TensorFlow

(DESCRIPTION)
Text, Programming Frameworks. Tensor Flow. Website, deep learning, dot, A.I.

(SPEECH)
Welcome to the last video for this week.

There are many great, deep learning programming frameworks.

One of them is TensorFlow.

I'm excited to help you start to learn to use TensorFlow.

What I want to do in this video is show you the basic structure of a TensorFlow program, and then leave you to practice, learn more details, and practice them yourself in this week's problem exercise.

This week's problem exercise will take some time to do so please be sure to leave some extra time to do

(DESCRIPTION)
New slide, Motivating problem.

(SPEECH)
it.

As a motivating problem, let's say that you have some cost function J that you want to minimize.

And for this example, I'm going to use this highly simple cost function J(w) = w squared- 10w + 25.

So that's the cost function.

You might notice that this function is actually (w- 5) squared.

If you expand out this quadratic, you get the expression above, and so the value of w that minimizes this is w = 5.

But let's say we didn't know that, and you just have this function.

Let's see how you can implement something in TensorFlow to minimize this.

Because a very similar structure of program can be used to train neural networks where you can have some complicated cost function J(w, b) depending on all the parameters of your neural network.

And the, similarly, you'll be able to use TensorFlow so automatically try to find values of w and b that minimize this cost function.

But let's start with the simpler example on

(DESCRIPTION)
New slide, Python is opened, a Tensor Flow example.

(SPEECH)
the left.

So, I'm running Python in my Jupyter notebook and to start up TensorFlow, you import numpy as np and it's idiomatic to use import tensorflow as tf.

Next, let me define the parameter w.

So in TensorFlow, you're going to use tf.Variable to define a parameter.

Dtype=tf.float32.

And then let's define the cost function.

So remember the cost function was w squared- 10w + 25.

So let me use tf.add.

So I'm going to have w squared + tf.multiply.

So the second term was -10.w.

And then I'm going to add that 25.

So let me put another tf.add over there.

So that defines the cost J that we had.

And then, I'm going to write train = tf.train.GradientDescentOptimizer.

Let's use a learning rate of 0.01 and the goal is to minimize the cost.

And finally, the following few lines are quite idiomatic.

Init = tf.global_variables_initializer and then session = tf.Sessions.

So it starts a TensorFlow session.

Session.run init to initialize global variables.

And then, for TensorFlow's evaluative variable, we're going to use sess.run w.

We haven't done anything yet.

So with this line above, initialize w to zero and define a cost function.

We define train to be our learning algorithm which uses a GradientDescentOptimizer to minimize the cost function.

But we haven't actually run the learning algorithm yet, so session.run, we evaluate w, and let me print(session.run(w).

So if we run that, it evaluates w to be equal to 0 because we haven't run anything yet.

Now, let's do session.run train.

So what this will do is run one step of GradientDescent.

And then let's evaluate the value of w after one step of GradientDescent and print that.

So we do that of the one set of GradientDescent, w is now 0.1.

Let's now run 1000 iterations of GradientDescent so .run(train).

And lets then print(session.run w).

So this would run a 1,000 iterations of GradientDescent, and at the end w ends up being 4.9999.

Remember, we said that we're minimizing w- 5 squared so the absolute value of w is 5 and it got very close to this.

So hope this gives you a sense of the broad structure of a TensorFlow program.

And as you do the following exercise and play with more TensorFlow course yourself, some of these functions that I'm using here will become more familiar.

Some things to notice about this, w is the parameter which I optimize so we're going to declare that as a variable.

And notice that all we had to do was define a cost function using these add and multiply and so on functions.

And TensorFlow knows automatically how to take derivatives with respect to the add and multiply as was other functions.

Which is why you only had to implement basically four prop and it can figure out how to do the back problem or the grading computation.

Because that's already built in to the add and multiply as well as the squaring functions.

By the way, in cases notation seems really ugly, TensorFlow actually has overloaded the computation for the usual plus, minus, and so on.

So you could also just write this nicer format for the cost and comment that out and rerun this and get the same result.

So once w is declared to be a TensorFlow variable, the squaring, multiplication, adding, and subtraction operations are overloaded.

So you don't need to use this uglier syntax that I had above.

Now, there's just one more feature of TensorFlow that I want to show you, which is this example minimize a fix function of w.

One of the function you want to minimize is the function of your training set.

So whether you have some training data, x and when you're training a neural network the training data x can change.

So how do you get training data into a TensorFlow program?

So I'm going to define t and x which is think of this as train a relevant training data or really the training data with both x and y, but we only have x in this example.

So just going to define x with placeholder and it's going to be a type float32 and let's make this a [3,1] array.

And what I'm going to do is whereas the cost here have fixed coefficients in front of the three terms in this quadratic was 1 times w squared- 10*w + 25.

We could turn these numbers 1- 10 and 25 into data.

So what I'm going to do is replace the cost with cost = x[0][0]*w squared + x[1][0]*w + x[2][0].

Well, times 1.

So now x becomes sort of like data that controls the coefficients of this quadratic function.

And this placeholder function tells TensorFlow that x is something that you provide the values for later.

So let's define another array, coefficient = np.array, [1.], [-10.] and yes, the loss value was [25.].

So that's going to be the data that we're going to plug into x.

So finally we need a way to get this array coefficients into the variable x and the syntax to do that is just doing the training step.

That the values for will need to be provided for x, I'm going to set here, feed_dict = x:coefficients, And I'm going to change this, I'm going to copy and paste put that there as well.

All right, hopefully, I didn't have any syntax errors.

Let's try re-running this and we get the same results hopefully as before.

And now, if you want to change the coefficients of this quadratic function, let's say you take this [-10.] and change it to 20, [-20].

And let's change this to 100.

So this is now a function x- 10 squared.

And if I re-run this, hopefully, I find that the value that minimizes x- 10 squared is w = 10.

Let's see, cool, great and we get w very close to 10 after running 1,000 integrations of GradientDescent.

So what you see more of when you do that from exercise is that a placeholder in TensorFlow is a variable whose value you assign later.

And this is a convenient way to get your training data into the cost function.

And the way you get your data into the cost function is with this syntax when you're running a training iteration to use the feed_dict to set x to be equal to the coefficients here.

And if you are doing mini batch GradientDescent where on each iteration, you need to plug in a different mini batch, then on different iterations you use the feed_dict to feed in different subsets of your training sets.

Different mini batches into where your cost function is expecting to see data.

So hopefully this gives you a sense of what TensorFlow can do.

And the thing that makes this so powerful is all you need to do is specify how to compute the cost function.

And then, it takes derivatives and it can apply a gradient optimizer or an add-on optimizer or some other optimizer with just pretty much one or two lines of codes.

(DESCRIPTION)
New slide, Code example. Import numpy as N.P. Import Tensor Flow as T.F. Coeffecients equal N.P. dot, array, parentheses, bracket, 1, bracket, comma, bracket, 25, bracket, parentheses. New line, w equals T.F., variable, parentheses, bracket, 0, bracket, comma, D type, equals, T.F., dot, float, 3, 2, parentheses. New line, X equals T.F., dot, place holder, parentheses, T.F., dot, float, 3, 2, bracket, 3, comma, 1, bracket, parentheses. New line, cost, equals x, bracket, zero, bracket, bracket, 0, bracket, star, W, star, star, 2, plus, x, bracket, 1, bracket, bracket, 0, bracket, star, W, plus, X, bracket, 2, bracket, bracket, 0, bracket, pound sign, W, minus, 5, star, star, 2. New line, train, equals, T.F., dot, train, dot, gradient descent optimizer, parentheses, 0.01, parentheses, dot, minimize, parentheses, cost, parentheses. New line, in it, equals, T.F., dot, global, underscore, variables, initializer, parentheses, parentheses. New line, session, equals, T.F., dot, Session, parentheses, parentheses. New line, session, dot, run, parenthese, in it, parentheses. New line, print, parentheses, session, dot, run, parentheses W, parentheses. New line, for i in range 1000: session, dot, run, parentheses, train, feed, underscore, dict, equals, parentheses, x, colon, coefficients, parentheses. Final line, print, parentheses, session, dot, run, parentheses, W, parentheses.

(SPEECH)
So here's the code again.

I've cleaned this up just a little bit.

And in case some of these functions or variables seem a little bit mysterious to use, they will become more familiar after you've practiced with it a couple times by working through their problem exercise.

Just one last thing I want to mention.

These three lines of code are quite idiomatic in TensorFlow, and what some programmers will do is use this alternative format.

Which basically does the same thing.

Set session to tf.Session() to start the session, and then use the session to run init, and then use the session to evaluate, say, w and then print the result.

But this with construction is used in a number of TensorFlow programs as well.

It more or less means the same thing as the thing on the left.

But the with command in Python is a little bit better at cleaning up in cases an error in exception while executing this inner loop.

So you see this is the following exercise as well.

So what is this code really doing?

Let's focus on this equation.

The heart of a TensorFlow program is something to compute at cost, and then TensorFlow automatically figures out the derivatives in how to minimize that costs.

So what this equation or what this line of code is doing is allowing TensorFlow to construct a computation draft.

And a computation draft does the following, it takes x[0][0], it takes w and then it goes w gets squared.

And then x[0][0] gets multiplied with w squared, so you have x[0][0]*w squared, and so on, right?

And eventually, you know, this gets built up to compute this xw, x[0][0]*w squared + x[1][0]*w + and so on.

And so eventually, you get the cost function.

And so the last term to be added would be x [2][0] where it gets added to be the cost.

I won't write other format for the cost.

And the nice thing about TensorFlow is that by implementing basically four prop applications through this computation draft, the computed cost, TensorFlow already has that built in.

All the necessary backward functions.

So remember how training a deep neural network has a set of forward functions instead of backward functions.

Programming frameworks like Tensor Flow have already built-in the necessary backward functions.

Which is why by using the built-in functions to compute the forward function, it can automatically do the backward functions as well to implement back propagation through even very complicated functions and compute derivatives for you.

So that's why you don't need to explicitly implement back prop.

This is one of the things that makes the programming frameworks help you become really efficient.

If you look at the TensorFlow documentation, I just want to point out that the TensorFlow documentation uses a slightly different notation than I did for drawing the computation draft.

So it uses x[0][0] w.

And then, rather than writing the value, like w squared, the TensorFlow documentation tends to just write the operation.

So this would be a, square operation, and then these two get combined in the multiplication operation and so on.

And then, a final note, I guess that would be an addition operation where you add x to 0 to find the final value.

So for the purposes of this class, I thought that this notation for the computation draft would be easier for you to understand.

But if you look at the TensorFlow documentation, if you look at the computation drafts in the documentation, you see this alternative convention where the notes are labeled with the operations rather than with the value.

But both of these representations represent basically the same computation draft.

And there are a lot of things that you can with just one line of code in programming frameworks.

For example, if you don't want to use GradientDescent, but instead you want to use the add-on Optimizer by changing this line of code, you can very quickly swap it, swap in a better optimization algorithm.

So all the modern deep learning programming framework support things like this and makes it really easy for you to code up even pretty complex neural networks.

So I hope this is helpful for giving you a sense of the typical structure of a TensorFlow program.

To recap the material from this week, you saw how to systematically organize the hyper parameter search process.

We also talked about batch normalization and how you can use that to speed up training of your neural networks.

And finally, we talked about programming frameworks of deep learning.

There are many great programming frameworks.

And we had this last video focusing on TensorFlow.

With that, I hope you enjoyed this week's programming exercise and that helps you gain even more familiarity with these ideas.