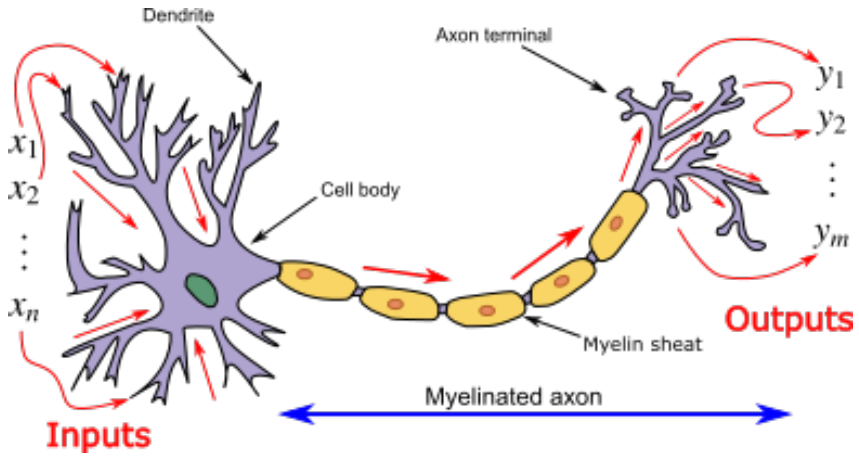
Neural Networks

Oana Cocarascu & Helen Yannakoudakis

Department of Informatics
King's College London

Artificial Neural Networks

- Inspired by biological neurons and their connectionist nature.



- Synapses (junctions that allow signal transmission between the axons and the dendrites) are excitatory or inhibitory and may change over time.
- When inputs reach a threshold, an action potential (electrical pulse) is sent along the axon to the outputs.
- Human brain: around 10^{11} neurons, 10^{15} synapses

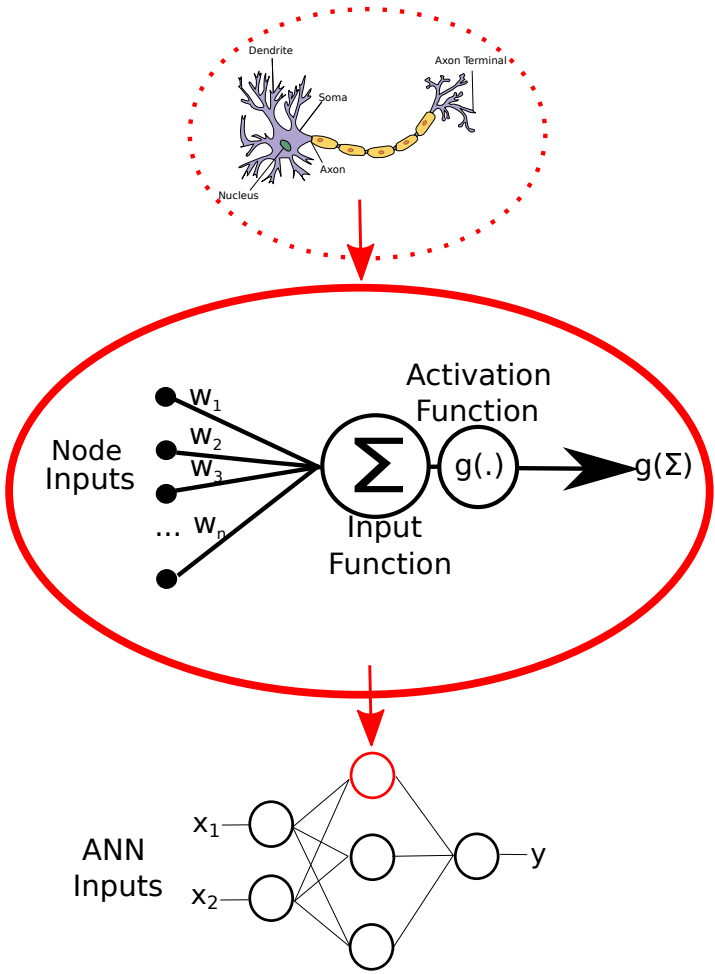
  $\rightarrow$ a cycle time of 10^{-3} seconds

KING'S College LONDON

Artificial Neural Networks

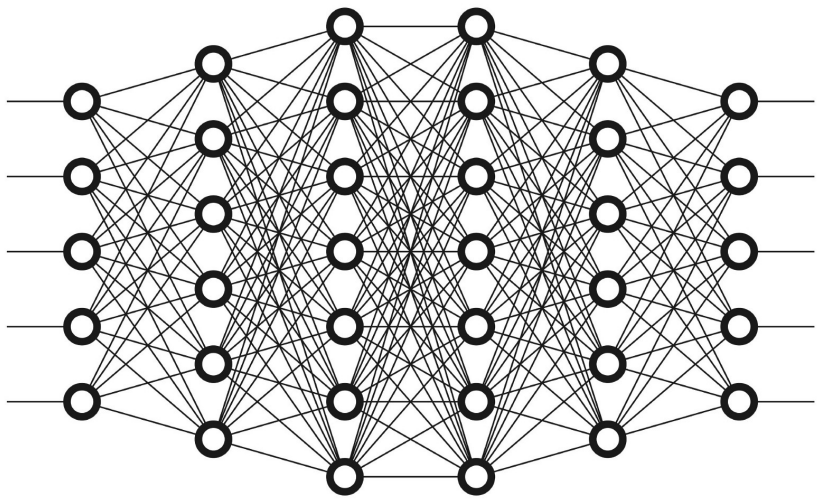
- The building block of an ANN is a neuron.
- ANNs are made up of nodes which have:
  - input edges, each with some weight
  - output edges (with weights)
  - an activation level (function of inputs)
- Weights can be positive or negative and may change over time (learning).
- Input function is the weighted sum of the activation levels of inputs.
- The activation level is a non-linear transfer function g of this input:

$$\text{activation}_j = g(s_j) = g(\sum_{i=1}^{N} w_{i,j} x_i)$$



Artificial Neural Networks

- Artificial Neural Network (ANN): a mathematical function that can produce approximations of real values, discrete values (integer or ordinal/categorical) or vectors of values.
- Built out of a densely interconnected set of units where each unit takes a number of real-valued inputs (possibly the outputs of other units) and produces a single real-valued output (which may become the input to other units).

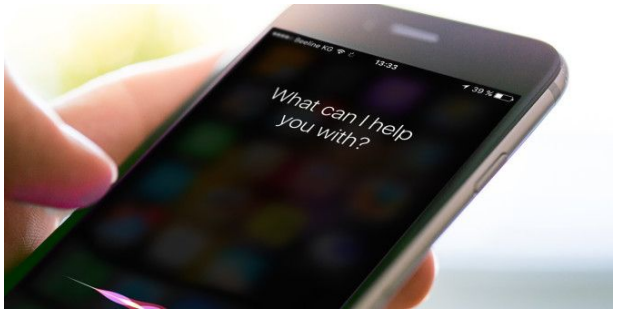


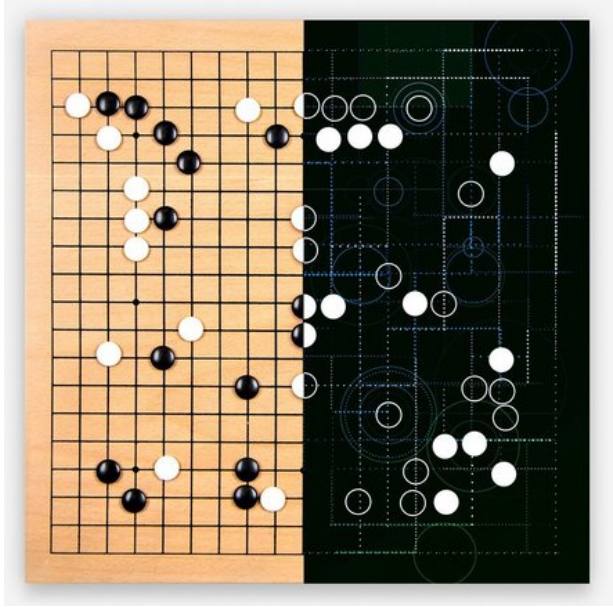
Artificial Neural Networks

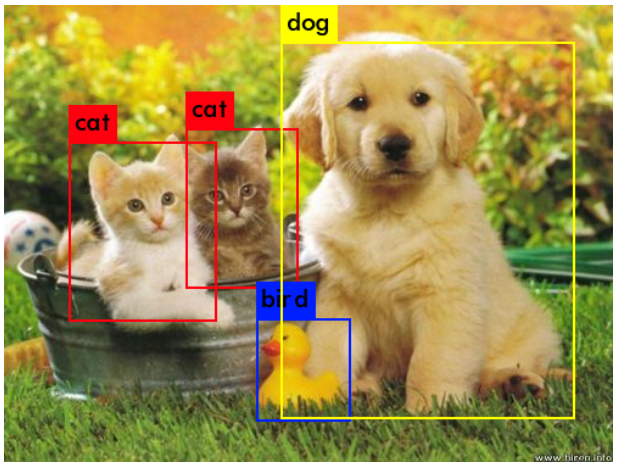
- Whilst motivated by biological neurons, there are many complexities that are not modeled by ANNs (many are inconsistent with biological neurons):
  → ANNs units output a single value
  → Biological neurons output a complex time series of spikes
- Two directions in NN research historically:
  → Trying to simulate how the (human) brain works: aim to model the process inside the human brain
  → Producing machine learning methodologies: aim to model the outcome, without trying to mimic the process

KING'S College LONDON

Artificial Neural Networks - Applications





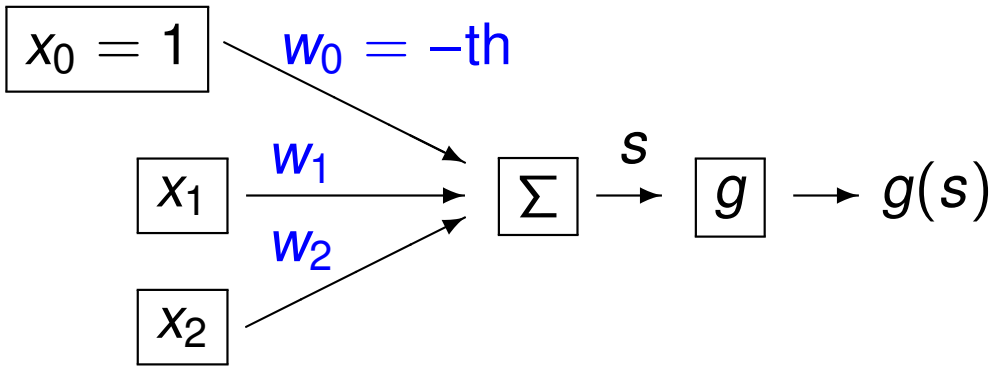


Perceptron

- The simplest type of ANN.
- Consists of a single neuron.
- The input of the neuron is the weighted sum of the inputs plus the bias term and the output is some function of the input

  $\rightarrow$ e.g. 1 if the result is greater than some threshold and -1 otherwise
- A perceptron can compute linearly separable functions.

Perceptron



- x_1, x_2 are inputs; x_0 is 1
- w_1, w_2 are synaptic weights; w_0 is a **bias** weight
- *th* is a threshold
- s is the dot product of the input (**x**) and weight (**w**) vectors, plus the bias (w_0):

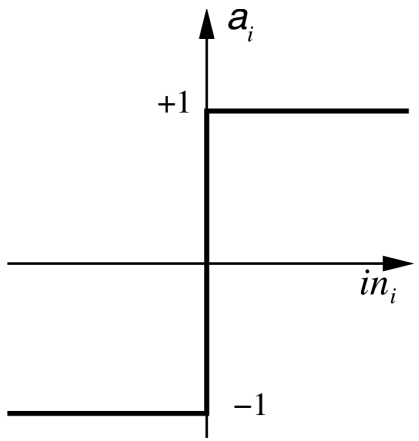
$$\begin{aligned} s &= w_0 x_0 + w_1 x_1 + w_2 x_2 \\ &= -\text{th} + w_1 x_1 + w_2 x_2 \end{aligned}$$

- g is a transfer/threshold/activation function applied to s

Transfer/threshold functions

- **Sign** function:

$$g(s) = \begin{cases} 1, & s > 0 \\ -1, & s < 0 \end{cases}$$



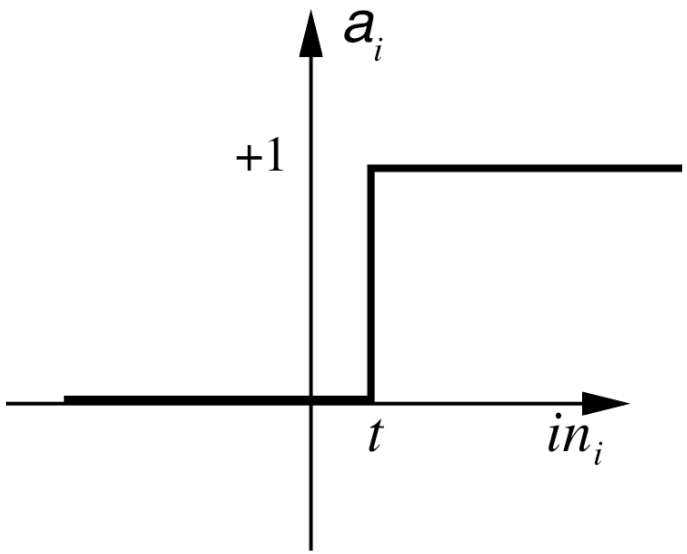
(also at threshold rather than at 0)

  - The output is either 1 or -1 depending on the input.
  - It can be used for classification:
    - If the total input is positive, the sample is assigned to class 1.
    - If the total input is negative, the sample is assigned to class -1.

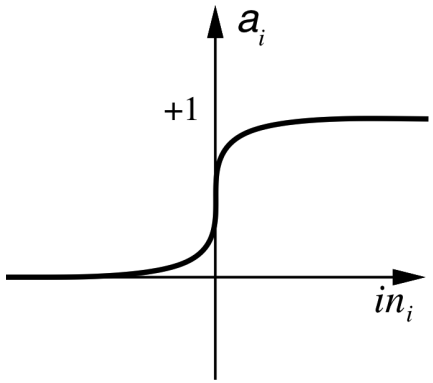
Transfer/threshold functions

- **Step** function:

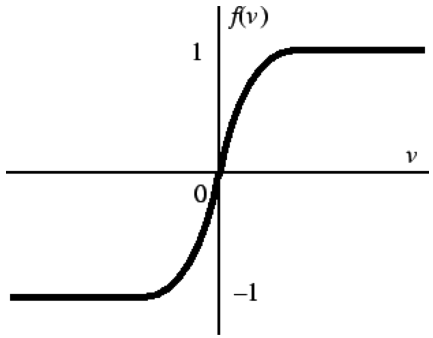
$$g(s) = \begin{cases} 1, & s \geq t \\ 0, & s < t \end{cases}$$



Transfer/threshold functions



(a) sigmoid



(b) tanh

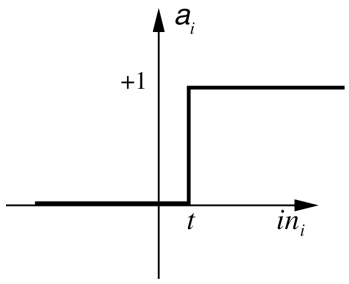
- **Sigmoid** function:

$$g(s) = 1/(1 + e^{-s})$$

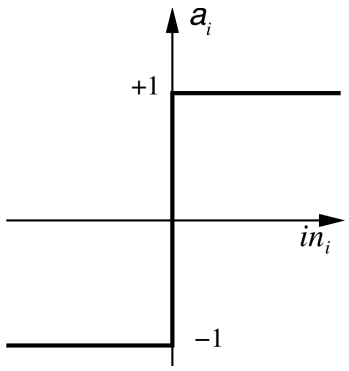
- **Hyperbolic tangent** (tanh) function:

$$\begin{aligned} g(s) &= \tanh(s) \\ &= (e^{s} - e^{-s})/(e^{s} + e^{-s}) \\ &= 2\Big(1/(1 + e^{-2s})\Big) - 1 \end{aligned}$$

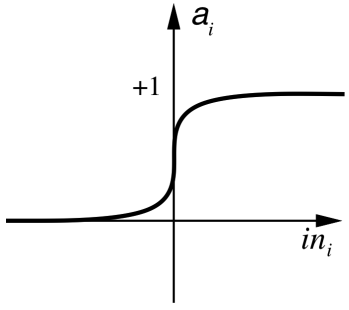
Perceptron



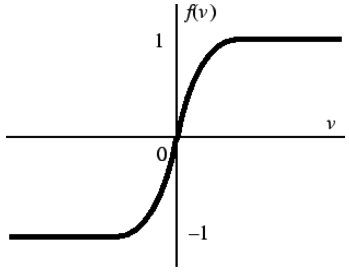
(a) step



(b) sign



(a) sigmoid



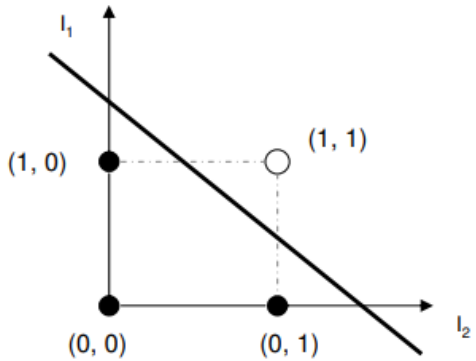
(b) tanh

- Perceptrons are a family of related but different methods.

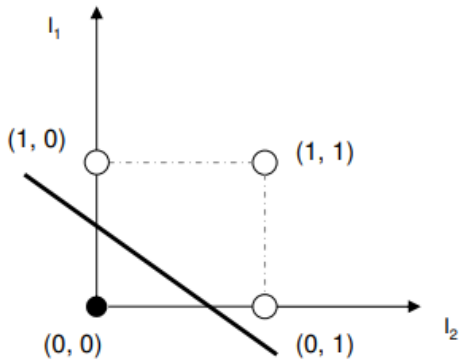
Perceptron

- A single perceptron can represent many boolean functions.
- A perceptron can compute linearly separable functions.

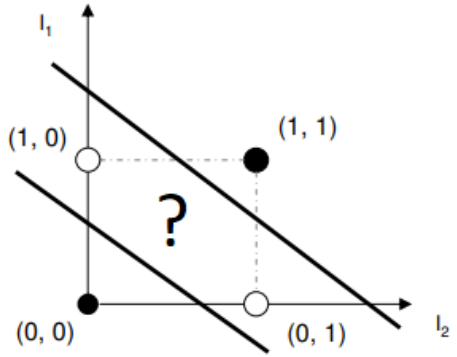
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0



Perceptron

- A perceptron can compute linearly separable functions.
- Using the step-function, we can have, for example:

AND $\qquad w_1 = w_2 = 1.0, \quad w_0 = -1.5$

I_1	I_2	s	$g(s)$
0	0	$-1.5 \times 1 + 1 \times 0 + 1 \times 0 = -1.5$	0
0	1	$-1.5 \times 1 + 1 \times 0 + 1 \times 1 = -0.5$	0
1	0	$-1.5 \times 1 + 1 \times 1 + 1 \times 0 = -0.5$	0
1	1	$-1.5 \times 1 + 1 \times 1 + 1 \times 1 = 0.5$	1

Perceptron

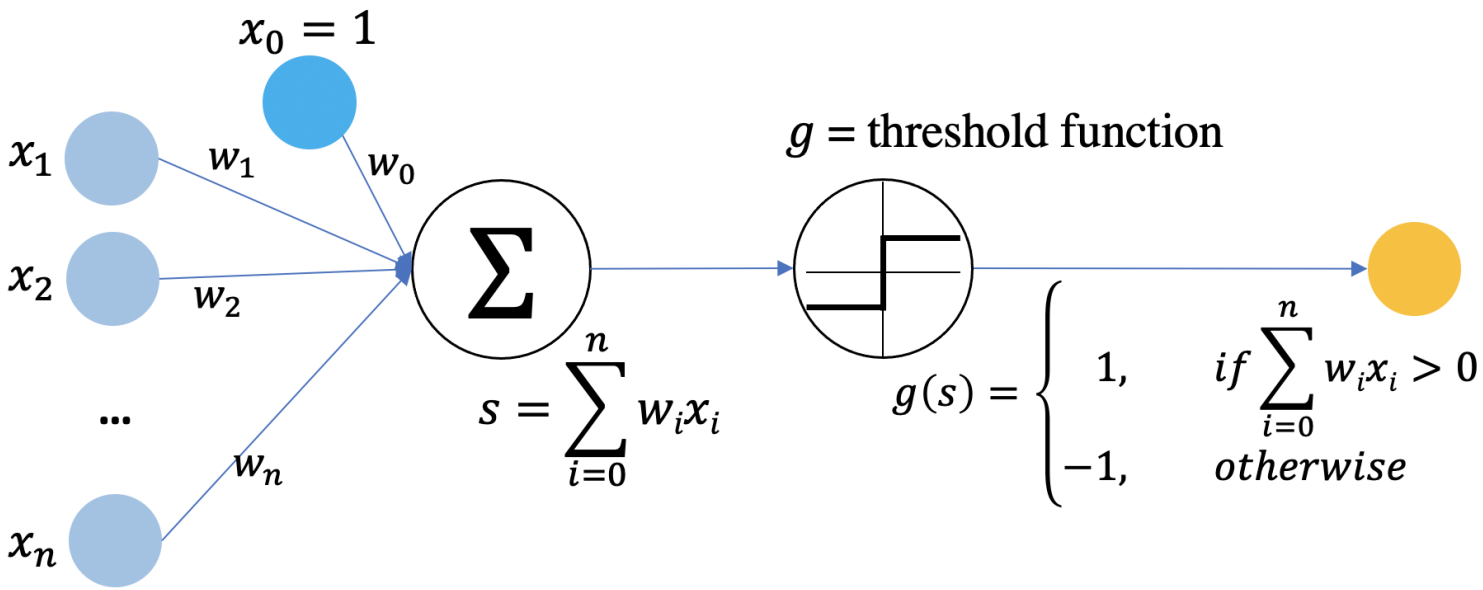
- A perceptron can compute linearly separable functions.
- Using the step-function, we can have, for example:

OR $\qquad w_1 = w_2 = 1.0, \quad w_0 = -0.5$

I_1	I_2	s	$g(s)$
0	0	$-0.5 \times 1 + 1 \times 0 + 1 \times 0 = -0.5$	0
0	1	$-0.5 \times 1 + 1 \times 0 + 1 \times 1 = 0.5$	1
1	0	$-0.5 \times 1 + 1 \times 1 + 1 \times 0 = 0.5$	1
1	1	$-0.5 \times 1 + 1 \times 1 + 1 \times 1 = 1.5$	1

Single-layer networks

- A perceptron is a *single-layer* neural network (i.e. one input layer and one output layer).



- Equivalent to the unit on slide 8.

Single-layer networks

- “Perceptron” represents a single unit.
- However, there can be multiple perceptrons in a single layer network (Multi-unit perceptron).
- Individual units are trained independently.
- We will consider (single unit) perceptrons.

Single-layer networks

- Two class classifier: 1 or -1.
- If the data is linearly separable, then the error correction method will find the linear boundary between classes.
- For n inputs, the boundary is a hyperplane:

$$w_0 x_0 + w_1 x_1 + \ldots + w_n x_n = 0$$

$$\sum_{i=0}^{n} w_i x_i = 0$$

$$\mathbf{w} \cdot \mathbf{x} = 0$$

  where $x_0 = 1$

- The idea is to find the weights $w_0 \ldots w_n$ so that the members of one class all have value 1 and the members of the other class all have value -1.

Error correction method

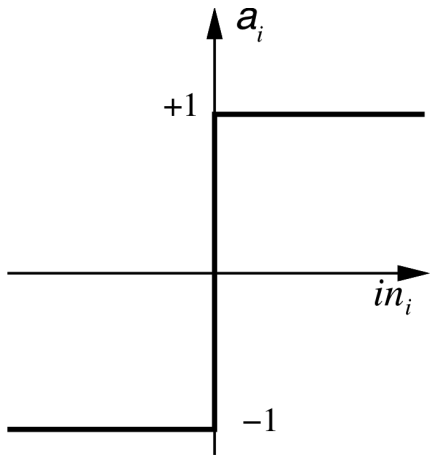
- We train a perceptron by adjusting its weights:
  - begin with random weights
  - repeat until stopping condition:
    for each training example, update each weight by

$$w_i \leftarrow w_i + \Delta w_i$$
$$\Delta w_i = \alpha(t - \mathbf{g}(s))x_i$$

  where:
  - α is the learning rate
  - t is the target output for the network
  - $g(s)$ is the prediction produced by the neural network
- $g(s)$ and s are defined on slide 16.
- The learning rate moderates the degree to which weights are changed at each step.

Error correction method

- By training to ± 1, we are implicitly using the sign function:



- The bias weight w_0 sets the threshold at which the sign-function is applied.
- Without w_0 in the model, the threshold is 0.
- With w_0, the threshold is $-w_0$.

Error correction method - Intuition

$$w_i \leftarrow w_i + \alpha(t - \mathbf{g}(s))x_i$$

- If $t = \mathbf{g}(s)$, no update needed.
- If $t > \mathbf{g}(s)$ (i.e. perceptron outputs -1 when the target is 1)
  - $\rightarrow$ the weights must be altered to increase the value of $\mathbf{w} \cdot \mathbf{x}$
  - $\rightarrow$ if $x_i > 0$, then increase w_i (otherwise, decrease)
- If $t < \mathbf{g}(s)$ (i.e. perceptron outputs 1 when the target is -1)
  - $\rightarrow$ decrease weights associated with positive x_i

KING'S College LONDON

Error correction method - Issues

- Convergence theorem for the perceptron learning rule

  $\rightarrow$ If the boundary between classes is linear, and α is small enough, the error correction method will find it.
  $\rightarrow$ The method will converge within a finite number of steps and it will classify all training examples correctly.
- If the boundary between classes is non-linear, the error correction method can fail to find (an approximation to) it because no linear hyperplane can separate the data perfectly.
- The delta rule can overcome this, by producing a boundary that approximates the non-linear decision boundary.
- This is gradient descent once again.

Delta rule

- Consider an unthresholded perceptron (i.e. a linear unit)
- Use an error function E to show how far the output is from our desired output:

$$\begin{aligned} error &= E(\mathbf{w}) \\ &= \frac{1}{2}\sum_{j=1}^{M}(t_j - s_j)^2 \end{aligned}$$

where:

  - M is the number of instances in our dataset
  - t_j is the target evaluated for the j-th element
  - s_j is the prediction evaluated for the j-th instance

- The goal is to minimise this error.

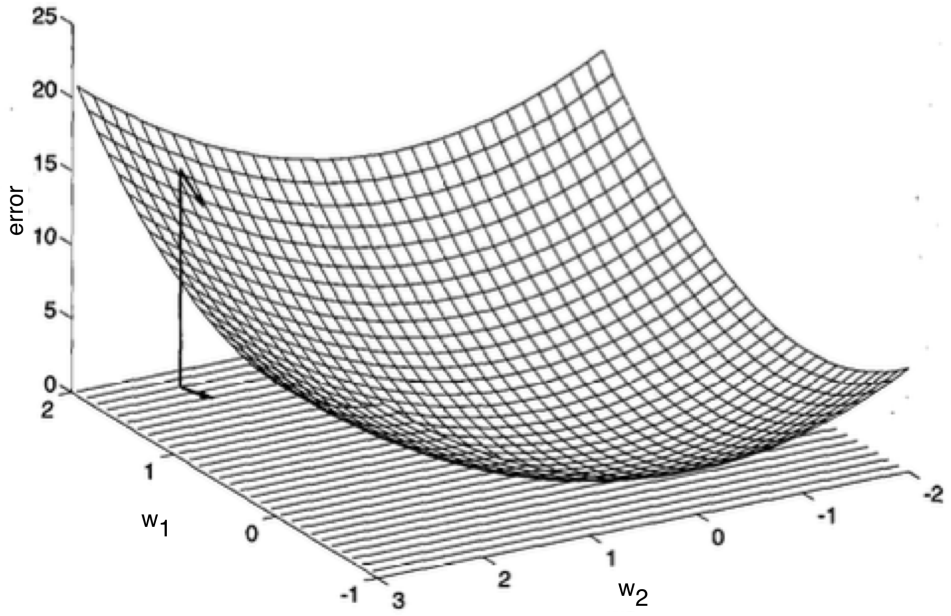
Delta rule

$$E(\mathbf{w}) = \frac{1}{2}\sum_{j=1}^{M}(t_j - s_j)^2$$

- Note that the error is defined in terms of s not $g(s)$.
- We are not considering the transfer/threshold function.

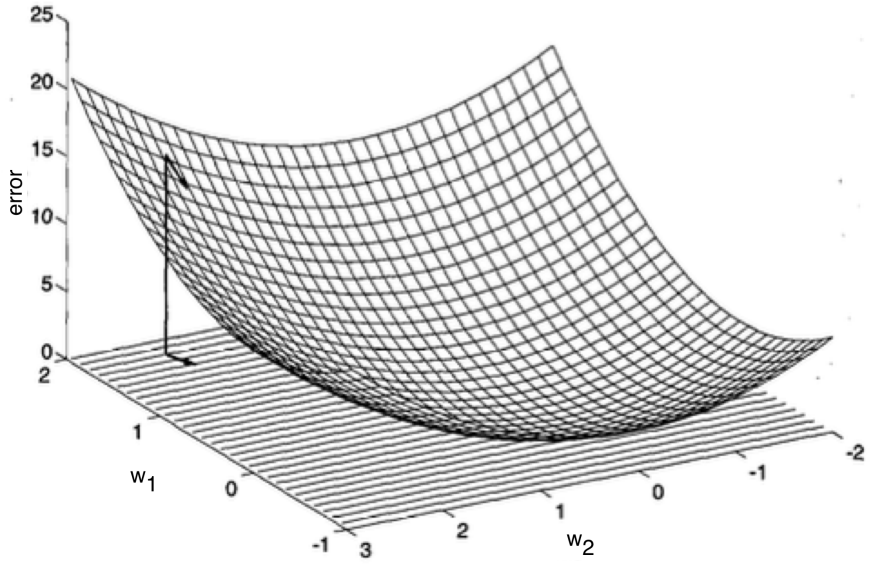
  $\rightarrow$ We are (effectively) training the unit without the transfer function by looking at the output of the weighted sum.
- Linear regression.
- We use gradient descent to find the weight values that minimise the error function.

Delta rule



- This illustrates the solution space or landscape for a simple perceptron with two weights $\{w_1, w_2\}$.
- With gradient descent, the idea is to move down the landscape, into the valley where the error is small.

Delta rule



- The gradient points in the direction of greatest increase of the function at each point.
- The arrow shows the negated gradient at one particular point.

  → This is the direction in the plane producing steepest descent along the error surface.

Delta rule

- The gradient is computed by taking the derivative of E with respect to each of the n elements in the weights vector $\mathbf{w}$:

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{j=1}^{M}(t_j - s_j)^2 \\
&= \frac{1}{2}\sum_{j=1}^{M}\frac{\partial}{\partial w_i}(t_j - s_j)^2 \\
&= \frac{1}{2}\sum_{j=1}^{M}2(t_j - s_j)\frac{\partial}{\partial w_i}(t_j - s_j) \\
&= \sum_{j=1}^{M}(t_j - s_j)\frac{\partial}{\partial w_i}(t_j - \sum_{i=0}^{n} w_i x_{i,j}) \\
&= \sum_{j=1}^{M}(t_j - s_j)(-x_{i,j})
\end{aligned}$$

Delta rule

- The rule for updating each of the weights is:

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = -\alpha \frac{\partial E}{\partial w_i}$$

- Since the gradient gives the direction of the steepest increase in E, the negative of this vector gives the direction of steepest decrease. Thus the update rule is:

$$w_i \leftarrow w_i + \alpha \sum_{j=1}^{M} (t_j - s_j) x_{i,j}$$

- The update rule is *exactly* what we used for linear regression.
- Don't forget w_0.

Batch and stochastic gradient descent

- What we have just described is batch gradient descent.
  - Sum errors over all training examples before adjusting weights.
- Presenting all training examples once is called an epoch.
- We can also train by stochastic gradient descent.
  - Adjust weights at each example:

$$w_i \leftarrow w_i + \alpha(t - s)x_i$$

- Same issues with stochastic gradient descent as before — may not converge.

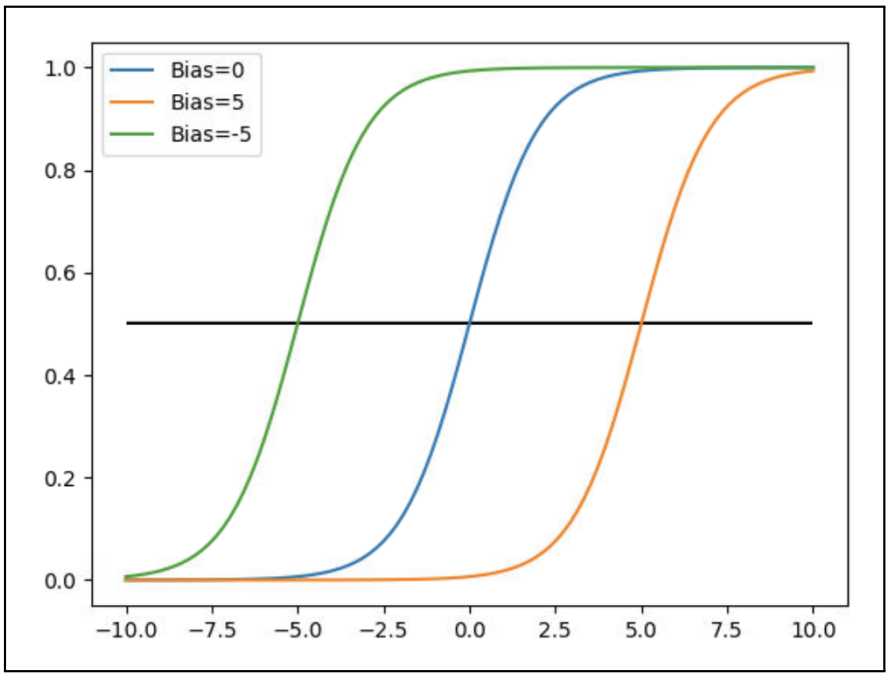
KING'S College LONDON

Stochastic gradient descent pseudocode

initialise each w_i to some small random value
loop until convergence
    for each datapoint (x, t) compute output s
        for each weight w_i
            $w_i \leftarrow w_i + \alpha(t - s)x_i$

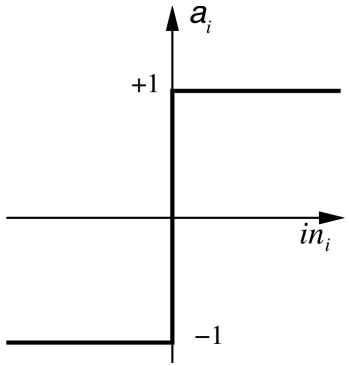
After learning

- With the delta rule, we learn with no transfer function.
- However, we add the transfer function back in afterwards.
- That means we move from a regression to a classification.
- This gives a sharp(ish) boundary between classes, depending on the transfer function used.
- The *bias weight*, w_0, sets the midpoint of the transfer function.

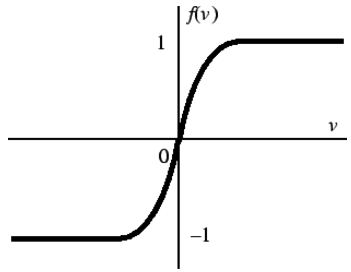


After learning

- When classes are 1 and -1.

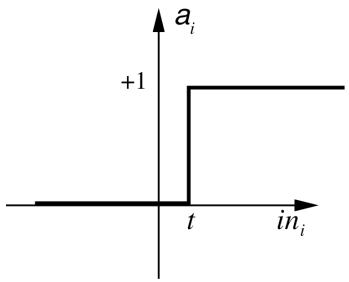


(a) sign

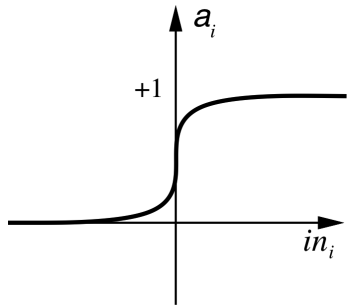


(d) tanh

- When classes are 0 and 1.



(a) step



(d) sigmoid

Delta rule

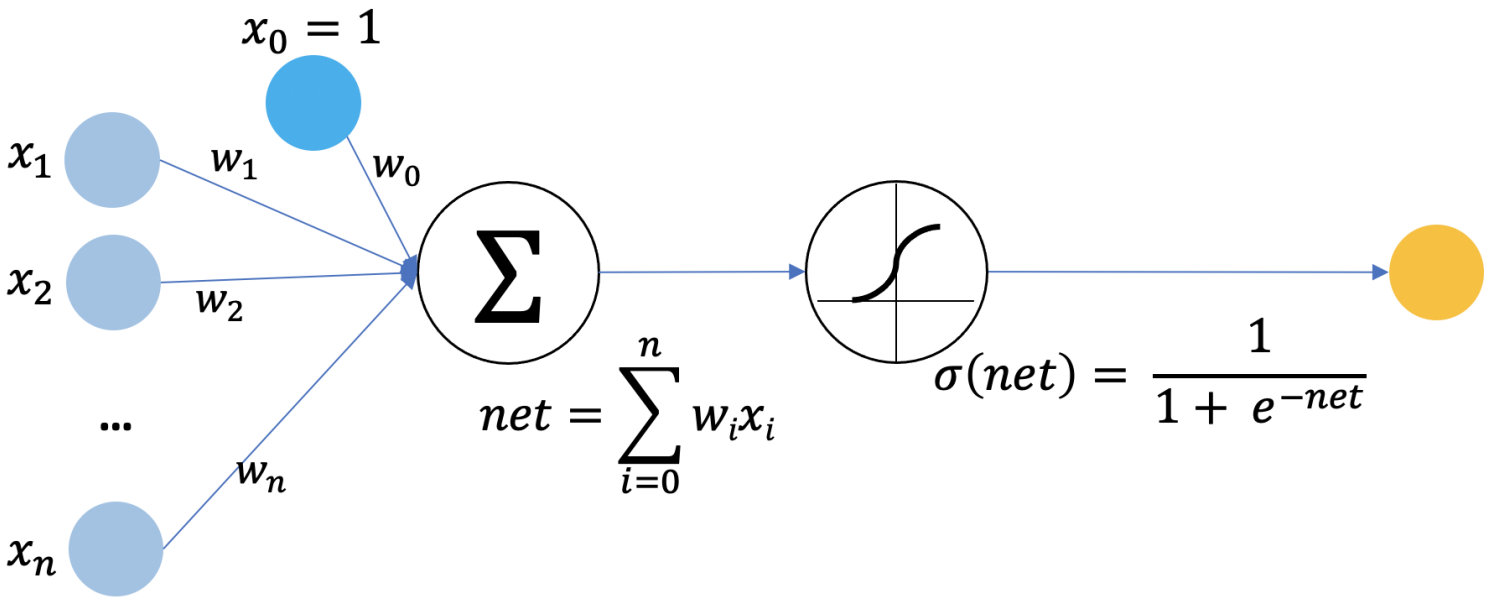
- Modulo the issues with the issues with stochastic gradient descent (see lecture 2), the delta rule will find a hyperplane that minimises the error to a non-linear boundary.
- Error function has a global minimum.

Generalised delta rule

- The delta rule updates weights based on the error in the unthresholded linear combination of inputs.
- Generalised delta rule keeps the transfer function.
- BUT the transfer function has to be differentiable.
- Examples include: sigmoid, tanh.

Generalised delta rule

- Perceptron with sigmoid transfer function.



Generalised delta rule

- Using chain rule:

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial s}\frac{\partial s}{\partial w_i} = \frac{\partial E}{\partial s}\frac{\partial}{\partial w_i}\left(\sum_{i=0}^{n} w_i x_i\right) = \frac{\partial E}{\partial s} x_i$$

$$\frac{\partial E}{\partial s} = \frac{\partial E}{\partial g(s)}\frac{\partial g(s)}{\partial s}$$

$$\begin{aligned}
\frac{\partial E}{\partial g(s)} &= \frac{\partial}{\partial g(s)}\frac{1}{2}(t - g(s))^2 \\
&= \frac{1}{2}2(t - g(s))\frac{\partial}{\partial g(s)}(t - g(s)) \\
&= -(t - g(s))
\end{aligned}$$

Generalised delta rule

- Putting it all together:

$$\frac{\partial E}{\partial w_i} = -(t - g(s))\frac{\partial g(s)}{\partial s}x_i$$

- The sigmoid function is:

$$g(s) = \frac{1}{1 + e^{-s}}$$

- With this function, we have the partial derivative:

$$\frac{\partial g(s)}{\partial s} = g(s)(1 - g(s))$$

- Thus we have:

$$\frac{\partial E}{\partial w_i} = -(t - g(s))g(s)(1 - g(s))x_i$$

KING'S College LONDON

Generalised delta rule

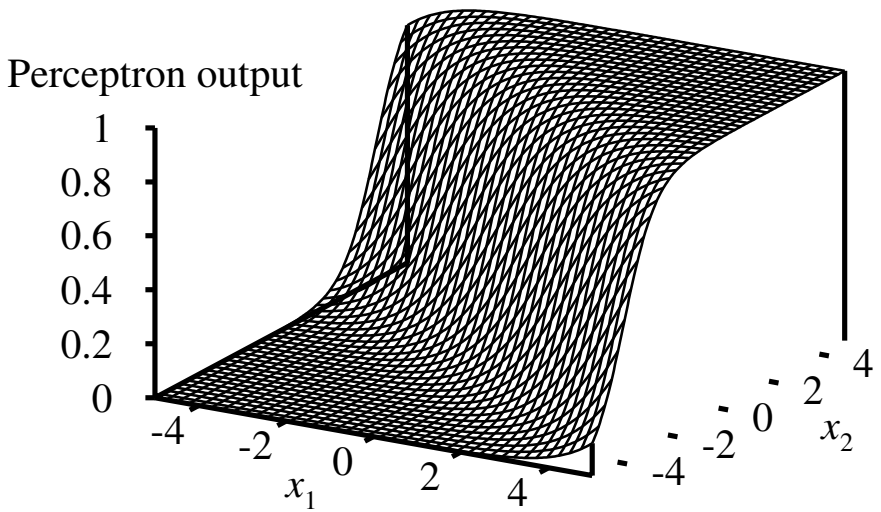
- This gives us another rule for changing weights:

$$w_i \leftarrow w_i + \alpha(t - g(s))g(s)(1 - g(s))x_i$$

- With the sigmoid, $g(s)(1 - g(s))$ varies in value from 0 to 1.
- It has value 0 when $g(s)$ is 0 or 1.
- It has maximum value of 0.25 when $g(s)$ has value 0.5 (and the input to the sigmoid is 0).

Generalised delta rule

- Boundary for a 2-input perceptron with sigmoid transfer function:



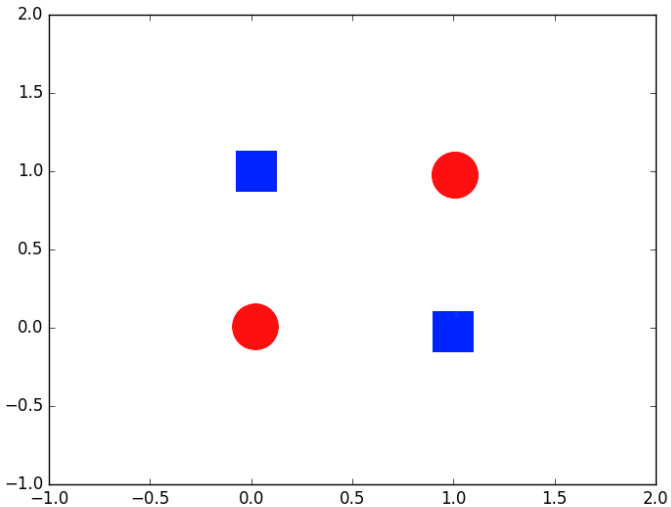
(Russell & Norvig)

Single-layer networks

- Three methods for training single layer networks:
  - Error correction method
  - Delta rule
  - Generalised delta rule
- We have only looked at training single perceptrons, but we can use the same methods to train networks with a single layer and multiple neurons.
- Treat each element independently.

Limitations of single-layer networks

- Problem: many functions are not linearly separable
  - e.g., XOR



Limitations of single-layer networks

- x_1 XOR x_2 can be written as:
  (x_1 AND x_2) NOR (x_1 NOR x_2)

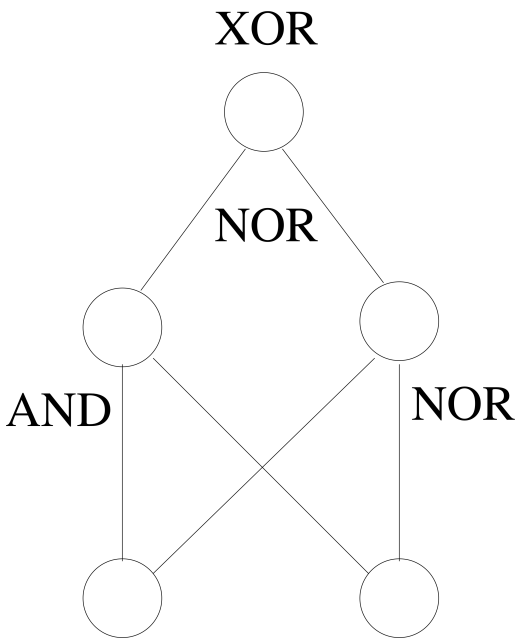
I_1	I_2	a(x_1 AND x_2)	b(x_1 NOR x_2)	a NOR b
0	0	0	1	0
0	1	0	0	1
1	0	0	0	1
1	1	1	0	0

Limitations of single-layer networks

- Possible solution: a multi-layer network

  x_1 XOR x_2 can be written as:
  (x_1 AND x_2) NOR (x_1 NOR x_2)

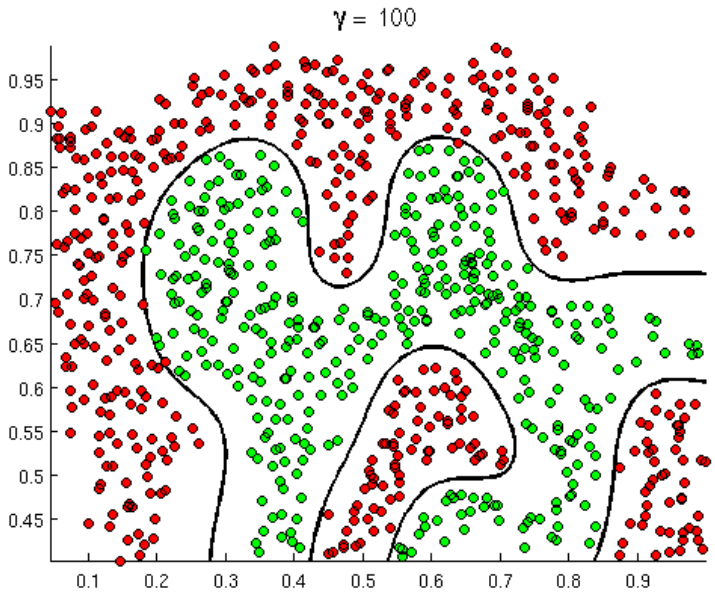
- AND, OR and NOR can be implemented by perceptrons



- Problem: How do we train it to learn a new function?

Multi-layer networks

- A multi-layer perceptron is usually known as a(n) “(artificial) neural network” (i.e. one input layer, one output layer and one or more hidden layers of processing units).
- We are interested in networks capable of representing highly non-linear functions.



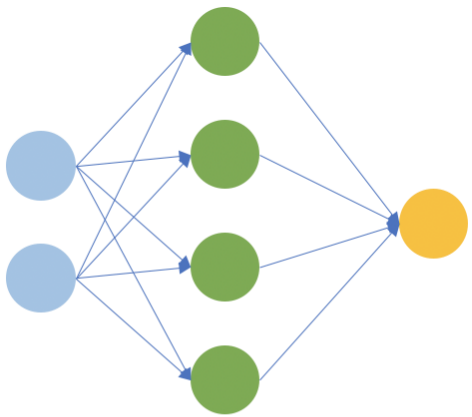
- Unit for constructing multi-layer networks:
  - A unit whose output is a non-linear function of its inputs but whose output is also a differentiable function of its inputs.
    → Sigmoid unit

Multi-layer networks

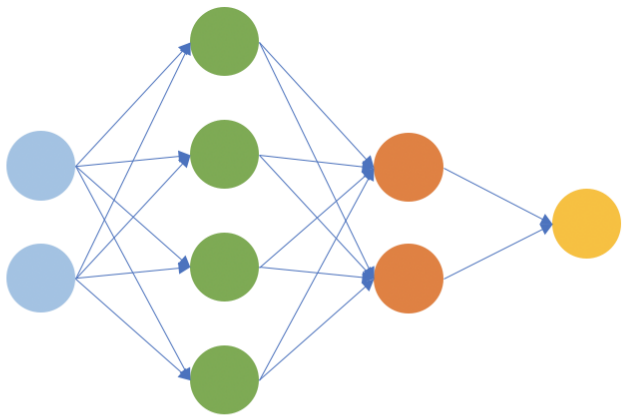
- A feedforward network has a layered structure

  → Each layer consists of units which receive their input from units from a layer directly below and send their output to units in a layer directly above the unit.

  → There are no connections within a layer.



Input layer | Hidden layer | Output layer



Input layer | First hidden layer | Second hidden layer | Output layer

Multi-layer networks

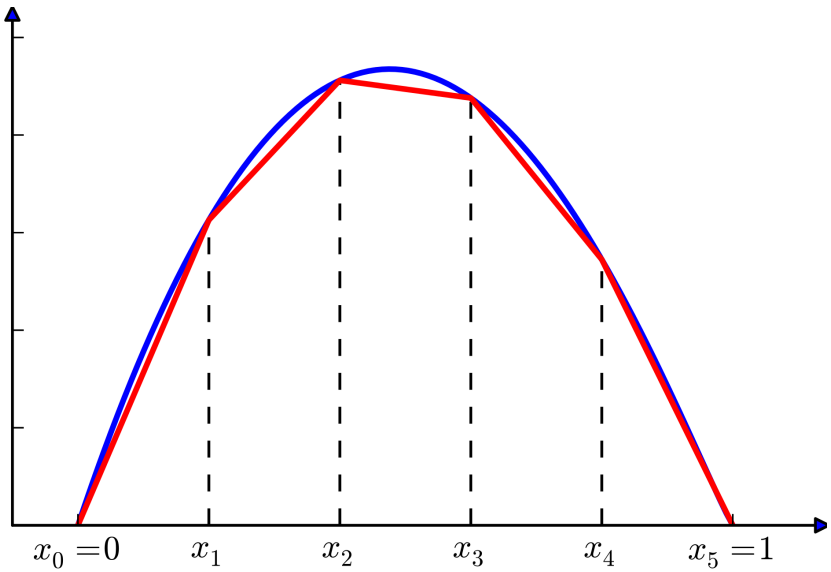
- A multi-layer network is a non-linear function between the inputs and the outputs.
- The hidden units
  - produce hidden values
  - use non-linear threshold functions (i.e. the hidden values are non-linear functions of the inputs)
- Typically, a non-linear threshold function is applied at the end.
  - This ensures that the output is a value between $[0, 1]$, or $[-1, 1]$ as before.
- Weights are applied at each layer.
- During learning, weights are updated so that the output matches the input.

KING'S College LONDON

Non-linearity

- A multi-layer network of linear functions creates a piece-wise linear boundary.

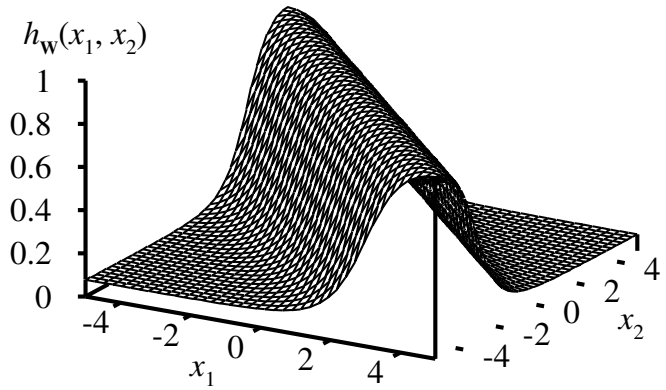
  → A series of linear segments



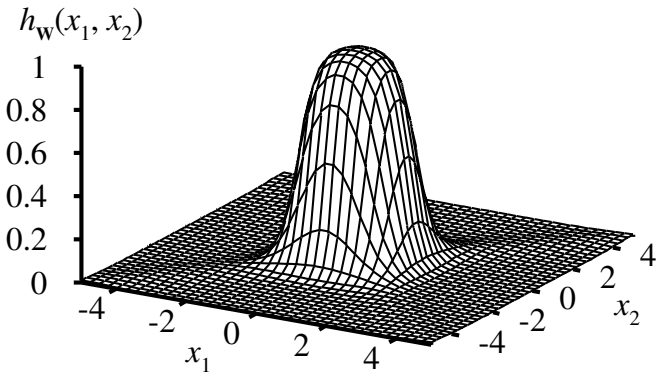
  - A multi-layer linear network is equivalent to a single-layer linear network.

- A smooth non-linear boundary needs non-linear outputs on the constituent units.

Non-linearity



(a) ridge



(b) bump

(Russell & Norvig)

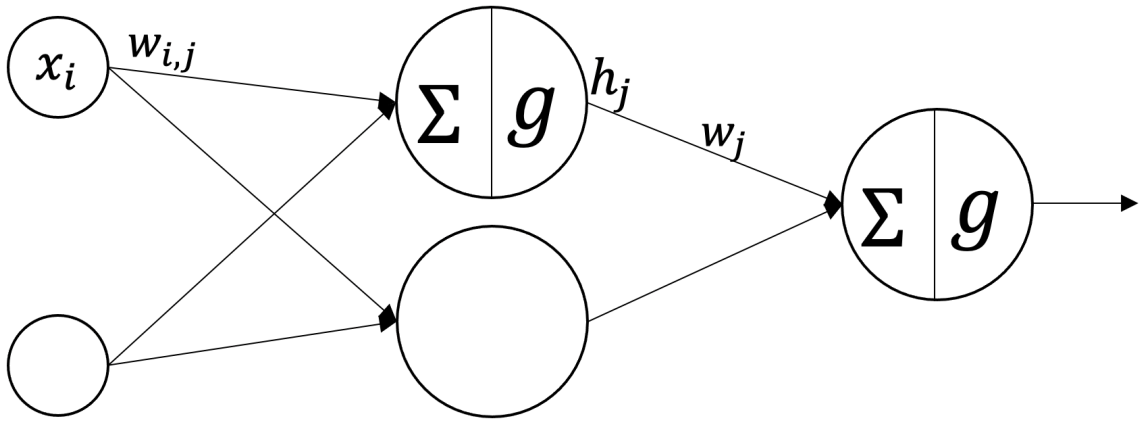
- (a) shows the result of combining two opposite-facing soft threshold functions to produce a ridge.
- (b) shows the result of combining two ridges to produce a bump.

Representational power

- Universal function approximator theorem
  - A feed-forward neural network with at least one hidden layer (and sufficient number of hidden units) can approximate almost any continuous function

Multi-layer networks

- In a multi-layer network, each layer is evaluated in sequence.



- For example:
  1. input → hidden (N input nodes, H hidden nodes):

$$\forall\, j \in \{1 \ldots h\} : h_j = g\left(\sum_{i=1}^{N} x_i w_{i,j}\right)$$

  2. hidden → output (H hidden nodes, 1 output node):

$$out = g\left(\sum_{v=1}^{H} h_j w_j\right)$$

where $g(\cdot)$ is the threshold function

Multi-layer networks

- Note we are assuming the same function $g(\cdot)$ is used throughout.

  $\rightarrow$ Not necessarily so.

- Sometimes a bias is added in at each layer.
  - input $\rightarrow$ hidden (N input nodes, H hidden nodes):

$$\forall\, j \in \{1 \ldots h\} : h_j = g\left(b_i + \sum_{i=1}^{N} x_i w_{i,j}\right)$$

  - hidden $\rightarrow$ output (H hidden nodes, 1 output node):

$$out = g\left(b_j + \sum_{v=1}^{H} h_j w_j\right)$$

- We have assumed a single hidden layer.

  $\rightarrow$ More complex structures are now often used.

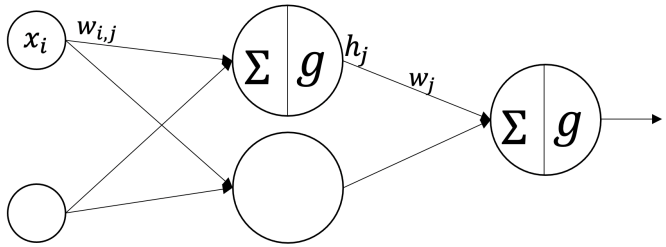
Multi-layer networks

- The training process uses gradient descent.
- Backpropagation is the classic approach to doing this.
- The weights are adjusted by moving backwards
  1. First from the *hidden* ← *output* layer
  2. Then from the *input* ← *hidden* layer
- The amount of adjustment is proportional to the value of the error function:
  - Big errors ⇒ Big adjustments
  - Small errors ⇒ Small adjustments
- The error rate should decline during the training process.

KING'S College LONDON

Backpropagation (hidden to output weights)

- Error at the output: $E = y - g(s)$
- Can use this to change the weights between the hidden layer and the output



- Introduce a "modified error" which is the product of error E and the weighted sum input into *out* (note the use of sigmoid):

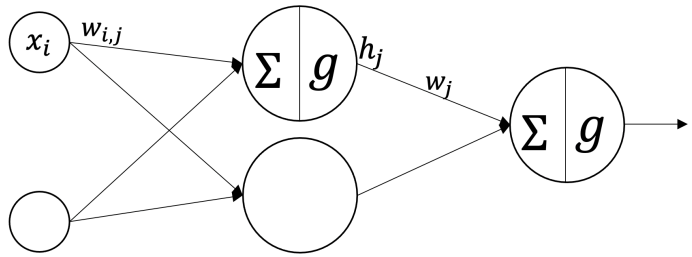
$$\Delta = g(s)(1-g(s))(y-g(s)) = g\left(\sum_{j=0}^{H} h_j w_j\right)\left(1 - g\left(\sum_{j=0}^{H} h_j w_j\right)\right)E$$

- The update for w_j becomes:

$$w_j \leftarrow w_j + \alpha h_j \Delta$$

Backpropagation (input to hidden weights)

- h_j is responsible for some fraction of the error Δ in every input node it is connected to.
- Weigh Δ according to w_j to give a Δ_j for each node in the hidden layer.



$$\Delta_j = g\left(\sum_{i=0}^{N} x_i w_{i,j}\right)\left(1 - g\left(\sum_{i=0}^{N} x_i w_{i,j}\right)\right) w_j \Delta \qquad (1)$$

- Again, this is very similar to the update for a single perceptron, where the error term is $w_j\Delta$.
- This then gives us a rule for updating each of the weights $w_{i,j}$ on the links between h_j and input x_i:

$$w_{i,j} \leftarrow w_{i,j} + \alpha x_i \Delta_j$$

Backpropagation

- Typically outputs not output.
  - Training examples have multiple outputs: $\mathbf{y} = y_1, \ldots, y_o$
  - The neural network can have multiple outputs: $\mathbf{s} = s_1, \ldots, s_o$
  - Thresholded outputs are also a vector: $\mathbf{g}(\mathbf{s}) = g(s_1), \ldots, g(s_o)$
- Treat each output unit just like we treated the single output unit we considered.
- Instead of Eq 1, define the effect of each output o on each hidden unit j:

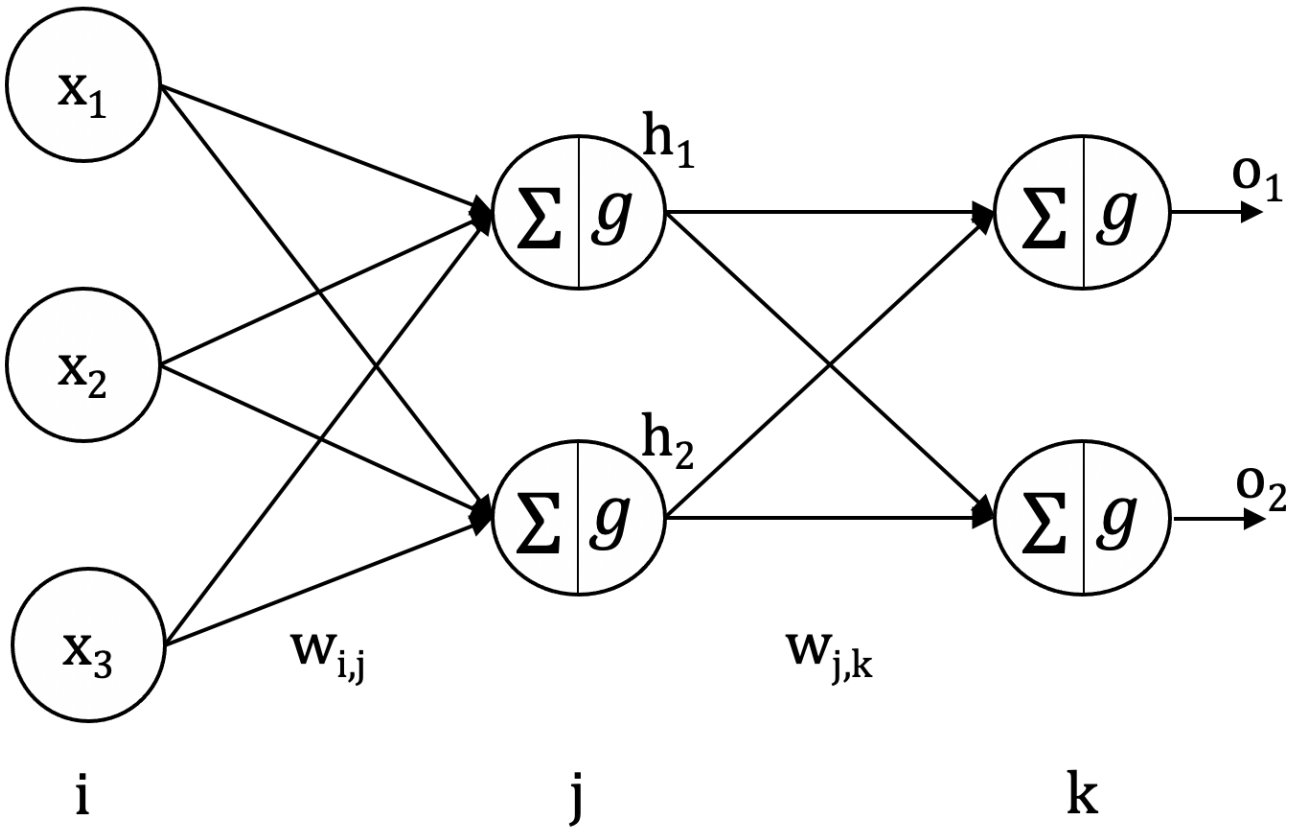
$$\delta_{j,o} = g\left(\sum_{i=0}^{N} x_i w_{i,j}\right)\left(1 - g\left(\sum_{i=0}^{N} x_i w_{i,j}\right)\right) w_j \Delta$$

- Then, the total update at each hidden unit j is:

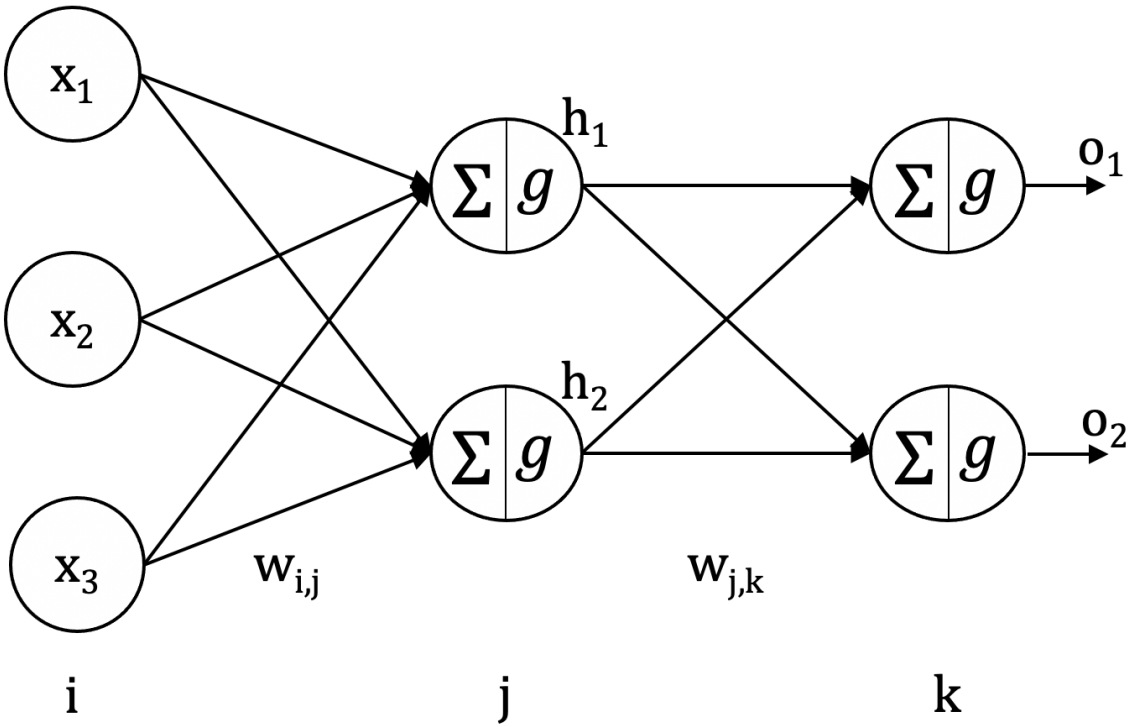
$$\Delta_j = \sum_o \delta_{j,o}$$

and we then use Δ_j as before.

Backpropagation example



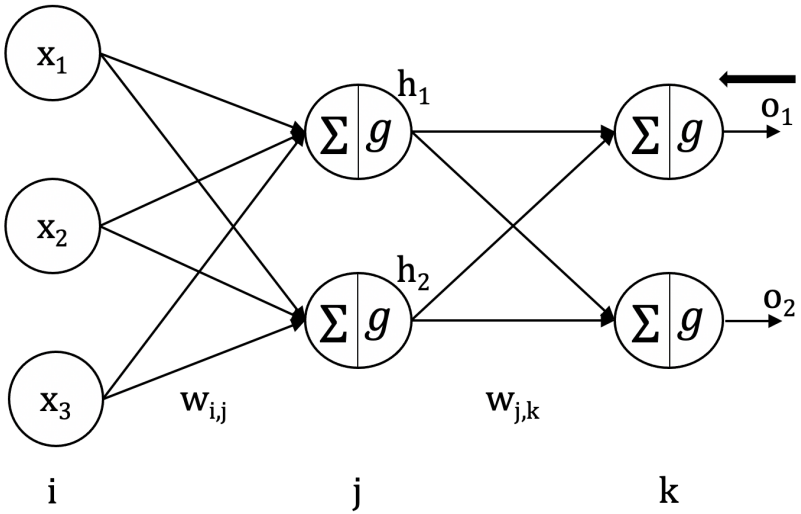
Backpropagation example



$$h_j = g\left(\sum_i x_i w_{i,j}\right) = g(s_j)$$

$$o_k = g\left(\sum_j h_j w_{j,k}\right) = g(s_k)$$

Backpropagation example



$$E_{total} = \frac{1}{2}\sum_k (y_k - o_k)^2$$

- We need to compute $\frac{\partial E}{\partial w_{j,k}}$
- Using the chain rule: $\frac{\partial E}{\partial w_{j,k}} = \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial s_k}\frac{\partial s_k}{\partial w_{j,k}}$

Backpropagation example

$$\frac{\partial E}{\partial w_{j,k}} = \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial s_k}\frac{\partial s_k}{\partial w_{j,k}}$$

$$\frac{\partial E}{\partial o_k} = \frac{\partial}{\partial o_k}\left(\frac{1}{2}\sum_k (y_k - o_k)^2\right) = -(y_k - o_k)$$

$$\frac{\partial o_k}{\partial s_k} = \frac{\partial g(s_k)}{\partial s_k} = g(s_k)(1 - g(s_k))$$

$$\frac{\partial s_k}{\partial w_{j,k}} = \frac{\partial}{\partial w_{j,k}}\left(\sum_j h_j w_{j,k}\right) = h_j$$

$$\frac{\partial E}{\partial w_{j,k}} = \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial s_k}\frac{\partial s_k}{\partial w_{j,k}} = -\underbrace{(y_k - o_k)g(s_k)(1 - g(s_k))}_{\delta_k} h_j = -\delta_k h_j$$

KING'S College LONDON

Backpropagation example

- The rule for updating weights is:

$$w_i \leftarrow w_i - \alpha \frac{\partial E}{\partial w_i}$$

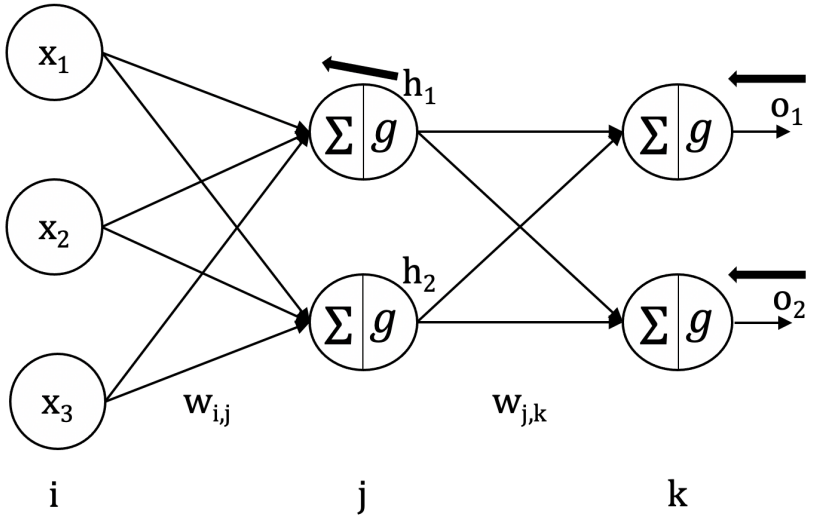
- We obtained:

$$\frac{\partial E}{\partial w_{j,k}} = -\delta_k h_j$$

- Thus:

$$w_{j,k} \leftarrow w_{j,k} + \alpha \delta_k h_j$$

Backpropagation example



- We need to compute $\frac{\partial E}{\partial w_{i,j}}$
- Using the chain rule: $\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial h_j} \frac{\partial h_j}{\partial s_j} \frac{\partial s_j}{\partial w_{i,j}}$

Backpropagation example

$$\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial h_j}\frac{\partial h_j}{\partial s_j}\frac{\partial s_j}{\partial w_{i,j}}$$

$$\frac{\partial h_j}{\partial s_j} = \frac{\partial g(s_j)}{\partial s_j} = g(s_j)(1 - g(s_j)) \qquad \frac{\partial s_j}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}}\left(\sum_i x_i w_{i,j}\right) = x_i$$

$$\frac{\partial E}{\partial h_j} = \sum_k \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial s_k}\frac{\partial s_k}{\partial h_j} \qquad \frac{\partial s_k}{\partial h_j} = \frac{\partial}{\partial h_j}\left(\sum_j h_j w_{j,k}\right) = w_{j,k}$$

$$\frac{\partial E}{\partial h_j} = \sum_k \underbrace{-(y_k - o_k)g(s_k)(1 - g(s_k))}_{\delta_k} w_{j,k} = -\sum_k \delta_k w_{j,k}$$

$$\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial h_j}\frac{\partial h_j}{\partial s_j}\frac{\partial s_j}{\partial w_{i,j}} = \underbrace{-x_i\, g(s_j)(1 - g(s_j)) \sum_k \delta_k w_{j,k}}_{\delta_j} = -\delta_j x_i$$

Backpropagation example

- The rule for updating weights is:

$$w_i \leftarrow w_i - \alpha \frac{\partial E}{\partial w_i}$$

- We obtained:

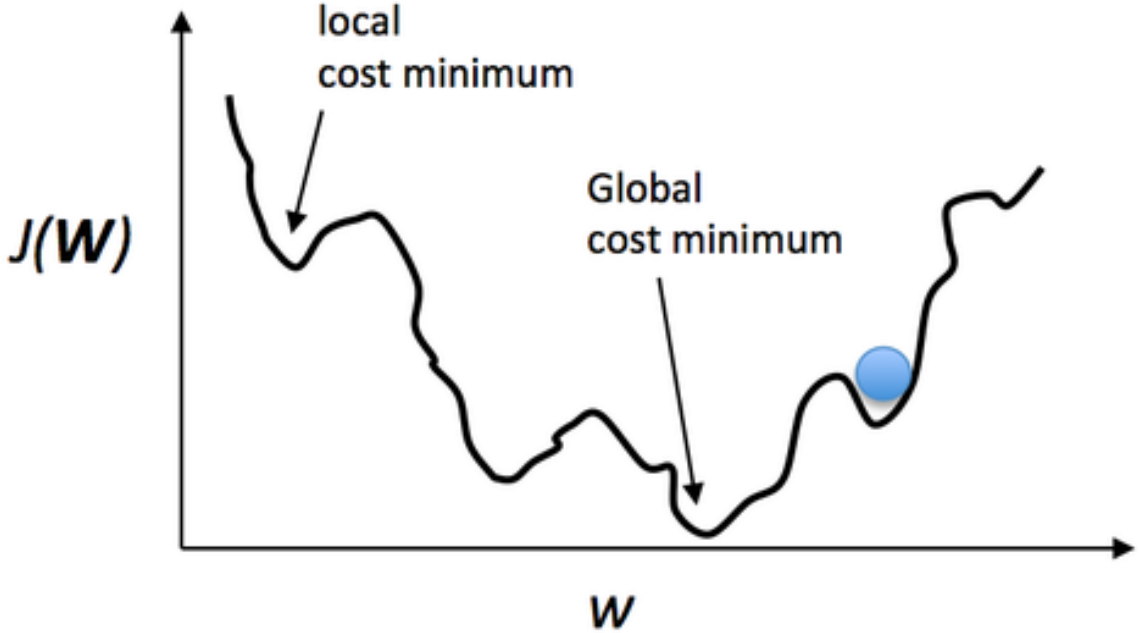
$$\frac{\partial E}{\partial w_{i,j}} = -\delta_j x_i$$

- Thus:

$$w_{i,j} \leftarrow w_{i,j} + \alpha \delta_j x_i$$

Backpropagation

- The error surface for multilayer networks may contain many different local minima.
- Backpropagation over multi-layer networks is only guaranteed to converge toward some local minimum and not necessarily to the global minimum error.
- Despite this, backpropagation is a highly effective function approximation method in practice.



Momentum

- There are many variations on the backpropagation algorithm.
- One common variation is to introduce a momentum term.
- Keep track of the weight updates in the previous iteration and make the weight update in the nth iteration depend on the update in the $(n-1)$th iteration.
- Instead of updating according to:

$$w_j \leftarrow w_j + \Delta_{w_j}$$
$$\Delta_{w_j} = \alpha h_j \Delta$$

we have:

$$\Delta_{w_j}(n) = \alpha h_j \Delta + \mu \Delta_{w_j}(n-1)$$

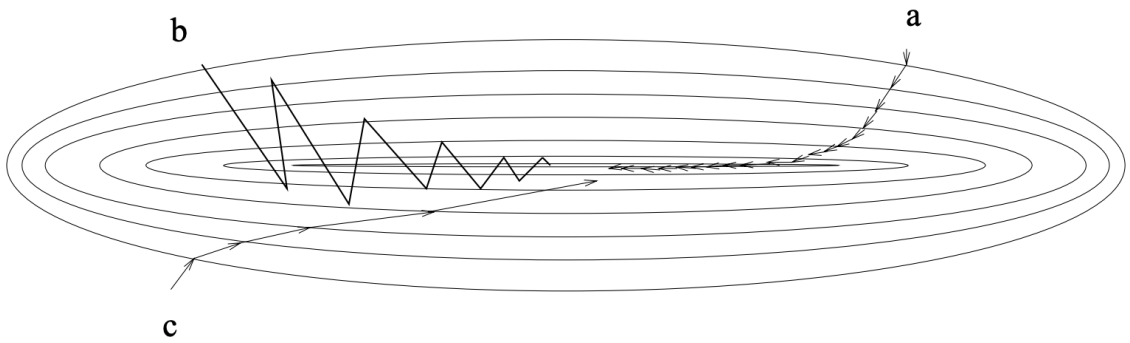
where μ is the momentum.

- Then update in the nth iteration using:

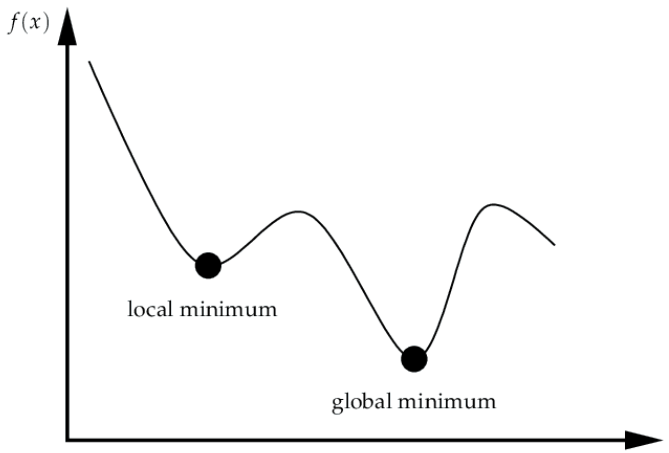
$$w_j \leftarrow w_j + \Delta_{w_j}(n)$$

Momentum

- The descent in weight space:
  - a: small learning rate
  - b: large learning rate (see oscillations)
  - c: large learning rate and momentum

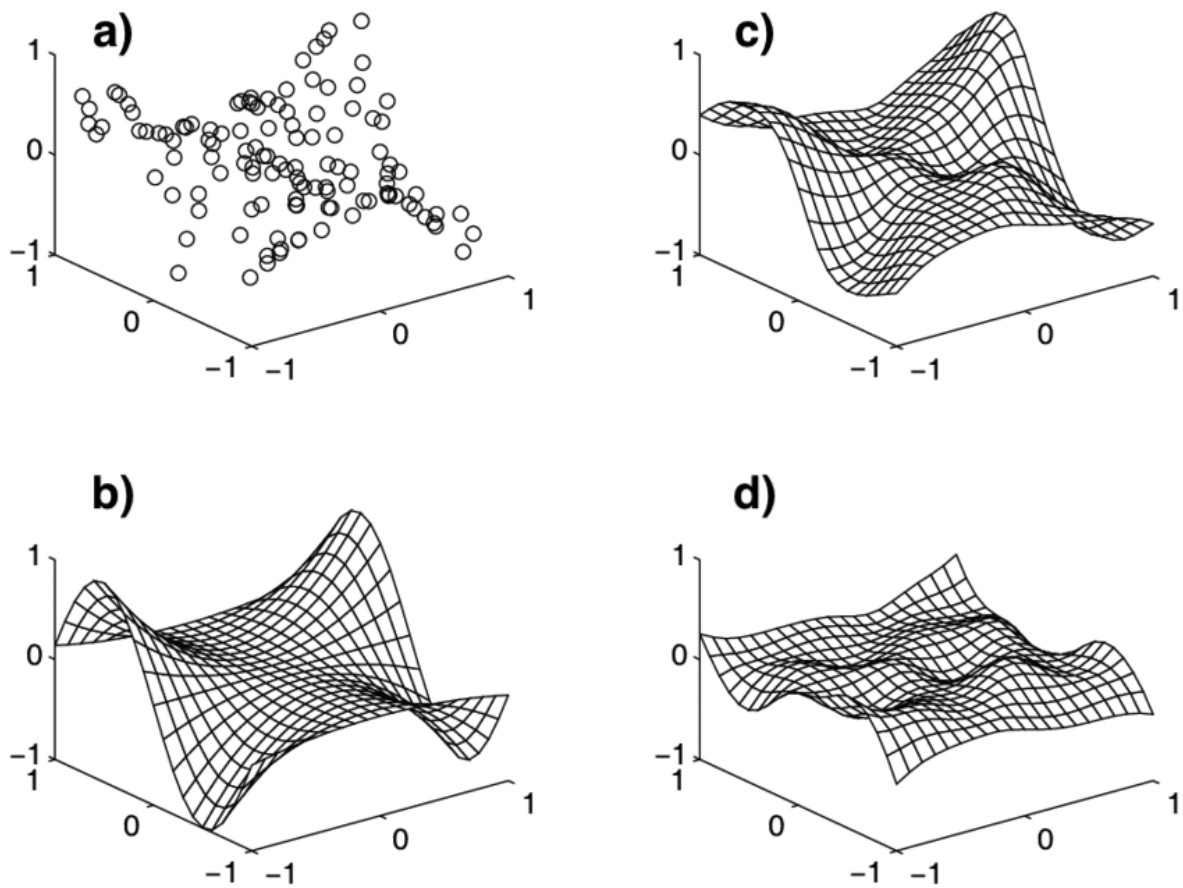


- Can “roll over” local minima rather than getting stuck.



Function approximation

- Function approximation with feedforward networks:
  - a: the original learning samples
  - b: the function that generated the learning samples
  - c: the approximation with the network
  - d: the error in the approximation



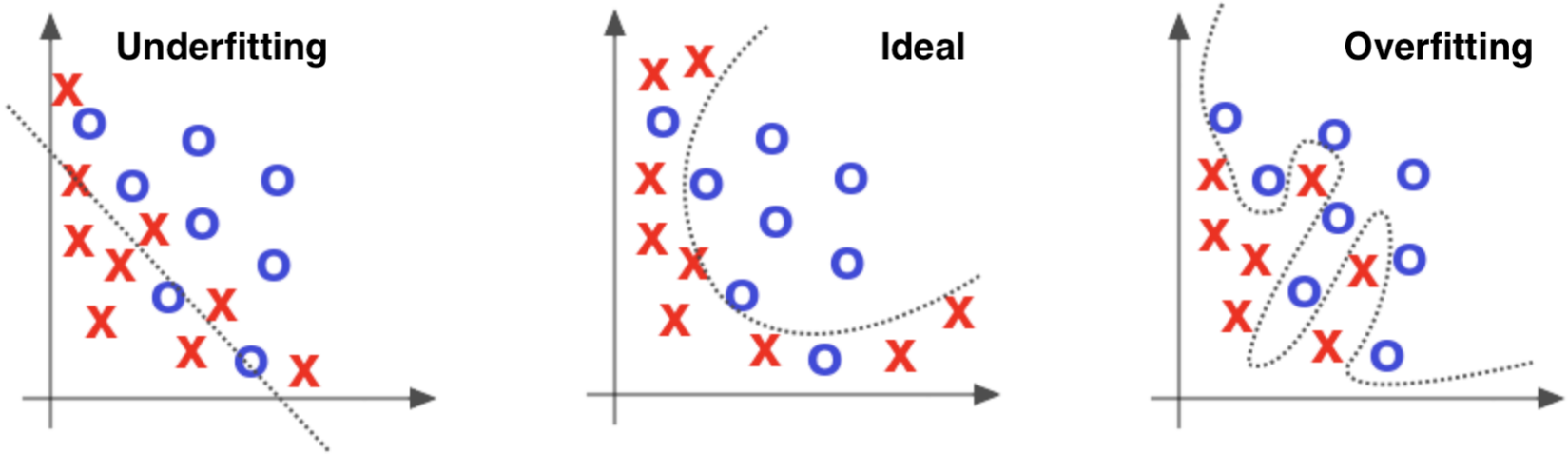
Design and implementation

- Many issues arise when implementing a neural network.
- Input encoding and output encoding.
- When to stop adjusting weights?
- Network graph structure: number of hidden layers and nodes.
- Learning parameters (learning rate α, and momentum μ).

KING'S College LONDON

Overfitting

- When the network predicts the training data very well but fails on the test data set.

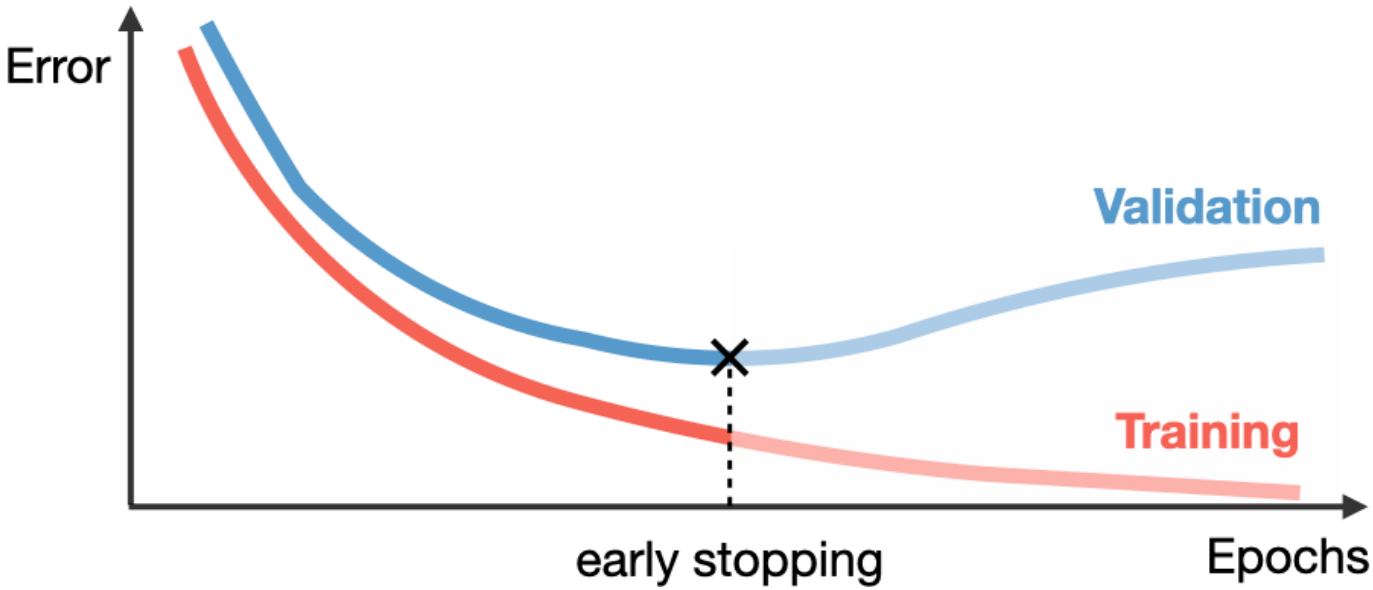


Generalisation

- The generalisation accuracy is a measure of how well the network generalises to data sets outside of the training set.
- One method of overcoming generalisation accuracy problems is to tweak the weights each iteration, by small amounts.
- Another method is to introduce a validation data set (in addition to the standard training and test sets).
- Run the network on the training data, adjusting the weights and expecting to see rapid improvement in performance on the training data.
- Also run the network on the validation data and keep the set of weights that performs best on the validation data.
- In the end, return the set of weights that did best on the validation set, not the ones that did best on the training set.

KING'S College LONDON

Generalisation



- Early stopping
  - Evaluate on validation data after each epoch and stop training when the performance of the model has not improved for several epochs.

Dynamically modifying network structure

- We have considered neural networks with a fixed graph structure.
- The structure of the network can change dynamically.
- For example start with a network with 0 hidden nodes.
- Then gradually add nodes as you train the network.

Training tips

- Rescale inputs and outputs to be in the range 0 to 1 or -1 to 1.
- Initialise weights to very small random values.
- Stochastic or batch learning.
- Three different ways to prevent overfitting:
  - limit the number of hidden nodes or connections
  - limit the training time, using a validation set
  - weight decay (decrease each weight by some small factor during each iteration)
- Adjust learning rate and momentum to suit the particular task.

KING'S College LONDON

Training tips

When to consider Neural Networks?

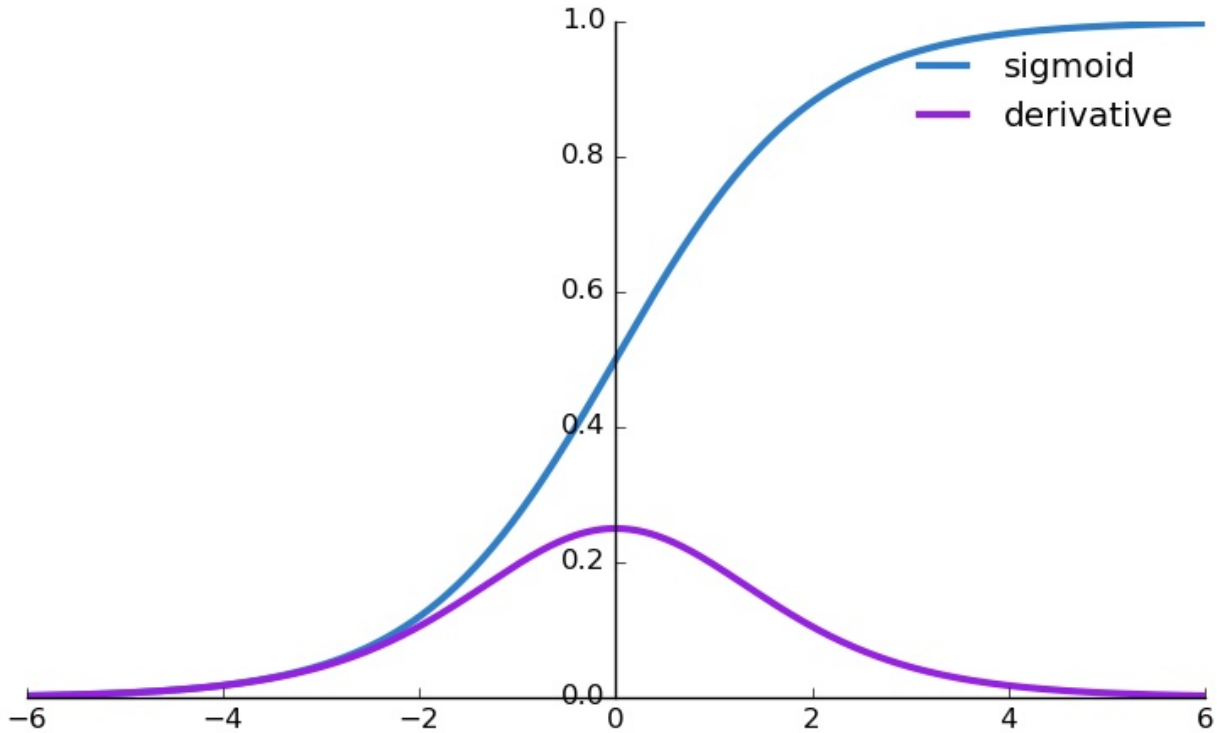
- Input is high-dimensional discrete or real-valued.
- Output is discrete, real valued or vector of values.
- Possibly noisy data. ANNs are quite robust to noise in the training data.
- Form of target function is unknown.
- Long training times are acceptable. However, evaluating a new instance is quite fast.
- Human readability of result is unimportant.

KING'S College LONDON

Deep neural networks

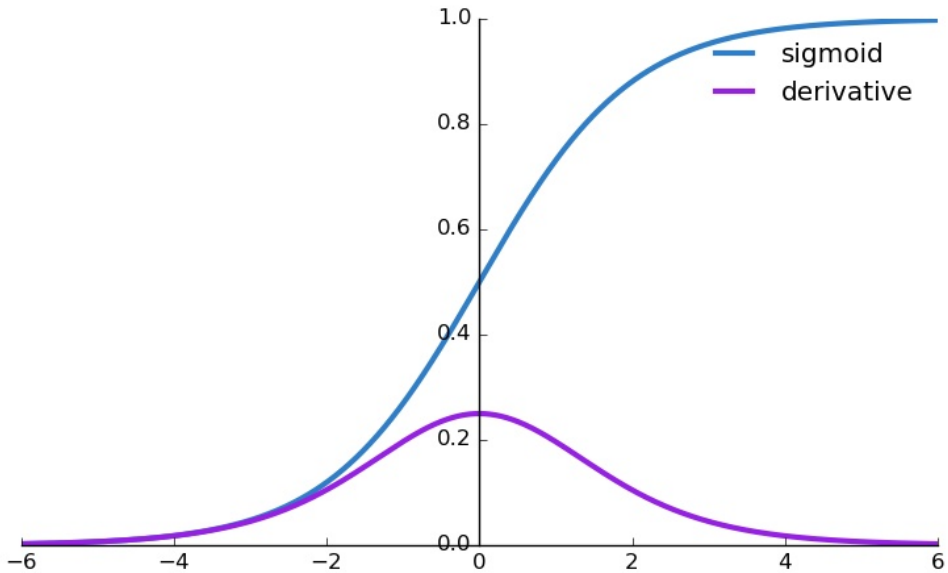
- Neural networks have recently surged in popularity.
- Deep learning = deep neural networks = neural networks with many layers.
- The weights of a neural network are updated through backpropagation by finding the gradients.
- In the case of a (very) deep NN, the gradient vanishes or explodes as it propagates backward which leads to vanishing and exploding gradient.

Vanishing gradient



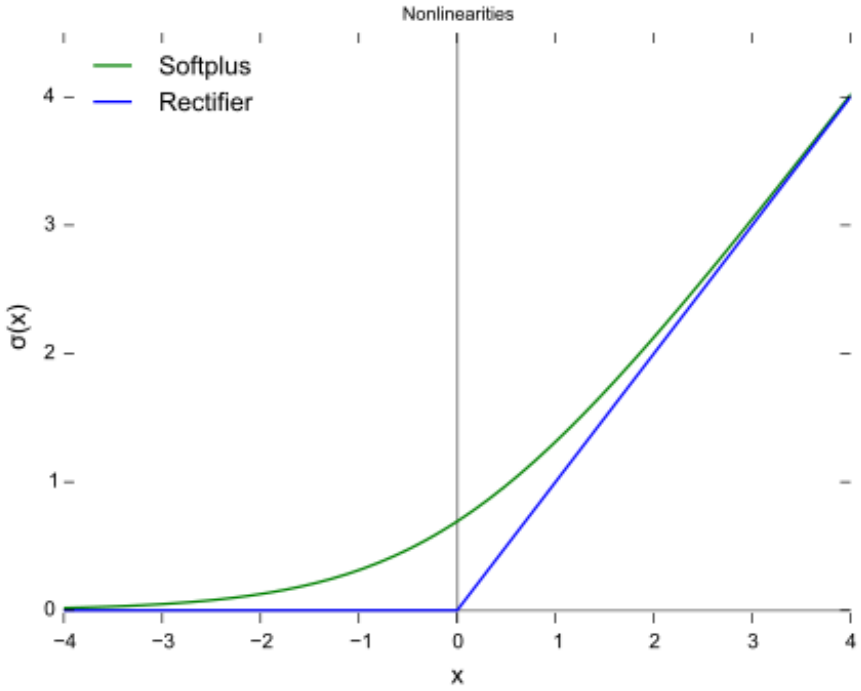
- The derivative of the sigmoid is $g(s)(1 - g(s))$.
- It has maximum value of 0.25 when $g(s)$ has value 0.5.

Vanishing gradient



- Using the chain rule, the derivatives of each layer are multiplied from the final layer to the initial layer to compute the derivatives of the initial layers.
- As the number of layers increases, the number of products required to update the weights increases.
- If the gradients become very small, the updates to the weights will be negligible.

Deep neural networks



- Rectifier transfer function.
- Re(ctifier) L(inear) U(nit): $ReLU(x) = max(0, x)$
- Helps prevent the vanishing gradient problem.

Deep neural networks

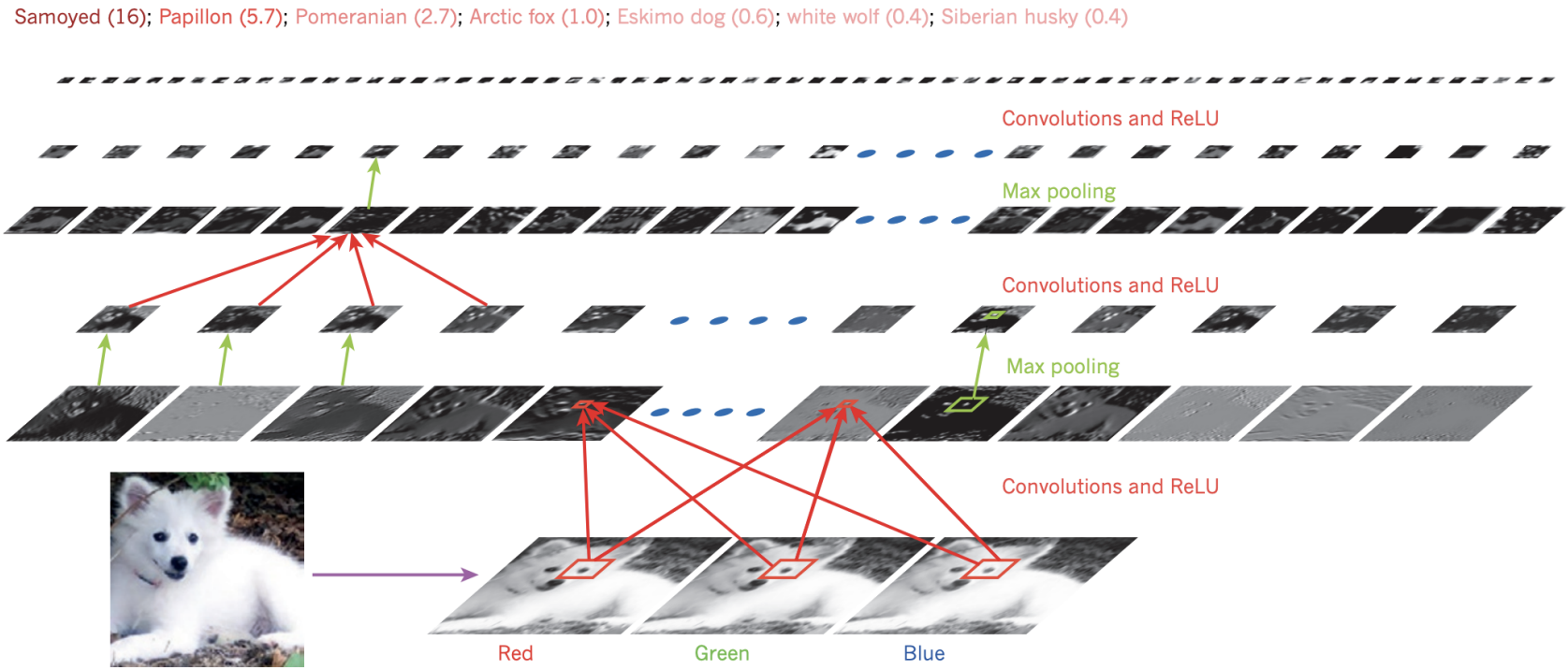
- Specialised ANN architectures:
  - Convolutional Neural Networks (for image data)
  - Recurrent Neural Networks (for sequence data)
- This a (very) high-level overview of (some) deep neural networks.

Convolutional neural networks

- Solving an image classification problem using ANN would require to convert a 2D image into a 1-dimensional vector prior to training the model and thus losing the spatial features of an image (e.g. arrangement of pixels and the relationship between them in an image).
- Convolutional neural network (CNN) captures the spatial features from an image.

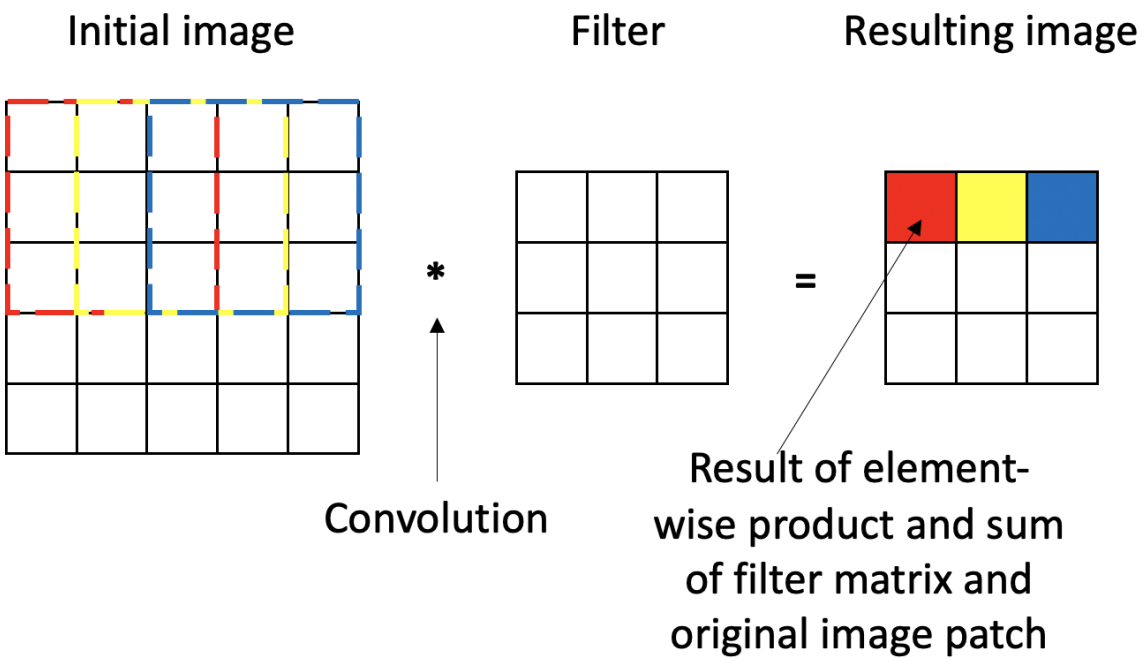
Convolutional neural networks

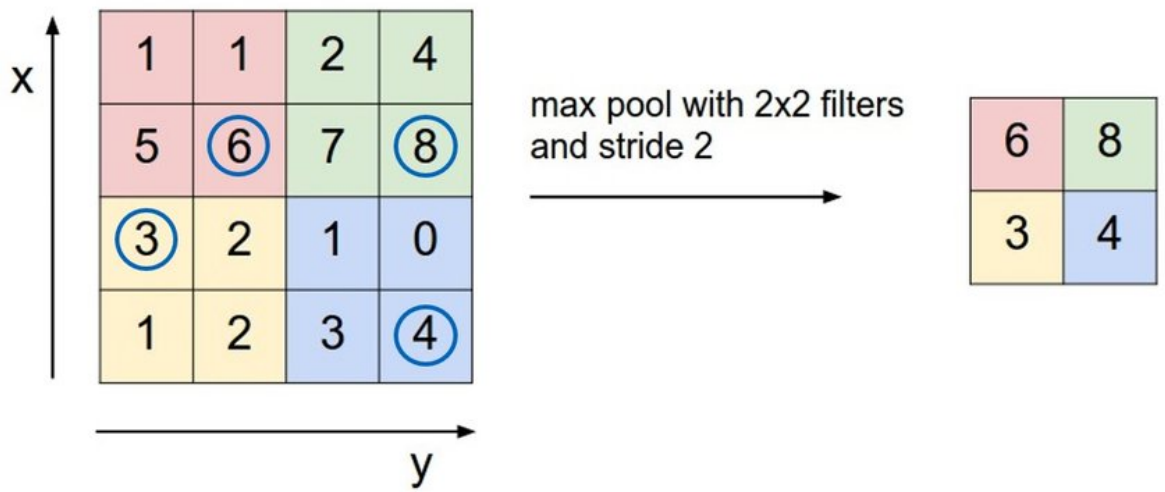
- CNN for image classification



(Deep learning, Nature)

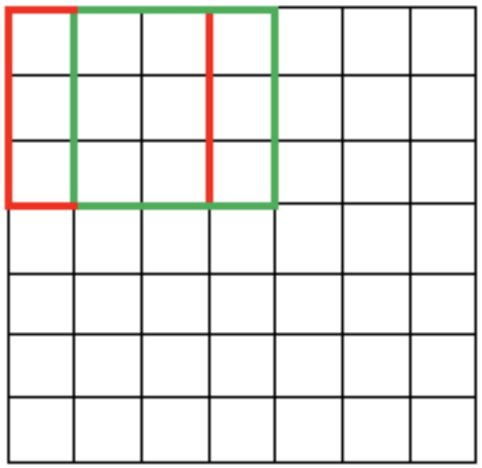
Convolutional neural networks

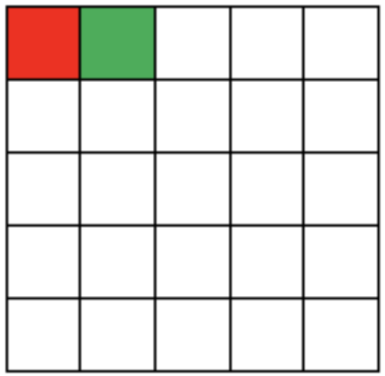




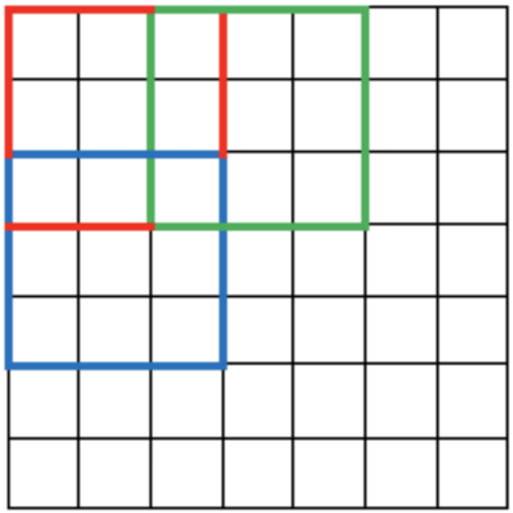
Convolutional neural networks

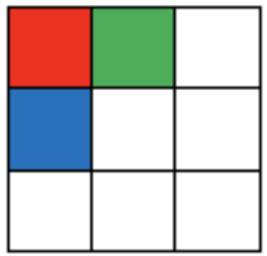
- Initial image (7×7) and stride 1 → Resulting image: (5×5)





- Initial image (7×7) and stride 2 → Resulting image: (3×3)



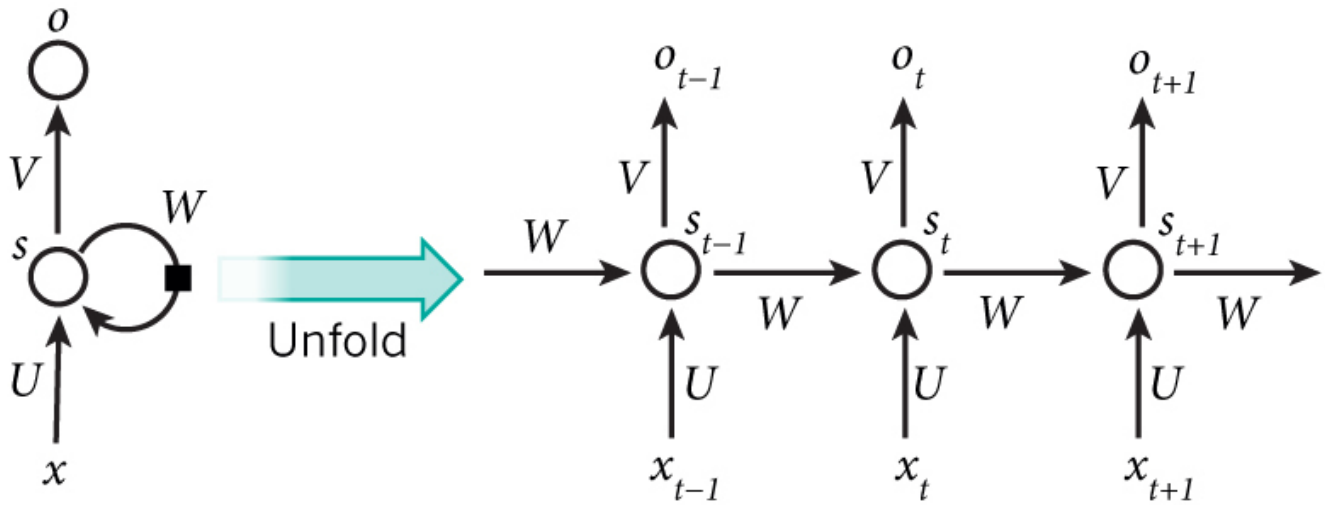


Recurrent neural networks

- What to use to solve problems related to:
  - time series data
  - text data
  - audio data
- ANN cannot capture sequential information in the input data.
- Recurrent Neural Network (RNN) has a recurrent connection on the hidden state which ensures that sequential information is captured in the input data.
- RNNs process an input sequence one element at a time, maintaining in their hidden units a ‘state vector’ that contains information about the history of all the past elements of the sequence.

Recurrent neural networks

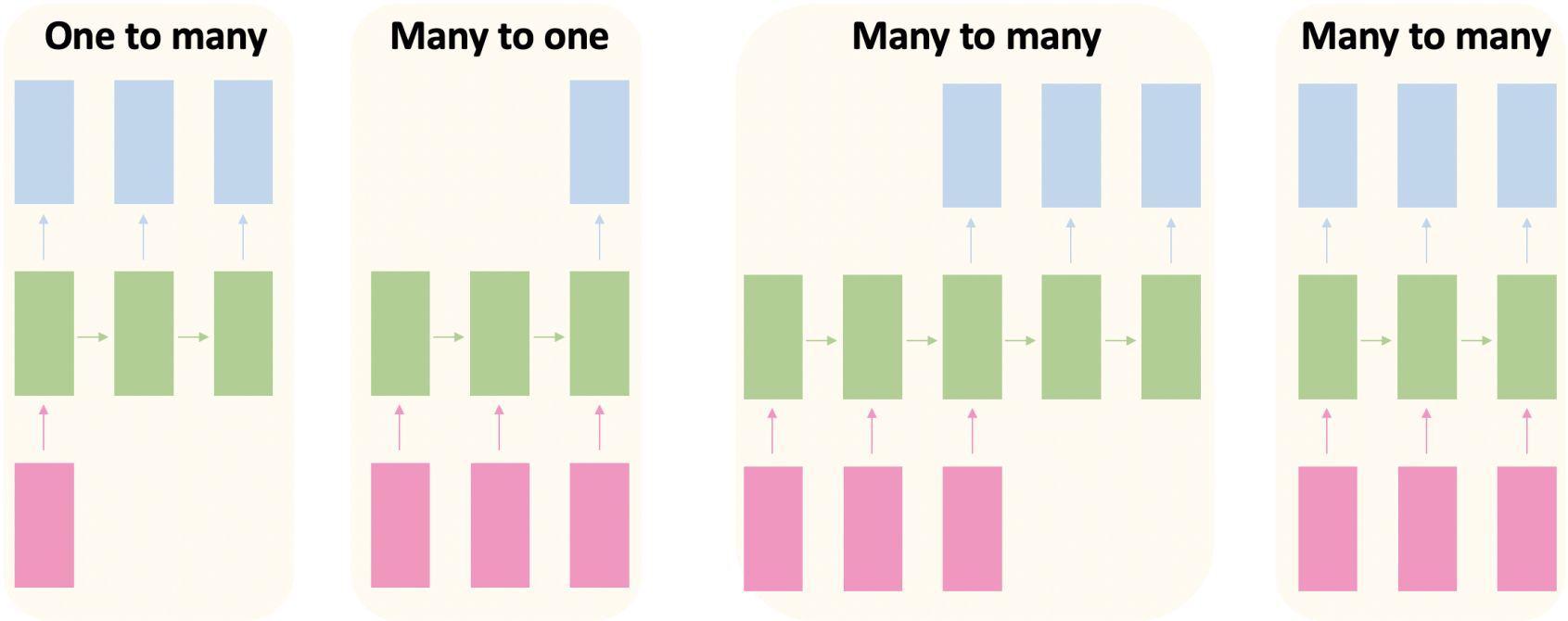
- RNN can be seen as a feedforward neural network by unrolling according to the number of time steps.



Deep learning, Nature

- We can then apply backpropagation but in the interest of memory consumption and computational efficiency, other techniques are applied (i.e. backpropagation through time).
- Variations of RNN that deal with the vanishing gradient problem that RNNs suffer from:
  - Long Short-Term Memory units (LSTM)
  - Gated Recurrent Unit (GRU)

Recurrent neural networks



- Sequence output (image captioning).
- Sequence input (sentiment analysis).
- Sequence input and sequence output (machine translation)
- Synced sequence input and output (video classification on each frame).

KING'S College LONDON

Deep neural networks - open issues

- Require lots of data.
  - Overfitting is a big problem.
  - GANs (Generative Adversarial Networks) may be a solution.
  - Also, combination with symbolic methods.
- Lots of CPUs/GPUs needed to train.
  - Tens to hundreds of thousands of epochs.
  - Lots of power and lots of heat.
  - Not so great in a climate emergency.
  - GPT-3 has been trained on 175 billion parameters.
- Explainability.

Summary

- We looked at various neural networks.
- Started with the biological plausibility (or not).
- Perceptron.
  - Three different ways to train.
- Multilayer networks.
- Backpropagation.
- Deep neural networks.