Training a softmax classifier

(DESCRIPTION)
Text, Multi-class classification. Training a softmax classifier. Website, deep learning, dot, A.I.

(SPEECH)
In the last video, you learned about the soft master, the softmax activation function.

In this video, you deepen your understanding of softmax classification, and also learn how the training model that uses a softmax

(DESCRIPTION)
New slide, Understanding softmax.

(SPEECH)
layer.

Recall our earlier example where the output layer computes z[L] as follows.

So we have four classes, c = 4 then z[L] can be (4,1) dimensional vector and we said we compute t which is this temporary variable that performs element y's exponentiation.

And then finally, if the activation function for your output layer, g[L] is the softmax activation function, then your outputs will be this.

It's basically taking the temporarily variable t and normalizing it to sum to 1.

So this then becomes a(L).

So you notice that in the z vector, the biggest element was 5, and the biggest probability ends up being this first probability.

The name softmax comes from contrasting it to what's called a hard max which would have taken the vector Z and matched it to this vector.

So hard max function will look at the elements of Z and just put a 1 in the position of the biggest element of Z and then 0s everywhere else.

And so this is a very hard max where the biggest element gets a output of 1 and everything else gets an output of 0.

Whereas in contrast, a softmax is a more gentle mapping from Z to these probabilities.

So, I'm not sure if this is a great name but at least, that was the intuition behind why we call it a softmax, all this in contrast to the hard max.

And one thing I didn't really show but had alluded to is that softmax regression or the softmax identification function generalizes the logistic activation function to C classes rather than just two classes.

And it turns out that if C = 2, then softmax with C = 2 essentially reduces to logistic regression.

And I'm not going to prove this in this video but the rough outline for the proof is that if C = 2 and if you apply softmax, then the output layer, a[L], will output two numbers if C = 2, so maybe it outputs 0.842 and 0.158, right?

And these two numbers always have to sum to 1.

And because these two numbers always have to sum to 1, they're actually redundant.

And maybe you don't need to bother to compute two of them, maybe you just need to compute one of them.

And it turns out that the way you end up computing that number reduces to the way that logistic regression is computing its single output.

So that wasn't much of a proof but the takeaway from this is that softmax regression is a generalization of logistic regression to more than two classes.

(DESCRIPTION)
New slide, Loss function.

(SPEECH)
Now let's look at how you would actually train a neural network with a softmax output layer.

So in particular, let's define the loss functions you use to train your neural network.

Let's take an example.

Let's see of an example in your training set where the target output, the ground true label is 0 1 0 0.

So the example from the previous video, this means that this is an image of a cat because it falls into Class 1.

And now let's say that your neural network is currently outputting y hat equals, so y hat would be a vector probability is equal to sum to 1.

0.1, 0.4, so you can check that sums to 1, and this is going to be a[L].

So the neural network's not doing very well in this example because this is actually a cat and assigned only a 20% chance that this is a cat.

So didn't do very well in this example.

So what's the last function you would want to use to train this neural network?

In softmax classification, they'll ask me to produce this negative sum of j=1 through 4.

And it's really sum from 1 to C in the general case.

We're going to just use 4 here, of yj log y hat of j.

So let's look at our single example above to better understand what happens.

Notice that in this example, y1 = y3 = y4 = 0 because those are 0s and only y2 = 1.

So if you look at this summation, all of the terms with 0 values of yj were equal to 0.

And the only term you're left with is -y2 log y hat 2, because we use sum over the indices of j, all the terms will end up 0, except when j is equal to 2.

And because y2 = 1, this is just -log y hat 2.

So what this means is that, if your learning algorithm is trying to make this small because you use gradient descent to try to reduce the loss on your training set.

Then the only way to make this small is to make this small.

And the only way to do that is to make y hat 2 as big as possible.

And these are probabilities, so they can never be bigger than 1.

But this kind of makes sense because x for this example is the picture of a cat, then you want that output probability to be as big as possible.

So more generally, what this loss function does is it looks at whatever is the ground true class in your training set, and it tries to make the corresponding probability of that class as high as possible.

If you're familiar with maximum likelihood estimation statistics, this turns out to be a form of maximum likelyhood estimation.

But if you don't know what that means, don't worry about it.

The intuition we just talked about will suffice.

Now this is the loss on a single training example.

How about the cost J on the entire training set.

So, the class of setting of the parameters and so on, of all the ways and biases, you define that as pretty much what you'd guess, sum of your entire training sets are the loss, your learning algorithms predictions are summed over your training samples.

And so, what you do is use gradient descent in order to try to minimize this class.

Finally, one more implementation detail.

Notice that because C is equal to 4, y is a 4 by 1 vector, and y hat is also a 4 by 1 vector.

So if you're using a vectorized limitation, the matrix capital Y is going to be y(1), y(2), through y(m), stacked horizontally.

And so for example, if this example up here is your first training example then the first column of this matrix Y will be 0 1 0 0 and then maybe the second example is a dog, maybe the third example is a none of the above, and so on.

And then this matrix Y will end up being a 4 by m dimensional matrix.

And similarly, Y hat will be y hat 1 stacked up horizontally going through y hat m, so this is actually y hat 1.

All the output on the first training example then y hat will these 0.3, 0.2, 0.1, and 0.4, and so on.

And y hat itself will also be 4 by m dimensional

(DESCRIPTION)
New slide, Gradient descent with softmax.

(SPEECH)
matrix.

Finally, let's take a look at how you'd implement gradient descent when you have a softmax output layer.

So this output layer will compute z[L] which is C by 1 in our example, 4 by 1 and then you apply the softmax attribution function to get a[L], or y hat.

And then that in turn allows you to compute the loss.

So with talks about how to implement the forward propagation step of a neural network to get these outputs and to compute that loss.

How about the back propagation step, or gradient descent?

Turns out that the key step or the key equation you need to initialize back prop is this expression, that the derivative with respect to z at the loss layer, this turns out, you can compute this y hat, the 4 by 1 vector, minus y, the 4 by 1 vector.

So you notice that all of these are going to be 4 by 1 vectors when you have 4 classes and C by 1 in the more general case.

And so this going by our usual definition of what is dz, this is the partial derivative of the class function with respect to z[L].

If you are an expert in calculus, you can derive this yourself.

Or if you're an expert in calculus, you can try to derive this yourself, but using this formula will also just work fine, if you have a need to implement this from scratch.

With this, you can then compute dz[L] and then sort of start off the back prop process to compute all the derivatives you need throughout your neural network.

But it turns out that in this week's primary exercise, we'll start to use one of the deep learning program frameworks and for those primary frameworks, usually it turns out you just need to focus on getting the forward prop right.

And so long as you specify it as a primary framework, the forward prop pass, the primary framework will figure out how to do back prop, how to do the backward pass for you.

So this expression is worth keeping in mind for if you ever need to implement softmax regression, or softmax classification from scratch.

Although you won't actually need this in this week's primary exercise because the primary framework you use will take care of this derivative computation for you.

So that's it for softmax classification, with it you can now implement learning algorithms to characterized inputs into not just one of two classes, but one of C different classes.

Next, I want to show you some of the deep learning programming frameworks which can make you much more efficient in terms of implementing deep learning algorithms.

Let's go on to the next video to discuss that.