# Section 3: Backpropagation

## Table of Contents



## Introduction

Backpropagation is a fundamental algorithm in the training of artificial neural networks, though it is not the algorithm directly responsible for updating the network's weights – this is *gradient descent*, which we'll cover in the next lecture. It was popularised by David Rumelhart, Geoffrey Hinton and Ronald Williams in the paper *Learning representations by back-propagating errors*.

Backpropagation enables the efficient computation of gradients. This process works by propagating errors backward through the network, from the output layer to the input layer, allowing us to subsequently iteratively adjust the network's weights to minimise the loss – the error between predicted and actual outputs. This step-by-step computation works fir arbitrarily complex networks, and backpropagation has proven particularly important for training multilayered deep networks.

The introduction of backpropagation marked a turning point in AI, making deep learning much more practical, ultimately resulting in significant advancements across a range of domains.

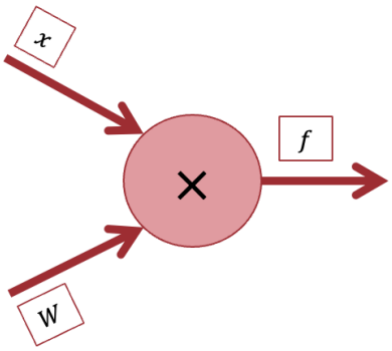
## Computational Graphs:

Before we take this any further, let's look at how we can draw some computational graphs – this is a key skill that will allow us to break down even very complex functions into manageable steps. It is a visual way to represent mathematical operations and how data flows through them. You can think of it as a flowchart where each node represents an operation (like addition, multiplication, or more complex functions), and the edges (or joining lines) represent the inputs and outputs of those operations.

We'll start with a simple example using our classic linear classifier:

$$f(x, W) = Wx$$

We can draw out the computational graph for this function like this:




Computational graphs are especially useful in machine learning because they help us track how variables interact and how to compute derivatives efficiently.

Have a go at drawing the computational graph for the following function:

$$f(x, y, z) = x(y + z)$$

---

---

Check out the worksheets on the github if you'd like to have more practice at this!

# How Does Backprop Work?

The backpropagation algorithm involves two main steps: the Forward Pass and the Backward Pass.

## The Forward Pass

In our forward pass, the input data is fed into the input layer. We'll use the graph that we've drawn out up above (there's space here to recreate it, to keep everything nice and clear):

---

**Problem 1:**

---

1. Let's plug in the inputs: $x = 3, y = 4, z = -2$.

2. And compute our forward pass, calculating our output at *q* and the output of the network overall.

## The Backward Pass

In our backward pass, the error (the difference between the predicted and actual output) is propagated back through the network – ultimately to adjust the weights of our network. We adjust these weights using *gradients*, which are computed with the chain rule. These gradients indicate how much each weight should be adjusted to minimise the error in the next iteration. Our backward pass continues layer by layer, hopefully ensuring that the network learns and improves its performance.

1. First let's write out our expressions for the gradient that we need to compute on our graph above – using the chain rule as necessary.

2. Let's calculate the gradients of our network now.

# Practice Problems

| Problem Number | Function |
|---|---|
| 2 | $z = \frac{1}{1 - e^{-x}}$ |
| 3 | $z = x^2 + y$ |
| 4 | $z = (x + y) \times (p + q)$ |

Use the space at the back of this booklet to write out your answer (and copy down the other ones during the other groups' presentations)!

# Pros and Cons of Using Backpropagation

The key benefits of using the backpropagation algorithm are:

- It is "easy" to implement in that we can use relatively simple operations to compute the gradients of complex networks. It is however also easy to make errors in our computation, so it's important to double-check our maths.
- It's efficient (or more efficient than a lot of approaches) – and works across a range of different networks.

What are some potential challenges? Note some down here to discuss in your groups: