1-41

Module 3

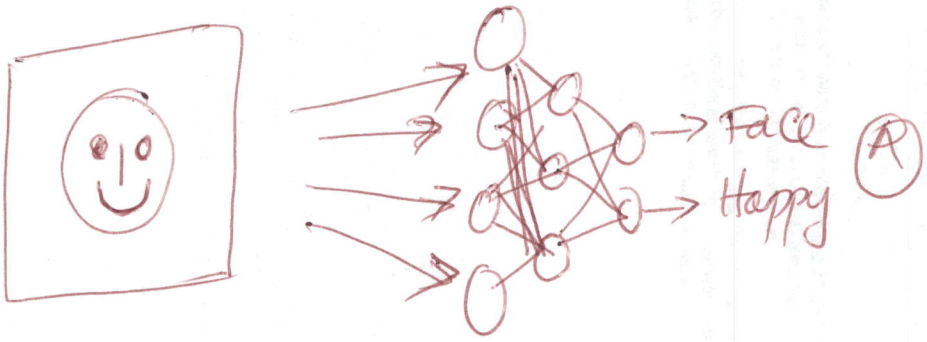
Simple Neural Network:

Here we going to see how we motivate chain rule will enable us to iteratively update the values of all the weights and biases, such that the network learns to classify the input data, based on a set of training examples.

When we say that we are training a network, we typically mean using some kind of labelled data, which are pairs of matching inputs and outputs

Eg. If we were to build a network to recognise pictures of faces and predict whether they were happy or not for our training data

Each of the inputs might be the intensity of a single pixel from the image And this would be paired with an output which just says ~~whether~~ whether this image contains a face, or ~~whether~~ whether it was a happy face



The classic training method is called

BACKPROPAGATING.

It looks first at the output neuron, and then works back through the network

If we start by ~~starting~~ choosing a simple structure, such as the one shown here (A)

with 4 input units, 3 hidden units, and 2 output units

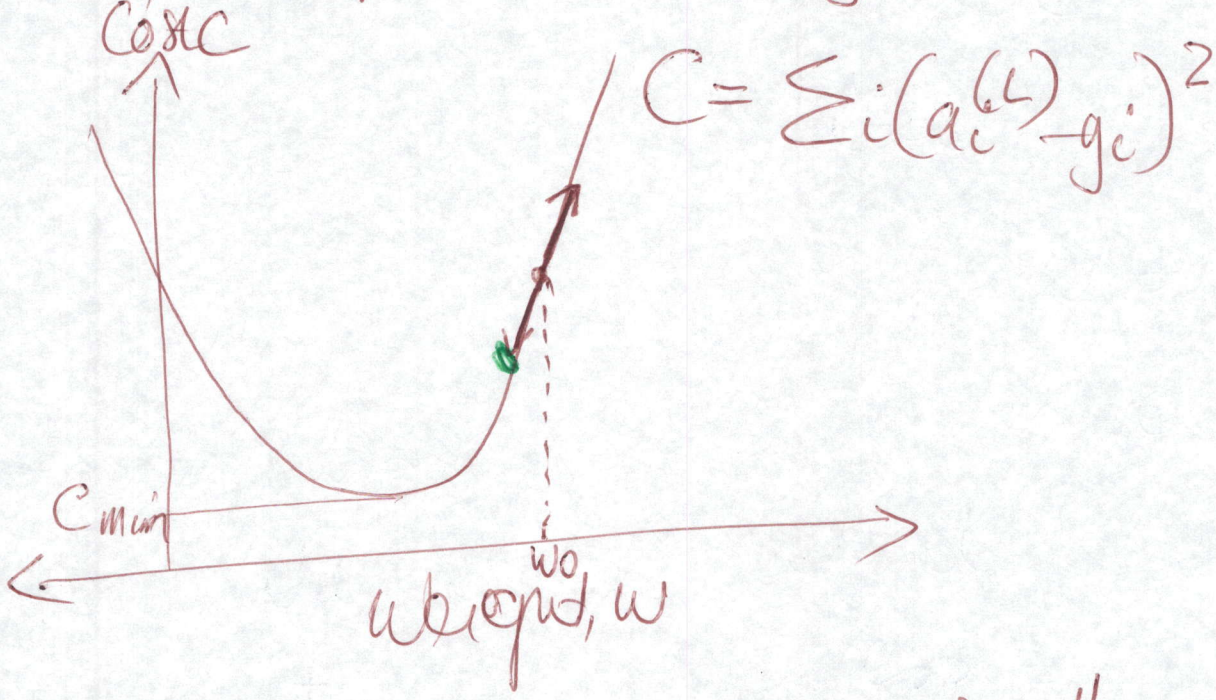
What we trying to do, is find the 18 weight and 5 bias that Cause of our network to Set match the training inputs to their labels.

Initially we will set all our weight and bias to random number; So initially when we pass one data into our network what we get out will be meaningless

However, we can then define a Cost function, which is simply the sum of the Squares of the differences between the desired outputs y, and output that our untrained network gives as

$$C = \sum_i (a_i^{(L)} - y_i)^2$$

If we were to focus on the relationship between one specific weight and the resulting cost function, it might look something like this

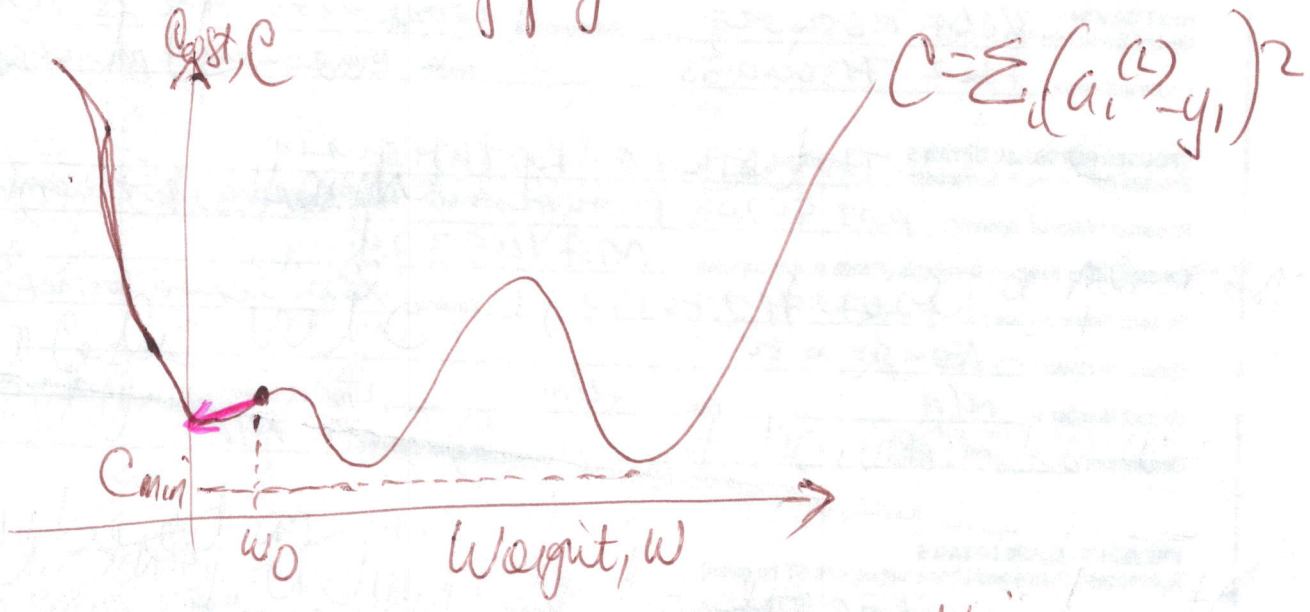


where if it either too large or too small the cost is high, but at one specific value, the cost is at a minimum

Now, based on our understanding of calculus if we were somehow are able to work out the gradient of C w.r.t the variable w, at some initial point w_0 then we can simply head in the opposite direction.

For example, at the point ●, shown on graph the gradient is positive, and therefore increasing w, would also increase the cost. So we should make w smaller, to improve our network.

However, at this point, it's worth noting that our cost function may look something more like the wiggly curve here



which is several local minima, and this more complicated to navigate

Furthermore, this plot is just considering one of our weights in isolation.
But what we really like to do is find the minimum of the multidimensional ~~height~~ hypersurface
Much like the 2D example, ~~like~~ we saw in the previous model.
Also like the previous models, if we want to head downhill, we will need to build the Jacobian, by gathering together the partial derivatives of the cost function wrt to all the relevant variables
So, now that we know what we want, we just need to look again at our simple 2 Node network

And at this point, we can immediately write down a Chain Rule expression for the partial derivative of the cost wrt either ~~with wrt~~ our weight or our ~~bise~~ bias

And we have highlighted the $a^{(i)}$ term, which links the two derivatives

However it's often convenient to make use of an additional term, which we will call Z, that will ~~hold~~ hold our weighted activation plus bias term.

This will allow us to think about differentiating a particular sigmoid function we happen to choose separately

We must ~~do~~ therefore include an additional link in our derivative chain

We now have the two layer Chain rule Expression we require to navigate the two dimensional WB space, in order to minimize the Cost of this simple network for set of training examples

$$a^{(0)} \; O \text{———} O \; a^{(1)}$$

$$z^{(1)} = wa^{(0)} + b$$

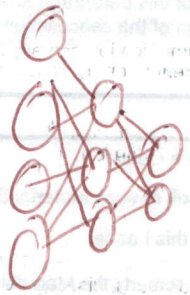
$$a^{(1)} = \sigma(z^{(1)})$$

$$C = (a^{(1)} - y)^2$$

$$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial w}$$

$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial b}$$

Clearly things are going to get a little more complicated when we add more neurons



$$z^{(L)} = w^{(L)} \cdot a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$

$$C = \sum_i (a_i^{(L)} - y_i)^2$$

But fundamentally we still just applying the chain rule to link each of our weights and biases back to ~~each~~ its effect on the cost.

Ultimately allowing us to train our network.

In following exercises we going to work through how to extend what we saw of simple one, to multilayer networks