_{O_k^o} - O_{O_k^o}) \cdot (1 + O_{O_k^o})(1 - O_{O_k^o}) \cdot O_j^H + w_{jk} \quad (14)$$

Calculation of $\bar{v}_{ij}$

Like, $\frac{\partial e_k}{\partial w_{jk}}$, we can calculate $\frac{\partial e_k}{\partial V_{ij}}$ using the chain rule of differentiation as follows,

$$\frac{\partial e_k}{\partial v_{ij}} = \frac{\partial e_k}{\partial O_{O_k^o}} \cdot \frac{\partial O_{O_k^o}}{\partial I_k^o} \cdot \frac{\partial I_k^o}{\partial O_j^H} \cdot \frac{\partial O_j^H}{\partial I_j^H} \cdot \frac{\partial I_j^H}{\partial v_{ij}} \tag{15}$$

Now,

$$e_k = \frac{1}{2}(T_{O_k^o} - O_{O_k^o})^2 \tag{16}$$

$$O_k^o = \frac{e^{\theta_o I_k^o} - e^{-\theta_o I_k^o}}{e^{\theta_o I_k^o} + e^{-\theta_o I_k^o}} \tag{17}$$

$$I_k^o = w_{1k} \cdot O_1^H + w_{2k} \cdot O_2^H + \cdots + w_{jk} \cdot O_j^H + \cdots w_{mk} \cdot O_m^H \tag{18}$$

$$O_j^H = \frac{1}{1 + e^{-\theta_H I_j^H}} \tag{19}$$

Calculation of $\bar{v}_{ij}$

...continuation from previous page ...

$$I_j^H = v_{ij} \cdot O_1^H + v_{2j} \cdot O_2^H + \cdots + v_{ij} \cdot O_j^I + \cdots v_{ij} \cdot O_l^I \tag{20}$$

Thus

$$\frac{\partial e_k}{\partial O_{O_k^o}} = -(T_{O_k^o} - O_{O_k^o}) \tag{21}$$

$$\frac{\partial O_k^o}{\partial I_k^o} = \theta_o(1 + O_{O_k^o})(1 - O_{O_k^o}) \tag{22}$$

$$\frac{\partial I_k^o}{\partial O_j^H} = w_{ik} \tag{23}$$

$$\frac{\partial O_j^H}{\partial I_j^H} = \theta_H \cdot (1 - O_j^H) \cdot O_j^H \tag{24}$$

$$\frac{\partial I_j^H}{\partial v_j^i} = O_i^I ={}_i^I I \tag{25}$$

Calculation of $\bar{v}_{ij}$

From the above equations, we get

$$\frac{\partial e_k}{\partial v_{ij}} = -\theta_o \cdot \theta_H (T_{O_k^o} - O_{O_k^o}) \cdot (1 - O_{O_k^o}^2) \cdot O_j^H \cdot I_i^H \cdot w_{jk} \tag{26}$$

Substituting the value of $\frac{\partial e_k}{\partial v_{ij}}$ using Eq. (4), we have

$$\Delta v_{ij} = \eta \cdot \theta_o \cdot \theta_H (T_{O_k^o} - O_{O_k^o}) \cdot (1 - O_{O_k^o}^2) \cdot O_j^H \cdot I_i^H \cdot w_{jk} \tag{27}$$

Therefore, the updated value of v_{ij} can be obtained using Eq.(4)

$$\bar{v}_{ij} = v_{ij} + \eta \cdot \theta_o \cdot \theta_H (T_{O_k^o} - O_{O_k^o}) \cdot (1 - O_{O_k^o}^2) \cdot O_j^H \cdot I_i^H \cdot w_{jk} \tag{28}$$

Writing in matrix form for the calculation of $\bar{V}$ and $\bar{W}$

we have

$$\Delta w_{jk} = \eta \left| \theta_o \cdot (T_{O_k^o} - O_{O_k^o}) \cdot (1 - O_{O_k^o}^2) \right| \cdot O_j^H \quad (29)$$

is the update for k-th neuron receiving signal from j-th neuron at hidden layer.

$$\Delta v_{ij} = \eta \cdot \theta_o \cdot \theta_H (T_{O_k^o} - O_{O_k^o}) \cdot (1 - O_{O_k^o}^2) \cdot (1 - O_j^H) \cdot O_j^H \cdot I_i^I \cdot w_{jk} \quad (30)$$

is the update for j-th neuron at the hidden layer for the i-th input at the i-th neuron at input level.

Calculation of $\bar{W}$

Hence,

$$[\Delta W]_{m\times n} = \eta \cdot \left[O^H\right]_{m\times 1} \cdot [N]_{1\times n} \tag{31}$$

where

$$[N]_{1\times n} = \left\{\theta_o(T_{O_k^o} - O_{O_k^o}) \cdot (1 - O^2_{O_k^o})\right\} \tag{32}$$

where $k = 1, 2, \cdots n$

Thus, the updated weight matrix for a sample input can be written as

$$\left[\bar{W}\right]_{m\times n} = [W]_{m\times n} + [\Delta W]_{m\times n} \tag{33}$$

Calculation of $\bar{V}$

Similarly, for $[\bar{V}]$ matrix, we can write

$$\Delta v_{ij} = \eta \cdot \left| \theta_o (T_{O_k^o} - O_{O_k^o}) \cdot (1 - O_{O_k^o}^2) \cdot w_{jk} \right| \cdot \left| \theta_H (1 - O_j^H) \cdot O_j^H \right| \cdot \left| I_i^I \right| \quad (34)$$

$$= \eta \cdot w_j \cdot \theta^H \cdot (1 - O_j^H) \cdot O_j^H \quad (35)$$

Thus,

$$\Delta V = \left[I^I \right]_{l \times 1} \times \left[M^T \right]_{1 \times m} \quad (36)$$

or

$$\left[\bar{V} \right]_{l \times m} = [V]_{l \times m} + \left[I^I \right]_{l \times 1} \times \left[M^T \right]_{1 \times m} \quad (37)$$

This calculation of Eq. (32) and (36) for one training data $t \in < T_O, T_I >$. We can apply it in incremental mode (i.e. one sample after another) and after each training data, we update the networks V and W matrix.

Batch mode of training

A batch mode of training is generally implemented through the minimization of mean square error (MSE) in error calculation. The MSE for k-th neuron at output level is given by

$$\bar{E} = \frac{1}{2} \cdot \frac{1}{|T|} \sum_{t=1}^{|T|} \left({T^t}_{O_k^o} - {O^t}_{O_k^o} \right)^2$$

where $|T|$ denotes the total number of training scenariso and t denotes a training scenario, i.e. $t \in < T_O, T_I >$

In this case, Δw_{jk} and Δv_{ij} can be calculated as follows

$$\Delta w_{jk} = \frac{1}{|T|} \sum_{\forall t \in T} \frac{\partial \bar{E}}{\partial W}$$

and

$$\Delta v_{ij} = \frac{1}{|T|} \sum_{\forall t \in T} \frac{\partial \bar{E}}{\partial V}$$

Once Δw_{jk} and Δv_{ij} are calculated, we will be able to obtain $\bar{w}_{jk}$ and $\bar{v}_{ij}$

Any questions??