Mathematics in Deep Learning

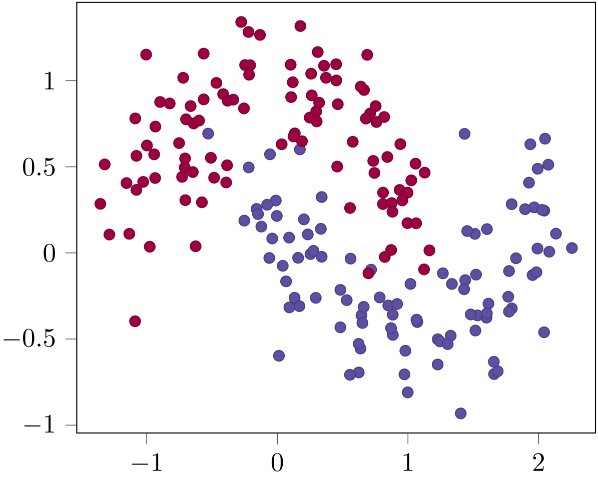
Lachlan Jones

Math199

2024

A classification problem

Let's build some networks to seperate this dataset of red and blue dots



Simple perceptron networks

Perceptrons are the building block of Neural Networks

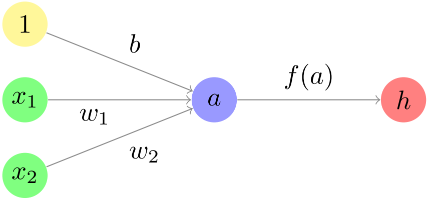


Figure: Simple perceptron network with two inputs

Simple perceptron networks

Perceptrons are the building block of Neural Networks

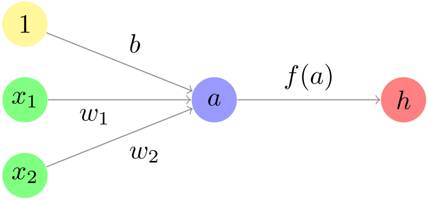


Figure: Simple perceptron network with two inputs

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$

Simple perceptron networks

Perceptrons are the building block of Neural Networks

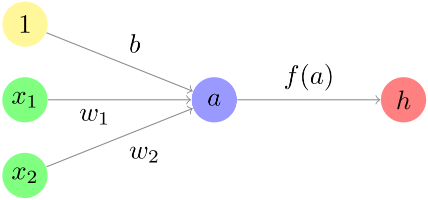


Figure: Simple perceptron network with two inputs

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- f is a non-linear activation function

Simple perceptron networks

Perceptrons are the building block of Neural Networks

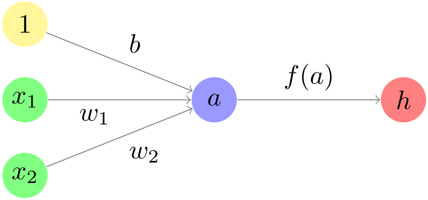


Figure: Simple perceptron network with two inputs

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- f is a non-linear activation function
- the weights w_1 and w_2 and bias b are parameters

Simple perceptron networks

Limited to linear decision boundaries:

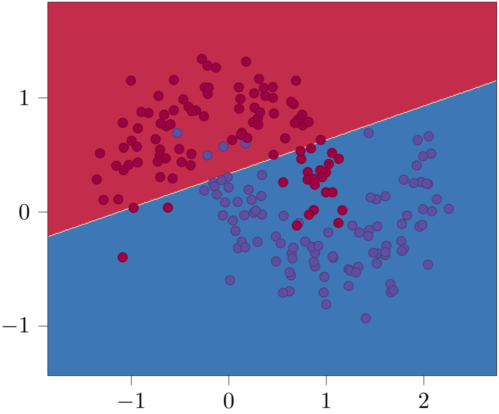


Figure: Simple perceptron decision boundary

MLP networks

- Multi-layered perceptron
- Combine multiple perceptrons together into a large network

MLP networks

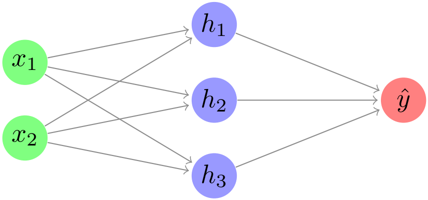


Figure: MLP network with two inputs and one hidden layer

Remark

We've tidied up our diagram to not have biases and activation functions, but they're still used in calculations.

MLP networks

Calculations:

$$a_1 = w_{1,1} \cdot x_1 + w_{1,2} \cdot x_2 + b_1$$
$$a_2 = w_{2,1} \cdot x_1 + w_{2,2} \cdot x_2 + b_2$$
$$a_3 = w_{3,1} \cdot x_1 + w_{3,2} \cdot x_2 + b_3$$

MLP networks

Calculations:

$$a_1 = w_{1,1} \cdot x_1 + w_{1,2} \cdot x_2 + b_1$$

$$a_2 = w_{2,1} \cdot x_1 + w_{2,2} \cdot x_2 + b_2$$

$$a_3 = w_{3,1} \cdot x_1 + w_{3,2} \cdot x_2 + b_3$$

This is a typical system of equations, so we can tidy things up with matrix algebra:

MLP networks

Calculations:

$$a_1 = w_{1,1} \cdot x_1 + w_{1,2} \cdot x_2 + b_1$$

$$a_2 = w_{2,1} \cdot x_1 + w_{2,2} \cdot x_2 + b_2$$

$$a_3 = w_{3,1} \cdot x_1 + w_{3,2} \cdot x_2 + b_3$$

This is a typical system of equations, so we can tidy things up with matrix algebra:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

MLP networks

Calculations:

$$a_1 = w_{1,1} \cdot x_1 + w_{1,2} \cdot x_2 + b_1$$

$$a_2 = w_{2,1} \cdot x_1 + w_{2,2} \cdot x_2 + b_2$$

$$a_3 = w_{3,1} \cdot x_1 + w_{3,2} \cdot x_2 + b_3$$

This is a typical system of equations, so we can tidy things up with matrix algebra:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

With an activation function, the output of the hidden layer is:

$$\underline{h} = f(W \cdot \underline{x} + \underline{b})$$

MLP networks

Typically, we have many hidden layers in our network architecture.

Generally:

$$\underline{h}_i = f(W_i \cdot \underline{h}_{i-1} + \underline{b}_i)$$

MLP networks

Typically, we have many hidden layers in our network architecture.

Generally:

$$\underline{h}_i = f(W_i \cdot \underline{h}_{i-1} + \underline{b}_i)$$

Remark

We can think of the network as a recurrence relation, passing the output of one hidden layer as the input to the next layer.

MLP networks

The combination of hidden layers and non-linear activation functions gives us a non-linear decision boundary:

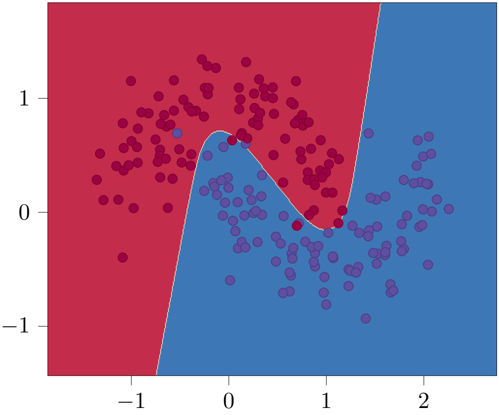


Figure: MLP decision boundary

Training

- Networks don't just magically start with great performance, we have to train them on our dataset.

Training

- Networks don't just magically start with great performance, we have to train them on our dataset.
- Training is achieved by updating our parameters, the weights and biases.

Training

- Networks don't just magically start with great performance, we have to train them on our dataset.
- Training is achieved by updating our parameters, the weights and biases.
- We achieve this using calculus.

Training

We define a loss function that depends on the parameters:

$$J(\theta_1, ..., \theta_n)$$

A loss function compares the error of a network's current prediction against human labelling of data.

We want to find a local minima of this function.

Training

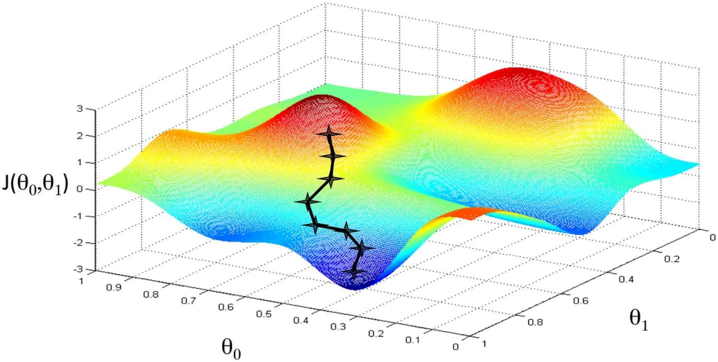
We can compute the gradient of the loss function, which tells us how to make small updates to our parameters:

$$\nabla J(\theta_1, ..., \theta_n) = \begin{bmatrix} \frac{\partial J}{\partial \theta_1} \\ \frac{\partial J}{\partial \theta_2} \\ \vdots \\ \frac{\partial J}{\partial \theta_n} \end{bmatrix}$$

These are partial derivatives with respect to each parameter. You'll learn about partial derivatives later in semester 2.

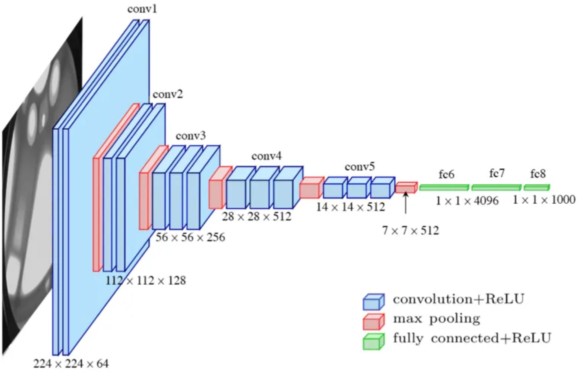
Training

A great way to visualise this is walking down a mountain, the gradient tells us the steepest way down.



Conclusion

These are the foundational concepts that are used in advanced architecture:



While they may seem scary, neural networks are just algebra and calculus.

Will AI take over?

At the end of the day, neural networks are simply matching a trend to some data: