Assignment 5.2

January 6, 2021

0.1 File information

File: Assignment_5.2.ipynb

Name: Amie Davis

Date: 1/6/2021

Course: DSC650 - Big Data

Assignment Number: 5.2

Purpose: Implement the news classifier found in section 3.5 of Deep Learning with Python.

1 Classifying newswires: a multi-class classification example

1.1 This file contains code from Deep Learning with Python

www.manning.com/books/deep-learning-with-python

Copyright 2018 Francois Chollet

1.2 Data Source: The Reuters dataset - comes packaged with Keras.

[2]:
```
import keras
keras.__version__
```

[2]:
```
'2.3.1'
```

1.3 Load the data

[3]:
```
# Use Keras Reuters dataset for newswires and topics
# There are 46 separate topics
# Each topic has at least 10 examples in the training set.
# Split data into training & test datasets
# Keep the top 10,000 most frequently occurring words

from keras.datasets import reuters

(train_data, train_labels), (test_data, test_labels) = reuters.
 ↪load_data(num_words=10000)
```

[4]:
```
# Review Data

train_data[10]
```

[4]:
```
[1,
 245,
 273,
 207,
 156,
 53,
 74,
 160,
 26,
 14,
 46,
 296,
 26,
 39,
 74,
 2979,
 3554,
 14,
 46,
 4689,
 4329,
 86,
 61,
 3499,
 4795,
 14,
 61,
 451,
 4329,
 17,
 12]
```

[5]:
```
# Review Labels
train_labels[10]
```

[5]:
```
3
```

1.4 Prepare the data

[6]:
```
# Use one-hot-encoding to turn data into vectors of 0s and 1s
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
```

```
    # Create an all-zero matrix of shape (len(sequences), dimension)
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.    # set specific indices of results[i] to 1s
    return results

# Vectorize training data
x_train = vectorize_sequences(train_data)

# Vectorize test data
x_test = vectorize_sequences(test_data)
```

[7]:
```
# Review Vectorized Data
x_train[0]
```

[7]:
```
array([0., 1., 1., …, 0., 0., 0.])
```

[8]:
```
# Use one-hot-encoding on labels

def to_one_hot(labels, dimension=46):
    results = np.zeros((len(labels), dimension))
    for i, label in enumerate(labels):
        results[i, label] = 1.
    return results

# Vectorize training labels
one_hot_train_labels = to_one_hot(train_labels)

# Vectorize test labels
one_hot_test_labels = to_one_hot(test_labels)
```

1.5 Build Keras Neural Network Model

[9]:
```
from keras.utils.np_utils import to_categorical

one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)
```

[10]:
```
# Define the model
from keras import models
from keras import layers

# input_shape is size of data vector
# 64 hidden layers (Must be greater than number of classes)
# Use softmax for output fxn since multi classification problem
model = models.Sequential()
```

```
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))
```

[11]:
```
# Compile Model

# Use categorical_crossentropy for multi classification problem
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

1.6 Validate Model

[12]:
```
# Create Validation Set (set apart 1,000 samples)
x_val = x_train[:1000]
partial_x_train = x_train[1000:]

y_val = one_hot_train_labels[:1000]
partial_y_train = one_hot_train_labels[1000:]
```

[13]:
```
# Train Model
# Collect measurement logs
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
WARNING:tensorflow:From C:\Users\amomu\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables
is deprecated. Please use tf.compat.v1.global_variables instead.

Train on 7982 samples, validate on 1000 samples
Epoch 1/20
7982/7982 [==============================] - 1s 100us/step - loss: 2.7280 -
accuracy: 0.5217 - val_loss: 1.8033 - val_accuracy: 0.6390
Epoch 2/20
7982/7982 [==============================] - 1s 82us/step - loss: 1.4828 -
accuracy: 0.6966 - val_loss: 1.3606 - val_accuracy: 0.7130
Epoch 3/20
7982/7982 [==============================] - 1s 79us/step - loss: 1.1000 -
accuracy: 0.7633 - val_loss: 1.1703 - val_accuracy: 0.7420
Epoch 4/20
7982/7982 [==============================] - 1s 80us/step - loss: 0.8636 -
accuracy: 0.8186 - val_loss: 1.1118 - val_accuracy: 0.7460
Epoch 5/20
7982/7982 [==============================] - 1s 83us/step - loss: 0.6846 -
```

```
accuracy: 0.8578 - val_loss: 0.9911 - val_accuracy: 0.7980
Epoch 6/20
7982/7982 [==============================] - 1s 78us/step - loss: 0.5442 -
accuracy: 0.8877 - val_loss: 0.9283 - val_accuracy: 0.8120
Epoch 7/20
7982/7982 [==============================] - 1s 81us/step - loss: 0.4400 -
accuracy: 0.9082 - val_loss: 0.9160 - val_accuracy: 0.8110
Epoch 8/20
7982/7982 [==============================] - 1s 82us/step - loss: 0.3531 -
accuracy: 0.9261 - val_loss: 0.8929 - val_accuracy: 0.8160
Epoch 9/20
7982/7982 [==============================] - 1s 81us/step - loss: 0.2922 -
accuracy: 0.9376 - val_loss: 0.9007 - val_accuracy: 0.8190
Epoch 10/20
7982/7982 [==============================] - 1s 79us/step - loss: 0.2497 -
accuracy: 0.9437 - val_loss: 0.9369 - val_accuracy: 0.8110
Epoch 11/20
7982/7982 [==============================] - 1s 78us/step - loss: 0.2131 -
accuracy: 0.9483 - val_loss: 0.8927 - val_accuracy: 0.8120
Epoch 12/20
7982/7982 [==============================] - 1s 79us/step - loss: 0.1845 -
accuracy: 0.9503 - val_loss: 0.9233 - val_accuracy: 0.8240
Epoch 13/20
7982/7982 [==============================] - 1s 81us/step - loss: 0.1650 -
accuracy: 0.9533 - val_loss: 0.9760 - val_accuracy: 0.8110
Epoch 14/20
7982/7982 [==============================] - 1s 79us/step - loss: 0.1526 -
accuracy: 0.9531 - val_loss: 1.0695 - val_accuracy: 0.7930
Epoch 15/20
7982/7982 [==============================] - 1s 80us/step - loss: 0.1407 -
accuracy: 0.9562 - val_loss: 1.0179 - val_accuracy: 0.8090
Epoch 16/20
7982/7982 [==============================] - 1s 82us/step - loss: 0.1356 -
accuracy: 0.9546 - val_loss: 1.1051 - val_accuracy: 0.8000
Epoch 17/20
7982/7982 [==============================] - 1s 77us/step - loss: 0.1267 -
accuracy: 0.9569 - val_loss: 1.0788 - val_accuracy: 0.7940
Epoch 18/20
7982/7982 [==============================] - 1s 79us/step - loss: 0.1217 -
accuracy: 0.9569 - val_loss: 1.0581 - val_accuracy: 0.8030
Epoch 19/20
7982/7982 [==============================] - 1s 79us/step - loss: 0.1192 -
accuracy: 0.9559 - val_loss: 1.0829 - val_accuracy: 0.7980
Epoch 20/20
7982/7982 [==============================] - 1s 80us/step - loss: 0.1121 -
accuracy: 0.9588 - val_loss: 1.0887 - val_accuracy: 0.8040
```

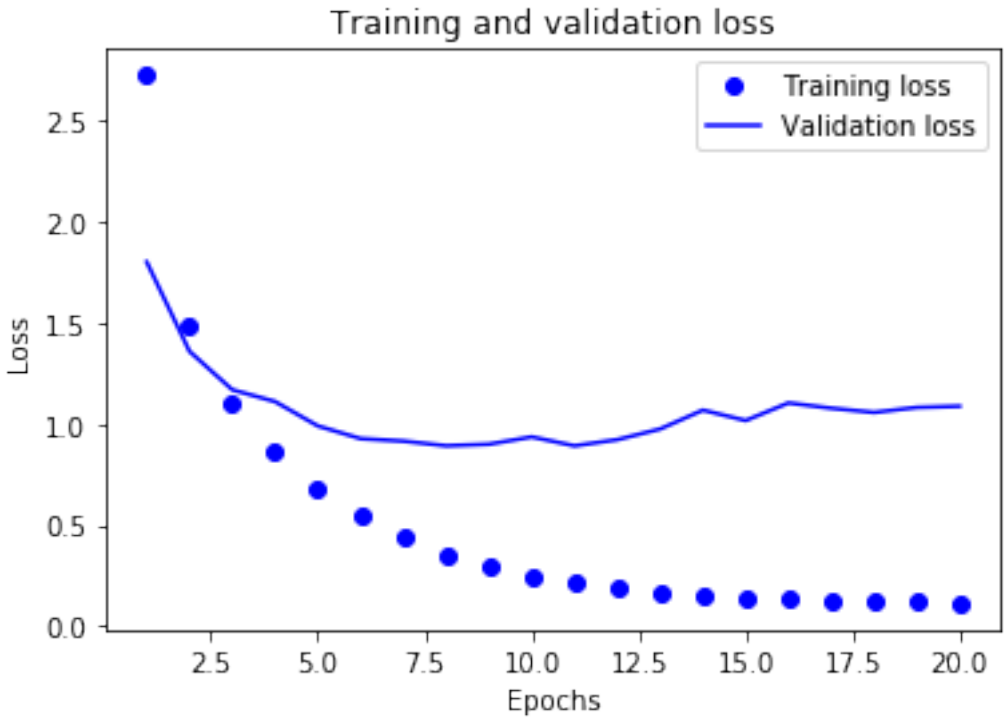
[14]:
```
# Plot the training and validation loss
import matplotlib.pyplot as plt

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

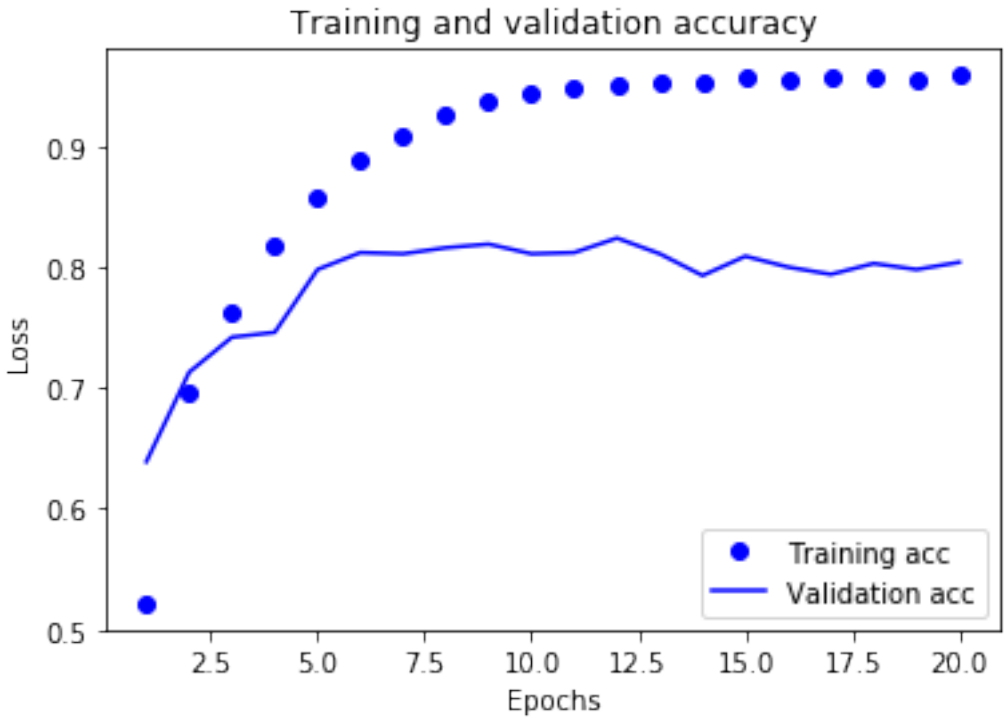


[15]:
```
# Plot the training and validation accuracy
plt.clf()   # clear figure

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
```

```
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



1.7 Re-Train & Evaluate Model

[16]:
```
# Re-train model based on plots
# Accuracy gets lower after 7.5 epochs
# Loss increases lower after 7.5 epochs
# Re-train with 8 epochs

model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
```

```
              metrics=['accuracy'])

model.fit(partial_x_train,
          partial_y_train,
          epochs=8,
          batch_size=512,
          validation_data=(x_val, y_val))
results = model.evaluate(x_test, one_hot_test_labels)
```

```
Train on 7982 samples, validate on 1000 samples
Epoch 1/8
7982/7982 [==============================] - 1s 93us/step - loss: 2.6350 -
accuracy: 0.5044 - val_loss: 1.7732 - val_accuracy: 0.6340
Epoch 2/8
7982/7982 [==============================] - 1s 86us/step - loss: 1.4569 -
accuracy: 0.7046 - val_loss: 1.3285 - val_accuracy: 0.7230
Epoch 3/8
7982/7982 [==============================] - 1s 81us/step - loss: 1.0759 -
accuracy: 0.7735 - val_loss: 1.1609 - val_accuracy: 0.7390
Epoch 4/8
7982/7982 [==============================] - 1s 86us/step - loss: 0.8537 -
accuracy: 0.8193 - val_loss: 1.0629 - val_accuracy: 0.7810
Epoch 5/8
7982/7982 [==============================] - 1s 86us/step - loss: 0.6859 -
accuracy: 0.8583 - val_loss: 0.9899 - val_accuracy: 0.8080
Epoch 6/8
7982/7982 [==============================] - 1s 81us/step - loss: 0.5509 -
accuracy: 0.8893 - val_loss: 0.9825 - val_accuracy: 0.7950
Epoch 7/8
7982/7982 [==============================] - 1s 82us/step - loss: 0.4443 -
accuracy: 0.9077 - val_loss: 0.9363 - val_accuracy: 0.8110
Epoch 8/8
7982/7982 [==============================] - 1s 92us/step - loss: 0.3601 -
accuracy: 0.9252 - val_loss: 0.9200 - val_accuracy: 0.8220
2246/2246 [==============================] - 0s 99us/step
```

[17]:
```
# Show Evaluation results
results
```

[17]:
```
[1.001386606364195, 0.7849510312080383]
```

Achieves an accuracy of ~78%.

1.8 Use Model to Generate predictions

[19]:
```
# Show predictions as vector of length 46 (number of classes) that total 1␣
↪(100%)
# Each entry is probability of that class
predictions = model.predict(x_test)
```

[20]:
```
# The largest entry is the predicted class, i.e. the class with the highest␣
↪probability:
np.argmax(predictions[0])
```

[20]: 3