CS 509: Pattern Recognition

SM ZOBAED-C00300901

Assignment-3

May 10, 2019

1. Using default settings, I implement tensorflow code to perform DNN classification considering 2 hidden layers. The number of neurons in hidden layer 1 and 2 are 250 and 125 respectively. I train the model using the provided concaveData.npy and its label. Each epoch, I check whether it requires to update the maximum accuracy. 200 epochs are used. After running the code, using the provided test data, I obtain the maximum accuracy:**.978** that comes from epoch number **184**.

Listing 1: Code Snippet: question 1 (hidden layer creation and DNN classification)

```
# create two hidden layers and output layer
with tf.name_scope("dnn"):
    hidden1 = tf.layers.dense(X, n_hidden1, name="hidden1",
activation=tf.nn.relu)
    hidden2 = tf.layers.dense(hidden1, n_hidden2, name="hidden2",
activation=tf.nn.relu)
    logits = tf.layers.dense(hidden2, n_outputs, name="outputs")

# define cost function to train network
with tf.name_scope("loss"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=t,
logits=logits)
    loss = tf.reduce_mean(xentropy, name="loss")

# define how to train
learning_rate = 0.01
with tf.name_scope("train"):
 optimizer = tf.train.GradientDescentOptimizer(learning_rate)
 training_step = optimizer.minimize(loss)

# how to evaluate model
with tf.name_scope("eval"):
 correct = tf.nn.in_top_k(logits, t, 1)
 accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
# initialize
init = tf.global_variables_initializer()
maxx_acc=0.0
n_epochs=200
with tf.Session() as session:
    init.run()
    for epoch in range(n_epochs):
        mini_batches = create_mini_batches(xdata,
                    tdata.reshape(tdata.shape[0],1), batch_size)
        for mini_batch in mini_batches:
            # do training step first
            X_batch, t_batch = mini_batch
            session.run(training_step, feed_dict={X:X_batch, t:t_batch})
            acc_train = accuracy.eval(feed_dict={X:X_batch, t:t_batch})
            acc_val = accuracy.eval(feed_dict={X:vxdata, t:vtdata})

            if(maxx_acc<acc_val):
                maxx_acc=acc_val
                epoch_for_maxx=epoch
                max_acc_train=acc_train



print("*****DNN Parameters*****")
print("Number of Hidden Layers: ", 2)

print("Neurons in Hidden layer1: ",n_hidden1 )
print("Neurons in Hidden layer2: ",n_hidden2 )

print("No. of input samples: ", p)
print('DNN Model With default settings, Max accuracy at epoch no. : ',
              epoch_for_maxx, '   Maximum Accuracy of the model : ', maxx_acc)
```

2. I create two additional hidden layers with the aforementioned model. The number of neurons in my layer 3 and 4 are 100 and 50 respectively. I use 200 epoch. Each epoch, I check whether it requires to update the maximum accuracy. After running the code, using the provided test data, I obtain the maximum accuracy:**.987** that comes from epoch number **143**.

Listing 2: Code Snippet: question 1 (4 hidden layer creation and DNN classification)

```
# create two hidden layers and output layer
with tf.name_scope("dnn"):
    hidden1 = tf.layers.dense(X, n_hidden1, name="hidden1",
activation=tf.nn.relu)
    hidden2 = tf.layers.dense(hidden1, n_hidden2, name="hidden2",
activation=tf.nn.relu)
    hidden3 = tf.layers.dense(hidden2, n_hidden3, name="hidden3",
activation=tf.nn.relu)
    hidden4 = tf.layers.dense(hidden3, n_hidden4, name="hidden4",
activation=tf.nn.relu)
    logits = tf.layers.dense(hidden4, n_outputs, name="outputs")
```

3. I load the training and test dataset (provided concaveData.npy and testData.npy files) in xdata and vxdata. Later, I add constant 5090 to each input feature and create modified train and test dataset. I save the data sets into two npy files: `modified_train` and `modified_test.npy` respectively and utilize to train and test the model. As we observe the higher accuracy comes from 2nd model, we use 2nd model to solve question 3. The code snippet is provided below.

   Listing 3: Code Snippet: question 3

   ```
   xdata=np.load ("concaveData.npy")
   tdata= np.load("concaveTarget.npy")
   N, p = xdata.shape
   modified_x=xdata+5090
   np.save("modified_train",modified_x)
   modified_x=np.load("modified_train.npy")


   vxdata = np.load("testData.npy")
   v_shape_x, v_shape_y=vxdata.shape
   vtdata = np.load("testTarget.npy")
   modified_test=vxdata+5090
   np.save("modified_test",modified_test)
   modified_test=np.load("modified_test.npy")

   # create four hidden layers and output layer
   with tf.name_scope("dnn"):
       hidden1 = tf.layers.dense(X, n_hidden1, name="hidden1",
   activation=tf.nn.relu)
       hidden2 = tf.layers.dense(hidden1, n_hidden2, name="hidden2",
   activation=tf.nn.relu)
       hidden3 = tf.layers.dense(hidden2, n_hidden3, name="hidden3",
   activation=tf.nn.relu)
       hidden4 = tf.layers.dense(hidden3, n_hidden4, name="hidden4",
   activation=tf.nn.relu)
       logits = tf.layers.dense(hidden4, n_outputs, name="outputs")
   ```

   After running the code, using the modified train and test data, I obtain the maximum accuracy:**.4**. The accuracy is unchanged in every iteration. Adding such large constant create issue in the model building. We need to perform further process (e.g., scaling) before building model.

4. I change the initialization function of the hidden layers from tf.layers.dense() to "he" initialization. The code snippet is provided below.

   Listing 4: Code Snippet: question 4

   ```
   # create four hidden layers and output layer
   with tf.name_scope("dnn"):
       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")
       hidden2 = tf.layers.dense(hidden1, n_hidden2, name="hidden2",
                                 activation=tf.nn.relu, kernel_initializer=he_init)
       hidden3 = tf.layers.dense(hidden2, n_hidden3, name="hidden3",
                                 activation=tf.nn.relu, kernel_initializer=he_init)
       hidden4 = tf.layers.dense(hidden3, n_hidden4, name="hidden4",
                                 activation=tf.nn.relu, kernel_initializer=he_init)
   ```

```
    logits = tf.layers.dense(hidden4, n_outputs, name="outputs", kernel_initializer=he_in
```

We build the model considering 4 hidden layers that is based on question 2. Hence, the accuracy comparison between $Heinitialization$ version and $tf.layers.dense()$ function:
He initialization: .989
tf.layers.dense(): .987

5. I modify the first model. In the first model, I had two hidden layers. I change the changed the activation function and use *tf.nn.elu* function. With the first model, I compare 4th model where I use 4 hidden layers with *He initialization*. I use 200 epoch.
   Accuracy obtained from modified 1st model: .974 Accuracy obtained from 4th model: .988

Listing 5: Code Snippet: question 5

```
#Modified verison of Question 1
with tf.name_scope("dnn"):
    hidden1 = tf.layers.dense(X, n_hidden1, name="hidden1",
activation=tf.nn.elu)
    hidden2 = tf.layers.dense(hidden1, n_hidden2, name="hidden2",
activation=tf.nn.elu)
    logits = tf.layers.dense(hidden2, n_outputs, name="outputs")



#Question 4
with tf.name_scope("dnn"):
    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")
    hidden2 = tf.layers.dense(hidden1, n_hidden2, name="hidden2",
                              activation=tf.nn.relu, kernel_initializer=he_init)
    hidden3 = tf.layers.dense(hidden2, n_hidden3, name="hidden3",
                              activation=tf.nn.relu, kernel_initializer=he_init)
    hidden4 = tf.layers.dense(hidden3, n_hidden4, name="hidden4",
                              activation=tf.nn.relu, kernel_initializer=he_init)
    logits = tf.layers.dense(hidden4, n_outputs, name="outputs", kernel_initializer=he_in
```

6. I perform normalization on model 1. I 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. Therefore, we modify the code of answer 1 (2 hidden layers) accordingly.

Listing 6: Code Snippet: question 6

```
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
training = tf.placeholder_with_default(False, shape=(), name="training")
t = tf.placeholder(tf.int64, shape=(None), name="t")


# create two hidden layers and output layer
with tf.name_scope("dnn"):
    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)


# define cost function to train network
with tf.name_scope("loss"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=t,
logits=logits)
    loss = tf.reduce_mean(xentropy, name="loss")

# define how to train
learning_rate = 0.01
with tf.name_scope("train"):
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    training_step = optimizer.minimize(loss)
    extra_update_steps = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
```

After the modification, we obtain accuracy=**.99** and find in epoch 191 which is greater than the prior that provides accuracy=**.978**