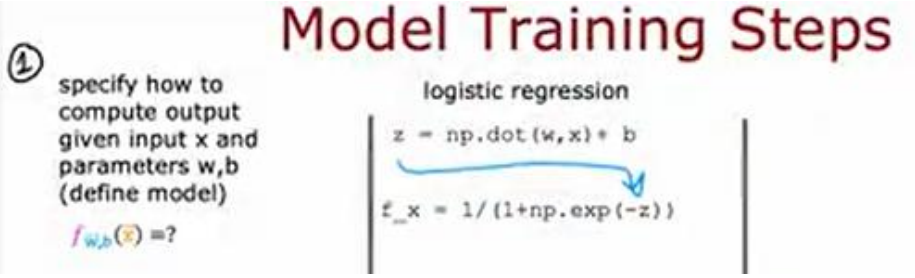
Training details

Let's take a look at the details of what the TensorFlow code for training a neural network is actually doing. Let's dive in. Before looking at the details of training in neural network, let's recall how you had trained a logistic regression model in the previous course. Step 1 of building a logistic regression model was you would specify how to compute the output given the input feature x and the parameters w and b.

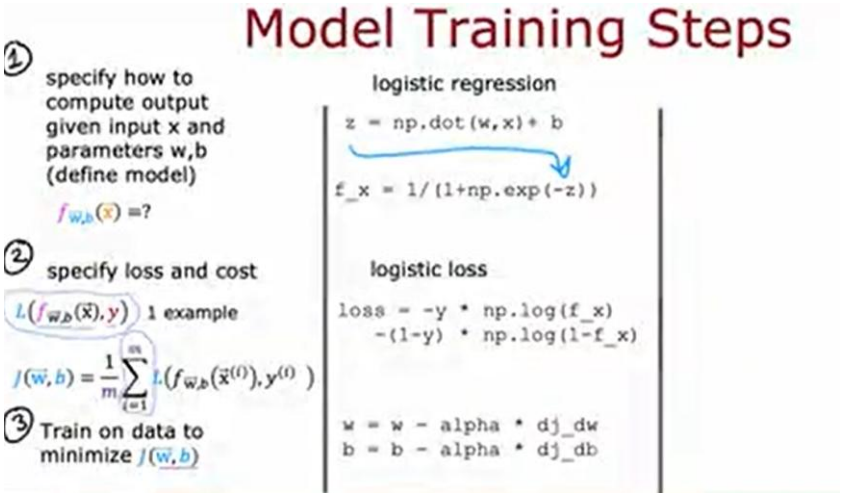
In the first course we said the logistic regression function predicts f of x is equal to G. The sigmoid function applied to W. product X plus B which was the sigmoid function applied to W.X plus B. If Z is the dot product of W of X plus B, then F of X is 1 over 1 plus e to the negative z, so those first step were to specify what is the input to output function of logistic regression, and that depends on both the input x and the parameters of the model.



The second step we had to do to train the literacy regression model was to specify the loss function and also the cost function, so you may recall that the loss function said, if religious regression opus f of x and the ground truth label, the actual label and a training set was y then the loss on that single training example was negative y log f of x minus one minus y times log of one minus f of x.

This was a measure of how well is logistic regression doing on a single training example x comma y. Given this definition of a loss function, we then define the cost function, and the cost function was a function of the parameters W and B, and that was just the average that is taking an average overall M training examples of the loss function computed on the M training examples, X1, Y1 through XMYM, and remember that in the convention we're using the loss function is a function of the output of the learning algorithm and the ground truth label as computed over a single training example whereas the cost function J is an average of the loss function computed over your entire training set.

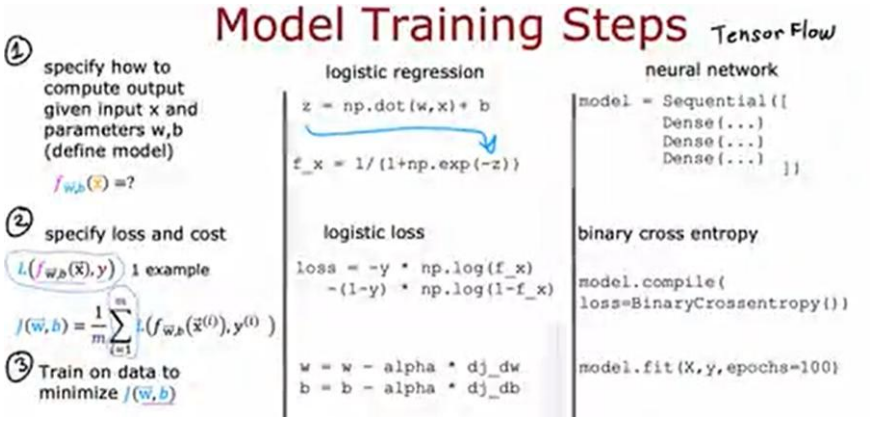
That was step two of what we did when building up logistic regression. Then the third and final step to train a logistic regression model was to use an algorithm specifically gradient descent to minimize that cost function J of WB to minimize it as a function of the parameters W and B. We minimize the cost J as a function of the parameters using gradient descent where W is updated as W minus the learning rate alpha times the derivative of J with respect to W. And B similarly is updated as B minus the learning rate alpha times the derivative of J with respect to B.



If these three steps. Step one, specifying how to compute the outputs given the input X and parameters, step 2 specify loss and costs, and step three minimize the cost function we trained logistic regression. The same three steps is how we can train a neural network in TensorFlow. Now let's look at how these three steps map to training a neural network.

We'll go over this in greater detail on the next three slides but really briefly. Step one is specify how to compute the output given the input x and parameters W and B that's done with this code snippet which should be familiar from last week of specifying the neural network and this was actually enough to specify the computations needed in forward propagation or for the inference algorithm for example.

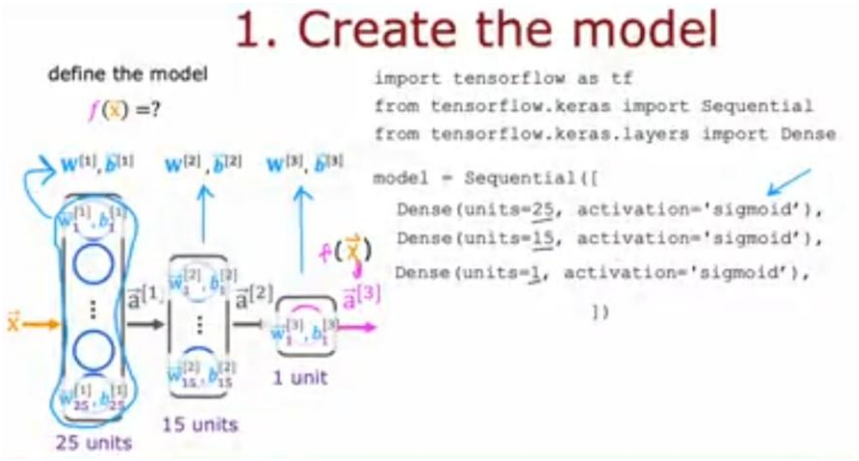
The second step is to compile the model and to tell it what loss you want to use, and here's the code that you use to specify this loss function which is the binary cross entropy loss function, and once you specify this loss taking an average over the entire training set also gives you the cost function for the neural network, and then step three is to call function to try to minimize the cost as a function of the parameters of the neural network.



Let's look in greater detail in these three steps in the context of training a neural network. The first step, specify how to compute the output given the input x and parameters w and b. This code snippet specifies the entire architecture of the neural network. It tells you that there are 25 hidden units in the first hidden layer, then the 15 in the next one, and then one output unit and that we're using the sigmoid activation value.

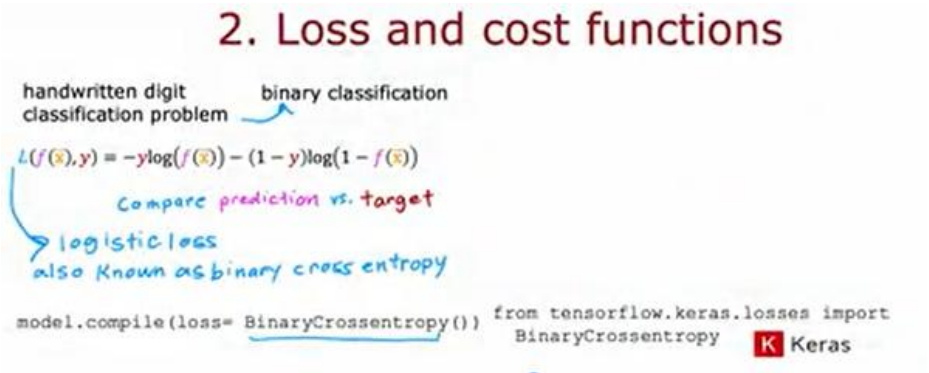
Based on this code snippet, we know also what are the parameters w1, v1 though the first layer parameters of the second layer and parameters of the third layer. This code snippet specifies the

entire architecture of the neural network and therefore tells TensorFlow everything it needs in In order to compute the output a 3 or f of x as a function of the input x and the parameters, here we have written w l and b l. Let's go on to step 2.



In the second step, you have to specify what is the loss function. That will also define the cost function we use to train the neural network. For the handwritten digit classification problem where images are either of a zero or a one the most common by far, loss function to use is this one is actually the same loss function as what we had for logistic regression is negative y log f of x minus 1 minus y times log 1 minus f of x, where y is the ground truth label, sometimes also called the target label y, and f of x is now the output of the neural network.

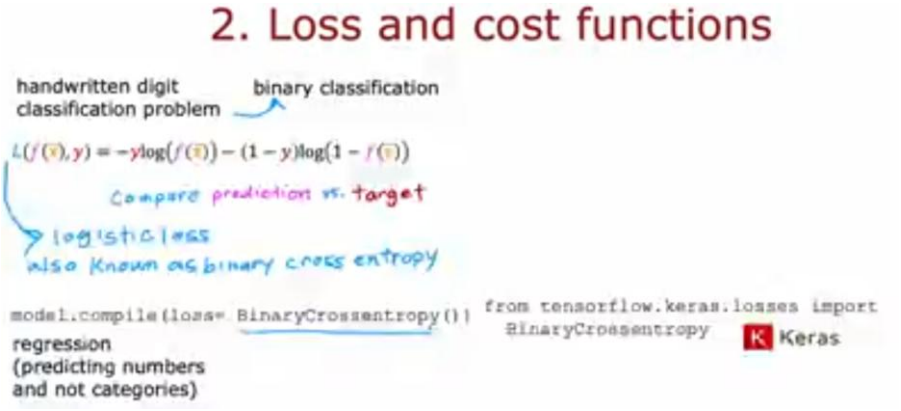
In TensorFlow, this is called the binary cross-entropy loss function. Where does that name come from? Well, it turns out in statistics this function on top is called the cross-entropy loss function, so that's what cross-entropy means, and the word binary just reemphasizes or points out that this is a binary classification problem because each image is either a zero or a one. The syntax is to ask TensorFlow to compile the neural network using this loss function.



Another historical note, carers was originally a library that had developed independently of TensorFlow is actually totally separate project from TensorFlow. But eventually it got merged into TensorFlow, which is why we have tf.Keras library.

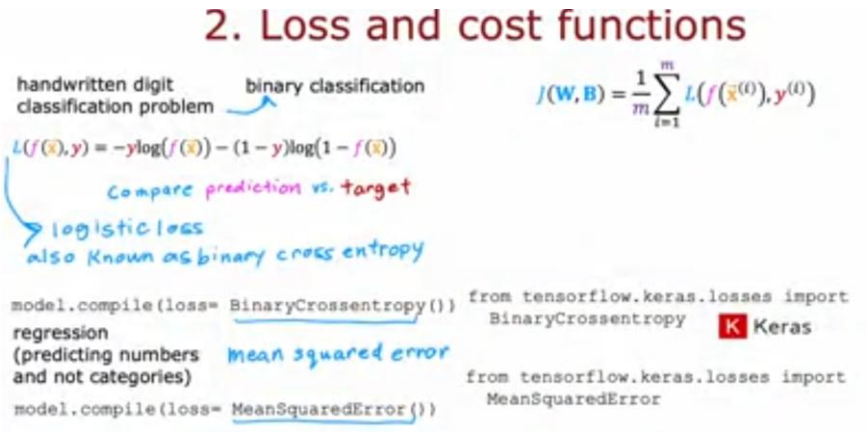
losses dot the name of this loss function. By the way, I don't always remember the names of all the loss functions and TensorFlow, but I just do a quick web search myself to find the right name and then I plug that into my code. Having specified the loss with respect to a single training example, TensorFlow knows that it costs you want to minimize is then the average, taking the average over all m training examples of the loss on all of the training examples.

Optimizing this cost function will result in fitting the neural network to your binary classification data. In case you want to solve a regression problem rather than a classification problem. You can also tell TensorFlow to compile your model using a different loss function.



For example, if you have a regression problem and if you want to minimize the squared error loss. Here is the squared error loss. The loss with respect to if your learning algorithm outputs f of x with a target or ground truth label of y, that's 1/2 of the squared error. Then you can use this loss function in TensorFlow, which is to use the maybe more intuitively named mean squared error loss function.

Then TensorFlow will try to minimize the mean squared error. In this expression, I'm using j of capital w comma capital b to denote the cost function. The cost function is a function of all the parameters into neural network.



You can think of capital W as including W1, W2, W3. All the W parameters and the entire new network and be as including b1, b2, and b3. If you are optimizing the cost function respect to w and b, if we tried to optimize it with respect to all of the parameters in the neural network.

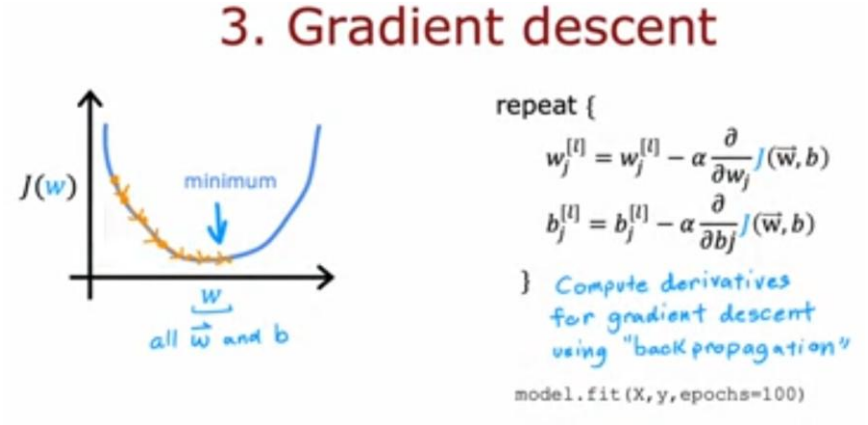
Up on top as well, I had written f of x as the output of the neural network, but we can also write f of w b if we want to emphasize that the output of the neural network as a function of x depends on all the parameters in all the layers of the neural network.

That's the loss function and the cost function. Finally, you will ask TensorFlow to minimize the cross-function. You might remember the gradient descent algorithm from the first course.

If you're using gradient descent to train the parameters of a neural network, then you are repeatedly, for every layer l and for every unit j, update wlj according to wlj minus the learning rate alpha times the partial derivative with respect to that parameter of the cost function j of wb and

similarly for the parameters b as well. After doing, say, 100 iterations of gradient descent, hopefully, you get to a good value of the parameters.

In order to use gradient descent, the key thing you need to compute is these partial derivative terms. What TensorFlow does, and, in fact, what is standard in neural network training, is to use an algorithm called backpropagation in order to compute these partial derivative terms. TensorFlow can do all of these things for you. It implements backpropagation all within this function called fit. All you have to do is call model.fit, x, y as your training set, and tell it to do so for 100 iterations or 100 epochs.



In fact, what you see later is that TensorFlow can use an algorithm that is even a little bit faster than gradient descent, and you'll see more about that later this week as well. Now, I know that we're relying heavily on the TensorFlow library in order to implement a neural network. One pattern I've seen across multiple ideas is as the technology evolves, libraries become more mature, and most engineers will use libraries rather than implement code from scratch. There have been many other examples of this in the history of computing.

Once, many decades ago, programmers had to implement their own sorting function from scratch, but now sorting libraries are quite mature that you probably call someone else's sorting function rather than implement it yourself, unless you're taking a computing class and I ask you to do it as an exercise.

Today, if you want to compute the square root of a number, like what is the square root of seven, well, once programmers had to write their own code to compute this, but now pretty much everyone just calls a library to take square roots, or matrix operations, such as multiplying two matrices together. When deep learning was younger and less mature, many developers, including me, were implementing things from scratch using Python or C++ or some other library.

But today, deep learning libraries have matured enough that most developers will use these libraries, and, in fact, most commercial implementations of neural networks today use a library like TensorFlow or PyTorch. But as I've mentioned, it's still useful to understand how they work under the hood so that if something unexpected happens, which still does with today's libraries, you have a better chance of knowing how to fix it. Now that you know how to train a basic neural network, also called a multilayer perceptron, there are some things you can change about the neural network that will make it even more powerful. In the next video, let's take a look at how you can swap in different activation functions as an alternative to the sigmoid activation function we've been using. This will make your neural networks work even much better