**Introduction to TensorFlow for Artificial Intelligence, Machine Learning, and Deep Learning**

by deeplearning.ai

**Taught by:**  [Laurence Moroney](https://www.coursera.org/instructor/lmoroney), AI Advocate

Google Brain

|  |  |
| --- | --- |
| **Basic Info** | Course 1 of 4 in the [TensorFlow in Practice Specialization](https://www.coursera.org/specializations/tensorflow-in-practice" \t "_blank) |
| **Level** | Intermediate |

[A primer in machine learning](https://www.coursera.org/learn/introduction-tensorflow/lecture/PoOzi?t=119" \t "_blank)

Ultimately machine learning is very similar but we're just flipping the axes.

[The ‘Hello World’ of neural networks](https://www.coursera.org/learn/introduction-tensorflow/lecture/kr51q?t=312" \t "_blank)

but the neural network isn't positive. So it will figure out a realistic value for Y. That's the second main reason. When using neural networks, as they try to figure out the answers for everything, they deal in probability.

Those six points are linear but there's no guarantee that for every X, the relationship will be Y equals 2X minus 1.

Ultimately there are two main reasons. The first is that you trained it using very little data.

SGD which stands for stochastic gradient descent.

The neural network has no idea of the relationship between X and Y, so it makes a guess. Say it guesses Y equals 10X minus 10. It will then use the data that it knows about, that's the set of Xs and Ys that we've already seen to measure how good or how bad its guess was. The loss function measures this and then gives the data to the optimizer which figures out the next guess. So the optimizer thinks about how good or how badly the guess was done using the data from the loss function. Then the logic is that each guess should be better than the one before. As the guesses get better and better, an accuracy approaches 100 percent, the term convergence is used

It's really good to understand that as you want to optimize your models but the nice thing for now about TensorFlow and keras is that a lot of that math is implemented for you in functions.

There are two function roles that you should be aware of though and these are loss functions and optimizers.

so it's a single neuron. Successive layers are defined in sequence, hence the word sequential.

There's only one dense here. So there's only one layer and there's only one unit in it, so it's a single neuron.

In keras, you use the word dense to define a layer of connected neurons

The simplest possible neural network is one that has only one neuron in it,

A neural network is basically a set of functions which can learn patterns

Python and TensorFlow and an API in TensorFlow called keras

[Coding a Computer Vision Neural Network](https://www.coursera.org/learn/introduction-tensorflow/lecture/HUiYr?t=20" \t "_blank)

They should always match. The first layer is a flatten layer with the input shaping 28 by 28. Now, if you remember our images are 28 by 28, so we're specifying that this is the shape that we should expect the data to be in

The last layer has 10 neurons in it because we have ten classes of clothing in the dataset. They should always match

[Using Callbacks to control training](https://www.coursera.org/learn/introduction-tensorflow/lecture/AIkt8?t=42" \t "_blank)

What we'll now do is write a callback in Python. Here's the code. It's implemented as a separate class, but that can be in-line with your other code. It doesn't need to be in a separate file. In it, we'll implement the on\_epoch\_end function, which gets called by the callback whenever the epoch ends. It also sends a logs object which contains lots of great information about the current state of training. For example, the current loss is available in the logs, so we can query it for certain amount. For example, here I'm checking if the loss is less than 0.4 and canceling the training itself. Now that we have our callback, let's return to the rest of the code, and there are two modifications that we need to make. First, we instantiate the class that we just created, we do that with this code. Then, in my model.fit, I used the callbacks parameter and pass it this instance of the class.

[Walk through a notebook with Callbacks](https://www.coursera.org/learn/introduction-tensorflow/lecture/WqpzX?t=32" \t "_blank)

the second epoch begins, it has already dropped below 0.4, but the callback hasn't been hit yet. That's because we set it up for on epoch end. It's good practice to do this, because with some data and some algorithms, the loss may vary up and down during the epoch, because all of the data hasn't yet been processed. So, I like to wait for the end to be sure

[What are convolutions and pooling?](https://www.coursera.org/learn/introduction-tensorflow/lecture/JSKji?t=145" \t "_blank)

Of these four, pick the biggest value and keep just that. So, for example, you can see it here. My 16 pixels on the left are turned into the four pixels on the right, by looking at them in two-by-two grids and picking the biggest value. This will preserve the features that were highlighted by the convolution, while simultaneously quartering the size of the image. We have the horizontal and vertical axes.

A quick and easy way to do this, is to go over the image of four pixels at a time, i.e, the current pixel and its neighbors underneath and to the right of it.

Now, that's a very basic introduction to what convolutions do, and when combined with something called pooling, they can become really powerful. But simply, pooling is a way of compressing an image.

The idea here is that some convolutions will change the image in such a way that certain features in the image get emphasized.

Repeat this for each neighbor and each corresponding filter value, and would then have the new pixel with the sum of each of the neighbor values multiplied by the corresponding filter value, and that's a convolution.

For every pixel, take its value, and take a look at the value of its neighbors. If our filter is three by three, then we can take a look at the immediate neighbor, so that you have a corresponding three by three grid. Then to get the new value for the pixel, we simply multiply each neighbor by the corresponding value in the filter.

So, what's convolution? You might ask. Well, if you've ever done any kind of image processing, it usually involves having a filter and passing that filter over the image in order to change the underlying image.

[Implementing convolutional layers](https://www.coursera.org/learn/introduction-tensorflow/lecture/PjlKf?t=73" \t "_blank)

Now, of course, you might wonder what the 64 filters are. It's a little beyond the scope of this class to define them, but they aren't random. They start with a set of known good filters in a similar way to the pattern fitting that you saw earlier, and the ones that work from that set are learned over time.

Here we're specifying the first convolution. We're asking keras to generate 64 filters for us. These filters are 3 by 3, their activation is relu, which means the negative values will be thrown way, and finally the input shape is as before, the 28 by 28. That extra 1 just means that we are tallying using a single byte for color depth. As we saw before our image is our gray scale, so we just use one byte.

But remember, it's not just one compress five-by-five image instead of the original 28 by 28, there are a number of convolutions per image that we specified, in this case 64. So, there are 64 new images of five-by-five that had been fed in. Flatten that out and you have 25 pixels times 64, which is 1600. So, you can see that the new flattened layer has 1,600

so we're down to 11 by 11, add another two-by-two max-pooling to have this rounding down, and went down, down to five-by-five images

max-pooling layers. Now, remember we specified it to be two-by-two, thus turning four pixels into one, and having our x and y. So, now our output gets reduced from 26 by 26, to 13 by 13.

If your filter is five-by-five for similar reasons, your output will be four smaller on x, and four smaller on y. So, that's y with a three by three filter, our output from the 28 by 28 image, is now 26 by 26, we've removed that one pixel on x and y, and each of the borders.

After all, isn't the data 28 by 28, so y is the output, 26 by 26. The key to this is remembering that the filter is a three by three filter. Consider what happens when you start scanning through an image starting on the top left. So, for example with this image of the dog on the right, you can see zoomed into the pixels at its top left corner. You can't calculate the filter for the pixel in the top left, because it doesn't have any neighbors above it or to its left. In a similar fashion, the next pixel to the right won't work either because it doesn't have any neighbors above it

model.summary method. This allows you to inspect the layers of the model, and see the journey of the image through the convolutions, and here is the output. It's a nice table showing us the layers, and some details about them including the output shape.

This next line of code will then create a pooling layer. It's max-pooling because we're going to take the maximum value. We're saying it's a two-by-two pool, so for every four pixels, the biggest one will survive as shown earlier.

[Improving the Fashion classifier with convolutions](https://www.coursera.org/learn/introduction-tensorflow/lecture/BiLp0?t=128" \t "_blank)

The Keras API gives us each convolution and each pooling and each dense, etc. as a layer. So with the layers API, I can take a look at each layer's outputs, so I'll create a list of each layer's output. I can then treat each item in the layer as an individual activation model if I want to see the results of just that layer.

[Understanding ImageGenerator](https://www.coursera.org/learn/introduction-tensorflow/lecture/kqRHk?t=212" \t "_blank)

class mode. Now, this is a binary classifier

The images will be loaded for training and validation in batches where it's more efficient than doing it one by one.

when you use other datasets they may not always be uniformly sized. So this is really useful for you

Now, images might come in all shapes and sizes and unfortunately for training a neural network, the input data all has to be the same size, so the images will need to be resized to make them consistent.

It's a common mistake that people point the generator at the sub-directory. It will fail in that circumstance. You should always point it at the directory that contains sub-directories that contain your images. The names of the sub-directories will be the labels for your images that are contained within them. So make sure that the directory you're pointing to is the correct one.

[Defining a ConvNet to use complex images](https://www.coursera.org/learn/introduction-tensorflow/lecture/DzQa3?t=77" \t "_blank)

Remember before when you created the output layer, you had one neuron per class, but now there's only one neuron for two classes. That's because we're using a different activation function where sigmoid is great for binary classification, where one class will tend towards zero and the other class tending towards one. You could use two neurons here if you want, and the same softmax function as before, but for binary this is a bit more efficient.

So there are three bytes per pixel. One byte for the red, one for green, and one for the blue channel, and that's a common 24-bit color pattern.

[Training the ConvNet with fit\_generator](https://www.coursera.org/learn/introduction-tensorflow/lecture/hs4xc?t=163" \t "_blank)

multi-class classification with Softmax. Where you'll get a list of values with one value for the probability of each class and all of the probabilities adding up to 1.

And the verbose parameter specifies how much to display while training is going on. With verbose set to 2, we'll get a little less animation hiding the epoch progress.

training generator that you set up earlier. This streams the images from the training directory. Remember the batch size you used when you created it, it was 20, that's important in the next step. There are 1,024 images in the training directory, so we're loading them in 128 at a time. So in order to load them all, we need to do 8 batches. So we set the steps\_per\_epoch to cover that.

model.fit\_generator, and that's because we're using a generator instead of datasets.

RMSprop, where you can adjust the learning rate to experiment with performance.

ten items of fashion, you might remember that your loss function was a categorical cross entropy. But because we're doing a binary choice here, let's pick a binary\_crossentropy instead