In [4]:
```
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
```

In [5]:
```
results
```

Out[5]:
```
array([[[0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 1., 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., 0., 0., 0., 0.]],

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

In [6]:
```
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
```

In [7]:
```
results
```

Out[7]:
```
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 [8]:
```
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))
```

```
Using TensorFlow backend.

Found 9 unique tokens.
```

In [9]:
```
one_hot_results
```

Out[9]:
```
array([[0., 1., 1., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.]])
```

In [10]:
```
one_hot_results.shape
```

Out[10]:
```
(2, 1000)
```

In [11]:
```
from keras.layers import Embedding
embedding_layer = Embedding(1000, 64)
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\keras\back
end\tensorflow_backend.py:66: The name tf.get_default_graph is deprecated. Pl
ease use tf.compat.v1.get_default_graph instead.
```

In [13]:
```
from keras.datasets import imdb
from keras import preprocessing
max_features = 10000
maxlen = 20
(x_train, y_train), (x_test, y_test) = imdb.load_data(
num_words=max_features)
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

In [14]:
```
from keras.models import Sequential
from keras.layers import Flatten, Dense
model = Sequential()
model.add(Embedding(10000, 8, input_length=maxlen))
model.add(Flatten())
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)
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\keras\back
end\tensorflow_backend.py:541: The name tf.placeholder is deprecated. Please
use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\keras\back
end\tensorflow_backend.py:4432: The name tf.random_uniform is deprecated. Ple
ase use tf.random.uniform instead.

WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\keras\opti
mizers.py:793: The name tf.train.Optimizer is deprecated. Please use tf.compa
t.v1.train.Optimizer instead.

WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\keras\back
end\tensorflow_backend.py:3657: The name tf.log is deprecated. Please use tf.
math.log instead.

WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\tensorflow
\python\ops\nn_impl.py:180: add_dispatch_support.<locals>.wrapper (from tenso
rflow.python.ops.array_ops) is deprecated and will be removed in a future ver
sion.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Model: "sequential_1"
_________________________________________________________________
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
_________________________________________________________________
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\keras\back
end\tensorflow_backend.py:1033: The name tf.assign_add is deprecated. Please
use tf.compat.v1.assign_add instead.

Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 3s 142us/step - loss: 0.6679 -
acc: 0.6211 - val_loss: 0.6134 - val_acc: 0.6942
Epoch 2/10
20000/20000 [==============================] - 2s 114us/step - loss: 0.5353 -
acc: 0.7530 - val_loss: 0.5203 - val_acc: 0.7352
Epoch 3/10
20000/20000 [==============================] - 2s 109us/step - loss: 0.4576 -
acc: 0.7880 - val_loss: 0.4990 - val_acc: 0.7478
Epoch 4/10
20000/20000 [==============================] - 2s 109us/step - loss: 0.4205 -
acc: 0.8082 - val_loss: 0.4916 - val_acc: 0.7560
Epoch 5/10
20000/20000 [==============================] - 2s 109us/step - loss: 0.3953 -
acc: 0.8238 - val_loss: 0.4919 - val_acc: 0.7552
Epoch 6/10
```

```
20000/20000 [==============================] - 3s 128us/step - loss: 0.3748 -
acc: 0.8349 - val_loss: 0.4938 - val_acc: 0.7578
Epoch 7/10
20000/20000 [==============================] - 2s 121us/step - loss: 0.3554 -
acc: 0.8465 - val_loss: 0.4978 - val_acc: 0.7584
Epoch 8/10
20000/20000 [==============================] - 4s 178us/step - loss: 0.3368 -
acc: 0.8557 - val_loss: 0.5031 - val_acc: 0.7556
Epoch 9/10
20000/20000 [==============================] - 3s 149us/step - loss: 0.3187 -
acc: 0.8663 - val_loss: 0.5089 - val_acc: 0.7550
Epoch 10/10
20000/20000 [==============================] - 2s 111us/step - loss: 0.3014 -
acc: 0.8760 - val_loss: 0.5156 - val_acc: 0.7548
```

In [15]:

```
import os
imdb_dir='D:/Deeplearning/datasets/imdbdataset/aclImdb_v1/aclImdb'
train_dir=os.path.join(imdb_dir,'train')
labels=[]
texts=[]
for label_type in ['neg','pos']:
    dir_name=os.path.join(train_dir,label_type)
    for fname in os.listdir(dir_name):
        if fname[-4:]=='.txt':
            f=open(os.path.join(dir_name,fname),encoding='utf8')
            texts.append(f.read())
            f.close()
            if label_type=='neg':
                 labels.append(0)
            else:
                 labels.append(1)
```

In [16]:
```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import numpy as np
maxlen = 100
training_samples = 200
validation_samples = 10000
max_words = 10000
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
data = pad_sequences(sequences, maxlen=maxlen)
labels = np.asarray(labels)
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
x_train = data[:training_samples]
y_train = labels[:training_samples]
x_val = data[training_samples: training_samples + validation_samples]
y_val = labels[training_samples: training_samples + validation_samples]
```

```
Found 88582 unique tokens.
Shape of data tensor: (25000, 100)
Shape of label tensor: (25000,)
```

In [19]:
```
glove_dir = 'D:/Deeplearning/datasets/glove.6B'
embeddings_index = {}
f = open(os.path.join(glove_dir, 'glove.6B.100d.txt'), encoding='utf')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print('Found %s word vectors.' % len(embeddings_index))
```

```
Found 400000 word vectors.
```

In [21]:
```
embedding_dim = 100
embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    if i < max_words:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

In [24]:
```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

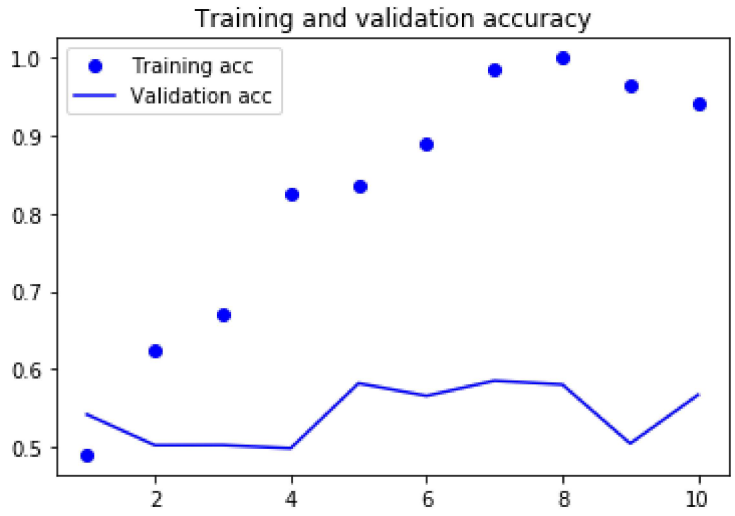
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 100, 100)          1000000
_________________________________________________________________
flatten_3 (Flatten)          (None, 10000)             0
_________________________________________________________________
dense_4 (Dense)              (None, 32)                320032
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,320,065
Trainable params: 1,320,065
Non-trainable params: 0
_________________________________________________________________
```

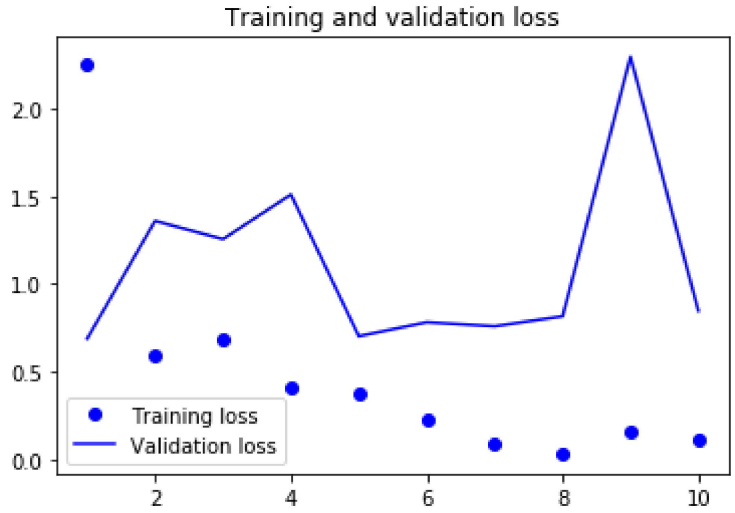
In [25]:
```
model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False
```

In [26]:
```
model.compile(optimizer='rmsprop',
loss='binary_crossentropy',
metrics=['acc'])
history = model.fit(x_train, y_train,
epochs=10,
batch_size=32,
validation_data=(x_val, y_val))
model.save_weights('pre_trained_glove_model.h5')
```

```
Train on 200 samples, validate on 10000 samples
Epoch 1/10
200/200 [==============================] - 1s 7ms/step - loss: 2.2557 - acc:
0.4900 - val_loss: 0.6898 - val_acc: 0.5416
Epoch 2/10
200/200 [==============================] - 1s 5ms/step - loss: 0.5913 - acc:
0.6250 - val_loss: 1.3597 - val_acc: 0.5020
Epoch 3/10
200/200 [==============================] - 1s 5ms/step - loss: 0.6849 - acc:
0.6700 - val_loss: 1.2557 - val_acc: 0.5022
Epoch 4/10
200/200 [==============================] - 1s 5ms/step - loss: 0.4132 - acc:
0.8250 - val_loss: 1.5097 - val_acc: 0.4981
Epoch 5/10
200/200 [==============================] - 1s 6ms/step - loss: 0.3813 - acc:
0.8350 - val_loss: 0.7035 - val_acc: 0.5816
Epoch 6/10
200/200 [==============================] - 1s 6ms/step - loss: 0.2271 - acc:
0.8900 - val_loss: 0.7831 - val_acc: 0.5655
Epoch 7/10
200/200 [==============================] - 1s 6ms/step - loss: 0.0915 - acc:
0.9850 - val_loss: 0.7601 - val_acc: 0.5849
Epoch 8/10
200/200 [==============================] - 1s 5ms/step - loss: 0.0357 - acc:
1.0000 - val_loss: 0.8167 - val_acc: 0.5800
Epoch 9/10
200/200 [==============================] - 1s 6ms/step - loss: 0.1551 - acc:
0.9650 - val_loss: 2.2936 - val_acc: 0.5042
Epoch 10/10
200/200 [==============================] - 1s 5ms/step - loss: 0.1095 - acc:
0.9400 - val_loss: 0.8481 - val_acc: 0.5668
```

In [28]:
```
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(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





In [29]:
```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
model.compile(optimizer='rmsprop',
loss='binary_crossentropy',
metrics=['acc'])
history = model.fit(x_train, y_train,
epochs=10,
batch_size=32,
validation_data=(x_val, y_val))
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, 100, 100)          1000000
_________________________________________________________________
flatten_4 (Flatten)          (None, 10000)             0
_________________________________________________________________
dense_6 (Dense)              (None, 32)                320032
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,320,065
Trainable params: 1,320,065
Non-trainable params: 0
_________________________________________________________________
Train on 200 samples, validate on 10000 samples
Epoch 1/10
200/200 [==============================] - 2s 10ms/step - loss: 0.6946 - acc:
0.4550 - val_loss: 0.6920 - val_acc: 0.5219
Epoch 2/10
200/200 [==============================] - 1s 7ms/step - loss: 0.5000 - acc:
1.0000 - val_loss: 0.6927 - val_acc: 0.5269
Epoch 3/10
200/200 [==============================] - 1s 6ms/step - loss: 0.2652 - acc:
1.0000 - val_loss: 0.6932 - val_acc: 0.5248
Epoch 4/10
200/200 [==============================] - 1s 6ms/step - loss: 0.1134 - acc:
1.0000 - val_loss: 0.6939 - val_acc: 0.5326
Epoch 5/10
200/200 [==============================] - 1s 7ms/step - loss: 0.0523 - acc:
1.0000 - val_loss: 0.6979 - val_acc: 0.5302
Epoch 6/10
200/200 [==============================] - 1s 6ms/step - loss: 0.0262 - acc:
1.0000 - val_loss: 0.7007 - val_acc: 0.5326
Epoch 7/10
200/200 [==============================] - 1s 6ms/step - loss: 0.0143 - acc:
1.0000 - val_loss: 0.7039 - val_acc: 0.5363
Epoch 8/10
200/200 [==============================] - 1s 6ms/step - loss: 0.0083 - acc:
1.0000 - val_loss: 0.7089 - val_acc: 0.5344
Epoch 9/10
200/200 [==============================] - 1s 6ms/step - loss: 0.0050 - acc:
1.0000 - val_loss: 0.7128 - val_acc: 0.5334
Epoch 10/10
200/200 [==============================] - 1s 6ms/step - loss: 0.0030 - acc:
1.0000 - val_loss: 0.7166 - val_acc: 0.5362
```

In [31]:
```
test_dir = os.path.join(imdb_dir, 'test')
labels = []
texts = []
for label_type in ['neg', 'pos']:
    dir_name = os.path.join(test_dir, label_type)
    for fname in sorted(os.listdir(dir_name)):
        if fname[-4:] == '.txt':
            f = open(os.path.join(dir_name, fname),encoding='utf')
            texts.append(f.read())
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)
sequences = tokenizer.texts_to_sequences(texts)
x_test = pad_sequences(sequences, maxlen=maxlen)
y_test = np.asarray(labels)
```

In [32]:
```
model.load_weights('pre_trained_glove_model.h5')
model.evaluate(x_test, y_test)
```

```
25000/25000 [==============================] - 3s 103us/step
```

Out[32]:
```
[0.8409046409893036, 0.57416]
```

In [33]:
```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import numpy as np
maxlen = 500
training_samples = 200
validation_samples = 10000
max_words = 10000
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
data = pad_sequences(sequences, maxlen=maxlen)
labels = np.asarray(labels)
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
x_train = data[:training_samples]
y_train = labels[:training_samples]
x_val = data[training_samples: training_samples + validation_samples]
y_val = labels[training_samples: training_samples + validation_samples]
```

```
Found 87393 unique tokens.
Shape of data tensor: (25000, 500)
Shape of label tensor: (25000,)
```

In [38]:
```
embedding_dim = 100
embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    if i < max_words:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

embedding_vector.shape

In [39]:
```
embedding_vector.shape
```

Out[39]:
```
(100,)
```

In [40]:
```
len(word_index)
```

Out[40]:
```
87393
```

In [41]:
```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 500, 100)          1000000
_________________________________________________________________
flatten_5 (Flatten)          (None, 50000)             0
_________________________________________________________________
dense_8 (Dense)              (None, 32)                1600032
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 33
=================================================================
Total params: 2,600,065
Trainable params: 2,600,065
Non-trainable params: 0
_________________________________________________________________
```

In [42]:
```
model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False
```

In [43]:
```
model.compile(optimizer='rmsprop',
loss='binary_crossentropy',
metrics=['acc'])
history = model.fit(x_train, y_train,
epochs=10,
batch_size=32,
validation_data=(x_val, y_val))
model.save_weights('pre_trained_glove_model_with_maxlen500.h5')
```

```
Train on 200 samples, validate on 10000 samples
Epoch 1/10
200/200 [==============================] - 4s 18ms/step - loss: 5.4170 - acc:
0.5550 - val_loss: 4.5301 - val_acc: 0.4944
Epoch 2/10
200/200 [==============================] - 3s 14ms/step - loss: 1.4793 - acc:
0.6100 - val_loss: 1.3307 - val_acc: 0.4963
Epoch 3/10
200/200 [==============================] - 3s 14ms/step - loss: 0.5510 - acc:
0.7150 - val_loss: 0.6907 - val_acc: 0.5476
Epoch 4/10
200/200 [==============================] - 3s 14ms/step - loss: 0.4283 - acc:
0.7950 - val_loss: 0.7311 - val_acc: 0.5296
Epoch 5/10
200/200 [==============================] - 3s 15ms/step - loss: 0.2270 - acc:
0.9450 - val_loss: 1.1137 - val_acc: 0.5314
Epoch 6/10
200/200 [==============================] - 3s 16ms/step - loss: 0.1149 - acc:
0.9850 - val_loss: 1.2881 - val_acc: 0.5105
Epoch 7/10
200/200 [==============================] - 3s 14ms/step - loss: 0.7179 - acc:
0.8550 - val_loss: 0.9706 - val_acc: 0.5340
Epoch 8/10
200/200 [==============================] - 3s 14ms/step - loss: 0.0422 - acc:
0.9950 - val_loss: 1.1515 - val_acc: 0.5164
Epoch 9/10
200/200 [==============================] - 3s 14ms/step - loss: 0.0166 - acc:
1.0000 - val_loss: 0.8314 - val_acc: 0.5633
Epoch 10/10
200/200 [==============================] - 3s 14ms/step - loss: 0.0080 - acc:
1.0000 - val_loss: 0.8109 - val_acc: 0.5699
```

In [45]:
```
test_dir = os.path.join(imdb_dir, 'test')
labels = []
texts = []
for label_type in ['neg', 'pos']:
    dir_name = os.path.join(test_dir, label_type)
    for fname in sorted(os.listdir(dir_name)):
        if fname[-4:] == '.txt':
            f = open(os.path.join(dir_name, fname),encoding='utf')
            texts.append(f.read())
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)
sequences = tokenizer.texts_to_sequences(texts)
x_test = pad_sequences(sequences, maxlen=maxlen)
y_test = np.asarray(labels)
```

In [46]:
```
model.load_weights('pre_trained_glove_model_with_maxlen500.h5')
model.evaluate(x_test, y_test)
```

```
25000/25000 [==============================] - 7s 270us/step
```

Out[46]:
```
[0.8168841763877869, 0.56632]
```

In [48]:
```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import numpy as np
maxlen = 500
training_samples = 200
validation_samples = 10000
max_words = 20000
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
data = pad_sequences(sequences, maxlen=maxlen)
labels = np.asarray(labels)
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
x_train = data[:training_samples]
y_train = labels[:training_samples]
x_val = data[training_samples: training_samples + validation_samples]
y_val = labels[training_samples: training_samples + validation_samples]
```

```
Found 87393 unique tokens.
Shape of data tensor: (25000, 500)
Shape of label tensor: (25000,)
```

In [49]:
```
embedding_dim = 100
embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    if i < max_words:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

In [50]:
```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_7 (Embedding)      (None, 500, 100)          2000000
_________________________________________________________________
flatten_6 (Flatten)          (None, 50000)             0
_________________________________________________________________
dense_10 (Dense)             (None, 32)                1600032
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 33
=================================================================
Total params: 3,600,065
Trainable params: 3,600,065
Non-trainable params: 0
_________________________________________________________________
```

In [51]:
```
model.compile(optimizer='rmsprop',
loss='binary_crossentropy',
metrics=['acc'])
history = model.fit(x_train, y_train,
epochs=10,
batch_size=32,
validation_data=(x_val, y_val))
model.save_weights('pre_trained_glove_model_with_maxlen500_incft.h5')
```

```
Train on 200 samples, validate on 10000 samples
Epoch 1/10
200/200 [==============================] - 4s 21ms/step - loss: 0.9553 - acc:
0.5250 - val_loss: 0.6960 - val_acc: 0.5034
Epoch 2/10
200/200 [==============================] - 3s 16ms/step - loss: 0.6257 - acc:
0.5950 - val_loss: 0.7068 - val_acc: 0.5020
Epoch 3/10
200/200 [==============================] - 3s 16ms/step - loss: 0.5473 - acc:
0.6100 - val_loss: 0.7165 - val_acc: 0.5026
Epoch 4/10
200/200 [==============================] - 3s 17ms/step - loss: 0.5088 - acc:
0.7000 - val_loss: 0.9153 - val_acc: 0.5024
Epoch 5/10
200/200 [==============================] - 4s 18ms/step - loss: 0.4739 - acc:
0.6900 - val_loss: 1.0239 - val_acc: 0.5033
Epoch 6/10
200/200 [==============================] - 3s 16ms/step - loss: 0.4028 - acc:
0.7000 - val_loss: 1.0053 - val_acc: 0.5023
Epoch 7/10
200/200 [==============================] - 3s 16ms/step - loss: 0.3542 - acc:
0.8000 - val_loss: 0.7608 - val_acc: 0.5029
Epoch 8/10
200/200 [==============================] - 3s 16ms/step - loss: 0.3107 - acc:
0.9350 - val_loss: 0.8585 - val_acc: 0.5025
Epoch 9/10
200/200 [==============================] - 3s 16ms/step - loss: 0.1588 - acc:
1.0000 - val_loss: 0.7873 - val_acc: 0.5037
Epoch 10/10
200/200 [==============================] - 3s 16ms/step - loss: 0.0833 - acc:
0.9900 - val_loss: 0.8195 - val_acc: 0.4941
```

In [52]:
```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import numpy as np
maxlen = 500
training_samples = 10000
validation_samples = 10000
max_words = 20000
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
data = pad_sequences(sequences, maxlen=maxlen)
labels = np.asarray(labels)
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
x_train = data[:training_samples]
y_train = labels[:training_samples]
x_val = data[training_samples: training_samples + validation_samples]
y_val = labels[training_samples: training_samples + validation_samples]
```

```
Found 87393 unique tokens.
Shape of data tensor: (25000, 500)
Shape of label tensor: (25000,)
```

In [53]:
```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_8 (Embedding)      (None, 500, 100)          2000000
_________________________________________________________________
flatten_7 (Flatten)          (None, 50000)             0
_________________________________________________________________
dense_12 (Dense)             (None, 32)                1600032
_________________________________________________________________
dense_13 (Dense)             (None, 1)                 33
=================================================================
Total params: 3,600,065
Trainable params: 3,600,065
Non-trainable params: 0
_________________________________________________________________
```

In [54]:
```
model.compile(optimizer='rmsprop',
loss='binary_crossentropy',
metrics=['acc'])
history = model.fit(x_train, y_train,
epochs=10,
batch_size=32,
validation_data=(x_val, y_val))
model.save_weights('pre_trained_glove_model_with_maxlen500_inctr.h5')
```

```
Train on 10000 samples, validate on 10000 samples
Epoch 1/10
10000/10000 [==============================] - 37s 4ms/step - loss: 0.6997 -
acc: 0.4936 - val_loss: 0.6932 - val_acc: 0.5030
Epoch 2/10
10000/10000 [==============================] - 35s 3ms/step - loss: 0.5423 -
acc: 0.7245 - val_loss: 0.8161 - val_acc: 0.5055
Epoch 3/10
10000/10000 [==============================] - 35s 3ms/step - loss: 0.0966 -
acc: 0.9793 - val_loss: 1.2070 - val_acc: 0.5006
Epoch 4/10
10000/10000 [==============================] - 34s 3ms/step - loss: 0.0179 -
acc: 0.9972 - val_loss: 1.5850 - val_acc: 0.5020
Epoch 5/10
10000/10000 [==============================] - 34s 3ms/step - loss: 0.0185 -
acc: 0.9975 - val_loss: 1.3962 - val_acc: 0.4982
Epoch 6/10
10000/10000 [==============================] - 35s 3ms/step - loss: 0.0088 -
acc: 0.9976 - val_loss: 2.0213 - val_acc: 0.4998
Epoch 7/10
10000/10000 [==============================] - 35s 3ms/step - loss: 0.0087 -
acc: 0.9978 - val_loss: 1.4575 - val_acc: 0.4987
Epoch 8/10
10000/10000 [==============================] - 35s 3ms/step - loss: 0.0053 -
acc: 0.9978 - val_loss: 2.0473 - val_acc: 0.4989
Epoch 9/10
10000/10000 [==============================] - 35s 3ms/step - loss: 0.0063 -
acc: 0.9978 - val_loss: 2.4881 - val_acc: 0.5006
Epoch 10/10
10000/10000 [==============================] - 39s 4ms/step - loss: 0.0043 -
acc: 0.9978 - val_loss: 2.4361 - val_acc: 0.4963
```

In [55]:
```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import numpy as np
maxlen = 100
training_samples = 12000
validation_samples = 500
max_words = 10000
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
data = pad_sequences(sequences, maxlen=maxlen)
labels = np.asarray(labels)
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
x_train = data[:training_samples]
y_train = labels[:training_samples]
x_val = data[training_samples: training_samples + validation_samples]
y_val = labels[training_samples: training_samples + validation_samples]
```

```
Found 87393 unique tokens.
Shape of data tensor: (25000, 100)
Shape of label tensor: (25000,)
```

In [59]:
```
from keras.layers import Embedding, Flatten, Dense
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_9 (Embedding)      (None, 100, 100)          1000000
_________________________________________________________________
flatten_8 (Flatten)          (None, 10000)             0
_________________________________________________________________
dense_14 (Dense)             (None, 32)                320032
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 33
=================================================================
Total params: 1,320,065
Trainable params: 1,320,065
Non-trainable params: 0
_________________________________________________________________
```

In [60]:
```
model.compile(optimizer='rmsprop',
loss='binary_crossentropy',
metrics=['acc'])
history = model.fit(x_train, y_train,
epochs=10,
batch_size=32,
validation_data=(x_val, y_val))
model.save_weights('pre_trained_glove_model_with_maxlen500_inctr12000.h5')
```

```
Train on 12000 samples, validate on 500 samples
Epoch 1/10
12000/12000 [==============================] - 16s 1ms/step - loss: 0.6944 -
acc: 0.4932 - val_loss: 0.6930 - val_acc: 0.4840
Epoch 2/10
12000/12000 [==============================] - 16s 1ms/step - loss: 0.4438 -
acc: 0.8141 - val_loss: 0.8757 - val_acc: 0.5240
Epoch 3/10
12000/12000 [==============================] - 15s 1ms/step - loss: 0.0765 -
acc: 0.9819 - val_loss: 1.2733 - val_acc: 0.4960
Epoch 4/10
12000/12000 [==============================] - 15s 1ms/step - loss: 0.0233 -
acc: 0.9961 - val_loss: 1.4820 - val_acc: 0.4900
Epoch 5/10
12000/12000 [==============================] - 17s 1ms/step - loss: 0.0137 -
acc: 0.9966 - val_loss: 1.6801 - val_acc: 0.4980
Epoch 6/10
12000/12000 [==============================] - 15s 1ms/step - loss: 0.0096 -
acc: 0.9969 - val_loss: 1.8234 - val_acc: 0.4980
Epoch 7/10
12000/12000 [==============================] - 15s 1ms/step - loss: 0.0055 -
acc: 0.9970 - val_loss: 2.2648 - val_acc: 0.4940
Epoch 8/10
12000/12000 [==============================] - 16s 1ms/step - loss: 0.0048 -
acc: 0.9973 - val_loss: 2.4340 - val_acc: 0.5000
Epoch 9/10
12000/12000 [==============================] - 16s 1ms/step - loss: 0.0041 -
acc: 0.9972 - val_loss: 2.6214 - val_acc: 0.5100
Epoch 10/10
12000/12000 [==============================] - 16s 1ms/step - loss: 0.0040 -
acc: 0.9975 - val_loss: 2.6987 - val_acc: 0.5040
```

In [63]:
```
sequences = tokenizer.texts_to_sequences(texts)
x_test = pad_sequences(sequences, maxlen=maxlen)
y_test = np.asarray(labels)
x_test_data =x_test[:200]
y_test_data = y_test[:200]
model.load_weights('pre_trained_glove_model_with_maxlen500_inctr12000.h5')
model.evaluate(x_test_data, y_test_data)
```

```
200/200 [==============================] - 0s 98us/step
```

Out[63]:
```
[6.021943397521973, 0.46]
```

In [62]:
```
x_test_data.shape
```

Out[62]:
```
(200, 500)
```

In [64]:
```
model.load_weights('pre_trained_glove_model_with_maxlen500_inctr12000.h5')
model.evaluate(x_test, y_test)
```

```
25000/25000 [==============================] - 3s 105us/step
```

Out[64]:
```
[5.332694325637817, 0.502]
```

In [ ]:
```

```