Lab 6

Aim: To implement a Recurrent Neural Network

```
docs = ['go india',
            'india india',
            'hip hip hurray',
            'india will win bro',
            'i love my india',
            'kohli is the greatest batsman',
            'sachin scored a lot of centuries',
            'dhoni won the second world cup for india',
            'will modi win the third term',
            'secular and diverse india']

from keras.preprocessing.text import Tokenizer
#tokenizer = Tokenizer(oov_token='<IoTSection>')
tokenizer = Tokenizer()

tokenizer.fit_on_texts(docs)

tokenizer.word_index
```

```
{'india': 1,
 'the': 2,
 'hip': 3,
 'will': 4,
 'win': 5,
 'go': 6,
 'hurray': 7,
 'bro': 8,
 'i': 9,
 'love': 10,
 'my': 11,
 'kohli': 12,
 'is': 13,
 'greatest': 14,
 'batsman': 15,
 'sachin': 16,
 'scored': 17,
 'a': 18,
 'lot': 19,
 'of': 20,
 'centuries': 21,
 'dhoni': 22,
 'won': 23,
 'second': 24,
 'world': 25,
```

```
 'cup': 26,
 'for': 27,
 'modi': 28,
 'third': 29,
 'term': 30,
 'secular': 31,
 'and': 32,
 'diverse': 33}
```

```
tokenizer.word_counts
```

```
OrderedDict([('go', 1),
             ('india', 7),
             ('hip', 2),
             ('hurray', 1),
             ('will', 2),
             ('win', 2),
             ('bro', 1),
             ('i', 1),
             ('love', 1),
             ('my', 1),
             ('kohli', 1),
             ('is', 1),
             ('the', 3),
             ('greatest', 1),
             ('batsman', 1),
             ('sachin', 1),
             ('scored', 1),
             ('a', 1),
             ('lot', 1),
             ('of', 1),
             ('centuries', 1),
             ('dhoni', 1),
             ('won', 1),
             ('second', 1),
             ('world', 1),
             ('cup', 1),
             ('for', 1),
             ('modi', 1),
             ('third', 1),
             ('term', 1),
             ('secular', 1),
             ('and', 1),
             ('diverse', 1)])
```

```
tokenizer.document_count
```

```
10
```

```
sequences = tokenizer.texts_to_sequences(docs)
sequences
```

```
[[6, 1],
 [1, 1],
 [3, 3, 7],
 [1, 4, 5, 8],
 [9, 10, 11, 1],
 [12, 13, 2, 14, 15],
 [16, 17, 18, 19, 20, 21],
 [22, 23, 2, 24, 25, 26, 27, 1],
 [4, 28, 5, 2, 29, 30],
 [31, 32, 33, 1]]
```

```
from keras.utils import pad_sequences

sequences = pad_sequences(sequences,padding='post')
sequences
```

```
array([[ 6,  1,  0,  0,  0,  0,  0,  0],
       [ 1,  1,  0,  0,  0,  0,  0,  0],
       [ 3,  3,  7,  0,  0,  0,  0,  0],
       [ 1,  4,  5,  8,  0,  0,  0,  0],
       [ 9, 10, 11,  1,  0,  0,  0,  0],
       [12, 13,  2, 14, 15,  0,  0,  0],
       [16, 17, 18, 19, 20, 21,  0,  0],
       [22, 23,  2, 24, 25, 26, 27,  1],
       [ 4, 28,  5,  2, 29, 30,  0,  0],
       [31, 32, 33,  1,  0,  0,  0,  0]], dtype=int32)
```

```
len(sequences[0])
```

```
8
```

```
sequences
```

```
array([[ 6,  1,  0,  0,  0,  0,  0,  0],
       [ 1,  1,  0,  0,  0,  0,  0,  0],
       [ 3,  3,  7,  0,  0,  0,  0,  0],
       [ 1,  4,  5,  8,  0,  0,  0,  0],
       [ 9, 10, 11,  1,  0,  0,  0,  0],
       [12, 13,  2, 14, 15,  0,  0,  0],
       [16, 17, 18, 19, 20, 21,  0,  0],
       [22, 23,  2, 24, 25, 26, 27,  1],
       [ 4, 28,  5,  2, 29, 30,  0,  0],
       [31, 32, 33,  1,  0,  0,  0,  0]], dtype=int32)
```

```
from keras import Sequential
from keras.layers import Dense, SimpleRNN, Embedding
model = Sequential()
model.add(Embedding(34,output_dim=2,input_length=8))
```

```
model.add(Dense(units = 2, activation='softmax'))
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 8, 2)              68

 dense_4 (Dense)             (None, 8, 2)              6

=================================================================
Total params: 74 (296.00 Byte)
Trainable params: 74 (296.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.compile('adam','accuracy')
```

```
pred = model.predict(sequences)
print(pred)
pred.shape
```

```
1/1 [==============================] - 0s 244ms/step
[[[0.5128169  0.4871832 ]
  [0.50593483 0.49406517]
  [0.47728473 0.5227153 ]
  [0.47728473 0.5227153 ]
  [0.47728473 0.5227153 ]
  [0.47728473 0.5227153 ]
  [0.47728473 0.5227153 ]
  [0.47728473 0.5227153 ]]

 [[0.50593483 0.49406517]
  [0.50593483 0.49406517]
  [0.47728473 0.5227153 ]
  [0.47728473 0.5227153 ]
  [0.47728473 0.5227153 ]
  [0.47728473 0.5227153 ]
  [0.47728473 0.5227153 ]
  [0.47728473 0.5227153 ]]

 [[0.5012797  0.49872032]
  [0.5012797  0.49872032]
  [0.48481685 0.51518315]
  [0.47728473 0.5227153 ]
  [0.47728473 0.5227153 ]
  [0.47728473 0.5227153 ]
  [0.47728473 0.5227153 ]
  [0.47728473 0.5227153 ]]
```

```
[[0.50593483 0.49406517]
 [0.5035462  0.49645385]
 [0.4906695  0.5093305 ]
 [0.49849063 0.50150937]
 [0.47728473 0.5227153 ]
 [0.47728473 0.5227153 ]
 [0.47728473 0.5227153 ]
 [0.47728473 0.5227153 ]]

[[0.49348876 0.5065113 ]
 [0.51095986 0.48904002]
 [0.5139852  0.48601478]
 [0.50593483 0.49406517]
 [0.47728473 0.5227153 ]
 [0.47728473 0.5227153 ]
 [0.47728473 0.5227153 ]
 [0.47728473 0.5227153 ]]

[[0.4952051  0.5047948 ]
 [0.5148093  0.48519066]
 [0.4865776  0.5134225 ]
 [0.4873007  0.51269937]
 [0.50977427 0.49022582]
 [0.47728473 0.5227153 ]
 [0.47728473 0.5227153 ]
 [0.47728473 0.5227153 ]]

[[0.4775283  0.52247167]
 [0.5062023  0.49379766]
 [0.5092295  0.49077052]
 [0.50273633 0.49726364]
 [0.5161321  0.48386788]
 [0.48259777 0.5174022 ]
 [0.47728473 0.5227153 ]
 [0.47728473 0.5227153 ]]

[[0.50934523 0.49065474]
 [0.5105865  0.48941356]
 [0.4865776  0.5134225 ]
 [0.50501454 0.4949855 ]
 [0.5084248  0.4915752 ]
 [0.48253918 0.5174609 ]
 [0.5083535  0.4916465 ]
 [0.50593483 0.49406517]]

[[0.5035462  0.49645385]
 [0.51851344 0.4814865 ]
 [0.4906695  0.5093305 ]
 [0.4865776  0.5134225 ]
```

```
  [0.48617196 0.513828  ]
  [0.5031508  0.49684924]
  [0.47728473 0.5227153 ]
  [0.47728473 0.5227153 ]]

 [[0.49673644 0.5032636 ]
  [0.5091956  0.49080437]
  [0.4768464  0.5231536 ]
  [0.50593483 0.49406517]
  [0.47728473 0.5227153 ]
  [0.47728473 0.5227153 ]
  [0.47728473 0.5227153 ]
  [0.47728473 0.5227153 ]]]
```

```
(10, 8, 2)
```

```
from keras.datasets import imdb
from keras import Sequential
from keras.layers import Dense,SimpleRNN,Embedding,Flatten
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
```

```
(X_train,y_train),(X_test,y_test) = imdb.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/imdb.npz
17464789/17464789 [==============================] - 0s 0us/step
```

```
X_train[0]
```

```
[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,
22665,
9,
35,
480,
284,
5,
150,
4,
172,
112,
167,
21631,
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,
19193,
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,
10311,
8,
4,
107,
117,
5952,
15,
256,
4,
31050,
7,
3766,
5,
723,
36,
71,
43,
530,
476,
26,
400,
317,
46,
7,
4,
12118,
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]

import numpy as np
max(X_train)

[1,
 88325,
 733,
 7,
 14,
 706,
 40354,
 1936,
 4,
 4423,
 12322,
 23,
 4,
 419,
 97,
 252,
 25,
 332,
 12,
 11,
 420,
 25,
 92,
 6,
 604,
 7,
 3368,
 1828,
 125,
 83,
 4,
 933,
 1411,
 7,
 4,
 6024,
 10805,
 8,
 26519,
 187,
 19,
 6,
 762,
 7,
```

```
6226,
11,
35,
1626,
6816,
4,
194,
2020,
223,
152,
193,
7880,
8,
14,
5,
4757,
8,
1261,
125,
4,
768,
31,
34,
31,
2256,
17683,
194,
2020,
309,
47,
6,
8849,
4191,
40,
393,
21,
9,
329,
629,
444,
74,
1335,
39,
1335,
5,
4,
48333,
91,
7,
```

```
4,
20,
47,
4,
1639,
4173,
156,
4470,
8,
30,
3368,
45514,
187,
11,
4,
1411,
5,
59784,
245,
14,
9,
6,
57,
352,
20,
11,
63,
55,
117,
571,
33,
222,
23,
268,
75,
81,
79,
8,
106,
4,
13537,
7,
4,
5186,
193,
6,
3014,
137,
31,
```

```
 7,
 41,
 915,
 9732,
 3631,
 21,
 164,
 266,
 7,
 14]
```

```
X_train = pad_sequences(X_train,padding='post',maxlen=50)
X_test = pad_sequences(X_test,padding='post',maxlen=50)
```

```
import numpy as np
np.max(X_train)
```

```
88585
```

```
X_train.shape
```

```
(25000, 50)
```

```
model = Sequential()


model.add(Embedding(90000,output_dim=2,input_length=50))

#model.summary()
model.add(SimpleRNN(32,return_sequences=False))
model.add(Dense(1,activation='sigmoid'))

model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 50, 2)             180000

 simple_rnn (SimpleRNN)      (None, 32)                1120

 dense_5 (Dense)             (None, 1)                 33

=================================================================
Total params: 181153 (707.63 KB)
Trainable params: 181153 (707.63 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['ac
c'])

model.fit(X_train,y_train,epochs=5,validation_data=(X_test,y_test))
```

```
Epoch 1/5
782/782 [==============================] - 21s 25ms/step - loss:
0.5955 - acc: 0.6541 - val_loss: 0.4638 - val_acc: 0.7830
Epoch 2/5
782/782 [==============================] - 18s 23ms/step - loss:
0.3627 - acc: 0.8446 - val_loss: 0.4292 - val_acc: 0.8122
Epoch 3/5
782/782 [==============================] - 18s 24ms/step - loss:
0.2454 - acc: 0.9059 - val_loss: 0.4469 - val_acc: 0.8038
Epoch 4/5
782/782 [==============================] - 18s 24ms/step - loss:
0.1739 - acc: 0.9387 - val_loss: 0.6181 - val_acc: 0.7861
Epoch 5/5
782/782 [==============================] - 18s 24ms/step - loss:
0.1305 - acc: 0.9552 - val_loss: 0.6063 - val_acc: 0.7889

<keras.src.callbacks.History at 0x7ca2cc136140>
```