Iowa State University

Department of Electrical and Computer Engineering

Deep Machine Learning: Theory and Practice

EE 526X

Homework 4

Author:
Vishal Deep

Instructor:
Dr. Zhengdao Wang

November 17, 2019

Iowa State University

1 Problem 1

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Nov 16 02:53:38 2019

@author: vishal
"""


import tensorflow as tf
from sklearn.model_selection import train_test_split
from time import time
import keras.backend as K
from tensorflow.python.keras.callbacks import TensorBoard

def processData():
    # Load fashion MNIST data set
    (x_train, y_train), (x_test, y_test) =tf.keras.datasets.fashion_mnist.load_data()
    # Split into train and validation data
    x_train, x_val, y_train, y_val =train_test_split(x_train, y_train, test_size=0.2)
    # reshape to conform with size
    x_train =x_train.reshape((x_train.shape[0], 28, 28, 1))
    x_test =x_test.reshape((x_test.shape[0], 28, 28, 1))
    x_val =x_val.reshape((x_val.shape[0], 28, 28, 1))
    # Normalize data
    x_train =x_train.astype('float32') /255
    x_test =x_test.astype('float32') /255
    x_val =x_val.astype('float32') /255
    # Do one hot encoding
    y_train =tf.keras.utils.to_categorical(y_train, 10)
    y_val =tf.keras.utils.to_categorical(y_val, 10)
    y_test =tf.keras.utils.to_categorical(y_test, 10)

    return x_train, y_train, x_val, y_val, x_test, y_test

# The network is adapted from the
def createModel():
    model =tf.keras.Sequential()
    model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, padding='same', activation='relu',
                                                 input_shape=(28, 28, 1)))
    model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, padding='same', activation='relu'))
    model.add(tf.keras.layers.MaxPooling2D(pool_size=2, strides=2))
    model.add(tf.keras.layers.Dropout(0.2))
    model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, activation='relu'))
    model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, activation='relu'))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.MaxPooling2D(pool_size=2, strides=2))
    model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(256, activation='relu'))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

```python
    model.summary()
    return model

def getFlops(model):
    run_meta =tf.RunMetadata()
    opts =tf.compat.v1.profiler.ProfileOptionBuilder.float_operation()
    # Keras session graph in the call to the profiler.
    flops =tf.compat.v1.profiler.profile(graph=K.get_session().graph,
                                run_meta=run_meta, cmd='op', options=opts)
    return flops.total_float_ops

batch_size =64
epochs =10
(x_train, y_train, x_val, y_val, x_test, y_test) =processData()

print("x_train shape:", x_train.shape, "y_train shape:", y_train.shape)

model =createModel()

# Print number of flops
print(getFlops(model))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

tensoboard =TensorBoard(log_dir="logs/{}".format(time()))

# Train the network
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=(x_val, y_val),
                                                  callbacks=[tensoboard])

# Evaluate the model on test data
[loss, acc] =model.evaluate(x_test, y_test, verbose=2)
print('\nTest accuracy:', acc)
```

```
==================Model Analysis Report======================

Doc:
op: The nodes are operation kernel type, such as MatMul, Conv2D. Graph nodes belonging to the same type are aggregated together.
flops: Number of float operations. Note: Please read the implementation for the math behind it.

Profile:
node name | # float_ops
Mul 478.11k float_ops (100.00%, 49.97%)
Add 477.22k float_ops (50.03%, 49.88%)
Sub 910 float_ops (0.15%, 0.10%)
Rsqrt 512 float_ops (0.05%, 0.05%)
RealDiv 3 float_ops (0.00%, 0.00%)

======================End of Report==========================
x_train shape: (48000, 28, 28, 1) y_train shape: (48000, 10)
Model: "sequential"
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
conv2d (Conv2D) (None, 28, 28, 32) 320
_________________________________________________________________
conv2d_1 (Conv2D) (None, 28, 28, 32) 9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32) 0
_________________________________________________________________
dropout (Dropout) (None, 14, 14, 32) 0
_________________________________________________________________
conv2d_2 (Conv2D) (None, 12, 12, 64) 18496
_________________________________________________________________
conv2d_3 (Conv2D) (None, 10, 10, 64) 36928
```

```
-----------------------------------------------------------------
batch_normalization (BatchNo (None, 10, 10, 64) 256
-----------------------------------------------------------------
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64) 0
-----------------------------------------------------------------
dropout_1 (Dropout) (None, 5, 5, 64) 0
-----------------------------------------------------------------
flatten (Flatten) (None, 1600) 0
-----------------------------------------------------------------
dense (Dense) (None, 256) 409856
-----------------------------------------------------------------
batch_normalization_1 (Batch (None, 256) 1024
-----------------------------------------------------------------
dropout_2 (Dropout) (None, 256) 0
-----------------------------------------------------------------
dense_1 (Dense) (None, 10) 2570
=================================================================
Total params: 478,698
Trainable params: 478,058
Non-trainable params: 640
-----------------------------------------------------------------
956753
Train on 48000 samples, validate on 12000 samples
Epoch 1/10

   64/48000 [..............................] - ETA: 1:21:45 - loss: 3.1956 - acc: 0.0781
  384/48000 [..............................] - ETA: 13:38 - loss: 1.9243 - acc: 0.3568
  .
  .
47872/48000 [============================>.] - ETA: 0s - loss: 0.5091 - acc: 0.8166
48000/48000 [==============================] - 15s 318us/sample - loss: 0.5089 - acc: 0.8166 - val_loss: 0.3691 - val_acc
       : 0.8597
Epoch 2/10

   64/48000 [..............................] - ETA: 9s - loss: 0.3649 - acc: 0.8438
  384/48000 [..............................] - ETA: 8s - loss: 0.3306 - acc: 0.8750
  .
  .
47808/48000 [============================>.] - ETA: 0s - loss: 0.3293 - acc: 0.8807
48000/48000 [==============================] - 8s 177us/sample - loss: 0.3298 - acc: 0.8804 - val_loss: 0.3535 - val_acc:
       0.8700
Epoch 3/10

   64/48000 [..............................] - ETA: 8s - loss: 0.2136 - acc: 0.9219
  384/48000 [..............................] - ETA: 8s - loss: 0.3025 - acc: 0.8984

Epoch 9/10

   64/48000 [..............................] - ETA: 9s - loss: 0.0963 - acc: 0.9688
  384/48000 [..............................] - ETA: 8s - loss: 0.1696 - acc: 0.9219
  .
  .
47936/48000 [============================>.] - ETA: 0s - loss: 0.1775 - acc: 0.9340
48000/48000 [==============================] - 9s 177us/sample - loss: 0.1774 - acc: 0.9341 - val_loss: 0.1890 - val_acc:
       0.9327

10000/10000 - 1s - loss: 0.2106 - acc: 0.9249

Test accuracy: 0.9249
```

1.1 Calculation of Floating Point Operations

The caluclations of floating point operations of each layer is done below.

1. Conv1: $(28 \times 28 \times 32)$ $(3 \times 3 \times 1 + 1) = 250880$
2. Conv2: $(28 \times 28 \times 32)$ $(3 \times 3 \times 32) = 7225344$
3. Maxpool1: $(14 \times 14 \times 32)$ $(2 \times 2) = 25088$
4. Conv3: $(12 \times 12 \times 64)$ $(3 \times 3 \times 32) = 2654208$
5. Conv4: $(10 \times 10 \times 64)$ $(3 \times 3 \times 64) = 3686400$
6. Maxpool2: $(5 \times 5 \times 64)$ $(2 \times 2) = 6400$

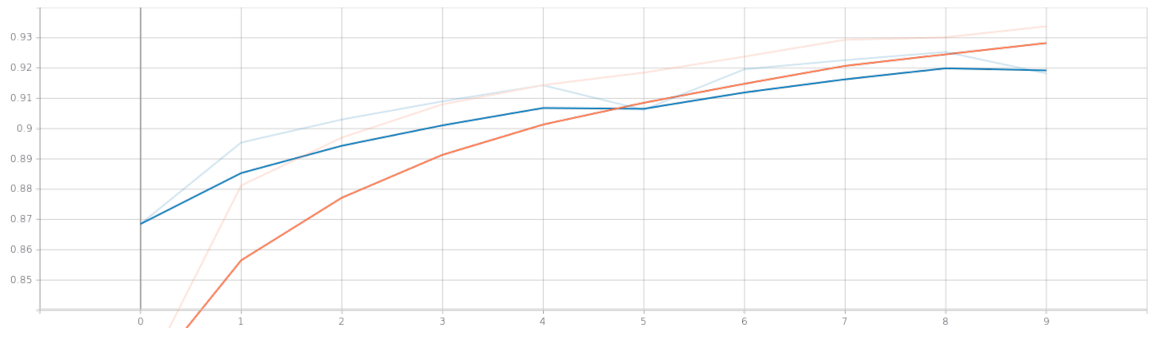

Figure 1: Accuracy vs Epochs. Orange shows training accuracy and blue shows test accuracy

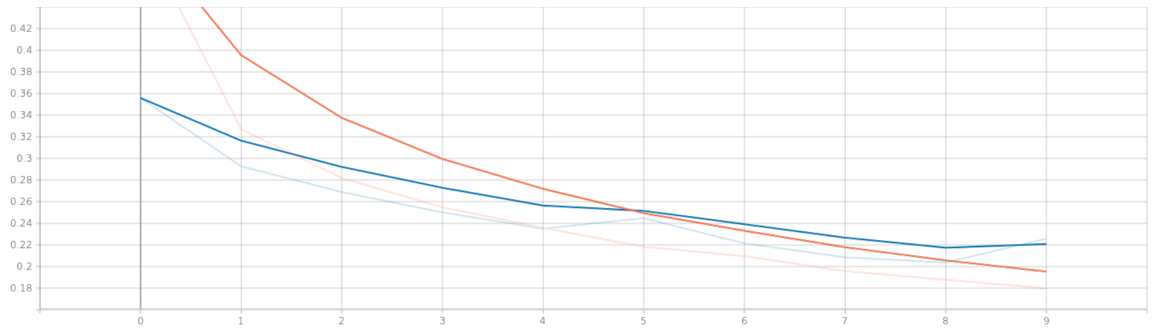

Figure 2: Loss vs Epochs. Orange shows training loss and blue shows test loss

7. Dense: $256 \times (10 \times 5 \times 5) = 64000$

- Total number of FLOPS in one forward pass = 13912320
- The number of floating point operations per iteration step (per mini-batch) = $3 \times 13912320 = 41736960$
- The number of floating point operations per epoch with batch size of 64 = $938 \times 13912320 = 1.305 \times 10^{10}$
- Total number of operations over the whole training process = 13.05×10^{10}

2 Problem 2

```
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
import tensorflow as tf
import math

def processData(look_back):
    # Load the data
    data =np.load('data.npy')
    data =data.astype('float32')

    # Normalize and scale data
    scaler =MinMaxScaler(feature_range=(0, 1))
    data =scaler.fit_transform(data)

    # Split Train and test data
    train_data =data[:66]
    test_data =data[66:]
    x_train, y_train =createXY(train_data, look_back)
    x_test, y_test =createXY(test_data, look_back)
    return x_train, y_train, x_test, y_test

def createXY(data, look_back=1):
    x, y =[], []
    for i in range(len(data)-look_back-1):
        tempData =data[i:(i+look_back), 0]
        x.append(tempData)
        y.append(data[i +look_back, 0])

    return np.array(x), np.array(y)

def createModel(look_back):
    # create the LSTM network
    model =tf.keras.Sequential()
    model.add(tf.keras.layers.LSTM(10, batch_input_shape=(None, 1, look_back), return_sequences=
                                                    True))
    model.add(tf.keras.layers.LSTM(20, return_sequences=True))
    model.add(tf.keras.layers.LSTM(20))
    model.add(tf.keras.layers.Dense(1, activation='linear'))
    model.summary()
    return model

look_back =10
epochs =100
batch_size =10

(x_train, y_train, x_test, y_test) =processData(look_back)

# reshape input to be [samples, time steps, features]
x_train =np.reshape(x_train, (x_train.shape[0], 1, x_train.shape[1]))
x_test =np.reshape(x_test, (x_test.shape[0], 1, x_test.shape[1]))

model =createModel(look_back)
```

```
model.compile(loss='mean_squared_error', optimizer='adam')

# train the model
model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size)

# Make Prediction
y_train_predict =model.predict(x_train)
y_test_predict =model.predict(x_test)

# reshape to match predicted values array
y_train =np.reshape(y_train, (y_train.shape[0], 1))
y_test =np.reshape(y_test, (y_test.shape[0], 1))

# calculate root mean squared error
trainRMSE =math.sqrt(mean_squared_error(y_train[0], y_train_predict[0]))
print(f'Training RMSE: {trainRMSE}\n')
testRMSE =math.sqrt(mean_squared_error(y_test[0], y_test_predict[0]))
print(f'Test RMSE: {testRMSE}\n')
```

```
Model: "sequential_59"
----------------------------------------------------------------------
Layer (type) Output Shape Param #
======================================================================
lstm_15 (LSTM) (None, 1, 10) 840
----------------------------------------------------------------------
lstm_16 (LSTM) (None, 1, 20) 2480
----------------------------------------------------------------------
lstm_17 (LSTM) (None, 20) 3280
----------------------------------------------------------------------
dense_64 (Dense) (None, 1) 21
======================================================================
Total params: 6,621
Trainable params: 6,621
Non-trainable params: 0
----------------------------------------------------------------------
Epoch 1/100
55/55 [==============================] - 4s 78ms/sample - loss: 0.3621
Epoch 2/100
55/55 [==============================] - 0s 749us/sample - loss: 0.3398
Epoch 3/100
55/55 [==============================] - 0s 756us/sample - loss: 0.3171
Epoch 4/100
55/55 [==============================] - 0s 695us/sample - loss: 0.2933
.
.
.
Epoch 98/100
55/55 [==============================] - 0s 815us/sample - loss: 0.0334
Epoch 99/100
55/55 [==============================] - 0s 688us/sample - loss: 0.0331
Epoch 100/100
55/55 [==============================] - 0s 784us/sample - loss: 0.0333
Training RMSE: 0.058698832142092955

Test RMSE: 0.20008337101134305
```

The network has three LSTM layers with 10, 20, and 20 units and one Dense layer at the end with *linear* activation. The model summary is printed above with total number of parameters. While designing the network number of issues were faced and solved successfully. The input shape in the first layer was not aligned in the starting but was later solved. Other issue was this is not classification dataset but we have worked with classification problems in the class. It took some time to find out how to calculate the accuracy of the designed network. The design choices were made based on the exporation of different network architectures and finally using the one with highest accuracy or lower RMSE value.

- Training RMSE: 0.0587, Test RMSE: 0.20