Module 3 1:35:29

Simple Neural Network

Here we going to be introduced to concept of artificial neural networks

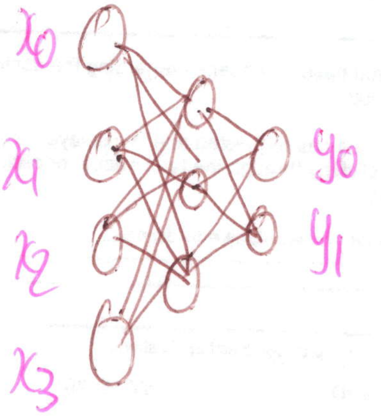
In just enough detail, that we will be ready to see how the multivariate Chain rule is crucial for bringing it to life

Safe to assume, we all have heard of NN's, and also aware that it is extremely powerful tool when applied to wide variety of important real world problems

Including image recognition and language translation

But how do they work?

often see the following diagrams:



$y = f(x)$

Where circles are our neurons
and lines are the network of connections
between them.

This may look far removed from what we
just learnt,
But fundamentally a NN is just a
mathematical function, which takes
input variable, gives you another variable back
where both of these variable could be vectors.

Lets now look at the simplest possible case, so we can translate these diagrams into some formulae.

Here we have a network, which takes in a single scalar variable, $a^{(0)}$ and returns another scalar $a^{(1)}$

$$a^{(0)} \; \circ\!\!-\!\!\!-\!\!\!-\!\!\!-\!\!\circ \; a^{(1)}$$

We can write this function down as follows

$$a^{(1)} = \sigma(wa^{(0)} + b)$$

where b and w are just numbers

But σ (sigma) is itself a function.

Its useful at this point to give each of these ~~functions~~ terms a name, which will help you keep track of what is going on.

$a \Rightarrow$ "activity"

$w \Rightarrow$ "~~l~~weight"

$b \Rightarrow$ "bias"

$\sigma \Rightarrow$ "activation function"

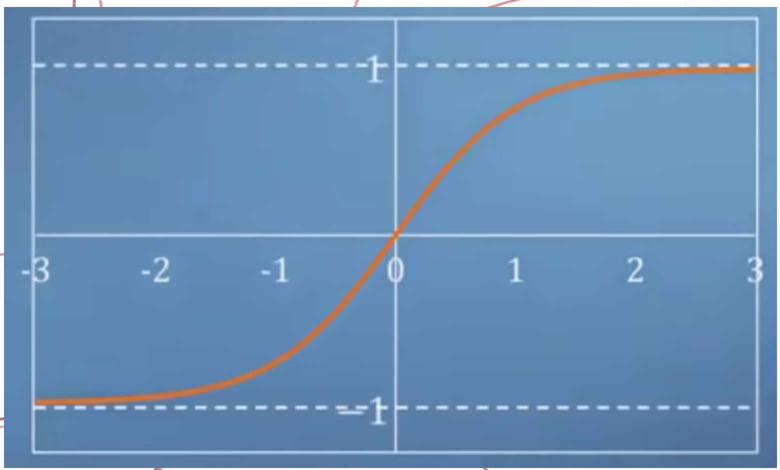
But why do all terms use a sensible letter except for σ (sigma)?

Its σ that gives NN'S its association to the Brain

Neurons in brain receive their ~~signals~~ information from their neighbours through Chemical and electrical stimulation

And when the sum of all these things stimulations goes beyond a certain threshold amount, the neuron is suddenly activated, and starts stimulating its neighbours ~~that~~ in turn.

An example of function that has this thresholding property is the hyperbolic tangent function, tanh



$$\sigma(x) = \tanh(x)$$

which is a nice, well behaved function, with a range $(-1, 1)$

we may not have met tanh before, but it is just a ratio of some exponential terms, and nothing our calculus tools can't already handle.

$$\sigma(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

tanh actually belongs to a family of similar functions all with the characteristic shape ('S')

Called Sigmoids

Hence, why we use σ (sigma) for this term.

OK, so here we are with our non linear function

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

that we can evaluate on our calculator

and also now know what all the terms are called

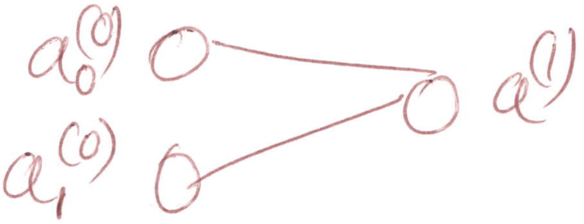
At start, we mentioned, that NN's can be used for image recognition, but so far our network, with its two scalar parameters w and b, does not look that it can do anything particularly interesting.

So what do we need to add?

Short answer, more neurons

So, now we just going (to add) start building up the complexity, while keeping track of how the notation adapts to cope.

Lets add an additional neuron to our input layer, we can still call the scalar output variable $a^{(1)}$, but we will need to tell difference between two inputs, lets call them $a_0^{(0)}$ and $a_1^{(0)}$

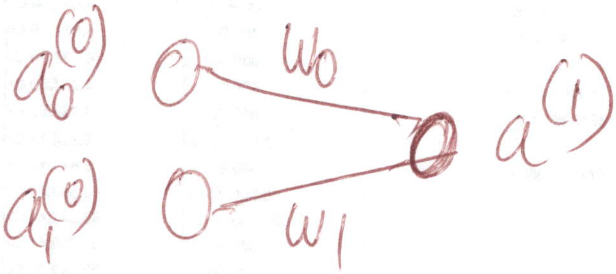


To include these new inputs to our equation, we simply say a_1 = σ of sum of these two inputs, each multiplied by their own weighting, plus the bias

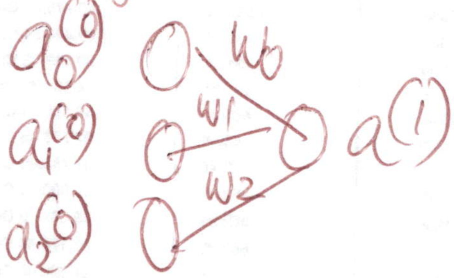
$$a^{(1)} = \sigma(w_0 a_0^{(0)} + w_1 a_1^{(0)} + b)$$

As we can see, each link in our network is associated with weight

So we can add those to our diagram



adding a third node, a_3^0 follows the same logic, and we just add this weight to our sum, But things are getting messy...



$$a^{(1)} = \sigma(w_0 a_0^{(0)} + w_1 a_1^{(0)} + w_2 a_2^{(0)} + b)$$

So, lets now generalize our expression to take n inputs, for which we can just the summation notation.

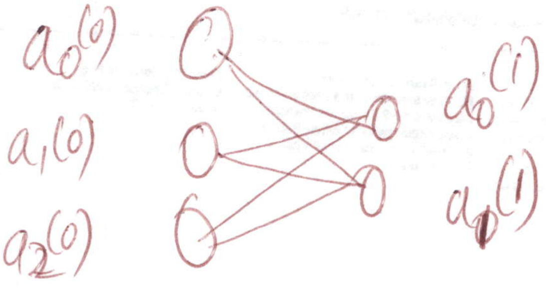
$$= \sigma\left(\left(\sum_{j=0}^{n} w_j a_j^{(0)}\right) + b\right)$$

or even better, notice that each input has a weight, so we can make vector of weights and vector of inputs, and just take the dot product to achieve the same effect

$$= \sigma(w \cdot a^{(0)} + b)$$

we can now have as many inputs as we ~~like~~ want in our input

Lets now apply the same logic to the output. $a_0^{(1)}$ and $a_1^{(1)}$



and they have <u>twice the number</u> of connections each one with its own weighting.

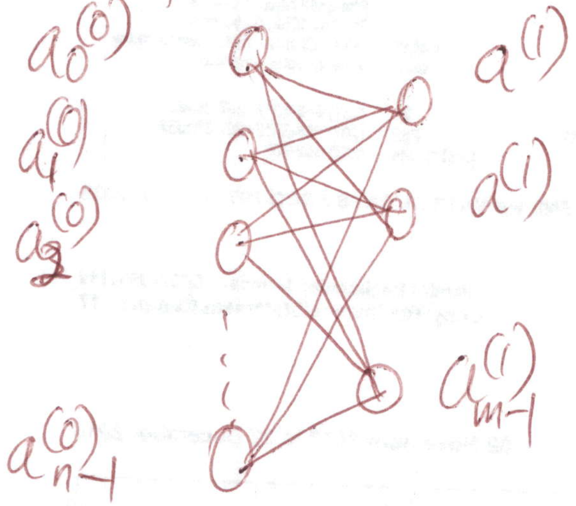
and each neuron has its own bias
So we can write a pair of equations to describe the scenario, with one for each of output.
where each equation contains the same values of $a^{(0)}$, but each has different bias and vector of weights.

$$a_0^{(1)} = \sigma(w_0 \cdot a^{(0)} + b_0)$$
$$a_1^{(1)} = \sigma(w_1 \cdot a^{(0)} + b_1)$$

Crunch down to compact vector form, where two outputs are each rows of column vector, meaning, we now hold our 2 weight vectors in weight matrix, and 2 biases in bias vector.

$$\boxed{a^{(1)} = \sigma(\mathbf{W}^{(1)} \cdot \mathbf{a}^{(0)} + \mathbf{b}^{(1)})}$$

Final Compact Equation in all its glory:

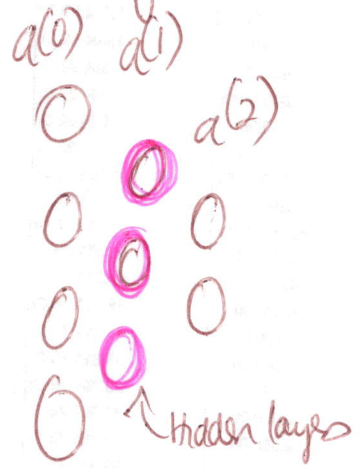


$$\boxed{a^{(1)} = \sigma\left(W^{(1)} \cdot a^{(0)} + b^{(1)}\right)}$$

Single layer Neural network, with m output, n inputs we can fully describe the function it represents, with the above equation

$$\begin{bmatrix} a_0^{(0)} \\ a_1^{(1)} \\ \vdots \\ a_{m-1}^{(1)} \end{bmatrix} = \sigma\left(\begin{bmatrix} w_{0,0}^{(1)} & w_{0,1}^{(1)} & \cdots & w_{0,n-1}^{(1)} \\ w_{1,0}^{(1)} & w_{1,1}^{(1)} & & w_{1,n-1}^{(1)} \\ \vdots & \vdots & & \vdots \\ w_{m-1,0}^{(1)} & w_{m-1,1}^{(1)} & & w_{m-1,n-1}^{(1)} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_{n-1}^{(0)} \end{bmatrix} + \begin{bmatrix} b_0^{(1)} \\ b_1^{(1)} \\ \vdots \\ b_{m-1}^{(1)} \end{bmatrix} \right)$$

But neural network have 1 or several layers of neurons, between inputs and outputs this is referred to as <u>hidden layers</u>



$$a^{(1)} = \sigma(w^{(1)} \cdot a^{(0)} + b^{(1)})$$
$$a^{(2)} = \sigma(w^{(2)} \cdot a^{(1)} + b^{(2)})$$

They behave in exactly the same way as we seen so far, except their output are the input to next layer.

with that we have all the LA in place for us to calculate the output of simple feedforward NN

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

13

Persuading your network to do something interesting, like image recognition, then becomes a matter of teaching it all the right weights and biases, which we will cover in next session