SUPERVISED LEARNING NEURAL NETWORKS

- A layered network of neurons is required to model functions that are *not* linearly separable functions.
- However, training these layered networks is more complex than training a single neuron.
- Supervised learning requires a training set that consists of input vectors and a target vector associated with each input vector.

- The NN learner uses the target vector to determine how well it has learned, and to guide adjustments to weight values to reduce its overall error.

Neural Network Types

1- Feedforward Neural Networks

- A standard feedforward neural network (FFNN), consisting of *three* layers: an *input* layer, a *hidden* layer (one or more) and an *output* layer.
- FFNN receives external signals and simply *propagate* these signals through all the layers to obtain the result (output) of the NN.
- There are *no* feedback connections to previous layers.

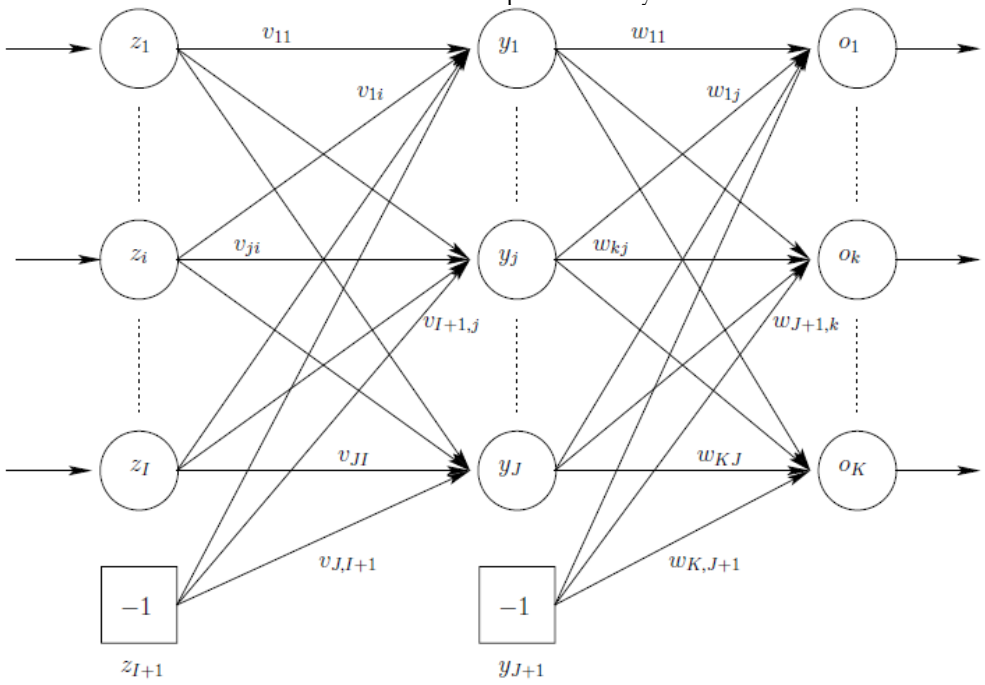


Figure 3.1 Feedforward Neural Network

- The output of a FFNN for any given input pattern z_p is calculated with a single forward pass through the network. For each output unit o_k, we have,

$$
\begin{aligned}
o_{k,p} &= f_{o_k}(net_{o_{k,p}}) \\
&= f_{o_k}\left(\sum_{j=1}^{J+1} w_{kj} f_{y_j}(net_{y_{j,p}})\right) \\
&= f_{o_k}\left(\sum_{j=1}^{J+1} w_{kj} f_{y_j}\left(\sum_{i=1}^{I+1} v_{ji} z_{i,p}\right)\right)
\end{aligned}
$$

where f_{ok} and f_{yj} are respectively the activation function for output unit o_k and hidden unit y_j; w_{kj} is the weight between output unit o_k and hidden unit y_j; $z_{i,p}$ is the value of input unit z_i of input pattern $\mathbf{z}_p$; the $(I + 1)$-th input unit and the $(J + 1)$-th hidden unit are bias units representing the threshold values of neurons in the next layer.

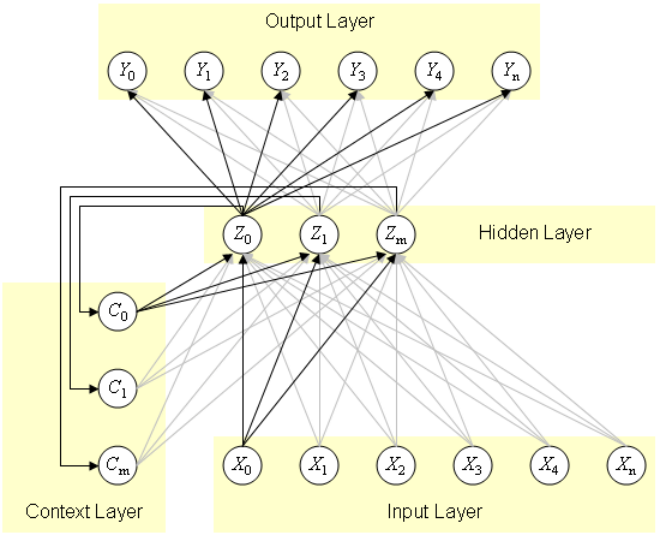
2- Simple Recurrent Neural Networks

Neural networks can be classified into:

- *Static Networks*: these networks have no feedback elements and contain no delays; the output is calculated directly from the input through feedforward connections.
- *Dynamic Networks*: in these networks, the output depends not only on the current input to the network, but also on the current or previous inputs, outputs, or states of the network.

Simple recurrent neural networks (SRNN) have feedback connections which add the ability to also learn the temporal characteristics of the data set.

Elman Network is an example of this type. The Elman SRNN makes a copy of the hidden layer (*the context layer*) to store the previous state of the hidden layer.



3- Time-Delay Neural Networks

- In this network, input patterns successively delayed in time.
- Initially, only $z_{i,p}(t)$, with $t = 0$, has a value and $z_{i,p}(t - t')$ is zero for all $i = 1, \ldots, I$ with time steps $t' = 1, \ldots, nt$; nt is the total number of time steps, or number of delayed patterns.
- Immediately after the first pattern is presented, and before presentation of the second pattern,

$$z_{i,p}(t-1) = z_{i,p}(t)$$

- After presentation of t' patterns and before the presentation of pattern $t' + 1$, for all $t = 1, \ldots, t'$:

$$z_{i,p}(t-t') = z_{i,p}(t-t'+1)$$

- This causes a total of nt patterns to influence the updates of weight values, thus allowing the temporal characteristics to drive the shaping of the learned function.
- Each connection between $z_{i,p}(t - t')$ and $z_{i,p}(t - t' + 1)$ has a value of 1.

- The output of a is calculated as:

$$o_{k,p} = f_{o_k}\left(\sum_{j=1}^{J+1} w_{kj} f_{y_j}\left(\sum_{i=1}^{I}\sum_{t=0}^{n_t} v_{j,i(t)} z_{i,p}(t) + z_{I+1} v_{j,I+1}\right)\right)$$

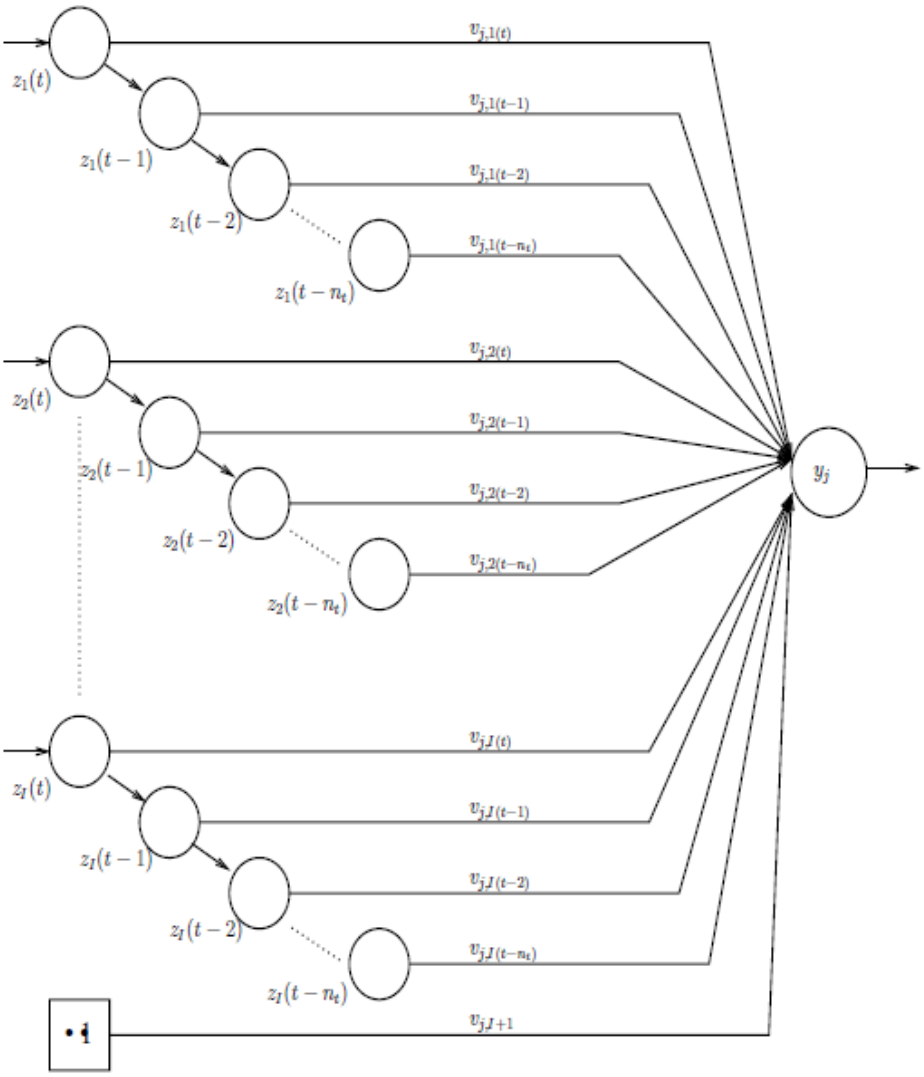


Figure 3.5 A Single Time-Delay Neuron

Supervised Learning Rules

This section explains approaches to train the NN such that the output of the network is an accurate approximation of the target values.

Consider a finite set of input-target pairs $D = \{dp = (\mathbf{z}_p, \mathbf{t}_p) | p = 1, \cdots, P\}$, For NN learning this is achieved by dividing the set D randomly into a training set DT, and a test set DG.

During learning, the function fNN is found which minimizes the empirical error:

$$\mathcal{E}_T(D_T; \mathbf{W}) = \frac{1}{P_T} \sum_{p=1}^{P_T} (F_{NN}(\mathbf{z}_p, \mathbf{W}) - \mathbf{t}_p)^2$$

wherePT is the total number of training patterns.

Optimization algorithms for training NNs are grouped into two classes:

- **Local optimization**, where the algorithm may get stuck in a local optimum without finding a global optimum. Gradient descent is example of local optimizers.
- **Global optimization**, where the algorithm searches for the global optimum by employing mechanisms to search larger parts of the search space. Global optimizer's evolutionary algorithms and swarm optimization.

Learning consists of adjusting weights until an acceptable empirical error has been reached. Two types of supervised learning algorithms exist based on when weights are updated:

- ***Stochastic/online learning***, where weights are adjusted after each pattern presentation. In this case the next input pattern is selected randomly from the training set, to prevent any bias that may occur due to the order in which patterns occur in the training set.
- ***Batch/offline learning***, where weight changes are accumulated and used to adjust weights only after all training patterns have been presented.

Gradient Descent Optimization

- Gradient descent (GD) optimization has led to one of the most popular learning algorithms (***backpropagation***)
- Learning iterations (***epochs***) consists of two phases:
  1- *Feedforward pass*, which simply calculates the output value(s) of the NN for each training pattern.
  2- *Backward propagation*, which propagates an error signal back from the output layer toward the input layer. Weights are adjusted as functions of the backpropagated error signal.
- Assume that the sum squared error (SSE) is used as the objective function. Then, for each pattern, z_p,

$$\mathcal{E}_p = \frac{1}{2} \left(\frac{\sum_{k=1}^{K} (t_{k,p} - o_{k,p})^2}{K} \right) \qquad (3.27)$$

where$K$ is the number of output units, and $t_{k,p}$ and $o_{k,p}$ are respectively the target and actual output values of the k-th output unit. We will assume sigmoid activation functions in the hidden and output layers with augmented vectors. Then:

$$o_k = f_{o_k}(net_{o_k}) = \frac{1}{1+e^{-net_{o_k}}} \tag{3.28}$$

and

$$y_j = f_{y_j}(net_{y_j}) = \frac{1}{1+e^{-net_{y_j}}} \tag{3.29}$$

Weights are updated, in the case of stochastic learning, according to the following equations:

$$w_{kj}(t) \quad += \quad \Delta w_{kj}(t) + \alpha \Delta w_{kj}(t-1) \tag{3.30}$$

$$v_{ji}(t) \quad += \quad \Delta v_{ji}(t) + \alpha \Delta v_{ji}(t-1) \tag{3.31}$$

where α is the momentum.

From (3.28),

$$\frac{\partial o_k}{\partial net_{o_k}} = \frac{\partial f_{o_k}}{\partial net_{o_k}} = (1-o_k)o_k = f'_{o_k} \tag{3.32}$$

and

$$\frac{\partial net_{o_k}}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \left(\sum_{j=1}^{J+1} w_{kj} y_j \right) = y_j \tag{3.33}$$

$$y = \frac{1}{1+e^{-z}}$$

which has a nice derivativ

$$\frac{dy}{dz} = y(1-y)$$

From equations (3.32) and (3.33),

$$\begin{aligned} \frac{\partial o_k}{\partial w_{kj}} &= \frac{\partial o_k}{\partial net_{o_k}} \frac{\partial net_{o_k}}{\partial w_{kj}} \\ &= (1-o_k)o_k y_j \\ &= f'_{o_k} y_j \end{aligned} \tag{3.34}$$

From equation (3.27),

$$\frac{\partial E}{\partial o_k} = \frac{\partial}{\partial o_k} \left(\frac{1}{2} \sum_{k=1}^{K} (t_k - o_k)^2 \right) = -(t_k - o_k) \tag{3.35}$$

Define the output error that needs to be back-propagated as:

$$\delta_{o_k} = \frac{\partial E}{\partial net_{o_k}}$$

Then:

$$\begin{aligned} \delta_{o_k} &= \frac{\partial E}{\partial net_{o_k}} \\ &= \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial net_{o_k}} \\ &= -(t_k - o_k)(1-o_k)o_k = -(t_k - o_k) f'_{o_k} \end{aligned} \tag{3.36}$$

Then, the changes in the hidden-to-output weights are computed from equations (3.35), (3.34) and (3.36) as:

$$\begin{aligned} \Delta w_{kj} &= \eta \left(-\frac{\partial E}{\partial w_{kj}} \right) \\ &= -\eta \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial w_{kj}} \\ &= -\eta \delta_{o_k} y_j \end{aligned} \tag{3.37}$$

Continuing with the input-to-hidden weights,

$$\frac{\partial y_j}{\partial net_{y_i}} = \frac{\partial f_{y_j}}{\partial net_{y_i}} = (1 - y_j)y_j = f'_{y_j} \tag{3.38}$$

and

$$\frac{\partial net_{y_j}}{\partial v_{ji}} = \frac{\partial}{\partial v_{ji}} \left(\sum_{i=1}^{I+1} v_{ji} z_i \right) = z_i \tag{3.39}$$

From equations (3.38) and (3.39),

$$\begin{aligned} \frac{\partial y_j}{\partial v_{ji}} &= \frac{\partial y_j}{\partial net_{y_j}} \frac{\partial net_{y_j}}{\partial v_{ji}} \\ &= (1 - y_j)y_j z_i = f'_{y_j} z_i \end{aligned} \tag{3.40}$$

and

$$\frac{\partial net_{o_k}}{\partial y_j} = \frac{\partial}{\partial y_j} \left(\sum_{j=1}^{J+1} w_{kj} y_j \right) = w_{kj} \tag{3.41}$$

From equations (3.36) and (3.41),

$$\begin{aligned} \frac{\partial E}{\partial y_j} &= \frac{\partial}{\partial y_j} \left(\frac{1}{2} \sum_{k=1}^{K} (t_k - o_k)^2 \right) \\ &= \sum_{k=1}^{K} \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial net_{o_k}} \frac{\partial net_{o_k}}{\partial y_j} \\ &= \sum_{k=1}^{K} \frac{\partial E}{\partial net_{o_k}} \frac{\partial net_{o_k}}{\partial y_j} \\ &= \sum_{k=1}^{K} \delta_{o_k} w_{kj} \end{aligned} \tag{3.42}$$

Define the hidden layer error, which needs to be back-propagated, from equations (3.42) and (3.38) as,

$$\begin{aligned} \delta_{y_j} &= \frac{\partial E}{\partial net_{y_j}} \\ &= \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial net_{y_j}} \\ &= \sum_{k=1}^{K} \delta_{o_k} w_{kj} f'_{y_j} \end{aligned} \tag{3.43}$$

Finally, the changes to input-to-hidden weights are calculated from equations (3.42), (3.40) and (3.43) as:

$$\begin{aligned}\Delta v_{ji} &= \eta\left(-\frac{\partial E}{\partial v_{ji}}\right)\\ &= -\eta\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_{ji}}\\ &= -\eta\delta_{y_j}z_i \qquad (3.44)\end{aligned}$$

If direct weights from the input to the output layer are included, the following additional weight updates are needed:

$$\begin{aligned}\Delta u_{ki} &= \eta\left(-\frac{\partial E}{\partial u_{ki}}\right)\\ &= -\eta\frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial u_{ki}}\\ &= -\eta\delta_{o_k}z_i \qquad (3.45)\end{aligned}$$

where u_{ki} is a weight from the i-th input unit to the k-th output unit.

- In the case of batch learning, weights are updated as given in equations (3.30) and (3.31), but with

$$\Delta w_{kj}(t) = \sum_{p=1}^{P_T}\Delta w_{kj,p}(t) \qquad (3.46)$$

$$\Delta v_{ji}(t) = \sum_{p=1}^{P_T}\Delta v_{ji,p}(t) \qquad (3.47)$$

Stochastic learning is summarized in Algorithm 3.1.

Algorithm 3.1 Stochastic Gradient Descent Learning Algorithm

Initialize weights, η, α, and the number of epochs $t = 0$;
while *stopping condition(s) not true* **do**
  Let $\mathcal{E}_T = 0$;
  for *each training pattern p* **do**
    Do the feedforward phase to calculate $y_{j,p}$ ($\forall\ j = 1, \cdots, J$) and $o_{k,p}$ ($\forall\ k = 1, \cdots, K$);
    Compute output error signals $\delta_{o_{k,p}}$ and hidden layer error signals $\delta_{y_{j,p}}$;
    Adjust weights w_{kj} and v_{ji} (backpropagation of errors);
    $\mathcal{E}_T += [\mathcal{E}_p = \sum_{k=1}^{K}(t_{k,p} - o_{k,p})^2]$;
  end
  $t = t + 1$;
end

Stopping criteria usually includes:

- Stop when a maximum number of epochs have been exceeded.
- Stop when the mean squared error (MSE) on the training set,

$$\mathcal{E}_T = \frac{\sum_{p=1}^{P_T} \sum_{k=1}^{K} (t_{k,p} - o_{k,p})^2}{P_T K}$$

is small enough.

Performance Measures

Performance measures includes: *accuracy, complexity* and *convergence*.

Accuracy

- Generalization is a very important aspect of neural network learning.
- It is a measure of how well the network interpolates to points not used during training.
- The aim of NN learning is therefore to learn the examples presented in the training set well, while still providing good generalization to examples not included in the training set.
- It is, however, possible that a NN exhibits a very low training error, but bad generalization due to ***overfitting*** (memorization) of the training patterns.
- The ***generalization error***, EG, is expressed as:

$$\varepsilon_G = \frac{\sum_{p=1}^{P_G}\sum_{k=1}^{K}(t_{k,p} - o_{k,p})^2}{P_G K}$$

Where $P_G$is the total number of patterns in the test set D_G.

- ***Overfitting*** of a training set means that the NN memorizes the training patterns, and consequently loses the ability to generalize. That is, NNs that overfit cannot predict correct output for data patterns not seen during training.
- If the NN is trained for too long, the excess free parameters start to memorize all the training patterns, and even noise contained in the training set.
- Estimations of generalization error during training can be used to detect the point of overfitting.
- The following figure shows the relationship between training and generalization errors as a function of training epochs. From the start of training, both the training and generalization errors decrease. In the case of oversized NNs, there is a point at which the training error continues to decrease, while the generalization error starts to increase. This is the point of *overfitting*.
- Training should stop as soon an increase in generalization error is observed (***Early Stopping***).

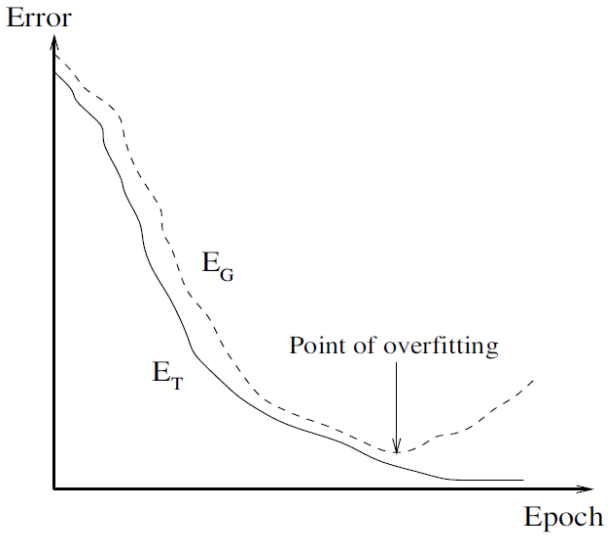


Figure 7.1 Illustration of Overfitting

In order to detect the point of overfitting, the original data set is divided into three disjoint sets, i.e. the *training* set D_T, the *generalization* set $D_G$and the *validation* set D_V. The validation set is then used to estimate the generalization error.

Complexity

The computational complexity of a NN is directly influenced by:

1. *The network architecture*: The larger the architecture, the more feedforward calculations are needed.
2. *The training set size*: The larger the training set size, the total number of learning calculations per epoch is increased.
3. *Complexity of the optimization method*: sophisticated optimization algorithms increase the computational complexity.

Convergence

The convergence characteristics of a NN can be described by the ability of the network to converge to specified error levels.

Weight Initialization

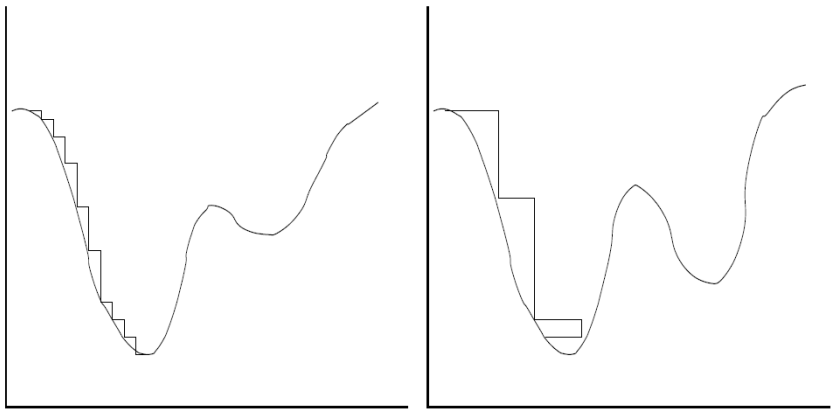
A sensible weight initialization strategy is to choose small random weights centered around 0. This will cause net input signals to be close to zero. Activation functions then output midrange values regardless of the values of input units. Hence, there is no bias toward any solution.

Learning Rate and Momentum

The convergence speed of NNs is directly proportional to the learning rate η. considering stochastic GD, the momentum term added to the weight updates also has the objective of improving convergence time.

Learning Rate

- If the learning rate is too small, the weight adjustments are correspondingly small. More learning iterations are then required to reach a local minimum.
- On the other hand, large η will have large weight updates. Convergence will initially be fast, but the algorithm will eventually oscillate without reaching the minimum.



(a) Small η (b) Large η gets stuck

- But how should the value of the learning rate be selected?

- One approach is to select a small value (e.g. 0.1) and to increase the value if convergence is too slow, or to decrease it if the error does not decrease fast enough.
- Another approach is to gradually reduce a large learning rate to a smaller value. This allows for large initial steps, and ensures small steps in the region of the minimum:

$$\eta(t) = \eta(0)e^{-t/\tau_2}$$

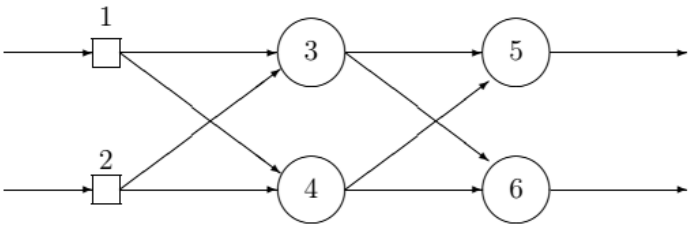
where τ_2 is a positive constant and $\eta(0)$ is the initial, large learning rate.

Momentum

- The idea of the momentum term is to average the weight changes, thereby ensuring that the search path is in the average downhill direction.
- The momentum term is then simply the previous weight change weighted by a scalar value α.
  - If $\alpha = 0$, then the weight changes are not influenced by past weight changes. The larger the value of α, the longer the change in the steepest descent direction has to be persevered in order to affect the direction in which weights are adjusted.
  - A static value of 0.9 is usually used.

Assignment

1- The following diagram represents a feed-forward neural network with onehidden layer:



The followingtable lists all the weights in the network:

$w_{13} = -2$	$w_{35} = 1$
$w_{23} = 3$	$w_{45} = -1$
$w_{14} = 4$	$w_{36} = -1$
$w_{24} = -1$	$w_{46} = 1$

Each of the nodes 3, 4, 5 and 6 uses the following activation function:

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Calculate the outputof the network (y5and y6) for each of the input patterns:

Pattern:	P_1	P_2	P_3	P_4
Node 1:	0	1	0	1
Node 2:	0	0	1	1