Iowa State University

Department of Electrical and Computer Engineering

Deep Machine Learning: Theory and Practice

EE 526X

Homework 3

Author:
Vishal Deep

Instructor:
Dr. Zhengdao Wang

October 17, 2019

1 Problem 1

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 13 15:20:36 2019

@author: vishal
"""
import tensorflow as tf

# Import Data
from tensorflow.examples.tutorials.mnist import input_data
mnist =input_data.read_data_sets("MNIST_data/", one_hot=True)

# Set up learning rate, epochs, and batch size
learningRate =0.01
epochsArr =[10, 50, 100]
batch_size =500

total_batch =int(len(mnist.train.labels) /batch_size)

def layer(nInput, nOutput, X, activation):
    W =tf.Variable(tf.random_normal([nInput, nOutput], stddev=0.01))
    b =tf.Variable(tf.random_normal([nOutput]))
    z =tf.add(tf.matmul(X, W), b)
    if activation =='ReLu':
        nOut =tf.nn.relu(z)
    elif activation =='Linear':
        nOut =z
    else:
        raise ValueError('activation values can only be ReLu or Linear')
    return nOut

def calcAccuracy(logit, y):
    # find max value from y and logit and compare it
    accuratePredictions =tf.equal(tf.argmax(y, 1), tf.argmax(logit, 1))
    # Cast to a float32 and calculate mean
    accuracy =tf.reduce_mean(tf.cast(accuratePredictions, tf.float32))
    return accuracy

def logStat(nEpochs, epoch, avg_cost, accuracy):
    if nEpochs <=10:
        if (epoch%2 ==0):
            print("Epoch: {}".format(epoch+1), "Cost: {:f}".format(avg_cost), "Training Acc: {:f}".
                                                                    format(accuracy))
    elif nEpochs >10 and epochs <=50:
        if (epoch%10 ==0):
            print("Epoch: {}".format(epoch+1), "Cost: {:f}".format(avg_cost), "Training Acc: {:f}".
                                                                    format(accuracy))
    elif nEpochs >50 and nEpochs <=100:
        if (epoch%20 ==0):
            print("Epoch: {}".format(epoch+1), "Cost: {:f}".format(avg_cost), "Training Acc: {:f}".
                                                                    format(accuracy))
    else:
        if (epoch%100 ==0):
            print("Epoch: {}".format(epoch+1), "Cost: {:f}".format(avg_cost), "Training Acc: {:f}".
                                                                format(accuracy))

# Placeholders for X and y
x =tf.placeholder(tf.float32, [None, 784])
y =tf.placeholder(tf.float32, [None, 10])

for epochs in epochsArr:
    print("Number of epochs: {}".format(epochs))
    # First Neural Network
    # Layer 1
    l1_out_nn1 =layer(784, 10, x, 'Linear')
    # Softmax + cross Entropy
    ce_nn1 =tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=l1_out_nn1, labels=y))
    # Optimizer Gradient Descent
    optimizer_nn1 =tf.train.GradientDescentOptimizer(learning_rate=learningRate).minimize(ce_nn1)
    # Initialize variables
    init =tf.global_variables_initializer()
    # calculate accuracy
    accuracy_nn1 =calcAccuracy(l1_out_nn1, y)

    print("========== NN1 session starts ==========")
    # start the session
    with tf.Session() as sess:
        # initialise the variables
        sess.run(init)

        for epoch in range(epochs):
            avg_cost =0
            for i in range(total_batch):
                XBatch, yBatch =mnist.train.next_batch(batch_size=batch_size)
                _, c =sess.run([optimizer_nn1, ce_nn1], feed_dict={x: XBatch, y: yBatch})
                avg_cost +=c /total_batch
            trainAcc_nn1 =sess.run(accuracy_nn1, feed_dict={x: XBatch, y: yBatch})
            logStat(epochs, epoch, avg_cost, trainAcc_nn1)
        testAcc_nn1 =sess.run(accuracy_nn1, feed_dict={x: mnist.test.images, y: mnist.test.labels})
        print("Test Accuracy: {:f}".format(testAcc_nn1))

    print("========== NN1 session Completed ==========")

    # Second Neural Network
    l1_out_nn2 =layer(784, 50, x, 'ReLu')
    l2_out_nn2 =layer(50, 50, l1_out_nn2, 'ReLu')
    l3_out_nn2 =layer(50, 10, l2_out_nn2, 'Linear')

    # Softmax + cross Entropy
    ce_nn2 =tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=l3_out_nn2, labels=y))
    # Optimizer Gradient Descent
    optimizer_nn2 =tf.train.GradientDescentOptimizer(learning_rate=learningRate).minimize(ce_nn2)
    # finally setup the initialisation operator
    init =tf.global_variables_initializer()
    # calculate accuracy
    accuracy_nn2 =calcAccuracy(l3_out_nn2, y)

    print("========== NN2 session starts ==========")
    # start the session
    with tf.Session() as sess:
        # initialise the variables
        sess.run(init)

        for epoch in range(epochs):
            avg_cost =0
            for i in range(total_batch):
                XBatch, yBatch =mnist.train.next_batch(batch_size=batch_size)
                _, c =sess.run([optimizer_nn2, ce_nn2], feed_dict={x: XBatch, y: yBatch})
                avg_cost +=c /total_batch
            trainAcc_nn2 =sess.run(accuracy_nn2, feed_dict={x: XBatch, y: yBatch})
# print("Epoch: {}".format(epoch+1), "Cost: {:f}".format(avg_cost), "Training Acc: {:f}".format(
                                                    trainAcc_nn2))
            logStat(epochs, epoch, avg_cost, trainAcc_nn2)

        testAcc_nn2 =sess.run(accuracy_nn2, feed_dict={x: mnist.test.images, y: mnist.test.labels})
        print("Test Accuracy: {:f}".format(testAcc_nn2))

    print("========== NN2 session Completed ==========")
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Number of epochs: 10
========== NN1 session starts ==========
Epoch: 1 Cost: 1.968383 Training Acc: 0.654000
Epoch: 3 Cost: 1.081156 Training Acc: 0.776000
Epoch: 5 Cost: 0.834734 Training Acc: 0.838000
Epoch: 7 Cost: 0.717895 Training Acc: 0.850000
Epoch: 9 Cost: 0.648424 Training Acc: 0.844000
Test Accuracy: 0.860500
========== NN1 session Completed ==========
========== NN2 session starts ==========
Epoch: 1 Cost: 2.518511 Training Acc: 0.122000
Epoch: 3 Cost: 2.284648 Training Acc: 0.232000
Epoch: 5 Cost: 2.258813 Training Acc: 0.212000
Epoch: 7 Cost: 2.217494 Training Acc: 0.176000
Epoch: 9 Cost: 2.161243 Training Acc: 0.178000
Test Accuracy: 0.181700
========== NN2 session Completed ==========
Number of epochs: 50
========== NN1 session starts ==========
Epoch: 1 Cost: 1.839826 Training Acc: 0.780000
Epoch: 11 Cost: 0.580703 Training Acc: 0.834000
Epoch: 21 Cost: 0.475618 Training Acc: 0.914000
Epoch: 31 Cost: 0.431121 Training Acc: 0.862000
Epoch: 41 Cost: 0.405112 Training Acc: 0.906000
Test Accuracy: 0.901500
========== NN1 session Completed ==========
========== NN2 session starts ==========
Epoch: 1 Cost: 2.360294 Training Acc: 0.112000
Epoch: 11 Cost: 2.276963 Training Acc: 0.230000
Epoch: 21 Cost: 2.023256 Training Acc: 0.252000
Epoch: 31 Cost: 1.616091 Training Acc: 0.472000
Epoch: 41 Cost: 1.133604 Training Acc: 0.652000
Test Accuracy: 0.772100
========== NN2 session Completed ==========
Number of epochs: 100
========== NN1 session starts ==========
Epoch: 1 Cost: 1.949118 Training Acc: 0.718000
Epoch: 21 Cost: 0.482889 Training Acc: 0.890000
Epoch: 41 Cost: 0.409774 Training Acc: 0.904000
Epoch: 61 Cost: 0.378109 Training Acc: 0.894000
Epoch: 81 Cost: 0.359247 Training Acc: 0.904000
Test Accuracy: 0.911500
========== NN1 session Completed ==========
========== NN2 session starts ==========
Epoch: 1 Cost: 2.405205 Training Acc: 0.108000
Epoch: 21 Cost: 1.865193 Training Acc: 0.270000
Epoch: 41 Cost: 1.101819 Training Acc: 0.584000
Epoch: 61 Cost: 0.691297 Training Acc: 0.798000
Epoch: 81 Cost: 0.430330 Training Acc: 0.874000
Test Accuracy: 0.891100
========== NN2 session Completed ==========
```

2 Problem 2

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 14 20:04:08 2019

@author: vishal
"""

import tensorflow as tf
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers

# Get data from tensor flow
mnist =tf.keras.datasets.mnist
# Split data between train and test data
(x, y),(x_test, y_test) =mnist.load_data()
# flatten the image data for all 60000 images
XTrain =x.reshape(-1, 784)
XTest =x_test.reshape(-1, 784)
# One hot encoding
yTrain =to_categorical(y)
yTest =to_categorical(y_test)

# Set up learning rate, epochs, and batch size
learningRate =0.01
batchSize =500
epochsArr =[10, 50, 100]

for epochs in epochsArr:
    print("========== NN1 session starts ===========")
    model_nn1 =Sequential([
      Dense(10, activation='softmax', input_shape=(784,))
    ])

    # Stochastic Gradient Descent optimizer
    sgd =optimizers.SGD(lr=learningRate, clipnorm=1.)
    # use crossentropy
    model_nn1.compile(optimizer=sgd, loss='categorical_crossentropy', metrics=['accuracy'])
    # Train the NN
    model_nn1.fit(XTrain, yTrain, epochs=epochs, batch_size=batchSize, verbose=2)
    # Calculate accuracy on test data
    [loss_nn1, accuracy_nn1] =model_nn1.evaluate(x=XTest, y=yTest, batch_size=batchSize, verbose=2)
    print("Test Accuracy: {:f}".format(accuracy_nn1))
    print("========== NN1 session completed ===========")

    print("========== NN2 session starts ===========")
    model_nn2 =Sequential([
      Dense(50, activation='relu', input_shape=(784,)),
      Dense(50, activation='relu'),
      Dense(10, activation='softmax'),
    ])

    # Stochastic Gradient Descent optimizer
    sgd =optimizers.SGD(lr=learningRate, clipnorm=1.)
    # use crossentropy
    model_nn2.compile(optimizer=sgd, loss='categorical_crossentropy', metrics=['accuracy'])
    # Train the NN
    model_nn2.fit(XTrain, yTrain, epochs=epochs, batch_size=batchSize, verbose=2)
    # Calculate accuracy on test data
    [loss_nn2, accuracy_nn2] =model_nn2.evaluate(x=XTest, y=yTest, batch_size=batchSize, verbose=2)
    print("Test Accuracy: {:f}".format(accuracy_nn2))
    print("========== NN2 session completed ===========")
```

```
Number of epochs: 10
========== NN1 session starts ===========
Epoch 1/10
 - 6s - loss: 58.6495 - accuracy: 0.4272
Epoch 2/10
 - 0s - loss: 12.9306 - accuracy: 0.7637
Epoch 3/10
 - 0s - loss: 9.0688 - accuracy: 0.8210
Epoch 4/10
 - 0s - loss: 7.5447 - accuracy: 0.8411
Epoch 5/10
 - 0s - loss: 6.6608 - accuracy: 0.8523
Epoch 6/10
 - 0s - loss: 6.0341 - accuracy: 0.8579
Epoch 7/10
 - 0s - loss: 5.5584 - accuracy: 0.8620
Epoch 8/10
 - 0s - loss: 5.1580 - accuracy: 0.8662
Epoch 9/10
 - 0s - loss: 4.8406 - accuracy: 0.8683
Epoch 10/10
 - 0s - loss: 4.5674 - accuracy: 0.8707
Test Accuracy: 0.876000
========== NN1 session completed ===========
========== NN2 session starts ===========
Epoch 1/10
 - 1s - loss: 29.6423 - accuracy: 0.3535
Epoch 2/10
 - 1s - loss: 5.4192 - accuracy: 0.6727
Epoch 3/10
 - 1s - loss: 3.0112 - accuracy: 0.7594
Epoch 4/10
 - 1s - loss: 2.1132 - accuracy: 0.7977
Epoch 5/10
 - 1s - loss: 1.6312 - accuracy: 0.8198
Epoch 6/10
 - 1s - loss: 1.3211 - accuracy: 0.8345
Epoch 7/10
 - 1s - loss: 1.1066 - accuracy: 0.8461
Epoch 8/10
 - 1s - loss: 0.9479 - accuracy: 0.8539
Epoch 9/10
 - 1s - loss: 0.8283 - accuracy: 0.8626
Epoch 10/10
 - 1s - loss: 0.7297 - accuracy: 0.8703
Test Accuracy: 0.866500
========== NN2 session completed ===========
Number of epochs: 50
========== NN1 session starts ===========
Epoch 1/50
 - 0s - loss: 63.6025 - accuracy: 0.4189
Epoch 2/50
 - 0s - loss: 13.0491 - accuracy: 0.7642
Epoch 3/50
 - 0s - loss: 8.9469 - accuracy: 0.8198
Epoch 4/50
 - 0s - loss: 7.3473 - accuracy: 0.8405

 .
 .
 .
Epoch 46/50
 - 0s - loss: 1.7235 - accuracy: 0.8938
Epoch 47/50
```

```
 − 0s − loss: 1.7137 − accuracy: 0.8931
Epoch 48/50
 − 0s − loss: 1.6825 − accuracy: 0.8936
Epoch 49/50
 − 0s − loss: 1.6626 − accuracy: 0.8916
Epoch 50/50
 − 0s − loss: 1.6322 − accuracy: 0.8929
Test Accuracy: 0.890400
========== NN1 session completed ===========
========== NN2 session starts ===========
Epoch 1/50
 − 1s − loss: 27.6743 − accuracy: 0.3245
Epoch 2/50
 − 1s − loss: 4.2701 − accuracy: 0.5955
Epoch 3/50
 − 1s − loss: 2.2076 − accuracy: 0.6626
Epoch 4/50
 − 1s − loss: 1.4462 − accuracy: 0.7050
.
.
.
Epoch 46/50
 − 1s − loss: 0.2258 − accuracy: 0.9350
Epoch 47/50
 − 1s − loss: 0.2239 − accuracy: 0.9350
Epoch 48/50
 − 1s − loss: 0.2216 − accuracy: 0.9352
Epoch 49/50
 − 1s − loss: 0.2181 − accuracy: 0.9356
Epoch 50/50
 − 1s − loss: 0.2166 − accuracy: 0.9370
Test Accuracy: 0.919900
========== NN2 session completed ===========
Number of epochs: 100
========== NN1 session starts ===========
Epoch 1/100
 − 0s − loss: 60.1685 − accuracy: 0.4158
Epoch 2/100
 − 0s − loss: 14.5209 − accuracy: 0.7547
Epoch 3/100
 − 0s − loss: 9.8338 − accuracy: 0.8148
Epoch 4/100
 − 0s − loss: 7.9235 − accuracy: 0.8352
.
.
.
Epoch 96/100
 − 0s − loss: 1.0928 − accuracy: 0.8948
Epoch 97/100
 − 0s − loss: 1.0831 − accuracy: 0.8970
Epoch 98/100
 − 0s − loss: 1.0674 − accuracy: 0.8969
Epoch 99/100
 − 0s − loss: 1.0682 − accuracy: 0.8964
Epoch 100/100
 − 0s − loss: 1.0557 − accuracy: 0.8969
Test Accuracy: 0.883700
========== NN1 session completed ===========
========== NN2 session starts ===========
Epoch 1/100
 − 1s − loss: 23.6178 − accuracy: 0.3468
Epoch 2/100
 − 1s − loss: 4.7503 − accuracy: 0.6693
Epoch 3/100
 − 1s − loss: 2.5675 − accuracy: 0.7556
Epoch 4/100
 − 1s − loss: 1.7952 − accuracy: 0.7940
.
.
.
Epoch 96/100
 − 1s − loss: 0.1688 − accuracy: 0.9516
Epoch 97/100
 − 1s − loss: 0.1670 − accuracy: 0.9512
Epoch 98/100
 − 1s − loss: 0.1662 − accuracy: 0.9523
Epoch 99/100
 − 1s − loss: 0.1663 − accuracy: 0.9513
Epoch 100/100
 − 1s − loss: 0.1631 − accuracy: 0.9520
Test Accuracy: 0.930600
========== NN2 session completed ===========
```