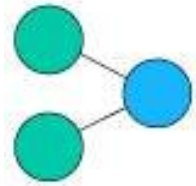


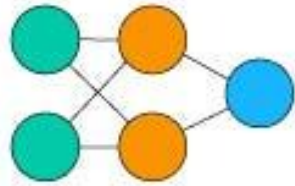
Artificial Neural Networks: Module 1 - Theory Artificial Intelligence

Zool Hilmi Ismail
Tokyo City University

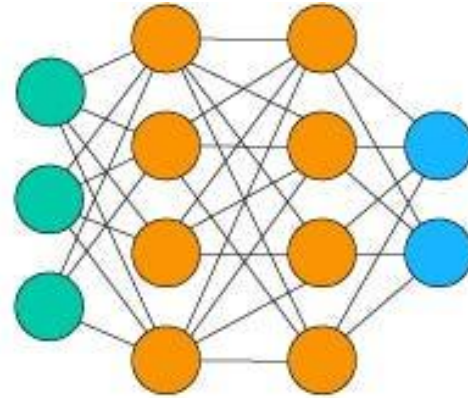
Architecture Types



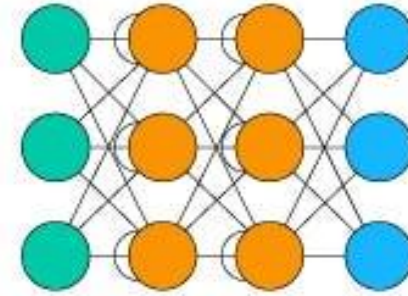
Single Layer Perceptron



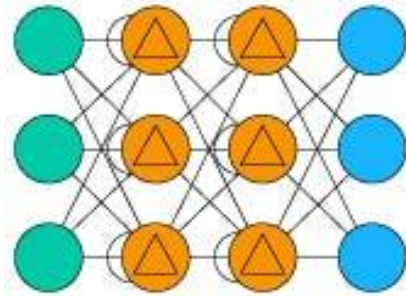
Radial Basis Network (RBN)



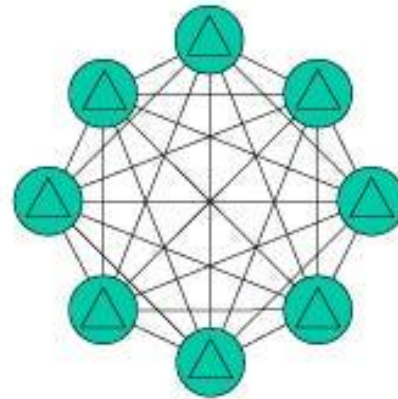
Multi Layer Perceptron



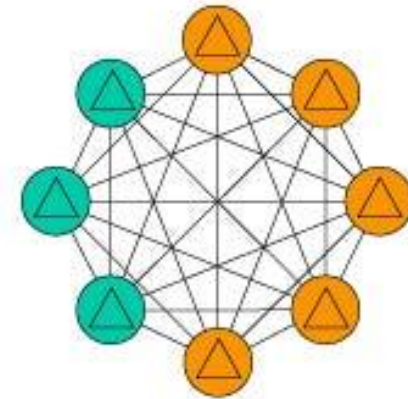
Recurrent Neural Network



LSTM Recurrent Neural Network



Hopfield Network



Boltzmann Machine

● Input Unit

● Hidden Unit

● Backfed Input Unit

● Output Unit

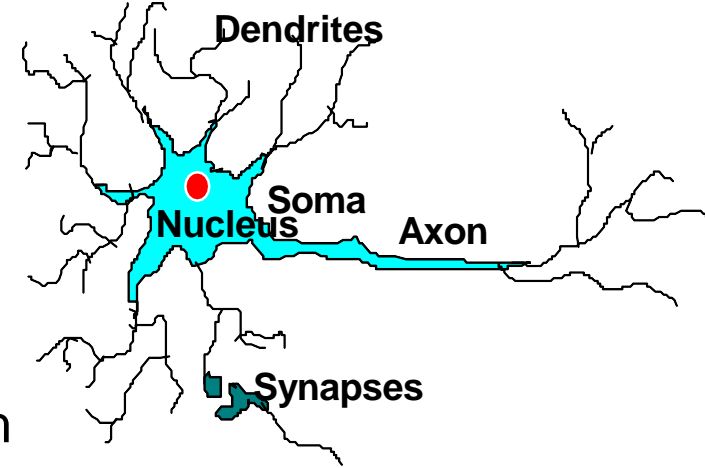
● Feedback with Memory Unit

● Probabilistic Hidden Unit

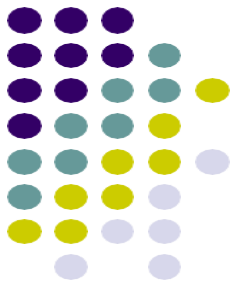
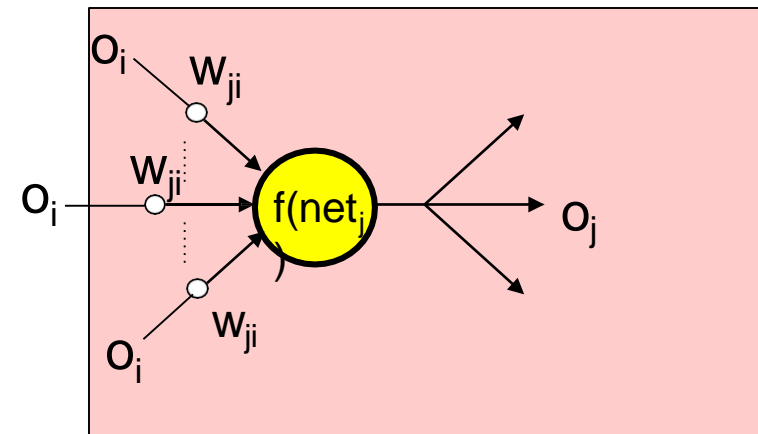
- The biological neuron has lent insights into the design of the artificial neuron.
- The McCulloch-Pitts neuron which is the first artificial neuron was formulated in 1943 (see Module 2.2)

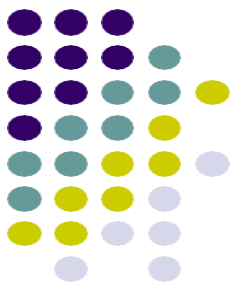
- **Soma**: also called the cell body which supports functions in the neuron.
- **Axon**: part of the neuron which carries the fired signal out of the neuron.
- **Dendrites**: very dense fiber-type of structure that are designed to receive incoming signals.
- **Synapses**: primary gateways where neurons communicate with each other. There is evidence that the chemical reactions at these synapses are altered when learning takes place.

A BIOLOGICAL NEURON



AN ARTIFICIAL NEURON





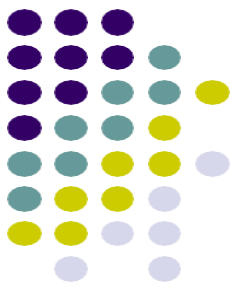
Learning In ANNs

- In all of the neural paradigms, the application of an ANN involves two phases:
 - (1) **Learning phase**
 - (2) **Testing phase**

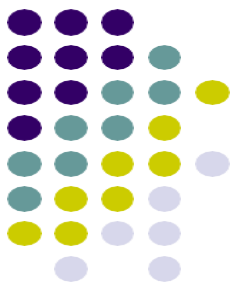
In the **learning** phase (usually offline) the ANN is trained until it has learned its tasks (through the adaptation of its weights) while the **testing** phase is used to solve the task.



Supervised Learning

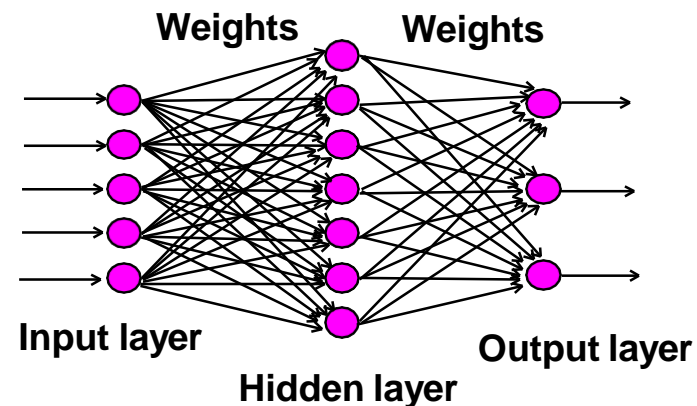


- In supervised learning the training patterns are provided to the ANN together with a teaching signal or target.
- The difference between the ANN output and the target is the error signal.
- Initially the output of the ANN gives a large error during the learning phase.
- The error is then minimized through continuous adaptation of the weights to solve the problem through a learning algorithm.

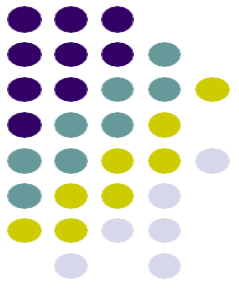


1.3.1 A Set of Processing Units (Neurons)

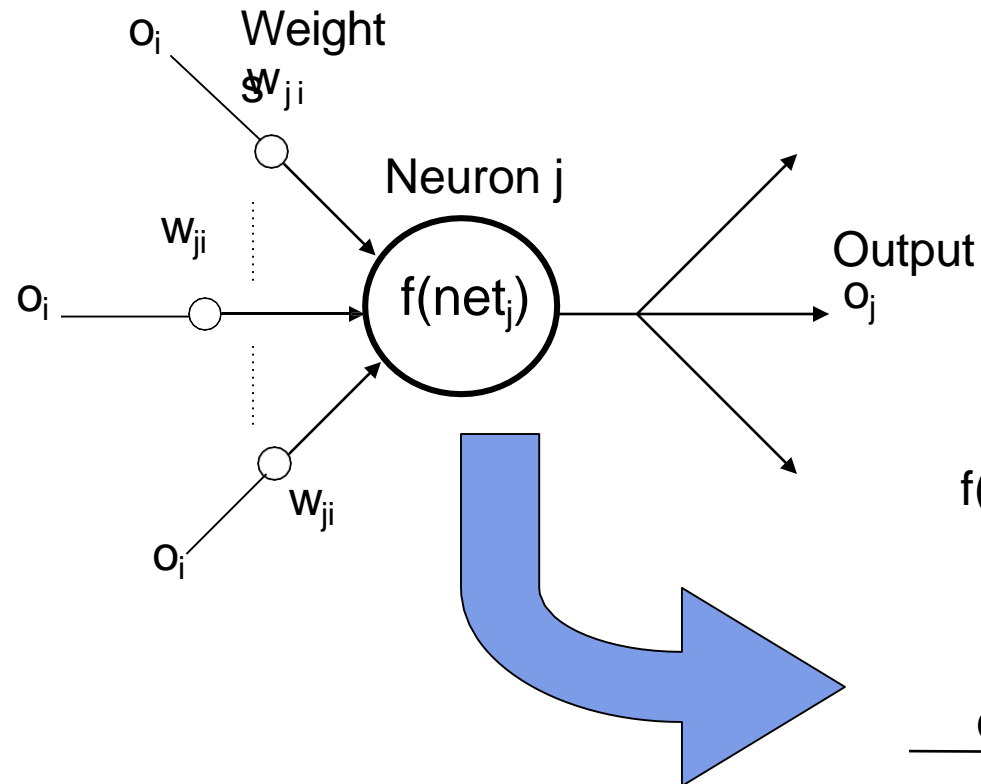
- Also called neurons or nodes.
- Represent elements over which meaningful patterns may be defined
- Need to define the role of each unit.
- Generally, we can characterize 3 types of units:
 - input
 - hidden
 - output
- The units are labeled according to the location of their layers.



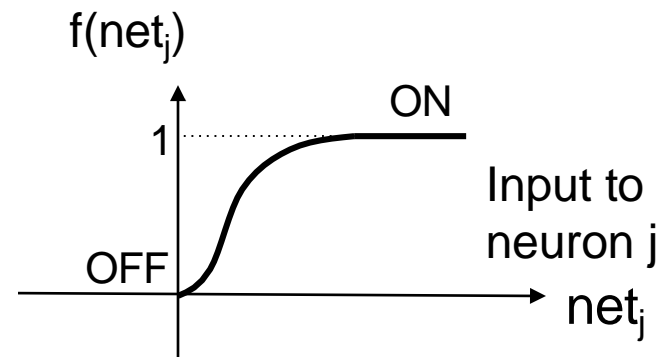
The Artificial Neuron Structure



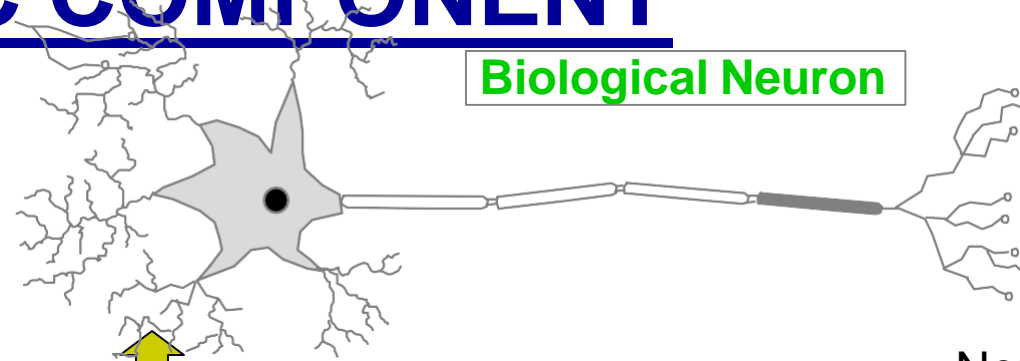
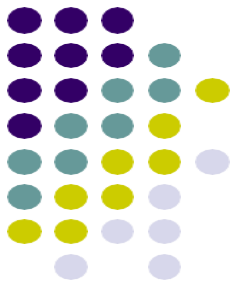
- A typical artificial neuron would have the following structure.
- All the inputs to the neuron would be summed up (net_j) and pass to an activation function to activate the neuron.



Example of an activation function



BASIC COMPONENT

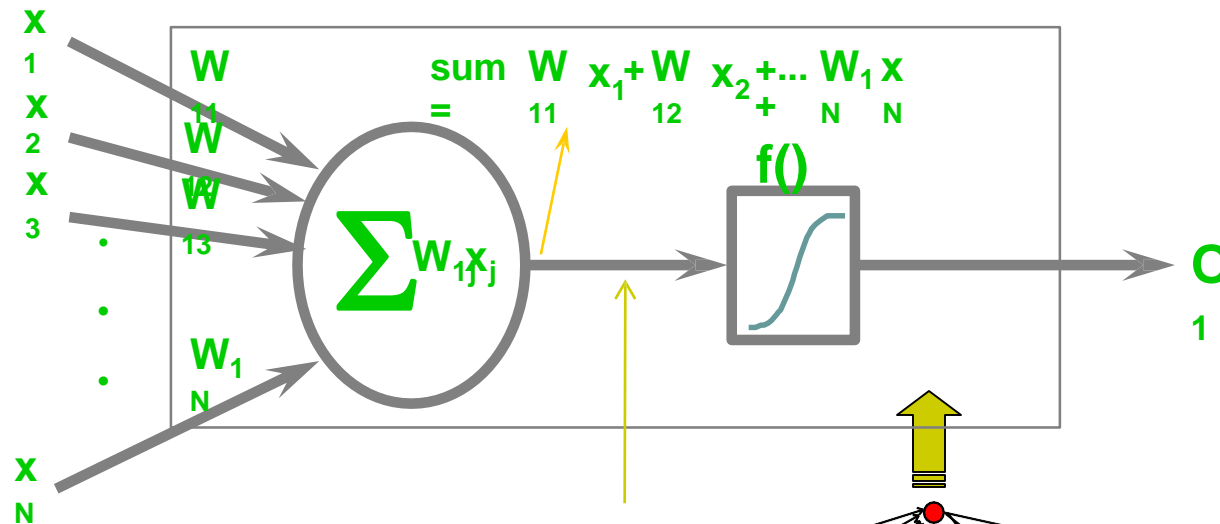


Biological Neuron

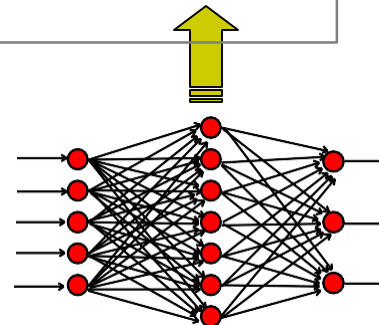
A single network



Artificial Neuron



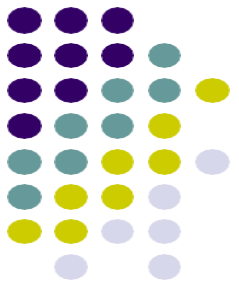
weights



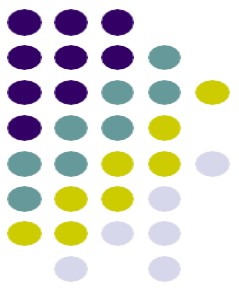
Neural Networks

Neuron or nodes are simple processors whose computing ability is typically restricted to a rule for combining input signals and an activation rule to calculate an output

A single unit combined together to perform complicated tasks

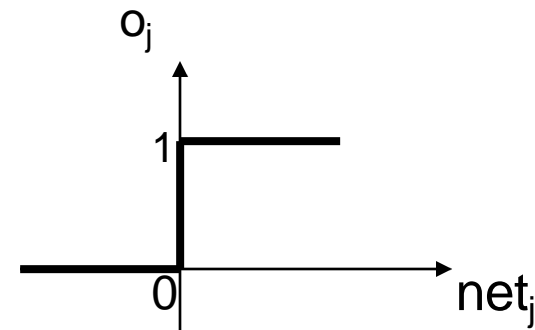
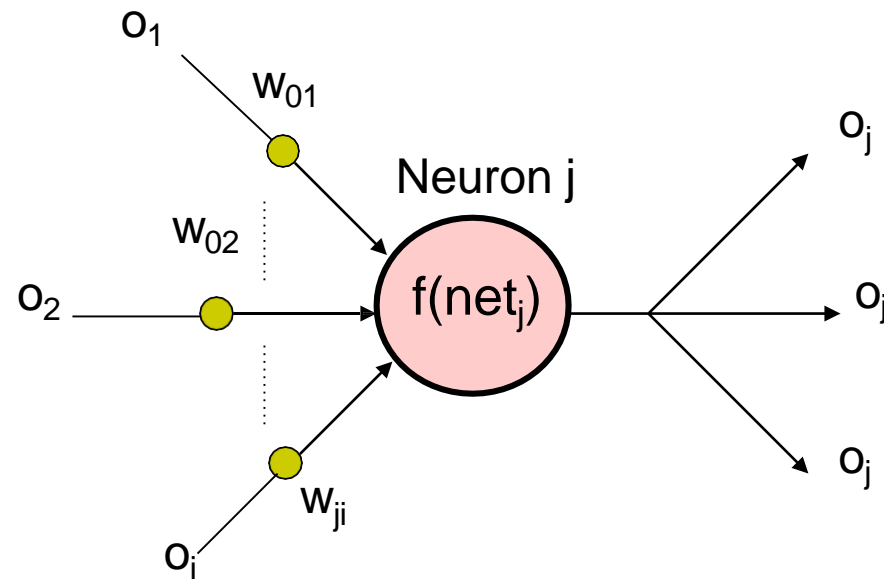


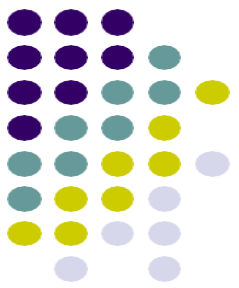
FORWARD: PERCEPTRON



Exercise 1.1:

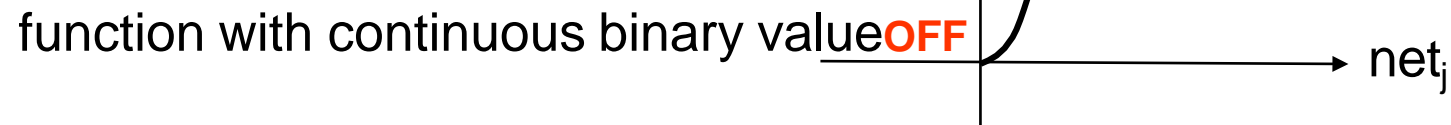
- Write down the mathematical expression for the output (O_j) of the following neuron if $f(\text{net}_j)$ is a threshold function and the inputs to the neuron is net_j .





1.3.2 The State of Activation

- Is the activity of the processing element (neuron).
- Usually the state of activation of neuron can either be ON (fired or 1) or OFF (unfired or 0).
- Is dependent on the application or task to be solved.
- Activation values can have minimum and maximum levels.
- Activation signals can be continuous or discrete.
- Typical examples of activation signals are:
 - binary (0,1)
 - bipolar (-1,1)
- Example of a sigmoid activation





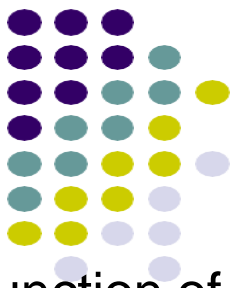
1.3.3 Output of Processing Units

- Usually every neuron or P. U. has an output and the output is a function of the input.
- The degree of activation of units determines the strength of signal transmitted out.
- An Output function can have the following expression:

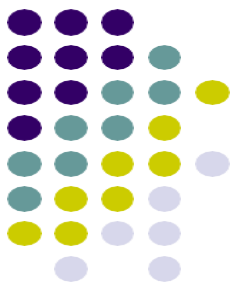
$$o_j = f_j(a_j(t))$$

- Generally, in many models,
output level = activation level

1.3.7 A Learning Rule



- To modify the patterns of connectivity a learning rule is needed as a function of experience.
- There are 3 ways of modifications :
 - development of new connections
 - loss of existing connections
 - modification of the strengths of connections (or weights) that already exist.
- The third way is the standard approach that is used in many neural paradigms - major research area in neural networks



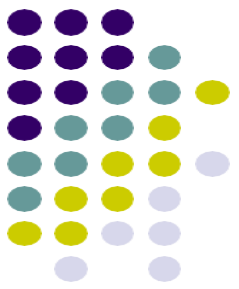
- Basically, the strengths of the connection weights (W) are modified through an adaptation algorithm called learning algorithm or rule.

$$W(new) = W(old) + \Delta W(new)$$

- Thus

- An example of the Hebbian Learning Rule is as follows:

$$\Delta W_{ij}(n) = G[net_j(n), W_{ji}(n-1)]H[net_k(n), W_{kj}(n-1)]$$



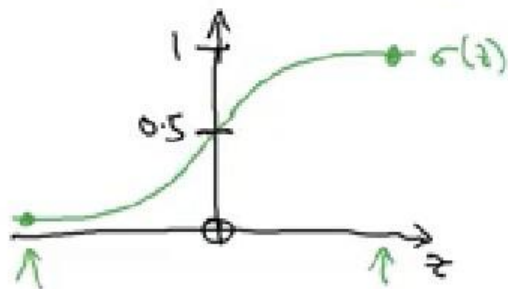
LOSS FUNCTION

Binary Cross Entropy

Given x , want $\hat{y} = \frac{P(y=1|x)}{0 \leq \hat{y} \leq 1}$
 $x \in \mathbb{R}^{n_x}$

Parameters: $\underline{w} \in \mathbb{R}^{n_x}$, $\underline{b} \in \mathbb{R}$.

Output $\hat{y} = \sigma(\underbrace{w^T x + b}_z)$



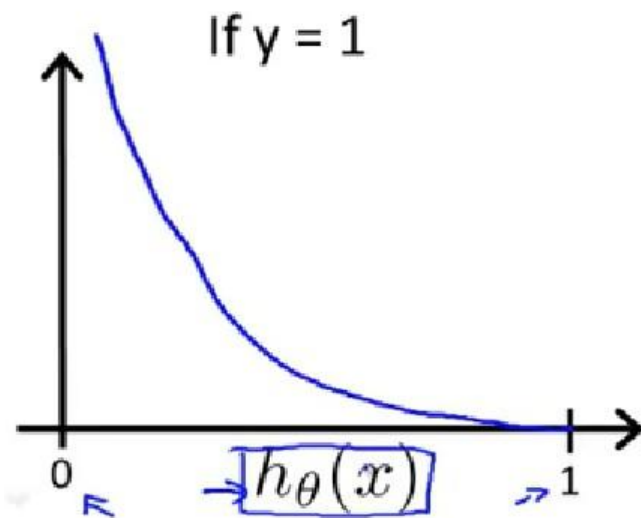
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

If z large $\sigma(z) \approx \frac{1}{1+0} = 1$

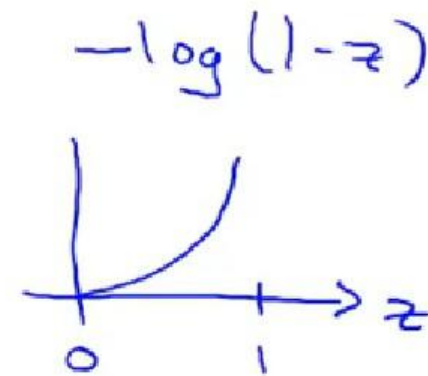
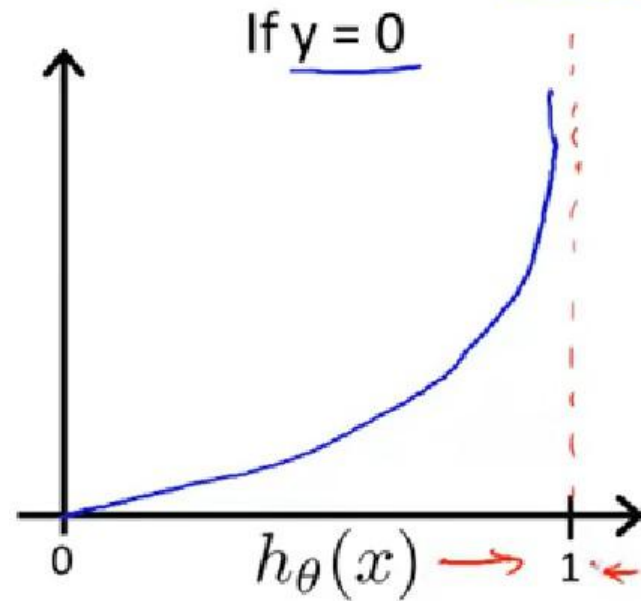
If z large negative number

$$\sigma(z) = \frac{1}{1 + e^{-z}} \approx \frac{1}{1 + \text{Big num}} \approx 0$$

$$\text{Cost}(\underline{h_\theta(x)}, y) = \begin{cases} \boxed{-\log(h_\theta(x))} & \text{if } y = 1 \\ \underline{-\log(1 - h_\theta(x))} & \text{if } y = 0 \end{cases}$$



$$\text{Cost}(h_{\theta}(x), y) = \begin{cases} -\log(h_{\theta}(x)) & \text{if } y = 1 \\ -\log(1 - h_{\theta}(x)) & \text{if } y = 0 \end{cases}$$



$$\rightarrow \hat{y}^{(i)} = \sigma(w^T \underline{x}^{(i)} + b), \text{ where } \sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}} \quad z^{(i)} = w^T x^{(i)} + b$$

Given $\{(\underline{x}^{(1)}, y^{(1)}), \dots, (\underline{x}^{(m)}, y^{(m)})\}$, want $\hat{y}^{(i)} \approx y^{(i)}$.

$x^{(i)}$
 $y^{(i)}$
 $z^{(i)}$ i -th example.

Loss (error) function: $\mathcal{L}(\hat{y}, y) = \frac{1}{2} (\hat{y} - y)^2$

$$\mathcal{L}(\hat{y}, y) = - (y \log \hat{y} + (1-y) \log (1-\hat{y})) \leftarrow$$

If $y=1$: $\mathcal{L}(\hat{y}, y) = -\log \hat{y} \leftarrow$ Want $\log \hat{y}$ large, want

If $y=0$: $\mathcal{L}(\hat{y}, y) = -\log (1-\hat{y}) \leftarrow$ Want $\log (1-\hat{y})$ large ...

$$\text{Cost function: } J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m} \sum_{i=1}^m [y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log (1-\hat{y}^{(i)})]$$

```
>>> -1*math.log(0.1)
2.3025850929940455
>>> -1*math.log(0.2)
1.6094379124341003
>>> -1*math.log(0.3)
1.2039728043259361
>>> -1*math.log(0.4)
0.916290731874155
>>> -1*math.log(0.5)
0.6931471805599453
>>> -1*math.log(0.6)
0.5108256237659907
>>> -1*math.log(0.7)
0.35667494393873245
>>> -1*math.log(0.8)
0.2231435513142097
>>> -1*math.log(0.9)
0.10536051565782628
>>> -1*math.log(1.0)
-0.0
```

Softmax



Softmax

Softmax classifiers give you *probabilities* for each class label

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \quad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$$

$$g^{[L]}(z^{[L]}) = \begin{bmatrix} e^5 / (e^5 + e^2 + e^{-1} + e^3) \\ e^2 / (e^5 + e^2 + e^{-1} + e^3) \\ e^{-1} / (e^5 + e^2 + e^{-1} + e^3) \\ e^3 / (e^5 + e^2 + e^{-1} + e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

- Softmax activation function

Loss Function – One Sample

$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ ← cat $y_2 = 1$
 $y_1 = y_3 = y_4 = 0$

$a^{[4]} = \hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$ ←

$C = 4$

$$\underbrace{\mathcal{L}(\hat{y}, y)}_{\text{small}} = - \sum_{j=1}^C y_j \log \hat{y}_j$$

↑

$$- y_2 \log \hat{y}_2 = \underline{-\log \hat{y}_2}$$

Make \hat{y}_2 big.

Loss Function – All Samples

$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ — cat $y_2 = 1$
 $y_1 = y_3 = y_4 = 0$


$a^{(1)} = \hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$ \ll

$C = 4$

$\mathcal{L}(\hat{y}, y) = - \sum_{j=1}^C y_j \log \hat{y}_j$
small

$\mathcal{J}(w^{(1)}, b^{(1)}, \dots) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

$-y_2 \log \hat{y}_2 = -\log \hat{y}_2$ make \hat{y}_2 big.



A Worked Softmax Example

| | Scoring Function |
|-------|------------------|
| Dog | -3.44 |
| Cat | 1.16 |
| Panda | 3.91 |



Input Image

| | Scoring Function | Unnormalized Probabilities |
|-------|------------------|----------------------------|
| Dog | -3.44 | 0.03 |
| Cat | 1.16 | 3.19 |
| Panda | 3.91 | 49.90 |

| | Scoring Function | Unnormalized Probabilities | Normalized Probabilities |
|-------|------------------|----------------------------|--------------------------|
| Dog | -3.44 | 0.0321 | 0.0006 |
| Cat | 1.16 | 3.1899 | 0.0601 |
| Panda | 3.91 | 49.8990 | 0.9393 |

| | Scoring Function | Unnormalized Probabilities | Normalized Probabilities | Negative Log Loss |
|-------|------------------|----------------------------|--------------------------|-------------------|
| Dog | -3.44 | 0.0321 | 0.0006 | |
| Cat | 1.16 | 3.1899 | 0.0601 | |
| Panda | 3.91 | 49.8990 | 0.9393 | 0.0626 |

MSE

$$MSE = \frac{1}{N} \sum_{i=1}^N (f_i - y_i)^2$$

where N is the number of data points,
 f_i the value returned by the model and
 y_i the actual value for data point i .



MULTI-PERCEPTRON - BACKPROPAGATION



$$w_{n+1} = w_n - \mu \frac{dJ}{dw} \quad (10.8)$$

where $\mu = \text{constant controlling speed of convergence}$.

The illustration of the steepest descent algorithm for solving the optimal coefficient(s) is described in Figure 10.6.

As shown in the first plot in Figure 10.6, if $\frac{dJ}{dw} < 0$, notice that $-\mu \frac{dJ}{dw} > 0$. The new coefficient w_{n+1} will be increased to approach the optimal value w^* by Equation (10.8). On the other hand, if $\frac{dJ}{dw} > 0$, as shown in the second plot in Figure 10.6, we see that $-\mu \frac{dJ}{dw} < 0$. The new coefficient w_{n+1} will be decreased to approach the optimal value w^* . When $\frac{dJ}{dw} = 0$, the best coefficient w_{n+1} is reached.

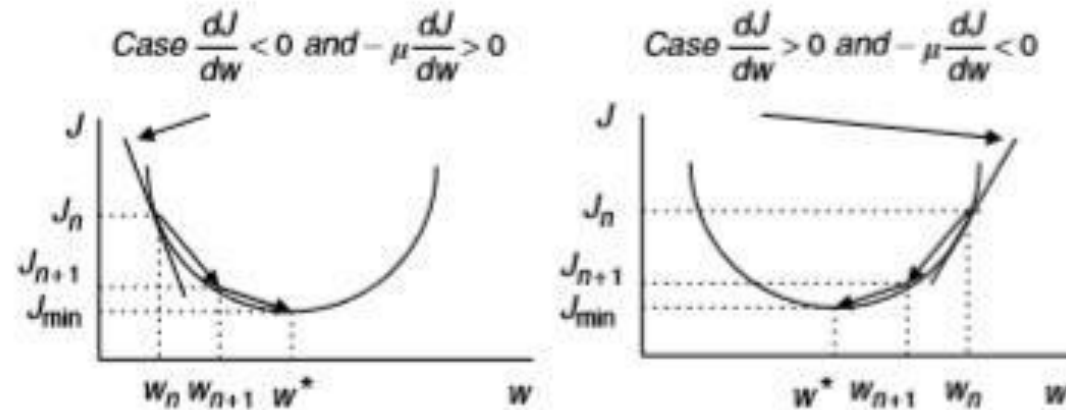
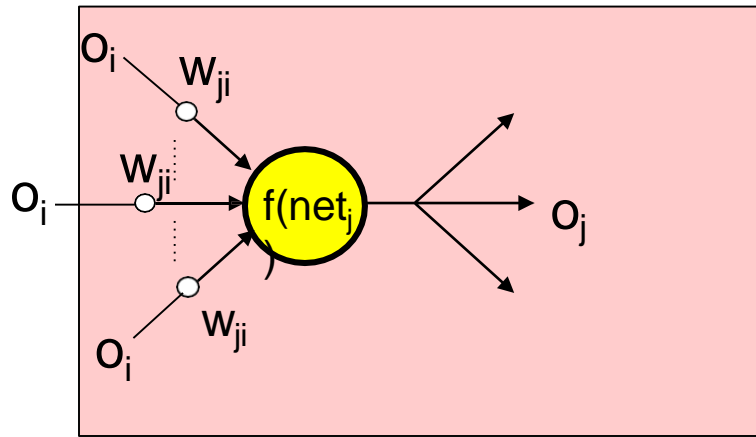


FIGURE 10.6 Illustration of the steepest descent algorithm.

AN ARTIFICIAL NEURON



$$e(p) = Y_d(p) - Y(p) \quad \text{where } p = 1, 2, 3, \dots \quad (6.4)$$

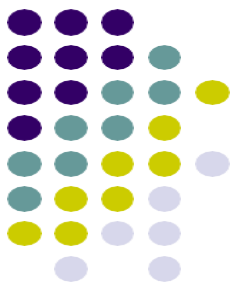
Iteration p here refers to the p th training example presented to the perceptron.

If the error, $e(p)$, is positive, we need to increase perceptron output $Y(p)$, but if it is negative, we need to decrease $Y(p)$. Taking into account that each perceptron input contributes $x_i(p) \times w_i(p)$ to the total input $X(p)$, we find that if input value $x_i(p)$ is positive, an increase in its weight $w_i(p)$ tends to increase perceptron output $Y(p)$, whereas if $x_i(p)$ is negative, an increase in $w_i(p)$ tends to decrease $Y(p)$. Thus, the following **perceptron learning rule** can be established:

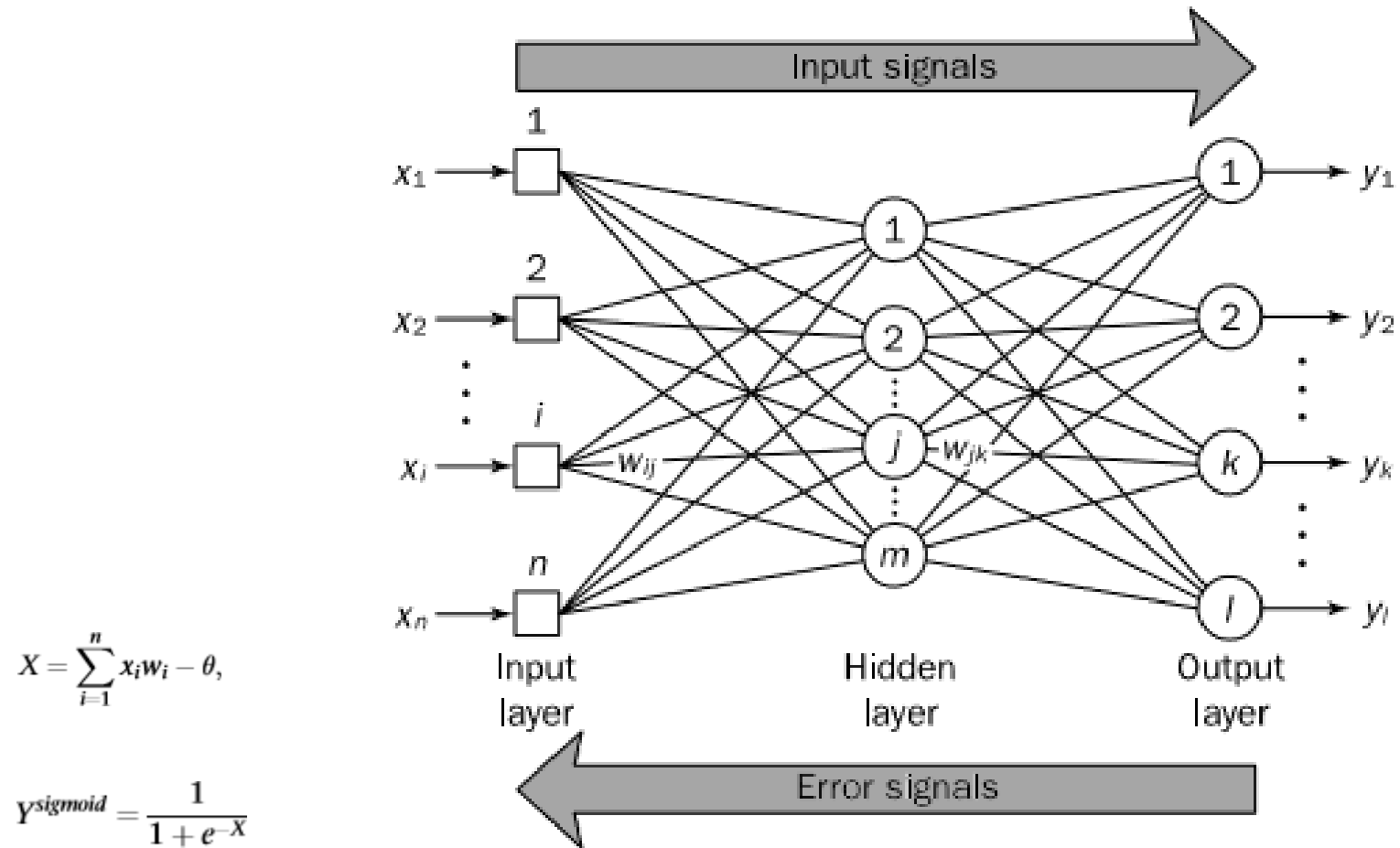
$$w_i(p+1) = w_i(p) + \alpha \times x_i(p) \times e(p), \quad (6.5)$$

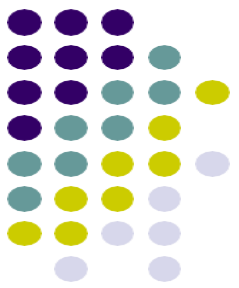
where α is the **learning rate**, a positive constant less than unity.



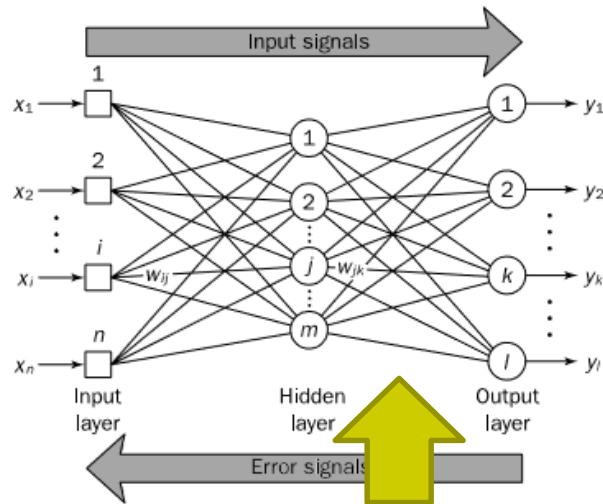


Backpropagation





Backpropagation



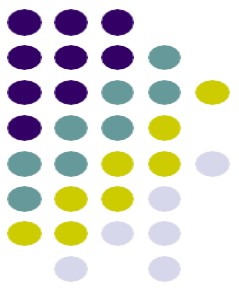
$$e_k(p) = y_{d,k}(p) - y_k(p),$$

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p),$$

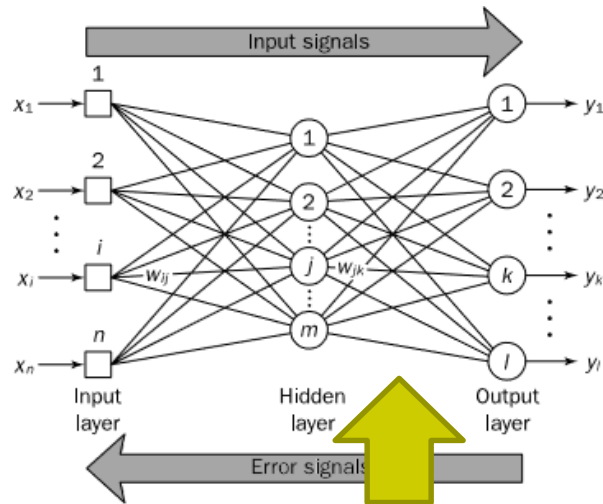
We use the output of neuron j in the hidden layer, y_j , instead of input x_i .

$$\Delta w_{jk}(p) = \alpha \times y_j(p) \times \delta_k(p),$$

where $\delta_k(p)$ is the error gradient at neuron k in the output layer at iteration p .



Backpropagation



Thus, for neuron k in the output layer, we have

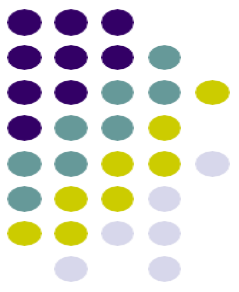
$$\delta_k(p) = \frac{\partial y_k(p)}{\partial X_k(p)} \times e_k(p),$$

For a sigmoid activation function

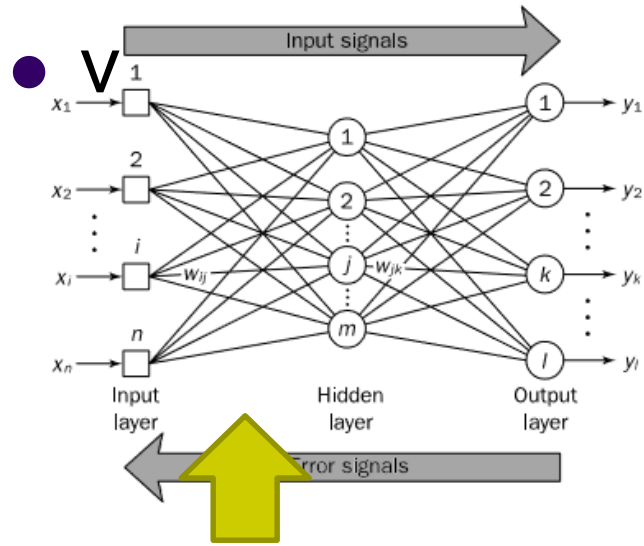
$$\delta_k(p) = \frac{\partial \left\{ \frac{1}{1 + \exp[-X_k(p)]} \right\}}{\partial X_k(p)} \times e_k(p) = \frac{\exp[-X_k(p)]}{\{1 + \exp[-X_k(p)]\}^2} \times e_k(p)$$

$$\delta_k(p) = y_k(p) \times [1 - y_k(p)] \times e_k(p),$$

$$y_k(p) = \frac{1}{1 + \exp[-X_k(p)]}.$$



Backpropagation



$\delta_j(p)$ represents the error gradient at neuron j in the hidden layer:

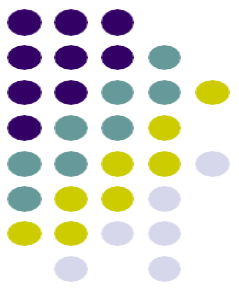
$$\Delta w_{ij}(p) = \alpha \times x_i(p) \times \delta_j(p),$$

$$\delta_j(p) = y_j(p) \times [1 - y_j(p)] \times \sum_{k=1}^l \delta_k(p) w_{jk}(p),$$

where l is the number of neurons in the output layer;

$$y_j(p) = \frac{1}{1 + e^{-X_j(p)}};$$

$$X_j(p) = \sum_{i=1}^n x_i(p) \times w_{ij}(p) - \theta_j;$$



Backpropagation – Step by Step

Step 1: *Initialisation*

Set all the weights and threshold levels of the network to random numbers uniformly distributed inside a small range (Haykin, 1999):

$$\left(-\frac{2.4}{F_i}, +\frac{2.4}{F_i}\right),$$

where F_i is the total number of inputs of neuron i in the network. The weight initialisation is done on a neuron-by-neuron basis.

Step 2: *Activation*

Activate the back-propagation neural network by applying inputs $x_1(p), x_2(p), \dots, x_n(p)$ and desired outputs $y_{d,1}(p), y_{d,2}(p), \dots, y_{d,n}(p)$.

(a) Calculate the actual outputs of the neurons in the hidden layer:

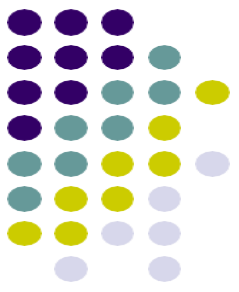
$$y_j(p) = \text{sigmoid} \left[\sum_{i=1}^n x_i(p) \times w_{ij}(p) - \theta_j \right],$$

where n is the number of inputs of neuron j in the hidden layer, and *sigmoid* is the sigmoid activation function.

(b) Calculate the actual outputs of the neurons in the output layer:

$$y_k(p) = \text{sigmoid} \left[\sum_{j=1}^m x_{jk}(p) \times w_{jk}(p) - \theta_k \right],$$

where m is the number of inputs of neuron k in the output layer.



Backpropagation – Step by Step

Step 3: *Weight training*

Update the weights in the back-propagation network propagating backward the errors associated with output neurons.

- (a) Calculate the error gradient for the neurons in the output layer:

$$\delta_k(p) = y_k(p) \times [1 - y_k(p)] \times e_k(p)$$

where

$$e_k(p) = y_{d,k}(p) - y_k(p)$$

Calculate the weight corrections:

$$\Delta w_{jk}(p) = \alpha \times y_j(p) \times \delta_k(p)$$

Update the weights at the output neurons:

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$$

- (b) Calculate the error gradient for the neurons in the hidden layer:

$$\delta_j(p) = y_j(p) \times [1 - y_j(p)] \times \sum_{k=1}^l \delta_k(p) \times w_{jk}(p)$$

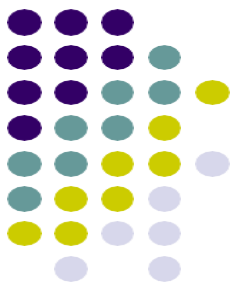
Calculate the weight corrections:

$$\Delta w_{ij}(p) = \alpha \times x_i(p) \times \delta_j(p)$$

Update the weights at the hidden neurons:

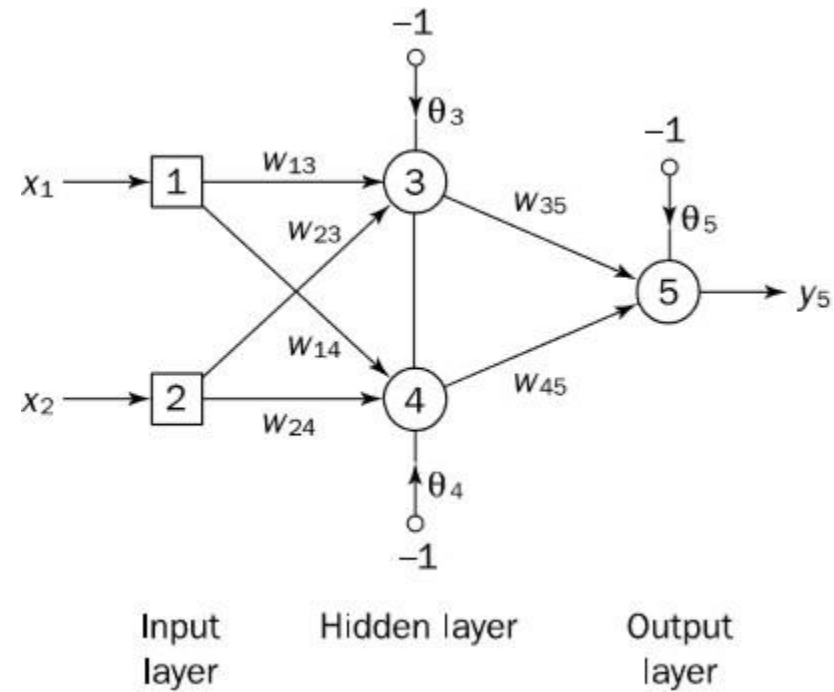
$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

Backpropagation – Step by Step



Step 4: *Iteration*

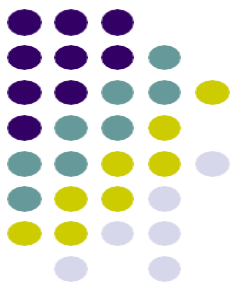
Increase iteration p by one, go back to Step 2 and repeat the process until the selected error criterion is satisfied.



| INPUT | | OUTPUT |
|-------|---|--------|
| A | B | C |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Truth Table

$w_{13} = 0.5$, $w_{14} = 0.9$, $w_{23} = 0.4$, $w_{24} = 1.0$, $w_{35} = -1.2$, $w_{45} = 1.1$,
 $\theta_3 = 0.8$, $\theta_4 = -0.1$ and $\theta_5 = 0.3$.



$$y_3 = \text{sigmoid}(x_1 w_{13} + x_2 w_{23} - \theta_3) = 1/[1 + e^{-(1 \times 0.5 + 1 \times 0.4 - 1 \times 0.8)}] = 0.5250$$

$$y_4 = \text{sigmoid}(x_1 w_{14} + x_2 w_{24} - \theta_4) = 1/[1 + e^{-(1 \times 0.9 + 1 \times 1.0 + 1 \times 0.1)}] = 0.8808$$

$$y_5 = \text{sigmoid}(y_3 w_{35} + y_4 w_{45} - \theta_5) = 1/[1 + e^{-(0.5250 \times 1.2 + 0.8808 \times 1.1 - 1 \times 0.3)}] = 0.5097$$

$$e = y_{d,5} - y_5 = 0 - 0.5097 = -0.5097$$



First, we calculate the error gradient for neuron 5 in the output layer:

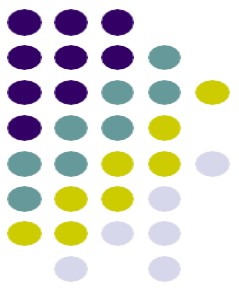
$$\delta_5 = y_5(1 - y_5)e = 0.5097 \times (1 - 0.5097) \times (-0.5097) = -0.1274$$

Then we determine the weight corrections assuming that the learning rate parameter, α , is equal to 0.1:

$$\Delta w_{35} = \alpha \times y_3 \times \delta_5 = 0.1 \times 0.5250 \times (-0.1274) = -0.0067$$

$$\Delta w_{45} = \alpha \times y_4 \times \delta_5 = 0.1 \times 0.8808 \times (-0.1274) = -0.0112$$

$$\Delta \theta_5 = \alpha \times (-1) \times \delta_5 = 0.1 \times (-1) \times (-0.1274) = 0.0127$$



Next we calculate the error gradients for neurons 3 and 4 in the hidden layer:

$$\delta_3 = y_3(1 - y_3) \times \delta_5 \times w_{35} = 0.5250 \times (1 - 0.5250) \times (-0.1274) \times (-1.2) = 0.0381$$

$$\delta_4 = y_4(1 - y_4) \times \delta_5 \times w_{45} = 0.8808 \times (1 - 0.8808) \times (-0.1274) \times 1.1 = -0.0147$$

We then determine the weight corrections:

$$\Delta w_{13} = \alpha \times x_1 \times \delta_3 = 0.1 \times 1 \times 0.0381 = 0.0038$$

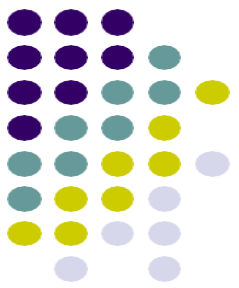
$$\Delta w_{23} = \alpha \times x_2 \times \delta_3 = 0.1 \times 1 \times 0.0381 = 0.0038$$

$$\Delta \theta_3 = \alpha \times (-1) \times \delta_3 = 0.1 \times (-1) \times 0.0381 = -0.0038$$

$$\Delta w_{14} = \alpha \times x_1 \times \delta_4 = 0.1 \times 1 \times (-0.0147) = -0.0015$$

$$\Delta w_{24} = \alpha \times x_2 \times \delta_4 = 0.1 \times 1 \times (-0.0147) = -0.0015$$

$$\Delta \theta_4 = \alpha \times (-1) \times \delta_4 = 0.1 \times (-1) \times (-0.0147) = 0.0015$$



At last, we update all weights and threshold levels in our network:

$$w_{13} = w_{13} + \Delta w_{13} = 0.5 + 0.0038 = 0.5038$$

$$w_{14} = w_{14} + \Delta w_{14} = 0.9 - 0.0015 = 0.8985$$

$$w_{23} = w_{23} + \Delta w_{23} = 0.4 + 0.0038 = 0.4038$$

$$w_{24} = w_{24} + \Delta w_{24} = 1.0 - 0.0015 = 0.9985$$

$$w_{35} = w_{35} + \Delta w_{35} = -1.2 - 0.0067 = -1.2067$$

$$w_{45} = w_{45} + \Delta w_{45} = 1.1 - 0.0112 = 1.0888$$

$$\theta_3 = \theta_3 + \Delta \theta_3 = 0.8 - 0.0038 = 0.7962$$

$$\theta_4 = \theta_4 + \Delta \theta_4 = -0.1 + 0.0015 = -0.0985$$

$$\theta_5 = \theta_5 + \Delta \theta_5 = 0.3 + 0.0127 = 0.3127$$



Consider a single-weight case of $y(n) = wx(n)$, and note that the error signal $e(n)$ is given by

$$e(n) = d(n) - wx(n). \quad (10.2)$$

Now let us solve the best weight w^* . Taking the square of the output error leads to

$$e^2(n) = (d(n) - wx(n))^2 = d^2(n) - 2d(n)wx(n) + w^2x^2(n). \quad (10.3)$$

Taking the statistical expectation of Equation (10.3), we have

$$E(e^2(n)) = E(d^2(n)) - 2wE(d(n)x(n)) + w^2E(x^2(n)). \quad (10.4)$$

Using the notations in statistics, we define

$$J = E(e^2(n)) = \text{MSE (mean squared error)}$$

$$\sigma^2 = E(d^2(n)) = \text{power of corrupted signal}$$

$$P = E(d(n)x(n)) = \text{cross-correlation between } d(n) \text{ and } x(n)$$

$$R = E(x^2(n)) = \text{autocorrelation}$$

We can view the statistical expectation as an average of the N signal terms, each being a product of two individual signal samples:

$$E(e^2(n)) = \frac{e^2(0) + e^2(1) + \dots + e^2(N-1)}{N}$$

or

$$E(d(n)x(n)) = \frac{d(0)x(0) + d(1)x(1) + \dots + d(N-1)x(N-1)}{N}$$

for a sufficiently large sample number of N . We can write Equation (10.4) as

$$J = \sigma^2 - 2wP + w^2R. \quad (10.5)$$

Since σ^2 , P , and R are constants, J is a quadratic function of w that may be plotted in Figure 10.5.

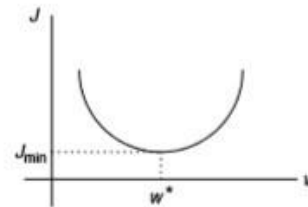


FIGURE 10.5 Mean square error quadratic function.



The best weight (optimal) w^* is at the location where the minimum MSE J_{\min} is achieved. To obtain w^* , taking a derivative of J and setting it to zero leads to

$$\frac{dJ}{dw} = -2P + 2wR = 0. \quad (10.6)$$

Solving Equation 10.6, we get the best weight solution as

$$w^* = R^{-1}P. \quad (10.7)$$

Notice that a few points need to be clarified for Equation (10.7):

1. Optimal coefficient(s) can be different for every block of data, since the corrupted signal and reference signal are unknown. The autocorrelation and cross-correlation may vary.
2. If a larger number of coefficients (weights) are used, the inverse matrix of R^{-1} may require a larger number of computations and may come to be ill-conditioned. This will make real-time implementation impossible.
3. The optimal solution is based on the statistics, assuming that the size of the data block, N , is sufficiently long. This will cause a long processing delay that will make real-time implementation impossible.