ECE472
Deep Learning - Assignment 3

Yuval Epstain Ofek

September 23, 2020

Summary

The MNIST dataset was found on the yann.lecun website (see link), and extracted. A validation set comprising of roughly 10% of the training set was created. The number of examples falling under each label was considered in the creation of the validation set, choosing to randomly move 10% of the examples of each label from the training set to the validation set (up to some rounding). The resulting probabilities for each of the labels differed by less than 0.001 across the two sets. The training, validation, and test sets were formatted to interface with the keras API through reshaping and making the labels categorical. ImageDataGenerator objects were created to perform data augmentation on the training set as well as to re-scale all three sets.

The architecture chosen, from input to output layer, is as follows: [Conv2D, MaxPooling2D, Conv2D, MaxPooling2D, Dropout, Dense, Dropout, Dense]. The first Conv2D layer had 16 (3,3) filters, a ReLU activation, and used L^2 regularization, while the second had 32 (3,3) filters with the other parameters the same. Both MaxPooling2D layers had a (2,2) pool size. Both dropout layers had a probability of dropout of 0.5. The first Dense layer had 32 neurons, a ReLU activation, and L^2 regularization enabled. The last Dense layer had 10 neurons with a softmax activation. The internal parameters such as learning rate and the regularization coefficients were left as initialized by keras.

A callback class was created to stop training once validation exceeds 96% accuracy at the end of an epoch, and used while training. This was done so that I have confidence that the model meets or exceeds the required 95.5% test accuracy. The model was trained using the Adam optimizer, a categorical cross-entropy loss, and has been trained until the callback occurred. Note that if a model did not have a callback in 30 epochs, I scrapped the entire model. I initially had a lot more filters in the convolutional layers and a much bigger Dense layer, but after running the model again and again and seeing that the validation still reaches the expected 95.5% after reducing the sizes and quantities I settled with the numbers listed above. The final test accuracy was 95.9% percent, exceeding the requirement of 95.5% test accuracy.

Appendix I- Python Code

```python
# -*- coding: utf-8 -*-
"""DeepLearningAssignment3.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1yZ2bbpfrv_dVTd3tAZy1tvG2duG2dX-N

**ECE472, Deep Learning - Assignment 3**

Submit by Sept. 23, 10pm

tldr: Classify mnist digits with a (optionally convoultional) neural network.
Get at least 95.5% accuracy on the test test.
"""

import numpy as np
import pandas as pd
import numpy.random as npr
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import gzip

#Parameters
npr.seed(25)
image_size = 28
train_images = 60_000
test_images = 10_000
VAL_PERCENT = 0.1

def read_values(file_name , read_val, size, num_read):
  '''
  Opens gzip compressed file with name file_name and reads the values in it,
  returning an array of integers of size: (size*size*num_read,), containing the
  information inside the file.

  Referenced:
  https://stackoverflow.com/questions/40427435/extract
    -images-from-idx3-ubyte-file-or-gzip-via-python
  '''
  f = gzip.open(file_name, 'r')
  f.read(read_val)
  buf = f.read(size*size * num_read)
  data = np.frombuffer(buf, dtype=np.uint8).astype('int')
  return data

#Get MNIST dataset (I used Colab notebook, so this worked):
#!wget http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
#!wget http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
#!wget http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
#!wget http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz

#read the values
data_tr = read_values('train-images-idx3-ubyte.gz',
                      16,image_size,train_images).reshape(train_images,-1)
label_tr = read_values('train-labels-idx1-ubyte.gz',
                       8,1,train_images)
data_test = read_values('t10k-images-idx3-ubyte.gz',
                        16,image_size,test_images).reshape(test_images,-1)
label_test = read_values('t10k-labels-idx1-ubyte.gz',
                         8,1,test_images)

#Make into dataframes because I find them nicer to work with
df_X_train = pd.DataFrame(
    np.concatenate((np.expand_dims(label_tr, axis = 1), data_tr), axis = 1),
    dtype = 'int'
    )
df_X_test = pd.DataFrame(
    np.concatenate((np.expand_dims(label_test, axis = 1), data_test), axis = 1),
    dtype = 'int'
    )

### CREATING THE VALIDATION SET:

#Number of examples to be put in validation set
num_val_examples = VAL_PERCENT* train_images
#number of examples with each label in current training set
num_examples_per_class = df_X_train.groupby(0)[0].count()
#number of examples from each label to move to the validation set
val_per_class = list((np.around(
    (num_examples_per_class/train_images)*num_val_examples
    ).astype('int')))
#number of labels
num_classes =len(num_examples_per_class)
```

```
#For each label in current training set, randomly pick examples
#so that their number match what was calculated earlier and add it
#to a big list
val_idxs = []
for i in range(num_classes):
  val_idxs.append(npr.choice(df_X_train[df_X_train[0] == i].index,
                             size = val_per_class[i]).tolist())
val_idxs = [item for sublist in val_idxs for item in sublist]

#take those chosen examples and make a validation set, while also
#removing them from the training set
df_X_val = df_X_train.loc[val_idxs]
df_X_train = df_X_train.drop(val_idxs)


#checking to make sure validation and training sets have same distribution
percent_classes_train = list(df_X_train.groupby(0)[0].count()/len(df_X_train))
percent_classes_val = list(df_X_val.groupby(0)[0].count()/len(df_X_val))

print('All probabilities match up to 1e-3:',
    any([(percent_classes_train[i] - percent_classes_val[i])<1e-3
 for i in range(len(percent_classes_val)) ]))

#extract labels
df_y_train = df_X_train.pop(0)
df_y_test = df_X_test.pop(0)
df_y_val = df_X_val.pop(0)

#to numpy and reshapes
X_train = df_X_train.to_numpy().reshape(-1, 28,28, 1)
y_train = df_y_train.to_numpy()

X_val = df_X_val.to_numpy().reshape(-1,28,28, 1)
y_val = df_y_val.to_numpy()

X_test = df_X_test.to_numpy().reshape(-1,28,28, 1)
y_test = df_y_test.to_numpy()

#make y categorical
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)
y_val = tf.keras.utils.to_categorical(y_val, num_classes)

# Training generator
```

```
train_datagen = ImageDataGenerator(
    rescale = 1./255,
    rotation_range=10,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.1,
    zoom_range = 0.05,
    fill_mode = 'nearest'
)
train_generator = train_datagen.flow(
    X_train,
    y_train,
    batch_size=32,
)

#Validation generator
val_datagen = ImageDataGenerator(
    rescale = 1./255
)
val_generator = val_datagen.flow(
    X_val,
    y_val,
)

#Test generator
test_datagen = ImageDataGenerator(
    rescale = 1./255
)
test_generator = test_datagen.flow(
    X_test,
    y_test,
)

#Create Model
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), input_shape = (28,28,1),
                           activation = 'relu',
                           activity_regularizer='l2'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32, (3,3), activation = 'relu',
                           activity_regularizer='l2'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(16, activation = 'relu', activity_regularizer='l2'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(num_classes,activation = 'softmax'),
])

# Compile Model.
model.compile(loss = "categorical_crossentropy" ,
             optimizer = 'adam',
             metrics = ['acc'])

# Callback class that stops training once validation accuracy reaches 96%
class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('val_acc')>0.96):
      print("\n\nValidation reached 96% accuracy so stopping training!")
      self.model.stop_training = True

callback = myCallback()

# Train the Model
history = model.fit(
    train_generator,
    epochs = 30,
    verbose = 1,
    validation_data = val_generator,
    callbacks = [callback]
)

# Learning Curve
plt.figure(figsize = (10,10))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.xlabel('Epochs', fontsize = 14)
h = plt.ylabel('Loss',fontsize = 14)
h.set_rotation(0)
plt.title('Learning Curve', fontsize = 18)
plt.legend(['Training Loss', 'Validation Loss'])
plt.show()

#Evaluate on test set
results = model.evaluate(test_generator, verbose=0)
print('Test Accuracy:', results[1])
```