# CMPE 142 HW4

Jordan Liss

March 17, 2017

## 1 Question 1: Getting Started with TensorFlow

(a) Tensorflow is machine learning package that used to make utilizing the tools of machine learning cleaner (Using less code) and quicker (exploiting GPUs). For example, one of the high-level API functions is tf.contrib.learn and its helps in managing datasets, estimators, training and inferences. A *tensor* consists of a set of primitive values shaped into an array of any number of dimensions. One characteristic is that the $rank(tensor) = dim(tensor)$. Tensors is a unique molding of a multidimensional matrix array. You create nodes as the usual method for inputting things into a function and these function must be masked in a system called a session. For ex:

```
node1 = tf.constant(3.0, tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
sess = tf.Session()
print(sess.run([node1, node2]))
```

The output was $[3.0, 4.0]$

To make more complicated computations Tensor nodes can be combined with operations (One can think of operations at nodes). Tensorflow uses *placeholders* to be handlers of parameterized external inputs, that can be a variable that can be used for later. Used to setup a "$function$"$like$ system. The "$function$"$like$ system, placeholder being made is $adder_node$.

```
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)
adder_node = a + b  # + provides a shortcut for tf.add(a, b)
```

A multi-level function system with placeholder looks like this:

```
add_and_triple = adder_node * 3.
print(sess.run(add_and_triple, {a: 3, b:4.5}))
```

The real meat and potatoes of machine learning is to train on data and then cross reference the model to a test set of data. In tensorflow the function (Or sequences of functions) for training is

```
optimizer = tf.train.GradientDescentOptimizer(0.01) # 0.01 is stepsize
train = optimizer.minimize(loss)
sess.run(init) # reset values to incorrect defaults.
for i in range(1000):
sess.run(train, {x:[1,2,3,4], y:[0,-1,-2,-3]})

print(sess.run([W, b]))
```

The function *tf.contrib.learn* as mentioned earlier can easily greatly simplify building a machine learning algorithm.

(b)

# 2 Question 2: Linear Regressor

Below is the code I used for the LinearRegressor I called the file: a4q2.py

```
import tensorflow as tf
import numpy as np

# Inputing File
a4q2_test=np.genfromtxt("a4q2_test.txt")
a4q2_train=np.genfromtxt("a4q2_train.txt")
xtr1=a4q2_train[:,0]
xtr2=a4q2_train[:,1]
xte1=a4q2_test[:,0]
xte2=a4q2_test[:,1]
ytr=a4q2_train[:,2]
yte=a4q2_test[:,2]


# Feature determination
featurestr = [tf.contrib.layers.real_valued_column("xtr1", dimension = 1),
tf.contrib.layers.real_valued_column("xtr2", dimension = 1)] # Need to add a feature param for each column
#These features are shared between both the training and test data
# featureste = [tf.contrib.layers.real_valued_column("xte1", dimension = 1),
tf.contrib.layers.real_valued_column("xte2", dimension = 1)]

# Determine the regressor (linear)
estimator = tf.contrib.learn.LinearRegressor(feature_columns=featurestr)

#Turn all inputs into numpy inputs
input_fe_tr = tf.contrib.learn.io.numpy_input_fn({"xtr1":xtr1, "xtr2":xtr2}, ytr,
batch_size=100,num_epochs=10000) #num_epochs = number of iterations batch_size = size of cross correlation datasets

# Fit model
estimator.fit(input_fn=input_fe_tr, steps=8801) #steps = number of steps
#
# Evaluate accuracy.
print(estimator.evaluate(input_fn=input_fe_tr))

# Test Evaluate accuracy.
input_fe_te = tf.contrib.learn.io.numpy_input_fn({"xtr1":xte1, "xtr2":xte2}, yte,
batch_size=100,num_epochs=10000)
print(estimator.evaluate(input_fn=input_fe_te))
```

The outputs I get are below:
When Trained: $'loss' : 0.002381359,' global'_{step} : 8801$
When Tested: $'loss' : 0.0025680144,' global'_{step} : 8801$

# 3 Question 3: Multinomial Logistics Regression

(a) The MNIST regression tutorial obtains data from Yann LeCun's website and is split into three parts: 55,000 data points of training data (mnist.train), 10,000 points of test data (mnist.test), and 5,000 points of validation data (mnist.validation). The data points have two parts an image of a handwritten digit and a corresponding label. Each image gets flatten. Softmax regression takes two steps: first we add up the evidence of our inputs being in certain classes, and then we convert that evidence into probabilities. Once the model is prepared we leverage the power of numpy to run the numerical computations.

(b) The original accuracy was 0.89. To improve the accuracy I changed the optimizer from *tf.train.GradientDescentOptimi* to *tf.train.ProximalGradientDescentOptimizer*, increased the iterations from 100 to 10000 and increased the batchsize to 1000. The updated mnist softmax python code is below:

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import sys

from tensorflow.examples.tutorials.mnist import input_data

import tensorflow as tf

FLAGS = None


def main(_):
# Import data
```

```
mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b

# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])

cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
train_step = tf.train.ProximalGradientDescentOptimizer(0.75).minimize(cross_entropy)
sess = tf.InteractiveSession()
tf.global_variables_initializer().run()
# Train
for _ in range(10000):
# Changed the iterations to run a 10000 instead of 100 times at 3/16/17 7:30 pm
# and obtained a accuracy of 0.9253
batch_xs, batch_ys = mnist.train.next_batch(1000)
sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

# Test trained model
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(sess.run(accuracy, feed_dict={x: mnist.test.images,
y_: mnist.test.labels}))

if __name__ == '__main__':
parser = argparse.ArgumentParser()
parser.add_argument('--data_dir', type=str, default='/tmp/tensorflow/mnist/input_data',
help='Directory for storing input data')
FLAGS, unparsed = parser.parse_known_args()
tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

The final accuracy I obtained is 0.927 or 92.7%

# 4 Question 4: Feed-Forward Neural Network

(a) The TensorFlow Mechanics 101 tutorial shows us how to train and evaluate a simple feed-forward network problem. The first step in all tensor flow problems is to build the graph by creating placeholders for the data, the graph is built from the mnist.py file according to a 3-stage pattern: inference(), loss(), and training(). The inference() function builds the graph as far as needed to return the tensor that would contain the output predictions. The loss() function further builds the graph by adding the required loss ops. The training() function adds the operations needed to minimize the loss via Gradient Descent. The next step is to train the iterative model. Once all of the build preparation has been completed and all of the necessary ops generated, a tf.Session is created for running the graph. Check the results of all of the build preparation has been completed and all of the necessary ops generated, a tf.Session is created for running the graph. Use the various tensorflow functions to record summaries of activities by using the TensorBoard. Evaluating the performance and output of the values use the $do_eval()$ function. Don't forget to check the accuracy of the outputs.

(b) Original accuracy is

```
Training Data Eval:
Num examples: 55000  Num correct: 49357  Precision @ 1: 0.8974
Validation Data Eval:
Num examples: 5000  Num correct: 4518  Precision @ 1: 0.9036
Test Data Eval:
Num examples: 10000  Num correct: 9023  Precision @ 1: 0.9023
```

The final accuracy numbers are as follows:

```
Training Data Eval:
Num examples: 55000  Num correct: 49357  Precision @ 1: 1.0000
Validation Data Eval:
Num examples: 5000  Num correct: 4518  Precision @ 1: 0.9892
Test Data Eval:
Num examples: 10000  Num correct: 9023  Precision @ 1: 0.9771
```

In order to obtain these numbers I modified the following in the fully connected feed.py file:

(a) Changed default batchsize to 2000 from 100 (line 268)

(b) Added two additional hidden layers (line 253 to 263)

(c) Changed hidden layer params all to 128 (line 253 to 263)

(d) Increased steps to 20000 from 2000 (line 238)