

①

Module 3

1:41

Simple Neural Network

We're going to see how we motivate this rule will enable 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.

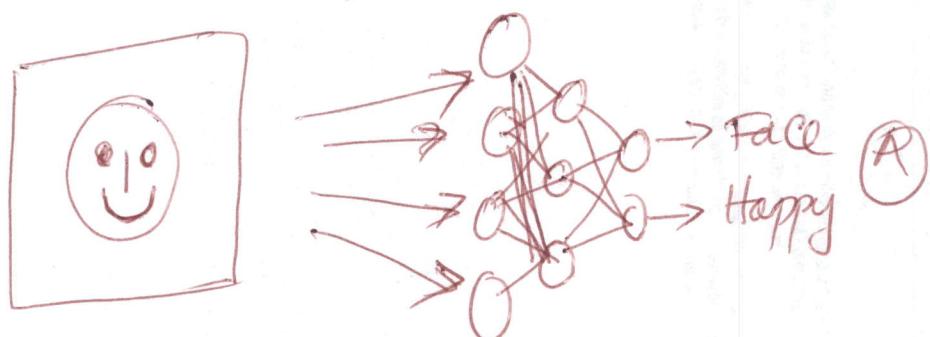
When we say that we are training ~~the~~^a network, we typically mean using some kind of labelled data, which are pairs of matching inputs and outputs

Eg. If we were to build a network to recognise pictures of faces and predict whether we're happy
Based on 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 the image
contains a face, or ^{whether it was} happy face



The classic training method is called

BACKPROPAGATION.

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

If we start by ~~saying~~ choosing a simple structure,

such as the one shown here

with 4 input units, 3 hidden units,

and 2 output units

(3)

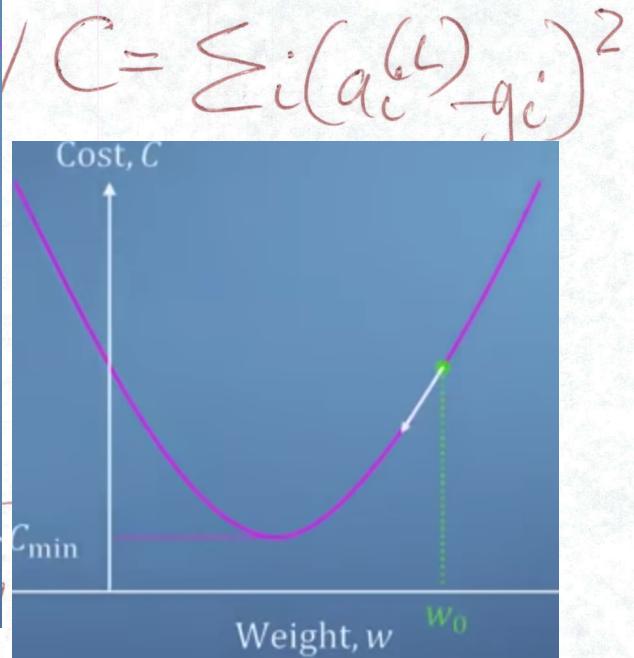
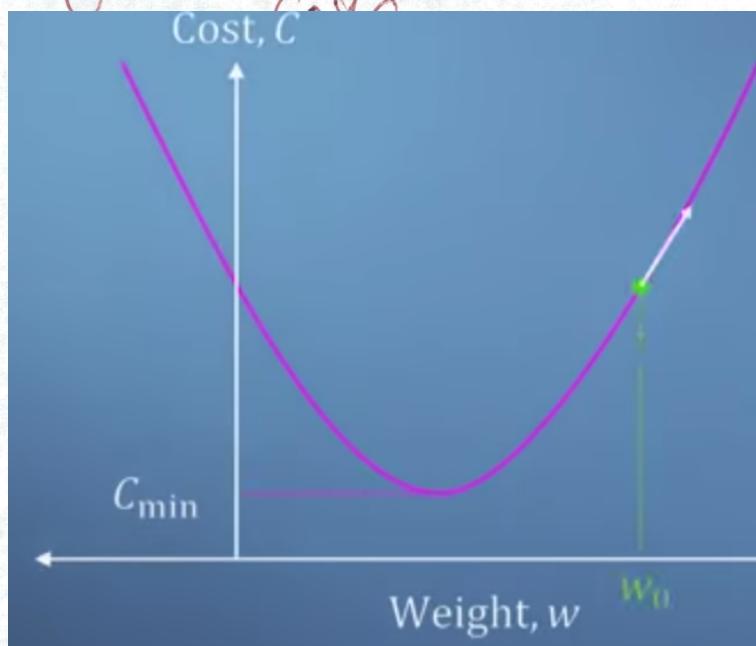
What we trying to do , is find the
 18 weight and 5 bias that
 cause our network to best match the
 training inputs to their labels.

Initially we will set all our weight and bias
 to random number; so initially when we
 pass some 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_j and output that our
 untrained network give us.

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

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



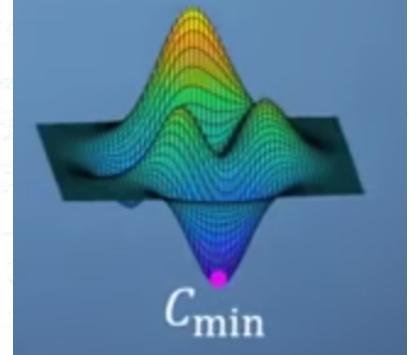
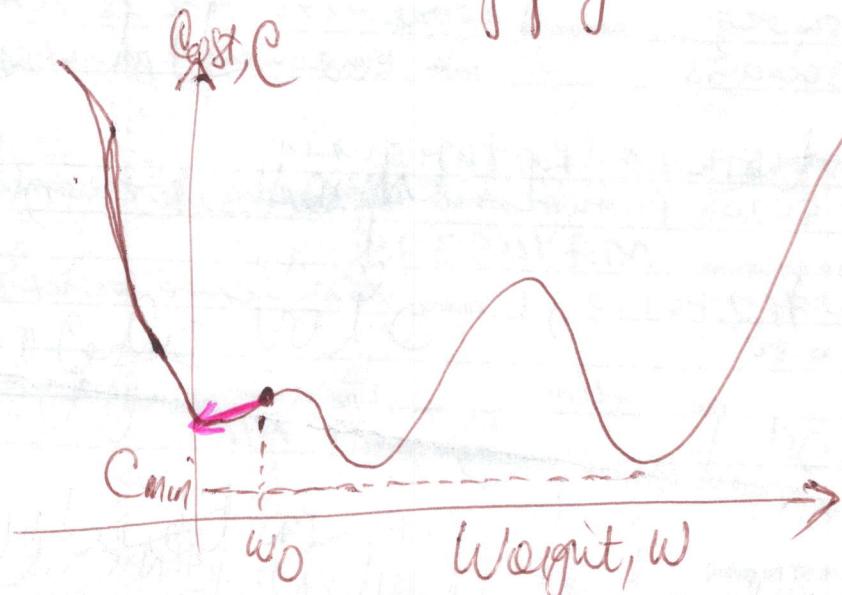
If we let w be either too large or too small the cost is ~~high~~ high, but at one specific value, the cost is at a minimum

Now, based on our understanding of calculus if we were somehow 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.

(5)

For example, at the point \bullet , shown on graph
 the gradient is positive, and therefore
 increasing w , would also increase the cost
 So we shall 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, 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 weight hyper surface

Much like the 2D example, ~~we saw~~ in the previous model.

Also like the previous model, we want to head down hill, we will need to build the Jacobian, by gathering together the partial derivatives of the cost function w.r.t to all the relevant variables.

So, now that we know what we after, we just need to look again at our simple 2 node network.

(7)

And at this point, we can immediately write down a Chain Rule expression for the partial derivative of the cost ~~is with respect to~~^{w_j} either with respect to weight or our bias.

And we have highlighted the $a^{(i)}$ term.

which links these two derivatives.

However it's often convenient to make use of an additional term, which

we will call Z_1 , that will hold hold our weighted activation plus bias terms

This will allow us to think about

differentiating ~~the~~ particular sigmoid function we happen to choose separately

we must therefore include an additional link in our derivative chain

P.32

We now have the two backpropagation expressions
 we require to propagate the two dimensions
 in the space, in order to minimize the
 Cost of this sample relevant for set of
 training examples

$$a^{(0)} \quad o \quad a^{(1)}$$

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

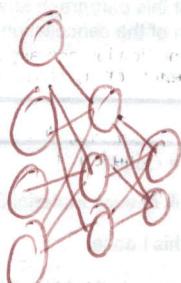
$$a^{(1)} = \sigma(z^{(1)})$$

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

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

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

Clarify 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 apply up
the chain rule to link each of our
weights and biases back to each its
effect on the cost.

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