Artificial Neural Network : Training

Credits:

Soft Computing Applications, Dr. Debasis Samanta, IIT Kharagpur

Soft Computing: Fundamentals and Applications, Dr. D. K. Pratihar, IIT Kharagpur

"Principles of Soft Computing" by S.N. Sivanandam & SN Deepa

Neural Networks, Fuzzy Logic and Genetic Algorithms by S. Rajasekaran and GAV Pai

Neural networks- a comprehensive foundation by Simon S Haykin

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.

Kinds of Learning

Parameter Learning: It involves changing and updating the connecting weights in the Neural Net.

Structure Learning: It focuses on changing the structure or architecture of the Neural Net.

Types of learning

The learning methods in neural networks are classified into three basic types :

- Supervised Learning,
- Unsupervised Learning and
- Reinforced Learning

These three types are classified based on :

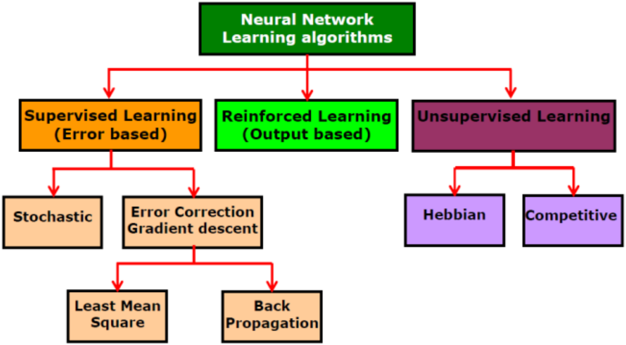
- presence or absence of **teacher** and
- the information provided for the system to learn.

These are further categorized, based on the **rules** used, as

- Hebbian,
- Gradient descent,
- Competitive and
- Stochastic learning.

Types of learning

- There are several learning techniques.
- A taxonomy of well known learning techniques are shown below.



Different learning techniques: Supervised learning

- **Supervised learning**

  In this learning, A teacher is present during the learning process and presents expected output.

- 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.
- The "error" generated is used to change network parameters that result improved performance.
- This type of training is called learning with the help of teacher.

Different learning techniques: Unsupervised learning

- **Unsupervised learning:** No teacher is present

  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 uses 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 : 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 compete and the winning neuron undergoes weight adjustment.
- This is why it is called a Winner-takes-all strategy.

Next, we will discuss a generalized approach of supervised learning to train different type of neural network architectures.

Classification of NN Systems

- ADALINE (Adaptive Linear Neural Element)
- ART (Adaptive Resonance Theory)
- AM (Associative Memory)
- BAM (Bidirectional Associative Memory)
- Boltzmann machines
- BSB (Brain-State-in-a-Box)
- Cauchy machines
- Hopfield Network
- LVQ (Learning Vector Quantization)
- Neoconition
- Perceptron
- RBF (Radial Basis Function)
- RNN (Recurrent Neural Network)
- SOFM (Self-organizing Feature Map)

		Learning Methods			
		Gradient descent	Hebbian	Competitive	Stochastic
Types of Architecture	Single-layer feed-forward	ADALINE, Hopfield, Percepton,	AM, Hopfield,	LVQ, SOFM	-
	Multi-layer feed- forward	CCM, MLFF, RBF	Neocognition		
	Recurrent Networks	RNN	BAM, BSB, Hopfield,	ART	Boltzmann and Cauchy machines

Table : Classification of Neural Network Systems with respect to learning methods and Architecture types

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 $(m + 1)$ inputs $x_0, x_1, x_2, \cdots, x_m$ 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. 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

Algorithm 1 Perceptron Learning

```
w = [w0, w1, w2, . . . , wm]
x = [1, x1, x2, . . . , xm]
P ← input with labels 1;
N ← input with labels 0;
Initialize w randomly;
while !convergence do
     Pick random x ∈ P ∪ N
     if x ∈ P and wᵀx < 0 then
          w = w + x
     if x ∈ N and wᵀx ≥ 0 then
          w = w − x
end
```

Single layer feed forward NN training

Algorithm 2 Perceptron Convergence Algorithm

Variables and Parameters:

$$
\begin{aligned}
\mathbf{x}(n) &= (m+1)\text{-by-1 input vector} \\
&= [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T \\
\mathbf{w}(n) &= (m+1)\text{-by-1 weight vector} \\
&= [b, w_1(n), w_2(n), \ldots, w_m(n)]^T \\
b &= \text{bias} \\
y(n) &= \text{actual response (quantized)} \\
d(n) &= \text{desired response} \\
\eta &= \text{learning-rate parameter, a positive constant less than unity}
\end{aligned}
$$

1. *Initialization.* Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time-step $n = 1, 2, \ldots$.
2. *Activation.* At time-step n, activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.
3. *Computation of Actual Response.* Compute the actual response of the perceptron as

$$y(n) = \operatorname{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$

   where $\operatorname{sgn}(\cdot)$ is the signum function.
4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron to obtain

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$

   where

$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases}$$

5. *Continuation.* Increment time step n by one and go back to step 2.

Single layer feed forward NN training

Note :

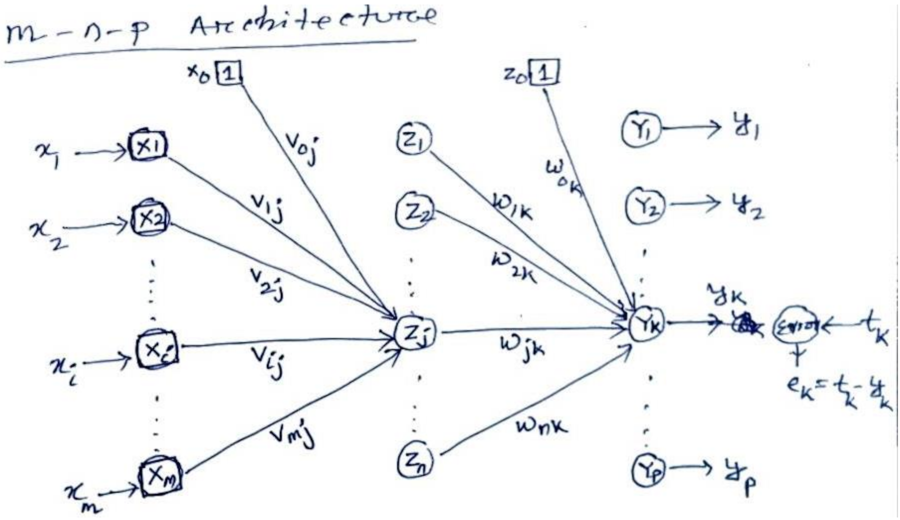
- The algorithm is based on the supervised learning technique
- ADALINE : Adaptive Linear Network Element is also an alternative neuron to perceptron
- If there are 10 number of neurons in the single layer feed forward neural network to be trained, then we have to iterate the algorithm for each perceptron 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 $m - n - p$ is shown in the next slide.

Back-propagation Network



Learning a MLFFNN

Whole learning method consists of the following three computations:

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

Input- Output data

Specifying a MLFFNN

Specifying a MLFFNN

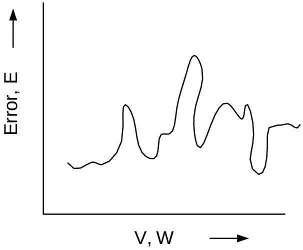
Back Propagation Algorithm

- The above discussion comprises how to calculate values of different parameters in $m - n - p$ multiple layer feed forward neural network.
- Next, we will discuss how to train such a neural network.
- We consider the most popular algorithm calledBack-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.

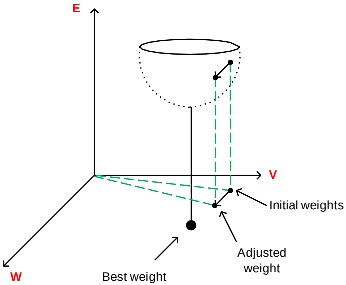
Method of Steepest Descent

- 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 T, the error surface, E can be represented as

$$E = \sum_{t=1}^{T} e^t(V, W, I_t)$$

  where I_t is the t-th input pattern in the training set and $e^t\ldots)$ denotes the error computation of the t-th input.
- Now, we will discuss the steepest descent method of computing error, given a changes in V and W matrices.

Calculation of error in a neural network

- Let us consider any k-th neuron at the output layer. For an input pattern $I_t \in T$ (input in training), the target output t_k of the k-th neuron, computed output be y_k.
- Then, the error e_k of the k-th neuron is defined corresponding to the input I_t as

$$E = \frac{1}{2}(t_k - y_k)^2$$

where y_k denotes the observed output of the k-th neuron.

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

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

- 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 output neuron.

Supervised learning: Back-propagation algorithm

$$E_k = \frac{1}{2}(t_k - y_k)^2$$

	Activation Function
Input	Identity(Linear AF)
Hidden	Binary Sigmoid (Log Sigmoid)
Output	Binary Sigmoid (Log Sigmoid)

Calculation of w_{jk}

Weight Adjustment

$$w_{jk}^{new} = w_{jk}^{old} + \Delta w_{jk}$$

where $\Delta w_{jk} = -\eta \left[\frac{\partial E_k}{\partial w_{jk}} \right]$

Learning rate.

$$v_{ij}^{new} = v_{ij}^{old} + \Delta v_{ij}$$

where $\Delta v_{ij} = -\eta \left[\frac{\partial E_k}{\partial v_{ij}} \right]$

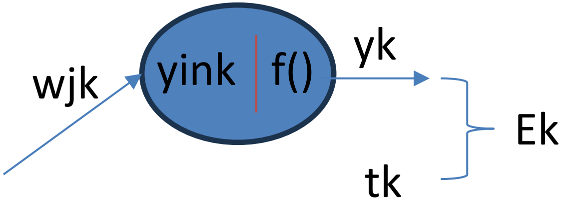
Calculation of w_{jk}

$$E_k = \frac{1}{2}(t_k - y_k)^2$$

Calculation of $\frac{\partial E_k}{\partial w_{jk}}$

Calculation of w_{jk}

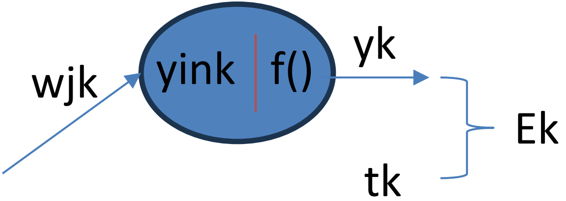
Input Hidden output
i j k



Using Chain-Rule of differentiation

$$\frac{\partial E_k}{\partial w_{jk}} = \frac{\partial E_k}{\partial y_k} \text{ x } \frac{\partial y_k}{\partial y_{ink}} \text{ x } \frac{\partial y_{ink}}{\partial w_{jk}}$$

Calculation of w_{jk}

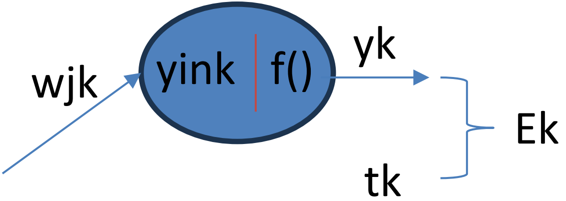


$$\frac{\partial E_k}{\partial w_{jk}} = \frac{\partial E_k}{\partial y_k} \times \frac{\partial y_k}{\partial y_{ink}} \times \frac{\partial y_{ink}}{\partial w_{jk}}$$

$$E_k = \frac{1}{2}(t_k - y_k)^2$$

$$\frac{\partial E_k}{\partial y_k} = -(t_k - y_k)$$

Calculation of w_{jk}



$$\frac{\partial E_k}{\partial w_{jk}} = \frac{\partial E_k}{\partial y_k} \times \frac{\partial y_k}{\partial y_{ink}} \times \frac{\partial y_{ink}}{\partial w_{jk}}$$

$$E_k = \frac{1}{2}(t_k - y_k)^2$$

$$\frac{\partial E_k}{\partial y_k} = -(t_k - y_k)$$

Calculation of w_{jk}

$$\frac{\partial E_k}{\partial w_{jk}} = \frac{\partial E_k}{\partial y_k} \times \frac{\partial y_k}{\partial y_{ink}} \times \frac{\partial y_{ink}}{\partial w_{jk}}$$

In the Output Layer: activation function is chosen Logistic Sigmoid.

$$y_k = f(x) = \frac{1}{1+e^{-\lambda x}} \qquad f'(x) = \lambda f(x)(1 - f(x))$$

$$y_k = f(y_{ink}) = \frac{1}{1+e^{-\lambda y_{ink}}}$$

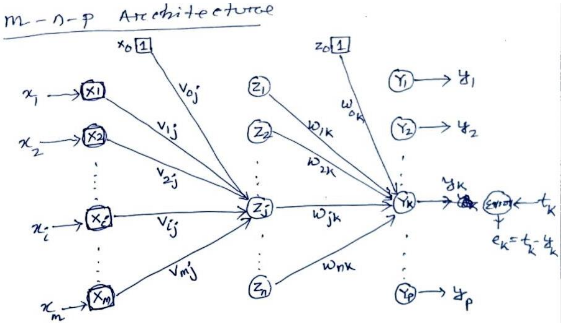
$$\frac{\partial y_k}{\partial y_{ink}} = \lambda y_k(1 - y_k)$$

Calculation of w_{jk}

$$\frac{\partial E_k}{\partial w_{jk}} = \frac{\partial E_k}{\partial y_k} \times \frac{\partial y_k}{\partial y_{ink}} \times \frac{\partial y_{ink}}{\partial w_{jk}}$$

$$y_{ink} = z_1 w_{1k} + z_2 w_{2k} + \ldots + z_j w_{jk} + \cdots + z_n w_{nk}$$

$$\frac{\partial y_{ink}}{\partial w_{jk}} = z_j$$



Putting all the above values together

$$\frac{\partial E_k}{\partial w_{jk}} = \frac{\partial E_k}{\partial y_k} \times \frac{\partial y_k}{\partial y_{ink}} \times \frac{\partial y_{ink}}{\partial w_{jk}}$$

$$-(t_k - y_k)\,\lambda y_k(1 - y_k) z_j$$

$$-\lambda(t_k - y_k)\, y_k(1 - y_k) z_j$$

$$\Delta w_{jk} = -\eta \frac{\partial E_k}{\partial w_{jk}} = \eta \delta_k z_j$$

Let λ = 1

Calculation of w_{jk}

$$\Delta w_{jk} = -\eta \frac{\partial E_k}{\partial w_{jk}} = \eta \delta_k z_j$$

In matrix form:

$$m \overset{V}{-} n \overset{W}{-} p$$

$$\Delta W_{n\,x\,p} = \eta z_{n\,x\,1} \delta_{1\,x\,p}$$

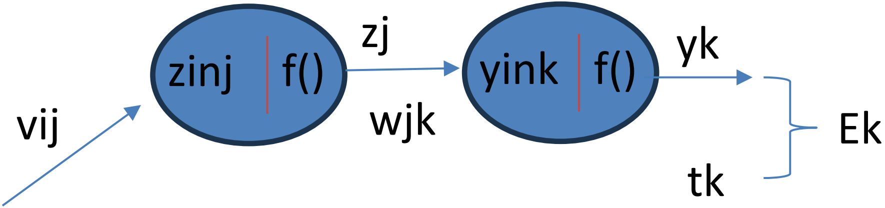
Learning rate

Output of Hidden Layer

Local gradient of output layer

Calculation of v_{ij}

Input Hidden output
i j k



Using Chain-Rule of differentiation

$$\frac{\partial E_k}{\partial v_{ij}} = \frac{\partial E_k}{\partial y_k} \times \frac{\partial y_k}{\partial y_{ink}} \times \frac{\partial y_{ink}}{\partial z_j} \times \frac{\partial z_j}{\partial z_{inj}} \times \frac{\partial z_{inj}}{\partial v_{ij}}$$

Calculation of v_{ij}

Input Hidden output
i j k

$$\frac{\partial E_k}{\partial v_{ij}} = \frac{\partial E_k}{\partial y_k} \times \frac{\partial y_k}{\partial y_{ink}} \times \frac{\partial y_{ink}}{\partial z_j} \times \frac{\partial z_j}{\partial z_{inj}} \times \frac{\partial z_{inj}}{\partial v_{ij}}$$

$$y_{ink} = z_1 w_{1k} + z_2 w_{2k} + \ldots + z_j w_{jk} + \cdots + z_n w_{nk}$$

$$\frac{\partial y_{ink}}{\partial z_j} = w_{jk}$$

Calculation of v_{ij}

Input Hidden output
i j k

$$\frac{\partial E_k}{\partial v_{ij}} = \frac{\partial E_k}{\partial y_k} \times \frac{\partial y_k}{\partial y_{ink}} \times \frac{\partial y_{ink}}{\partial z_j} \times \frac{\partial z_j}{\partial z_{inj}} \times \frac{\partial z_{inj}}{\partial v_{ij}}$$

In the Hidden Layer: activation function is chosen Logistic Sigmoid.

$$z_j = f(x) = \frac{1}{1+e^{-\lambda x}} \qquad f'(x) = \lambda f(x)(1 - f(x))$$

$$z_j = f(z_{inj}) = \frac{1}{1+e^{-\lambda y_{inj}}}$$

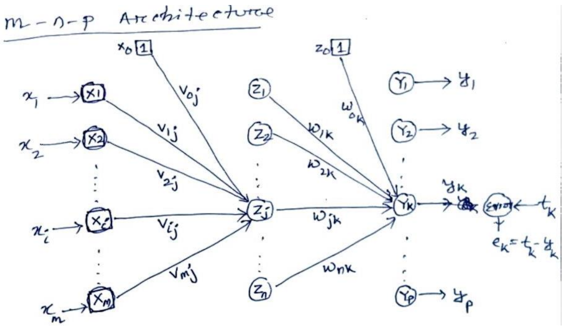
$$\frac{\partial z_j}{\partial z_{inj}} = \lambda z_j(1 - z_j)$$

Calculation of v_{ij}

$$\frac{\partial E_k}{\partial v_{ij}} = \frac{\partial E_k}{\partial y_k} \times \frac{\partial y_k}{\partial y_{ink}} \times \frac{\partial y_{ink}}{\partial z_j} \times \frac{\partial z_j}{\partial z_{inj}} \times \frac{\partial z_{inj}}{\partial v_{ij}}$$

$$z_{inj} = x_1 v_{1j} + x_2 v_{2j} + \ldots + x_i v_{ij} + \cdots + x_m w_{mj}$$

$$\frac{\partial z_{inj}}{\partial v_{ij}} = x_i$$



Calculation of v_{ij}

Input Hidden output
i j k

Putting all the above values together

$$\frac{\partial E_k}{\partial v_{ij}} = \frac{\partial E_k}{\partial y_k} \times \frac{\partial y_k}{\partial y_{ink}} \times \frac{\partial y_{ink}}{\partial z_j} \times \frac{\partial z_j}{\partial z_{inj}} \times \frac{\partial z_{inj}}{\partial v_{ij}}$$

$$\underbrace{-(t_k - y_k)\,\lambda y_k(1-y_k)}_{\delta_k} \qquad w_{jk} \qquad \lambda z_j(1-z_j) \qquad x_i$$

$$\underbrace{\delta_k \qquad w_{jk}}_{\delta_{inj}}$$

$$\underbrace{\delta_{inj} \qquad f'(z_{inj})}_{\delta_j}$$

$$\Delta v_{ij} = -\eta \frac{\partial E_k}{\partial v_{ij}} = \eta \delta_j x_i$$

Let $\lambda = 1$

Calculation of v_{ij}

$$\Delta v_{ij} = -\eta \frac{\partial E_k}{\partial v_{ij}} = \eta \delta_j x_i$$

In matrix form:

$$m \overset{V}{-} n \overset{W}{-} p$$

$$\Delta V_{m\,x\,n} = \eta x_{m\,x\,1} \delta_{1\,x\,n}$$

Learning rate

Input to Input Layer

Local gradient of hidden layer

Example

- **Problem :**

  Consider a typical problem where there are 5 training sets.

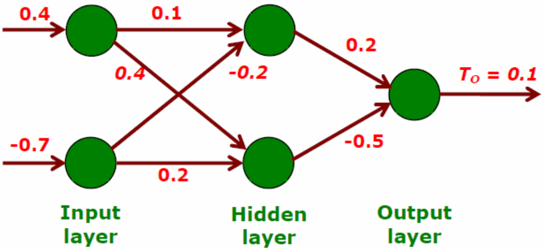
Table : Training sets

S. No.	Input I_1	Input I_2	Output O
1	0.4	-0.7	0.1
2	0.3	-0.5	0.05
3	0.6	0.1	0.3
4	0.2	0.4	0.25
5	0.1	-0.2	0.12

In this problem,

- there are two inputs and one output.
- the values lie between **-1** and **+1** i.e., no need to normalize the values.
- assume two neurons in the hidden layers.

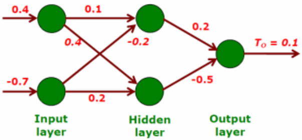
Example



Multi layer feed forward neural network (MFNN) architecture with data of the first training set

Example

Input the First Training Data



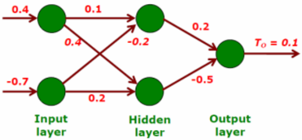
$$[x]^1 = \begin{bmatrix} 0.4 \\ -0.7 \end{bmatrix}_{2\ x\ 1}$$

$$[V]^0 = \begin{bmatrix} v11 & v12 \\ v21 & v22 \end{bmatrix}_{2\ x\ 2} = \begin{bmatrix} 0.1 & 0.4 \\ -0.2 & 0.2 \end{bmatrix}_{2\ x\ 2}$$

$$[W]^0 = \begin{bmatrix} w11 \\ w21 \end{bmatrix}_{2\ x\ 1} = \begin{bmatrix} 0.2 \\ -0.5 \end{bmatrix}_{2\ x\ 1}$$

Example

Hidden Layer Inputs



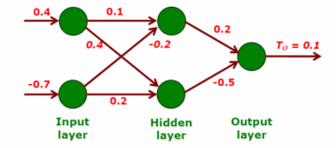
$$[z_{in}]^0 = \begin{bmatrix} z_{in1} \\ z_{in2} \end{bmatrix}_{2\ x\ 1} = V^T\ \mathrm{x} =$$

$$= \begin{bmatrix} v11 & v21 \\ v12 & v22 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \end{bmatrix}$$

$$= \begin{bmatrix} 0.1 & -0.2 \\ 0.4 & 0.2 \end{bmatrix}_{2\ x\ 2} \begin{bmatrix} 0.4 \\ -0.7 \end{bmatrix}_{2\ x\ 1} = \begin{bmatrix} 0.18 \\ 0.02 \end{bmatrix}_{2\ x\ 1}$$

Example

$$[z]^0 = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}_{2\,x\,1}$$



$$= \left\{ \begin{array}{c} \dfrac{1}{(1 + e^{-(0.18)})} \\[2em] \dfrac{1}{(1 + e^{-(0.02)})} \end{array} \right\} = \left\{ \begin{array}{c} 0.5448 \\ 0.505 \end{array} \right\}$$