CSCE 509 – Spring 2019
Assignment 3 // updated 01May19
DUE: May 11, 2019 at 5 p.m.

- Two data sets available on Moodle
  - {concaveData.npy, concaveTarget.npy}
  - {testData.npy, testTarget.npy}
- Write TensorFlow code to perform DNN classification with three (3) classes
- Use concave*.npy for training
- Use test*.npy for test
- Data is the data matrix; Target is the labeled targets from {0, 1, 2}
- Do each of the following steps. For each step: Note the accuracy of the classification using the test data set. Discuss the results.

1. Write TensorFlow code to perform DNN classification using default settings. Define your own architecture with two hidden layers. Calculate the number of parameters in your network. Do not let the number of parameters exceed the number of input samples in concave*.npy

2. Use one or two additional layers compared to (1) but be sure that the number of parameters do not exceed the number of input samples. Which has better accuracy performance? Or are they about the same?

3. Write Python code to read in the data sets. Add a large constant (such as "509" or "5090") to each input feature. Write the data sets as files, to be read in as input sets. Repeat the classification using the new input files with the architecture that has better performance in (1) or (2). What is the accuracy performance for the same number of epochs? If the accuracy performance is about the same, does it converge faster or slower or about the same?

4. Use the given data sets as used in (1) and (2). Use either of the two architectures. Change the tf.layers.dense() function initlialization to He initialization by using the variance_scaling_initializer() function:
```
he_init = tf.contrib.layers.variance_scaling_initializer(factor=2.0)
hidden1 = tf.layers.dense(X, n_hidden1, activation=tf.nn.relu,
      kernel_initializer=he_init, name="hidden1")
# do the same for other hidden layers
```
   What is the accuracy performance? Compare to either (1) or (2).

5. Take the architecture from either (1) or (2). Replace the relu activation function by the exponential linear unit (ELU). In the `tf.layers.dense` function, use `activation=tf.nn.elu`
   What is the accuracy performance? Compare to either (1) or (2) and to (4).

6. Perform batch normalization on either (1) or (2) as follows. We want to zero-center and normalize the inputs to the activation function of each layer by learning the mean and scales of the inputs for each layer. Modify the Python code as follows:

```
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
training = tf.placeholder_with_default(False, shape=(), name="training")
```

Then in defining the hidden layers:

```
hidden1 = tf.layers.dense(X, n_hidden1, name="hidden1")
batchnorm1 = tf.layers.batch_normalization(hidden1, training=training,
momentum=0.9)
bn1_act = tf.nn.elu(batchnorm1)
hidden2 = tf.layers.dense(bn1_act, n_hidden2, name="hidden2")
batchnorm2 = tf.layers.batch_normalization(hidden2, training=training,
momentum=0.9)
bn2_act = tf.nn.elu(batchnorm2)
# other hidden layers as needed
logits_bf_bn = tf.layers.dense(bn2_act, n_outputs, name="outputs")
logits = tf.layers.batch_normalization(logits_bf_bn, training=training,
momentum=0.9)
```

At the end of the construction phase, add this line:

```
extra_update_steps = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
```

In the execution phase; add the extra update steps and set training to True during the session.run:

```
session.run([training_step, extra_update_steps], feed_dict={training: True,
X:X_batch, y:y_batch})
```

Does the performance change?