The National University of Computer and
Emerging Sciences

Introduction to Neural Networks

Machine Learning for Data Science

Dr. Akhtar Jamil

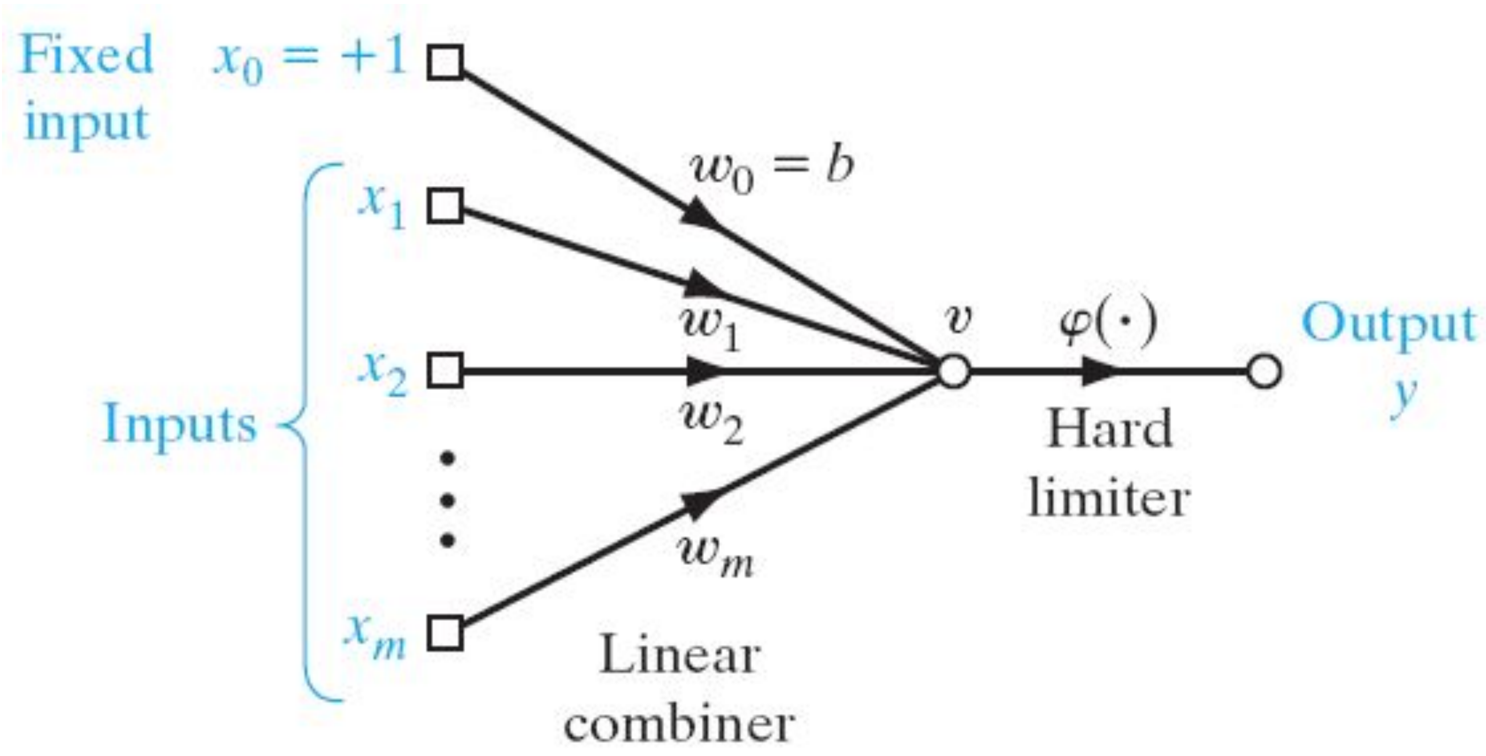
Department of Computer Science

Goals

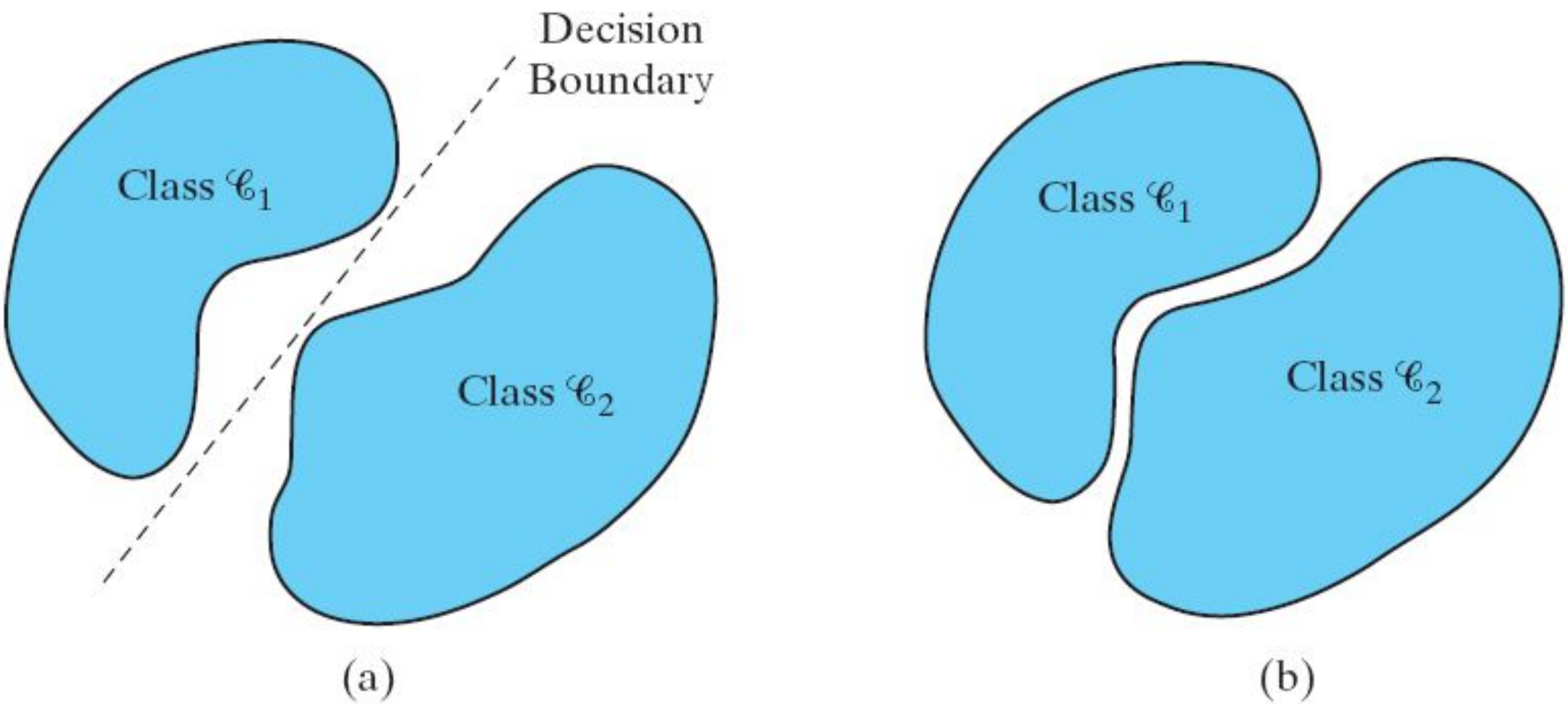
- Review of Previous Lecture
- Today's Lecture
  - Backpropagation

Previous Lecture

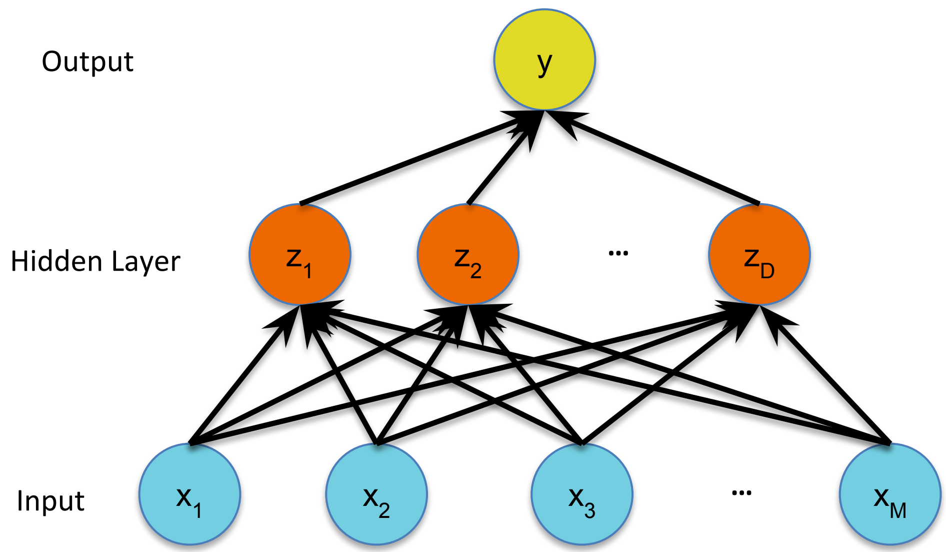
Rosenblatt's perceptron model



Rosenblatt's perceptron model



Neural Network



(E) Output (sigmoid)
$y = \frac{1}{1+\exp(-b)}$

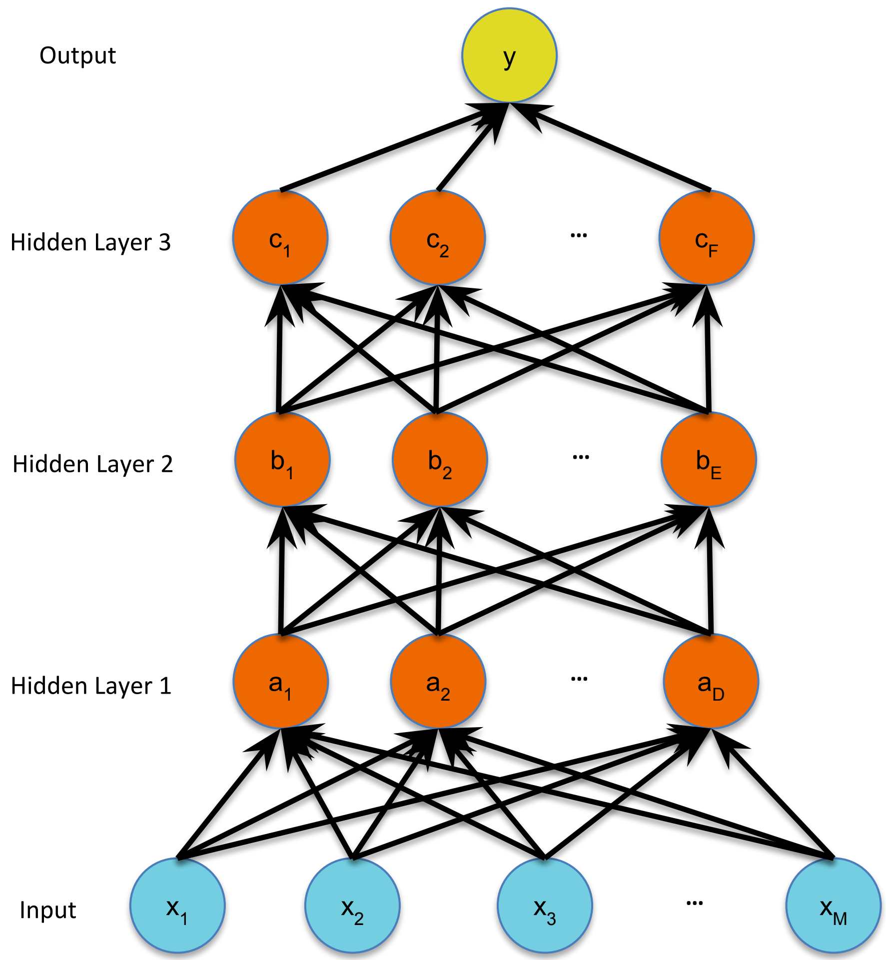
(D) Output (linear)
$b = \sum_{j=0}^{D} \beta_j z_j$

(C) Hidden (sigmoid)
$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

(B) Hidden (linear)
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

(A) Input
Given $x_i, \ \forall i$

Deeper Networks

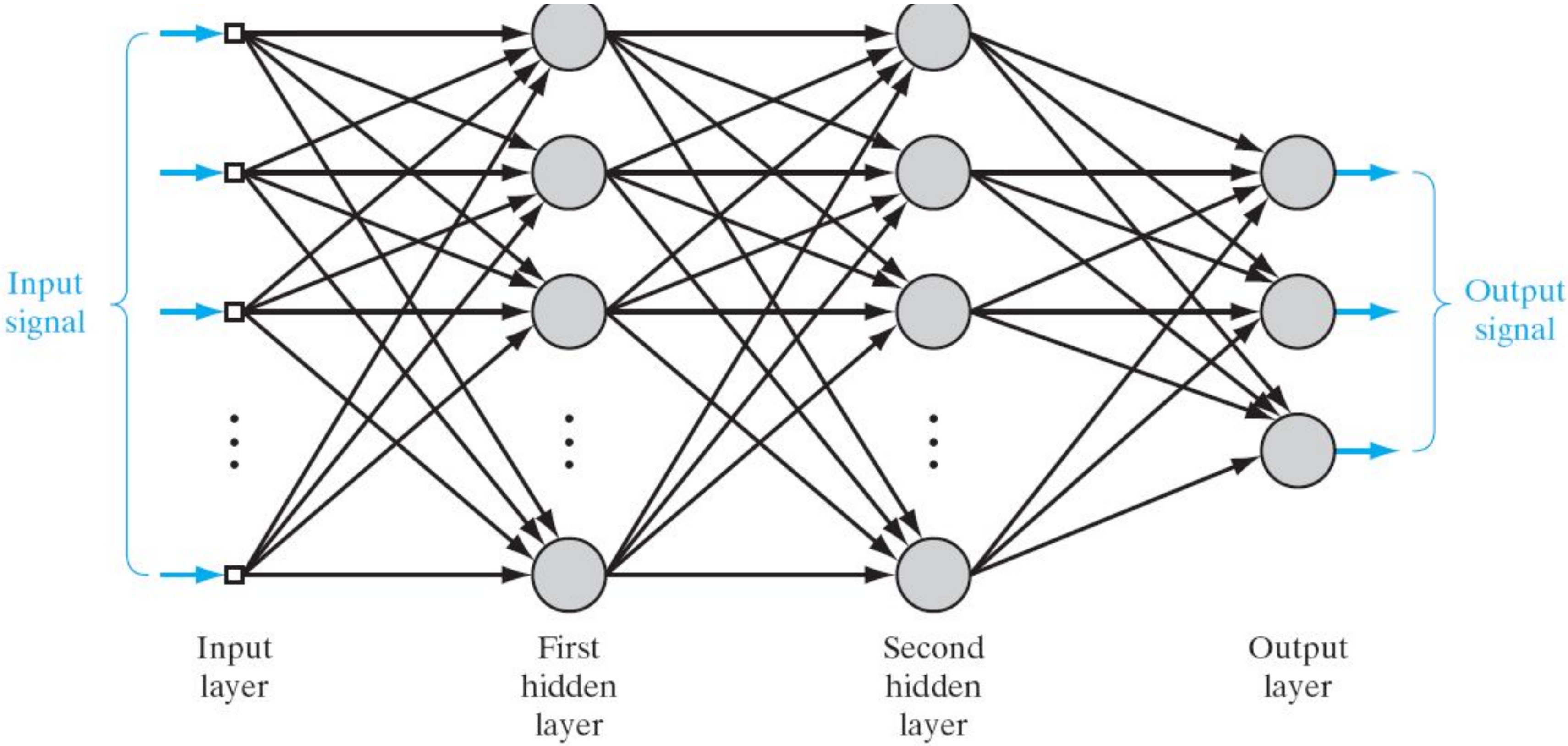


Today’s Lecture

Neural Network

- Most commonly used architecture is known as the multilayer perceptron.
- Basic features of MLP
  - A differentiable nonlinear activation function
  - One or more hidden layers
  - High degree of connectivity (fully connected)

Neural Network



Neural Network

- A popular method for the training of multilayer perceptron is the back-propagation algorithm (Generalized Delta Rule)
- ***Forward phase***:
  - The input data is propagated through the network, layer by layer, until it reaches the output.
- ***Backward phase:***
  - An error is calculated by comparing the output of the network with a desired response.
  - This error is propagated through the network in backward direction.

Neural Network

- Network architecture or topology plays an important role for neural net classification.
  - Optimal topology for the problem at hand.
- Empirically find the network architectures
  - The number of hidden layers, units, feedback connections, and so on.
- Selecting models (network topologies) and estimating parameters (training via backpropagation) enable you to try out alternate models.

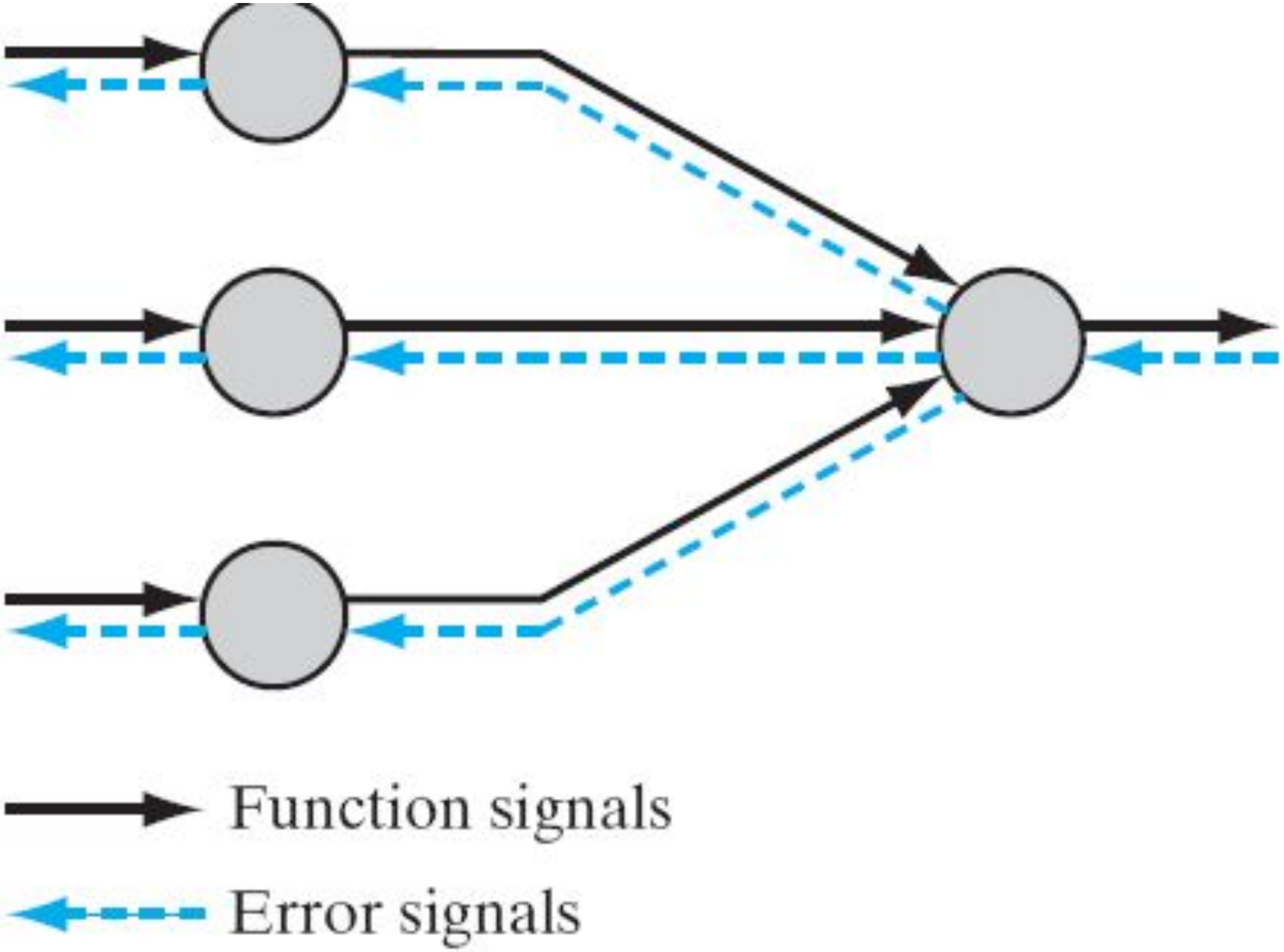
Neural Network

- Adjustments must be made to the weights of the network.
- Calculation of the adjustments for the output layer is straightforward
- It is more challenging for the hidden layers.

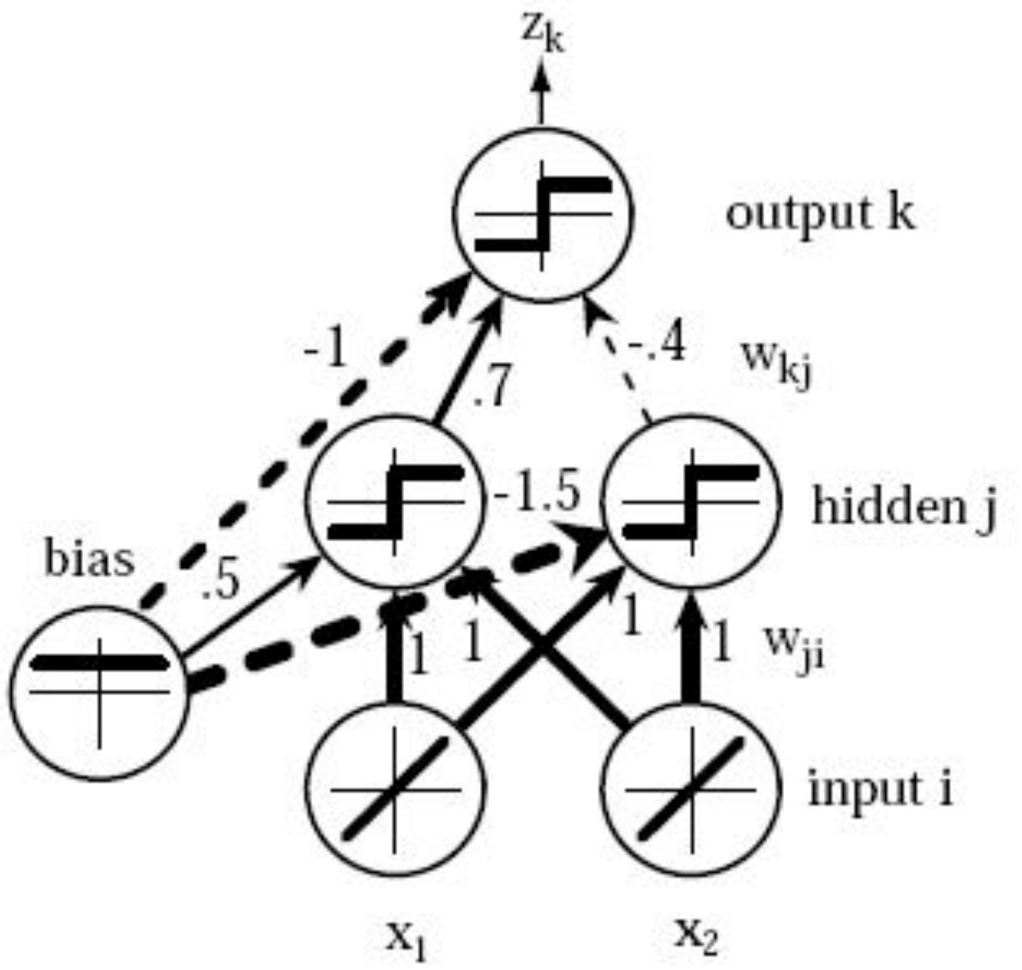
Neural Network

- **Hidden Neurons**
- The hidden neurons act as *feature detectors*
- The hidden neurons gradually “discover” the main features to understand the training data.
- They perform nonlinear transformations on the input
  - *Feature space.*
- Classes may be more easily separated from each other

Neural Network



Neural Network



Neural Network

- Each hidden unit performs the weighted sum of its inputs to form its (scalar) *net activation or net*.

$$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} \equiv \mathbf{w}_j^t \mathbf{x}$$

- The subscript *i* indices' units on the input layer,
- *j* for the hidden;
- w_{ji} denotes the input-to-hidden layer weights at the hidden unit *j*.

Neural Network

- Each hidden unit produces an output which is a nonlinear function of its activation, *f*(*net*)

$$y_j = f(\text{net}_j)$$

- f(x) is an activation function.
  - Many choices
  - Also called the transfer function or “nonlinearity” of a unit
  - Example, Sigmoid function

$$f(net) = \frac{1}{1 + e^{-Wx}}$$

Neural Network

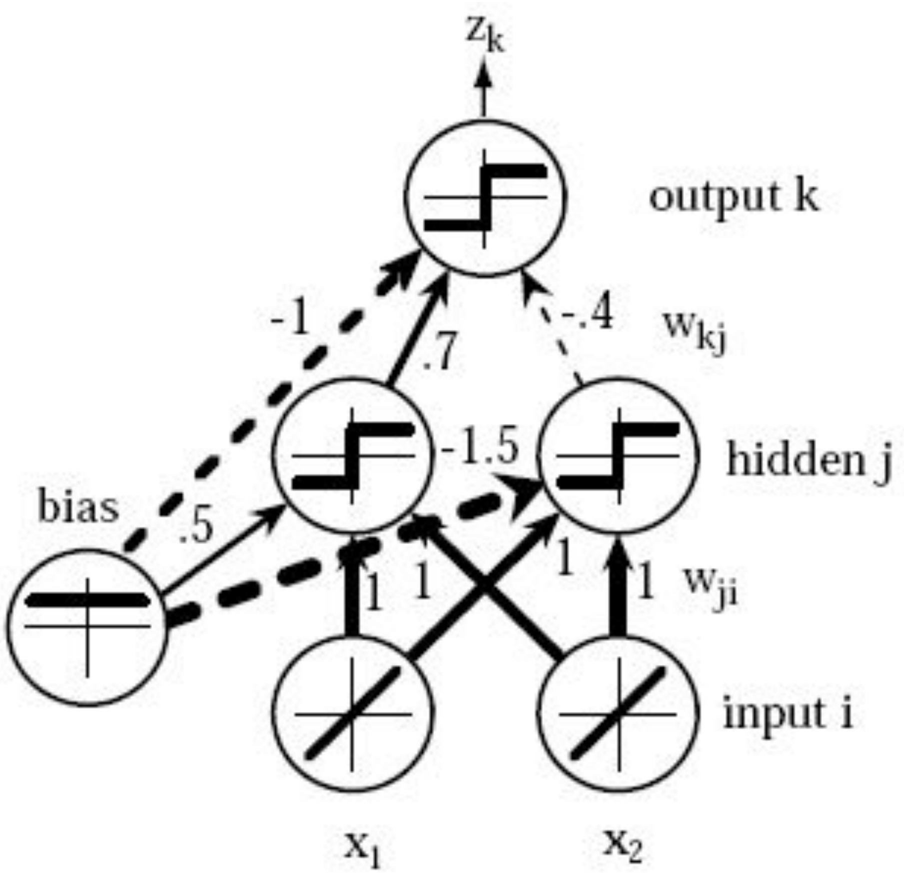
- Each output unit (neuron) similarly computes its net activation based on the hidden unit signals as:

$$net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} = \mathbf{w}_k^t \mathbf{y}$$

- Each output unit then computes the nonlinear function of its *net*

$$z_k = f(\text{net}_k)$$

Neural Network



Inputs		Output
X_1	X_2	X
0	0	0
0	1	1
1	0	1
1	1	0

Neural Network

- For a three layer network, we can generalize the formula to get the final output as

$$g_k(\mathrm{x}) \equiv z_k = f\left(\sum_{j=1}^{n_H} w_{kj} f\left(\sum_{i=1}^{d} w_{ji}x_i + w_{j0}\right) + w_{k0}\right)$$

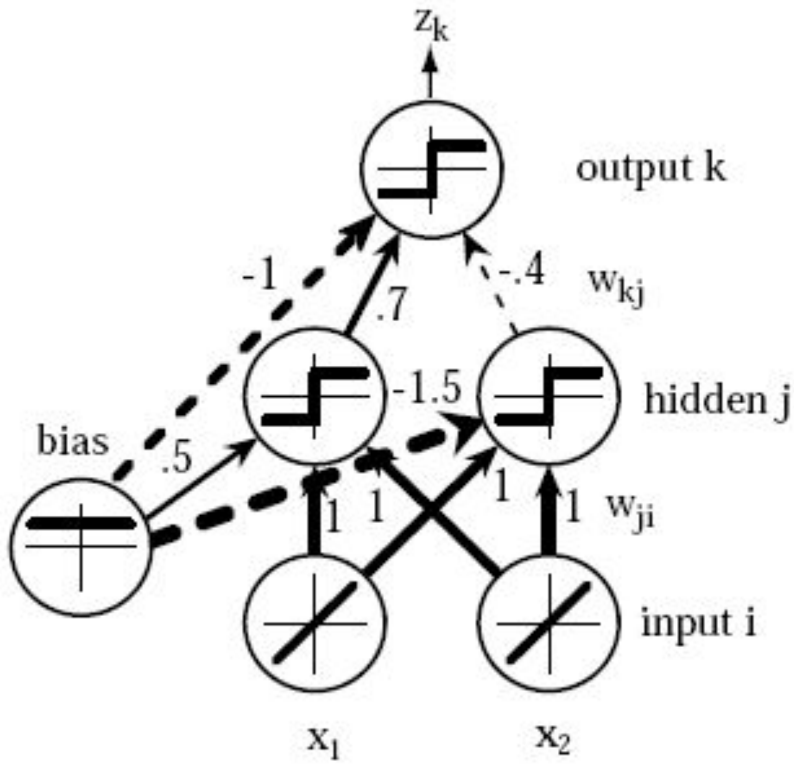
- Can we put different activation functions at the output layer to differ from those in the hidden layer?
  - Yes ☺

Backpropagation algorithm

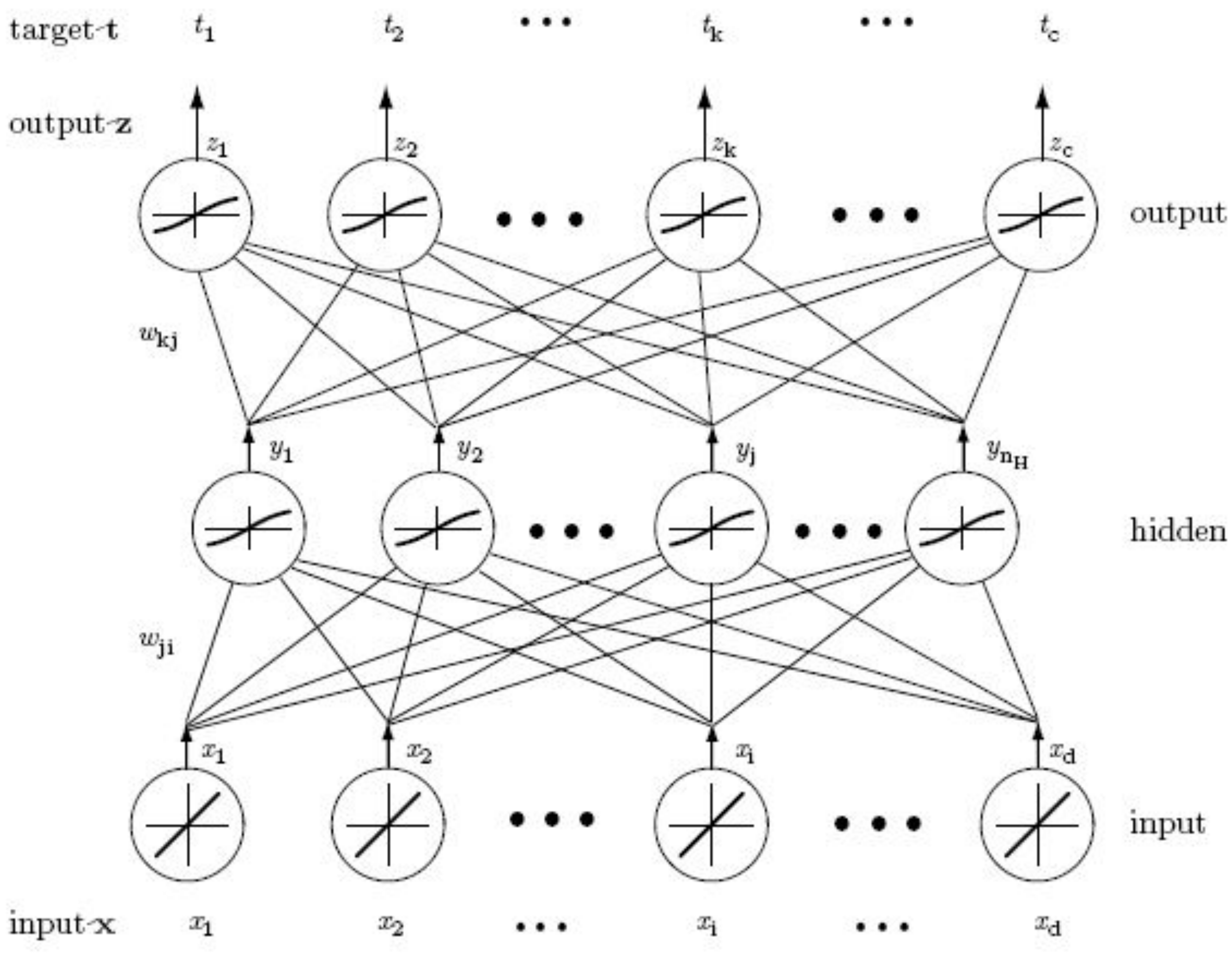
- The learning algorithm need to set the weights based on training samples and desired output.
- Backpropagation is one of the simplest method for supervised training of multilayer neural networks
- It is an extension of least mean squared error (LMS)

Backpropagation algorithm

- Input-to-hidden weights learning is challenging
  - "proper outputs" for a hidden unit are unknown
- This is called the *credit assignment* problem.
- Backpropagation can handle the problem for weight update



Backpropagation algorithm



Backpropagation algorithm

- For training the network, we pass the training data and get determine the output
- This output is then compared with the Target Labels (Supervised Learning)
- Cost function will match the produced output to the target labels to calculate the error
- Goal: To minimize the error

Backpropagation algorithm

- We can calculate the desired error for the output units as:

$$J(\mathbf{w}) \equiv 1/2 \sum_{k=1}^{c} (t_k - z_k)^2 = 1/2(\mathbf{t} - \mathbf{z})^2$$

- The backpropagation learning rule is based on gradient descent.
- The weights are initialized with random values, and are changed in a direction that will reduce the error:

$$\Delta\mathbf{w} = -\eta \frac{\partial J}{\partial \mathbf{w}}$$

- where η is the *learning rate*

Backpropagation algorithm

- The weight vector will be updated in iterative manner as:

$$\mathrm{w}(m + 1) = \mathrm{w}(m) + \Delta\mathrm{w}(m)$$

$$\mathrm{w}(m + 1) = \mathrm{w}(m) - \eta \frac{\partial J}{\partial w}$$

- The weights throughout the network can be updated using.

$$\Delta w_{mn} = -\eta \frac{\partial J}{\partial w_{mn}}$$

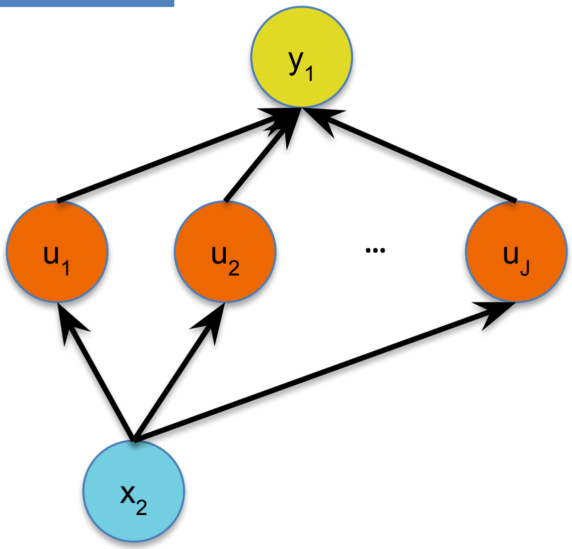
- Now the challenge is to update the weights
  - Some are not explicitly dependent on incoming weights

Chain Rule

Give $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k$$



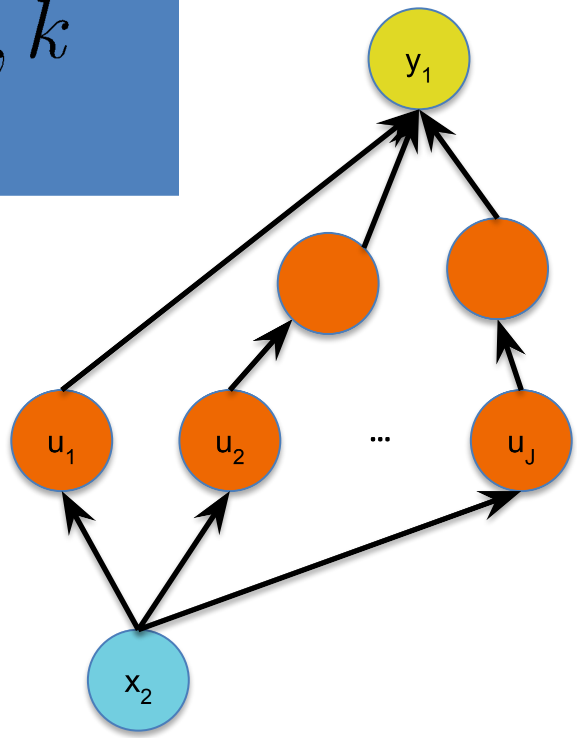
Chain Rule

Given: $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j}\frac{du_j}{dx_k}, \quad \forall i, k$$

Backpropagation is just repeated application of the **chain rule** from Calculus.



Backpropagation algorithm

- Consider first the hidden-to-output weights, w_{jk}.
- we must use the chain rule for differentiation:

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = \delta_k \frac{\partial net_k}{\partial w_{kj}}$$

- *Sensitivity* of output unit *k* is defined to be

$$\delta_k \equiv -\partial J / \partial net_k$$

Backpropagation algorithm

- Differentiate the cost function with unit's net activation to find how the overall error changes

$$J(\mathbf{w}) \equiv 1/2 \sum_{k=1}^{c} (t_k - z_k)^2$$

$$\delta_k \equiv -\partial J/\partial net_k = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k)$$

- From the equation

$$\frac{\partial net_k}{\partial w_{kj}} = y_i$$

$$net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0}$$

Backpropagation algorithm

- Taken together, these results give the weight update (learning rule) for the hidden-to-output weights:

$$\Delta w_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) f'(\text{net}_k) y_j$$

- This makes intuitive sense:
  - The weight update at unit *k should* depend on $t_k - z_k$
  - if we get the desired output ($t_k = z_k$), then there should be no weight change.

Backpropagation algorithm

- The learning rule for the input-to-hidden units can be calculated as:

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$$

Backpropagation algorithm

$$\begin{aligned}
\frac{\partial J}{\partial y_j} &= \frac{\partial}{\partial y_j}\left[1/2\sum_{k=1}^{c}(t_k - z_k)^2\right] \\
&= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}(t_k - z_k)f'(net_k)w_{jk}.
\end{aligned}$$

This last equation expresses how the hidden unit output, yj , affects the error at each output unit.

Backpropagation algorithm

- Similar to the sensitivity we calculated for the output units, we can also calculate the sensitivity for hidden units similar to previous equation as:

$$\delta_j \equiv f'(net_j) \sum_{k=1}^{c} w_{kj}\delta_k.$$

- This equation indicates that the sensitivity at a hidden unit is simply the sum of the individual sensitivities at the output units weighted by the hidden-to-output weights w_{jk}, all multiplied by $f(net_j)$.

Backpropagation algorithm

- Thus the learning rule for the input-to-hidden weights is:

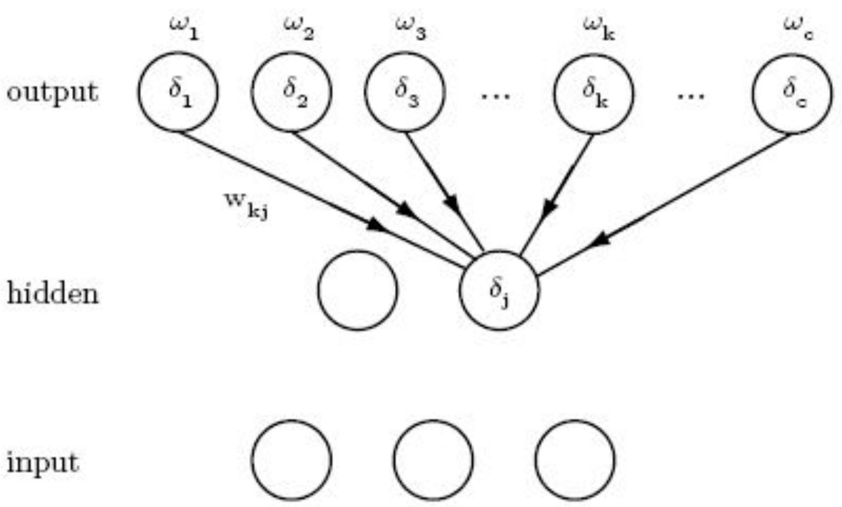
$$\Delta w_{ji} = \eta x_i \delta_j = \eta x_i f'(\text{net}_j) \sum_{k=1}^{c} w_{kj} \delta_k$$

- The "backpropagation of errors" (sensitivities δk) must be propagated from the output layer back to the hidden layer in order to perform the learning of the input-to-hidden weights.

- It is just performing a gradient descent in each layer with chain rule to update the model parameters (weights).

Backpropagation algorithm

- The sensitivity at a hidden unit is proportional to the weighted sum of the sensitivities at the output units.
- The output unit sensitivities are thus propagated "back" to the hidden units.



Backpropagation

- We can generalize the backpropagation algorithm to feed-forward networks with any number of layers
- It has the power to update the weights for all connections using gradient descent.
- Can also be used for training recurrent neural networks
  - With feedback connections

References

- Chapter 4, Neural Networks and Learning Machines, Haykin
- Chapter 5, Pattern Recognition and Machine Learning, Bishop

Thank You ☺