Nithilam Subbaian

9/26/2019

HW3

```
#!/bin/python3.6
import numpy as np
import tensorflow as tf
from tqdm import trange
import matplotlib.pyplot as plt
import gzip
import pydot

# Consider the mnist dataset consisting of 50,000 training images, and 10,000 test
images.
# Each instance is a 28 × 28 pixel handwritten digit zero through nine.
# Train a convolutional neural network for classification using the training set that
achieves
# at least 95.5% accuracy on the test set. Do not explicitly tune hyperparameters based
on the
#  test set performance, use a validation set taken from the training set as discussed in
class.
# Use dropout and an L2 penalty for regularization.

# Sources (in addition to TensorFlow documentation):
# http://yann.lecun.com/exdb/mnist/
# https://towardsdatascience.com/a-simple-2d-cnn-for-mnist-digit-recognition-a998dbc1e79a
# https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-
54c35b77a38d
# http://cs231n.github.io/convolutional-networks/
# https://medium.com/datadriveninvestor/my-take-at-the-mnist-dataset-97304dff2057

BATCH_SIZE = 50
EPOCHS = 3
FILTERS_1 = 36
KERNEL_SIZE_1 = 5

class Data(object):
    def __init__(self):

        self.train_images, self.train_labels = self.loadMNISTfromFile('train-images-idx3-
ubyte.gz', 'train-labels-idx1-ubyte.gz')
        features, labels  = self.train_images, self.train_labels
        rand_index = np.arange(60000)
        np.random.shuffle(rand_index)

        rand_features = np.reshape(features[rand_index], [-1,28,28,1])
        self.train_feature, self.valid_feature = rand_features[:-10000], rand_features[-
10000:]

        rand_labels = tf.keras.utils.to_categorical(labels[rand_index], 10)
        self.train_label, self.valid_label = rand_labels[:-10000], rand_labels[-10000:]

        self.test_images, self.test_labels = self.loadMNISTfromFile('t10k-images-idx3-
ubyte.gz', 't10k-labels-idx1-ubyte.gz')
        self.test_images = np.reshape(self.test_images, [-1,28,28,1])
        self.test_labels = tf.keras.utils.to_categorical(self.test_labels, 10)
```

```
    def loadMNISTfromFile(self, filename_images, filename_labels):
        # Read inputs into vector
        with gzip.open(filename_images, 'rb') as f:
            data = np.frombuffer(f.read(), np.uint8, offset=16)
        data = data.reshape(-1, 1, 28, 28) /255.0 #(examples, channels, rows, columns)

        # Repeat process for labels
        with gzip.open(filename_labels, 'rb') as f:
            label = np.frombuffer(f.read(), np.uint8, offset=8)

        return data, label

class Model(tf.Module):
    def __init__(self):

        self.model = tf.keras.Sequential()

        # 2D convolution layer (e.g. spatial convolution over images).
        self.model.add(tf.keras.layers.Conv2D(filters= FILTERS_1, # dimensionality of the
output space, the number of output filters in the convolution
                                              kernel_size= KERNEL_SIZE_1, # specifying
the height and width of the 2D convolution window.
                                              padding='same', # input image is padded
with zeroes so the size of output is the same
                                              activation='relu',
                                              input_shape=(28,28,1)))

        # Max Pooling - down-sample an input representation, reducing its dimensionality
and allowing for assumptions
        # to be made about features contained in the sub-regions binned.
        self.model.add(tf.keras.layers.MaxPooling2D(pool_size= (2, 2))) # factors by
which to downscale (vertical, horizontal)

        # Dropout - randomly setting a fraction rate of input units to 0 at each update
        # during training time --> helps prevent overfitting
        self.model.add(tf.keras.layers.Dropout(0.40))

        # Flattens input, bc too many dimensions
        # Only need classification output
        self.model.add(tf.keras.layers.Flatten())

        # Dense --> Your regular densely-connected NN layer
        # Implements the operation: output = activation(dot(input, kernel) + bias)
        # Output a softmax, turns matrix into output probabilities
        self.model.add(tf.keras.layers.Dense(10, activation='softmax',
kernel_regularizer=tf.keras.regularizers.l2(0.001)))

        # Configures the model for training
        self.model.compile(loss = 'categorical_crossentropy',
                           optimizer = 'adam',
                           metrics = ['accuracy'])

        self.model.summary()


if __name__ == "__main__":
    data = Data()
    CNNmodel = Model()
```

```
    history = CNNmodel.model.fit(data.train_feature,
                                 data.train_label,
                                 batch_size = BATCH_SIZE,
                                 epochs = EPOCHS,
                                 validation_data = (data.valid_feature,
data.valid_label))

    test_loss, test_accuracy = CNNmodel.model.evaluate(data.test_images,
data.test_labels, verbose = 0)

    # Accuracy and Loss on test set
    print("Test set loss is", test_loss)
    print("Test set Accuracy is", test_accuracy)

    plt.figure(1, figsize=[30,30])
    plt.subplot(121)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model Accuracy for Training and Validation Set')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Training', 'Validation'], loc='upper left')

    plt.subplot(122)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss for Training and Validation Set')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Training', 'Validation'], loc='upper left')
    plt.show()
```

```
(base) C:\Users\nithm.DESKTOP-LI5EB54\OneDrive\Documents>python mnistCNN1.py
2019-09-25 20:41:01.326749: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 36)        936
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 36)        0
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 36)        0
_________________________________________________________________
flatten (Flatten)            (None, 7056)              0
_________________________________________________________________
dense (Dense)                (None, 10)                70570
=================================================================
Total params: 71,506
Trainable params: 71,506
Non-trainable params: 0
_________________________________________________________________
Train on 50000 samples, validate on 10000 samples
Epoch 1/3
50000/50000 [==============================] - 38s 753us/sample - loss: 0.3008 - accuracy: 0.9256 - val_loss: 0.1634 - val_accuracy: 0.9661
Epoch 2/3
50000/50000 [==============================] - 39s 786us/sample - loss: 0.1598 - accuracy: 0.9679 - val_loss: 0.1382 - val_accuracy: 0.9760
Epoch 3/3
50000/50000 [==============================] - 36s 712us/sample - loss: 0.1385 - accuracy: 0.9732 - val_loss: 0.1268 - val_accuracy: 0.9776
Test set loss is 0.11539376320838929
Test set Accuracy is 0.9799
```

"sequential"

_____

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 36)	936
max_pooling2d (MaxPooling2D)	(None, 14, 14, 36)	0
dropout (Dropout)	(None, 14, 14, 36)	0
flatten (Flatten)	(None, 7056)	0
dense (Dense)	(None, 10)	70570

Total params: 71,506
Trainable params: 71,506
Non-trainable params: 0

_____

Train on 50000 samples, validate on 10000 samples
Epoch 1/3
50000/50000 [==============================] - 47s 937us/sample - loss: 0.2999 - accuracy: 0.9260 - val_loss: 0.1523 - val_accuracy: 0.9693
Epoch 2/3
50000/50000 [==============================] - 38s 763us/sample - loss: 0.1599 - accuracy: 0.9671 - val_loss: 0.1289 - val_accuracy: 0.9771
Epoch 3/3
50000/50000 [==============================] - 44s 884us/sample - loss: 0.1405 - accuracy: 0.9731 - val_loss: 0.1193 - val_accuracy: 0.9795

Test set loss is 0.1140761171221733
Test set Accuracy is 0.9788