Assignment 5.1

author: Rachel Nelson

class: DSC650

Assignment 5.1

Implement the movie review classifier found in section 3.4 of Deep Learning with Python.

127
```
# 3.1 loading the IMDB data set
from keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
train_data[0]
train_labels[0]
max([max(sequence) for sequence in train_data])

word_index = imdb.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
decoded_review = ' '.join([reverse_word_index.get(i - 3, '?') for i in train_data[0]])
decoded_review

word_index = imdb.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
decoded_review = ' '.join([reverse_word_index.get(i - 3, '?') for i in train_data[0]])
decoded_review
```

127
```
"? this film was just brilliant casting location scenery story direction everyone's really suited the par
```

128
```
# 3.2 encoding the integer sequences into a binary matrix
import numpy as np
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)
x_train[0]
```

128
```
array([0., 1., 1., ..., 0., 0., 0.])
```

129
```
# vectorize labels
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

130
```
# 3.3 the model definition
from keras import models
from keras import layers
```

```
model = models.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'))
```

131
```
# 3.4 compilling the model
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

132
```
# 3.5 configuring the optimizer

from keras import optimizers
from tensorflow.keras import optimizers

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
D:\College\venv\lib\site-packages\keras\optimizer_v2\rmsprop.py:130: UserWarning: The `lr` argument is de
  super(RMSprop, self).__init__(name, **kwargs)
```

133
```
# 3.6 using custom losses and metrics
from tensorflow.keras import optimizers
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
loss='binary_crossentropy',
metrics=['accuracy'])
from keras import losses
from keras import metrics
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
loss=losses.binary_crossentropy,
metrics=[metrics.binary_accuracy])
```

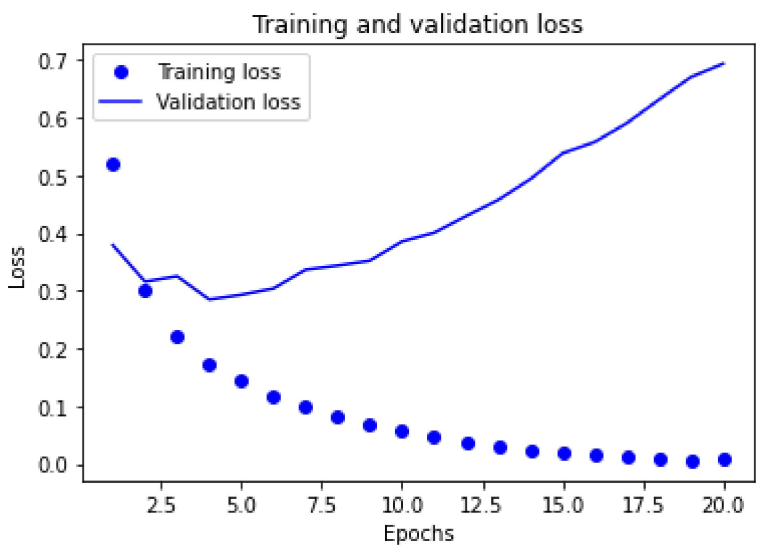
134
```
# 3.7 Setting aside a validation set
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

135
```
# 3.8 Training the model

history = model.fit(partial_x_train,partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))

history_dict = history.history
history_dict.keys()
print(history_dict)
```

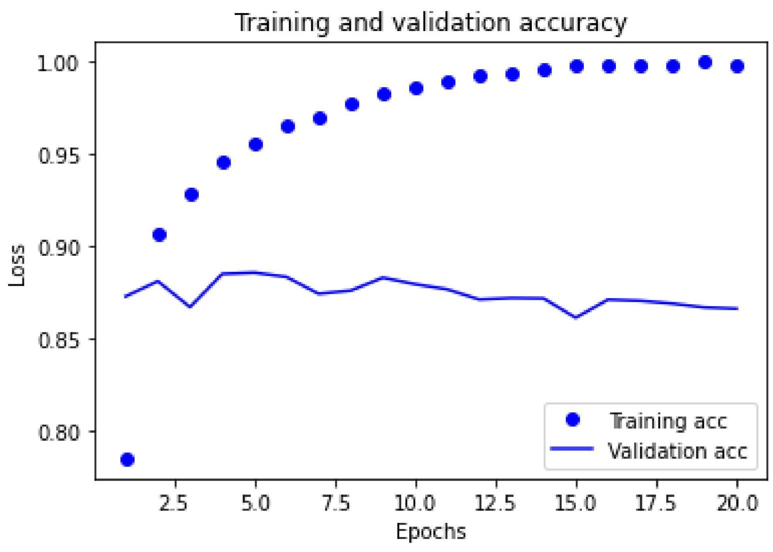
```
Epoch 1/20
30/30 [==============================] - 1s 21ms/step - loss: 0.5197 - binary_accuracy: 0.7850 - val_loss
Epoch 2/20
30/30 [==============================] - 0s 12ms/step - loss: 0.3014 - binary_accuracy: 0.9066 - val_loss
Epoch 3/20
30/30 [==============================] - 0s 12ms/step - loss: 0.2198 - binary_accuracy: 0.9283 - val_loss
Epoch 4/20
30/30 [==============================] - 0s 11ms/step - loss: 0.1728 - binary_accuracy: 0.9456 - val_loss
```

```
Epoch 5/20
30/30 [==============================] - 0s 12ms/step - loss: 0.1430 - binary_accuracy: 0.9558 - val_loss
Epoch 6/20
30/30 [==============================] - 0s 11ms/step - loss: 0.1152 - binary_accuracy: 0.9657 - val_loss
Epoch 7/20
30/30 [==============================] - 0s 11ms/step - loss: 0.1009 - binary_accuracy: 0.9697 - val_loss
Epoch 8/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0821 - binary_accuracy: 0.9773 - val_loss
Epoch 9/20
30/30 [==============================] - 0s 10ms/step - loss: 0.0676 - binary_accuracy: 0.9827 - val_loss
Epoch 10/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0570 - binary_accuracy: 0.9854 - val_loss
Epoch 11/20
30/30 [==============================] - 0s 10ms/step - loss: 0.0473 - binary_accuracy: 0.9888 - val_loss
Epoch 12/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0376 - binary_accuracy: 0.9929 - val_loss
Epoch 13/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0310 - binary_accuracy: 0.9939 - val_loss
Epoch 14/20
30/30 [==============================] - 0s 10ms/step - loss: 0.0240 - binary_accuracy: 0.9961 - val_loss
Epoch 15/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0193 - binary_accuracy: 0.9977 - val_loss
Epoch 16/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0156 - binary_accuracy: 0.9979 - val_loss
Epoch 17/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0132 - binary_accuracy: 0.9978 - val_loss
Epoch 18/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0098 - binary_accuracy: 0.9984 - val_loss
Epoch 19/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0056 - binary_accuracy: 0.9998 - val_loss
Epoch 20/20
30/30 [==============================] - 0s 10ms/step - loss: 0.0082 - binary_accuracy: 0.9980 - val_loss
{'loss': [0.5197033286094666, 0.3013951778411865, 0.2197771519422531, 0.17277181148529053, 0.142968386411
```

136
```
# 3.9 Plotting the training and validation loss
import matplotlib.pyplot as plt
acc = history.history['binary_accuracy']
val_acc = history.history['val_binary_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 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()
```



137
```
#  3.10 Plotting the training and validation accuracy

plt.clf()
acc = history.history['binary_accuracy']
val_acc = history.history['val_binary_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()
```



138
```
# 3.11 Retraining a model from scratch
model = models.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'))
model.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['accuracy'])
model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
```

```
49/49 [==============================] - 1s 7ms/step - loss: 0.4439 - accuracy: 0.8202�������������������
Epoch 2/4
49/49 [==============================] - 0s 7ms/step - loss: 0.2576 - accuracy: 0.9103�������������������
Epoch 3/4
49/49 [==============================] - 0s 7ms/step - loss: 0.1991 - accuracy: 0.9293�������������������
Epoch 4/4
49/49 [==============================] - 0s 7ms/step - loss: 0.1661 - accuracy: 0.9411�������������������
782/782 [==============================] - 1s 1ms/step - loss: 0.3252 - accuracy: 0.8722�����������������
```