

# Neural Learning

COMP9417, 22T2

## 1 Neural Learning

## 2 Recap: The Perceptron

## 3 Multi-layer Perceptron

## 4 Back-propagation

# Neural Learning

# Neural Learning

You'll typically see this field referred to as *deep learning*.

Deep learning has become the forefront of modern machine learning. With it comes many challenges and intricacies which are out of the scope of this course (see COMP9444, *Deep Learning Book* by Goodfellow et al).

# Neural Learning

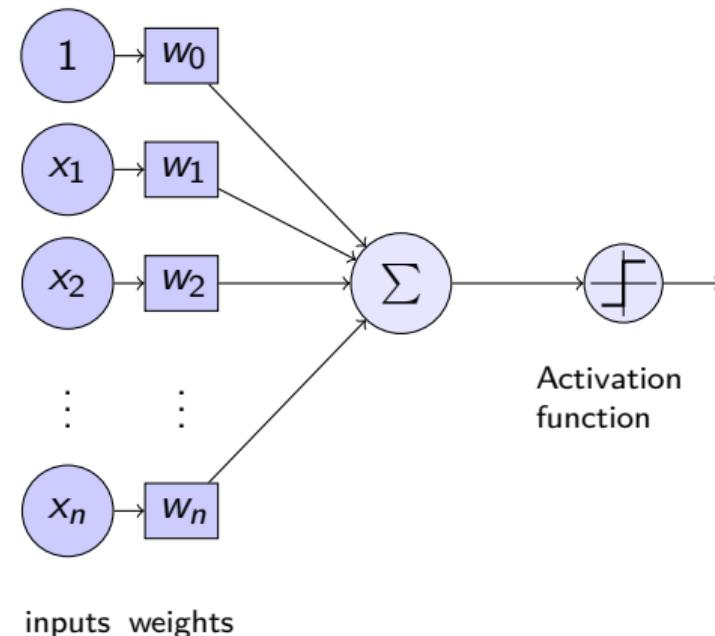
You'll typically see this field referred to as *deep learning*.

Deep learning has become the forefront of modern machine learning. With it comes many challenges and intricacies which are out of the scope of this course (see COMP9444, *Deep Learning Book* by Goodfellow et al).

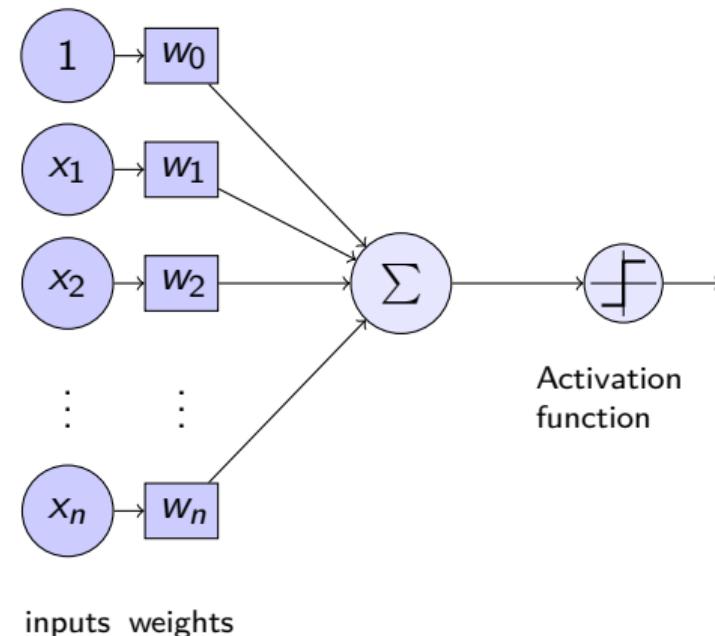
This course discusses what makes up neural networks, partially why they are effective and how they work.

## Recap: The Perceptron

# Recap: The Perceptron



# Recap: The Perceptron



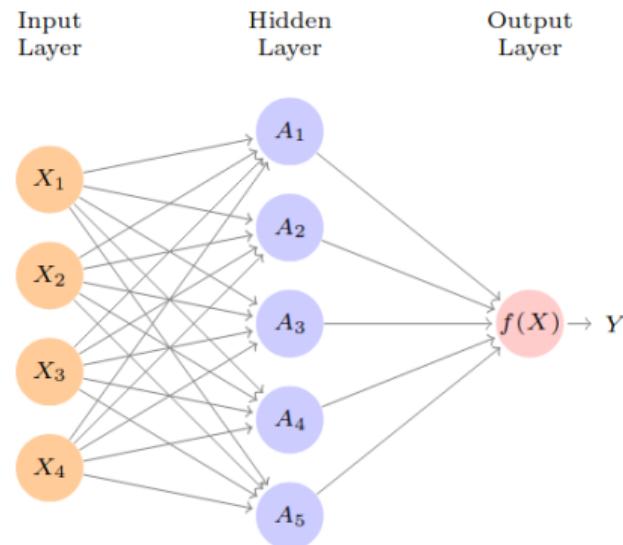
# Multi-layer Perceptron

# Multi-layer Perceptron

A multi-layer perceptron is where we *chain* these perceptrons to learn non-linear patterns.

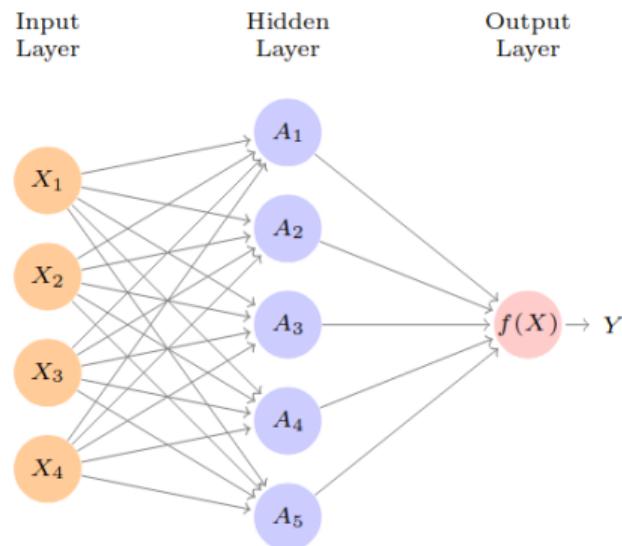
# Multi-layer Perceptron

A multi-layer perceptron is where we *chain* these perceptrons to learn non-linear patterns.



# Multi-layer Perceptron

A multi-layer perceptron is where we *chain* these perceptrons to learn non-linear patterns.

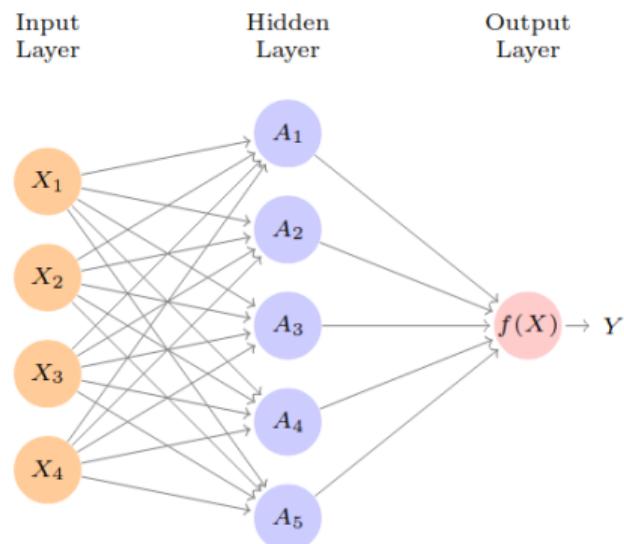


If we define the activation function used for the hidden layer as  $g$  and the weights for input features as  $\beta$ :

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

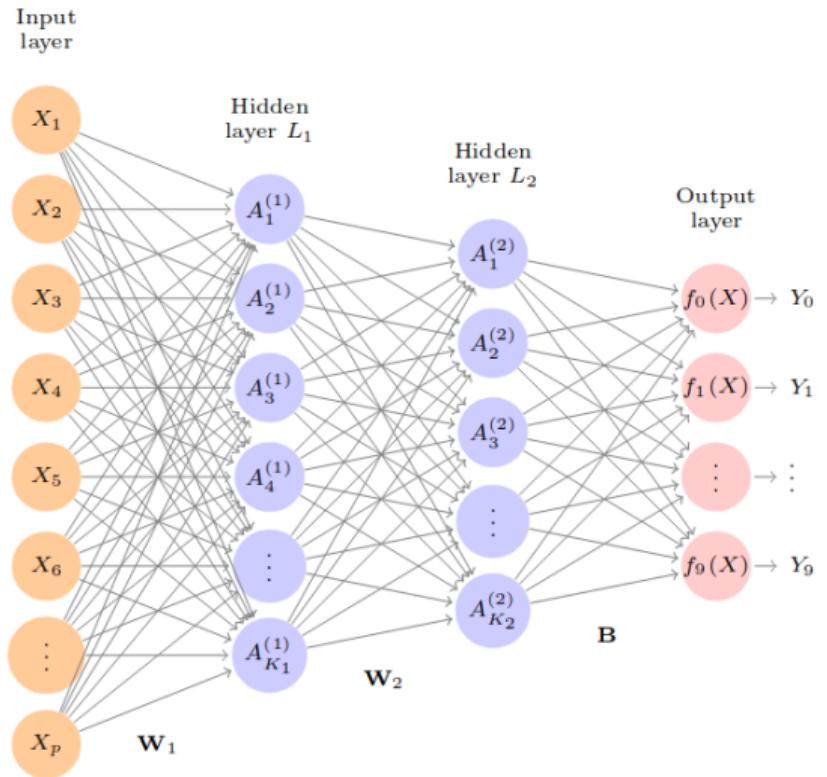
# Multi-layer Perceptron

A multi-layer perceptron is where we *chain* these 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  $\beta$ :

$$\begin{aligned}f(X) &= w_0 + \sum_{i=1}^n w_i A_i \\&= w_0 + \sum_{i=1}^n w_i g(\beta_{k0} + \sum_{j=1}^p \beta_{kj} X_i)\end{aligned}$$



# Back-propagation

# Back-propagation

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

# 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.

# Back-propagation

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

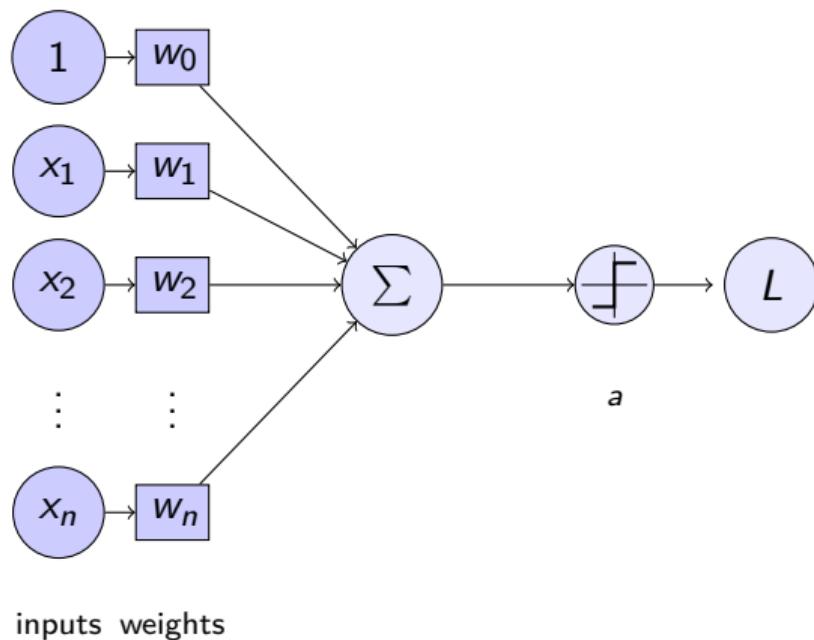
As always, we define an appropriate loss function and optimise it. Due to the complexity of the *function* which is the neural network, we'll need to perform gradient descent to gradually improve our model over time.

# Back-propagation

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

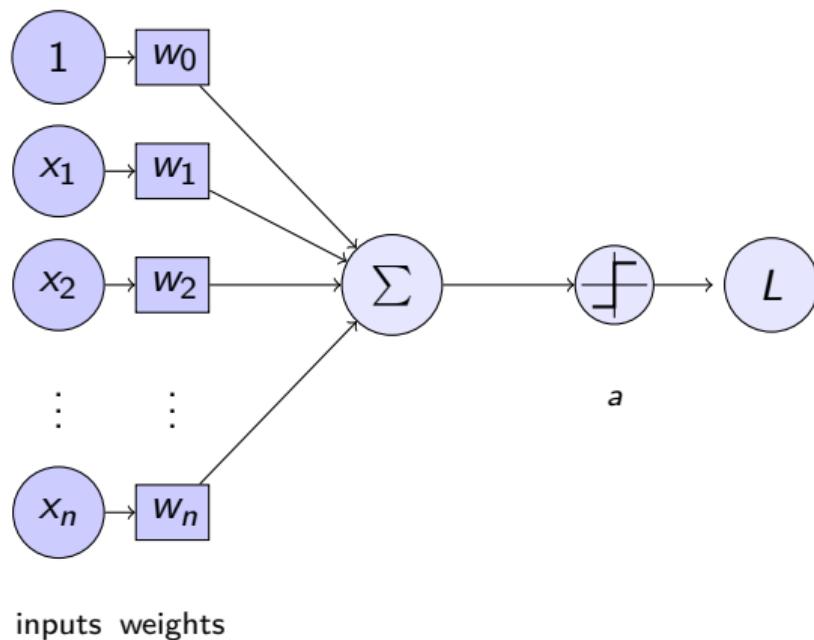
As always, we define an appropriate loss function and optimise it. Due to the complexity of the *function* which is the neural network, we'll need to perform gradient descent to gradually improve our model over time.

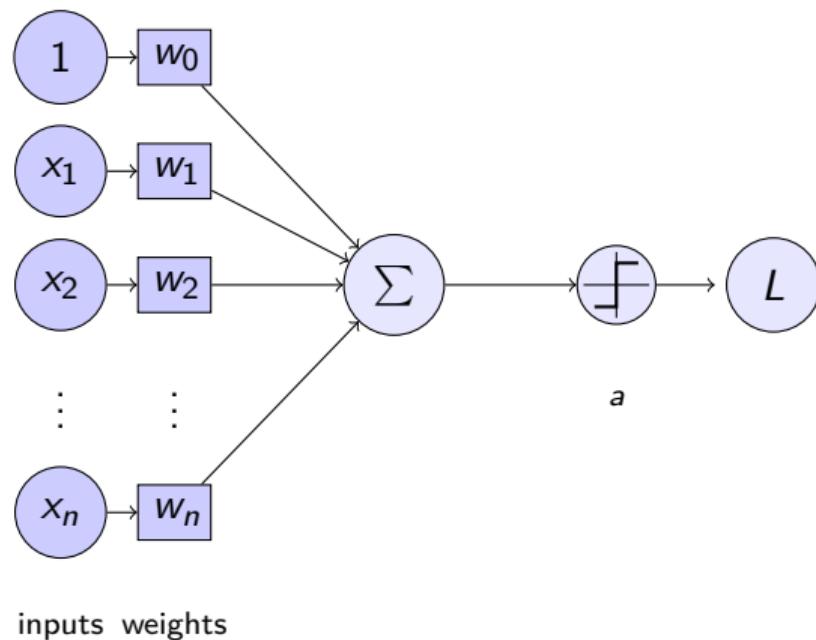
But how do we even calculate the gradient?



The loss is a function of the activation:

$$L(a, y)$$



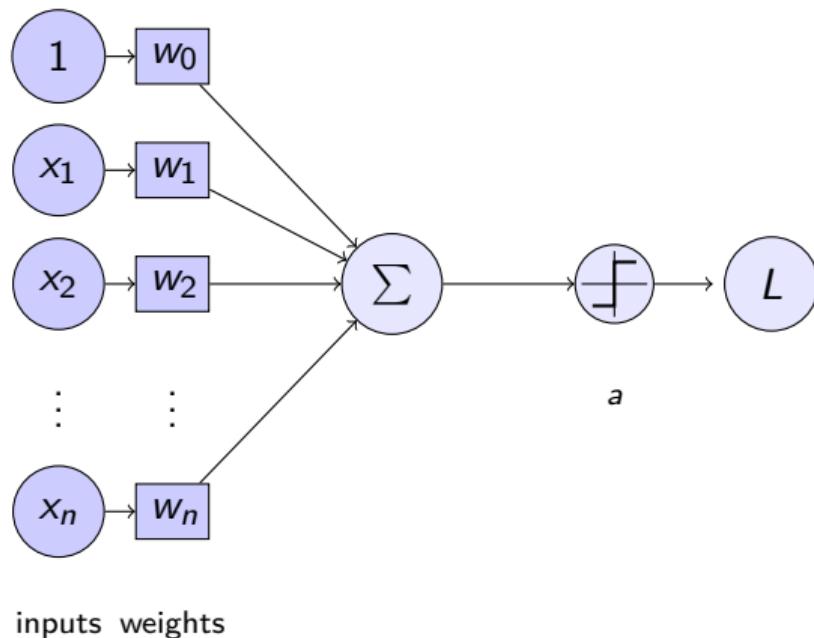


The loss is a function of the activation:

$$L(a, y)$$

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

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



The loss is a function of the activation:

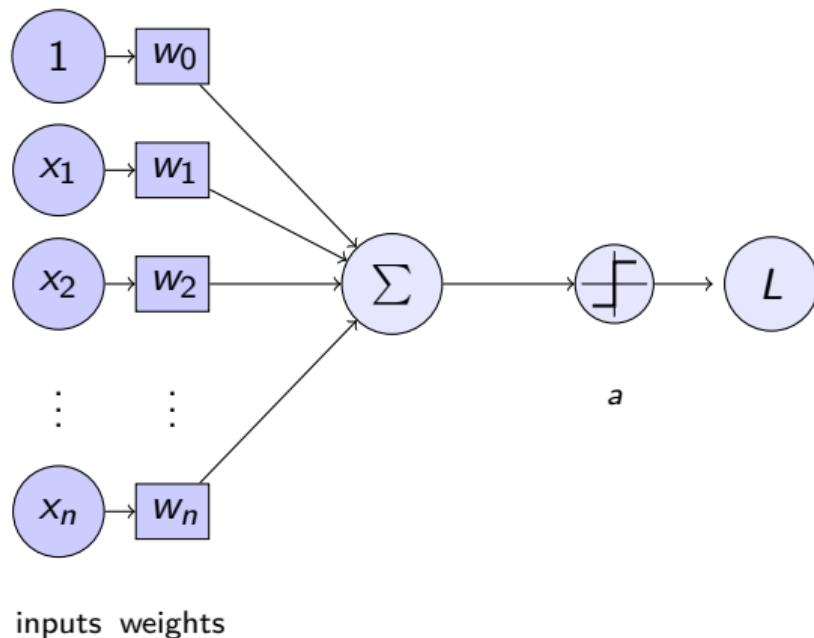
$$L(a, y)$$

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:

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



The loss is a function of the activation:

$$L(a, y)$$

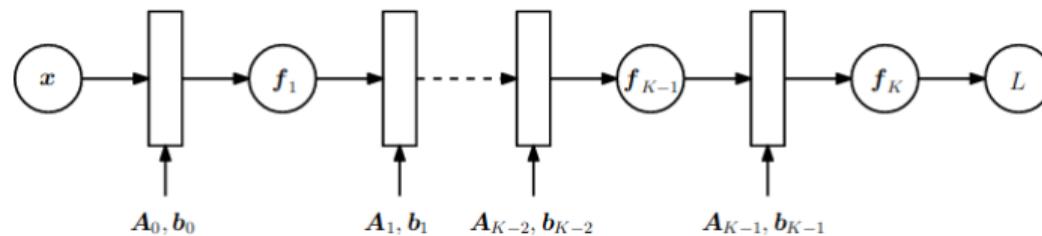
The activation is a function of the inputs, the weights and the bias:

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

What we want to optimise the loss function is:

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

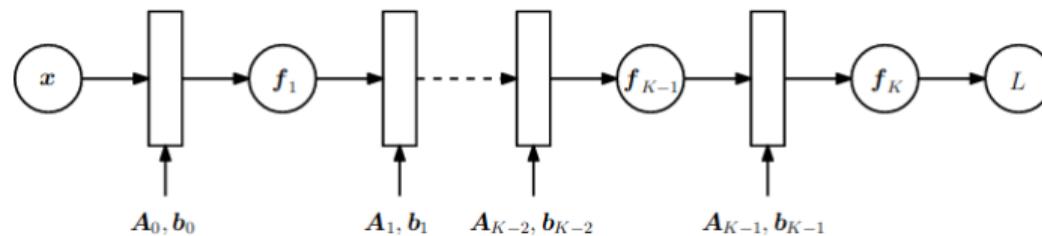
Say we have the following network architecture, where  $A$  represents the weights and  $b$  the bias:



If we say that  $\theta_K = \{A_K, b_K\}$  for a layer  $k$ . The gradient of our coefficients looks like this:

$$\frac{\partial L}{\partial \theta_{K-1}} = \frac{\partial L}{\partial f_K} \frac{\partial f_K}{\partial \theta_{K-1}}$$

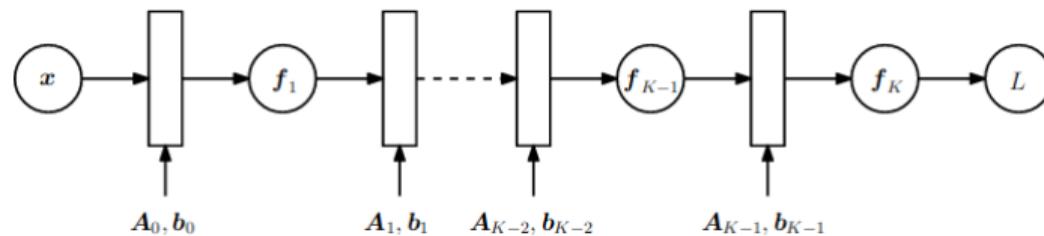
Say we have the following network architecture, where  $A$  represents the weights and  $b$  the bias:



If we say that  $\theta_K = \{A_K, b_K\}$  for a layer  $k$ . The gradient of our coefficients looks like this:

$$\frac{\partial L}{\partial \theta_{K-2}} = \frac{\partial L}{\partial f_K} \frac{\partial f_K}{\partial f_{K-1}} \frac{\partial f_{K-1}}{\partial \theta_{K-2}}$$

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:

$$\frac{\partial L}{\partial \theta_{K-3}} = \frac{\partial L}{\partial f_K} \frac{\partial f_K}{\partial f_{K-1}} \frac{\partial f_{K-1}}{\partial f_{K-2}} \frac{\partial f_{K-2}}{\partial \theta_{K-3}}$$

Using back-propagation, we can therefore calculate:

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

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.

Let's visualise a neural network working:

Tensorflow Playground