Neural Network fundamentals with TensorFlow

# Course Outline

## Tensorflow Basics

* Creation, Initializing, Saving and Restoring TensorFlow variables
* Feeding, Reading and Preloading TensorFlow data
* How to use TensorFlow infrastructure to train models at scale
* Visualizing and Evaluating models with TensorBoard

## TensorFlow Mechanics

* Inputs and Placeholders
* Build the Graph
  + Inference
  + Loss
  + Training
* Train the model
  + The graph
  + The session
  + Train loop
* Evaluate the model.
  + Build the eval graph
  + Eval output

## The perceptron

* Activation functions
* The perceptron learning algorithm
* Binary classification with the perceptron
* Document classification with the perceptron
* Limitations of the perceptron

## Support Vector Machines

* Kernels and the kernel trick.
* Maximum margin classification and support vectors

## Artificial Neural Networks

* Nonlinear decision boundaries
* Feedforward and feedback artificial neural networks
* Multilayer perceptrons
* Minimizing the cost function
* Forward propagation
* Back propagation
* Improving the way neural networks learn

## Convolutional Neural Networks

* Goals
* Model architecture
* Principles
* Code organization
* Launching and training the model.
* Evaluating a model.

# TensorFlow Basics.

## What is TensorFlow?

TensorFlow is an open source library and application programming interface (API) for building programs that employ machine learning for a range of tasks. Components of TensorFlow include a symbolic math library as well as interfaces for building and training machine learning applications to detect and decipher patterns and correlations in data. This is similar to the way that biological entities learn and reason.

TensorFlow was created and made available by Google, Inc and was released under the Apache 2.0 Open Source License on 9 November, 2015.

TensorFlow was based on an earlier system, also built at Google called DistBelief. DistBelief gained increasing popularity among Google’s (Later Alphabet) different companies and divisions for use in both commercial and research applications. Google spent significant resources to simply and refactor the code base of DistBelief into a robust, performant applications-grade library which was then called TensorFlow. Version 1.0.0 of TensorFlow was released on 11 February 2017. TensorFlow can run on single or multiple CPU’s and GPU’s. TensorFlow is available on 64 bit Linux, MacOS, Windows and mobile platforms including iOS and Android.

TensorFlow provides a number of programmatic API’s for various computer languages, including Python, C++, Java, GoLang and Rust.

Additionally, third party API’s are available for C#, Julia, R and Scala.

TensorFlow is known for use in image processing and recognition, deep learning with AlphaGo and other cutting edge applications in the field of Machine Learning.

## How does TensorFlow work?

TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that these neural networks and other machine learning algorithms perform on multidimensional data arrays, known as *tensors*.

Tensors are formally known as mutilinear maps from vector spaces to real numbers, so, for example,

* A scalar is a tensor
* A vector is a tensor
* A matrix is a tensor

Basically, tensors can be represented as a multidimensional array of numbers.

### TensorFlow vs. Numpy

TensorFlow and Numpy are similar. Both are N-d array libraries. While Numpy has Ndarray support, it doesn’t offer methods to create tensor functions and automatically compute derivatives. Additionally, numpy offers no support for using GPU’s, unlike TensorFlow.

Let’s take a look at a similar program using both standard Python with Numpy and a TensorFlow application.

Table 1. A simple numpy program.

|  |
| --- |
| #!/usr/bin/env python3  import numpy as np  # Create an array 'a' as a 2 by 2 dimensional array  # initialized to zeros.  a = np.zeros((2,2))  # Create an array 'b' as a 2 by 2 dimensional array initialized to  # ones.  b = np.ones((2,2))  # Add the two up. The axis parameter refers to columsn vs. rows.  # axis=0 refers aggregation along the row, axis=1 refers to  # aggregation along the columns.  print (np.sum (b,axis=1))  # The reshape method changes a to be a 1 X 4 array.  print (np.reshape(a,(1,4))) |

Now let’s take a look at the same program using TensorFlow.

Table 2*. A* simple TensorFlow example.

|  |
| --- |
| #!/usr/bin/env python3  import tensorflow as tf  # Running in interactive mode is good for debugging as it  # executes the tensorflow code straight away rather  # than having to create a session variable.  tf.InteractiveSession()  # Create our 'a' array as a 2 X 2 tensor initialized to zeros.  a = tf.zeros((2,2))  # Create our 'b' array as a 2 X 2 tensor initialized to ones.  b = tf.ones((2,2))  # Sum array a and b, reduction\_indices is similar to 'axis'.  # note that we have to invoke the eval() method to  # actually get the reduce\_sum method to run.  print (tf.reduce\_sum(b,reduction\_indices=1).eval())  # Print the shape of tensor 'a'.  print (a.get\_shape())  # Reshape 'a' as a 1 X 4 tensor and print it out.  print (tf.reshape(a,(1,4)).eval()) |

Following is a table of Numpy vs. TensorFlow statements for the previous examples.

Table 3. A summary of differences between Numpy and TensorFlow statements.

|  |  |
| --- | --- |
| **Numpy** | **TensorFlow** |
| a = np.zeros((2,2)); | a = tf.zeros((2,2)) |
| b = np.ones((2,2)) | b = tf.ones((2,2)) |
| np.sum(b, axis=1) | tf.reduce\_sum(a,reduction\_indices=[1]) |
| a.shape | a.get\_shape() |
| np.reshape(a, (1,4)) | tf.reshape(a, (1,4)) |
| np.dot(a,b) | tf.matmul(a, b) |
| b \* 5 + 1 | b \* 5 + 1 |
| a[0,0], a[:,0], a[0,:] | a[0,0], a[:,0], a[0,:] |

Note that TensorFlow requires explicit evaluation of operations. A big difference between the two is that the TensorFlow variables have no actual value until the TensorFlow operation is evaluated. What you will get when printing a tensorflow variable before execution is something like this:

Table 4. Output of a tensor variable before execution of the graph.

|  |
| --- |
| Tensor("zeros:0", shape=(2, 2), dtype=float32) |

This is what will happen in our simple TensorFlow example if you insert a print (a) in the code before you invoke the tf.reduce\_sum evaluation method.

### The TensorFlow session.

The TensorFlow documentation defines a session object as: “A Session object encapsulates the environment in which Tensor objects are evaluated.”

Let’s see how this would work.

Table 5. A simple example of using sessions in TensorFlow.

|  |
| --- |
| #!/usr/bin/env python3  import tensorflow as tf  # Here we declare two constants.  a = tf.constant (5.0)  b = tf.constant (6.0)  # Multiply the two tensors together to output a third tensor 'c'.  c = a \* b  # Nothing will run until we start a TensorFlow session.  # In the previous example, tf.InteractiveSession() was just  # syntactic sugar for the following.  # Note that we can use the Python 'with' to treat tf.Session  # as a context object, containing \_\_entry\_\_ and \_\_exit\_\_ methods.  with tf.Session() as sess:  # Note that c.eval() is, again, syntactic sugar for the sess.run(c)  # statment. sess.run() is an exxample of a Tensorflow fetch  # statement.  print (sess.run(c))  print (c.eval()) |

### The TensorFlow Computation Graph.

TensorFlow programs are structured as follows:

* A construction phase, which creates and assembles a graph.
* An execution phase, used to run operations inside the graph.

When training a TensorFlow model, we use variables to hold and update parameters. Variables are in-memory buffers that contain tensor objects.

Up until now, all the tensors we’ve seen have been constants, not variables. Let’s see how tensor variables work.

Table . The TensorFlow variable.

|  |
| --- |
| #!/usr/bin/env python3  import tensorflow as tf  W1 = tf.ones((2,2))  # Note that W2, unlike W1, is a node in the TensorFlow Graph.  # The name parameter is useful when we want to save or restore  # the values of the variable from a file.  W2 = tf.Variable(tf.zeros((2,2)),name='weights')  with tf.Session() as sess:  print ('\nW1')  print (sess.run(W1))  # Here we initialize all of the graph variaables.  sess.run(tf.global\_variables\_initializer())  # We have to run the graph in order to gain access to W2 to print it.  print ('\nW2')  print (sess.run(W2)) |

As noted previously, TensorFlow variables *must* be initialized via the global\_variables\_initializer() method before they will hold any values.