Activation functions

(DESCRIPTION)
Text, One hidden layer Neural Network. Activation functions. Website, deep learning, dot, A.I.

(SPEECH)
When you build your neural network, one of the choices you get to make is what activation function to use in the hidden layers, as well as what is the output units of your neural network.

So far, we've just been using the sigmoid activation function.

But sometimes other choices can work much better.

Let's take a look at some of the options.

(DESCRIPTION)
New slide, Activation functions.

(SPEECH)
In the fourth propagation steps for the neural network, we have these three steps where we use the sigmoid function here.

So that sigmoid is called an activation function.

And here's the familiar sigmoid function, a equals one over one plus e to the negative z.

So in the more general case, we can have a different function, g of z, which I'm going to write here, where g could be a nonlinear function that may not be the sigmoid function.

So for example, the sigmoid function goes within zero and one, and activation function that almost always works better than the sigmoid function is the tangent function or the hyperbolic tangent function.

So this is z, this is a, this is a equals tanh(z), and this goes between plus 1 and minus 1.

The formula for the tanh function is e to the z minus e to the negative z over their sum.

And is actually mathematically, a shifted version of the sigmoid function.

So, as a sigmoid function just like that, but shifted so that it now crosses a zero zero point and v scale, so it goes 15 minus 1 and plus 1.

And it turns out for hidden units, if you let the function g of z be equal to tanh(z), this almost always works better than the sigmoid function because the values between plus 1 and minus 1, the mean of the activations that come out of your head, and they are closer to having a 0 mean.

And so just as sometimes when you train a learning algorithm, you might center the data and have your data have 0 mean using a tanh instead of a sigmoid function.

It kind of has the effect of centering your data so that the mean of your data is closer to 0 rather than, maybe 0.5.

And this actually makes learning for the next layer a little bit easier.

We'll say more about this in the second course when we talk about optimization algorithms as well.

But one takeaway is that I pretty much never use the sigmoid activation function anymore.

The tanh function is almost always strictly superior.

The one exception is for the output layer because if y is either 0 or 1, then it makes sense for y hat to be a number, the one to output that's between 0 and 1 rather than between minus 1 and 1.

So the one exception where I would use the sigmoid activation function is when you are using binary classification, in which case you might use the sigmoid activation function for the output layer.

So g of z 2 here is equal to sigma of z 2.

And so what you see in this example is where you might have a tanh activation function for the hidden layer, and sigmoid for the output layer.

So deactivation functions can be different for different layers.

And sometimes to note that activation functions are different for different layers, we might use these square bracket superscripts as well to indicate that g of square bracket one may be different than g of square bracket two.

And again, square bracket one superscript refers to this layer, and superscript square bracket two refers to the output layer.

Now, one of the downsides of both the sigmoid function and the tanh function is that if z is either very large or very small, then the gradient or the derivative or the slope of this function becomes very small.

So if z is very large or z is very small, the slope of the function ends up being close to 0.

And so this can slow down gradient descent.

So one other choice that is very popular in machine learning is what's called the rectify linear unit.

So the value function looks like this.

And the formula is a = max(0,z).

So the derivative is 1, so long as z is positive.

And the derivative or the slope is 0, when z is negative.

If you're implementing this, technically the derivative when z is exactly 0 is not well defined.

But when you implement this in the computer, the answer you get exactly is z equals 0000000000000.

It's very small so you don't need to worry about it in practice.

You could pretend the derivative, when z is equal to 0, you can pretend it's either 1 or 0 and then you kind of work just fine.

So the fact that it's not differentiable, and the fact that, so here are some rules of thumb for choosing activation functions.

If your output is 0, 1 value, if you're using binary classification, then the sigmoid activation function is a very natural choice for the output layer.

And then for all other unit's ReLU, or the rectified linear unit, Is increasingly the default choice of activation function.

So if you're not sure what to use for your hidden layer, I would just use the ReLU activation function.

It's what you see most people using most days.

Although sometimes people also use the tanh activation function.

One disadvantage of the ReLU is that the derivative is equal to zero, when z is negative.

In practice, this works just fine.

But there is another version of the ReLU called the leaky ReLU.

I will give you the formula on the next slide.

But instead of it being 0 when z is negative, it just takes a slight slope like so, so this is called the leaky ReLU.

This usually works better than the ReLU activation function, although it's just not used as much in practice.

Either one should be fine, although, if you had to pick one, I usually just use the ReLU.

And the advantage of both the ReLU and the leaky ReLU is that for a lot of the space of Z, the derivative of the activation function, the slope of the activation function is very different from 0.

And so in practice, using the ReLU activation function, your neural network will often learn much faster than when using the tanh or the sigmoid activation function.

And the main reason is that there is less of these effects of the slope of the function going to 0, which slows down learning.

And I know that for half of the range of z, the slope of ReLU is 0, but in practice, enough of your hidden units will have z greater than 0.

So learning can still be quite fast for most training

(DESCRIPTION)
New slide, Pros and cons of activation functions.

(SPEECH)
examples.

So let's just quickly recap the pros and cons of different activation functions.

Here's a sigmoid activation function.

I will say never use this, except for the output layer, if you are doing binary classification, or maybe almost never use this.

And the reason I almost never use this is because the tanh is pretty much strictly superior.

So the tanh activation function is this.

And then the default, the most commonly used activation function is the ReLU, which is this.

So if you're not sure what else to use, use this one, and maybe feel free also to try the leaky ReLU.

Where it might be (0.01 z, z).

Right? So a is the max of 0.01 times z and z, so that gives you these some bends in the function.

And you might say, why is that constant 0.01?

Well, you can also make that another parameter of the learning algorithm.

And some people say that works even better.

But i hardly see people do that.

But if you feel like trying that in your application, please feel free to do so.

And you can just see how it works, and how well it works, and stick with it if it gives you a good result.

So I hope that gives you a sense of some of the choices of activation functions you can use in your neural network.

One of the themes we'll see in deep learning is that you often have a lot of different choices in how you code your neural network.

Ranging from number of hidden units, to the choice activation function, to how you initialize the ways which we'll see later.

A lot of choices like that.

And it turns out that it's sometimes difficult to get good guidelines for exactly what would work best for your problem.

So throughout these courses I keep on giving you a sense of what I see in the industry in terms of what's more or less popular.

But for your application, with your application's idiosyncrasies, it's actually very difficult to know in advance exactly what will work best.

So a common piece of advice would be, if you're not sure which one of these activation functions work best, try them all, and evaluate on a holdout validation set, or a development set, which we'll talk about later, and see which one works better, and then go with that.

And I think that by testing these different choices for your application, you'd be better at future-proofing your neural network architecture against the idiosyncracies of your problem, as well as evolutions of the algorithms.

Rather than if I were to tell you always use a ReLU activation and don't use anything else.

That just may or may not apply for whatever problem you end up working on either in the near future or in the distant future.

All right, so that was the choice of activation functions and you've seen the most popular activation functions.

There's one other question that sometimes you could ask, which is, why do you even need to use an activation function at all?

Why not just do away with that?

So let's talk about that in the next video, where you see why neural networks do need some sort of nonlinear activation function.