Deep Learning Assignment 3

Nikita Teplitskiy

September 23, 2020

1 Introduction

This assignment was completed with TensorFlow using a two layer perceptron. The completed model achieves a 95.99% accuracy on the provided test dataset and a 95.63% accuracy on the validation subset. It uses a total of 14,320 trainable parameters. As part of the assignment, a custom data loader titled "MNISTload.py" was developed and is provided below.

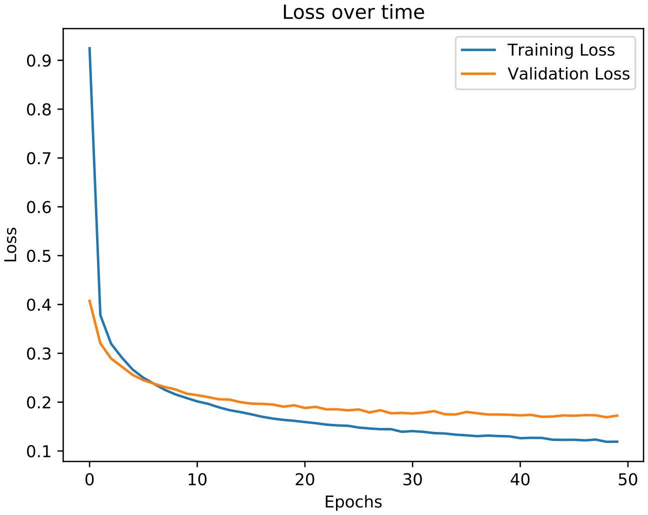

Figure 1: Loss over training iterations

From the loss plot one can see that the validation loss begins to exceed the training loss within the first 10 epochs. This suggests a degree of overfitting and increased regularization would be required to fix this. This was not done as the desired accuracy had been achieved.

2 Model

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import Model

from tensorflow.keras.layers import Dense, Flatten, Conv2D, Dropout
#from tensorflow.keras import Model

import numpy as np
import matplotlib.pyplot as plt
import MNISTload

mnist = tf.keras.datasets.mnist

(x_pool, y_pool), (x_test, y_test) = MNISTload.load_MNIST()
#convert to float
x_pool, x_test = x_pool.astype(”float32”) / 255.0, x_test.astype(”float32”) / 255.0

x_train, y_train = x_pool[:3*len(x_pool)//4], y_pool[:3*len(x_pool)//4]
x_valid, y_valid = x_pool[3*len(x_pool)//4:], y_pool[3*len(x_pool)//4:]

#try to get rid of these datasets
#can use Dataset.take Dataset.skip for some of these
train_ds = tf.data.Dataset.from_tensor_slices((x_train, y_train)).batch(32)
valid_ds = tf.data.Dataset.from_tensor_slices((x_valid, y_valid)).batch(32)
test_ds = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(1)

class MyModel(Model):
  def __init__(self):
    super(MyModel, self).__init__()
    self.flatten = Flatten() #just a convenience layer to reshape my inputs
    self.d1 = Dense(18, activation='elu', kernel_regularizer=keras.regularizers.l2(0.0001))
    #dropout hidden layer
    self.drp1 = Dropout(0.01)
    #output layer fully connected
    self.d2 = Dense(10)
  def call(self, x):
    x = self.flatten(x)
    x = self.d1(x)
    x = self.drp1(x)
    return self.d2(x)

# Create an instance of the model
model = MyModel()

loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
optimizer = tf.keras.optimizers.Adam(learning_rate=0.004, beta_1=0.9, beta_2=0.999)

train_loss = tf.keras.metrics.Mean(name='train_loss')
```

```
train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='train_accuracy')

valid_loss = tf.keras.metrics.Mean(name='valid_loss')
valid_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='valid_accuracy')

test_loss = tf.keras.metrics.Mean(name='test_loss')
test_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='test_accuracy')

model.compile(optimizer=optimizer, loss=loss_object, metrics=[train_accuracy])
history = model.fit(x_train, y_train, batch_size=1024, epochs=50, validation_data=(x_valid
model.summary()
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title("Loss over time")
plt.xlabel('Epochs'); plt.ylabel('Loss')
plt.savefig('loss.pdf', format='pdf')
plt.show()

#test set results
results = model.evaluate(x_test,y_test, batch_size=32)
print("Tests set loss and accuracy:", results)
```

3 Data Loader

```
import numpy as np
import matplotlib.pyplot as plt

testimgfile = open('t10k-images-idx3-ubyte', 'rb')
testlabelfile = open('t10k-labels-idx1-ubyte', 'rb')

trainimgfile = open('train-images-idx3-ubyte', 'rb')
trainlabelfile = open('train-labels-idx1-ubyte', 'rb')

def get_images(f):
    if (int.from_bytes(f.read(4), 'big') != 2051):
        print("Not␣an␣image␣file")
        return None
    else:
        num_images = int.from_bytes(f.read(4), 'big')
        #print(num_images)

        dim = (int.from_bytes(f.read(4), 'big'), int.from_bytes(f.read(4), 'big'))
        pixels = np.fromfile(f, np.uint8, num_images*dim[0]*dim[1], '', 0)
        images = pixels.reshape([num_images,28,28])
        return images

def get_labels(f):
    if (int.from_bytes(f.read(4), 'big') != 2049):
        print("Not␣a␣label␣file")
        return None
    else:
        num_labels = int.from_bytes(f.read(4), 'big')
        #print(num_labels)
        #offset here is 0 as I have already moved the file index
        labelvec = np.fromfile(f, np.uint8, num_labels, "", 0)
        return labelvec

def load_MNIST():
    x_train = get_images(trainimgfile)
    y_train = get_labels(trainlabelfile)
    x_test = get_images(testimgfile)
    y_test = get_labels(testlabelfile)
    return (x_train,y_train),(x_test,y_test)

if __name__ == '__main__':
    _, (x,y) = load_MNIST()
    #y = get_labels(testlabelfile)
    print(y[0])
    #x = get_images(testimgfile)
    plt.imshow(x[0,:,:])
    plt.show()
```