Week-3-2-First-TF-demo-script

(Off-Jupyter)
Now are are ready for our first TensorFlow demonstration.
The first thing we want to see is how to create simple graphs.
Let’s see how it works.
(Open Jupyter)
Here is our first Jupyter notebook, which is partly based on a notebook from the book ”Hands-on Machinle Learning with Scikit-Learn and TensorFlow" by A. Geron.
Let’s start going over its cells and see what they do.
The first cell are conventional imports, where we import numpy and matplotlib.
The second cell contains a utility function that we will use to clean the graph from time to time.
Now let’s move on with TensorFlow itself.
The first thing first, we import TensorFlow.
Then we clear the default graph, and create three nodes for two variables X and Y, and a constant A. (CELL 4)
They will be initialized to values 3, 4 and 2, at the run-time.
Now let’s check that our created nodes are on the default graph. (CELL 5)
As a short digression, nodes can also be created on some other graph. Here is an example of a node created on a separate graph rather than on the default graph. (CELL 6)
Let’s note that nodes that we created are not yet initialized. If we execute Cell 7, this is the result. (CELL 7)
Now, let’s create a function F equal X squared plus Y plus A. (CELL 8)
If we now print F, we see that it is a Tensor Op of type Add. (CELL 9)
There is no value of F, which is an example of LAZY EVALUATION in TensorFlow.
Now, everything is ready to run our first graph in TensorFlow.
To this end, we need to start a TensorFlow SESSION.
After we open a Session, we have to initialize our variables, and then evaluate the function. (CELL 10).
Please note that a constant does not need to be initialized.
Also note that a session should be manually closed after a run.
There is a more convenient way to automatically close the session using the “WITH” construction, as shown in this cell. (CELL 11).
Finally, this whole code can be made even shorter if we introduce a new node at the graph at the construction stage, which takes care of initialization of ALL VARIABLES AT ONCE.
We do it as shown in this cell. (CELL 12)
To check what node was created by TensorFlow, we can type INIT to see the result. (CELL 13)
Our next illustration shows the the lifecycle of a node value.
When you create a node, it only holds a value during an executing phase, when you run a TensorFlow session.
Let’s consider this example in CELL 14.
Here we create a node for a constant W equal 3, and then create three tensors X, Y, and Z.
Let’s assume we want to compute the values of Y and Z.
One way to do it would be like shown in CELL 14.
In this case, the TensorFlow graph would be traversed twice, to compute the values of Y and Z independently of each other, even though both Y and Z use the same value of X.
It is important to remember that all node values are dropped between different runs of the graph.
The only exception are VARIABLES, who start their life when their initializers are called, and end it when the session is closed.
So, after we run CELL 14, and see what is the value of X, we will see an un-initialized Tensor again! (CELL 15).
Also note that in CELL 13, the code evaluates both W and X twice, to calculate Y and Z in two separate runs of the graph!
This can be done more efficiently, by telling TensorFlow to do all calculations in one passing of the graph.
The syntax for this is shown in CELL 16.

Next, I would like to demonstrate the working of reverse-mode autodiff in TensorFlow.
Consider the function shown here.
It is an exponent of an exponent of an exponent.

This function is actually pretty similar to a loss function that is implemented by neuron networks, so this example might be useful for your understanding of the working of neural networks in TensorFlow.
Let’s see how we implement this function.
This is quite straightforward, and is shown in this cell (CELL 17).
Here, we first define the tensor-valued output of the input layer, then use it as an input to calculate the output of the second layer, and then finally use the last output to produce the value of the function.
We can put some syntactic sugar on top of it, and define each layer within its own scope, like is shown in this Cell (CELL 18).
This is useful for a visualization of the TensorFow graph, as we will see a bit later.
Now, let’s specify a point at which we want to compute the derivative.
I want to use a point where all intercepts W 00, W 10, W 20 are zeros, while all slopes W 01, W 11, and W 21 are ones.
So we do it here (CELL 19), and verify here (CELL 20).
Next, we compute all derivatives analytically, which is presented here.
You can go through the details of this analytic calculation with a notepad and a pencil after this video, but for now let’s see how TensorFlow computes these derivatives.
So, we clear the graph to start it from scratch, and create two variable nodes W and X, both of the floating type. (CELL 21)
Then we call our function, and return the values of all the layers F2, F1, and F0.
The next line is a KEY LINE.
Here we define a node for gradients of the outer function F2 with respect to all parameters W in the function.
This is simply done by calling TF.Gradient with two arguments: the name of the function, and variables with respect to which we want to compute the gradients .
Let’s execute this cell and see the node for Grads (EXECUTE CELL 21).
Now let’s move to the next CELL 22.
This is where we run our TensorFlow Graph.
We can run it twice to compute the values of functions and derivatives separately.
This is the code commented out here.
But for more efficiency, as we discussed a moment ago, it is better to run them in ONE TRAVERSAL of the graph, as shown here.
Let’s execute this cell and see what it produces (EXECUTE CELL 22).
And bum! We get 6 numbers, for six gradients of our functions.
You can verify manually that these numbers are correct using the explicit formulas above.
Now, after we ran the session, we can check the state of a node that computes the function.
Let’s run CELL 24 to see the result (EXECUTE CELL 23)
As expected, we see three Tensor nodes.
Finally, we can visualize the TensorFlow graph.
There are two ways to do it: either in a Jupyter notebook, or using TensorBoard.
Here I show the first method.
The magic sauce for this is given in CELL 24 using code in Geron’s book.
Let’s run this cell, and then run CELL 25.
And bum!
We get a visualization of a TensorFlow graph!
Now you can see why we introduced name scopes in our definition of the function.
TensorFlow puts all name scopes in separate boxes like these ones.
We can click on them and see Ops inside these boxes.
Later in our demos, I will show you how to run TensorBoard to visualize the graph, as well as to see the performance of training algorithms.
(Off-Jupyter:)
So, this was our first demo of TensorFlow.
We got familiar with the TensorFlow Graphs, Ops, Variables and Constants.
We also saw the working of the reverse-mode autodiff algorithm in TensorFlow, that enables automatic gradients of arbitrary functions.
In the next video, we will apply this automatic gradient calculation in practice to estimate a linear regression model.

(8:32 min in total)