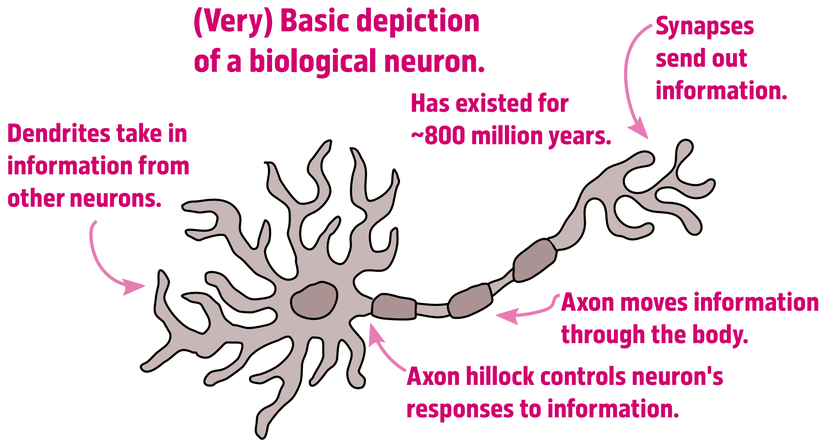
NEURAL NETWORKS and NEURONS

Neural networks are the tool that drives some of our best Machine Learning. Have you used ChatGPT, MidJourney, DeepL or Grammerly? These are all recent examples of highly successful neural networks.

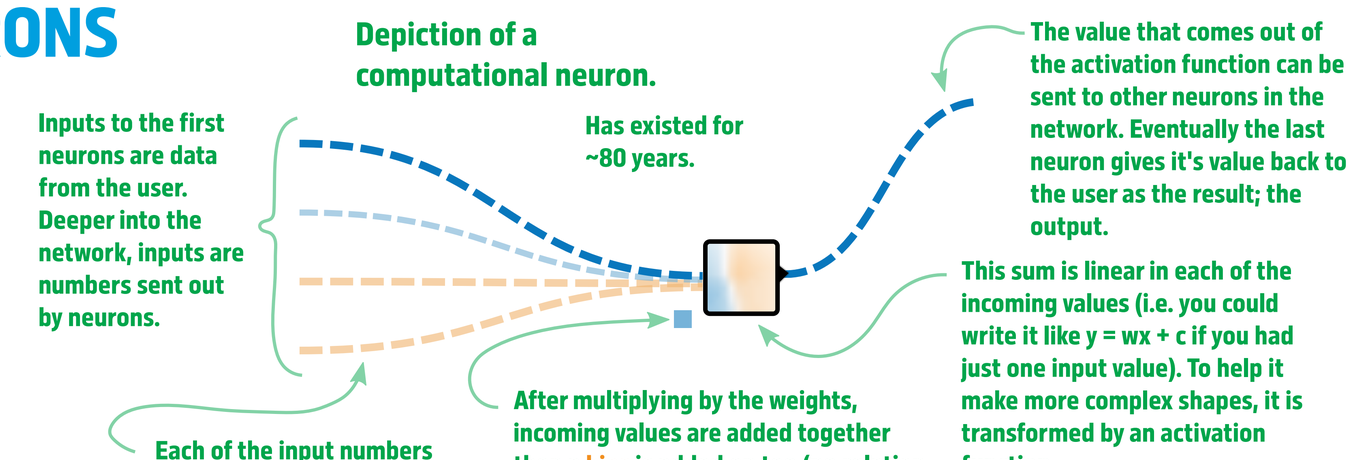
So how do they do it? They are each complicated and slightly different designs, but at the smallest level, they are all made of the same thing. As a particle physicist, the smallest level is the level that interests me the most; so let's have a close up look at these neural networks.

Neural networks are build of components called neurons. These neurons are inspired by neurons in animal (e.g. human) brains, and the neural network is sometimes described as resembling the brain.

(Very) Basic depiction of a biological neuron.



Depiction of a computational neuron.



$$y_{\text{out}} = f\left(\left(\sum_{\text{input}=i} w_i x_i\right) + b\right)$$

If you have some data, and you have a question that can be answered with numbers (which can be used to represent other things, like letters or colours), then it's probably possible to design a neural network that could answer it.

If the input values go through enough neurons, they can turn into almost any set of output values. This is known as the universal approximation theorem.

More complicated questions might need a lot of data or a very large computer, but normally the hardest part is getting the question right. Asking a question with a neural network is like asking a genie for wishes; you will get exactly what you asked for and may then realise it wasn't what you wanted.

OUR GOAL

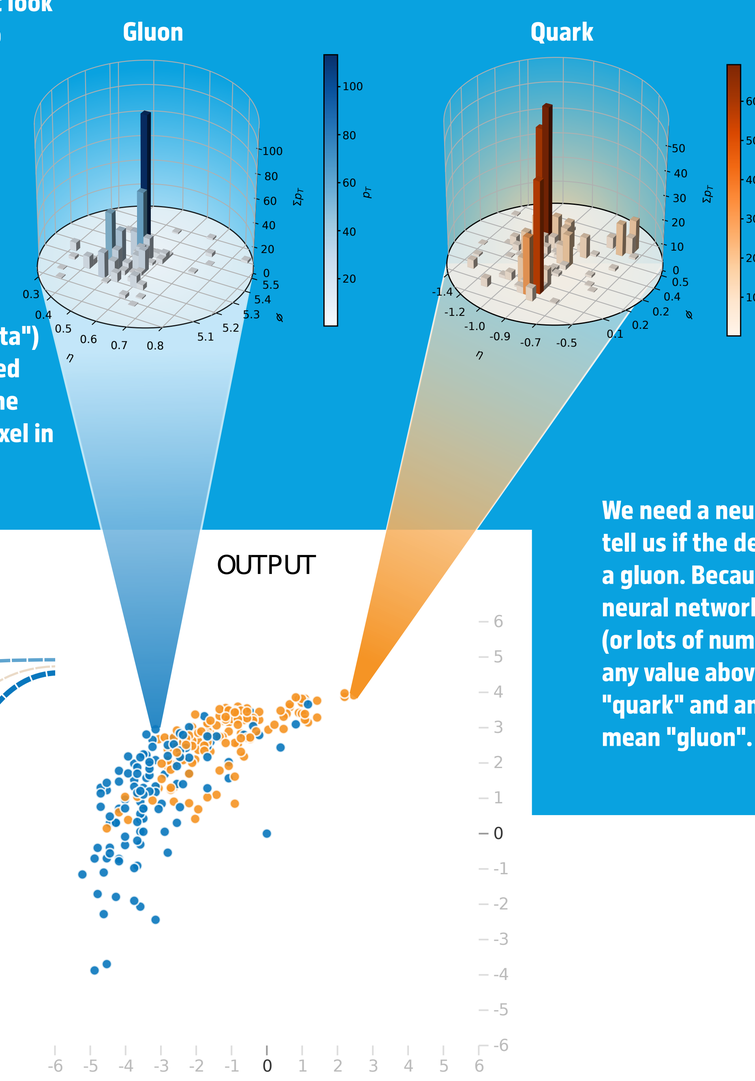
A classifier is a tool to identify what you are looking at; it divides observations into classes. We could have a classifier that divided photos into photos of dogs or cats. That's a task that a human could do too though.

One example of a neural network used in particle physics is a particle classifier. Identifying particles is often quite difficult.

We catch mystery particles in a detector that is a bit like a specialised camera. These particles break up into fragments when they hit our detector, and the detector can estimate how many fragments were made, and the energy of the fragments.

Different particles fragment in different ways, so by looking at the number of fragments, and the way the energy is shared between the fragments, we can estimate which particle hit the detector.

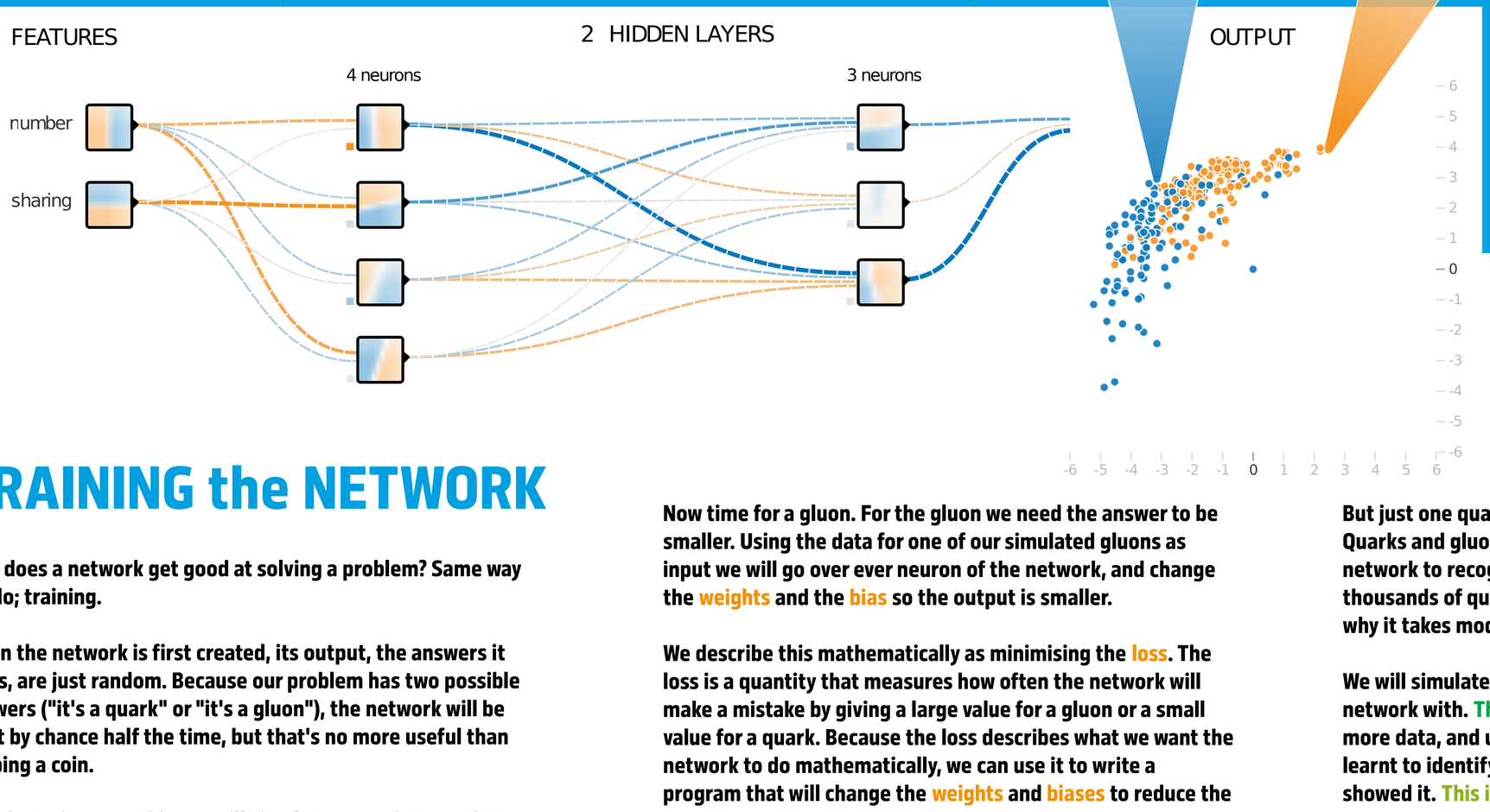
Here is a gluon and a quark. They don't look that different, do they?



η (pronounced "eta") and φ (pronounced "phi") measure the location of the pixel in the detector.

Each cuboid represents one pixel that measured some energy from a particle fragment. The colour and height of the cuboid represents how much "pT" (transverse momentum) was in that pixel. This is roughly the same as energy.

We need a neural network that can tell us if the detector saw a quark or a gluon. Because the output of neural networks is always a number (or lots of numbers) we will consider any value above 0.5 to mean "quark" and any value below 0.5 to mean "gluon".



TRAINING the NETWORK

How does a network get good at solving a problem? Same way we do; training.

When the network is first created, its output, the answers it gives, are just random. Because our problem has two possible answers ("it's a quark" or "it's a gluon"), the network will be right by chance half the time, but that's no more useful than flipping a coin.

In order to improve this, we will simulate some data, so that we know what the correct answer is. Remember that our network outputs a number? And we wanted the network to output a large number when the detector saw a quark, and a small number when it saw a gluon.

So we simulate lots of quarks and lots of gluons. We show the network a quark, and whatever value it gives back, we want to make it larger. We can change the behaviour of each neuron in the network by changing weights that it multiplies incoming particles with, and the bias that it adds on.

First let's train it on one of our simulated quarks. We give it the quark as input, and go over every neuron in the network and modify it a bit, changing it's weights and bias, so that the number it gives as the output for this quark is a bit larger.

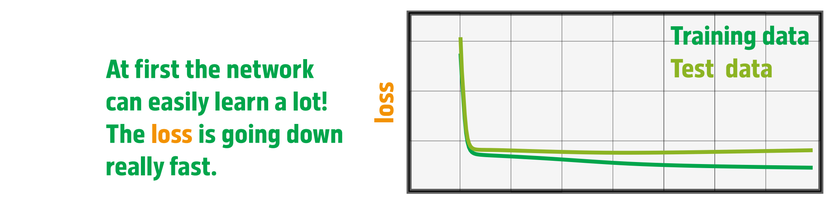
Now time for a gluon. For the gluon we need the answer to be smaller. Using the data for one of our simulated gluons as input we will go over ever neuron of the network, and change the weights and the bias so the output is smaller.

We describe this mathematically as minimising the loss. The loss is a quantity that measures how often the network will make a mistake by giving a large value for a gluon or a small value for a quark. Because the loss describes what we want the network to do mathematically, we can use it to write a program that will change the weights and biases to reduce the loss (the number of mistakes) automatically.

But just one quark and just one gluon isn't enough information. Quarks and gluons all look a bit different, and we need the network to recognise any quark and any gluon. So we will show it thousands of quarks and thousands of gluons! Which is part of why it takes modern computers to train a neural network.

We will simulate lots of data, both quarks and gluons, to train our network with. This is the training data. Then we simulate a bit more data, and use that to check that the network has really learnt to identify any quark or gluon, not just the ones we already showed it. This is called the test data.

Example of minimising the loss on training data.



At first the network can easily learn a lot! The loss is going down really fast.

Then all the easy things to learn are finished, learning slows down.

Sometimes the network starts performing worse on the test data than the training data. This shows that it's learning to identify individual items in the training data, and not general things about quarks and gluons.

Come try it out at our computer!

Henry Day-Hall and Konrad Helms