Imdb

January 18, 2021

[5]:

```
import keras
```

[ ]:

```

```

[6]:

```
from keras.datasets import imdb

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

[7]:

```
train_data[0]
```

[7]:

```
[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]
```

[8]:

```
train_labels[0]
```

[8]:

```
1
```

[9]:

```
max([max(sequence) for sequence in train_data])
```

[9]:

```
9999
```

[10]:

```
# word_index is a dictionary mapping words to an integer index
word_index = imdb.get_word_index()
# We reverse it, mapping integer indices to words
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
# We decode the review; note that our indices were offset by 3
# because 0, 1 and 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]])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/imdb_word_index.json
1646592/1641221 [==============================] - 0s 0us/step
```

[11]:

```
decoded_review
```

[11]:

```
"? 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"
```

[12]:

```
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

# Our vectorized training data
x_train = vectorize_sequences(train_data)
# Our vectorized test data
x_test = vectorize_sequences(test_data)
```

[13]:

```
x_train[0]
```

[13]:

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

[14]:

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

[15]:

```
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'))
```

[16]:

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

[17]:

```
from keras import optimizers

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

[18]:

```
from keras import losses
from keras import metrics

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

[19]:

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
```

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

[20]:

```
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 [==============================] - 1s 43ms/step - loss: 0.5593 -
binary_accuracy: 0.7430 - val_loss: 0.4304 - val_binary_accuracy: 0.8580
Epoch 2/20
30/30 [==============================] - 1s 28ms/step - loss: 0.3310 -
binary_accuracy: 0.9001 - val_loss: 0.3257 - val_binary_accuracy: 0.8814
Epoch 3/20
30/30 [==============================] - 1s 30ms/step - loss: 0.2384 -
binary_accuracy: 0.9264 - val_loss: 0.2836 - val_binary_accuracy: 0.8887
Epoch 4/20
30/30 [==============================] - 1s 33ms/step - loss: 0.1894 -
binary_accuracy: 0.9390 - val_loss: 0.2773 - val_binary_accuracy: 0.8889
Epoch 5/20
30/30 [==============================] - 1s 29ms/step - loss: 0.1543 -
binary_accuracy: 0.9511 - val_loss: 0.2781 - val_binary_accuracy: 0.8885
Epoch 6/20
30/30 [==============================] - 1s 29ms/step - loss: 0.1261 -
binary_accuracy: 0.9619 - val_loss: 0.2883 - val_binary_accuracy: 0.8872
Epoch 7/20
30/30 [==============================] - 1s 27ms/step - loss: 0.1057 -
binary_accuracy: 0.9695 - val_loss: 0.3391 - val_binary_accuracy: 0.8750
Epoch 8/20
30/30 [==============================] - 1s 27ms/step - loss: 0.0875 -
binary_accuracy: 0.9760 - val_loss: 0.3220 - val_binary_accuracy: 0.8840
Epoch 9/20
30/30 [==============================] - 1s 33ms/step - loss: 0.0718 -
binary_accuracy: 0.9822 - val_loss: 0.3427 - val_binary_accuracy: 0.8785
Epoch 10/20
30/30 [==============================] - 1s 45ms/step - loss: 0.0615 -
binary_accuracy: 0.9841 - val_loss: 0.3641 - val_binary_accuracy: 0.8805
Epoch 11/20
30/30 [==============================] - 1s 34ms/step - loss: 0.0523 -
binary_accuracy: 0.9882 - val_loss: 0.3837 - val_binary_accuracy: 0.8794
Epoch 12/20
30/30 [==============================] - 1s 28ms/step - loss: 0.0411 -
binary_accuracy: 0.9905 - val_loss: 0.4088 - val_binary_accuracy: 0.8767
Epoch 13/20
```

```
30/30 [==============================] - 1s 30ms/step - loss: 0.0335 -
binary_accuracy: 0.9932 - val_loss: 0.4428 - val_binary_accuracy: 0.8753
Epoch 14/20
30/30 [==============================] - 1s 27ms/step - loss: 0.0271 -
binary_accuracy: 0.9949 - val_loss: 0.4705 - val_binary_accuracy: 0.8736
Epoch 15/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0222 -
binary_accuracy: 0.9959 - val_loss: 0.4957 - val_binary_accuracy: 0.8717
Epoch 16/20
30/30 [==============================] - 1s 36ms/step - loss: 0.0158 -
binary_accuracy: 0.9983 - val_loss: 0.5317 - val_binary_accuracy: 0.8673
Epoch 17/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0129 -
binary_accuracy: 0.9984 - val_loss: 0.5690 - val_binary_accuracy: 0.8695
Epoch 18/20
30/30 [==============================] - 1s 30ms/step - loss: 0.0121 -
binary_accuracy: 0.9979 - val_loss: 0.6032 - val_binary_accuracy: 0.8667
Epoch 19/20
30/30 [==============================] - 1s 28ms/step - loss: 0.0079 -
binary_accuracy: 0.9993 - val_loss: 0.6310 - val_binary_accuracy: 0.8660
Epoch 20/20
30/30 [==============================] - 1s 32ms/step - loss: 0.0050 -
binary_accuracy: 0.9997 - val_loss: 0.7694 - val_binary_accuracy: 0.8570
```

[22]:

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

[22]:

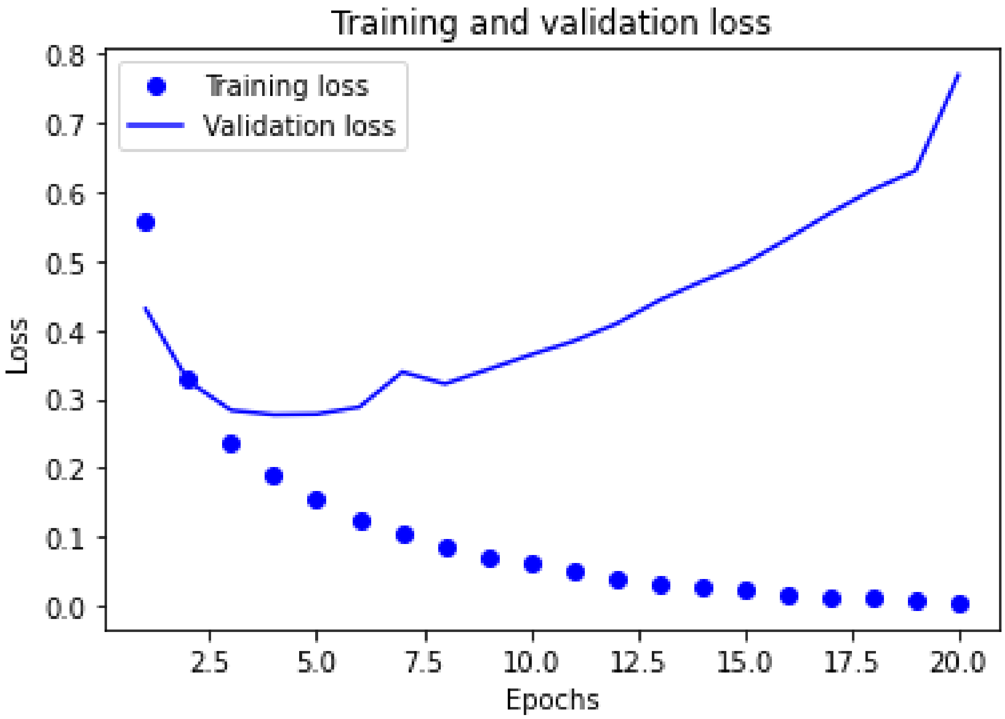
```
dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
```

[ ]:

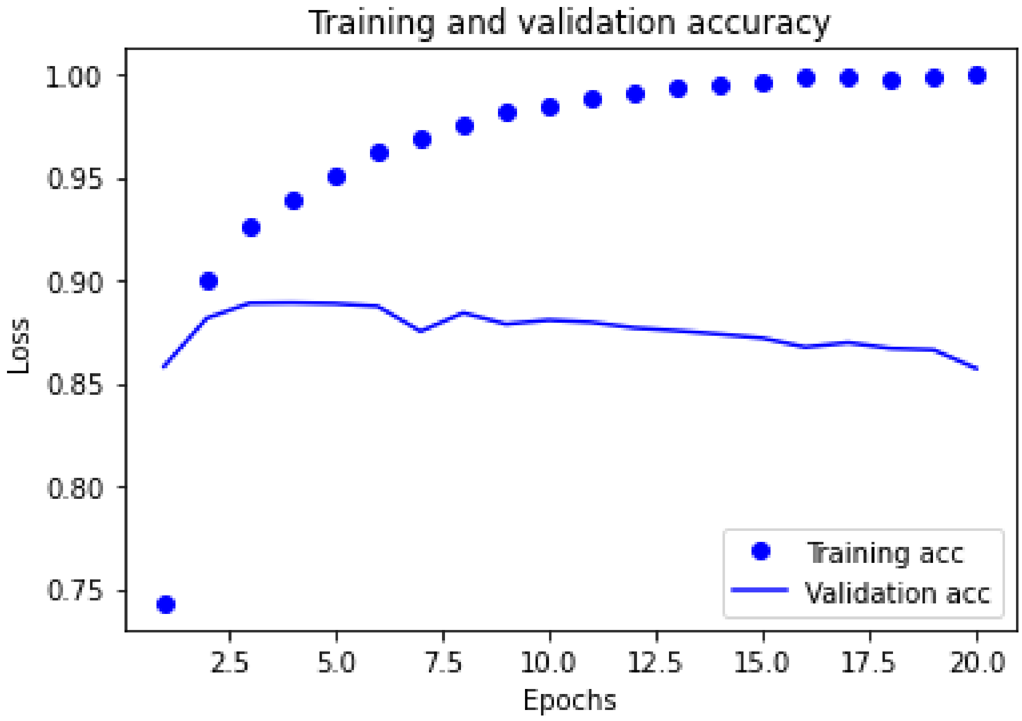
```python

```

[25]:

```python
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)

# "bo" is for "blue dot"
plt.plot(epochs, loss, 'bo', label='Training loss')
# b is for "solid blue line"
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()
```



[26]:
```
plt.clf()   # clear 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, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



[27]:

```
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 [==============================] - 0s 9ms/step - loss: 0.5202 - accuracy:
0.7878
Epoch 2/4
49/49 [==============================] - 0s 8ms/step - loss: 0.2946 - accuracy:
0.9020
Epoch 3/4
49/49 [==============================] - 0s 8ms/step - loss: 0.2161 - accuracy:
0.9236
Epoch 4/4
49/49 [==============================] - 0s 8ms/step - loss: 0.1769 - accuracy:
0.9384
```

```
782/782 [==============================] - 1s 2ms/step - loss: 0.3053 -
accuracy: 0.8798
```

[28]:
```
results
```

[28]:
```
[0.30530327558517456, 0.8797600269317627]
```

[29]:
```
model.predict(x_test)
```

[29]:
```
array([[0.27575997],
       [0.99998915],
       [0.97353494],
       …,
       [0.19384804],
       [0.14488876],
       [0.8090949 ]], dtype=float32)
```

[ ]:
```

```