Neural Networks and Deep Learning

COURSE: CS40002

Pallab Dasgupta
Professor,
Dept. of Computer Sc & Engg

The ML problem in regression

What is the function $f(.)$?

Solution: ***This is where the different ML methods come in***

- **Linear model: $f(x) = w^T x$**
- **Linear basis functions: $f(x) = w^T \phi(x)$**
  - **Where $\phi(x) = [\phi_0(x)\ \phi_1(x) \ldots \phi_L(x)]^T$ and $\phi_l(x)$ is the basis function.**
  - **Choices for the basis function:**
    - **Powers of x: $\phi_l(x) = x^l$**
    - **Gaussian / Sigmoidal / Fourier / …**
- **Neural networks**
- ***…***

Classification

Given training data set with:

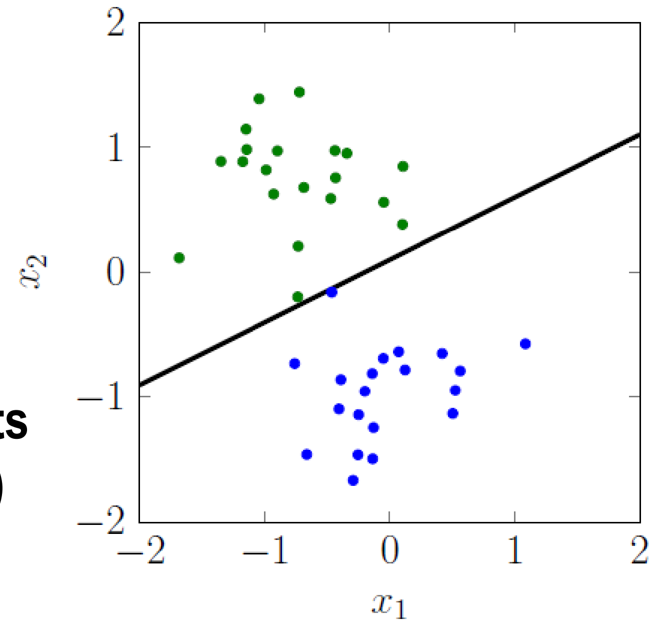
- **Input values:** $\boldsymbol{x_n} = [\boldsymbol{x_1}\ \boldsymbol{x_2}\ \ldots\ \boldsymbol{x_M}]^T$ **for** $\boldsymbol{n} = \mathbf{1} \ldots \boldsymbol{N}$.
- **Output class labels, for example:**
  - **0/1 or –1/+1 for binary classification problems**
  - **1 … K for multi-class classification problems**
  - **1-of-K coding scheme:**

$$\mathbf{y} = [\mathbf{0}\ \ldots \mathbf{0}\ \mathbf{1}\ \mathbf{0}\ \ldots \mathbf{0}]^T$$

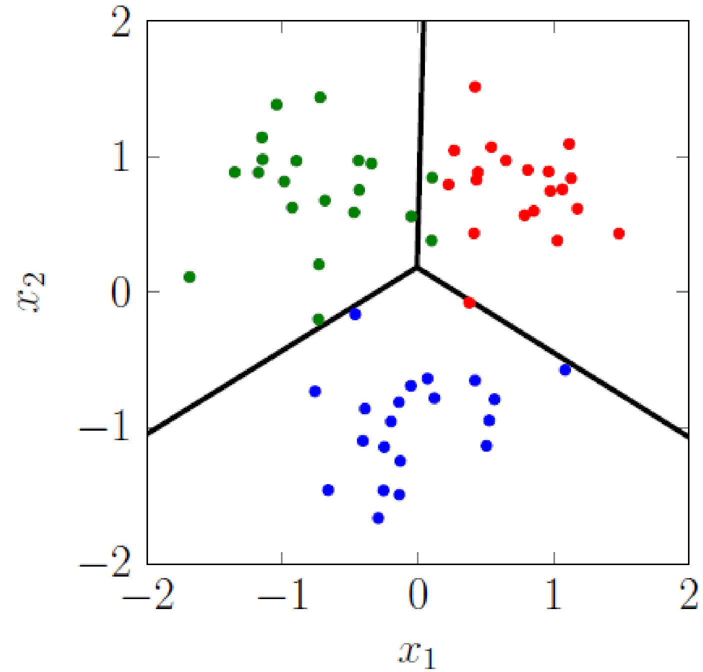
where, if $\boldsymbol{x_n}$ belongs to class *k*, then the *k*th bit is 1 and all others are 0.

Objective: Predict the output class for new, unknown inputs $\boldsymbol{\hat{x}_m}$.

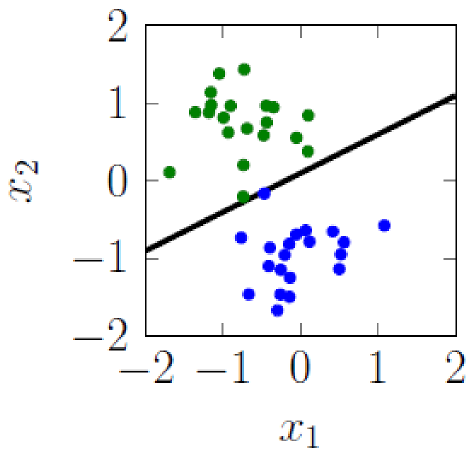
Classification strategies

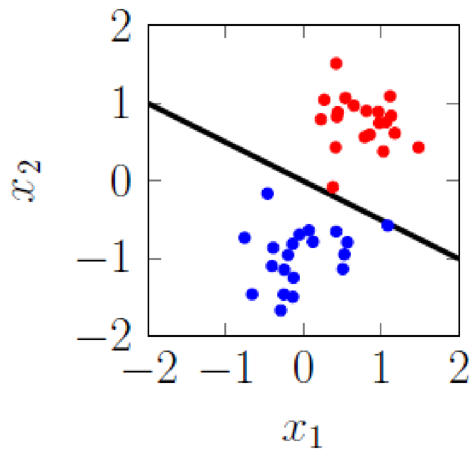


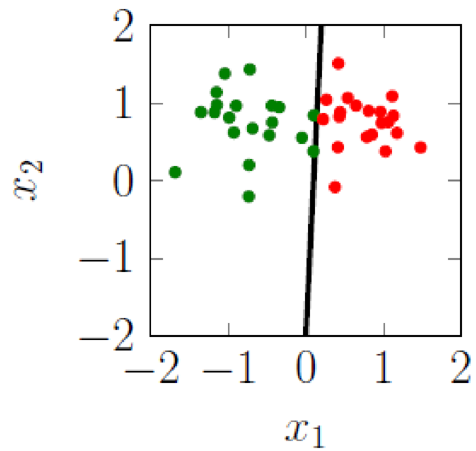
Linear discriminants
(2-class classifiers)

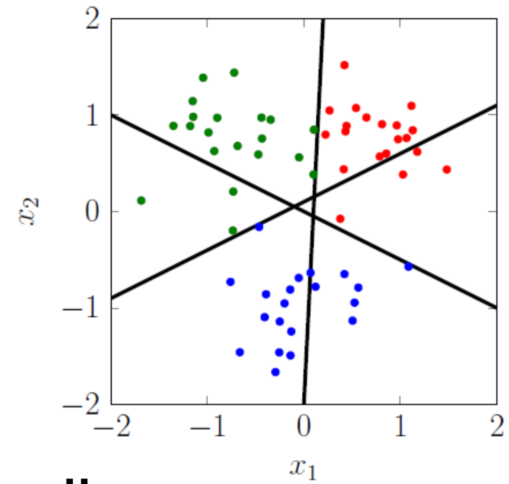


K-class discriminant









Combining 2-class classifiers to obtain multi-class classifiers is a bad idea !!

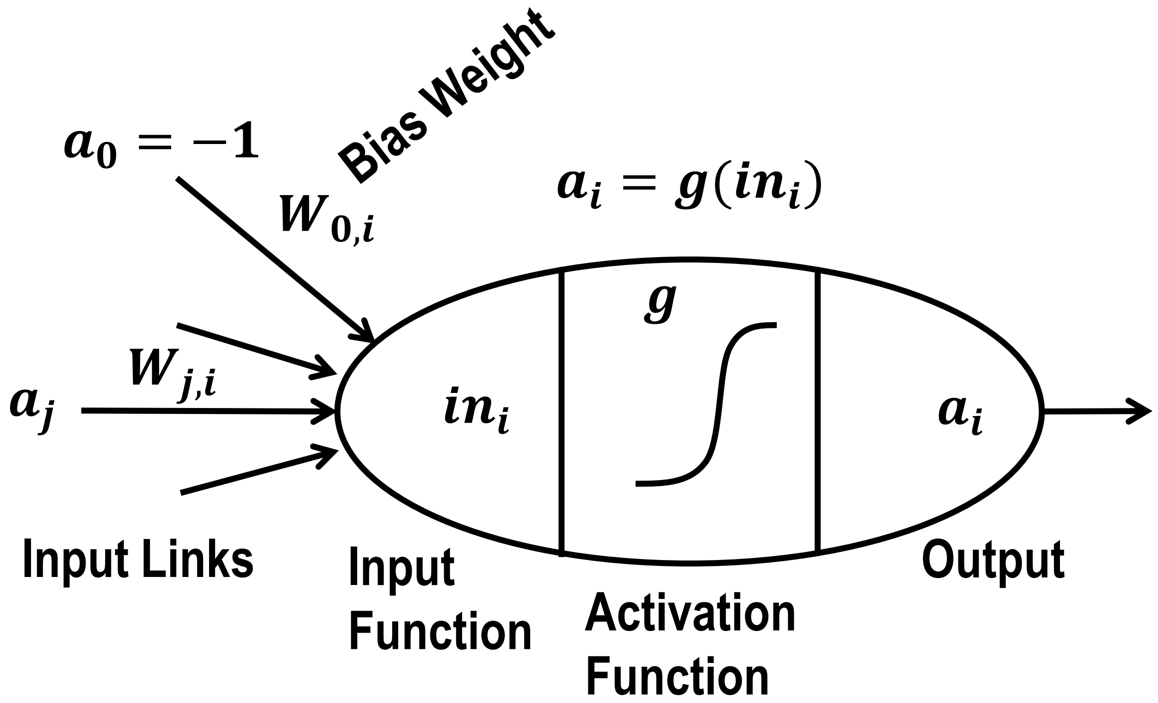
Neural Networks

A neural network consists of a set of nodes (neurons/units) connected by links

- Each link has a numeric weight

Each unit has:

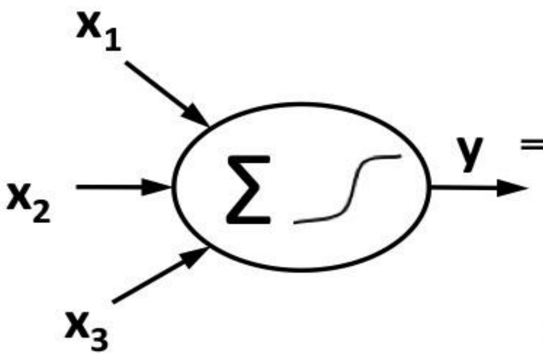
- a set of input links from other units,
- a set of output links to other units,
- a current activation level, and
- an activation function to compute the activation level in the next time step.



$$in_i = \sum_{j=0}^{n} W_{j,i} a_j \qquad a_i = g(in_i) = g\left(\sum_{j=0}^{n} W_{j,i} a_j\right)$$

Types of Neural Networks

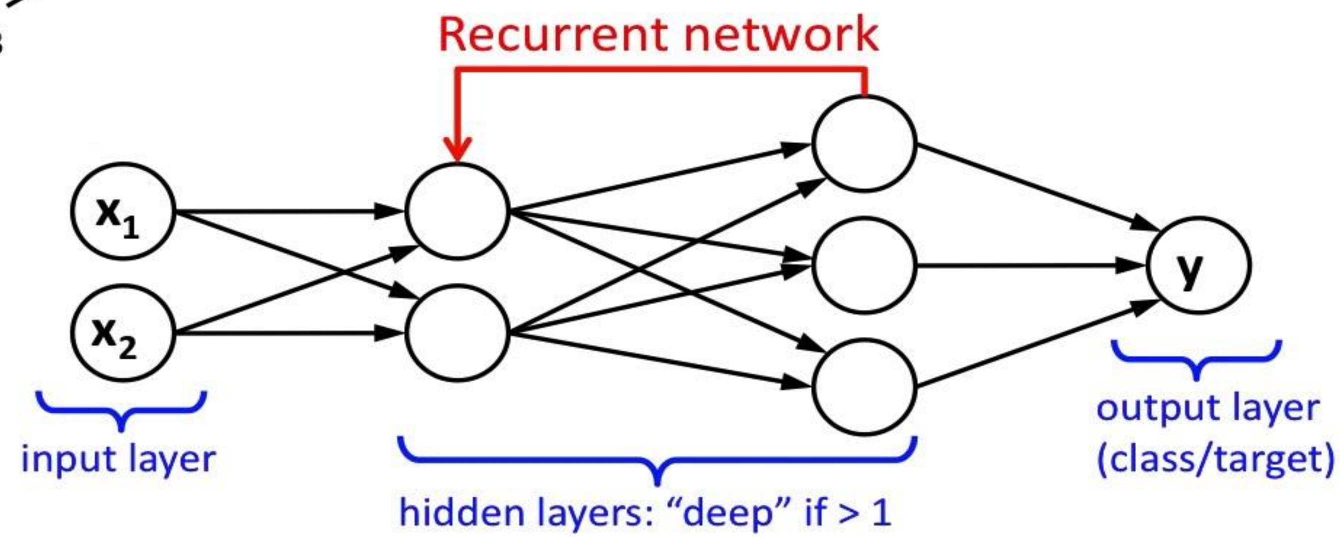
Activation Functions	
Name	**Formula**
Identity	$A(x) = x$
Sigmoid	$A(x) = \frac{1}{1 + e^{-x}}$
Tanh	$A(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Step	$A(x) = \begin{matrix} -1 & if\ x < 0 \\ 1 & if\ x \geq 0 \end{matrix}$



$$y = \sigma\left(b + \sum_i w_{ij} x_i\right)$$

Single neuron: perceptron,

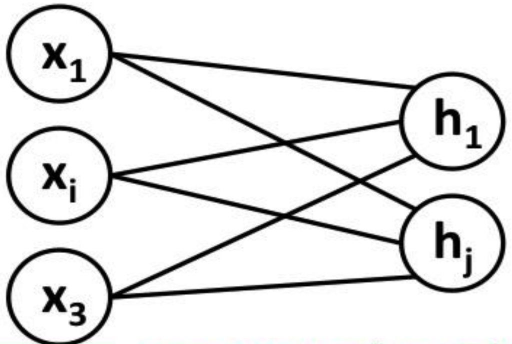
linear / logistic regression



Feed-forward network
(no cycles) -- non-linear classification & regression

P (input | hidden)

$$\sigma\left(\beta_i + \sum_j w_{ij} h_j\right) =$$



$$= \sigma\left(b_j + \sum_i w_{ij} x_i\right)$$

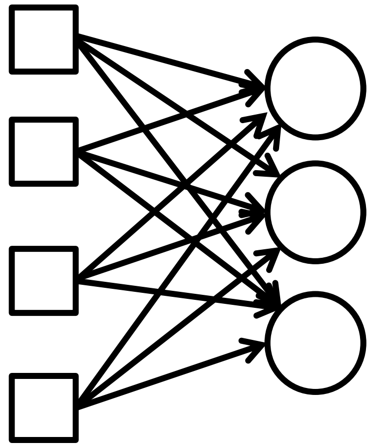
P (hidden | input)

same set of weights

Symmetric (RBM)
unsupervised, trained to maximize likelihood of input data

a mixture model

Learning in Single Layered Networks



Idea: Optimize the weights so as to minimize error function:

$$E = \frac{1}{2} Err^2 = \frac{1}{2}\left(y - g\left(\sum_{j=0}^{n} W_j x_j\right)\right)^2$$

We can use gradient descent to reduce the squared error by calculating the partial derivative of E with respect to each weight.

$$\frac{\partial E}{\partial W_j}$$

$$= Err \times \frac{\partial Err}{\partial W_j}$$

$$= Err \times \frac{\partial}{\partial W_j}\left(y - g\left(\sum_{j=0}^{n} W_j x_j\right)\right)$$

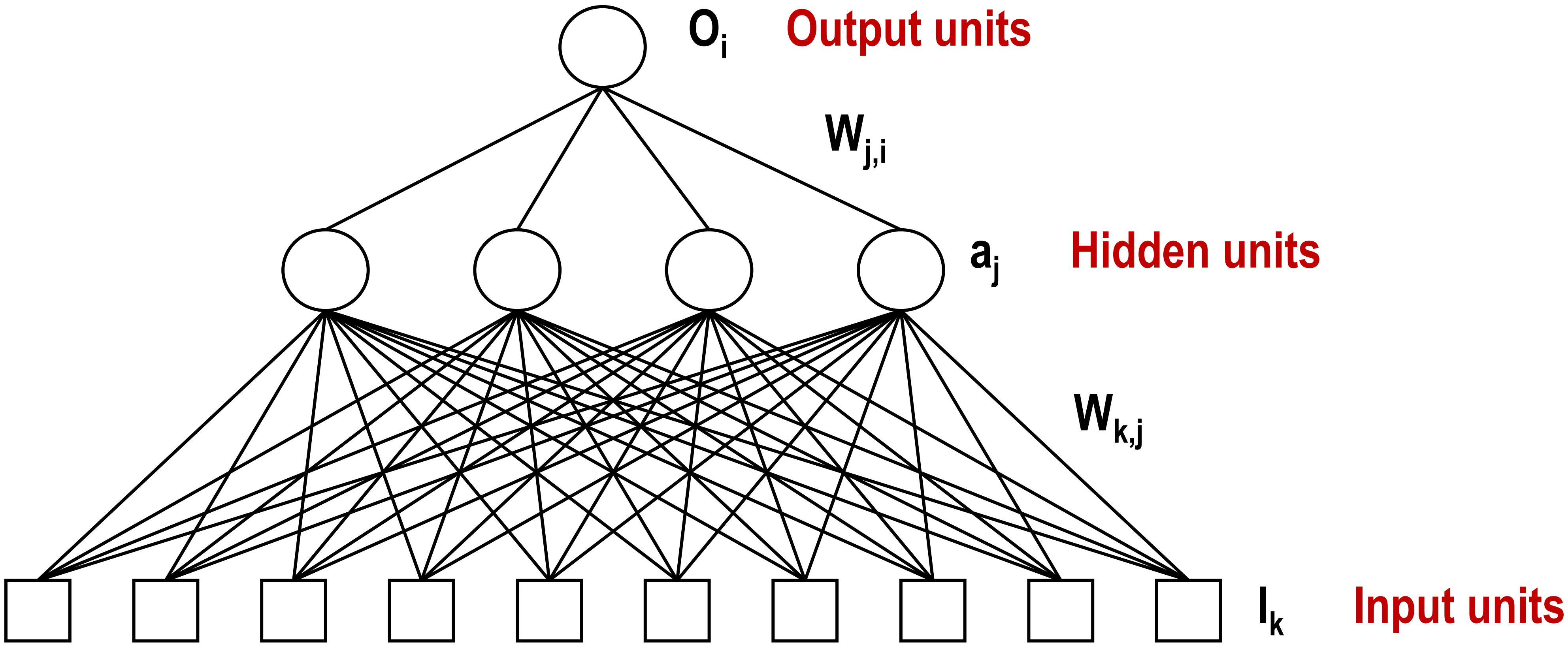
$$= -\mathrm{Err} \times g'(in) \times x_j$$

Weight update rule:

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

where α is the learning rate

Multi-Layer Feed-Forward Network



Weight updation rule at the output layer: $W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$ (same as single layer)

However in multilayer networks, the hidden layers also contribute to the error at the output.

So the important question is: *How do we revise the hidden layers?*

Back-Propagation Learning

- To update the connections between the input units and the hidden units, we need to define a quantity analogous to the error term for output nodes
- We do an error back-propagation, defining error as $\Delta_i = Err_i \times g'(in_i)$
- The idea is that a hidden node *j* is *responsible* for some fraction of the error in each of the output nodes to which it connects
- Thus the Δ_i values are divided according to the strength of the connection between the hidden node and the output node and are propagated back to provide the Δ_j values for the hidden layer.
- The propagation rule for the Δ values is the following:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$

- The update rule for the hidden layers is: $W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$

The mathematics behind the updation rule

The squared on a single example is defined as:

$$E = \frac{1}{2}\sum_i (y_i - a_i)^2$$

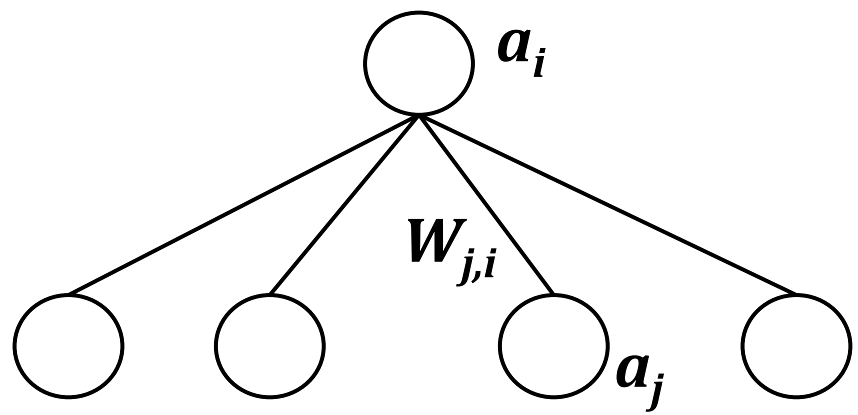
where the sum is over the nodes in the output layer. To obtain the gradient with respect to a specific weight $W_{j,i}$ in the output layer, we need only expand out the activation a_i as all other terms in the summation are unaffected by $W_{j,i}$

$$\frac{\partial E}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial g(in_i)}{\partial W_{j,i}} = -(y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial W_{j,i}}$$

$$= -(y_i - a_i)g'(in_i)\frac{\partial}{\partial W_{j,i}}\left(\sum_j W_{j,i}a_j\right)$$

$$= -(y_i - a_i)g'(in_i)a_j = -a_j\Delta_i$$

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$



The mathematics contd.

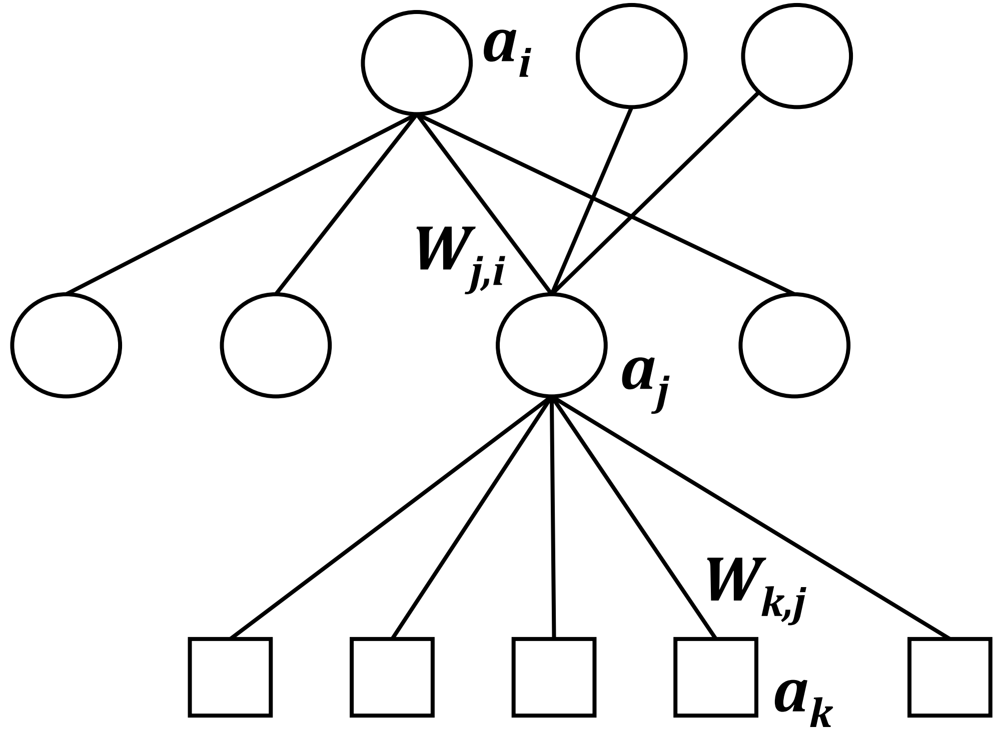
$$\frac{\partial E}{\partial W_{k,j}} = -\sum_i (y_i - a_i)\frac{\partial g(in_i)}{\partial W_{k,j}} = -\sum_i (y_i - a_i) g'(in_i)\frac{\partial in_i}{\partial W_{k,j}}$$

$$= -\sum_i \Delta_i \frac{\partial}{\partial W_{k,j}}\left(\sum_j W_{j,i} a_j\right) = -\sum_i \Delta_i W_{j,i}\frac{\partial a_j}{\partial W_{k,j}} = -\sum_i \Delta_i W_{j,i}\frac{\partial g(in_j)}{\partial W_{k,j}}$$

$$= -\sum_i \Delta_i W_{j,i} g'(in_j)\frac{\partial in_j}{\partial W_{k,j}}$$

$$= -\sum_i \Delta_i W_{j,i} g'(in_j)\frac{\partial}{\partial W_{k,j}}\left(\sum_k W_{k,j} a_k\right)$$

$$= -\sum_i \Delta_i W_{j,i} g'(in_j) a_k = -a_k \Delta_j$$



$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$$

Gradient Descent

- The weight updation rules define a single step of gradient descent
- Each training sample is presented and weights are updated
- This continues, until the training error converges to a (possibly local) minima

- At the end of the learning phase, the network is ready for use (generalization)

Boltzmann Machines

A Boltzmann machine is a network of units with an *energy* defined for the overall network. Its units produce binary results. The global energy, E, is:

$$E = -\left(\sum_{i<j} w_{ij} s_i s_j + \sum_i \theta_i s_i\right)$$

where:

- w_{ij} is the connection strength between unit j and unit i.
- s_i is the state, $s_i \in \{0,1\}$, of unit i
- θ_i is the bias of unit i in the global energy function. ($-\theta_i$ is the activation threshold for the unit)

$$\Delta E_i = \sum_{j>i} w_{ij} s_j + \sum_{j<i} w_{ji} s_j + \theta_i$$

- From this we obtain (the scalar T is called the *temperature*):

$$p_{i=on} = \frac{1}{1 + exp\left(-\frac{\Delta E_i}{T}\right)}$$

Explainable AI – Why Do You Think It Will Be Successful?

Source: DARPA