File Edit View Insert Cell Kernel Navigate Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [1]:
```
import numpy as np

samples = ['The cat sat on the mat.', 'The dog ate my homework.']

token_index = {}
for sample in samples:
    for word in sample.split():
        if word not in token_index:
            token_index[word] = len(token_index) + 1

max_length = 10

results = np.zeros(shape=(len(samples),
                          max_length,
                          max(token_index.values()) + 1))
for i, sample in enumerate(samples):
    for j, word in list(enumerate(sample.split()))[:max_length]:
        index = token_index.get(word)
        results[i, j, index] = 1.
results
```

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

       [[0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]]])
```

In [3]:
```
import string
samples = ['The cat sat on the mat.', 'The dog ate my homework.']
characters = string.printable
token_index = dict(zip(range(1, len(characters) + 1), characters))

max_length = 50
results = np.zeros((len(samples), max_length, max(token_index.keys()) + 1))
for i, sample in enumerate(samples):
    for j, character in enumerate(sample):
        index = token_index.get(character)
        results[i, j, index] = 1.

results
```

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

```
[[1., 1., 1., ..., 1., 1., 1.],
 [1., 1., 1., ..., 1., 1., 1.],
 [1., 1., 1., ..., 1., 1., 1.],
 ...,
 [0., 0., 0., ..., 0., 0., 0.],
 [0., 0., 0., ..., 0., 0., 0.],
 [0., 0., 0., ..., 0., 0., 0.]])
```

In [7]:

```
from keras.preprocessing.text import Tokenizer

samples = ['The cat sat on the mat.', 'The dog ate my homework.']

tokenizer = Tokenizer(num_words=1000)
tokenizer.fit_on_texts(samples)

sequences = tokenizer.texts_to_sequences(samples)

one_hot_results = tokenizer.texts_to_matrix(samples, mode='binary')

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

```
Found 9 unique tokens.
```

1 Listing 6.5. Instantiating an Embedding layer

In [8]:

```
from keras.layers import Embedding

embedding_layer = Embedding(1000, 64)#每一個Layer 1000的常用字 #64維度vector
```

2 Listing 6.6. Loading the IMDB data for use with an Embedding layer

In [9]:

```
from keras.datasets import imdb
from keras import preprocessing

max_features = 10000 #10000常用字典
maxlen = 20    #20字詞解釋

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
```

In [15]:

```
from keras.utils.data_utils import pad_sequences
```

In [18]:

```
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
```

3 Listing 6.7. Using an Embedding layer and classifier on the IMDB data

In [19]:

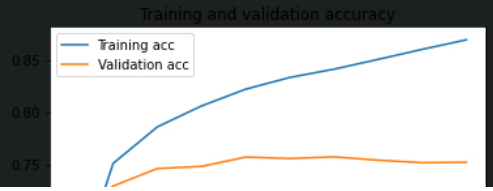
```
from keras.models import Sequential
from keras.layers import Flatten, Dense

model = Sequential()
model.add(Embedding(10000, 8, input_length=maxlen))
model.add(Flatten())#壓平 covert the tensors as vectors
model.add(Dense(1, activation='sigmoid'))

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

```
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)#隨機幫你切割驗證集
```

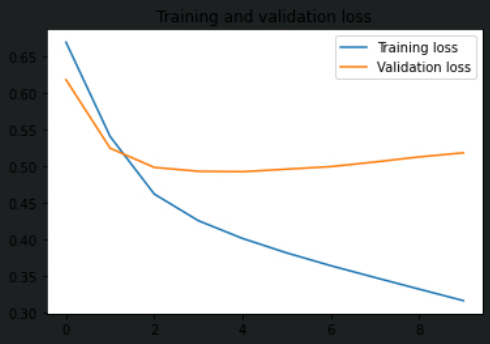
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 20, 8)             80000

 flatten (Flatten)           (None, 160)               0

 dense (Dense)               (None, 1)                 161

=================================================================
Total params: 80,161
Trainable params: 80,161
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
625/625 [==============================] - 3s 3ms/step - loss: 0.6694 - acc: 0.6222 - val_loss: 0.6182 - val_acc: 0.6952
Epoch 2/10
625/625 [==============================] - 1s 2ms/step - loss: 0.5405 - acc: 0.7509 - val_loss: 0.5245 - val_acc: 0.7292
Epoch 3/10
625/625 [==============================] - 1s 2ms/step - loss: 0.4618 - acc: 0.7858 - val_loss: 0.4983 - val_acc: 0.7460
Epoch 4/10
625/625 [==============================] - 1s 2ms/step - loss: 0.4254 - acc: 0.8059 - val_loss: 0.4930 - val_acc: 0.7480
Epoch 5/10
625/625 [==============================] - 1s 2ms/step - loss: 0.4011 - acc: 0.8220 - val_loss: 0.4924 - val_acc: 0.7570
Epoch 6/10
625/625 [==============================] - 1s 2ms/step - loss: 0.3814 - acc: 0.8332 - val_loss: 0.4959 - val_acc: 0.7556
Epoch 7/10
625/625 [==============================] - 2s 2ms/step - loss: 0.3639 - acc: 0.8411 - val_loss: 0.4994 - val_acc: 0.7572
Epoch 8/10
625/625 [==============================] - 2s 2ms/step - loss: 0.3478 - acc: 0.8505 - val_loss: 0.5057 - val_acc: 0.7540
Epoch 9/10
625/625 [==============================] - 2s 2ms/step - loss: 0.3318 - acc: 0.8601 - val_loss: 0.5127 - val_acc: 0.7516
Epoch 10/10
625/625 [==============================] - 1s 2ms/step - loss: 0.3161 - acc: 0.8692 - val_loss: 0.5183 - val_acc: 0.7520
```

In [20]:

```
import matplotlib.pyplot as plt

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

epochs = range(len(acc))

plt.plot(epochs, acc, label='Training acc')
plt.plot(epochs, val_acc, label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, label='Training loss')
plt.plot(epochs, val_loss, label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
#上面的代碼得到了約76%的驗證準確度，這對於只考慮每個影評的前20個詞的效果
```




0.70
0.65
0
2
4
6
8


Training and validation loss
Training loss
Validation loss
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0
2
4
6
8