UNIVERSITY OF LINCOLN

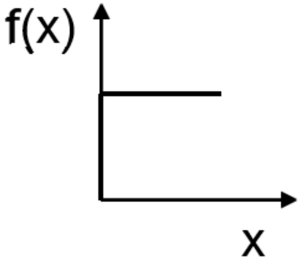
# NEURAL COMPUTING

## Lecture: Neural networks - Supervised learning

# TYPES OF ACTIVATION FUNCTIONS

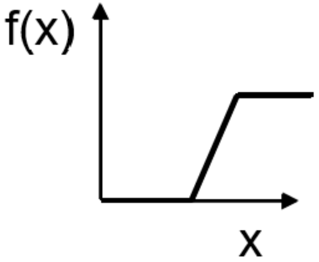
## Threshold Function

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



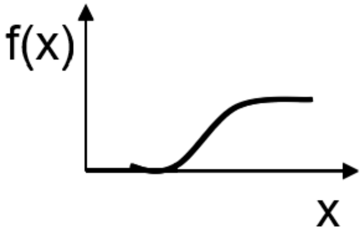

## Piecewise-Linear Function

$$f(x)=\begin{cases}1 & if\ x\geq 1.5\\ x-0.5 & if\ 0.5<x<1.5\\ 0 & if\ x\leq 0.5\end{cases}$$



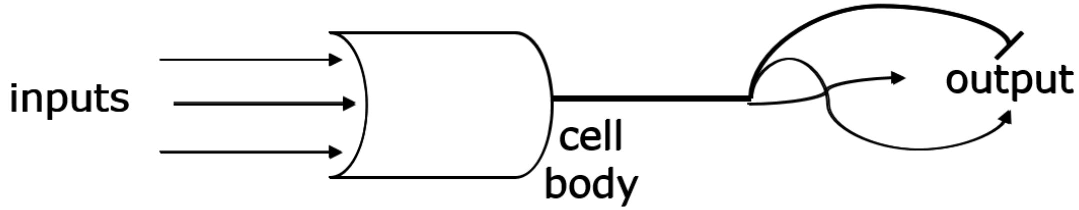

## Sigmoid Function

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



# McCulloch-Pitts Neuron

In analogy to a biological neuron, we can think of a virtual neuron that crudely mimics the biological neuron and performs analogous computation.

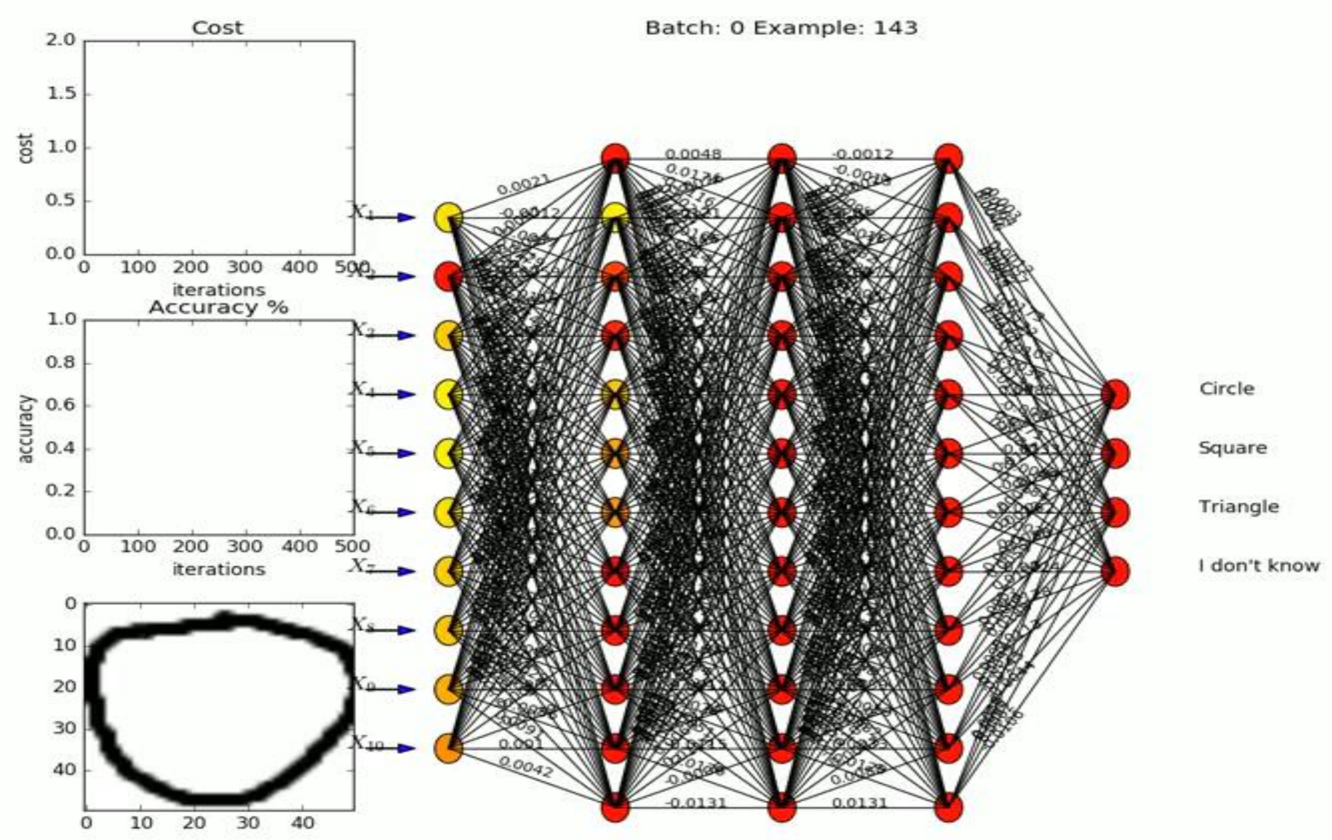



Just like biological neurons, this artificial neuron neuron will have:

Inputs (like biological dendrites) carry signal to cell body.

A body (like the soma), sums over inputs to compute output, and outputs (like synapses on the axon) transmit the output downstream

# CONNECTION WEIGHTS: ANIMATION VIDEO




**Watch the video here:** https://www.youtube.com/shorts/pKJYHt6AKvU

# LEARNING PROCESS IN ANN

- A neural network learns about its environment through an iterative process of adjustments applied to its synaptic weights and thresholds.

- Ideally, the network becomes more knowledgeable about its environment after each iteration of the learning process.

There are three broad types of learning:

1. **Supervised learning** (i.e. learning with an external teacher)
2. **Unsupervised learning** (i.e. learning with no help)
3. **Reinforcement learning** (i.e. learning with limited feedback)

# LEARNING BY ERROR MINIMIZATION

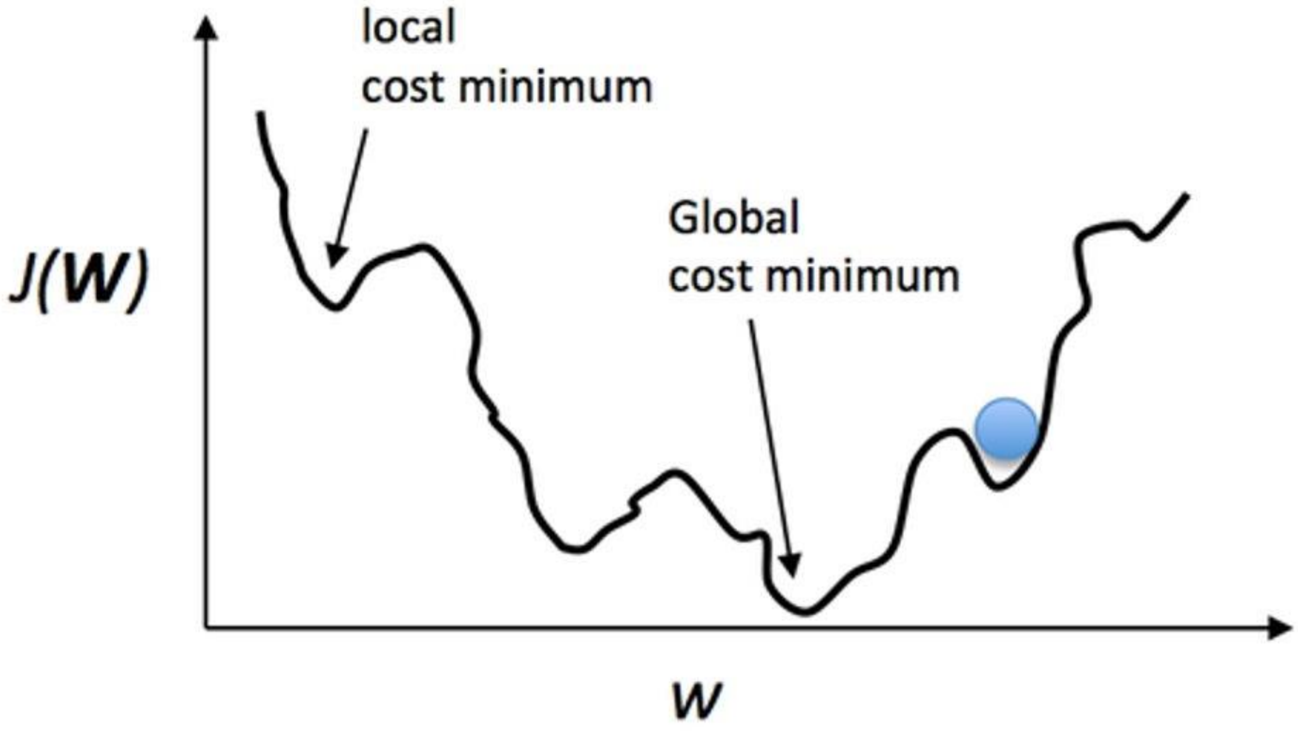
- We adjust the network weights **w** to minimize the difference between the actual and th desired outputs

- To do so, we can define a **Cost Function** to quantify this difference:

$$J(w) \sim y_{target} - y$$

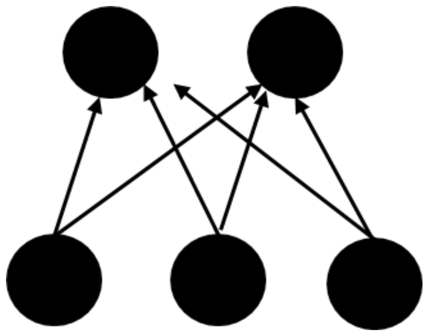
# GRADIENT DESCENT

**Gradient descent** is an optimization algorithm that approaches a local minimum of a function by taking steps proportional to the negative of the gradient of the function as the current point.



# DELTA LEARNING RULE

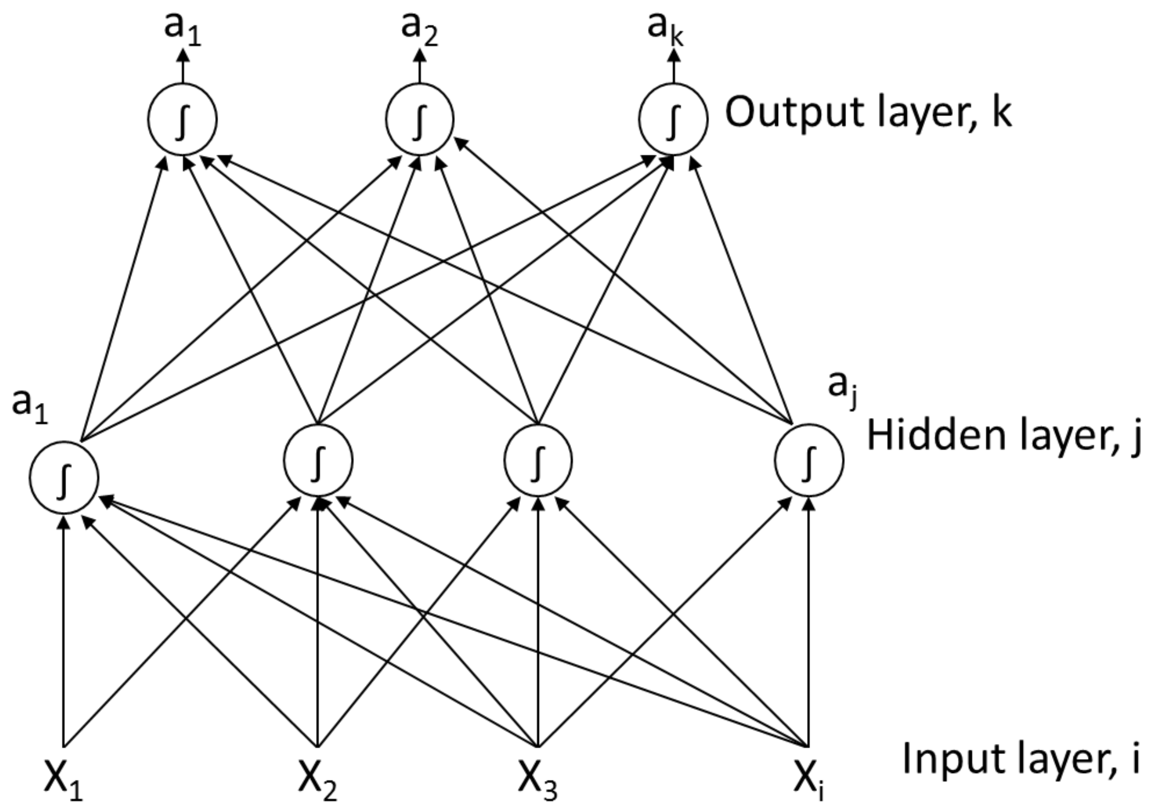
$$\boxed{w = w_{old} + \eta \delta x}$$



where

- $\delta = y_{target} - y$ is the local gradient for the neuron
- $x$ is the input to the system
- $\eta$ is a constant that controls the learning rate (amount of increment/update $\Delta$w at each training step)

# MULTI-LAYER PERCEPTRON




k: output layer
j: hidden layer
i: input layer

$w_{kj}$: weight from hidden to output layer

$w_{ji}$: weight from input to hidden layer

a: output
t: target output
net: combined input

# SIGMOID FUNCTION PROPERTIES

- Approximates the threshold function
- Smoothly differentiable everywhere
- Positive slope

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

- Derivative of sigmoidal function is:

$$f^{'}(a) = f(a) \cdot (1 - f(a))$$

# GRADIENT DESCENT ON ERROR

$$E = \frac{1}{2}\sum_k (t_k - a_k)^2$$

Total error in the network

$$\Delta W \propto -\frac{\partial E}{\partial W}$$

Adjust network weights to reduce overall error

$$\Delta w_{kj} \propto -\frac{\partial E}{\partial w_{kj}}$$

$$\Delta w_{kj} = -\varepsilon \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}}$$

via chain rule

# GRADIENT DESCENT ON ERROR

Take the set of training patterns you wish the network to learn

Set up the network with **N** input units fully connected to **M** hidden non-linear hidden units via connections with weights $w_{ij}$, which in turn are fully connected to **P** output units via connections with weights $w_{jk}$

Generate random initial weights, e.g. from range [-wt, +wt]

Select appropriate error function $J(w_{jk})$ and learning rate $\eta$

Apply the weight update equation $\Delta w_{jk}=-\eta\partial J(w_{jk})/\partial w_{jk}$ to each weight $w_{jk}$ for each training pattern p.

Do the same to all hidden layers.

Repeat step 5-6 until the network error function is 'small enough'

# Any Questions?

## GET IN TOUCH

**Email**

Ztayeb@lincoln.ac.uk

**Office In:**

Room: INB3203 - Office Hours Tuesday: 16h00-18h00

UNIVERSITY OF LINCOLN