1 Listing 6.21. Numpy implementation of a simple RNN

In [1]:

```python
import numpy as np
timesteps = 100#選前100的觀察值
input_features = 32#32 embeding vector
output_features = 64
inputs = np.random.random((timesteps, input_features))
state_t = np.zeros((output_features,))
W = np.random.random((output_features, input_features))
U = np.random.random((output_features, output_features))
b = np.random.random((output_features,))#Z0=0
successive_outputs = []
for input_t in inputs:
 output_t = np.tanh(np.dot(W, input_t) + np.dot(U, state_t) + b)#類似標準化，使範圍界在-pi/2到pi/2之間
 successive_outputs.append(output_t)#產出新的output
 state_t = output_t#取代
```

In [2]:

```python
final_output_sequence = np.concatenate(successive_outputs, axis=0)
```

In [3]:

```python
output_t = np.tanh(np.dot(W, input_t) + np.dot(U, state_t) + b)
```

In [4]:

```python
#3D tensors
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN
model = Sequential()
model.add(Embedding(10000, 32))#10000 token 常用的字
model.add(SimpleRNN(32))#32dim vector
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 32)          320000

 simple_rnn (SimpleRNN)      (None, 32)                2080

=================================================================
Total params: 322,080
Trainable params: 322,080
Non-trainable params: 0
_________________________________________________________________
```

In [5]:

```python
#2D tensors
model = Sequential()
model.add(Embedding(10000, 32))
model.add(SimpleRNN(32, return_sequences=True))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, None, 32)          320000

 simple_rnn_1 (SimpleRNN)    (None, None, 32)          2080

=================================================================
Total params: 322,080
Trainable params: 322,080
Non-trainable params: 0
_________________________________________________________________
```

In [6]:

```python
model = Sequential()
model.add(Embedding(10000, 32))
#換成4 recurrent layers
model.add(SimpleRNN(32, return_sequences=True))
model.add(SimpleRNN(32, return_sequences=True))
model.add(SimpleRNN(32, return_sequences=True))
model.add(SimpleRNN(32))
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, None, 32)          320000

 simple_rnn_2 (SimpleRNN)    (None, None, 32)          2080

 simple_rnn_3 (SimpleRNN)    (None, None, 32)          2080

 simple_rnn_4 (SimpleRNN)    (None, None, 32)          2080

 simple_rnn_5 (SimpleRNN)    (None, 32)                2080

=================================================================
Total params: 328,320
Trainable params: 328,320
Non-trainable params: 0
_________________________________________________________________
```

2 Listing 6.22. Preparing the IMDB data

In [12]:

```
from keras.datasets import imdb
#from keras.preprocessing import sequence
from keras.utils.data_utils import pad_sequences
max_features = 10000#10000 tokens
maxlen = 500
batch_size = 32
print('Loading data...')
(input_train, y_train), (input_test, y_test) = imdb.load_data(
 num_words=max_features)
print(len(input_train), 'train sequences')
print(len(input_test), 'test sequences')
print('Pad sequences (samples x time)')
input_train = pad_sequences(input_train, maxlen=maxlen)
input_test = pad_sequences(input_test, maxlen=maxlen)
print('input_train shape:', input_train.shape)
print('input_test shape:', input_test.shape)
```

```
Loading data...
25000 train sequences
25000 test sequences
Pad sequences (samples x time)
input_train shape: (25000, 500)
input_test shape: (25000, 500)
```

Listing 6.23. Training the model with Embedding and SimpleRNN layers

In [13]:

```
from keras.layers import Dense
model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(input_train, y_train,
 epochs=10,
 batch_size=128,
 validation_split=0.2)#validation data
```

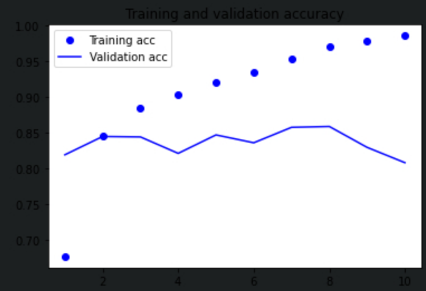
```
Epoch 1/10
157/157 [==============================] - 31s 181ms/step - loss: 0.5941 - acc: 0.6766 - val_loss: 0.4403 - val_acc: 0.8188
Epoch 2/10
157/157 [==============================] - 24s 154ms/step - loss: 0.3754 - acc: 0.8452 - val_loss: 0.3536 - val_acc: 0.8444
Epoch 3/10
157/157 [==============================] - 31s 196ms/step - loss: 0.2966 - acc: 0.8838 - val_loss: 0.3999 - val_acc: 0.8438
Epoch 4/10
157/157 [==============================] - 27s 169ms/step - loss: 0.2522 - acc: 0.9036 - val_loss: 0.3956 - val_acc: 0.8208
Epoch 5/10
157/157 [==============================] - 24s 151ms/step - loss: 0.2131 - acc: 0.9194 - val_loss: 0.3744 - val_acc: 0.8466
Epoch 6/10
157/157 [==============================] - 23s 147ms/step - loss: 0.1771 - acc: 0.9344 - val_loss: 0.4161 - val_acc: 0.8356
Epoch 7/10
157/157 [==============================] - 24s 152ms/step - loss: 0.1307 - acc: 0.9532 - val_loss: 0.5102 - val_acc: 0.8572
Epoch 8/10
157/157 [==============================] - 24s 154ms/step - loss: 0.0918 - acc: 0.9700 - val_loss: 0.4789 - val_acc: 0.8584
Epoch 9/10
157/157 [==============================] - 24s 151ms/step - loss: 0.0692 - acc: 0.9774 - val_loss: 0.5045 - val_acc: 0.8292
Epoch 10/10
157/157 [==============================] - 23s 149ms/step - loss: 0.0489 - acc: 0.9853 - val_loss: 0.5966 - val_acc: 0.8078
```

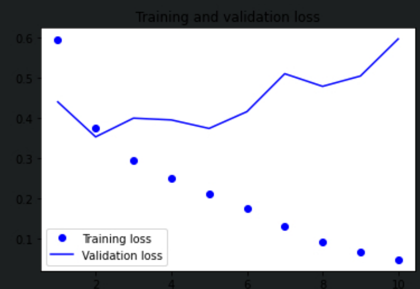
3 Listing 6.24. Plotting results

In [14]:

```
import matplotlib.pyplot as plt
acc = history.history['acc']
```

```python
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()
```





- 看出有點overfiting，基本上0.85的acc很極限了

4 Listing 6.25. Pseudocode details of the LSTM architecture (1/2)

```
output_t = activation(dot(state_t, Uo) + dot(input_t, Wo) + dot(C_t, Vo) + bo)
i_t = activation(dot(state_t, Ui) + dot(input_t, Wi) + bi)
f_t = activation(dot(state_t, Uf) + dot(input_t, Wf) + bf)
k_t = activation(dot(state_t, Uk) + dot(input_t, Wk) + bk)
Listing 6.26. Pseudocode details of the LSTM architecture (2/2)
c_t+1 = i_t * k_t + c_t * f_t
```

5 Listing 6.27. Using the LSTM layer in Keras

In [15]:

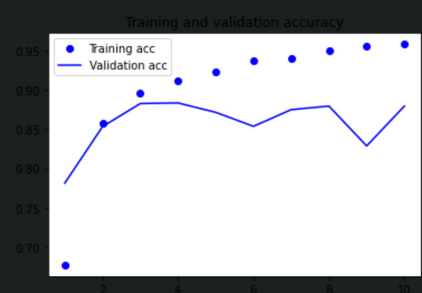
```python
from keras.layers import LSTM
model = Sequential()
model.add(Embedding(max_features, 32))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
 loss='binary_crossentropy',
 metrics=['acc'])
history = model.fit(input_train, y_train,
 epochs=10,
 batch_size=128,
 validation_split=0.2)
```

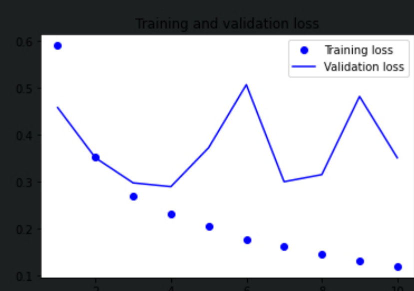
```
Epoch 1/10
157/157 [==============================] - 60s 359ms/step - loss: 0.5898 - acc: 0.6771 - val_loss: 0.4573 - val_acc: 0.7814
Epoch 2/10
157/157 [==============================] - 61s 389ms/step - loss: 0.3528 - acc: 0.8571 - val_loss: 0.3502 - val_acc: 0.8534
Epoch 3/10
157/157 [==============================] - 55s 348ms/step - loss: 0.2680 - acc: 0.8960 - val_loss: 0.2967 - val_acc: 0.8828
```

```
Epoch 4/10
157/157 [==============================] - 54s 342ms/step - loss: 0.2306 - acc: 0.9122 - val_loss: 0.2886 - val_acc: 0.8836
Epoch 5/10
157/157 [==============================] - 53s 339ms/step - loss: 0.2033 - acc: 0.9239 - val_loss: 0.3716 - val_acc: 0.8716
Epoch 6/10
157/157 [==============================] - 57s 364ms/step - loss: 0.1746 - acc: 0.9373 - val_loss: 0.5062 - val_acc: 0.8538
Epoch 7/10
157/157 [==============================] - 60s 383ms/step - loss: 0.1612 - acc: 0.9408 - val_loss: 0.2992 - val_acc: 0.8750
Epoch 8/10
157/157 [==============================] - 55s 353ms/step - loss: 0.1445 - acc: 0.9499 - val_loss: 0.3143 - val_acc: 0.8796
Epoch 9/10
157/157 [==============================] - 55s 349ms/step - loss: 0.1309 - acc: 0.9554 - val_loss: 0.4811 - val_acc: 0.8288
Epoch 10/10
157/157 [==============================] - 54s 344ms/step - loss: 0.1186 - acc: 0.9585 - val_loss: 0.3502 - val_acc: 0.8796
```

In [16]:

```python
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()
```





* 提升至0.87