TensorFlow

Chapter 2 – (Introduction to Deep Learning)

Tensorflow Program - 1

```
import tensorflow as tf
```

- Imports tensorflow library in python

```
x = tf.constant("Hello World")
```

- Adds tensorflow constant to the computation, here it represents a string. In above statement, x represents the computational graph.

```
ses = tf.Session()
```

- Creates tensorflow session. A graph defining the computation is attached to the session.

```
print(ses.run(x))
```

- Session object run evaluates the constant x and access its values.

Tensorflow Program - 2

```
x = tf.constant(2.0)
```

- x is the python variable with a TF constant value of 2.0

```
z = tf.placeholder(tf.float32)
```

- z is the python variable with a TF placeholder as a value. Place holder variables are assigned value at later point in the program. Python dictionary is used to assign values to place holder as shown below.
- `print(ses.run(comp,feed_dict={z:3.0}))`

Tensorflow

- Tensorflow's fundamental data structure are tensors.
  - Tensor's are of type multi-dimension arrays
  - 15 types of different tensors
  - Each tensor has a shape
- Two by three matrix has [2,3] shape.
- Vector of length 4 has [4] shape.
- Shape of [1,4] and [4,1] matrices are different.

Tensorflow & Neural Network Variables

```
bt = tf.random_normal([10], stddev=.1)
b = tf.Variable(bt)
```

- First line create tensor of shape [10] with standard deviation of 0.1.
- Second line takes the tensor and creates graph which generates variable of same size and shape.

```
W = tf.Variable(tf.random_normal([784,10],stddev=.1))
```

- W is a tensor with shape of [784, 10] and standard deviation of 0.1

```
ses.run(tf.initialize_all_variables())
```

- W and b has to be initialized before they can be used. Above line initializes all the variables declared until this point in the program.
- Using variables without initializing will throw an error.
- Initializing variables can be done multiple ways. Tensorflow also supports initializing variables in one program which can be stored in a file and other program can read those values to initialize variables.

Tensorflow Forward Feed Neural Network

```
0 import tensorflow as tf
1 from tensorflow.examples.tutorials.mnist import input_data
2 mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

- These lines import tensorflow and imports MNIST data base for the programs to work with.

```
4 batchSz=100
5 W = tf.Variable(tf.random_normal([784, 10],stddev=.1))
6 b = tf.Variable(tf.random_normal([10],stddev=.1))
```

- Batch size is set to 100 for test data and train data for processing. W and b are tensorflow variables as explained in the previous slide.

```
8 img=tf.placeholder(tf.float32, [batchSz,784])
9 ans = tf.placeholder(tf.float32, [batchSz, 10])
```

- Here img is the the tensorflow place holder variable containing image data for processing in the shape of [batchSz, 784].
- And ans represents the correct answer of shape [batchSz, 10]. Each image in a given batch will be represented as an array with correct position marked as 1 and rest of the positions are kept 0.

Tensorflow Forward Feed Neural Network – cont'd

```
11 prbs = tf.nn.softmax(tf.matmul(img, W) + b)
12 xEnt = tf.reduce_mean(-tf.reduce_sum(ans * tf.log(prbs),
13                                       reduction_indices=[1]))
```

- The first defines forward neural network pass. It specifies that we want to feed batch size of images into linear units. It applies softmax on all results to provide vector of probabilities. The matrix multiplication gives the output of [100, 10] matrix to which biases are added and that result is fed to softmax providing probabilities for each image in our batch.
- Second line multiplies the two tensors of the same shape and reduce_sum sums all rows providing [100,1] array of correct probability. Reduce mean provide average of all columns.

Tensorflow Forward Feed Neural Network – cont'd

```
14 train = tf.train.GradientDescentOptimizer(0.5).minimize(xEnt)
15 numCorrect= tf.equal(tf.argmax(prbs,1), tf.argmax(ans,1))
16 accuracy = tf.reduce_mean(tf.cast(numCorrect, tf.float32))
```

- The first line uses gradient descent and cross entropy loss calculated previously to measure weight change. The learning rate is kept at 0.5. Different loss function and learning rate can be used in this line.
- Next two lines confirms the number of correct assumption and based on the answer calculates the accuracy of the run.

```
18 sess = tf.Session()
19 sess.run(tf.initialize_all_variables())
```

- Above lines creates tensorflow session and initialize all variables declared to this point in the program.

Tensorflow Forward Feed Neural Network – cont'd

```
21 for i in range(1000):
22   imgs, ans = mnist.train.next_batch(batchSz)
23   sess.run(train, feed_dict={img: imgs, ans: ans})
```

- These lines train the model on the MNIST data loaded previously on the program.
- Here forward pass has not been mentioned directly but gradient descent call figures this out. It looks for xEnt which requires probs which in turn computes forward pass computation.

```
25 sumAcc=0
26 for i in range(1000):
27    imgs, ans= mnist.test.next_batch(batchsz)
28    sumAcc+=sess.run(accuracy, feed_dict={img: imgs, ans: ans})
29 print "Test Accuracy: %r" % (sumAcc/1000)
```

- Here the MNIST test data is used to test model accuracy.

Multi-layered Neural Network

- Adding linear layer to the previous NN implementation does not help.
  - y = XW, here W is the weight and has shape of [784, 10].
  - We transform 784 pixel values here in to 10 logits.
  - If another linear layer U is added of shape [784, 784] which feeds into layer V of shape [784, 10] then,

$$\begin{aligned} \mathbf{y} &= (\mathbf{x}\mathbf{U})\mathbf{V} \\ &= \mathbf{x}(\mathbf{U}\mathbf{V}) \end{aligned}$$

  - Above equation proves that adding another linear layer does not help as capabilities provided by two layered NN can be achieved by single layer as well.

Multi-layered Neural Network

- Solution to previous problem is to add non-linear computation between layers.
- Rectified Linear Unit (relu) is the most common used option and it is defined as,

$$\rho(x) = \max(x, 0)$$

- Non-linear layers put in between are known as activation function.
- Another popular activation function is sigmoid function and it is defined as,

$$S(x) = \frac{e^{-x}}{1 + e^{-x}}$$

Multi-layered Neural Network in Tensorflow

```
1 U = tf.variable(tf.random_normal([784,784], std_dev=.1))
2 bU = tf.variable(tf.random_normal([784], std_dev=.1)
3 V = tf.variable(tf.random_normal([784,10], std\_dev=.1))
4 bV = tf.variable(tf.random_normal([10], std_dev=.1)
5 l1Output = matmul(img,U)+bu
6 l1Output=tf.relu(l1Output)
7 prbs=tf.softmax(matmul((l1Ouput,V)+bv)
```

- In above program, line 1 to 4 declares variables U and V for two layers. U has shape of [784, 784] and V has shape of [784, 10].
- Line 5 and 6 provides the output of layer1. Here the use of Rectified Linear Unit is extra compared to single layer neural network program.
- With this approach, the accuracy of the program increased to 94% from 92% in single layer neural network program.

Checkpointing

- In tensorflow, it is very useful to save tensors after computation. This saved data can be used at later point in time or in different program. This process is called checkpointing.
  - saveOb = tf.train.Saver() -> Creates Tensorflow saver object
  - saveOb.save(sess, “mylatest.ckpt”) -> Saves current tensors in mylatest.ckpt file. Creates 3 files with each save.
    - mylatest.ckpt.data-00000-of-00001
    - mylatest.ckpt.index
    - mylatest.ckpt.meta
  - saveOb.restore(sess, “mylatest.ckpt”) -> Restores the tensors from mylatest.ckpt file.

Tensordot

- This is a generalization of matrix multiplication.
- Matrix multiplication of A and B requires that last dimension of A and second last dimension of B have to be same. Also all n-2 dimension of both matrices are identical.
  - i.e. A has dimension of [2,3,4] and B is of dimension [2,4,6] then resulting multiplication will be of dimension [2,4,6].
  - Here the dot product of first row of matrix A and first column of matrix B provides top left corner element of resulting matrix.
  - The same can be achieved through tensordot using,
    - tf.tensordot(A, B, [[1], [0]])
    - The third argument is the list of two elements. The first element is the dimension of first argument and the second element is the dimension of second argument.

Simplifying Tensorflow graph creation

- Tensorflow makes neural network program small and compact compared to plain python programming.
- Multi-layered program previously worked with just 7 lines of tensorflow program. Tensorflow provides capabilities to add layers with common requirements very easy.
  - tf.contrib.layers.fully.connected
  - A layer is fully connected if all its units are fully connected to the subsequent layer's units.
  - To define such connected layers, we typically do
    - Create weights W
    - Create biases b
    - Do matrix multiplications and add biases
    - Apply activation function.

Simplifying Tensorflow graph creation – cont'd

- If tensorflow.contrib.layers are imported then all 4 previous steps can be done with
  - layerOut = layers.fully.connected(layerIn, outSz, activeFn)
  - This call initialize matrix using Xavier Initialization. It returns layerIn times the matrix and adds biases. Activation function activeFn is applied to this output. Default activation function is set to Relu.
- Previous 7 line multi-layered program can be rewritten in just 2 lines.
  - L1Output = layers.fully.connected(img, 756)
    prbs = layers.fully.connected(L1Output, 10, tf.nn.softmax)