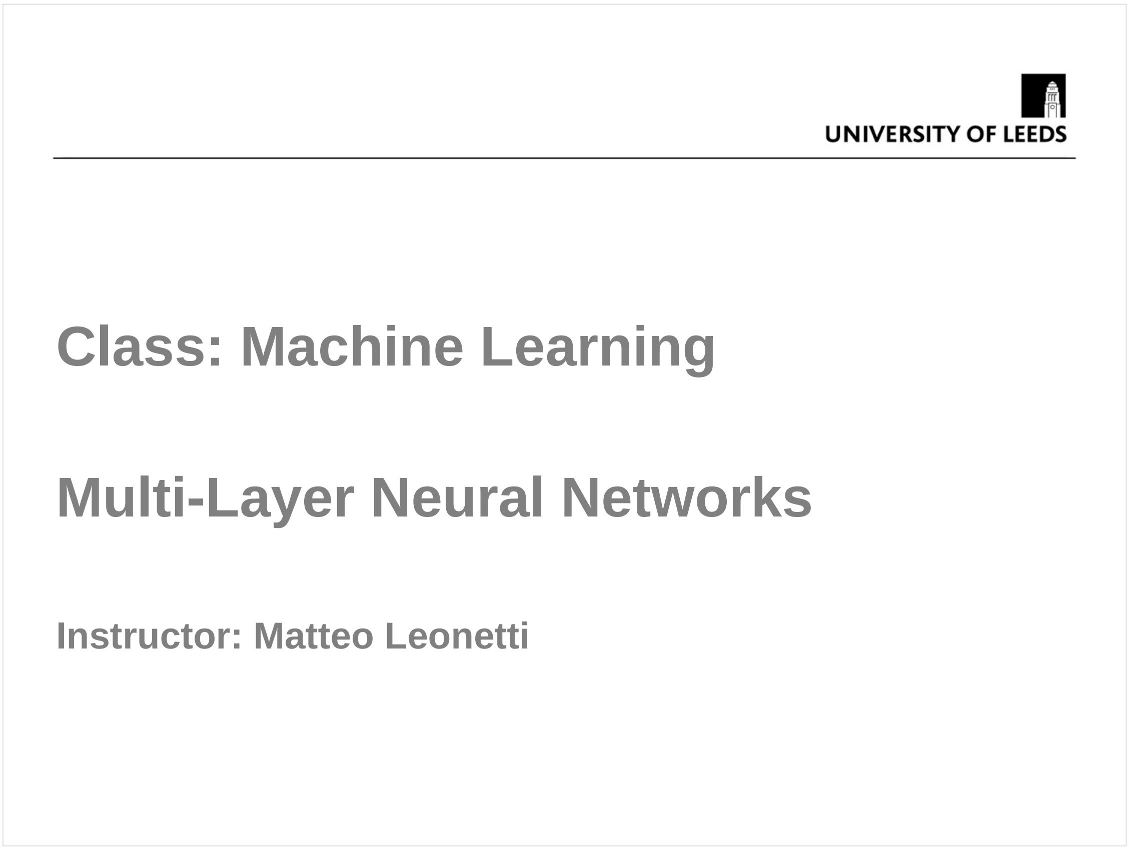
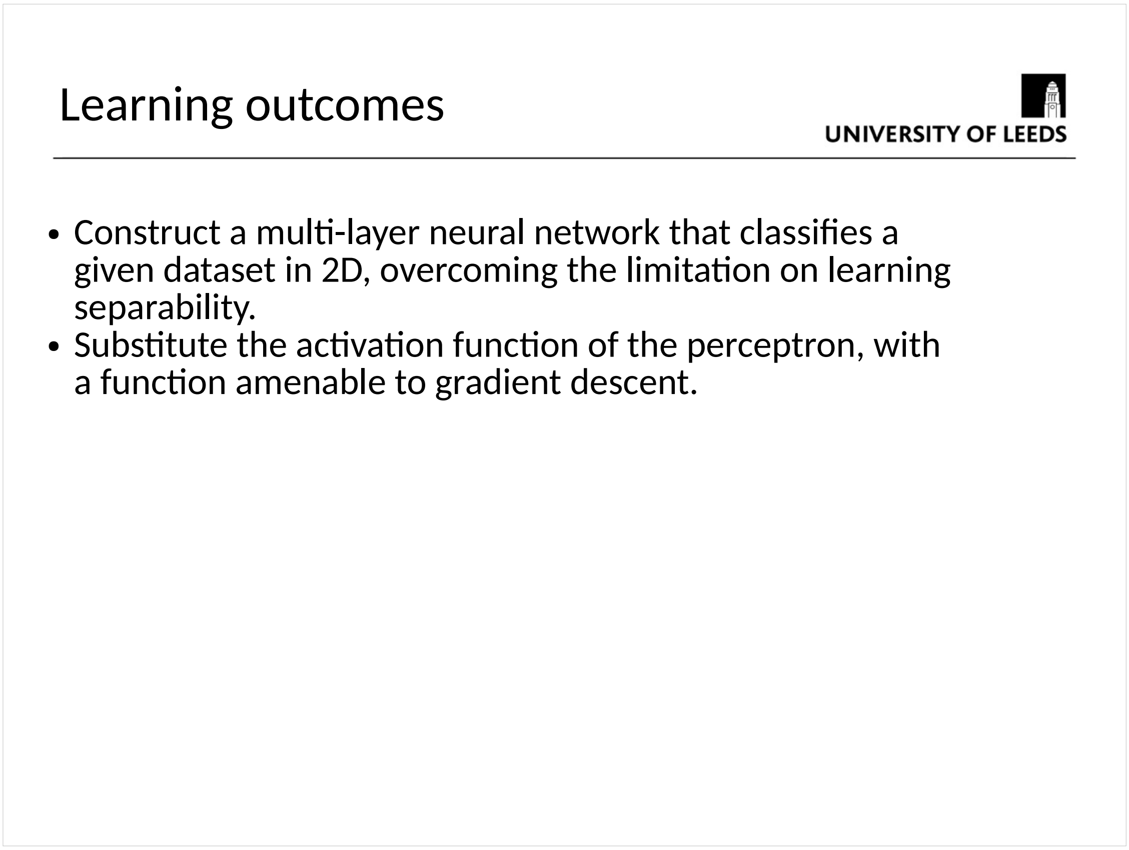


UNIVERSITY OF LEEDS

# Class: Machine Learning

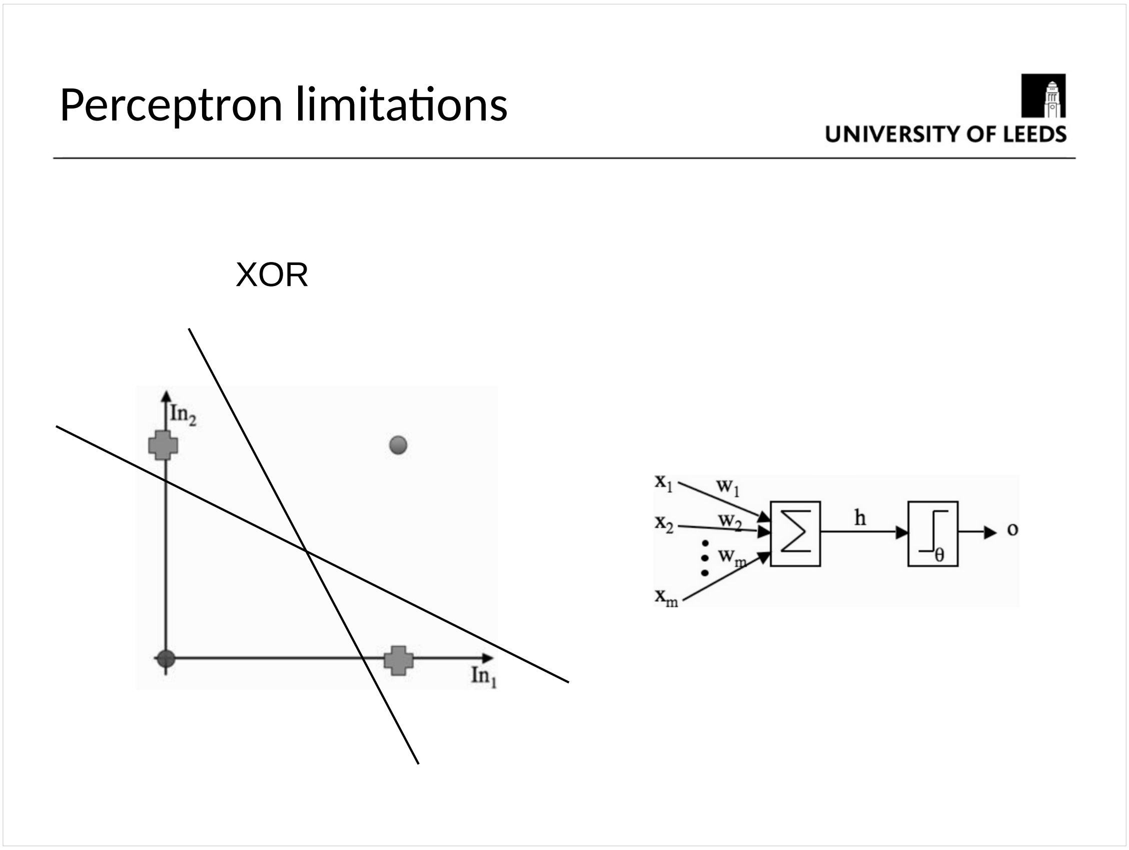
# Multi-Layer Neural Networks

## Instructor: Matteo Leonetti



# Learning outcomes

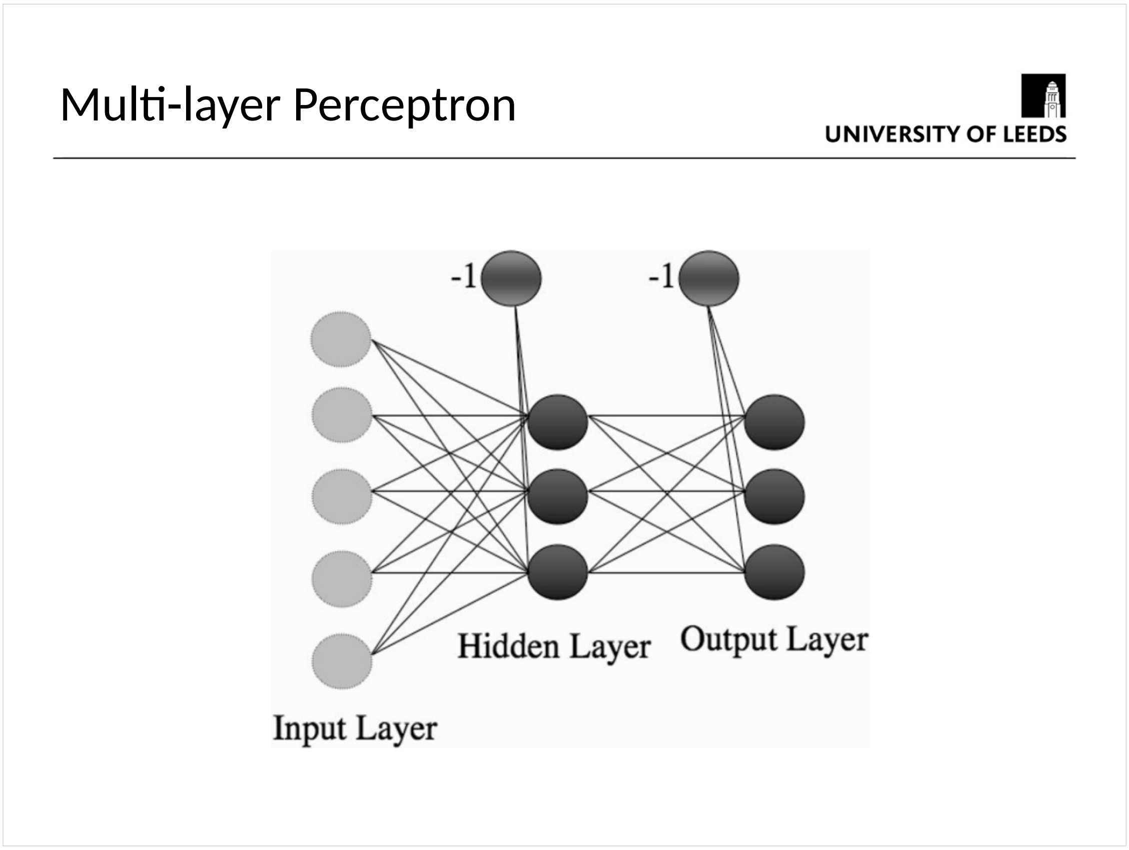
- Construct a multi-layer neural network that classifies a given dataset in 2D, overcoming the limitation on learning separability.
- Substitute the activation function of the perceptron, with a function amenable to gradient descent.



## Perceptron limitations

XOR

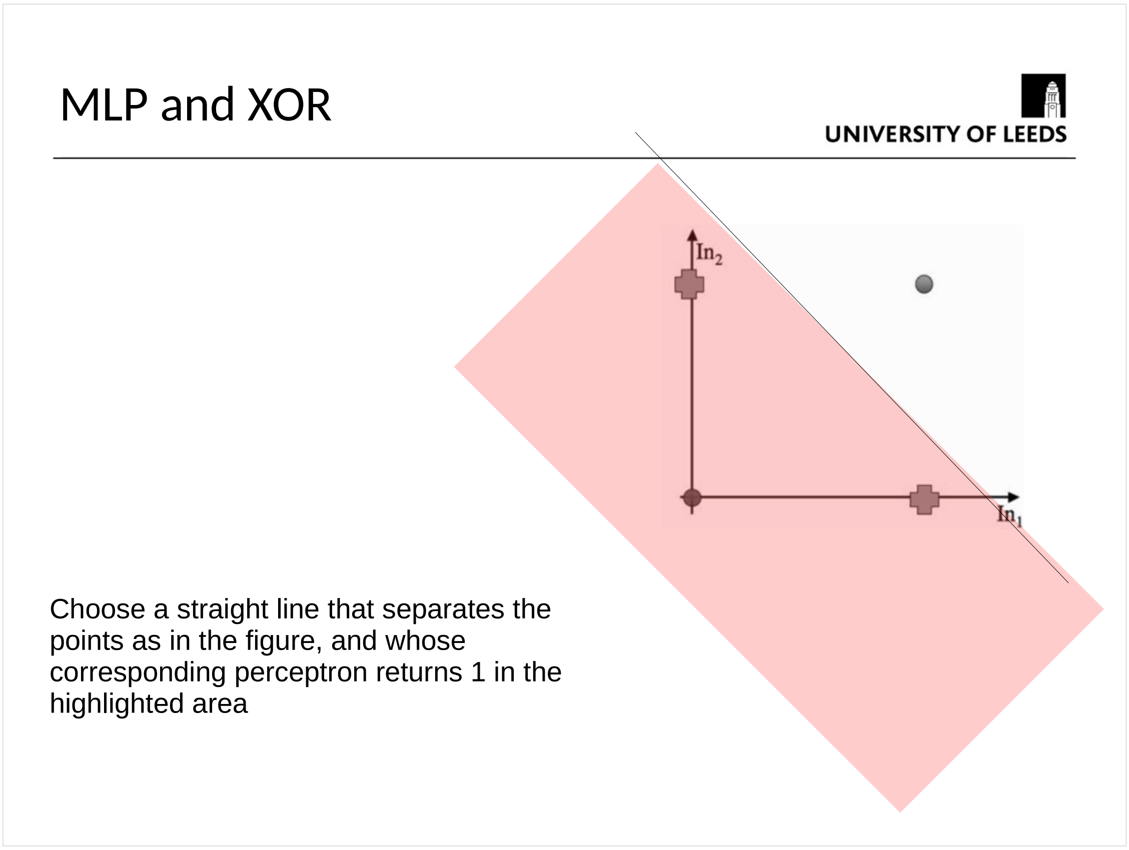
We saw that the perceptron can only classify datasets that are linearly separable. How can we address this limitation?



## Multi-layer Perceptron

The multi-layer perceptron, as the name suggests, is a feed-forward neural network composed of at least two (but may be many) layers of neurons.

We still have one output neuron per class (like the perceptron), but the layers between the inputs and the output (called the “hidden” layers) can have any size.

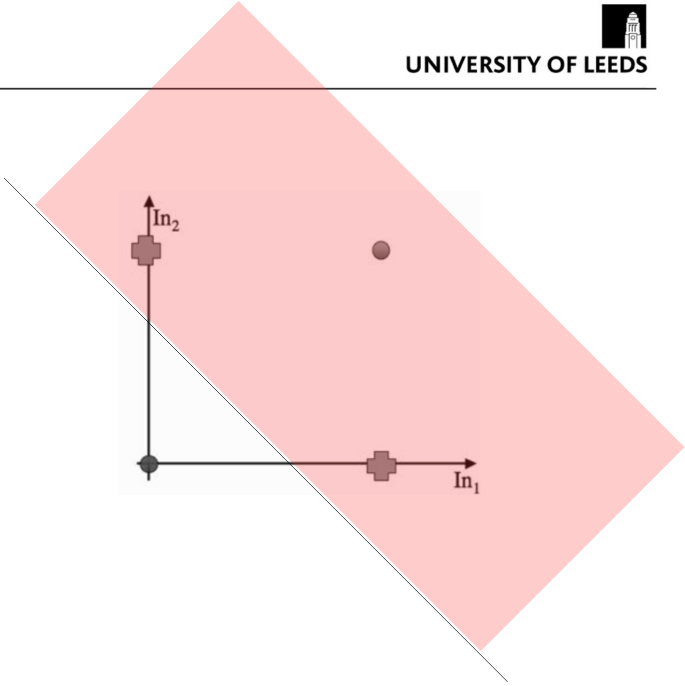


## MLP and XOR

Choose a straight line that separates the points as in the figure, and whose corresponding perceptron returns 1 in the highlighted area

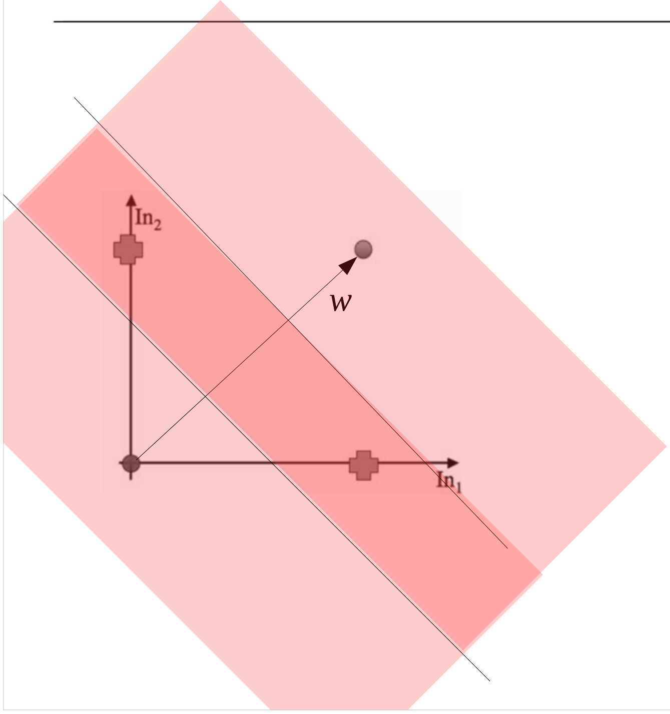
Let’s see how an MLP can be used to classify the XOR and overcome the limitation on linear separability.

# MLP and XOR



Now for the other points

# MLP and XOR



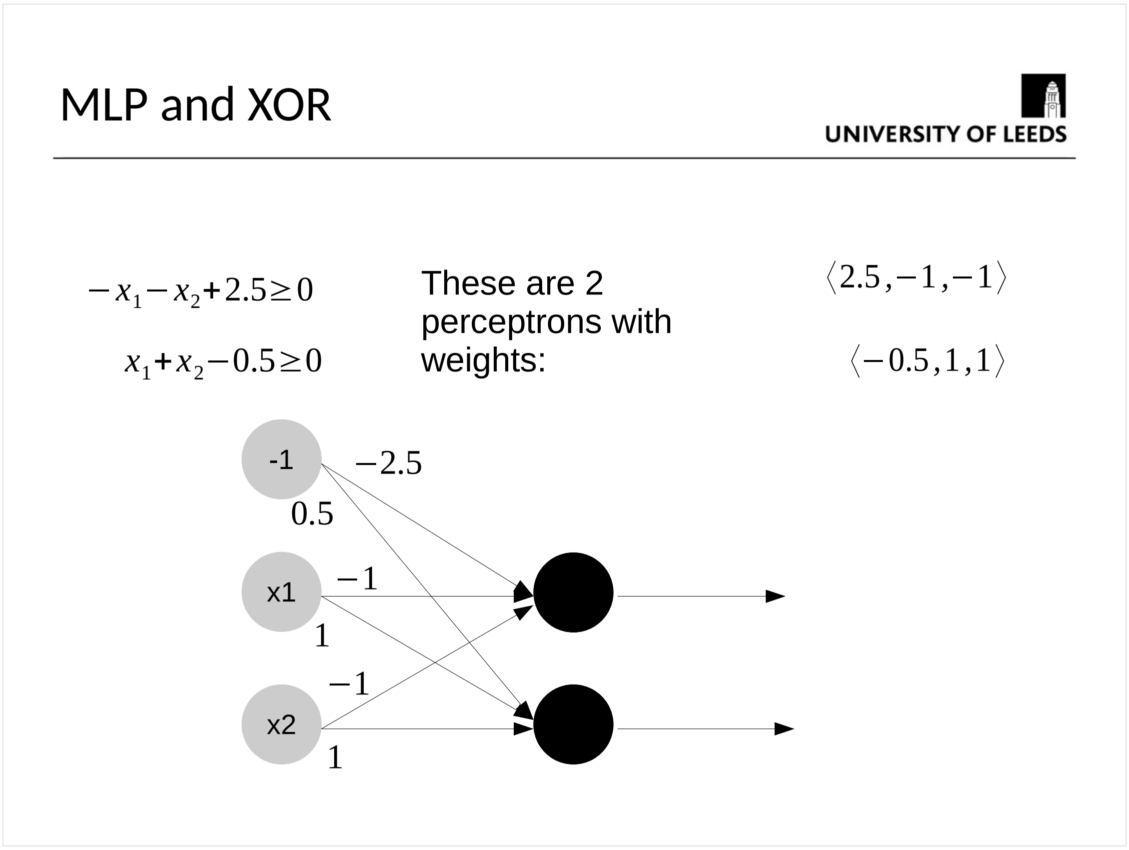
Possible solution:

$$-x_1 - x_2 + 2.5 = 0$$

$$w = \langle 2.5, -1, -1 \rangle$$

$$x_1 + x_2 - 0.5 = 0$$

$$w = \langle -0.5, 1, 1 \rangle$$



## MLP and XOR

UNIVERSITY OF LEEDS

$-x_1 - x_2 + 2.5 \geq 0$

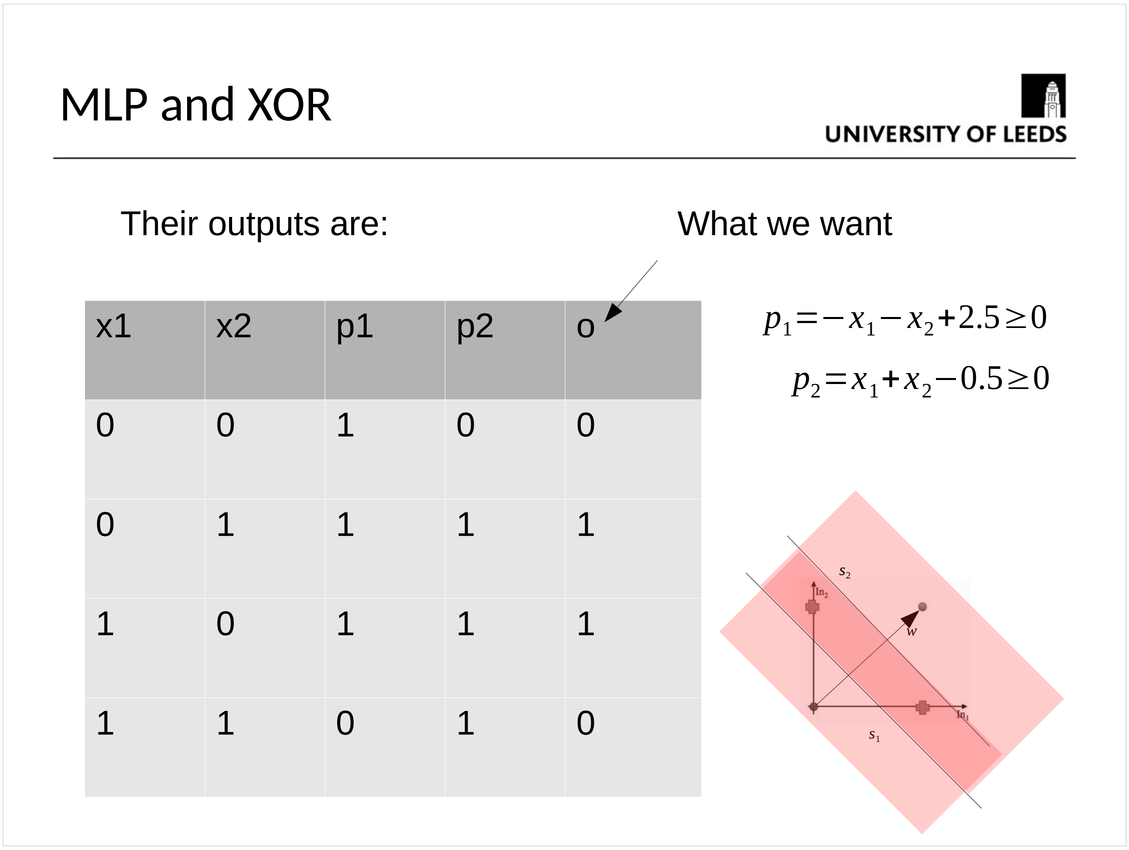
$x_1 + x_2 - 0.5 \geq 0$

These are 2 perceptrons with weights:

$\langle 2.5, -1, -1 \rangle$

$\langle -0.5, 1, 1 \rangle$

These are the two neurons that correspond to the two half-planes chosen. Now we need an additional layer to compute their intersection.



## MLP and XOR

UNIVERSITY OF LEEDS

Their outputs are:

What we want

| x1 | x2 | p1 | p2 | o |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 |

$$p_1 = -x_1 - x_2 + 2.5 \geq 0$$

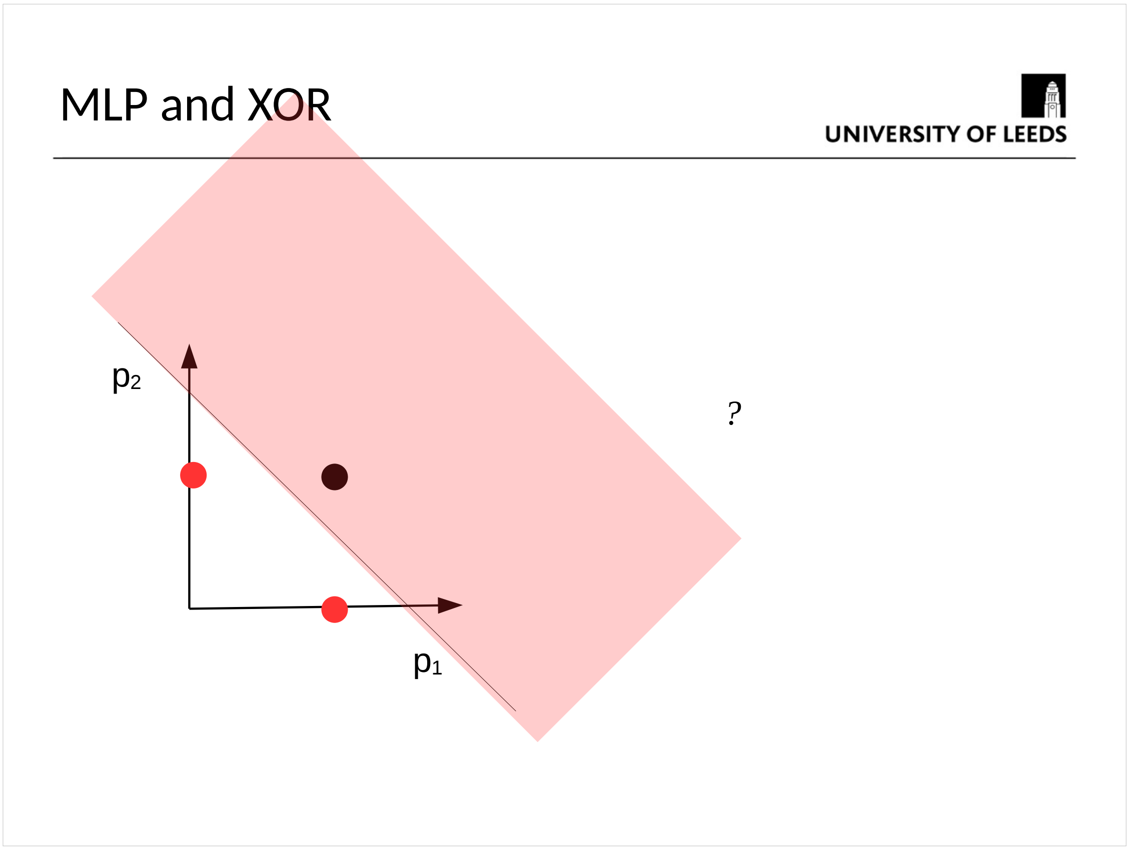
$$p_2 = x_1 + x_2 - 0.5 \geq 0$$

Let’s create a table to visualise what function we need. The first two columns are the components of the inputs.

The second and third column are the outputs of the two neurons we just created.

These two outputs, are in turn the input of the last layer. The last column is where we write our desired output, that is, 0 for input <0,0> and <1,1> and 1 for input <1,0> and <0,1>.

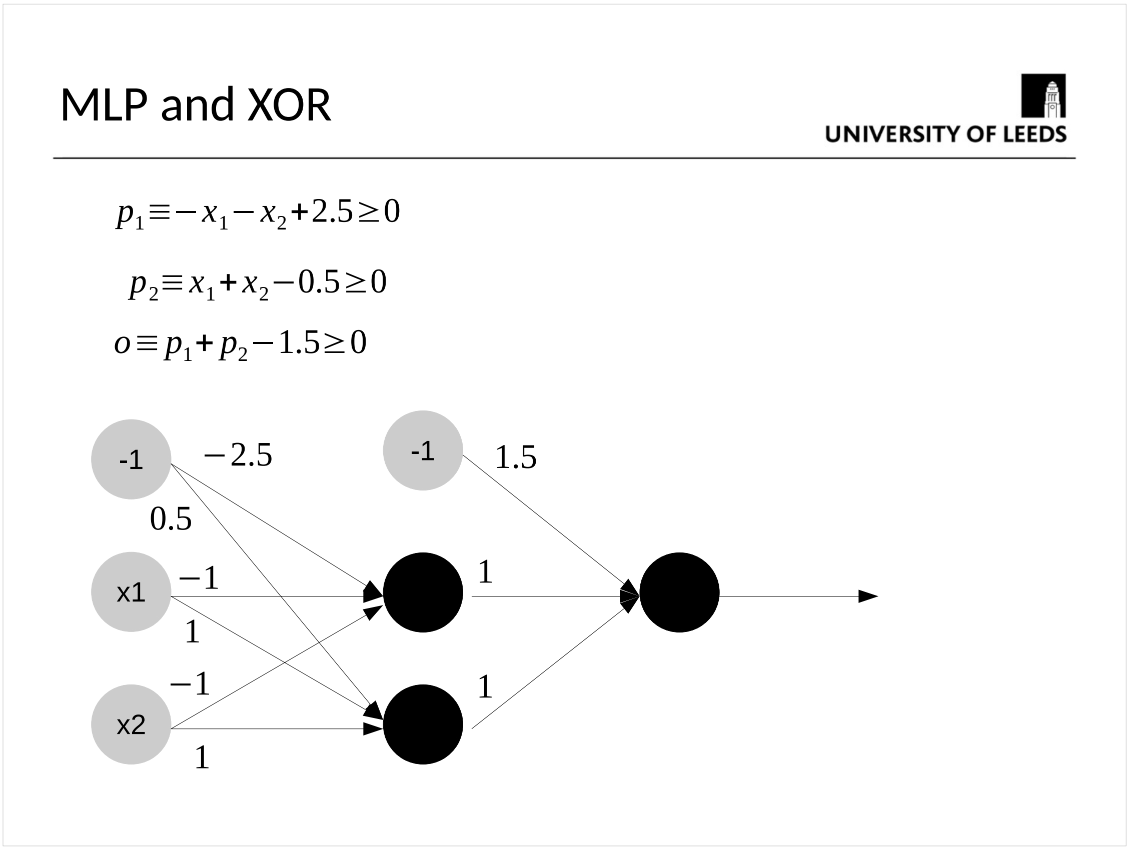
Now, let’s focus on the last three columns, where p1 and p2 are the inputs and o is the output. What function is that? It’s an AND. So the output layer of the MLP implements the AND function (which makes sense, since we wanted an intersection… what function do you think we would need if we wanted a union?)



## MLP and XOR

UNIVERSITY OF LEEDS

We do the same process again for the output layer, where we construct a perceptron that implements the AND function.



## MLP and XOR

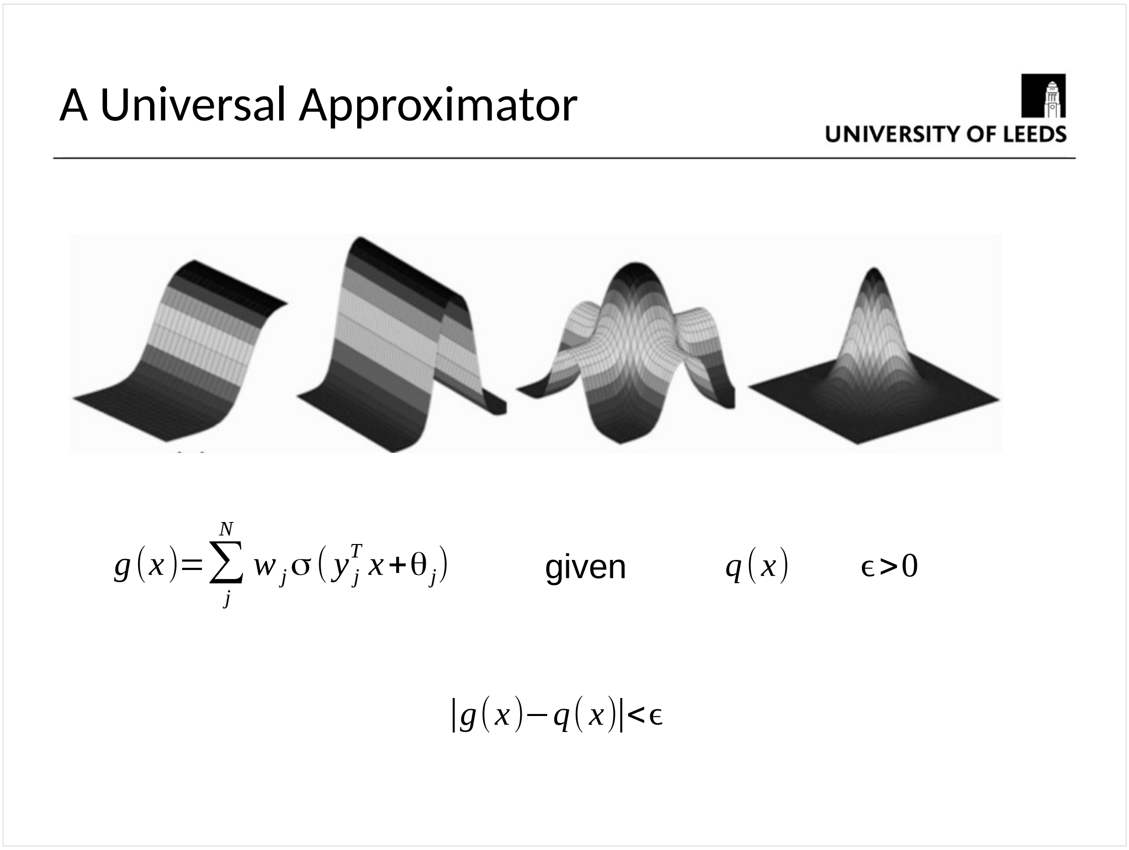
UNIVERSITY OF LEEDS

$$p_1 \equiv -x_1 - x_2 + 2.5 \geq 0$$

$$p_2 \equiv x_1 + x_2 - 0.5 \geq 0$$

$$o \equiv p_1 + p_2 - 1.5 \geq 0$$

This is the final MLP for the XOR!



# A Universal Approximator

UNIVERSITY OF LEEDS

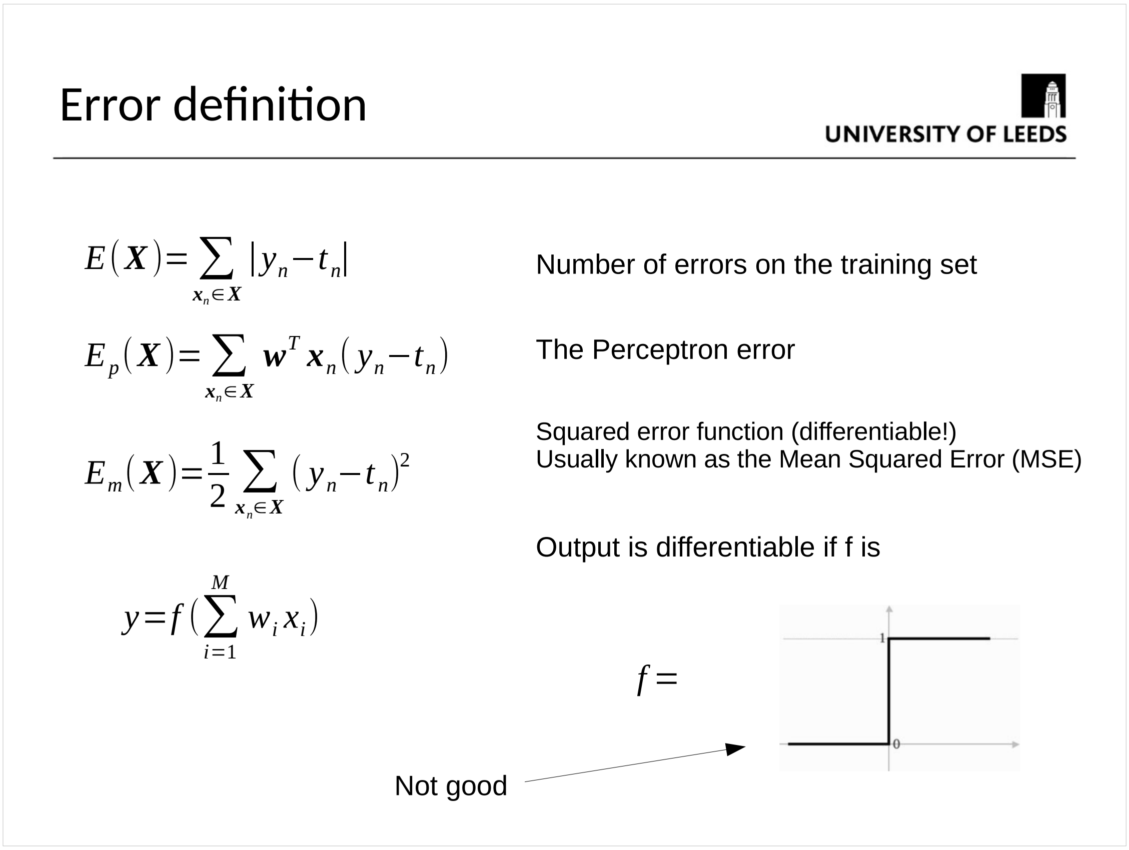
$$g(x)=\sum_{j}^{N} w_j \sigma(y_j^T x+\theta_j) \qquad \text{given} \qquad q(x) \qquad \epsilon>0$$

$$|g(x)-q(x)|<\epsilon$$

MLP is a universal function approximator, that is, it can represent any function.

From a theoretical point of view, this can be done with a single hidden layer. In practice that hidden layer would have to be incredibly large.

Recent development in neural networks (often referred to as “deep” learning), favour “deep” structures, where many layers can be used to create an hierarchical classification.



## Error definition

UNIVERSITY OF LEEDS

$$E(\boldsymbol{X}) = \sum_{\boldsymbol{x}_n \in \boldsymbol{X}} |y_n - t_n|$$

Number of errors on the training set

$$E_p(\boldsymbol{X}) = \sum_{\boldsymbol{x}_n \in \boldsymbol{X}} \boldsymbol{w}^T \boldsymbol{x}_n (y_n - t_n)$$

The Perceptron error

$$E_m(\boldsymbol{X}) = \frac{1}{2} \sum_{\boldsymbol{x}_n \in \boldsymbol{X}} (y_n - t_n)^2$$

Squared error function (differentiable!)
Usually known as the Mean Squared Error (MSE)

Output is differentiable if f is

$$y = f\left(\sum_{i=1}^{M} w_i x_i\right)$$

$f =$

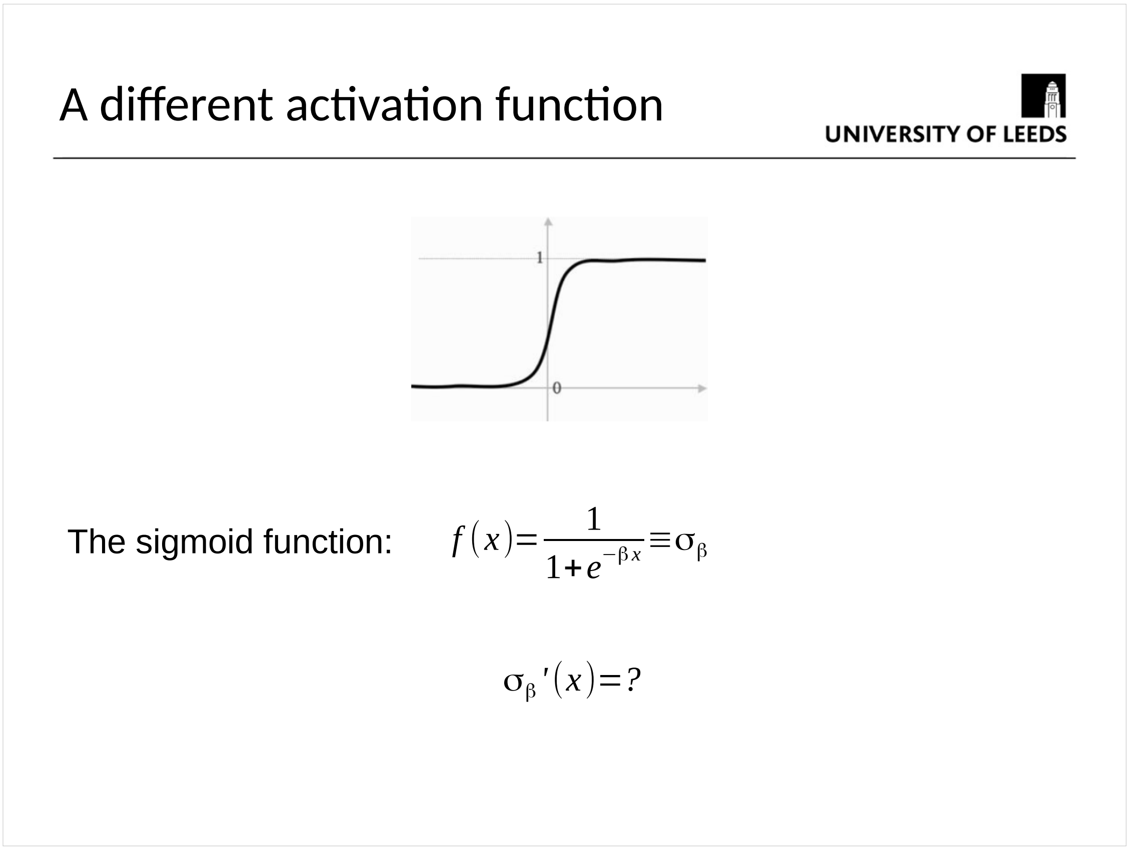
Not good

In the definition of the perceptron error, we took advantage of the fact that the perceptron implements a linear decision boundary, and we noted that the value of the linear function evaluated on any point was proportional to the distance of the point from the hyperplane.

The function implemented by the multi-layer perceptron, however, is not linear, and quite a bit more complicated. Consequently, we need a more general definition of the error.

Going back to the original idea of using the number of mistakes, we can do something similar, but differentiable, such as the mean squared error (MSE)

The MSE is differentiable (indeed, it's just a quadratic function) with respect to the parameters, as long as the activation function is also differentiable. So far we used a step function for the activation, which is not good in this case.



## A different activation function

UNIVERSITY OF LEEDS

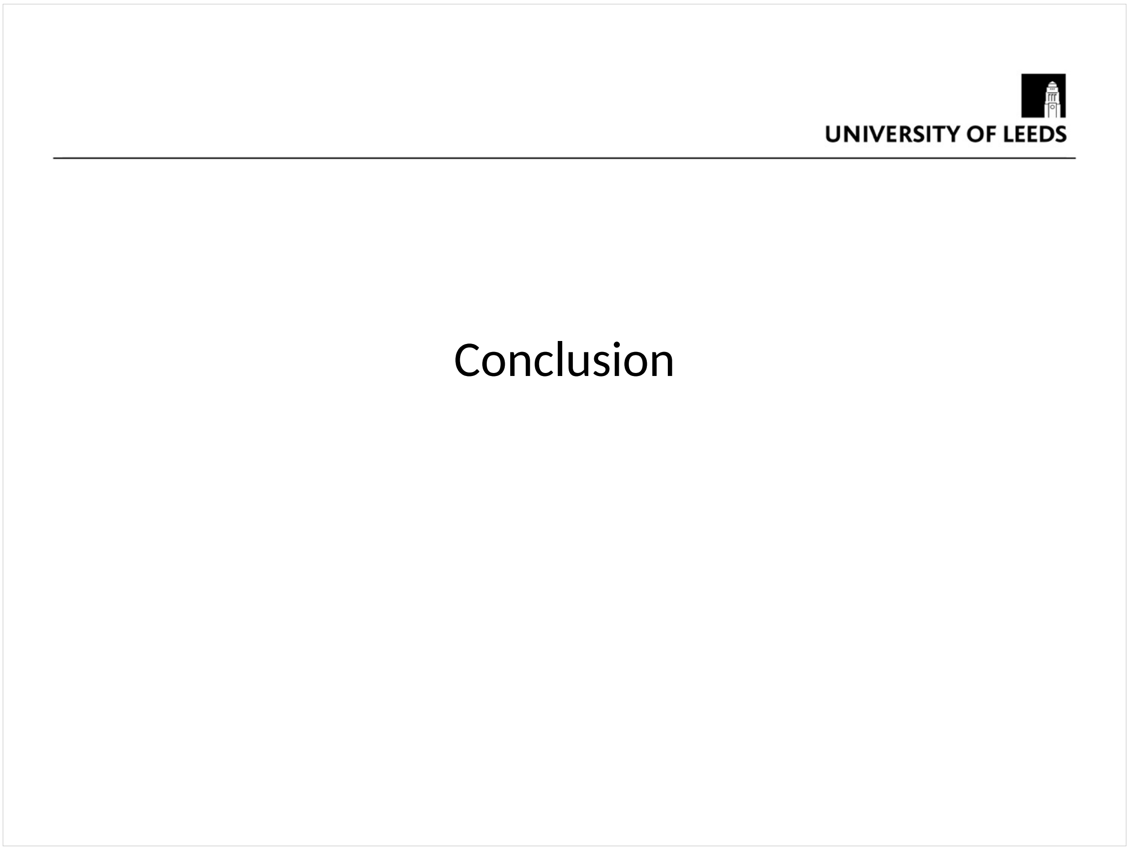
The sigmoid function: $f(x)=\frac{1}{1+e^{-\beta x}}\equiv\sigma_{\beta}$

$$\sigma_{\beta}'(x)=?$$

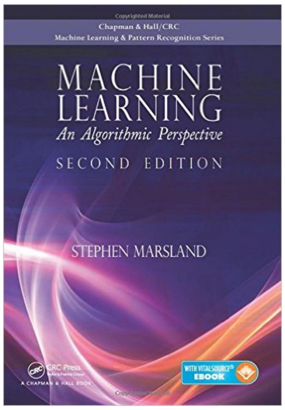
We substitute the step function with a function called sigmoid.

The sigmoid is similar to the step function in shape, but it’s rounded at the edges, so that it is differentiable everywhere.

What is its derivative?



# Conclusion


Chapman & Hall/CRC
Machine Learning & Pattern Recognition Series
MACHINE LEARNING
An Algorithmic Perspective
SECOND EDITION
STEPHEN MARSLAND
CRC Press
A CHAPMAN & HALL BOOK
WITH VITALSOURCE® EBOOK

Chapter 4