Holly Figueroa

DSC650 Week 5

Book Exercise 3.4

Binary Classification of Movie Reviews

In [ ]:
```
from keras.src.engine.sequential import Sequential
from keras.datasets import imdb
import numpy as np
from keras import layers
from keras.models import Sequential
```

In [ ]:
```
# Load data
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=100
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/i
mdb.npz
17464789/17464789 [==============================] - 0s 0us/step
```

In [ ]:
```
# Encode the integer sequences into a binary matrix
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results
```

In [ ]:
```
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

In [ ]:
```
# Vectorize labels
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

In [ ]:
```
# define model
model=Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

In [ ]:
```
# Vaidate - create validation set
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

In [ ]:
```
# Model training
model.compile(optimizer =  'rmsprop',
              loss = 'binary_crossentropy',
              metrics =['acc'])
```

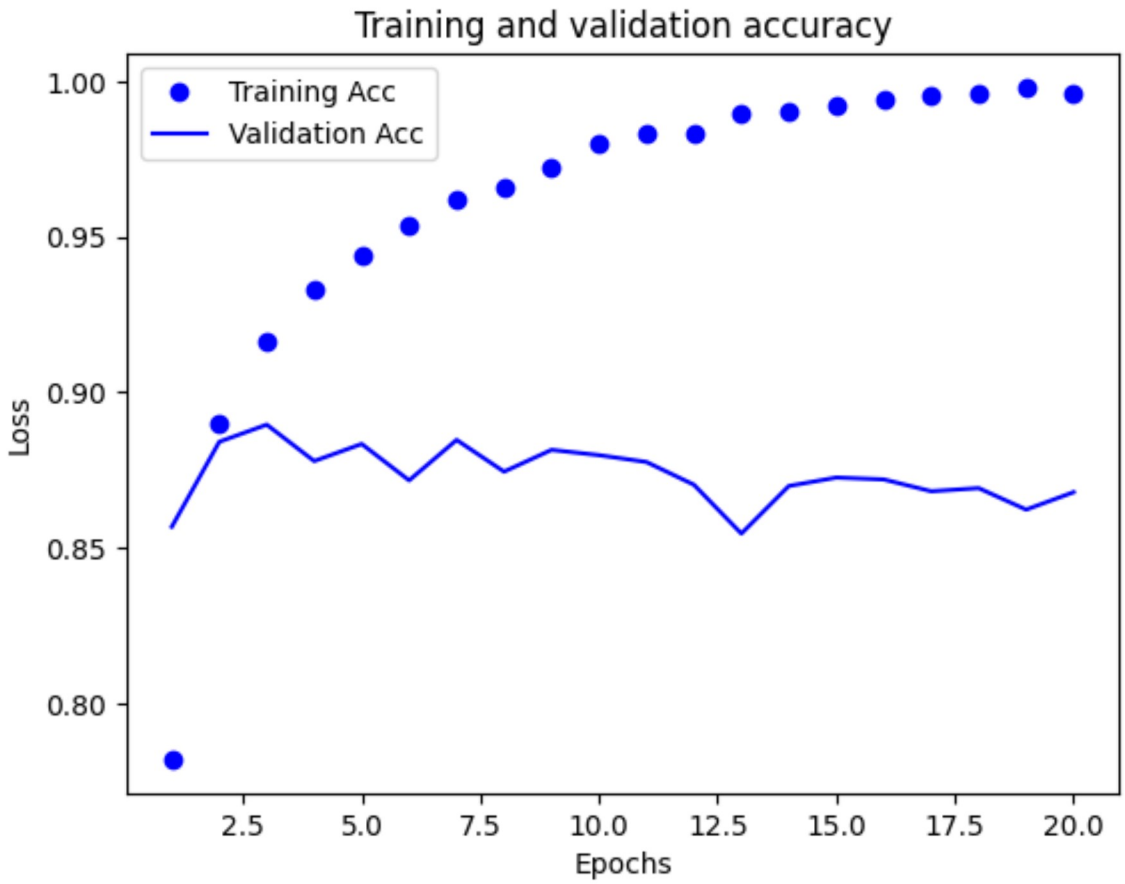
In [ ]:
```
epochs = 20
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=epochs,
                    batch_size=512,
                    validation_data=(x_val,y_val))
```

```
Epoch 1/20
30/30 [==============================] - 3s 67ms/step - loss: 0.5262 - acc: 0.7821
- val_loss: 0.4026 - val_acc: 0.8568
Epoch 2/20
30/30 [==============================] - 1s 30ms/step - loss: 0.3243 - acc: 0.8898
- val_loss: 0.3088 - val_acc: 0.8841
Epoch 3/20
30/30 [==============================] - 1s 27ms/step - loss: 0.2416 - acc: 0.9165
- val_loss: 0.2810 - val_acc: 0.8896
Epoch 4/20
30/30 [==============================] - 2s 66ms/step - loss: 0.1938 - acc: 0.9333
- val_loss: 0.3003 - val_acc: 0.8779
Epoch 5/20
30/30 [==============================] - 2s 51ms/step - loss: 0.1640 - acc: 0.9439
- val_loss: 0.2922 - val_acc: 0.8834
Epoch 6/20
30/30 [==============================] - 2s 54ms/step - loss: 0.1379 - acc: 0.9538
- val_loss: 0.3263 - val_acc: 0.8717
Epoch 7/20
30/30 [==============================] - 2s 54ms/step - loss: 0.1181 - acc: 0.9619
- val_loss: 0.3017 - val_acc: 0.8847
Epoch 8/20
30/30 [==============================] - 1s 44ms/step - loss: 0.1039 - acc: 0.9660
- val_loss: 0.3342 - val_acc: 0.8745
Epoch 9/20
30/30 [==============================] - 1s 42ms/step - loss: 0.0909 - acc: 0.9723
- val_loss: 0.3361 - val_acc: 0.8815
Epoch 10/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0732 - acc: 0.9797
- val_loss: 0.3531 - val_acc: 0.8798
Epoch 11/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0626 - acc: 0.9835
- val_loss: 0.3748 - val_acc: 0.8776
Epoch 12/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0591 - acc: 0.9834
- val_loss: 0.4069 - val_acc: 0.8703
Epoch 13/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0457 - acc: 0.9899
- val_loss: 0.5102 - val_acc: 0.8546
Epoch 14/20
30/30 [==============================] - 1s 22ms/step - loss: 0.0399 - acc: 0.9899
- val_loss: 0.4520 - val_acc: 0.8699
Epoch 15/20
30/30 [==============================] - 1s 22ms/step - loss: 0.0354 - acc: 0.9923
- val_loss: 0.4677 - val_acc: 0.8726
Epoch 16/20
30/30 [==============================] - 1s 26ms/step - loss: 0.0288 - acc: 0.9943
- val_loss: 0.4920 - val_acc: 0.8720
Epoch 17/20
30/30 [==============================] - 1s 27ms/step - loss: 0.0255 - acc: 0.9956
- val_loss: 0.5322 - val_acc: 0.8682
Epoch 18/20
30/30 [==============================] - 1s 26ms/step - loss: 0.0216 - acc: 0.9963
- val_loss: 0.5482 - val_acc: 0.8692
Epoch 19/20
30/30 [==============================] - 1s 29ms/step - loss: 0.0172 - acc: 0.9980
- val_loss: 0.5997 - val_acc: 0.8623
Epoch 20/20
30/30 [==============================] - 1s 26ms/step - loss: 0.0190 - acc: 0.9959
```

```
- val_loss: 0.5954 - val_acc: 0.8679
```

In [ ]:
```
# Plotting training and validation loss
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

# Create axis range based on epochs
x_axis = [i for i in range(1, epochs+1)]

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

plt.show()
```

In [ ]:
```
# Plot training and validation accuracy
acc_values = history_dict['acc']
val_acc_values = history_dict['val_acc']

plt.plot(x_axis, acc_values,'bo', label='Training Acc')
plt.plot(x_axis, val_acc_values, 'b', label='Validation Acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



In [ ]:
```
# define new model
model=Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

In [ ]:
```
# Model training
model.compile(optimizer =  'rmsprop',
              loss = 'binary_crossentropy',
              metrics =['acc'])

model.fit(x_train,y_train, epochs=4, batch_size=512)

# Results
results = model.evaluate(x_test,y_test)
results
```

```
Epoch 1/4
49/49 [==============================] - 1s 18ms/step - loss: 0.4863 - acc: 0.8137
Epoch 2/4
49/49 [==============================] - 1s 15ms/step - loss: 0.2911 - acc: 0.8993
Epoch 3/4
49/49 [==============================] - 1s 14ms/step - loss: 0.2300 - acc: 0.9174
Epoch 4/4
49/49 [==============================] - 1s 14ms/step - loss: 0.1936 - acc: 0.9297
782/782 [==============================] - 2s 2ms/step - loss: 0.2922 - acc: 0.8832
```

Out[ ]:
```
[0.2922216057777405, 0.8831999897956848]
```

In [ ]:
```
#!jupyter nbconvert --to html /content/Assignment5_1.ipynb
```

```
[NbConvertApp] Converting notebook /content/Assignment5_1.ipynb to html
[NbConvertApp] Writing 672554 bytes to /content/Assignment5_1.html
```

Holly Figueroa

DSC650 Week 5

Book Exercise 3.5

A Single-Label, Multiclass Classification of Newswires

In [ ]:
```
# Loading dataset
from keras.datasets import reuters
import numpy as np

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

len(train_data)
len(test_data)
```

Out[ ]:
```
2246
```

In [ ]:
```
# How to decode newswires back to text
word_index = reuters.get_word_index()
reverse_word_index = dict([(value, key) for (key,value) in word_index.items()])
decoded_newswire= ' '.join([reverse_word_index.get(i-3,'?') for i in train_data[0]]
```

In [ ]:
```
decoded_newswire
```

Out[ ]:
```
'? ? ? said as a result of its december acquisition of space co it expects earnings
per share in 1987 of 1 15 to 1 30 dlrs per share up from 70 cts in 1986 the company
said pretax net should rise to nine to 10 mln dlrs from six mln dlrs in 1986 and re
ntal operation revenues to 19 to 22 mln dlrs from 12 5 mln dlrs it said cash flow p
er share this year should be 2 50 to three dlrs reuter 3'
```

In [ ]:
```
# Encoding the data
# Encode the integer sequences into a binary matrix
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

In [ ]:
```
print( x_train.shape, x_test.shape)
```

```
(8982, 10000) (2246, 10000)
```

In [ ]:
```
from keras.utils import to_categorical

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

In [ ]:
```
# Model Definition

from keras import models
from keras import layers

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'))
```

In [ ]:
```
# Compileing the model
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

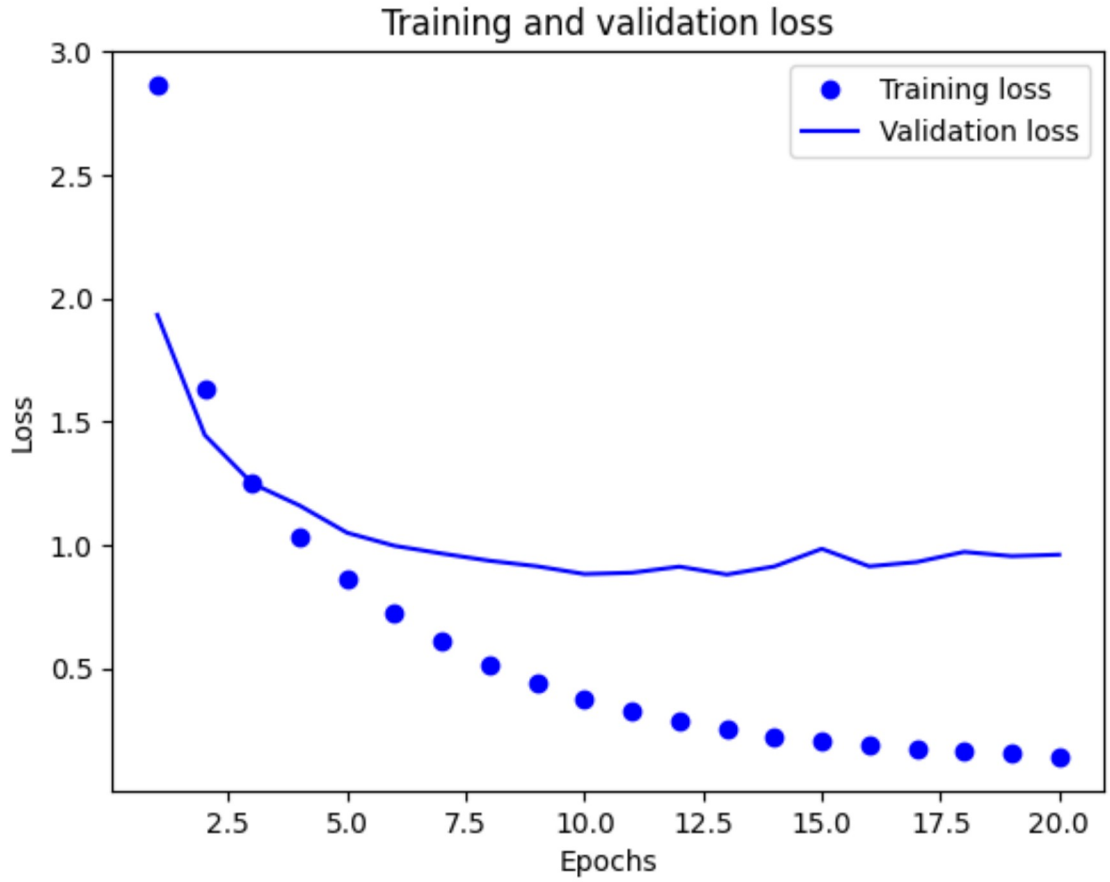
In [ ]:
```
# Setting asside a validation set
x_val = x_train[:1000]
partial_x_train = x_train[1000:]

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

In [ ]:
```
# Training the model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
16/16 [==============================] - 7s 249ms/step - loss: 2.8657 - accuracy:
0.4681 - val_loss: 1.9331 - val_accuracy: 0.6110
Epoch 2/20
16/16 [==============================] - 2s 127ms/step - loss: 1.6351 - accuracy:
0.6696 - val_loss: 1.4459 - val_accuracy: 0.6960
Epoch 3/20
16/16 [==============================] - 2s 123ms/step - loss: 1.2554 - accuracy:
0.7362 - val_loss: 1.2498 - val_accuracy: 0.7380
Epoch 4/20
16/16 [==============================] - 1s 69ms/step - loss: 1.0359 - accuracy: 0.
7816 - val_loss: 1.1600 - val_accuracy: 0.7520
Epoch 5/20
16/16 [==============================] - 1s 75ms/step - loss: 0.8623 - accuracy: 0.
8178 - val_loss: 1.0496 - val_accuracy: 0.7770
Epoch 6/20
16/16 [==============================] - 2s 94ms/step - loss: 0.7235 - accuracy: 0.
8433 - val_loss: 0.9968 - val_accuracy: 0.7890
Epoch 7/20
16/16 [==============================] - 2s 97ms/step - loss: 0.6088 - accuracy: 0.
8721 - val_loss: 0.9654 - val_accuracy: 0.7900
Epoch 8/20
16/16 [==============================] - 1s 62ms/step - loss: 0.5153 - accuracy: 0.
8938 - val_loss: 0.9364 - val_accuracy: 0.8150
Epoch 9/20
16/16 [==============================] - 1s 60ms/step - loss: 0.4398 - accuracy: 0.
9089 - val_loss: 0.9139 - val_accuracy: 0.8100
Epoch 10/20
16/16 [==============================] - 1s 68ms/step - loss: 0.3771 - accuracy: 0.
9228 - val_loss: 0.8825 - val_accuracy: 0.8170
Epoch 11/20
16/16 [==============================] - 2s 102ms/step - loss: 0.3257 - accuracy:
0.9303 - val_loss: 0.8878 - val_accuracy: 0.8130
Epoch 12/20
16/16 [==============================] - 1s 54ms/step - loss: 0.2841 - accuracy: 0.
9392 - val_loss: 0.9124 - val_accuracy: 0.8080
Epoch 13/20
16/16 [==============================] - 1s 54ms/step - loss: 0.2517 - accuracy: 0.
9437 - val_loss: 0.8806 - val_accuracy: 0.8090
Epoch 14/20
16/16 [==============================] - 1s 50ms/step - loss: 0.2206 - accuracy: 0.
9503 - val_loss: 0.9131 - val_accuracy: 0.8130
Epoch 15/20
16/16 [==============================] - 1s 51ms/step - loss: 0.2043 - accuracy: 0.
9514 - val_loss: 0.9847 - val_accuracy: 0.7970
Epoch 16/20
16/16 [==============================] - 1s 52ms/step - loss: 0.1879 - accuracy: 0.
9531 - val_loss: 0.9136 - val_accuracy: 0.8110
Epoch 17/20
16/16 [==============================] - 1s 54ms/step - loss: 0.1752 - accuracy: 0.
9524 - val_loss: 0.9311 - val_accuracy: 0.8060
Epoch 18/20
16/16 [==============================] - 1s 75ms/step - loss: 0.1643 - accuracy: 0.
9544 - val_loss: 0.9723 - val_accuracy: 0.7960
Epoch 19/20
16/16 [==============================] - 1s 91ms/step - loss: 0.1536 - accuracy: 0.
9563 - val_loss: 0.9550 - val_accuracy: 0.7960
Epoch 20/20
16/16 [==============================] - 1s 83ms/step - loss: 0.1415 - accuracy: 0.
```

```
9558 - val_loss: 0.9609 - val_accuracy: 0.8170
```

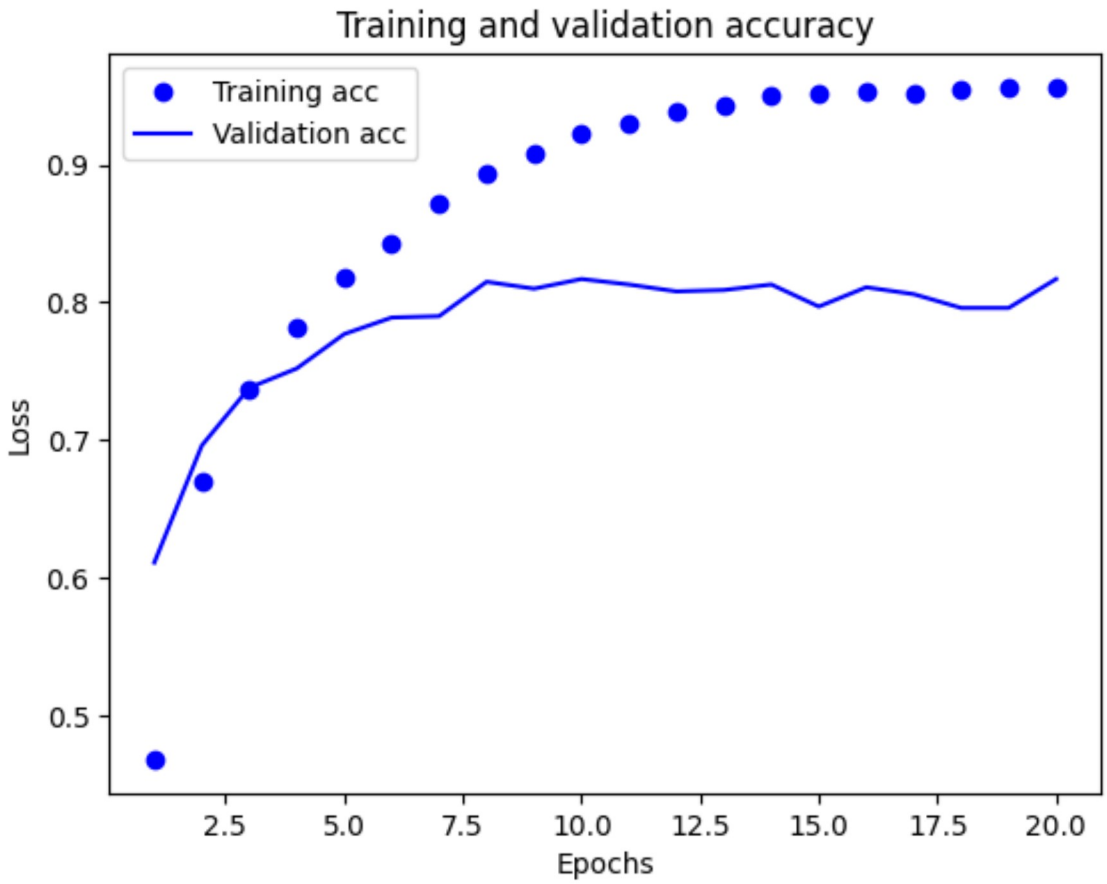
In [ ]:
```python
# Plotting the training and validation loss
import matplotlib.pyplot as plt

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()
```



In [ ]:
```python
history_dict = history.history
print(history_dict.keys())
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [ ]:
```
# Plotting the training and validation accuracy
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()
```



In [ ]:
```
# Retraining the model
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'))

# Compile
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
# Fit
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=9,
                    batch_size=512,
                    validation_data=(x_val, y_val))
# Results
results = model.evaluate(x_test, one_hot_test_labels)
```

```
Epoch 1/9
16/16 [==============================] - 2s 74ms/step - loss: 2.6363 - accuracy: 0.
5026 - val_loss: 1.7769 - val_accuracy: 0.6040
Epoch 2/9
16/16 [==============================] - 1s 56ms/step - loss: 1.5041 - accuracy: 0.
6689 - val_loss: 1.3409 - val_accuracy: 0.7070
Epoch 3/9
16/16 [==============================] - 1s 52ms/step - loss: 1.1421 - accuracy: 0.
7532 - val_loss: 1.1660 - val_accuracy: 0.7370
Epoch 4/9
16/16 [==============================] - 1s 56ms/step - loss: 0.9357 - accuracy: 0.
7938 - val_loss: 1.0682 - val_accuracy: 0.7600
Epoch 5/9
16/16 [==============================] - 1s 64ms/step - loss: 0.7757 - accuracy: 0.
8276 - val_loss: 0.9862 - val_accuracy: 0.7870
Epoch 6/9
16/16 [==============================] - 1s 64ms/step - loss: 0.6386 - accuracy: 0.
8621 - val_loss: 0.9379 - val_accuracy: 0.7930
Epoch 7/9
16/16 [==============================] - 1s 62ms/step - loss: 0.5390 - accuracy: 0.
8829 - val_loss: 0.9074 - val_accuracy: 0.8090
Epoch 8/9
16/16 [==============================] - 1s 75ms/step - loss: 0.4495 - accuracy: 0.
9030 - val_loss: 0.9181 - val_accuracy: 0.7950
Epoch 9/9
16/16 [==============================] - 2s 98ms/step - loss: 0.3795 - accuracy: 0.
9197 - val_loss: 0.8690 - val_accuracy: 0.8160
71/71 [==============================] - 0s 6ms/step - loss: 0.9530 - accuracy: 0.7
823
```

In [ ]:
```
# Generating predictions
predictions = model.predict(x_test)

# see predict is as vector with 46 len
predictions[0].shape
```

```
71/71 [==============================] - 0s 4ms/step
```

Out[ ]:
```
(46,)
```

In [ ]:
```
# see coefficients sum to one
np.sum(predictions[0])
```

Out[ ]:
```
1.0000001
```

In [ ]:
```
# see predicted class
np.argmax(predictions[0])
```

Out[ ]:
```
3
```

In [ ]:
```
# Experiment with dense layers
model = models.Sequential()
model.add(layers.Dense(64, activation = 'relu', input_shape=(10000, )))
model.add(layers.Dense(32, activation = 'relu'))
model.add(layers.Dense(46, activation = 'softmax'))

# Compile
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
# Fit
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=12,
                    batch_size=400,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/12
20/20 [==============================] - 2s 69ms/step - loss: 2.6825 - accuracy: 0.
5480 - val_loss: 1.8315 - val_accuracy: 0.6440
Epoch 2/12
20/20 [==============================] - 1s 51ms/step - loss: 1.5278 - accuracy: 0.
6877 - val_loss: 1.3754 - val_accuracy: 0.7080
Epoch 3/12
20/20 [==============================] - 1s 49ms/step - loss: 1.1600 - accuracy: 0.
7491 - val_loss: 1.2106 - val_accuracy: 0.7400
Epoch 4/12
20/20 [==============================] - 1s 51ms/step - loss: 0.9492 - accuracy: 0.
7914 - val_loss: 1.1178 - val_accuracy: 0.7400
Epoch 5/12
20/20 [==============================] - 1s 49ms/step - loss: 0.7874 - accuracy: 0.
8236 - val_loss: 1.0256 - val_accuracy: 0.7690
Epoch 6/12
20/20 [==============================] - 1s 50ms/step - loss: 0.6556 - accuracy: 0.
8572 - val_loss: 0.9566 - val_accuracy: 0.7890
Epoch 7/12
20/20 [==============================] - 1s 45ms/step - loss: 0.5462 - accuracy: 0.
8817 - val_loss: 0.9251 - val_accuracy: 0.7950
Epoch 8/12
20/20 [==============================] - 1s 61ms/step - loss: 0.4559 - accuracy: 0.
8984 - val_loss: 0.8945 - val_accuracy: 0.8060
Epoch 9/12
20/20 [==============================] - 2s 76ms/step - loss: 0.3854 - accuracy: 0.
9149 - val_loss: 0.8684 - val_accuracy: 0.8180
Epoch 10/12
20/20 [==============================] - 1s 71ms/step - loss: 0.3282 - accuracy: 0.
9296 - val_loss: 0.8743 - val_accuracy: 0.8160
Epoch 11/12
20/20 [==============================] - 1s 50ms/step - loss: 0.2826 - accuracy: 0.
9380 - val_loss: 0.8931 - val_accuracy: 0.8170
Epoch 12/12
20/20 [==============================] - 1s 50ms/step - loss: 0.2471 - accuracy: 0.
9446 - val_loss: 0.8876 - val_accuracy: 0.8210
```

In [ ]:
```
results = model.evaluate(x_test, one_hot_test_labels)
```

```
71/71 [==============================] - 0s 4ms/step - loss: 0.9653 - accuracy: 0.7
845
```

In [ ]:
```
#!jupyter nbconvert --to html /content/Assignment5_2.ipynb
```

Holly Figueroa

DSC650 Week 5

Book Exercise 3.6

A Regression to Predict Home Prices

In [2]:
```
# Loading Data
from keras.datasets import boston_housing

(train_data, train_targets), (test_data, test_targets)=boston_housing.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/b
oston_housing.npz
57026/57026 [==============================] - 0s 0us/step
```

In [3]:
```
# Check Shape
print(train_data.shape, test_data.shape)
```

```
(404, 13) (102, 13)
```

In [4]:
```
train_targets
```

Out[4]:
```
array([15.2, 42.3, 50. , 21.1, 17.7, 18.5, 11.3, 15.6, 15.6, 14.4, 12.1,
       17.9, 23.1, 19.9, 15.7,  8.8, 50. , 22.5, 24.1, 27.5, 10.9, 30.8,
       32.9, 24. , 18.5, 13.3, 22.9, 34.7, 16.6, 17.5, 22.3, 16.1, 14.9,
       23.1, 34.9, 25. , 13.9, 13.1, 20.4, 20. , 15.2, 24.7, 22.2, 16.7,
       12.7, 15.6, 18.4, 21. , 30.1, 15.1, 18.7,  9.6, 31.5, 24.8, 19.1,
       22. , 14.5, 11. , 32. , 29.4, 20.3, 24.4, 14.6, 19.5, 14.1, 14.3,
       15.6, 10.5,  6.3, 19.3, 19.3, 13.4, 36.4, 17.8, 13.5, 16.5,  8.3,
       14.3, 16. , 13.4, 28.6, 43.5, 20.2, 22. , 23. , 20.7, 12.5, 48.5,
       14.6, 13.4, 23.7, 50. , 21.7, 39.8, 38.7, 22.2, 34.9, 22.5, 31.1,
       28.7, 46. , 41.7, 21. , 26.6, 15. , 24.4, 13.3, 21.2, 11.7, 21.7,
       19.4, 50. , 22.8, 19.7, 24.7, 36.2, 14.2, 18.9, 18.3, 20.6, 24.6,
       18.2,  8.7, 44. , 10.4, 13.2, 21.2, 37. , 30.7, 22.9, 20. , 19.3,
       31.7, 32. , 23.1, 18.8, 10.9, 50. , 19.6,  5. , 14.4, 19.8, 13.8,
       19.6, 23.9, 24.5, 25. , 19.9, 17.2, 24.6, 13.5, 26.6, 21.4, 11.9,
       22.6, 19.6,  8.5, 23.7, 23.1, 22.4, 20.5, 23.6, 18.4, 35.2, 23.1,
       27.9, 20.6, 23.7, 28. , 13.6, 27.1, 23.6, 20.6, 18.2, 21.7, 17.1,
        8.4, 25.3, 13.8, 22.2, 18.4, 20.7, 31.6, 30.5, 20.3,  8.8, 19.2,
       19.4, 23.1, 23. , 14.8, 48.8, 22.6, 33.4, 21.1, 13.6, 32.2, 13.1,
       23.4, 18.9, 23.9, 11.8, 23.3, 22.8, 19.6, 16.7, 13.4, 22.2, 20.4,
       21.8, 26.4, 14.9, 24.1, 23.8, 12.3, 29.1, 21. , 19.5, 23.3, 23.8,
       17.8, 11.5, 21.7, 19.9, 25. , 33.4, 28.5, 21.4, 24.3, 27.5, 33.1,
       16.2, 23.3, 48.3, 22.9, 22.8, 13.1, 12.7, 22.6, 15. , 15.3, 10.5,
       24. , 18.5, 21.7, 19.5, 33.2, 23.2,  5. , 19.1, 12.7, 22.3, 10.2,
       13.9, 16.3, 17. , 20.1, 29.9, 17.2, 37.3, 45.4, 17.8, 23.2, 29. ,
       22. , 18. , 17.4, 34.6, 20.1, 25. , 15.6, 24.8, 28.2, 21.2, 21.4,
       23.8, 31. , 26.2, 17.4, 37.9, 17.5, 20. ,  8.3, 23.9,  8.4, 13.8,
        7.2, 11.7, 17.1, 21.6, 50. , 16.1, 20.4, 20.6, 21.4, 20.6, 36.5,
        8.5, 24.8, 10.8, 21.9, 17.3, 18.9, 36.2, 14.9, 18.2, 33.3, 21.8,
       19.7, 31.6, 24.8, 19.4, 22.8,  7.5, 44.8, 16.8, 18.7, 50. , 50. ,
       19.5, 20.1, 50. , 17.2, 20.8, 19.3, 41.3, 20.4, 20.5, 13.8, 16.5,
       23.9, 20.6, 31.5, 23.3, 16.8, 14. , 33.8, 36.1, 12.8, 18.3, 18.7,
       19.1, 29. , 30.1, 50. , 50. , 22. , 11.9, 37.6, 50. , 22.7, 20.8,
       23.5, 27.9, 50. , 19.3, 23.9, 22.6, 15.2, 21.7, 19.2, 43.8, 20.3,
       33.2, 19.9, 22.5, 32.7, 22. , 17.1, 19. , 15. , 16.1, 25.1, 23.7,
       28.7, 37.2, 22.6, 16.4, 25. , 29.8, 22.1, 17.4, 18.1, 30.3, 17.5,
       24.7, 12.6, 26.5, 28.7, 13.3, 10.4, 24.4, 23. , 20. , 17.8,  7. ,
       11.8, 24.4, 13.8, 19.4, 25.2, 19.4, 19.4, 29.1])
```

In [5]:
```
# Normalizing Data
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data/=std

test_data -= mean
test_data /= std
# Never normalize test data using the test data metrics
```

In [7]:
```
# Model Definition
from keras import models
from keras import layers

def build_model():
  model = models.Sequential()
  model.add(layers.Dense(64, activation = 'relu',
                         input_shape=(train_data.shape[1],)))
  model.add(layers.Dense(64, activation='relu'))
  model.add(layers.Dense(1))
  model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
  return model
```

In [8]:
```
# K-Fold Validation
import numpy as np

k=4
num_val_samples = len(train_data) // k
num_epochs = 100
all_scores = []

for i in range(k):
  print('processing fold #', i)
  val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
  val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]

  partial_train_data = np.concatenate(
    [train_data[:i * num_val_samples],
    train_data[(i + 1) * num_val_samples:]],
    axis=0)
  partial_train_targets = np.concatenate(
    [train_targets[:i * num_val_samples],
    train_targets[(i + 1) * num_val_samples:]],
    axis=0)

  model = build_model()
  model.fit(partial_train_data, partial_train_targets,
            epochs=num_epochs, batch_size=1, verbose=0)
  val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)
  all_scores.append(val_mae)
```

```
processing fold # 0
processing fold # 1
processing fold # 2
processing fold # 3
```

In [9]:
```
np.mean(all_scores)
```

Out[9]:
```
2.590188056230545
```

In [12]:
```
# Saving the validation logs at each fold
num_epochs = 500
all_mae_histories = []
for i in range(k):
  print('processing fold #', i)
  val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
  val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]

  partial_train_data = np.concatenate(
    [train_data[:i * num_val_samples],
    train_data[(i + 1) * num_val_samples:]],
    axis=0)

  partial_train_targets = np.concatenate(
    [train_targets[:i * num_val_samples],
    train_targets[(i + 1) * num_val_samples:]],
    axis=0)
  model = build_model()
  history = model.fit(partial_train_data, partial_train_targets,
                      validation_data=(val_data, val_targets),
                      epochs=num_epochs, batch_size=1, verbose=0)
  mae_history = history.history['val_mae']
  all_mae_histories.append(mae_history)
```

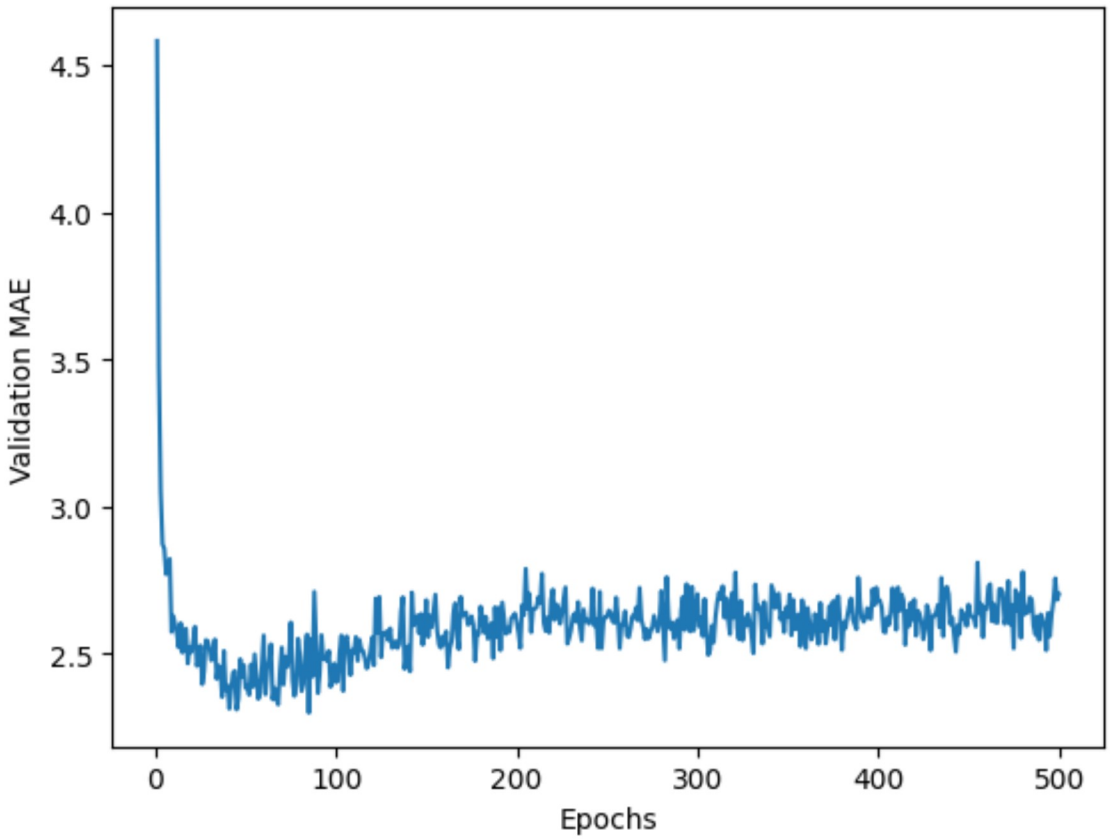
```
processing fold # 0
processing fold # 1
processing fold # 2
processing fold # 3
```

In [11]:
```
# Correct example code key reference
history_dict = history.history
history_dict.keys()
```

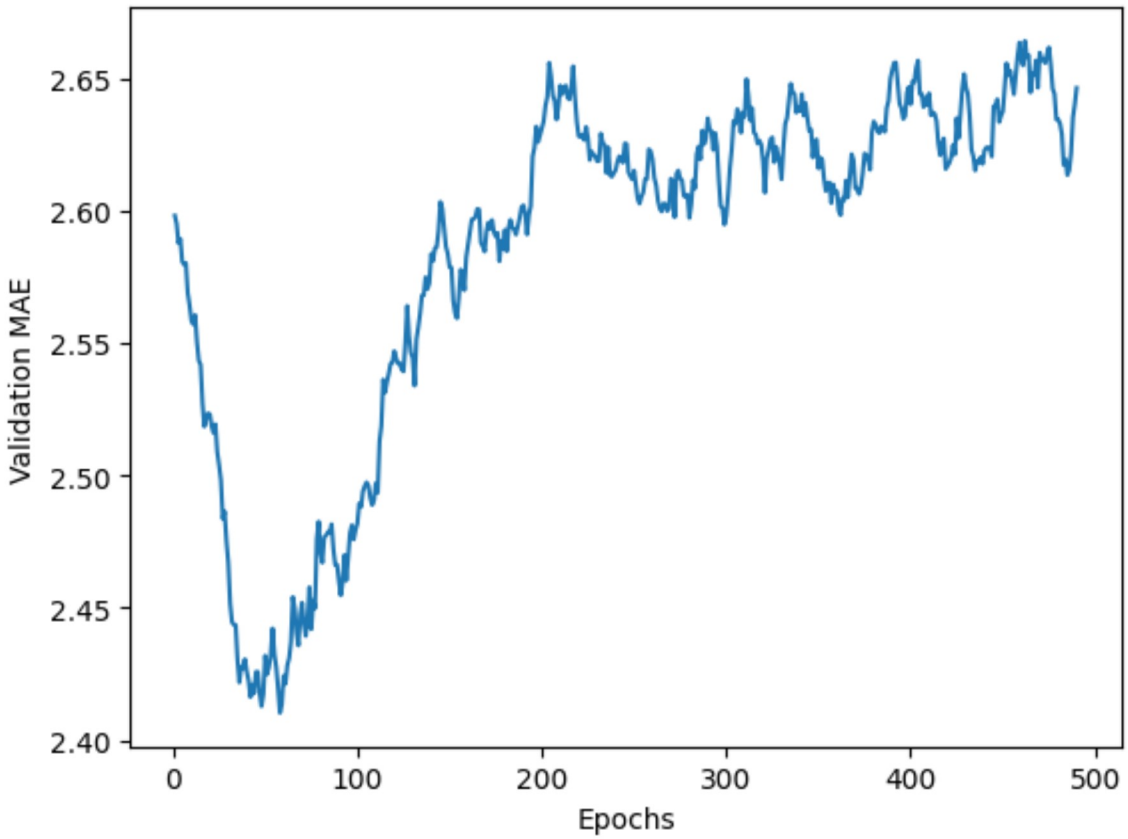
Out[11]:
```
dict_keys(['loss', 'mae', 'val_loss', 'val_mae'])
```

In [13]:
```
# Building the istory of successive mean K-fole validation scores
average_mae_history = [
  np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)]
```

In [14]:
```
# Plotting validation scores
import matplotlib.pyplot as plt

plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```



In [16]:
```
# Plotting validation scores, excluding the first 10 data points
def smooth_curve(points, factor=0.9):
  smoothed_points = []
  for point in points:
    if smoothed_points:
      previous = smoothed_points[-1]
      smoothed_points.append(previous * factor + point * (1 - factor))
    else:
      smoothed_points.append(point)
  return smoothed_points

smooth_mae_history = smooth_curve(average_mae_history[10:])

plt.plot(range(1, len(smooth_mae_history) + 1), smooth_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```



In [17]:
```
# Training the final model
model = build_model()
model.fit(train_data, train_targets,
          epochs=80, batch_size=16, verbose=0)
test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)
```

```
4/4 [==============================] - 0s 8ms/step - loss: 18.3617 - mae: 2.6747
```

In [18]:
```
test_mae_score
```

Out[18]:
```
2.6747214794158936
```

In [ ]:

In [ ]: