單字的 one-hot encoding

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

# 初始資料：每一個樣本是一個輸入項目(在這範例中，樣本是一個句子，但也可以是整個文件)
samples = ['The cat sat on the mat.', 'The dog ate my homework.']

# 建立資料中所有 tokens 的索引
token_index = {}

for sample in samples:
    for word in sample.split():  # 分詞並移除樣本中的標點符號與特殊字元
        if word not in token_index:
            token_index[word] = len(token_index) + 1  # 為每個文字指定一個唯一索引

max_length = 10  # 將樣本向量化，每次只專注處理每個樣本中的第一個 max_length 文字

results = np.zeros(shape=(len(samples),  # 用來儲存結果的 Numpy array
                          max_length,
                          max(token_index.values()) + 1))
print('shape :',results.shape)
```

```
shape : (2, 10, 11)
```

In [2]:
```
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.
print('token_index :',token_index)
```

```
token_index : {'The': 1, 'cat': 2, 'sat': 3, 'on': 4, 'the': 5, 'mat.': 6, 'dog': 7, 'ate': 8, 'my': 9, 'homework.': 10}
```

字元的 one-hot encoding

In [3]:
```
import string
import numpy as np

samples = ['The cat sat on the mat.', 'The dog ate my homework.']
characters = string.printable  # 所有可印出的 ASCII 字元的字串, '0123456789abc....'
print('字元長度',len(characters))

token_index = dict(zip(characters, range(1, len(characters) + 1)))

max_length = 50
results = np.zeros((len(samples), max_length, max(token_index.values()) + 1))
print('shape :',results.shape)
```

```
字元長度 100
shape : (2, 50, 101)
```

In [4]:
```
for i, sample in enumerate(samples):
    for j, character in enumerate(sample):
        index = token_index.get(character)
        results[i, j, index] = 1.
print(results[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.
 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.]
```

用 Keras 做文字的 one-hot encoding

In [5]:
```
from keras.preprocessing.text import Tokenizer

# 初始資料
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)
print('sequences :',sequences)

one_hot_results = tokenizer.texts_to_matrix(samples, mode='binary')
print('one_hot shape :',one_hot_results.shape)

word_index = tokenizer.word_index
print('word_index :',word_index)
print('找到 %s 個唯一的 tokens.' % len(word_index))
```

```
Using TensorFlow backend.
D:\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:523: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
D:\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:524: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
D:\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:525: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
D:\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:526: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
D:\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:527: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
D:\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:532: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
sequences : [[1, 2, 3, 4, 1, 5], [1, 6, 7, 8, 9]]
one_hot shape : (2, 1000)
word_index : {'the': 1, 'cat': 2, 'sat': 3, 'on': 4, 'mat': 5, 'dog': 6, 'ate': 7, 'my': 8, 'homework': 9}
找到 9 個唯一的 tokens.
```

使用雜湊技巧的單字 one-hot encoding

In [6]:
```
samples = ['The cat sat on the mat.', 'The dog ate my homework.']

dimensionality = 1000
max_length = 10

results = np.zeros((len(samples), max_length, dimensionality))
for i, sample in enumerate(samples):
    for j, word in list(enumerate(sample.split()))[:max_length]:
        index = abs(hash(word)) % dimensionality
        results[i, j, index] = 1.
print('shape :',results.shape)
```

```
shape : (2, 10, 1000)
```

使用嵌入向量層學習文字嵌入向量

In [7]:
```
# 建立一個嵌入層 (Embedding Layer)
from keras.layers import Embedding

# 建立嵌入向量層至少須指定兩個參數
embedding_layer = Embedding(1000, 64)
```

In [8]:
```
# 載入 IMDB, 整理成適合供 Embedding 層使用的資料
from keras.datasets import imdb
from keras import preprocessing

# 設定作為特徵的文字數量
max_features = 10000
# 在 20 個文字之後切掉文字資料
maxlen = 20
```

In [11]:
```
#(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

# save np.load
np_load_old = np.load

# modify the default parameters of np.load
np.load = lambda *a,**k: np_load_old(*a, allow_pickle=True, **k)

# call load_data with allow_pickle implicitly set to true
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=10000)

# restore np.load for future normal usage
np.load = np_load_old
print('x_train shape :',x_train.shape)

# 將資料以整數 lists 載入
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
print(x_train.shape)
print(x_train[0])

# 將整數 lists 轉換為 2D 整數張量
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

```
x_train shape : (25000,)
(25000, 20)
[  65   16   38 1334   88   12   16  283    5   16 4472  113  103   32
   15   16 5345   19  178   32]
```

In [12]:
```
# 把 IMDB 資料提供給 Embedding layer和分類器
from keras.models import Sequential
from keras.layers import Flatten, Dense, Embedding

model = Sequential()
model.add(Embedding(10000, 8, input_length=maxlen)) # ←1...

model.add(Flatten()) # ← 2...

model.add(Dense(1, activation='sigmoid')) # ← 在頂部加上分類器
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)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 20, 8)             80000
_________________________________________________________________
flatten_1 (Flatten)          (None, 160)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 161
=================================================================
Total params: 80,161
Trainable params: 80,161
Non-trainable params: 0
_________________________________________________________________
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 20s 1ms/step - loss: 0.6759 - acc: 0.6050 - val_loss: 0.6398 - val_acc: 0.6814
Epoch 2/10
20000/20000 [==============================] - 2s 107us/step - loss: 0.5657 - acc: 0.7427 - val_loss: 0.5467 - val_acc: 0.7206
Epoch 3/10
20000/20000 [==============================] - 2s 110us/step - loss: 0.4752 - acc: 0.7808 - val_loss: 0.5113 - val_acc: 0.7384
Epoch 4/10
20000/20000 [==============================] - 2s 109us/step - loss: 0.4263 - acc: 0.8077 - val_loss: 0.5008 - val_acc: 0.7452
Epoch 5/10
20000/20000 [==============================] - 2s 110us/step - loss: 0.3930 - acc: 0.8258 - val_loss: 0.4981 - val_acc: 0.7538
Epoch 6/10
20000/20000 [==============================] - 2s 106us/step - loss: 0.3668 - acc: 0.8395 - val_loss: 0.5014 - val_acc: 0.7530
Epoch 7/10
20000/20000 [==============================] - 2s 111us/step - loss: 0.3435 - acc: 0.8533 - val_loss: 0.5052 - val_acc: 0.7520
Epoch 8/10
20000/20000 [==============================] - 2s 107us/step - loss: 0.3223 - acc: 0.8657 - val_loss: 0.5132 - val_acc: 0.7486
Epoch 9/10
20000/20000 [==============================] - 2s 111us/step - loss: 0.3022 - acc: 0.8766 - val_loss: 0.5213 - val_acc: 0.7490
Epoch 10/10
20000/20000 [==============================] - 2s 110us/step - loss: 0.2839 - acc: 0.8860 - val_loss: 0.5303 - val_acc: 0.7466
```

In [ ]:
```

```