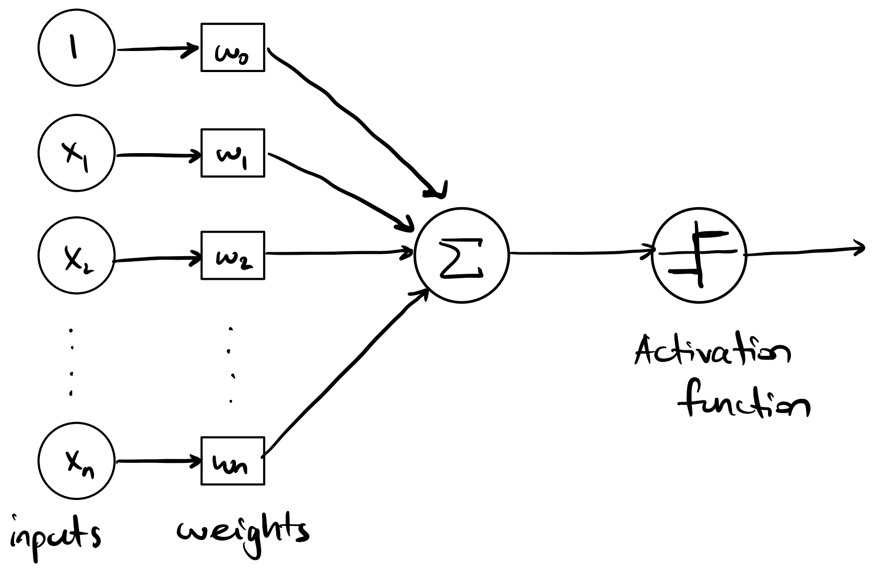
COMP9417 - Week 9 Tutorial notes

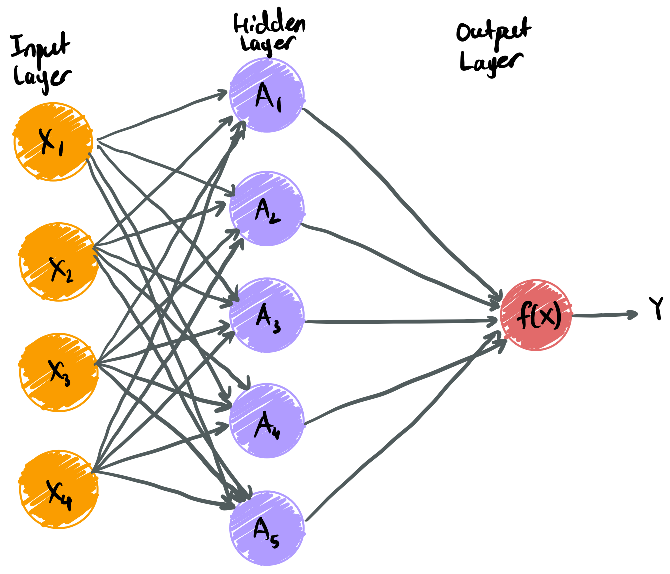
Neural Learning

Recap: The Perceptron



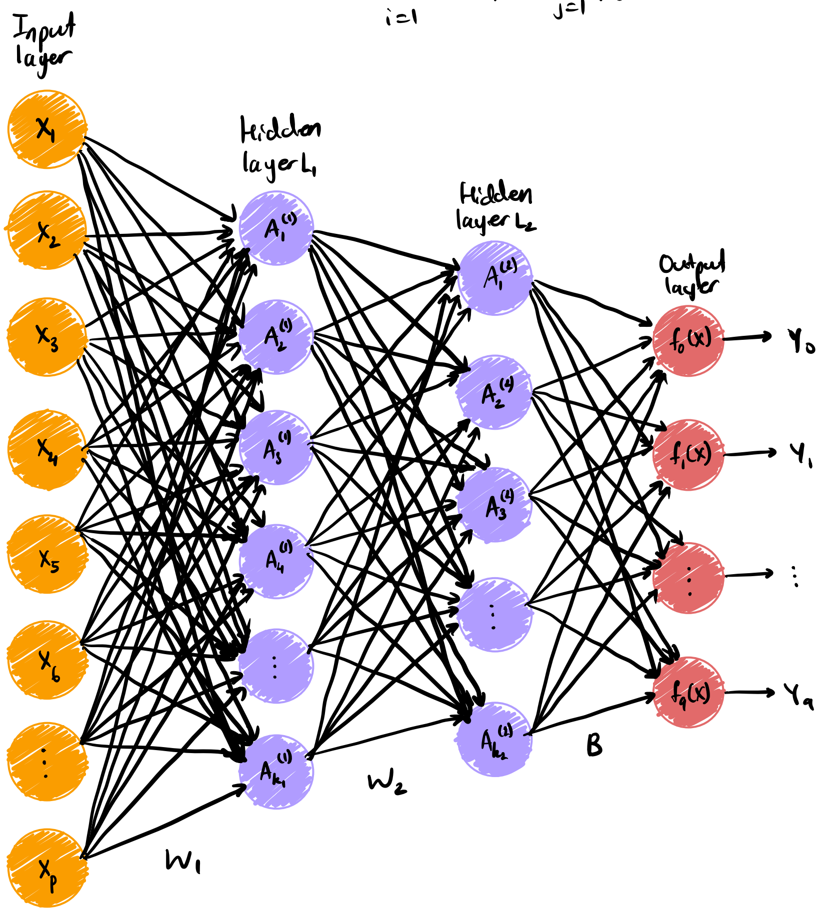
Multi-layer Perceptron

- A multi-layer perceptron is where we chain the perceptrons to learn non-linear patterns



If we define the activation function used for the hidden layer as g and the weights for input features as β:

$$
\begin{aligned}
f(X) &= w_0 + \sum_{i=1}^{n} w_i A_i \\
&= w_0 + \sum_{i=1}^{n} w_i g(X_i) \\
&= w_0 + \sum_{i=1}^{n} w_i g\left(\beta_0 + \sum_{j=1}^{p} \beta_j X_i\right)
\end{aligned}
$$



Back-propagation

The main problem now becomes: How do we learn this large number of weights?

As always, we define an appropriate loss function and optimise it. Due to the complexity of the function which is the neural network, we'll need to perform gradient descent to gradually improve our model over time

But how do we calculate the gradient?

The loss function of the activation:

$$L(a, y)$$

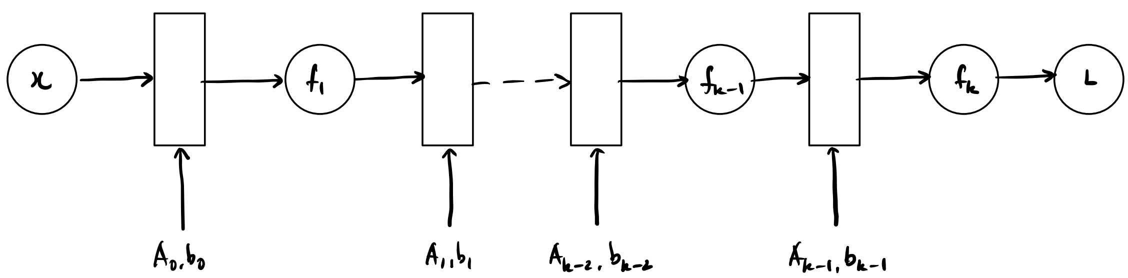
The activation is a function of the inputs, the weights and the bias:

$$a(x_1, \dots, x_n, w_0, \dots, w_n)$$

What we want to optimise the loss function is:

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial w_i}$$

Say we have the following network architecture, where A represents the weights and b the bias:



If we say that $\theta_k = \{A_k, b_k\}$ for a layer k The gradient of our coefficients looks like this:

$$\frac{\partial L}{\partial \theta_{k-1}} = \frac{\partial L}{\partial f_k} \frac{\partial f_k}{\partial f_{k-1}} \frac{\partial f_{k-1}}{\partial f_{k-2}} \frac{\partial f_{k-2}}{\partial \theta_{k-3}}$$

Using back-propagation, we can therefore calculate:

$$\nabla L(\theta, y) = \left[\frac{\partial L}{\partial \theta_1}, \frac{\partial L}{\partial \theta_2}, \dots, \frac{\partial L}{\partial \theta_k} \right]$$

We can then all of our parameters (in the basic case):

$$\theta^{(t)} = \theta^{(t-1)} - \nabla L(\theta, y)$$

Typically, the optimiser will be some form of stochastic gradient descent (minibatch in some cases) as classic gradient descent is expensive for a large number of parameters and data points