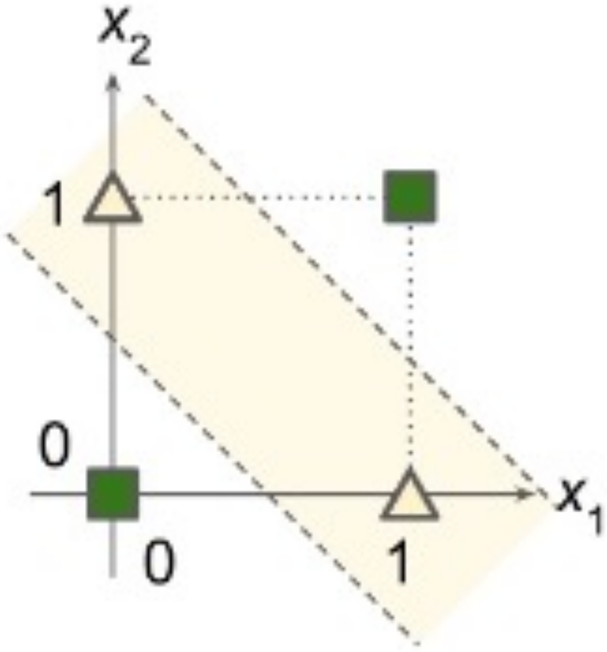
# The Multi-Layer Perceptron

Reference : Hands On Machine Learning, Aurélien Géron

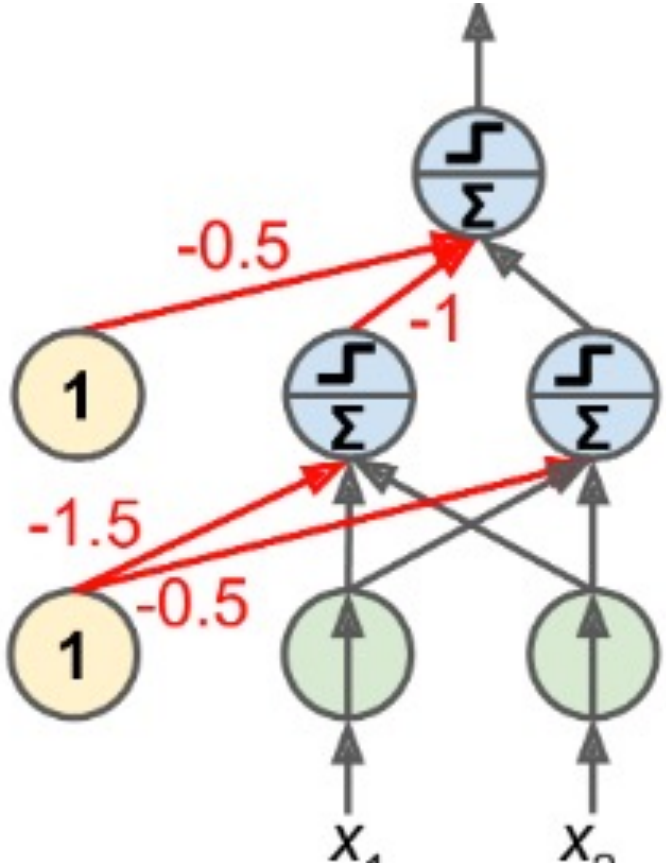
# Why Multiple Layers ?

- Certain trivial problems prove unsolvable for the simple perceptron.
- The XOR (*exclusive OR*) problem in particular is one of the most famous examples.

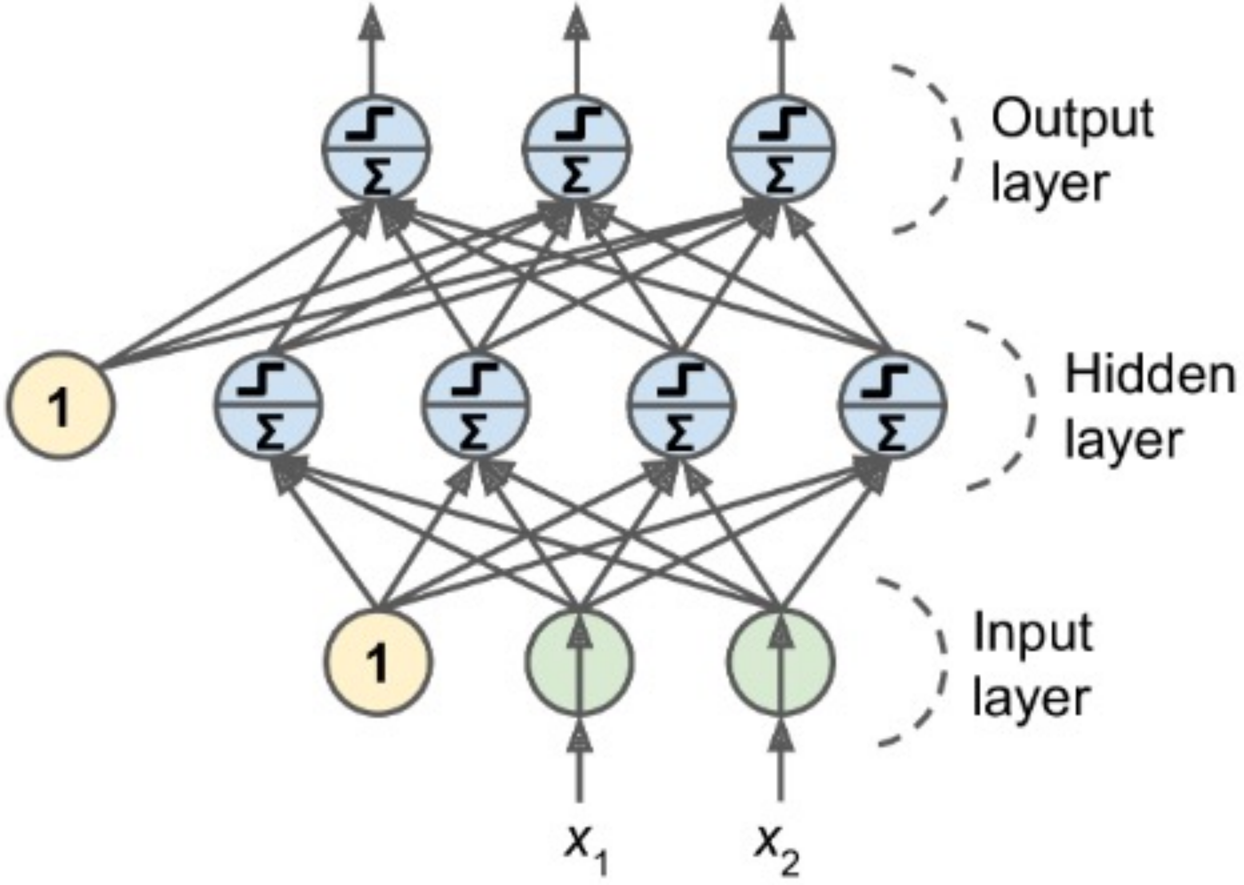


# Why Multiple Layers ?

- Turns out some of these limitations can be elminiated by **adding layers** :
- This network solves the XOR problem for example
- With inputs (0,0) or (1, 1) the network outputs 0, and with inputs (0, 1) or (1, 0) it outputs 1.
- All connections have a weight equal to 1, except the four connections where the weight is shown.



# Multi-Layer Perceptron



# Multi-Layer Perceptron

- An MLP is composed of one (passthrough) input layer, one or more layers of TLUs, called *hidden layers*, and one final layer of TLUs called the *output layer*

- The layers close to the input layer are usually called the *lower* layers, and the ones close to the outputs are usually called the *upper* layers.

- Every layer except the output layer includes a *bias neuron* and is *fully connected* to the next layer.

- The signal flows only in one direction (from the inputs to the outputs), so this architecture is an example of a **feedforward neural network** (FNN).

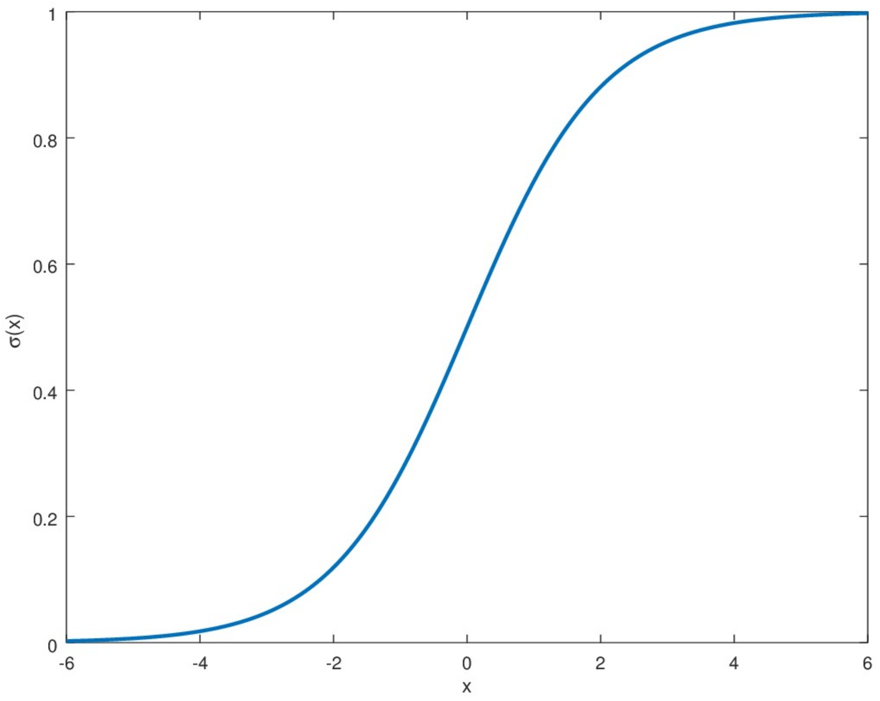
# How do we train this neural network ?

- 1986 : Rumelhart & al. Introduced the ***Backpropagation*** algorithm, still used today.
- Basically a version of Gradient Descent :
  - It's able to compute the gradient of the network's error with regards to *every single model parameter.*
  - In other words it finds out how each connection weight and each bias term should be tweaked in order to reduce the error.
  - Then a gradient descent step is performed (e.g. for the 1st neuron of the 1st layer) :

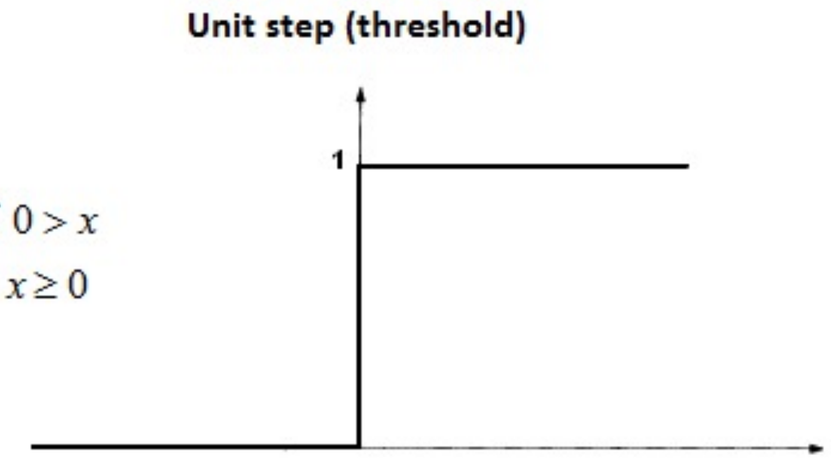
$$weights_{N=1,\,L=1} \;=\; weights_{N=1,L=1} - learningRate \;*\; \frac{\partial Cost}{\partial weights_{N=1,L=1}}$$

# Activation functions

- One key changed was made to the neurons for backprop to work:
- The *step* function was replaced with the logistic function.
- Step function contains only flat segments vs. Sigmoid/logistic function which has a non-zero derivative everywhere :



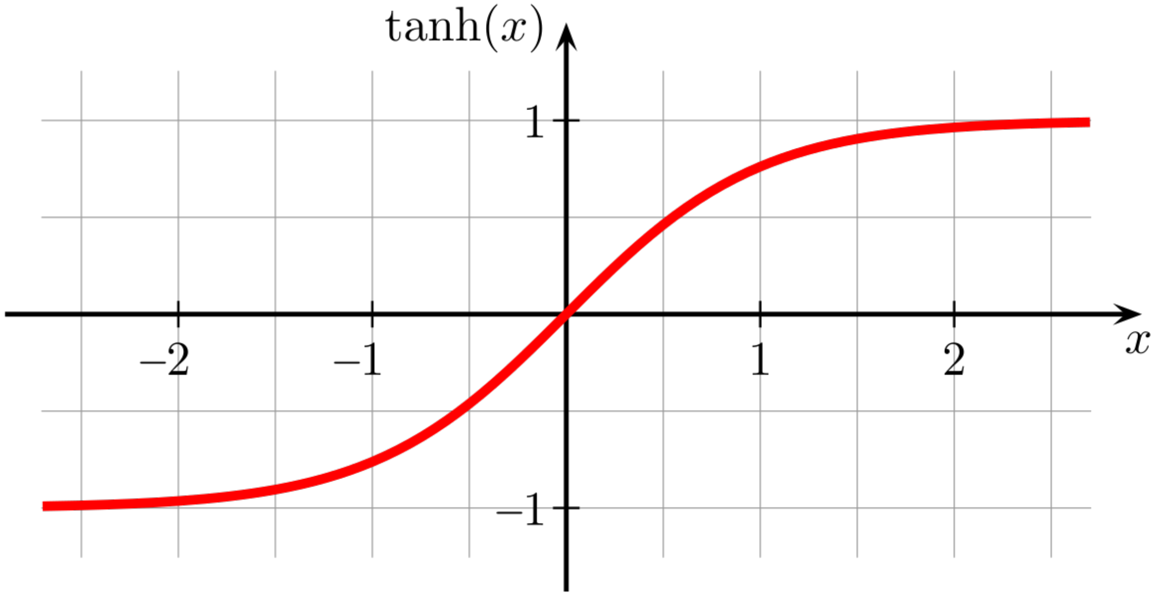

Sigmoid




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

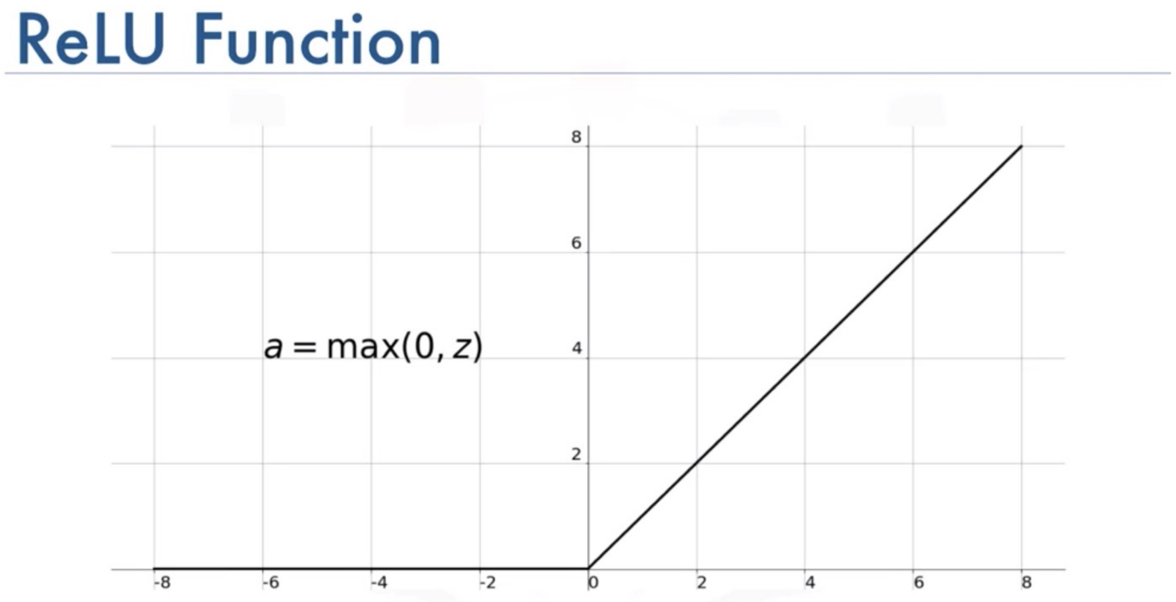
# Other Activation functions

- *The hyperbolic tangent function* tanh(z) = 2σ(2z) – 1
  Just like the logistic function it is S-shaped, continuous, and differentiable, but its output value ranges from –1 to 1 (instead of 0 to 1 in the case of the logistic function)



# Other Activation functions

- The Rectified Linear Unit function: ReLU(z) = max(0, z)
  It is continuous but unfortunately not differentiable at z = 0 (the slope changes abruptly, which can make Gradient Descent bounce around), and its derivative is 0 for z < 0.
- However, in practice it works very well and has the advantage of being fast to compute.



# Why do we need activation functions ?

- Why does a neuron not just output the weighted some of its inputs and pass it on to the next neurons ?

- Why is a **non-linear** activation function needed ?

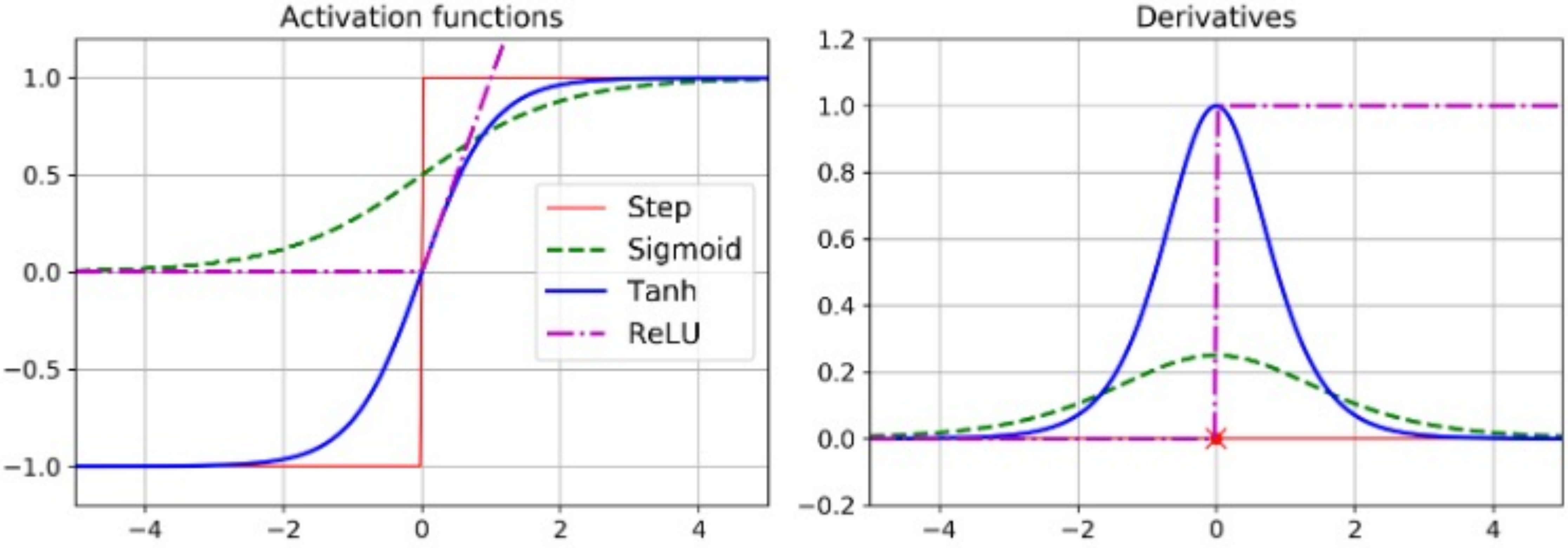
# Why do we need activation functions ?

- => If you chain several linear transformations, you get 1 linear transformation, which doesn’t allow us to solve complex problems.
- If $f(x) = 2x + 3$ and $g(x) = 5x - 1$, then chaining these two linear functions gives us another linear function:

$$f(g(x)) = 2(5x - 1) + 3 = \mathbf{10x + 1}$$

- So if there isn’t a non-linear function betwen the layers, then even a deep stack of layers is equivalent to a single layer…

# Activation Functions and their derivatives



# For visual/illustrated explanations…

- 3Bue1Brown’s videos on neural nets
- Jay Alammar’s blogposts on neural nets
- Why neural nets can learn almost anything
- To play around with a neural net
- The Absolutely Simplest Neural Network Backpropagation Example

# Classification MLPs

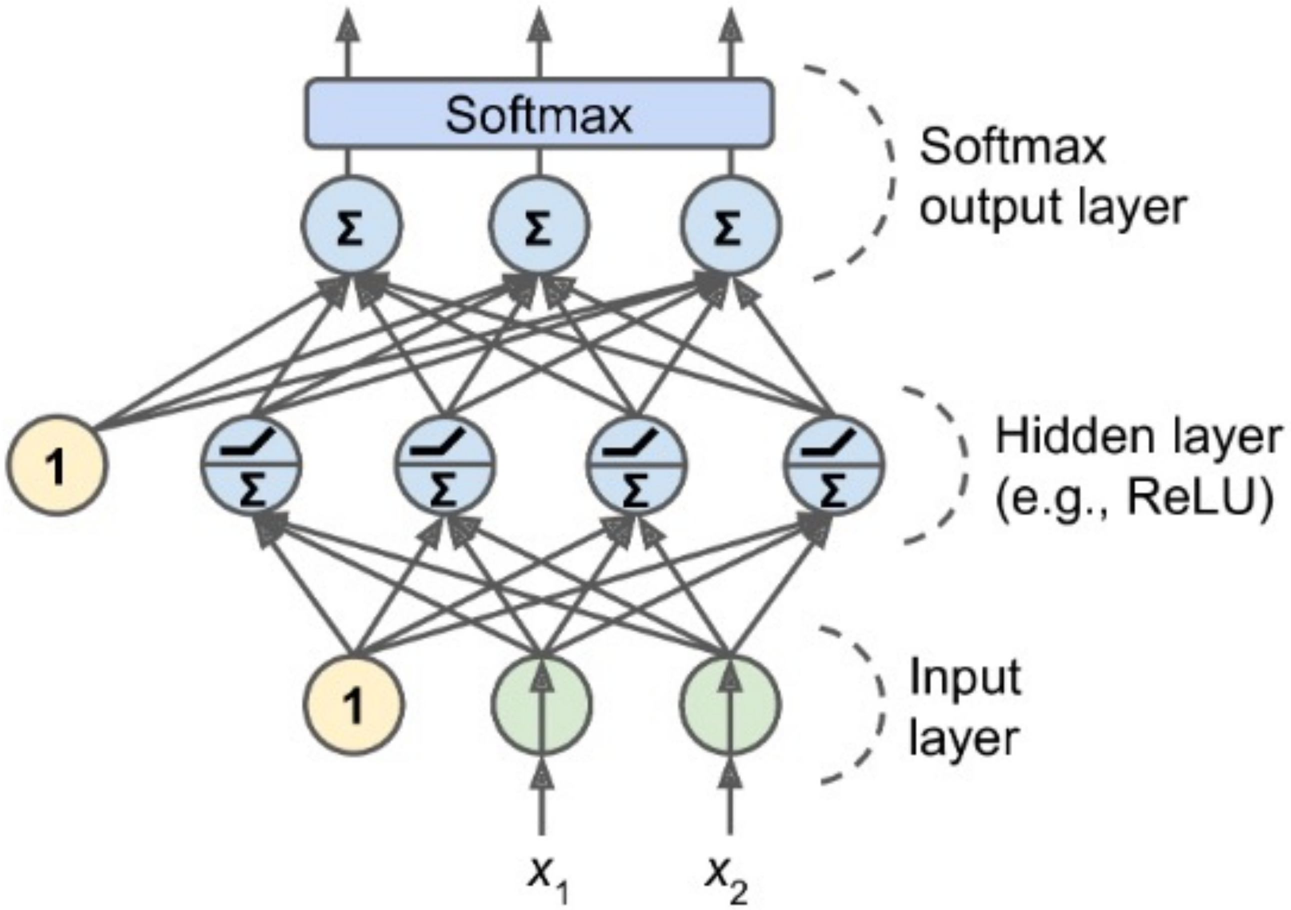
- **Binary Classifiaction** :
- 1 single output neuron using the logistic activation function.
- It outputs a number between 0 and 1, which you can interpret as the estimated probability of the positive class.
- Obviously, the estimated probability of the negative class is equal to one minus that number.

# Classification MLPs

- **Multiclass Classification:**
  - If each example fed into the network can only belong to a single class (the classes are mutually exclusive — there can't be a combination of classes, only 1)
  - Then you need 1 output neuron *per class* and you need to use the ***softmax*** activation function for the whole output layer.
  - This function ensures the probabilities produced by each output neuron are between 0 and 1, and add up to 1.

  - Softmax wikipedia
  - Softmax vs argmax

# Multiclass Classification MLP



# Loss function

- We are predicting a *probability distribution*, hence we need to use a function which can output a small value when the probaility distributions are very similar and a large value otherwise.

- This is called the *cross-entropy* and originates from a field called *information theory*.

- Videos on :
  - Entropy
  - Cross-entropy
  - Entropy, Cross-Entropy and KL Divergence

# Cross entropy formula

- See this article for an explanation of how cross-entropy works for classification.
- The loss (or cost) for 1 single example would be defined as :

$$J(\mathbf{\Theta}) = -\sum_{k=1}^{K} y_k \log(\hat{p}_k)$$

$y_k$ is the target probability that the example belongs to class $k$. This is generally equal to 0 or 1.

$\Rightarrow$ For each class, we compute the highlighted part of the formula and then sum.

$\Rightarrow$ When there are only 2 classes, this loss function is actually equivalent to the loss for Logistic Regression.

# Typical Network Architecture for Classification

| Hyperparameter | Binary Classification | Multiclass Classification (also works for 2 classses) |
|---|---|---|
| # input neurons | 1 per feature | Idem |
| # hidden layers | Depends on the problem. Typically 1-5, but as many as you want is possible. | Idem |
| # neurons per hidden layer | Depends on the problem. Typically 10-100. | Idem |
| # output neurons | 1 | 1 per class |
| Output Layer Activation | Sigmoid | Softmax |
| Loss Function | Binary Cross-Entropy | Cross-Entropy |