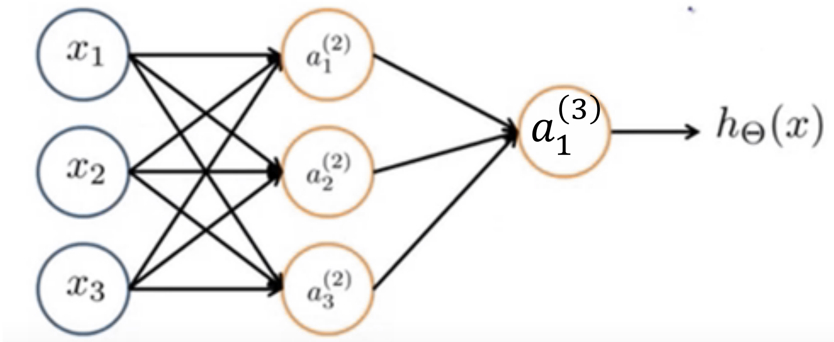
Imperial College London

A Simple Neural Network in Matrix Form



Layer $l = 1$ Layer $l = 2$ Layer $l = 3$

$a^{(l)} =$ *Activation* vector of layer l

$\theta^{(l)} =$ *Matrix of weights controlling mapping from layer l to layer l+1*
$\theta_{jk}^{(l)}$ *= weight from neuron k in layer* (l) *to neuron* j *in layer* $(l+1)$

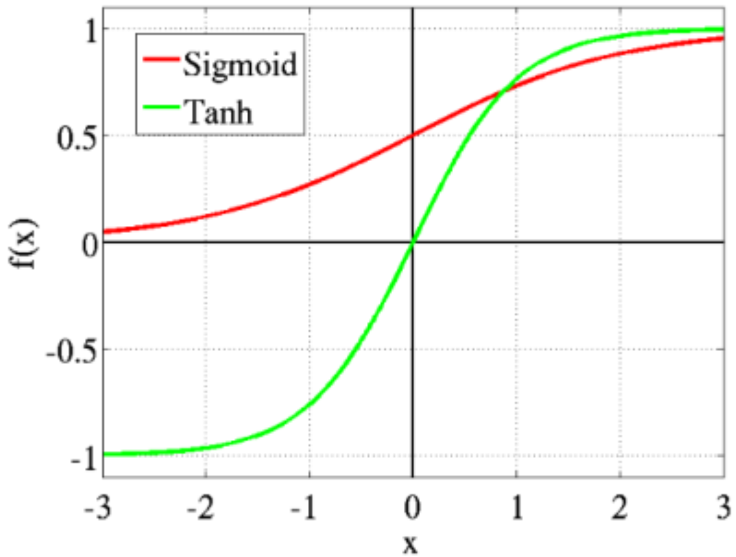
For example (if we have a bias term):

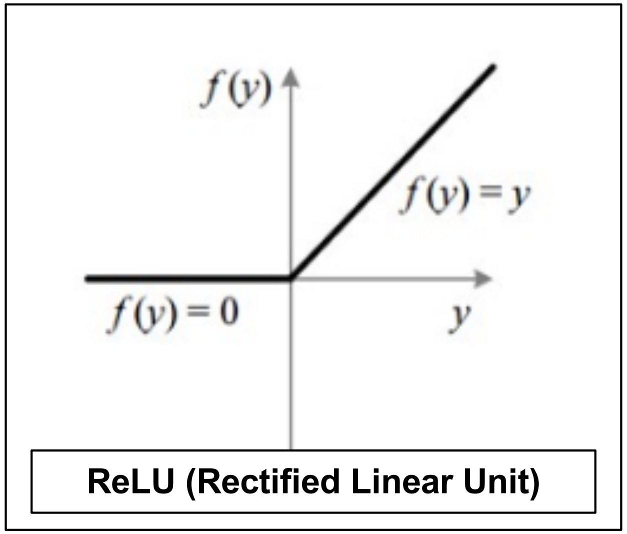
$$a^{(2)} = \begin{pmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{pmatrix} = g\left(\theta^{(1)} a^{(1)}\right) = g\left(\theta^{(1)} x\right) = g\left(\begin{pmatrix} \theta_{10}^{(1)} & \theta_{11}^{(1)} & \theta_{12}^{(1)} & \theta_{13}^{(1)} \\ \theta_{20}^{(1)} & \theta_{21}^{(1)} & \theta_{22}^{(1)} & \theta_{23}^{(1)} \\ \theta_{30}^{(1)} & \theta_{31}^{(1)} & \theta_{32}^{(1)} & \theta_{33}^{(1)} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}\right)$$

g *is the "Activation Function"*

Imperial College London

Possible Choices for the Non-Linear Activation Function g





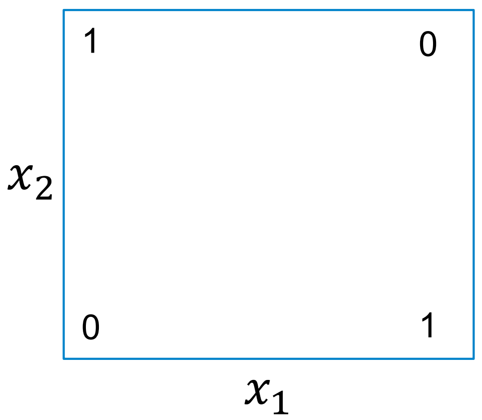
Tanh *is between -1 and +1 with mean zero (instead of mean 0.5 for sigmoid function).*
ReLU *is such that gradient does not vanish for non-zero values, most often used activatio function in practice.*
Sigmoid σ *between 0 and 1, mostly used for output layer of binary classification problems, in order to allow a probabilistic interpretation of the result.*

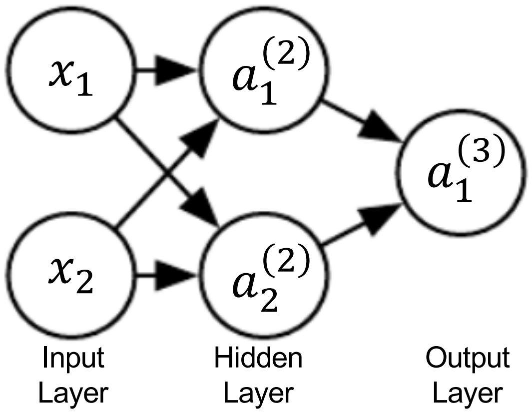
Imperial College London

Modelling the XOR Function with a Neural Network (1)

The ”Exclusive OR” or XOR function: when one of the input x_1, x_2 values is equal to 1 and the other to 0, XOR returns the value 1, otherwise 0.

This cannot be approximated by a single neuron, as we saw that the Logistic Regression Decision Boundary was a line. We need a hidden layer! Let us try the simplest possible neural network:





Inspired from Goodfellow et al, 2016

Modelling the XOR Function with a Neural Network (4)

It is easy to see that :

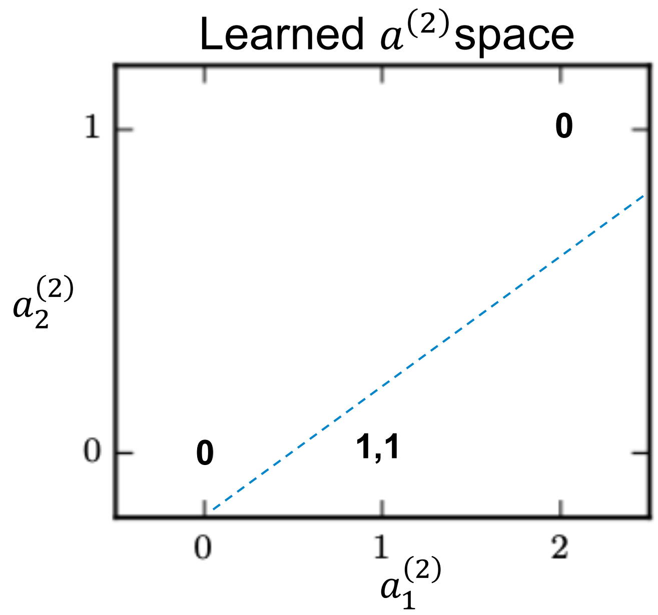
$$\begin{pmatrix}x_1\\x_2\end{pmatrix} = \begin{pmatrix}0\\0\end{pmatrix} \Rightarrow \begin{pmatrix}a_1^{(2)}\\a_2^{(2)}\end{pmatrix} = ReLU\begin{pmatrix}0\\-1\end{pmatrix} = \begin{pmatrix}0\\0\end{pmatrix}$$

$$\begin{pmatrix}x_1\\x_2\end{pmatrix} = \begin{pmatrix}0\\1\end{pmatrix} \Rightarrow \begin{pmatrix}a_1^{(2)}\\a_2^{(2)}\end{pmatrix} = ReLU\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}1\\0\end{pmatrix}$$

$$\begin{pmatrix}x_1\\x_2\end{pmatrix} = \begin{pmatrix}1\\0\end{pmatrix} \Rightarrow \begin{pmatrix}a_1^{(2)}\\a_2^{(2)}\end{pmatrix} = ReLU\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}1\\0\end{pmatrix}$$

$$\begin{pmatrix}x_1\\x_2\end{pmatrix} = \begin{pmatrix}1\\1\end{pmatrix} \Rightarrow \begin{pmatrix}a_1^{(2)}\\a_2^{(2)}\end{pmatrix} = ReLU\begin{pmatrix}2\\1\end{pmatrix} = \begin{pmatrix}2\\1\end{pmatrix}$$

The hidden layer changes the position of the four points such that they can now be linearly separated by the output layer!



..and the output layer $a_1^{(3)} = \sigma\left(\frac{5}{8}a_1^{(2)} - a_2^{(2)} - \frac{1}{2}\right)$ *gives XOR function for the four points!*

Inspired from Goodfellow et al, 2016

Imperial College London

Multi-Class Classification with Softmax

Forward propagation

$$a^{(1)} = x$$

$$z^{(2)} = \theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

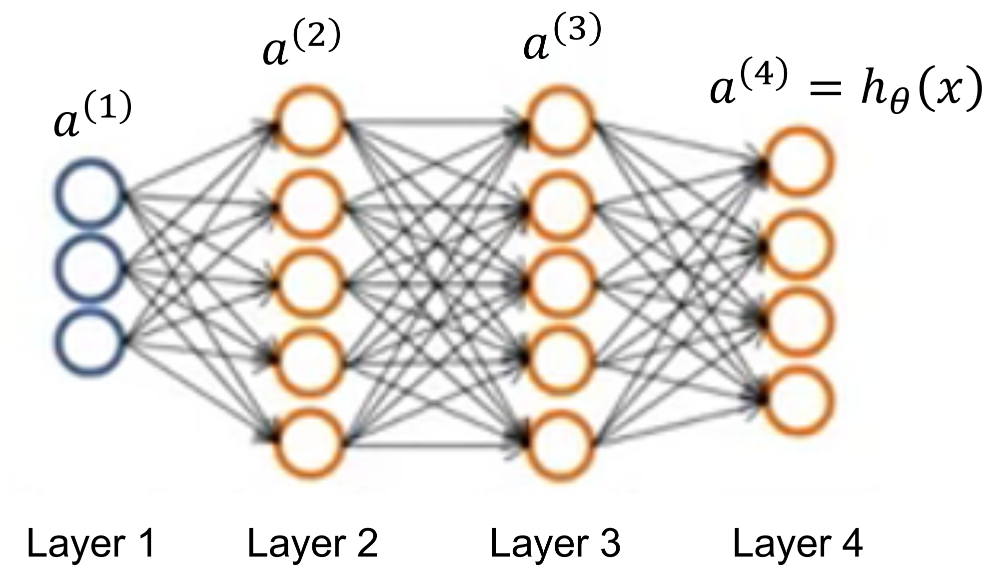
$$z^{(3)} = \theta^{(2)} a^{(2)}$$

$$a^{(3)} = g\left(z^{(3)}\right)$$

$$z^{(4)} = \theta^{(3)} a^{(3)}$$

$$a^{(4)} = \cancel{g}\left(z^{(4)}\right) = h_\theta(x)$$

Softmax



Imperial College London

Multi-Class Classification with Softmax

Suppose the output before applying the activation ($z^{(4)}$ in the previous slide) of the last layer of the neural network is $\begin{pmatrix} z_1 \\ z_2 \\ \cdot \\ z_i \\ \cdot \\ z_{n-1} \\ z_n \end{pmatrix}$ of size n

The **Softmax** function transforms it into an output probability vector:

$$\begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ \cdot \\ p_i \\ \cdot \\ p_{n-1} \\ p_n \end{pmatrix} = \begin{pmatrix} \frac{e^{z_1}}{\sum_{j=1}^{n} e^{z_j}} \\ \frac{e^{z_2}}{\sum_{j=1}^{n} e^{z_j}} \\ \cdot \\ \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \\ \cdot \\ \frac{e^{z_{n-1}}}{\sum_{j=1}^{n} e^{z_j}} \\ \frac{e^{z_n}}{\sum_{j=1}^{n} e^{z_j}} \end{pmatrix}$$

Imperial College London

Which Cross-Entropy Function for Training with Softmax? One-Hot Encoding for Defining each Class Membership.

Take the example of predicting whether the colour at one pixel of an image is brown, yellow or blue. We have three classes.

First approach

~~Code each colour as a number: 1 for brown, 2 for yellow, 3 for blue.~~
~~But this may create an artificial distance between brown and blue larger than between brown and yellow or yellow and blue!~~

Second approach: one-hot encoding

Represent the class of each pixel by a vector c of dimension equal to the number of classes:

If pixel is brown: $c = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, *If pixel is yellow:* $c = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$, *If pixel is blue:* $c = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$.

Imperial College London

Cost Function in Softmax Multi-Class Classification

For one hot-encoded yellow data point i of the Training Set: $y_i = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$

If $\begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix}$ is the probability vector calculated by Softmax for this point , the cross-entropy is defined as:

$$J(\theta) = -0 \times log p_1 - 1 \times log p_2 - 0 \times log p_3 = -log p_2$$

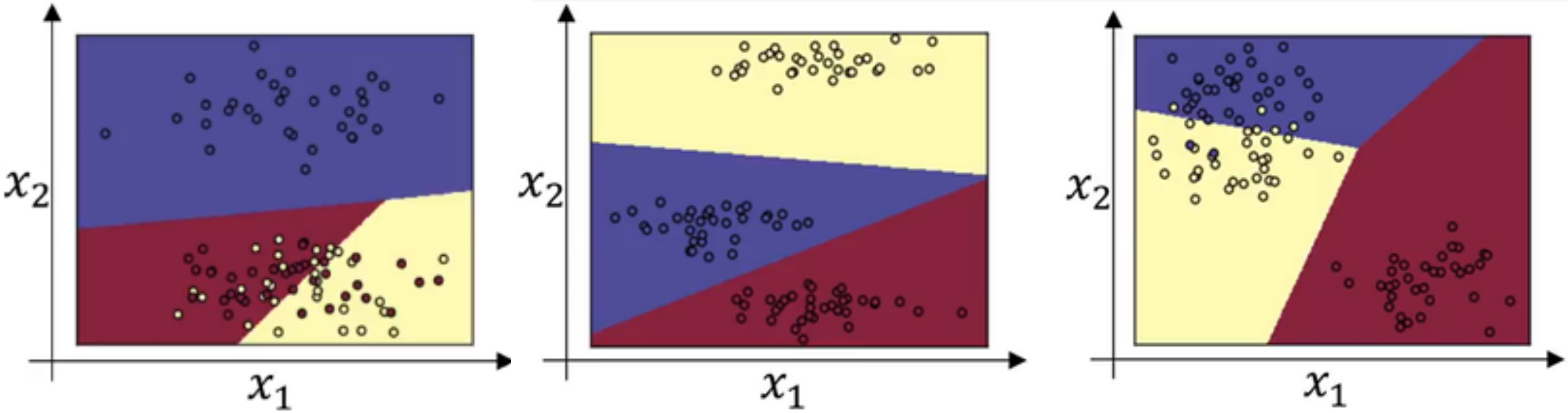
For the case of two classes, it is easy to check that we get the formula already seen on Monday.

For m data points, the above is summed for all the data points.

Examples of Classifying 2-D Data into 3 Classes

$$(x_1, x_2) \Rightarrow \overset{z}{\begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}} \Rightarrow \overset{p}{\begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix}}$$

$$\begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = Softmax \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = Softmax \begin{pmatrix} \theta_{10}^{(1)} + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 \\ \theta_{20}^{(1)} + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 \\ \theta_{30}^{(1)} + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 \end{pmatrix}$$



This neural network is quite simple and equivalent to a Logistic regression, hence the straight Decision Boundaries.

Once Trained, how does Softmax Predict a Test Point's Class?

Suppose the output of the last layer of the neural network is $\begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_i \\ \vdots \\ z_{n-1} \\ z_n \end{pmatrix}$ of size n

The **Softmax** function transforms it into an output probability vector:

$$\begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ \vdots \\ p_i \\ \vdots \\ p_{n-1} \\ p_n \end{pmatrix} = \begin{pmatrix} \frac{e^{z_1}}{\sum_{j=1}^{n} e^{z_j}} \\ \frac{e^{z_2}}{\sum_{j=1}^{n} e^{z_j}} \\ \vdots \\ \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \\ \vdots \\ \frac{e^{z_{n-1}}}{\sum_{j=1}^{n} e^{z_j}} \\ \frac{e^{z_n}}{\sum_{j=1}^{n} e^{z_j}} \end{pmatrix}$$

The class with the highest Softmax probability is selected!

Imperial College London

A Look at Back-Propagation Using a Simple Example (4)

$$a_1 \xrightarrow{w_1, b_1} a_2 \xrightarrow{w_2, b_2} a_3 \xrightarrow{w_3, b_3} a_4$$

Thanks to the formula: $\frac{\partial C}{\partial a_2} = \frac{\partial C}{\partial a_3} \frac{\partial a_3}{\partial z_3} \frac{\partial z_3}{\partial a_2} = (a_4 - y)g'(z_4)w_3 \; g'(z_3)w_1$

We can keep moving backwards and obtain the derivatives of C in w_1 and b_1

$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial a_2} \frac{\partial a_2}{\partial z_2} \frac{\partial z_2}{\partial w_1} = (a_4 - y)g'(z_4)w_3 \; g'(z_3)w_1 g'(z_2)x$$

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial a_2} \frac{\partial a_2}{\partial z_2} \frac{\partial z_2}{\partial b_1} = (a_4 - y)g'(z_4)w_3 \; g'(z_3)w_1 \; g'(z_2)$$

So we have obtained the six partial derivatives by back-propagation!

Gradient Descent: Different Ways to use the Data

Batch (also called Full-Batch) Gradient Descent:
Using all m training set data $(x_i, y_i)_{i=1,..,m}$ at each gradient descent Iteration

Stochastic Gradient Descent:
Use one single data (x_i, y_i) at each gradient descent iteration

Mini-Batch Gradient Descent:
Use small number (say a few tens or hundreds) of data (x_i, y_i) at each gradient descent iteration

*An **epoch** is a training iteration over <u>the whole training set</u>. It is thus composed of one single gradient descent iteration in the Batch case, and as many gradient descent iterations as there are training data in the Stochastic Gradient Descent case.*

https://playground.tensorflow.org

Imperial College London

Basic Neural Network Training Algorithm (one Epoch)

1. Initialize the training with random parameters $\theta_{jk}^{(l)}$
2. Calculate $h_\theta\left(x^{(i)}\right)$ for new mini-batch of training set data $x^{(i)}$
3. Calculate cost/loss function $J(\theta)$
4. Calculate gradients by back-propagation
5. Modify parameters $\theta_{jk}^{(l)}$ by gradient descent

Imperial College London

A Simple Way to See Supervised Neural Networks

Suppose we have m pairs of data.
Each pair is composed of a vector of dimension n and a vector of dimension p (the labels).

A neural network is simply a function that maps any vector of dimension n into a (discrete or continuous) vector of dimension p.

In order to calculate the parameters of this function, we train the parameters of the neural network by back-propagation using the m pairs of data as Training Set.

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{pmatrix} \xrightarrow[\textit{Neural Network}]{\textit{Feed}-\textit{Forward}} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ \vdots \\ a_{p-1} \\ a_p \end{pmatrix}$$

Input Vector

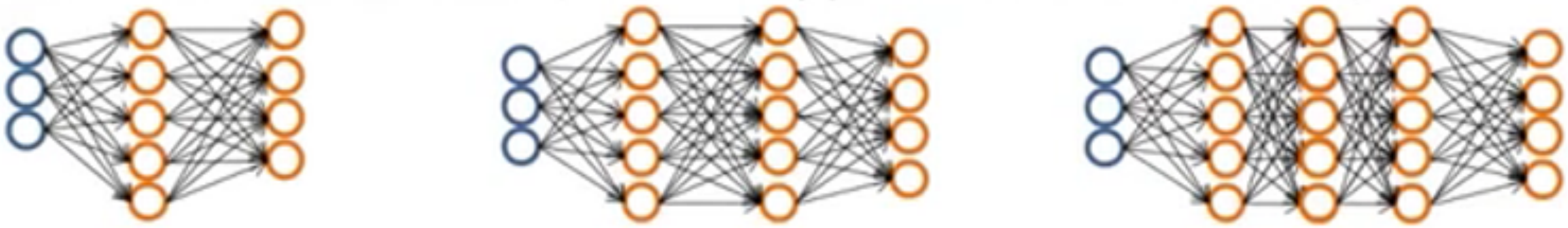
Output Labels

Imperial College
London

Architecture of a Feed-Forward Neural Network

Training a neural network

Pick a network architecture (connectivity pattern between neurons)



1. *Number of input and output units determined by number of features and number of outputs.*
2. *One hidden layer is a good starting point*
3. *If several hidden layers, good to start with the same number of units in each hidden layer*

How can the above choices be made more objective? See next lesson!