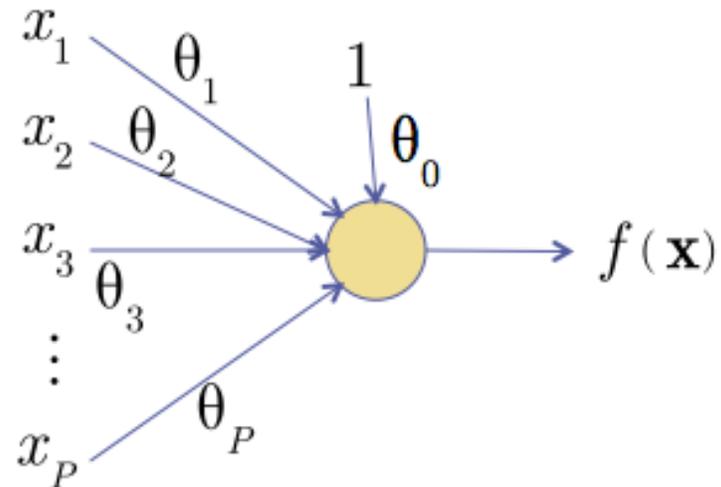
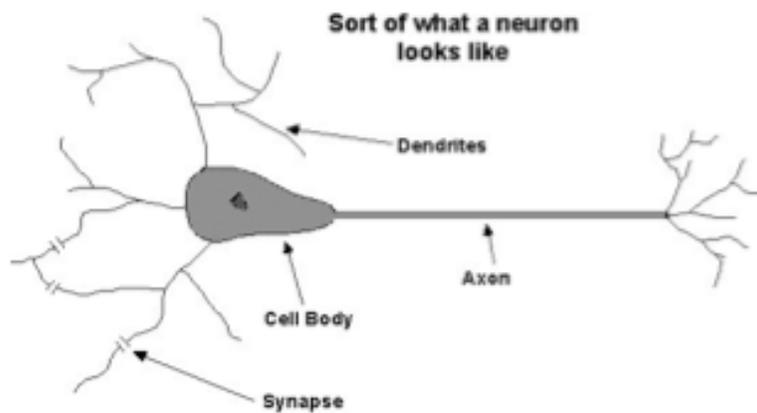


Neural Networks



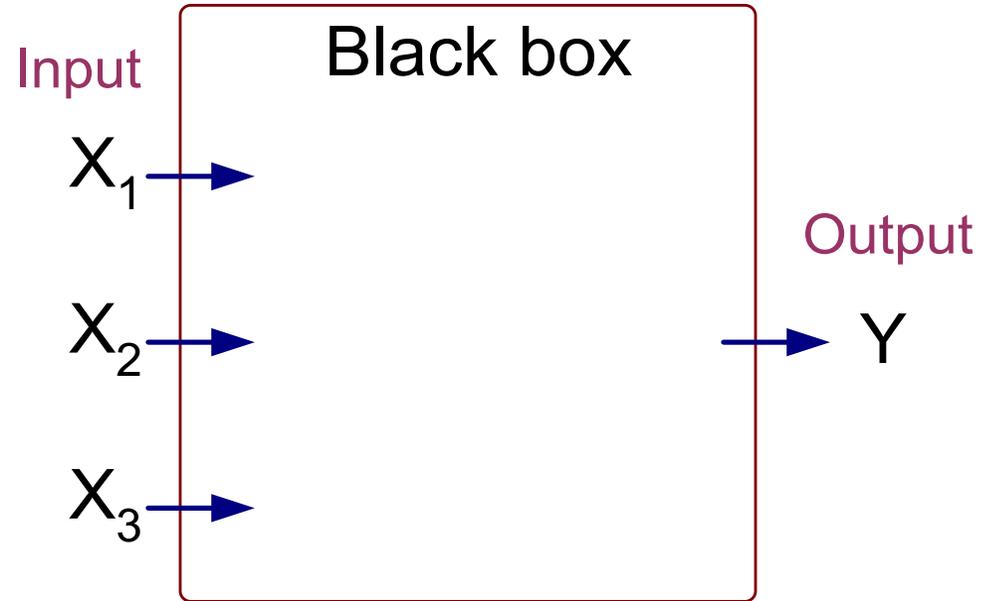
The Neuron Metaphor

- Neurons
 - accept information from multiple inputs,
 - transmit information to other neurons.
- Multiply inputs by weights along edges
- Apply some function to the set of inputs at each node



Artificial Neural Networks (ANN)

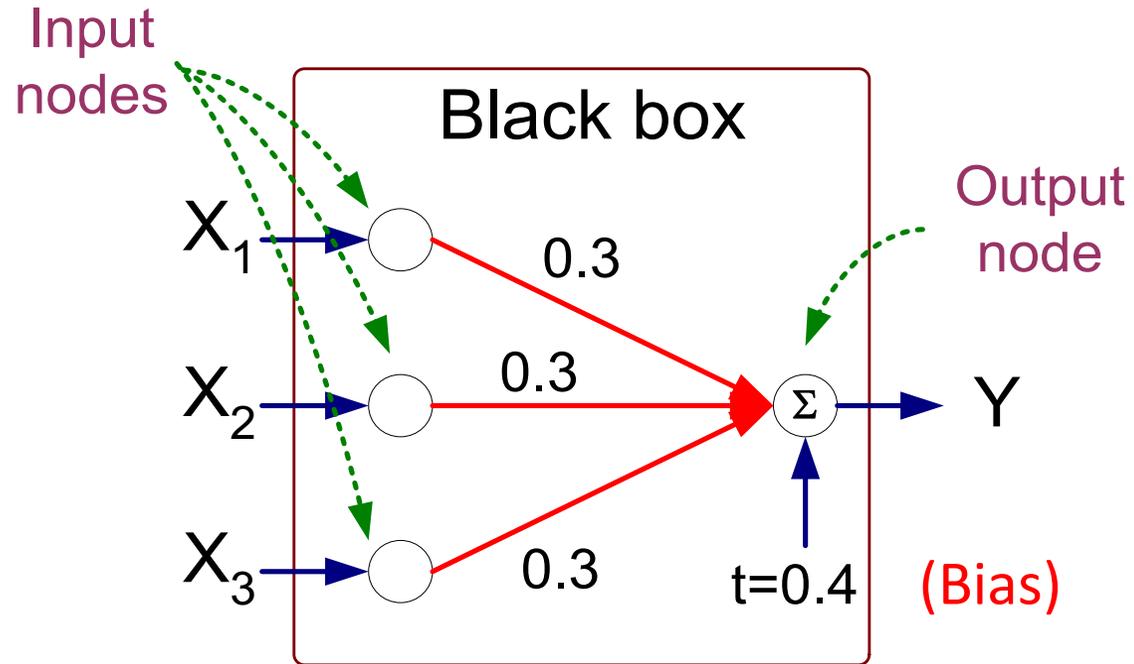
X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



Output Y is 1 if at least two of the three inputs are equal to 1.

Artificial Neural Networks (ANN)

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

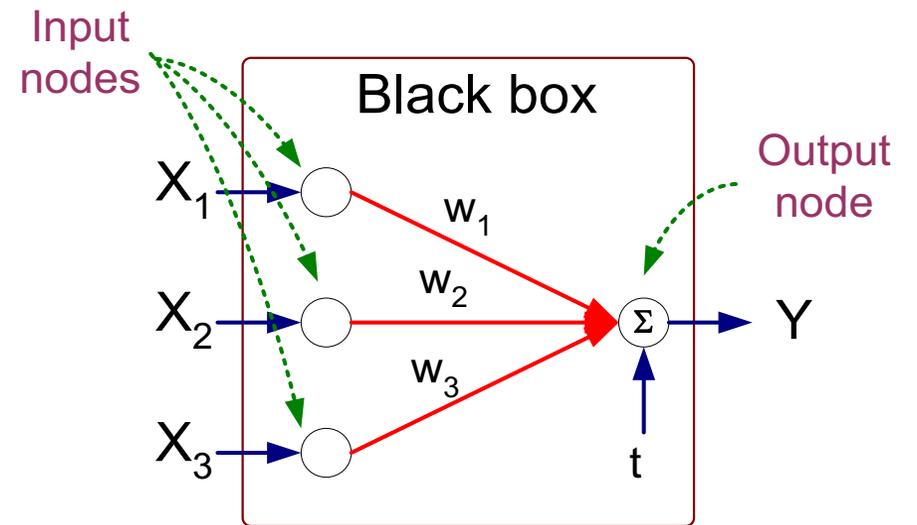


$$Y = \text{sign}(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where } \text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Artificial Neural Networks (ANN)

- Model is an assembly of inter-connected nodes and weighted links
- Output node sums up each of its input value according to the weights of its links
- Compare output node against some threshold t (also named bias b)



$$Y = \text{sign}\left(\sum_{i=1}^d w_i X_i - t\right)$$
$$= \text{sign}\left(\sum_{i=0}^d w_i X_i\right)$$

Characterizing the Artificial Neuron

- Input/Output signal may be:
 - Real value
 - Unipolar $\{0, 1\}$
 - Bipolar $[-1, +1]$
- **Weight** (w or sigma): θ_{ij} – strength of connection from unit j to unit i
- Learning amounts to **adjusting the weights** θ_{ij} by means of an **optimization algorithm** aiming to minimize a cost function, i.e., as in biological systems training a perceptron model amounts to adapting the weights of the links until they fit the input output relationships of the underlying data.

Characterizing the Artificial Neuron

- The bias b is a constant that can be written as $\theta_{i0}x_0$ with $x_0 = 1$ and $\theta_{i0} = b$ such that

$$net_i = \sum_{j=0}^n \theta_{ij}x_j$$

- The function $f(net_i(x))$ is the unit's **activation function**.
- The simplest case, f is the identity function, and the unit's output is just its net input. This is called a *linear unit*.
- Otherwise we can have a *sign unit*, or a *logistic unit*.

The Perceptron Classifier

Perceptron

- Single layer network
 - Contains only input and output nodes
- Activation function: $f = \text{sign}(w \bullet x)$
- Applying model is straightforward

$$Y = \text{sign}(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where } \text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

- $X_1 = 1, X_2 = 0, X_3 = 1 \Rightarrow y = \text{sign}(0.2) = 1$

Learning Iterative Procedure

- During the training phase the weight parameters are adjusted until the outputs of the perceptron become consistent with the true outputs of the training examples.
- Initialize the weights (w_0, w_1, \dots, w_m)
- Repeat
 - For each training example (x_i, y_i)
 - Compute $f(w^{(k)}, x_i)$
 - Update the weights: $w^{(k+1)} = w^{(k)} + \lambda [y_i - f(w^{(k)}, x_i)] x_i$
- Until stopping condition is met

Iteration index

$$w^{(k+1)} = w^{(k)} + \lambda [y_i - f(w^{(k)}, x_i)] x_i$$

Learning rate

Perceptron Learning Rule

- Weight update formula:

$$w^{(k+1)} = w^{(k)} + \lambda [y_i - f(w^{(k)}, x_i)] x_i ; \lambda : \text{learning rate}$$

- Intuition:

- Update weight based on error: $e = [y_i - f(w^{(k)}, x_i)]$
- If $y=f(x,w)$, $e=0$: no update needed
- If $y>f(x,w)$, $e=2$: weight must be increased so that $f(x,w)$ will increase
- If $y<f(x,w)$, $e=-2$: weight must be decreased so that $f(x,w)$ will decrease

The Learning Rate

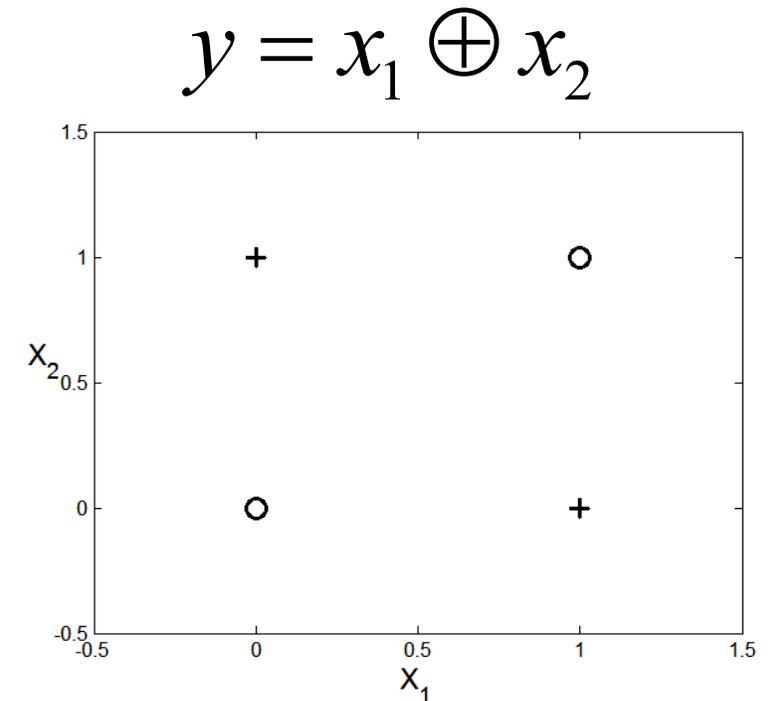
- Is a parameter with value between 0 and 1 used to control the amount of adjustment made in each iteration.
- If is close to 0 the new weight is mostly influenced by the value of the old weight.
- If it is close to 1, then the new weight is mostly influenced by the current adjustment.
- The learning rate can be adaptive: initially moderately large and the gradually decreases in subsequent iterations.

Nonlinearly Separable Data

- Since $f(w,x)$ is a linear combination of input variables, decision boundary is linear.
- For nonlinearly separable problems, the perceptron fails because no linear hyperplane can separate the data perfectly.
- An example of nonlinearly separable data is the XOR function.

XOR Data

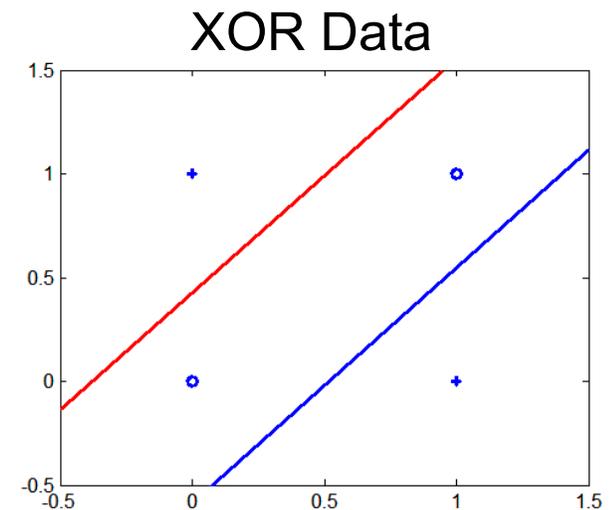
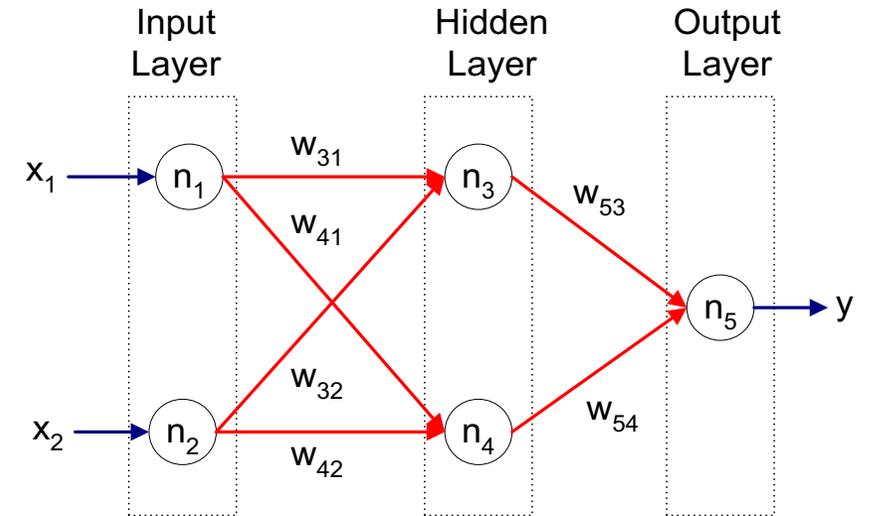
x_1	x_2	y
0	0	-1
1	0	1
0	1	1
1	1	-1



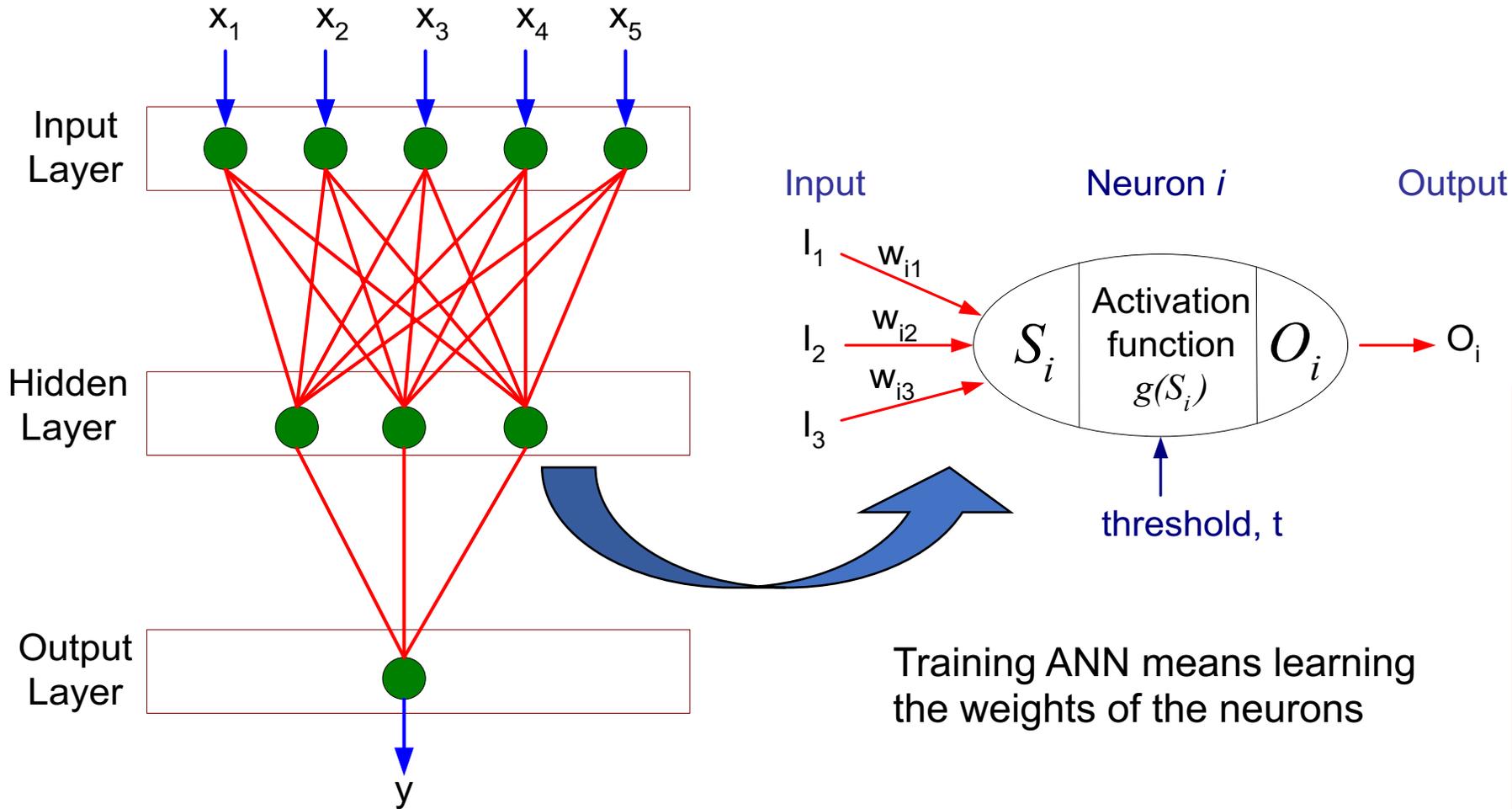
Multilayer Neural Network

Multilayer Neural Network

- **Hidden Layers:** intermediary layers between input and output layers.
- Different type of **activation functions** (sigmoid, linear, hyperbolic tangent, etc.).
- Multi-layer neural network can solve any type of classification task involving nonlinear decision surfaces.
- Perceptron is single layer.
- We can think to each hidden node as a perceptron that tries to construct one hyperplane, while the output node combines the results to return the decision boundary.



General Structure of ANN



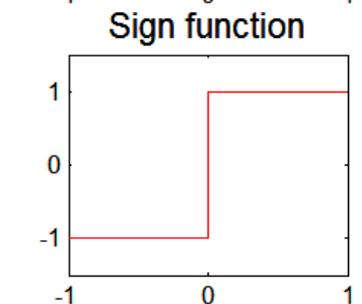
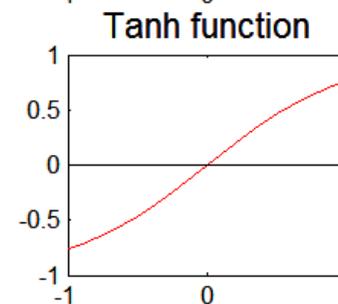
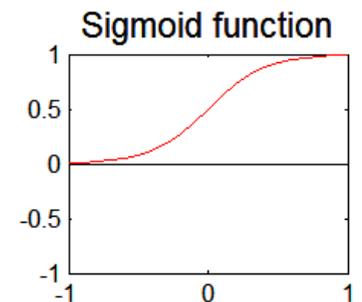
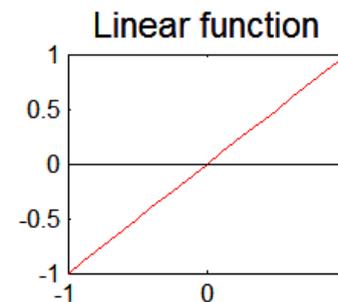
Training ANN means learning the weights of the neurons

- The neurons perform a linear transformation on this input using the weights and biases.
- An activation function is applied to it
- The output moves to the next hidden layer

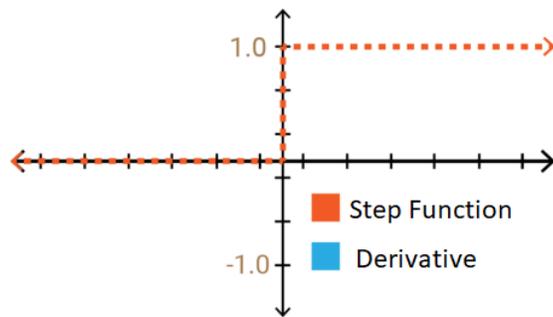
Artificial Neural Networks (ANN)

- Various types of neural network topology
 - single-layered network (perceptron) versus multi-layered network
 - Feed-forward versus recurrent network
- Various types of activation functions (f)

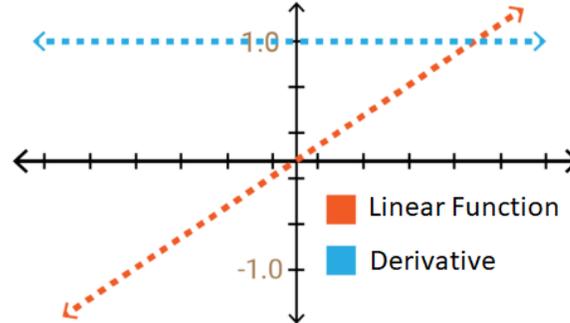
$$Y = f\left(\sum_i w_i X_i\right)$$



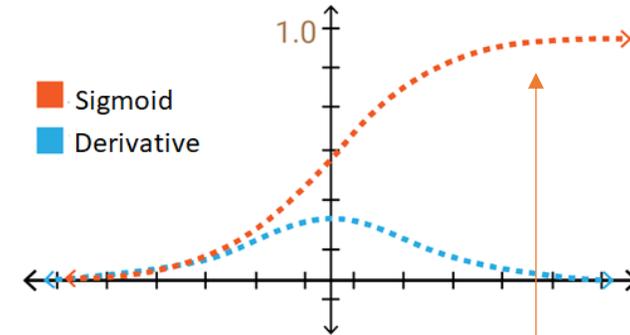
Activation Functions Summary



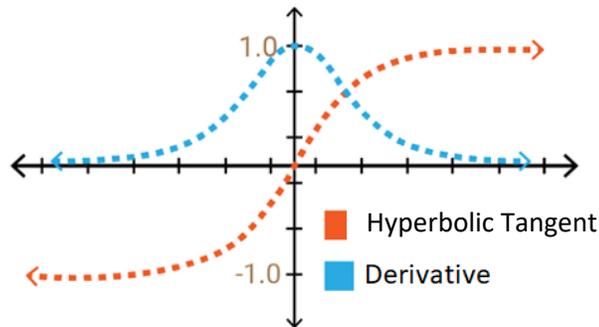
$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$



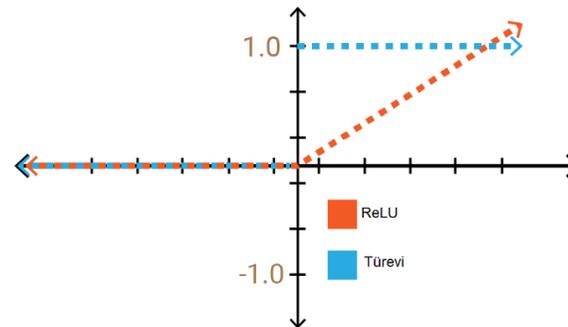
$$f(x) = x$$



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



$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



$$f(x) = \begin{cases} 0 \text{ (or } \epsilon) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

$$f(x_j) = \frac{e^{x_j}}{\sum_k e^{x_k}}$$

Softmax Function

Learning Multi-layer Neural Network

- Can we apply perceptron learning to each node, including hidden nodes?
- Perceptron computes error $e = y - f(w, x)$ and updates weights accordingly
- Problem: how to determine the true value of y for hidden nodes?
- Approximate error in hidden nodes by error in the output nodes
- Problems:
 - Not clear how adjustment in the hidden nodes affect overall error
 - No guarantee of convergence to optimal solution

Gradient Descent for Multilayer NN

- Error function to minimize: $E = \frac{1}{2} \sum_{i=1}^N \left(y_i - f\left(\sum_j w_j x_{ij}\right) \right)^2$
- Weight update: $w_j^{(k+1)} = w_j^{(k)} - \lambda \frac{\partial E}{\partial w_j}$
- Activation function f must be differentiable
- For sigmoid function: $w_j^{(k+1)} = w_j^{(k)} + \lambda \sum_i (y_i - o_i) o_i (1 - o_i) x_{ij}$
- Stochastic Gradient Descent (update the weight immediately)

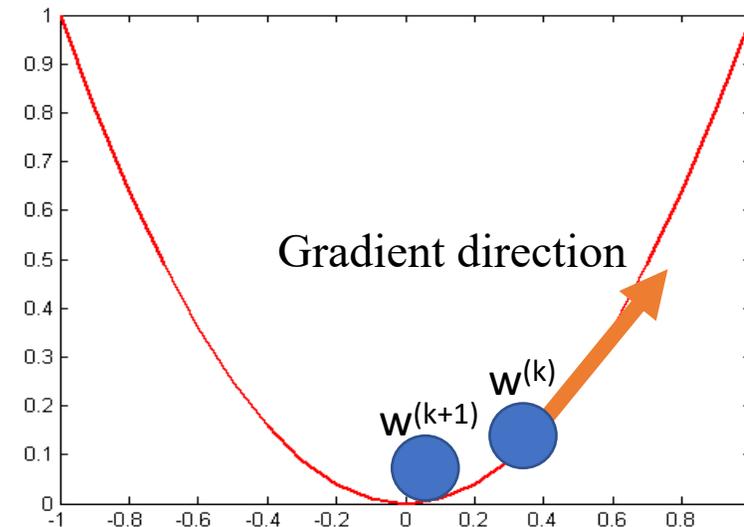
Quadratic function from which we can find a global minimum solution



Gradient Descent for Multilayer NN

- Weights are updated in the opposite direction of the gradient of the loss function.
- Gradient direction is the direction of uphill of the error function.
- By taking the negative we are going downhill.
- Hopefully to a minimum of the error.

$$w_j^{(k+1)} = w_j^{(k)} - \lambda \frac{\partial E}{\partial w_j}$$



Gradient Descent for Multilayer NN

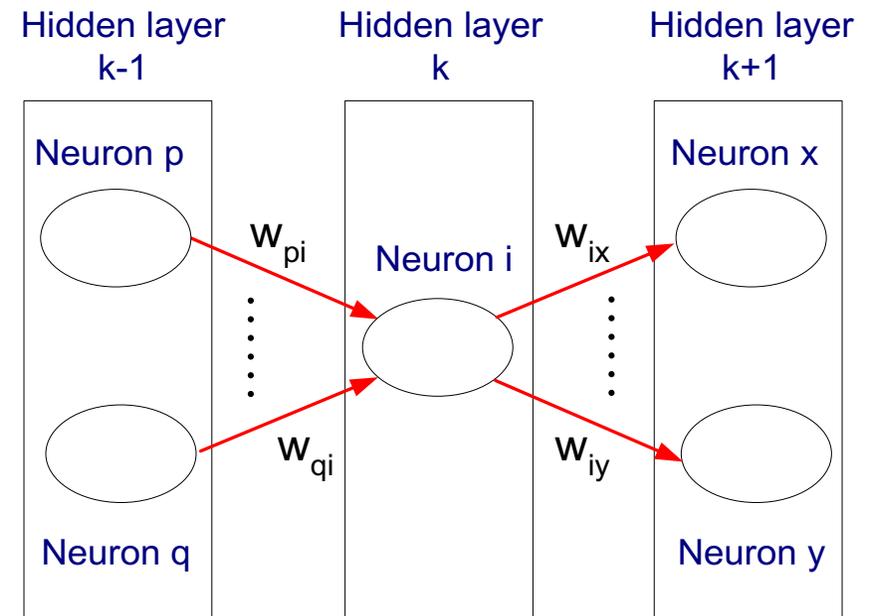
- For output neurons, weight update formula is the same as before (gradient descent for perceptron)

- For hidden neurons:

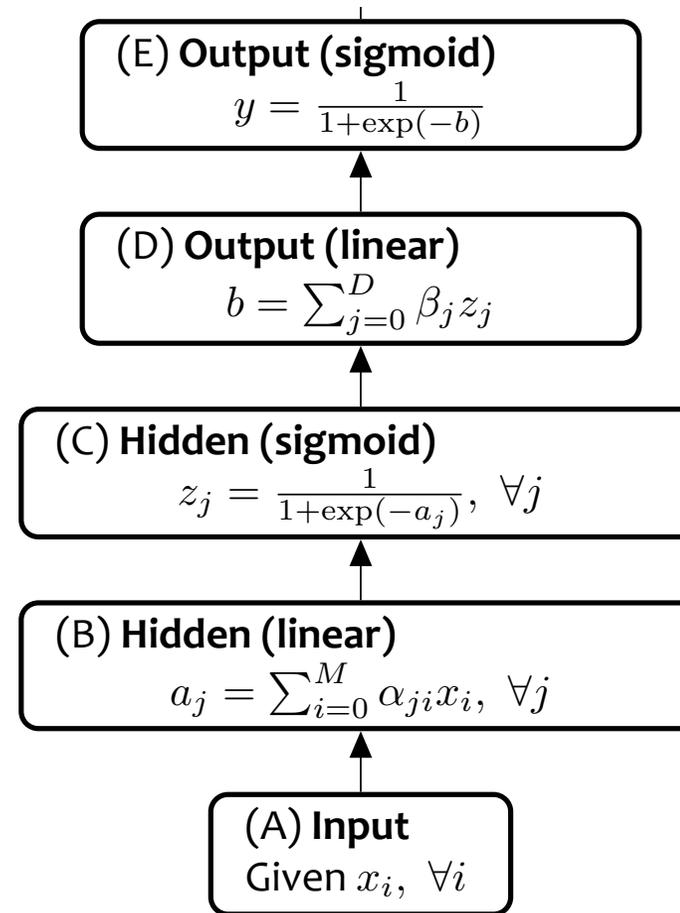
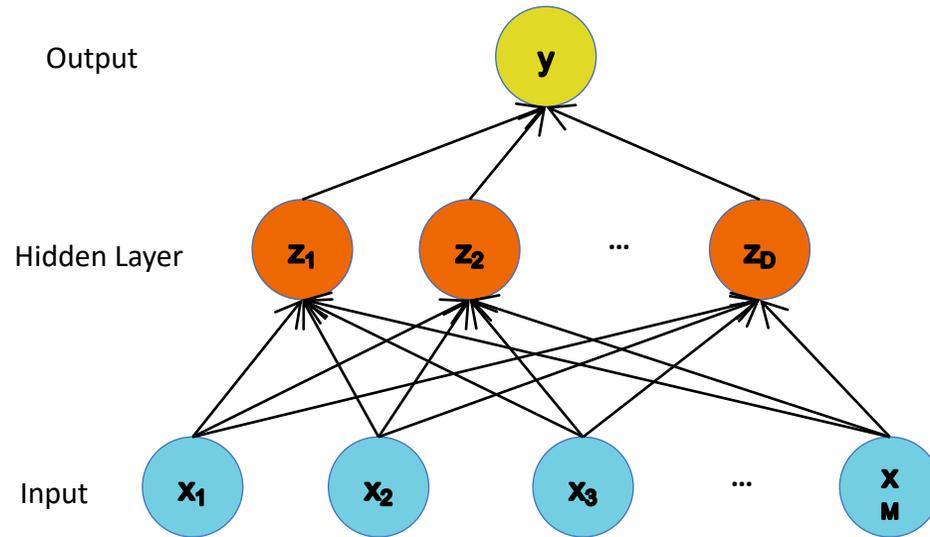
$$w_{pi}^{(k+1)} = w_{pi}^{(k)} + \lambda o_i (1 - o_i) \sum_{j \in \Phi_i} \delta_j w_{ij} x_{pi}$$

$$\text{Output neurons : } \delta_j = o_j (1 - o_j) (t_j - o_j)$$

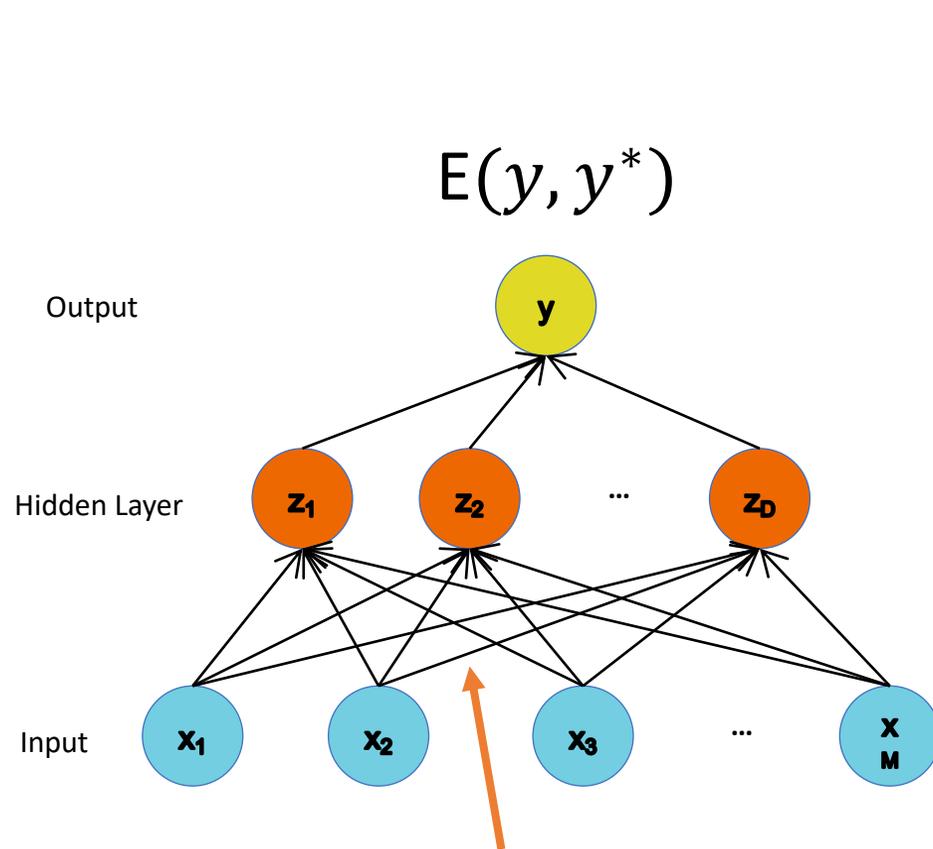
$$\text{Hidden neurons : } \delta_j = o_j (1 - o_j) \sum_{k \in \Phi_j} \delta_k w_{jk}$$



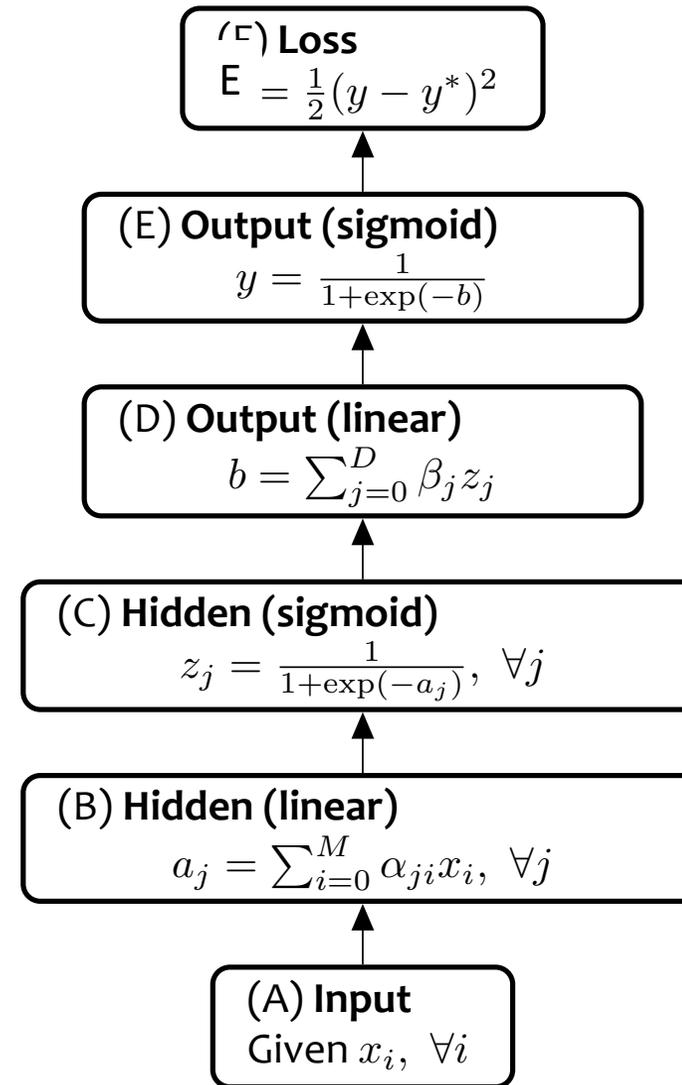
Training Multilayer NN



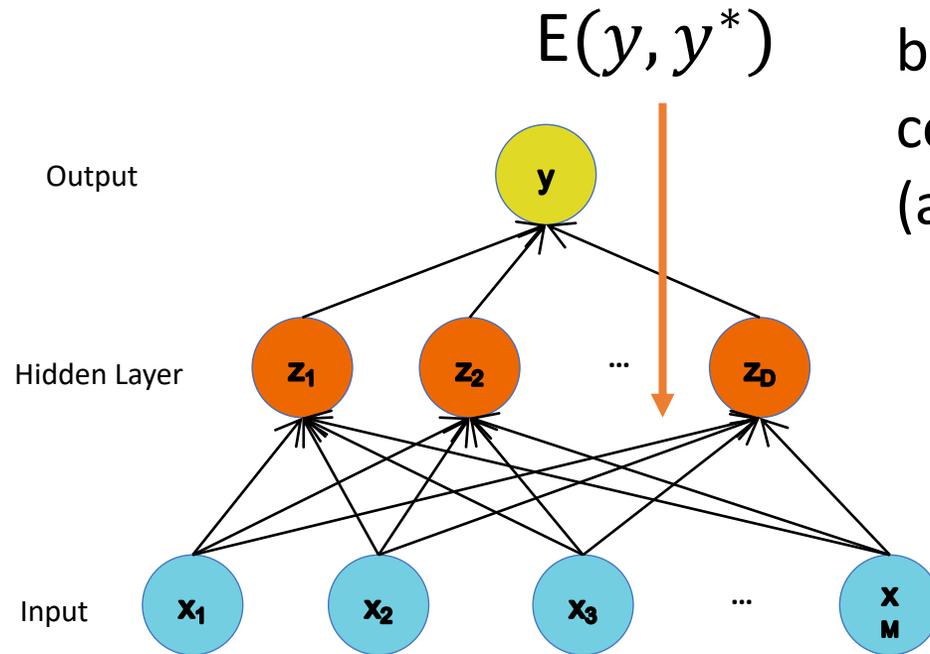
Training Multilayer NN



How do we update these weights given the loss is available only at the output unit?



Error Backpropagation



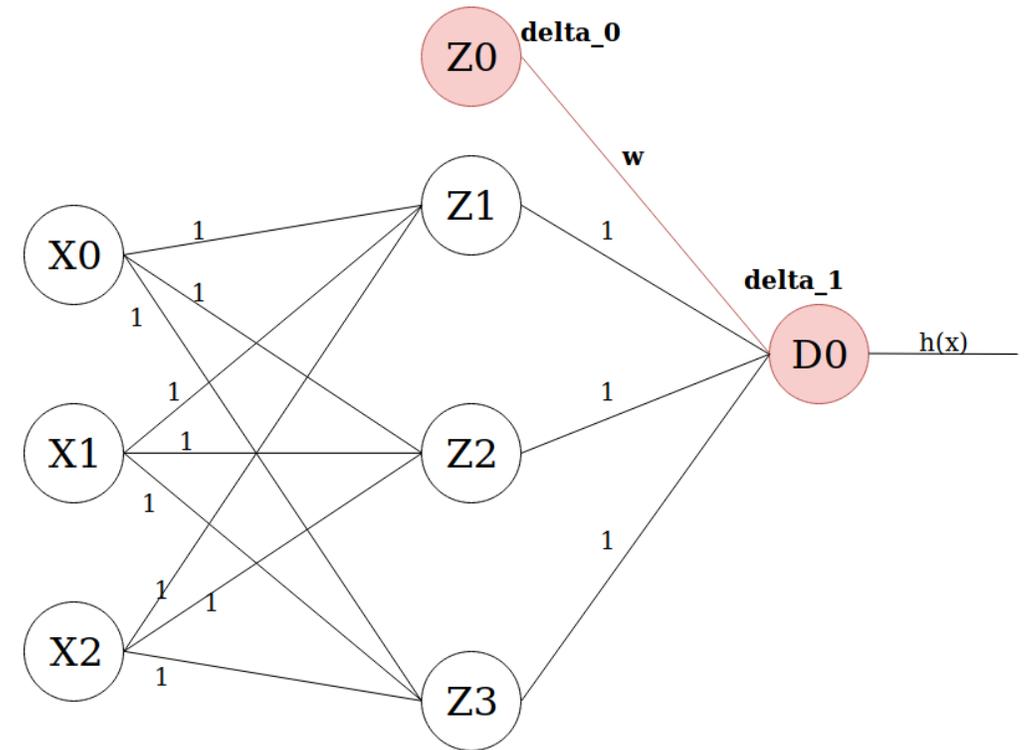
Error is computed at the output and propagated back to the input by chain rule to compute the contribution of each weight (a.k.a. derivative) to the loss

A 2-step process

1. **Forward pass** - Compute the network output
2. **Backward pass** – Compute the loss function gradients and update

Backpropagation in other words

- In order to get the loss of a node (e.g. Z0), we multiply the value of its corresponding $f'(z)$ by the loss of the node it is connected to in the next layer (delta_1), by the weight of the link connecting both nodes.
- We do the delta calculation step at every unit, back-propagating the loss into the neural net, and finding out what loss every node/unit is responsible for.



Backpropagation: a simple example

The setting:

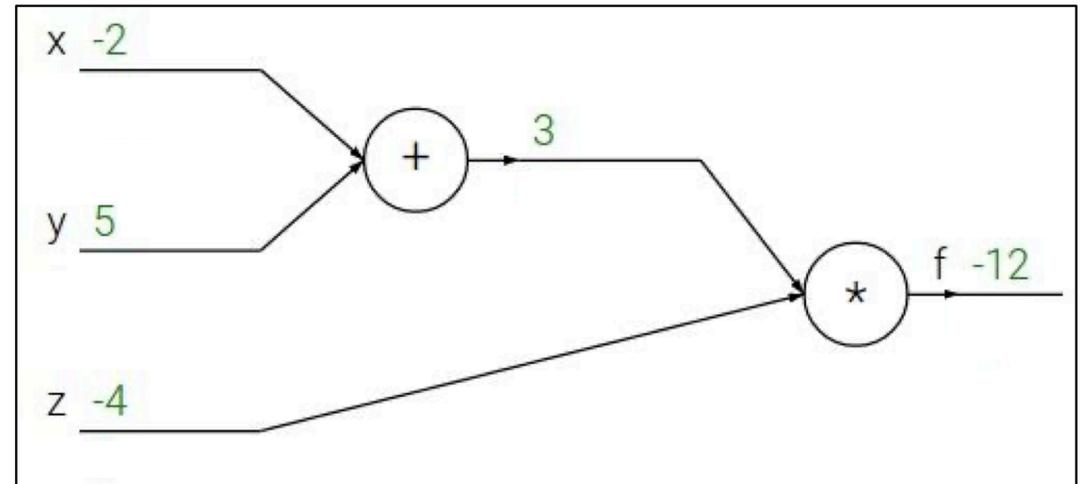
$$f(x, y, z) = (x + y)z$$

$$x = -2, \quad y = 5, \quad z = -4$$

$$q = x + y, \quad f = qz$$

For backpropagate the new weights, we need to compute:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



Backpropagation: a simple example

The setting:

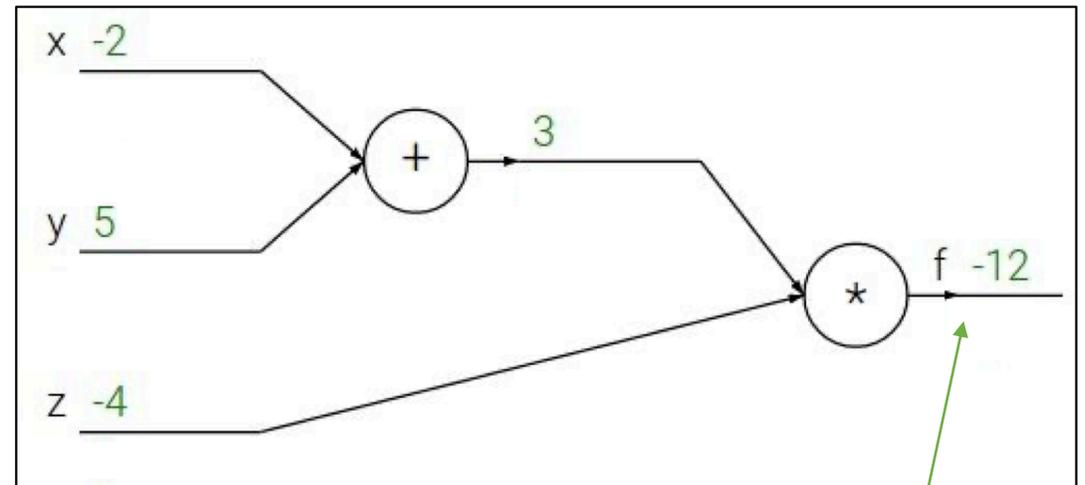
$$f(x, y, z) = (x + y)z$$

$$x = -2, \quad y = 5, \quad z = -4$$

$$q = x + y, \quad f = qz$$

For backpropagate the new weights, we need to compute:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



$$\frac{\partial f}{\partial f} = 1$$

Backpropagation: a simple example

The setting:

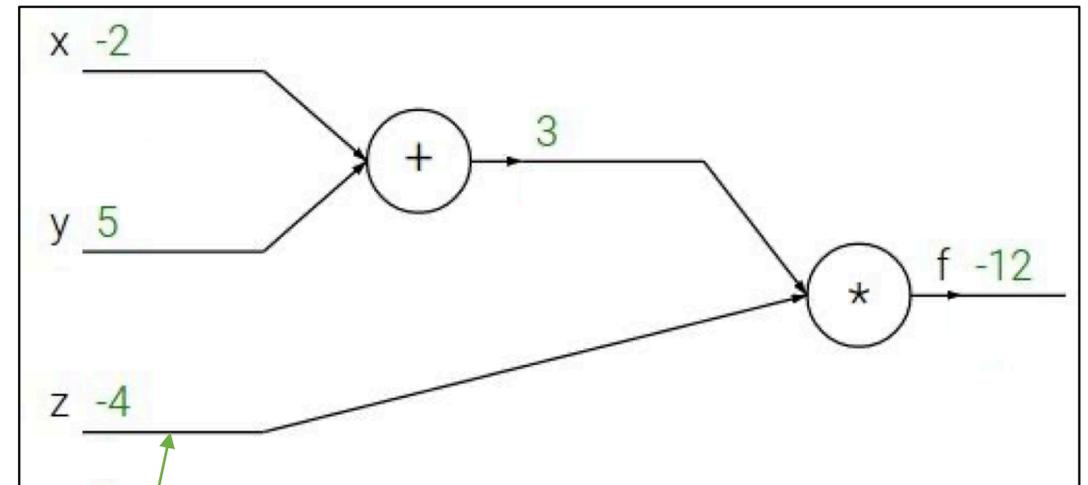
$$f(x, y, z) = (x + y)z$$

$$x = -2, \quad y = 5, \quad z = -4$$

$$q = x + y, \quad f = qz$$

For backpropagate the new weights, we need to compute:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



$$\frac{\partial f}{\partial z} = -3$$

Backpropagation: a simple example

The setting:

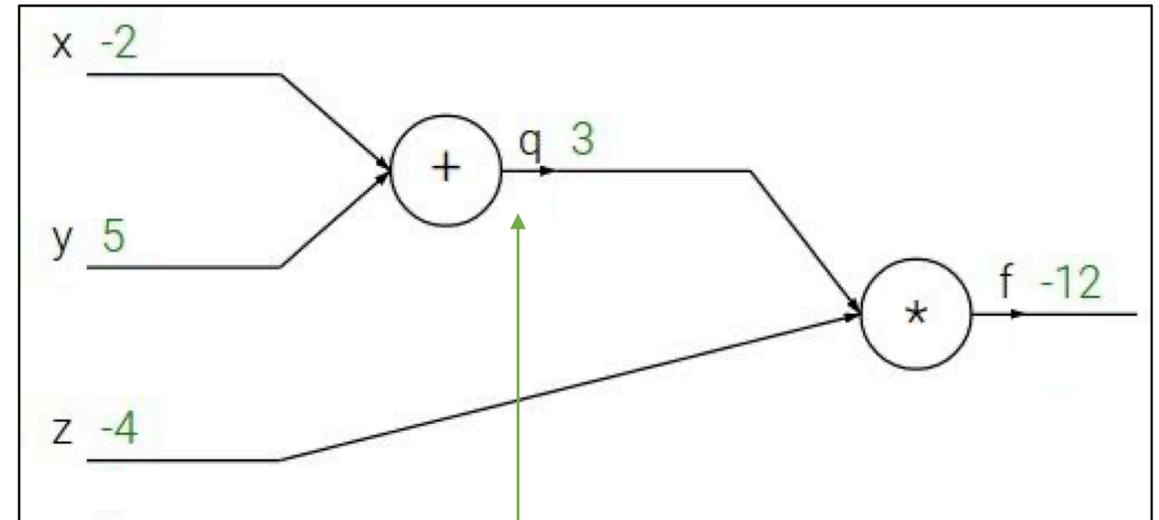
$$f(x, y, z) = (x + y)z$$

$$x = -2, \quad y = 5, \quad z = -4$$

$$q = x + y, \quad f = qz$$

For backpropagate the new weights, we need to compute:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



$$\frac{\partial f}{\partial q} = -4$$

Backpropagation: a simple example

The setting:

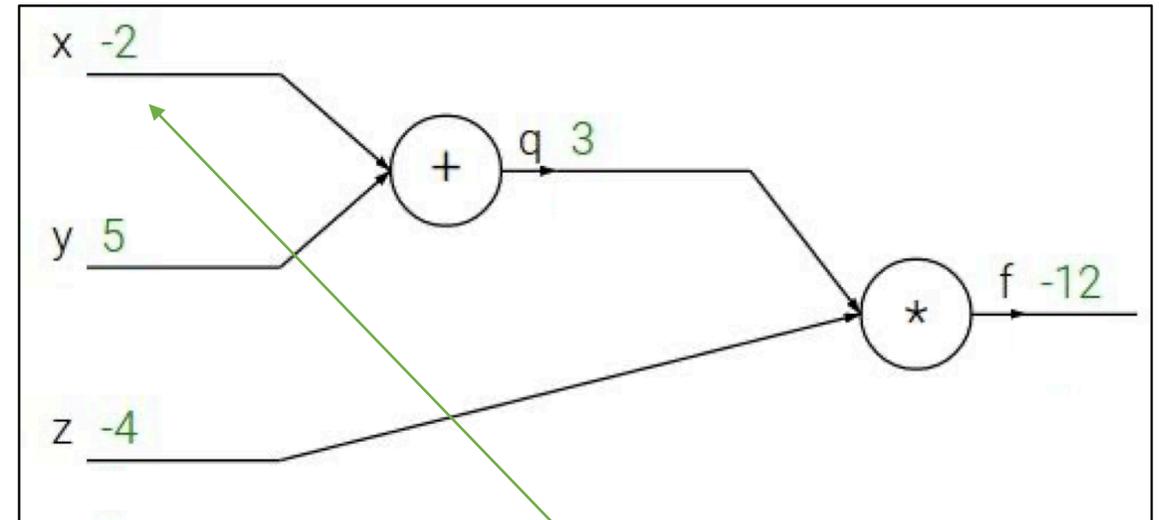
$$f(x, y, z) = (x + y)z$$

$$x = -2, \quad y = 5, \quad z = -4$$

$$q = x + y, \quad f = qz$$

For backpropagate the new weights, we need to compute:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} = -4$$

Backpropagation: a simple example

The setting:

$$f(x, y, z) = (x + y)z$$

$$x = -2, \quad y = 5, \quad z = -4$$

$$q = x + y,$$

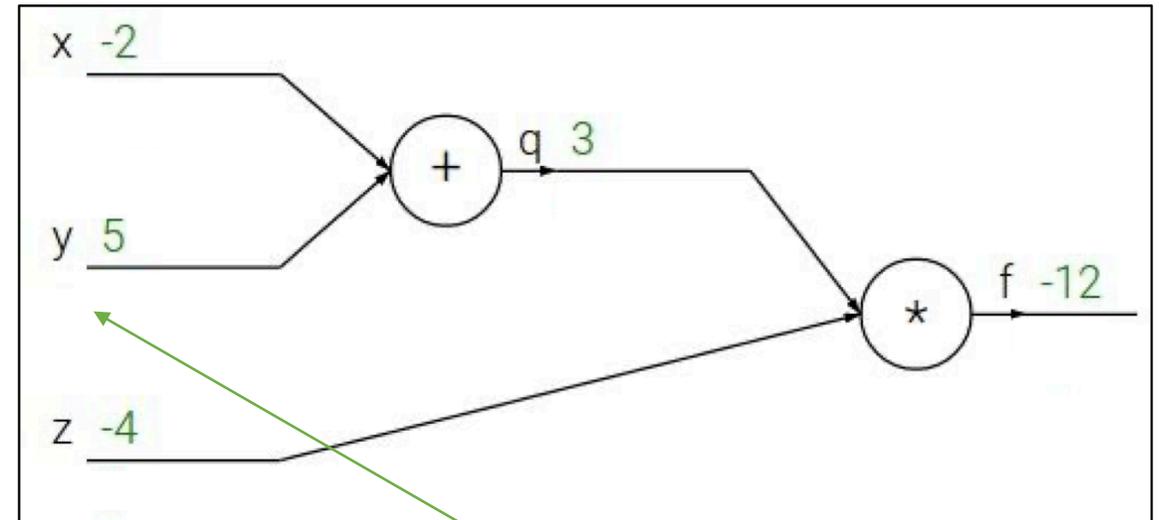
$$\text{Where } \frac{\partial q}{\partial y} = 1, \frac{\partial q}{\partial x} = 1,$$

$$f = qz,$$

$$\text{Where } \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

For backpropagate the new weights, we need to compute:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y} = -4$$

On the Key Importance of Error Functions

- The error/loss/cost function reduces all the various good and bad aspects of a possibly complex system down to a single number, a scalar value, which allows candidate solutions to be compared.
- It is important, therefore, that **the function faithfully represent our design goals.**
- If we choose a poor error function and obtain unsatisfactory results, the fault is ours for badly specifying the goal of the search.

Design Issues in ANN

- Number of nodes in input layer
 - One input node per binary/continuous attribute
 - k or $\log_2 k$ nodes for each categorical attribute with k values
- Number of nodes in output layer
 - One output for binary class problem
 - k or $\log_2 k$ nodes for k -class problem
- Number of nodes in hidden layer
- Initial weights and biases

Characteristics of ANN

- Multilayer ANN are universal approximators but could suffer from ***overfitting*** if the network is too large.
- Gradient descent may converge to ***local minimum***.
- Model building can be very time consuming, but testing can be very fast.
- Can handle redundant attributes because weights are automatically learnt.
- Sensitive to noise in training data.
- Difficult to handle missing attributes.

References

- Artificial Neural Network. Chapter 5.4 and 5.5. Introduction to Data Mining.

