Assignment 5.1

July 6, 2021

1 Assignment 5.1

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

[1]:
```
from tensorflow.keras.datasets import imdb
```

[2]:
```
# import data
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(
    num_words=10000)
```

```
<__array_function__ internals>:5: VisibleDeprecationWarning: Creating an ndarray
from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or
ndarrays with different lengths or shapes) is deprecated. If you meant to do
this, you must specify 'dtype=object' when creating the ndarray
/opt/conda/lib/python3.8/site-
packages/tensorflow/python/keras/datasets/imdb.py:159:
VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
/opt/conda/lib/python3.8/site-
packages/tensorflow/python/keras/datasets/imdb.py:160:
VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

[3]:
```
train_data[0]
```

[3]:
```
[1,
 14,
 22,
 16,
 43,
 530,
```

973,
1622,
1385,
65,
458,
4468,
66,
3941,
4,
173,
36,
256,
5,
25,
100,
43,
838,
112,
50,
670,
2,
9,
35,
480,
284,
5,
150,
4,
172,
112,
167,
2,
336,
385,
39,
4,
172,
4536,
1111,
17,
546,
38,
13,
447,
4,
192,
50,

16,
6,
147,
2025,
19,
14,
22,
4,
1920,
4613,
469,
4,
22,
71,
87,
12,
16,
43,
530,
38,
76,
15,
13,
1247,
4,
22,
17,
515,
17,
12,
16,
626,
18,
2,
5,
62,
386,
12,
8,
316,
8,
106,
5,
4,
2223,
5244,
16,

480,
66,
3785,
33,
4,
130,
12,
16,
38,
619,
5,
25,
124,
51,
36,
135,
48,
25,
1415,
33,
6,
22,
12,
215,
28,
77,
52,
5,
14,
407,
16,
82,
2,
8,
4,
107,
117,
5952,
15,
256,
4,
2,
7,
3766,
5,
723,
36,

```
71,
43,
530,
476,
26,
400,
317,
46,
7,
4,
2,
1029,
13,
104,
88,
4,
381,
15,
297,
98,
32,
2071,
56,
26,
141,
6,
194,
7486,
18,
4,
226,
22,
21,
134,
476,
26,
480,
5,
144,
30,
5535,
18,
51,
36,
28,
224,
92,
```

```
 25,
 104,
 4,
 226,
 65,
 16,
 38,
 1334,
 88,
 12,
 16,
 283,
 5,
 16,
 4472,
 113,
 103,
 32,
 15,
 16,
 5345,
 19,
 178,
 32]
```

[4]:
```
train_labels[0]
```

[4]:
```
1
```

[5]:
```
# No word should exceed 10,000
max([max(sequence) for sequence in train_data])
```

[5]:
```
9999
```

[18]:
```
# decoding reviews back to their text

# dictionary mapping words to an integer index
word_index = imdb.get_word_index()

# reverse it by mapping integer indices to words
reverse_word_index = dict(
    [(value, key) for (key, value) in word_index.items()])

# Decode the review - indices are offset by 3 because 0, 1, 2 are reserved␣
 ↪indices for "padding", "start of sequence" and "unknown"
decoded_review = " ".join(
    [reverse_word_index.get(i - 3, "?") for i in train_data[0]])
```

```
decoded_review
```

[18]:
```
"? this film was just brilliant casting location scenery story direction
everyone's really suited the part they played and you could just imagine being
there robert ? is an amazing actor and now the same being director ? father came
from the same scottish island as myself so i loved the fact there was a real
connection with this film the witty remarks throughout the film were great it
was just brilliant so much that i bought the film as soon as it was released for
? and would recommend it to everyone to watch and the fly fishing was amazing
really cried at the end it was so sad and you know what they say if you cry at a
film it must have been good and this definitely was also ? to the two little
boy's that played the ? of norman and paul they were just brilliant children are
often left out of the ? list i think because the stars that play them all grown
up are such a big profile for the whole film but these children are amazing and
should be praised for what they have done don't you think the whole story was so
lovely because it was true and was someone's life after all that was shared with
us all"
```

[19]:
```
# prepare the data
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)
```

[20]:
```
x_train[0]
```

[20]:
```
array([0., 1., 1., …, 0., 0., 0.])
```

[21]:
```
# vectorize labels so they can be fed into a neural network
y_train = np.asarray(train_labels).astype("float32")
y_test = np.asarray(test_labels).astype("float32")
```

[22]:
```
# Keras implementation
from keras import models
from keras import layers
```

[30]:
```
# build the model
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'))
```

[31]:
```
# compiling the model
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

[32]:
```
from keras import optimizers
```

[33]:
```
# configuring the optimizer
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

[34]:
```
# using custom losses & metrics
from keras import losses
from keras import metrics

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

[35]:
```
# validating the approach
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

[36]:
```
# train the model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

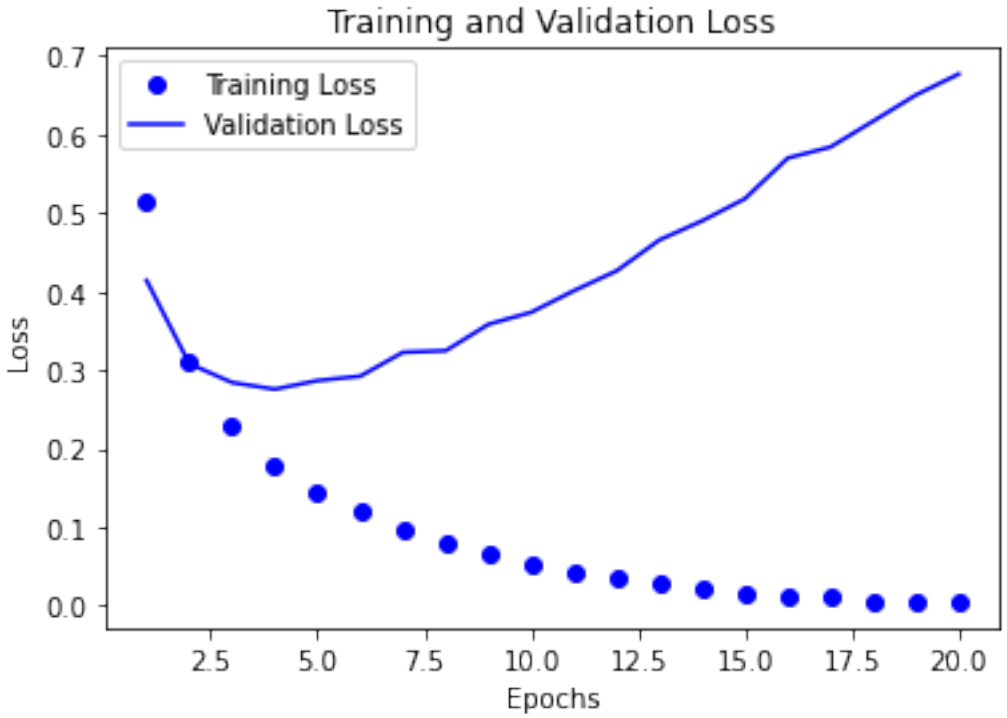
```
Epoch 1/20
30/30 [==============================] - 2s 44ms/step - loss: 0.5901 -
binary_accuracy: 0.7078 - val_loss: 0.4143 - val_binary_accuracy: 0.8486
Epoch 2/20
30/30 [==============================] - 0s 13ms/step - loss: 0.3347 -
binary_accuracy: 0.8967 - val_loss: 0.3083 - val_binary_accuracy: 0.8873
Epoch 3/20
30/30 [==============================] - 0s 14ms/step - loss: 0.2310 -
binary_accuracy: 0.9298 - val_loss: 0.2843 - val_binary_accuracy: 0.8884
Epoch 4/20
30/30 [==============================] - 1s 34ms/step - loss: 0.1774 -
binary_accuracy: 0.9495 - val_loss: 0.2757 - val_binary_accuracy: 0.8906
Epoch 5/20
30/30 [==============================] - 1s 38ms/step - loss: 0.1403 -
```

```
binary_accuracy: 0.9585 - val_loss: 0.2867 - val_binary_accuracy: 0.8874
Epoch 6/20
30/30 [==============================] - 1s 23ms/step - loss: 0.1185 -
binary_accuracy: 0.9665 - val_loss: 0.2923 - val_binary_accuracy: 0.8863
Epoch 7/20
30/30 [==============================] - 1s 24ms/step - loss: 0.0951 -
binary_accuracy: 0.9747 - val_loss: 0.3228 - val_binary_accuracy: 0.8778
Epoch 8/20
30/30 [==============================] - 1s 19ms/step - loss: 0.0803 -
binary_accuracy: 0.9804 - val_loss: 0.3248 - val_binary_accuracy: 0.8820
Epoch 9/20
30/30 [==============================] - 0s 15ms/step - loss: 0.0640 -
binary_accuracy: 0.9866 - val_loss: 0.3584 - val_binary_accuracy: 0.8747
Epoch 10/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0490 -
binary_accuracy: 0.9900 - val_loss: 0.3737 - val_binary_accuracy: 0.8793
Epoch 11/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0393 -
binary_accuracy: 0.9928 - val_loss: 0.4012 - val_binary_accuracy: 0.8754
Epoch 12/20
30/30 [==============================] - 0s 12ms/step - loss: 0.0319 -
binary_accuracy: 0.9951 - val_loss: 0.4265 - val_binary_accuracy: 0.8743
Epoch 13/20
30/30 [==============================] - 0s 10ms/step - loss: 0.0258 -
binary_accuracy: 0.9960 - val_loss: 0.4658 - val_binary_accuracy: 0.8760
Epoch 14/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0183 -
binary_accuracy: 0.9977 - val_loss: 0.4901 - val_binary_accuracy: 0.8743
Epoch 15/20
30/30 [==============================] - 0s 12ms/step - loss: 0.0134 -
binary_accuracy: 0.9990 - val_loss: 0.5183 - val_binary_accuracy: 0.8725
Epoch 16/20
30/30 [==============================] - 0s 12ms/step - loss: 0.0094 -
binary_accuracy: 0.9996 - val_loss: 0.5700 - val_binary_accuracy: 0.8707
Epoch 17/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0097 -
binary_accuracy: 0.9988 - val_loss: 0.5840 - val_binary_accuracy: 0.8698
Epoch 18/20
30/30 [==============================] - 0s 12ms/step - loss: 0.0052 -
binary_accuracy: 0.9998 - val_loss: 0.6170 - val_binary_accuracy: 0.8690
Epoch 19/20
30/30 [==============================] - 0s 12ms/step - loss: 0.0049 -
binary_accuracy: 0.9995 - val_loss: 0.6503 - val_binary_accuracy: 0.8670
Epoch 20/20
30/30 [==============================] - 0s 11ms/step - loss: 0.0032 -
binary_accuracy: 0.9998 - val_loss: 0.6769 - val_binary_accuracy: 0.8682
```

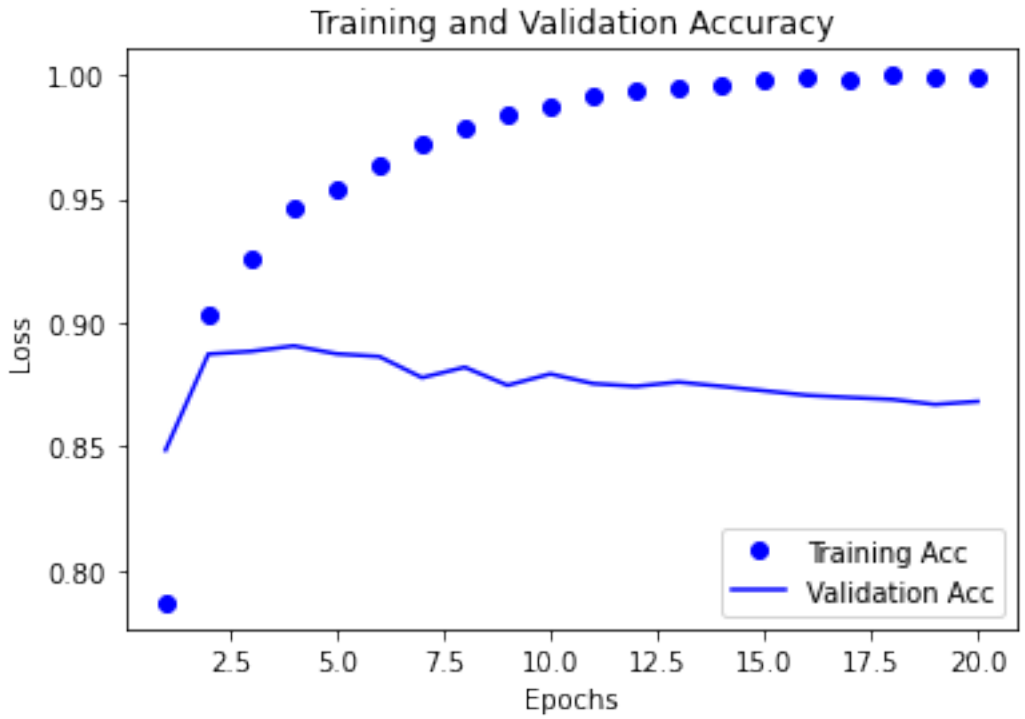
[39]:
```
history_dict = history.history
history_dict.keys()
```

[39]:
```
dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
```

[42]:
```
# plot training & validation loss
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
acc = history_dict["binary_accuracy"]

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

plt.plot(epochs, loss_values, 'bo', label='Training Loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



[44]:
```
# plot training & validation accuracy
plt.clf() # clears figure
acc_values = history_dict['binary_accuracy']
val_acc_values = history_dict['val_binary_accuracy']

plt.plot(epochs, acc, 'bo', label='Training Acc')
plt.plot(epochs, val_acc_values, 'b', label='Validation Acc')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



[47]:
```
# 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 8ms/step - loss: 0.5577 - accuracy:
0.7357
Epoch 2/4
49/49 [==============================] - 0s 8ms/step - loss: 0.2742 - accuracy:
0.9091
Epoch 3/4
49/49 [==============================] - 0s 8ms/step - loss: 0.1972 - accuracy:
0.9314
Epoch 4/4
49/49 [==============================] - 0s 6ms/step - loss: 0.1591 - accuracy:
0.9449
782/782 [==============================] - 1s 2ms/step - loss: 0.3313 -
accuracy: 0.8716
```

[48]:
```
# view results
results
```

[48]:
```
[0.3312642574310303, 0.8715599775314331]
```

[49]:
```
# predict with model
model.predict(x_test)
```

[49]:
```
array([[0.09626722],
       [0.9975097 ],
       [0.4017473 ],
       …,
       [0.07191551],
       [0.03973332],
       [0.40638703]], dtype=float32)
```

Reference: https://github.com/fchollet/deep-learning-with-python-notebooks

page xviii from book

[ ]: