Neural Networks

Neural networks

Goal: Build an intuition into how neural networks works and get familiar with the associated vocabulary.

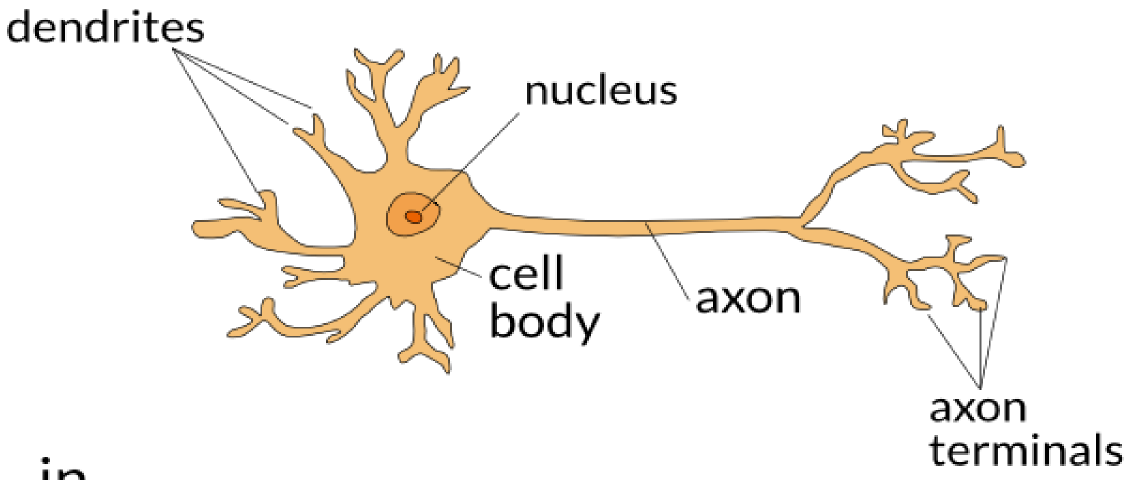
Program:

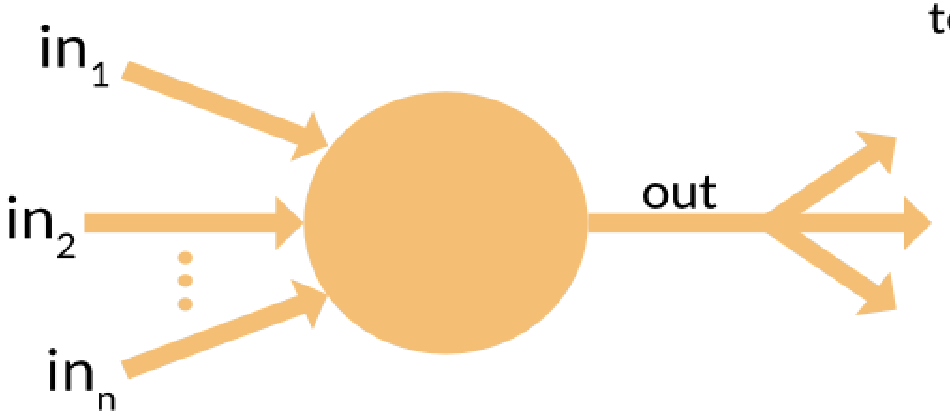
- Intutiton to neural neworks
- The anatomy of a network
- The training process (gradient descent)
- Backpropagation

GO DATA DRIVEN

Intuition

Neural Networks Intuition

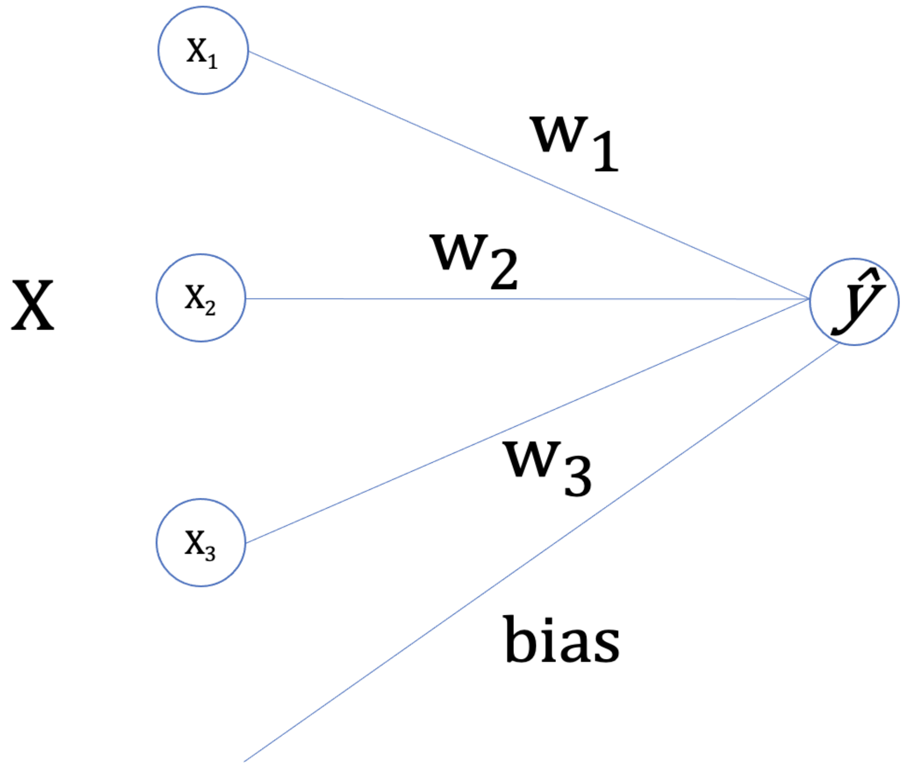




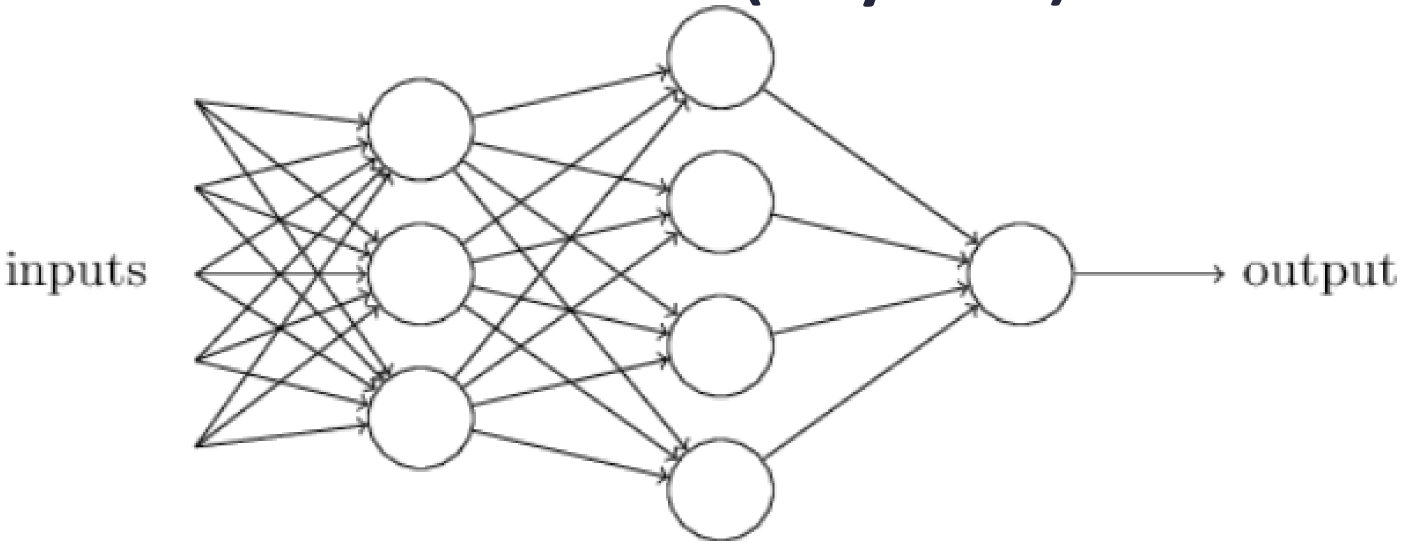
Anatomy of the network

Anatomy of the network (neurons)

Inputs (vector $\boldsymbol{x}$) are combined
with weights (vector $\boldsymbol{w}$),
a bias (value $\boldsymbol{b}$)
and a non-linear activation function ($\boldsymbol{\sigma}$)
to produce an output value,
i.e. output = $\boldsymbol{\sigma}(\boldsymbol{w}^T\boldsymbol{x}+\boldsymbol{b})$



Anatomy of the network (layers)



Layer: column of neurons stacked together that can receive the same inputs.

Hidden layer: intermediate layers between inputs and outputs.

Deep neural network: a neural network that contains many hidden layers, and can therefore provide solutions to more complicated and subtle decision problems.

GO DATA DRIVEN

The training process

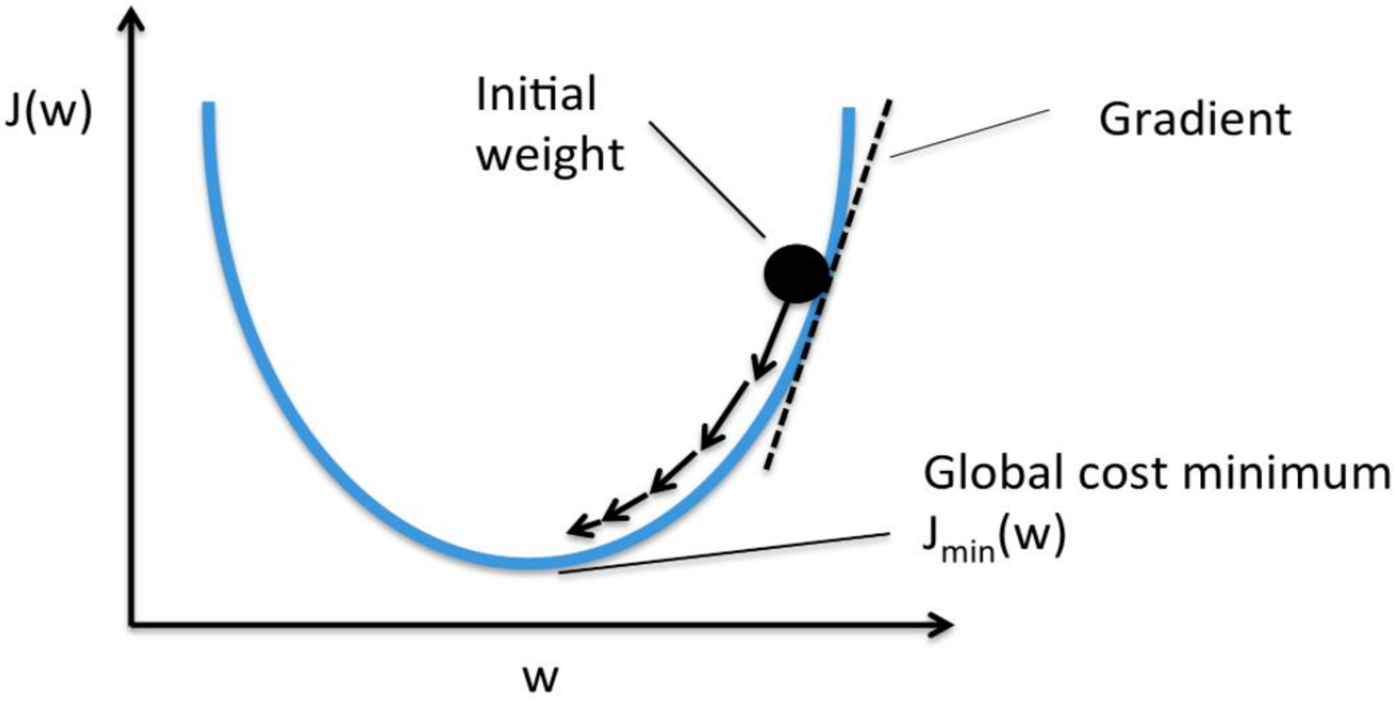
Training

- Training is the process of tuning the weights $\boldsymbol{w}$ in a network.
- Usually, to train a network we require input data $\boldsymbol{X}$ and corresponding target labels $\boldsymbol{y}$.
- We attempt to use the network f to predict make predictions, i.e. $\hat{y}=f(\boldsymbol{X},\boldsymbol{w})$.
- The weights should minimise a cost or loss function J, e.g.
  - $J = \boldsymbol{y} - \hat{y}$
  - $J = \frac{1}{2}(\boldsymbol{y} - \hat{y})^2$

Gradient descent

$$w' = w - \eta \frac{\partial J(w)}{\partial w}$$

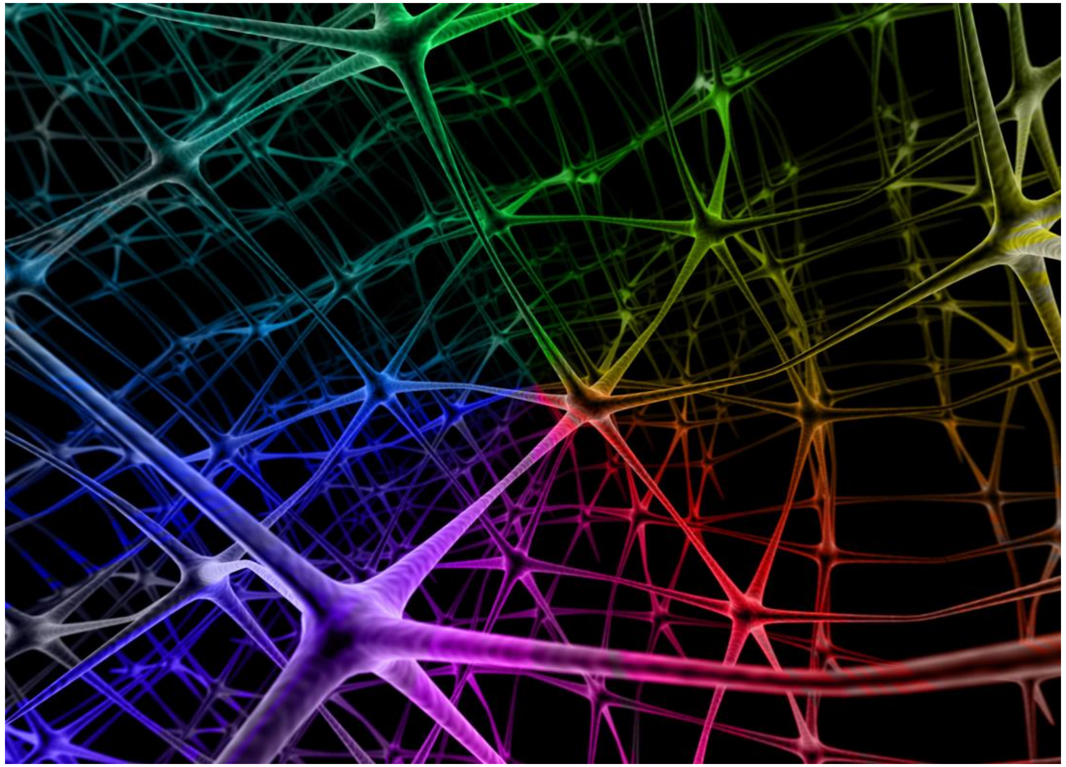
A process of following the gradients of the error function towards a minimum value.



Backpropogation

Backpropagation

We'll now examine backpropagation, a fast algorithm for computing the weight gradients which is based on the chain-rule.



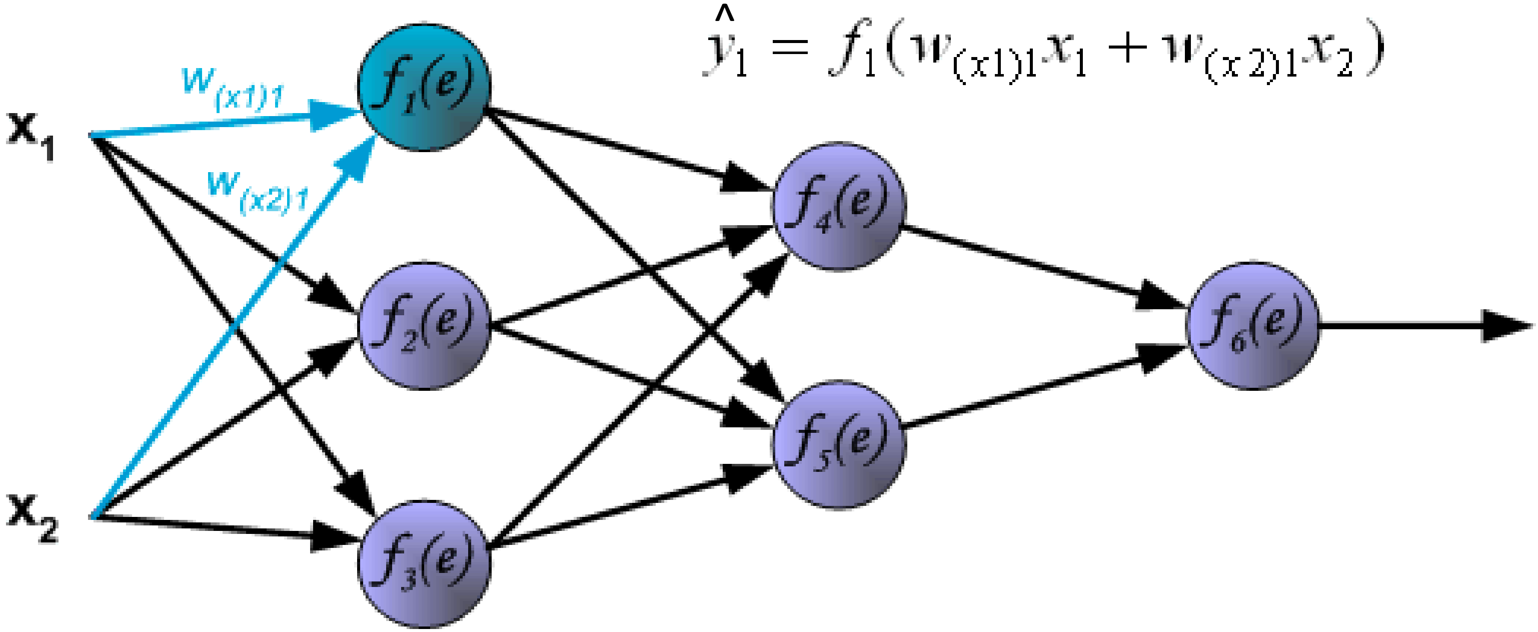
GO DATA DRIVEN

Weight updates

- **We want to find the best weights!**

- **Update weights using gradient descent**

- $w'_{(x1)1} = w_{(x1)1} - \eta \frac{\partial loss}{\partial w_{(x1)1}}$



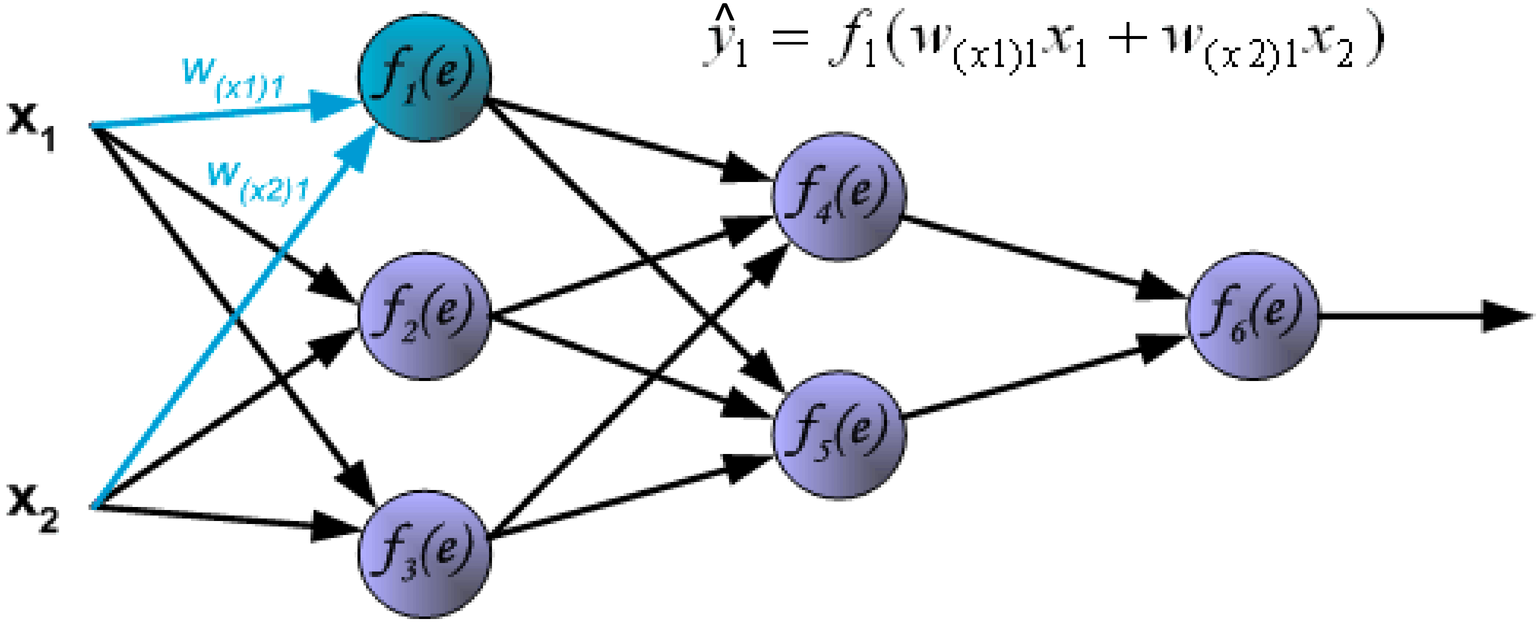
Weight updates

- We find $\frac{\partial loss}{\partial w_{(x1)1}}$ using the chain rule

We need to compute these gradients

- $\frac{\partial loss}{\partial w_{(x1)1}} = \frac{\partial loss}{\partial f_1(e)} \frac{\partial f_1(e)}{\partial e} \frac{\partial e}{\partial w_{(x1)1}}$, where e = $w_{(x1)1}x_1 + w_{(x2)1}x_2$

$$\hat{y}_1 = f_1(w_{(x1)1}x_1 + w_{(x2)1}x_2)$$



Weight updates

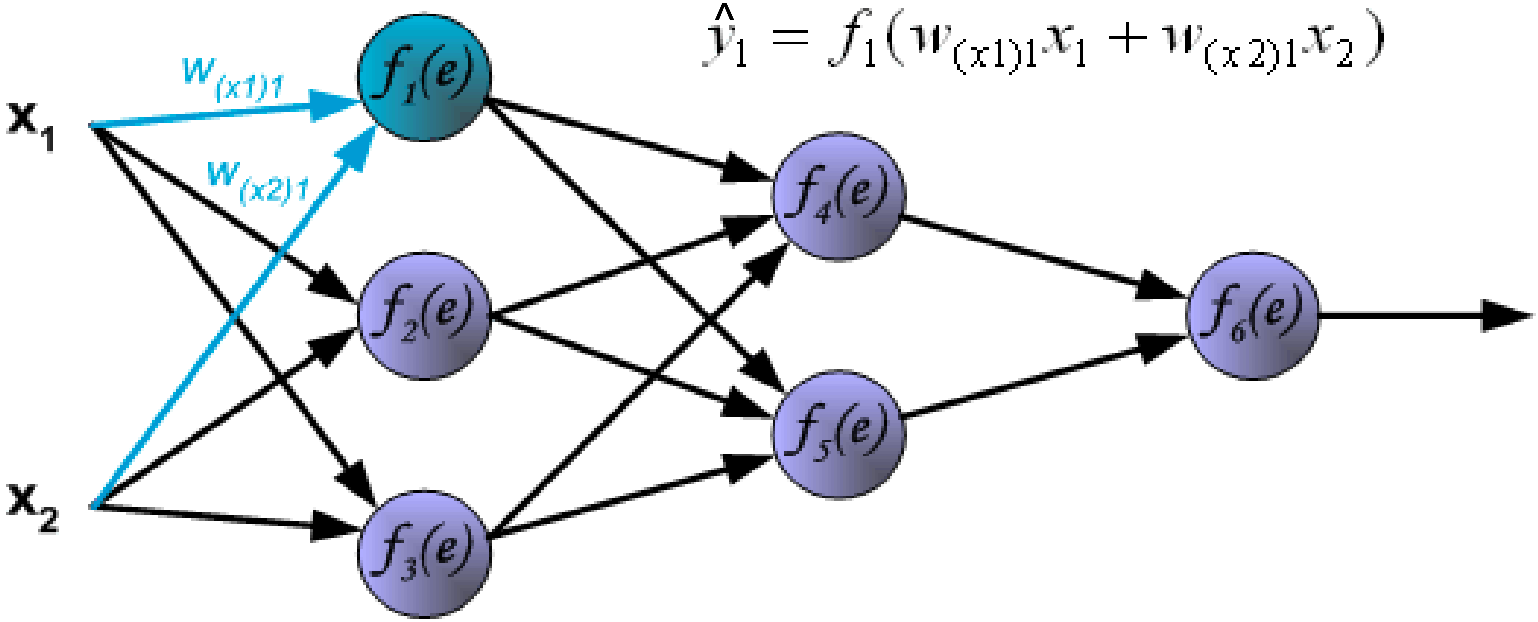
- **We find $\frac{\partial loss}{\partial w_{(x1)1}}$ using the chain rule**

We need to compute these gradients

- $\frac{\partial loss}{\partial w_{(x1)1}} = \frac{\partial loss}{\partial f_1(e)} \frac{\partial f_1(e)}{\partial e} \frac{\partial e}{\partial w_{(x1)1}}$, where $e = w_{(x1)1}x_1 + w_{(x2)1}x_2$

$$\hat{y}_1 = f_1(w_{(x1)1}x_1 + w_{(x2)1}x_2)$$

$$\frac{\partial e}{\partial w_{(x1)1}} = x_1$$



Weight updates

- **We find $\frac{\partial loss}{\partial w_{(x1)1}}$ using the chain rule**

We need to compute these gradients

- $\frac{\partial loss}{\partial w_{(x1)1}} = \frac{\partial loss}{\partial f_1(e)} \frac{\partial f_1(e)}{\partial e} x_1$

- This is the derivative of our activation function, typically we choose one that is easy to compute, e.g. the sigmoid

$$\boldsymbol{\sigma}'(\mathbf{e}) = \boldsymbol{\sigma}(\mathbf{e})(1 - \boldsymbol{\sigma}(\mathbf{e}))$$

GO DATA DRIVEN

Weight updates

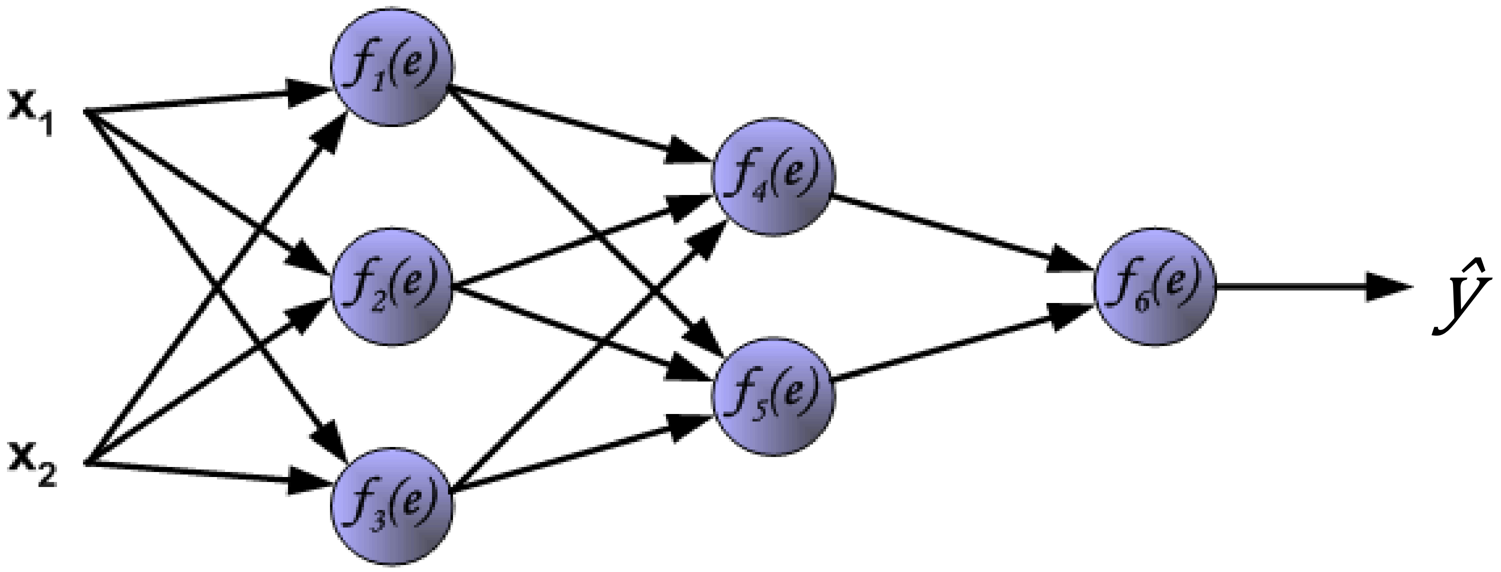
- **We find $\frac{\partial loss}{\partial w_{(x1)1}}$ using the chain rule**

We need to compute these gradients

- $\frac{\partial loss}{\partial w_{(x1)1}} = \frac{\partial loss}{\partial f_1(e)} \frac{\partial f_1(e)}{\partial e} x_1$

- We compute this derivative by backpropagating the error through the network.

Epoch



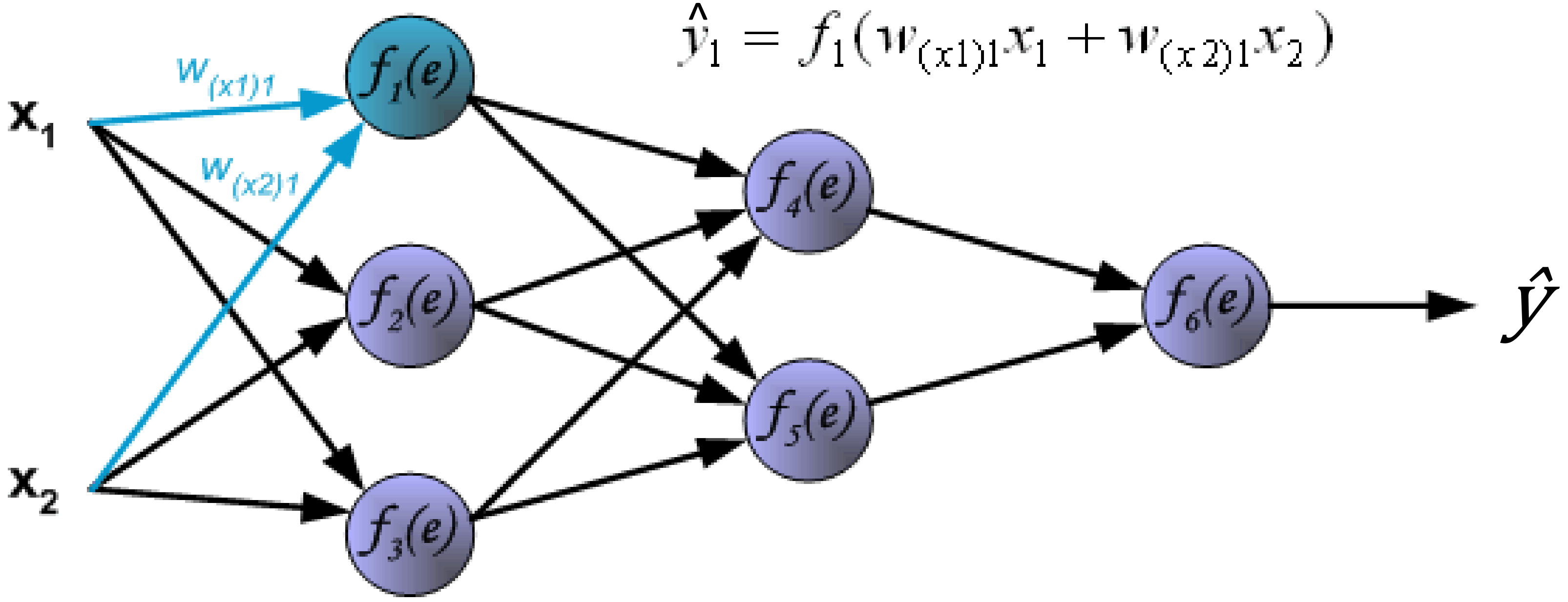
- Forward pass: a data sample is passed forward through the network to determine a prediction.
- Backward pass: recursively compute the error backwards from the last layer following the chain-rule and update the weights w.r.t. the known target output.
- Epoch: training the neural network with all the training data for one cycle.

GO DATA DRIVEN

Forward pass

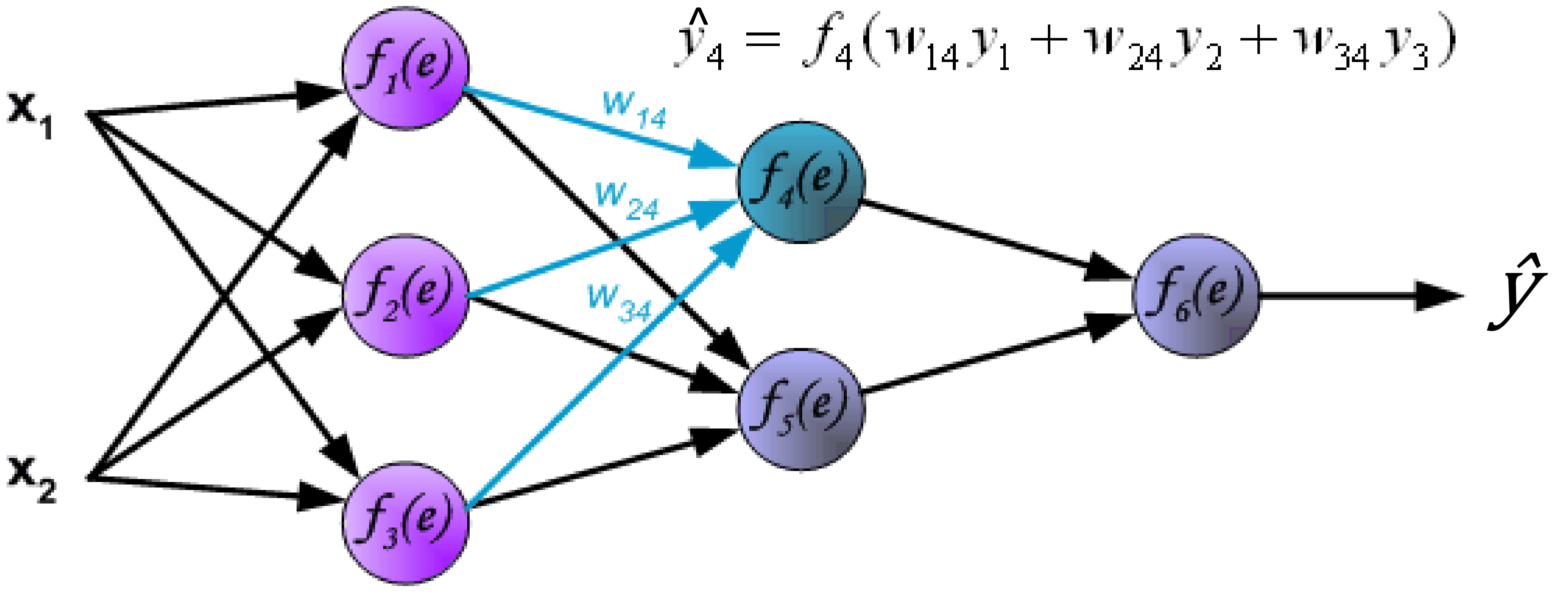
Feed data through the network.

$$\hat{y}_1 = f_1(w_{(x1)1} x_1 + w_{(x2)1} x_2)$$



Forward pass

Feed data through the network.



Compute the error

For example,

$$\text{loss} = \frac{1}{2}(\boldsymbol{y} - \hat{y})^2$$

Then we can compute the error rate $\delta = \frac{\partial\, loss}{\partial \hat{y}}$

$$\delta = \frac{\partial\, loss}{\partial \hat{y}} = -(\boldsymbol{y} - \hat{y})$$

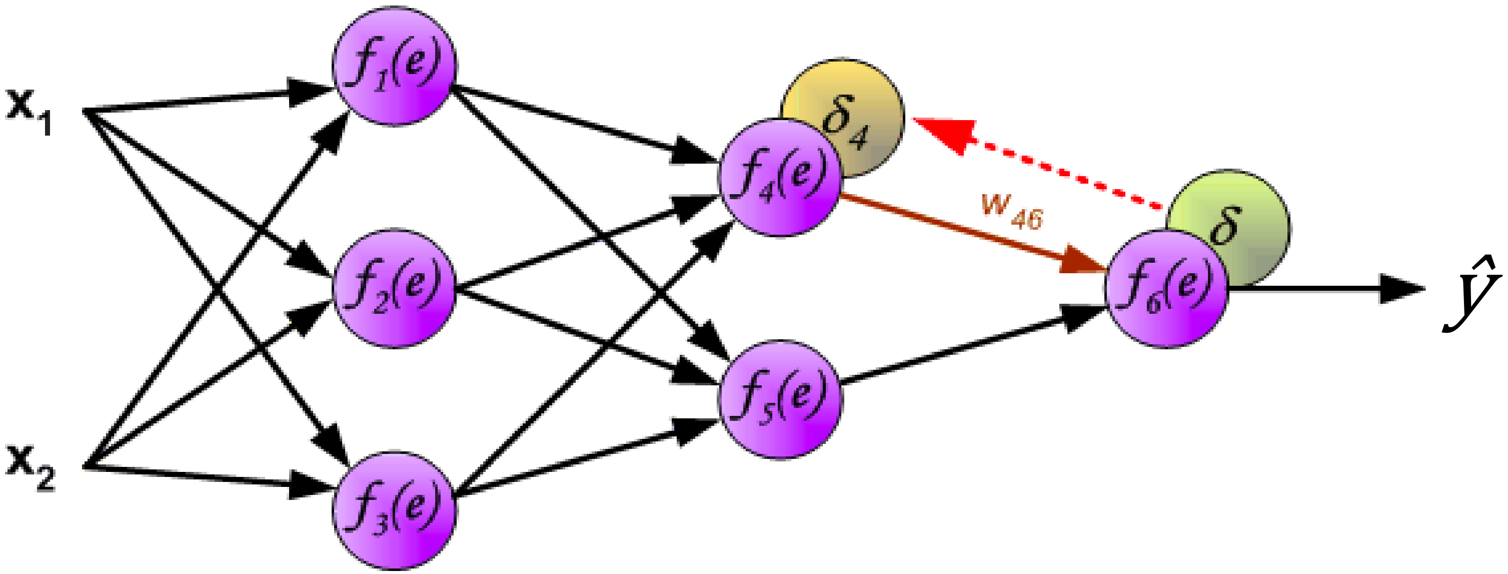
GO DATA DRIVEN

Backpropgation

Local error contribution

$$\delta_4 = \frac{\partial\, loss}{\partial f_4} = \frac{\partial\, loss}{\partial f_6}\frac{\partial f_6(e)}{\partial e}\frac{\partial e}{\partial f_4} = \delta\,\frac{\partial f_6(e)}{\partial e}\,w_4$$

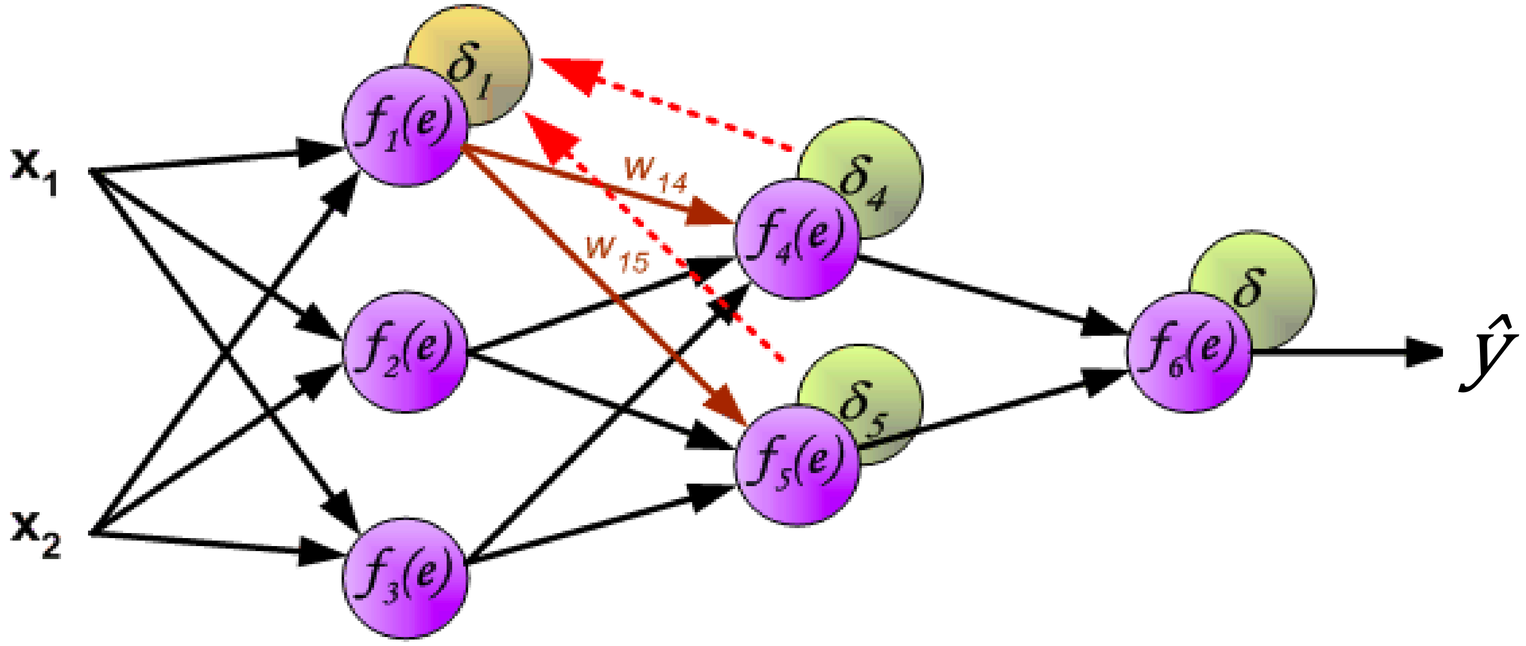
where $\hat{y} = f_6$ & $e = w_{46} f_4 + w_{56} f_6$



Backpropgation

Local error contribution

$$\delta_1 = \frac{\partial\ loss}{\partial f_1} = \frac{\partial\ loss}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial e} \frac{\partial e}{\partial f_4} = \delta_4 \frac{\partial f_4(e)}{\partial e} w_{14} + \delta_5 \frac{\partial f_5(e)}{\partial e} w_{15}$$

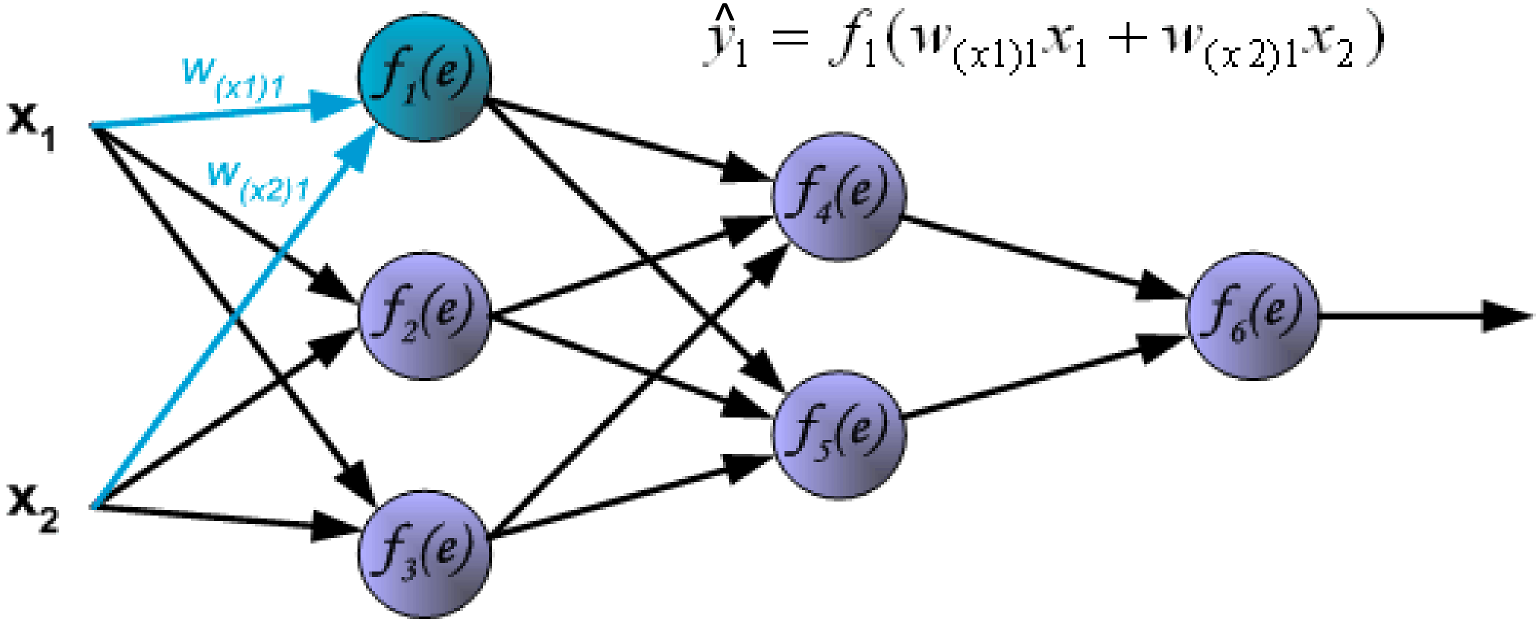


GO DATA DRIVEN

Backpropagation

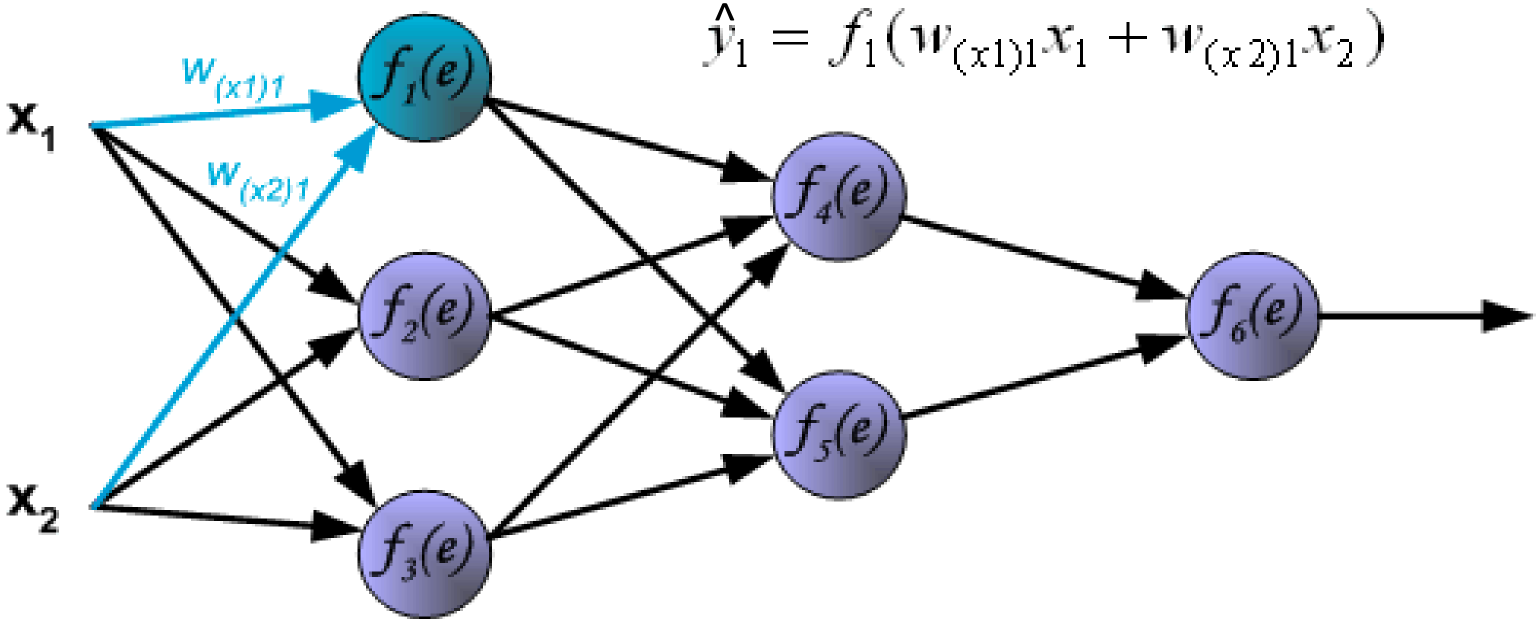
Previously we saw that

$$\frac{\partial loss}{\partial w_{(x1)1}} = \frac{\partial loss}{\partial f_1(e)} \frac{\partial f_1(e)}{\partial e} w_{(x1)1}$$



Backpropagation

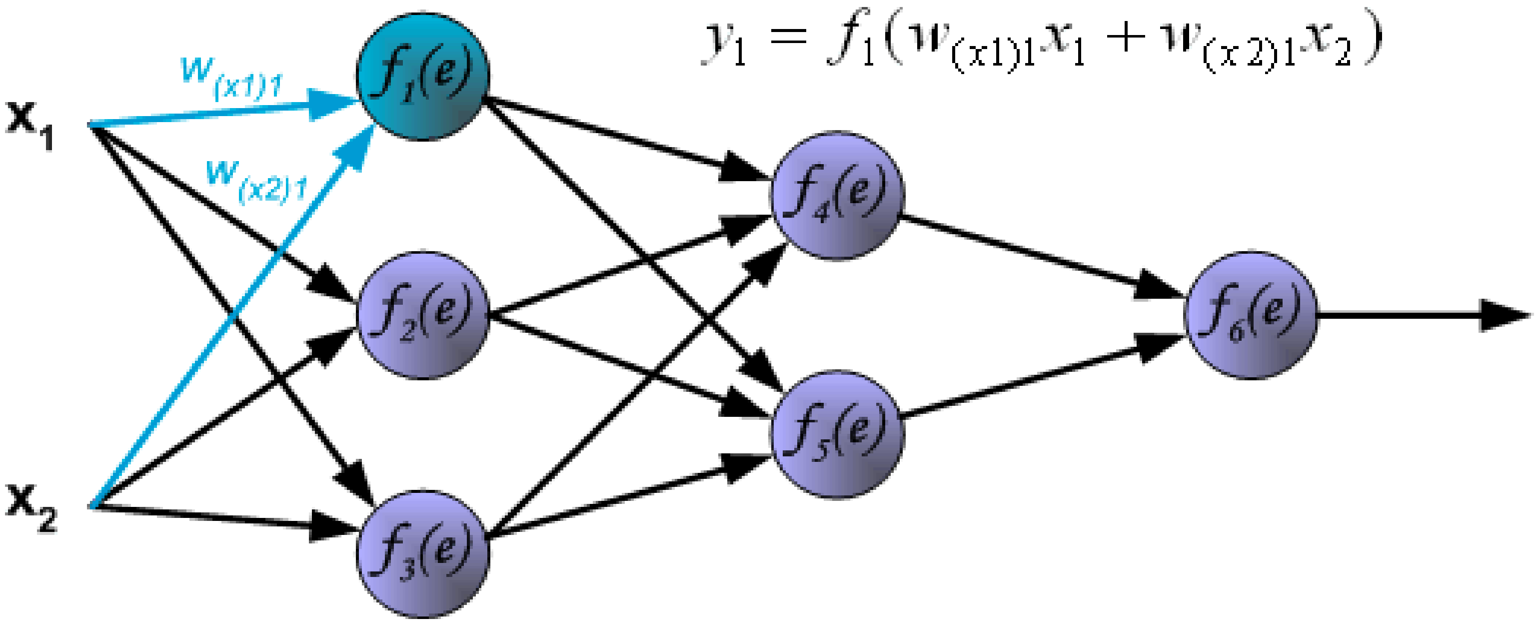
Thus, $\frac{\partial loss}{\partial w_{(x1)1}} = \delta_1 \frac{\partial f_1(e)}{\partial e}\ x_1$



Backpropagation

We can then optimise weight using gradient descent, e.g.

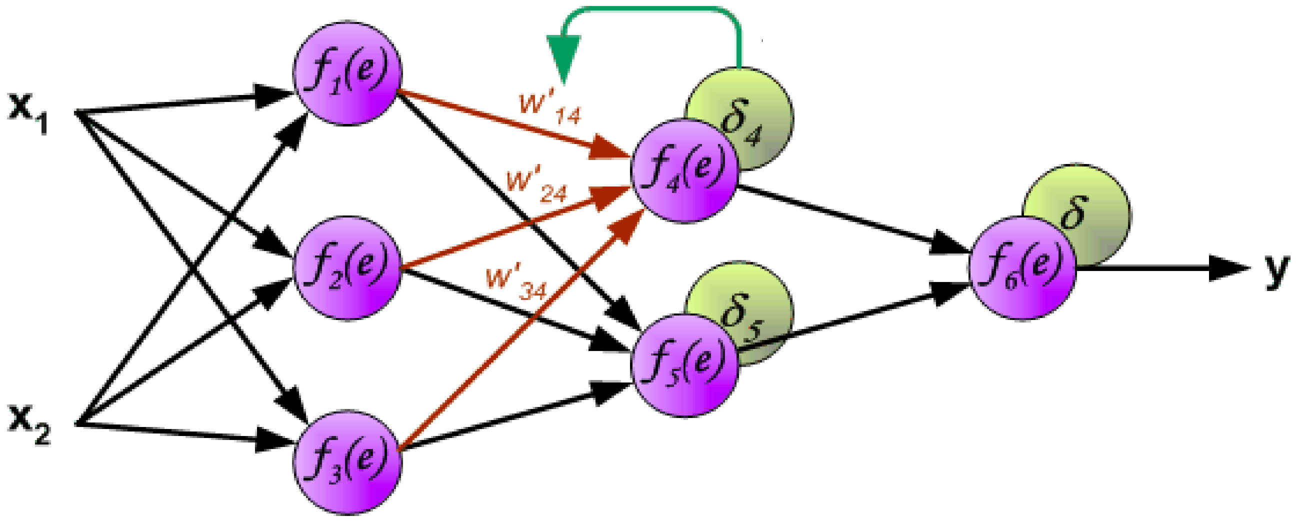
$$w'_{(x1)1} = w_{(x1)1} - \eta \delta_1 \frac{\partial f_1(e)}{\partial e} x_1$$



Backpropagation

We can then optimise weight using gradient descent, e.g.

$$w'_{14} = w_{14} - \eta \delta_4 \frac{\partial f_4(e)}{\partial e} f_1$$



Summary

Summary

In this section we have covered

- Intutiton to neural neworks
- The anatomy of a network
- The training process (gradient descent)
- Backpropagation

In the next exercise we will build a MLP to solve the XOR problem

XOR perceptron

XOR

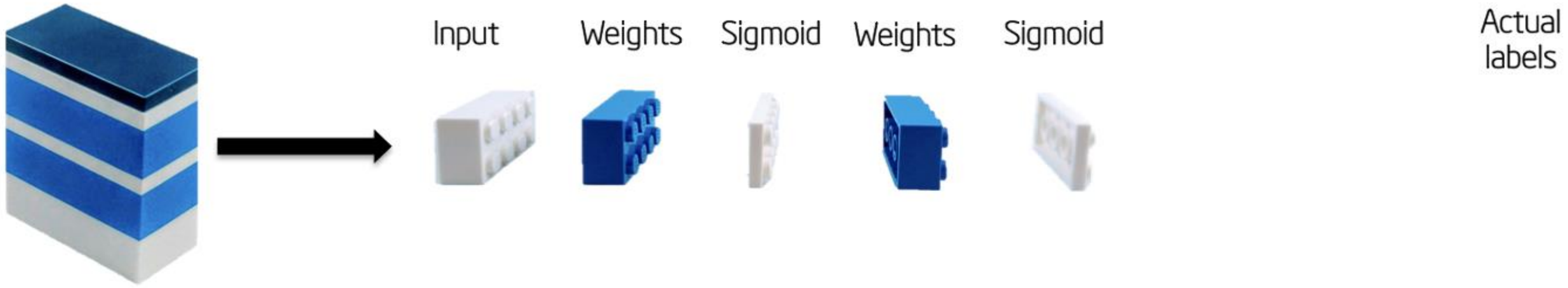
A XOR gate (or exclusive OR) is a logic gate that returns true (or 1) when one, and only one, of the inputs of the gate is true. Otherwise it returns false.

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

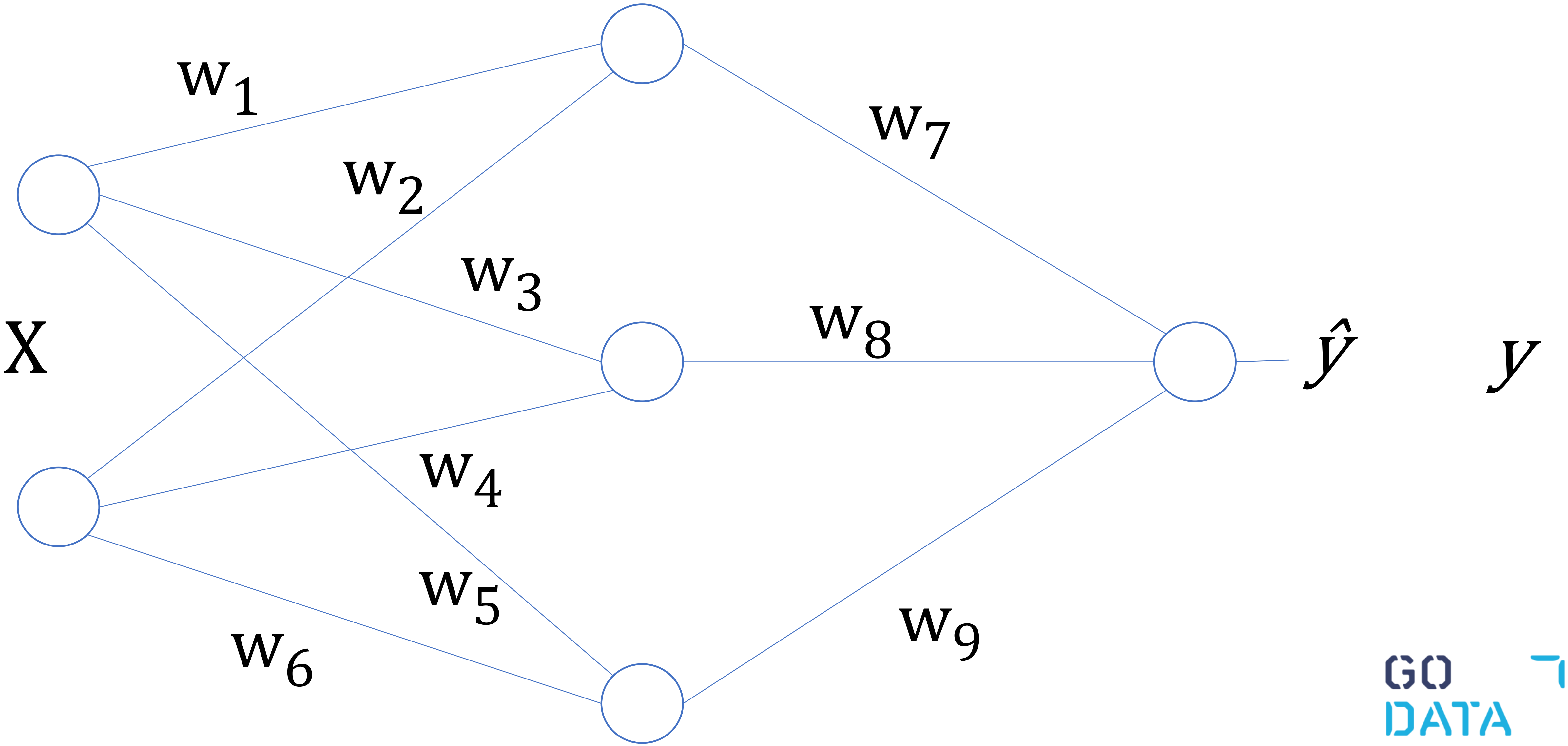
Whereas other logical operators (such as AND and OR) can be modeled with a single layer perceptron, the XOR operator is more complex and must be modeled with a multi-layer perceptron.

MLP

The MLP applies layers of transformations to our input. We compare the output of the model to our actual labels.

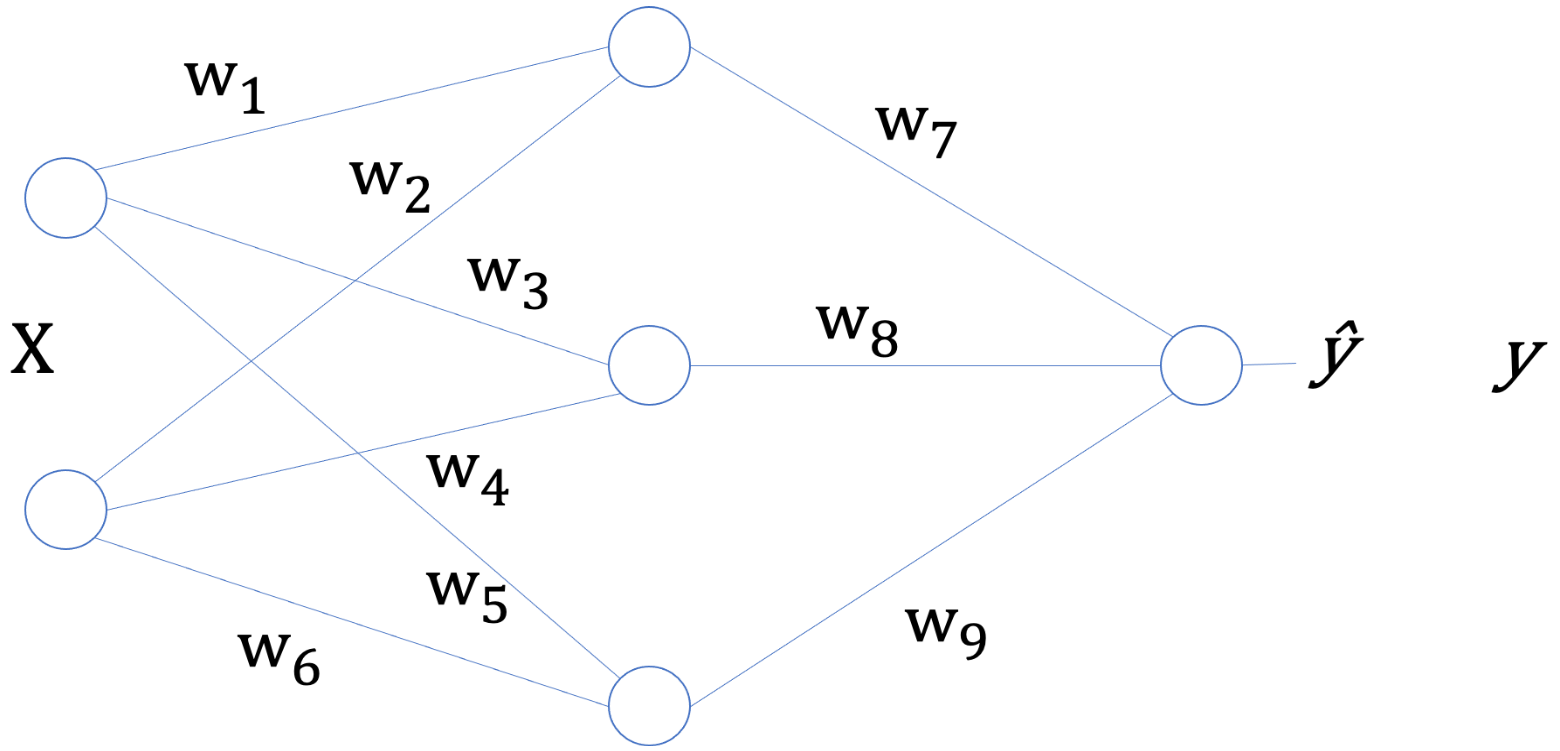


The next slide shows a schematic diagram of the MLP we will be creating



GO DATA DRIVEN

XOR MLP



$$\begin{bmatrix} X_1 & X_2 \end{bmatrix} \cdot \begin{bmatrix} w_1 & w_3 & w_2 \\ w_2 & w_4 & w_6 \end{bmatrix} \rightarrow \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix} \begin{bmatrix} w_7 \\ w_8 \\ w_9 \end{bmatrix} \rightarrow \begin{bmatrix} \hat{y} \end{bmatrix} \quad \begin{bmatrix} y \end{bmatrix}$$

Questions?