Implementing RNN with Keras

Muhammad Tamjid Rahman

Contents

Loading R packages

```
library(uuml)
```

1 Recurrent Neural Networks with Keras

Python packages

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import imdb
from tensorflow.keras import preprocessing
from tensorflow.keras.models import Sequential
from keras import layers
from keras import models
from tensorflow.keras.layers import Flatten, Dense, Dropout, Embedding, RNN, LSTM
from tensorflow.keras.layers import BatchNormalization
import matplotlib.pyplot as plt
```

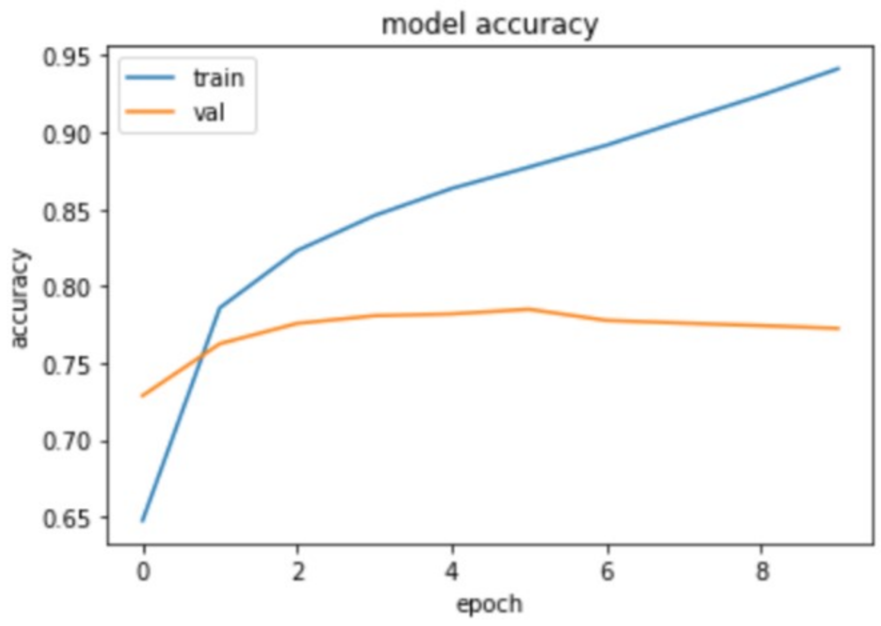
```
# Number of words to consider as features
max_features = 10000
# First 30 words(among the most frequent 10,000) would be considered of a single review
maxlen = 30
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

```
model = Sequential()
model.add(Embedding(10000, 16, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()
```

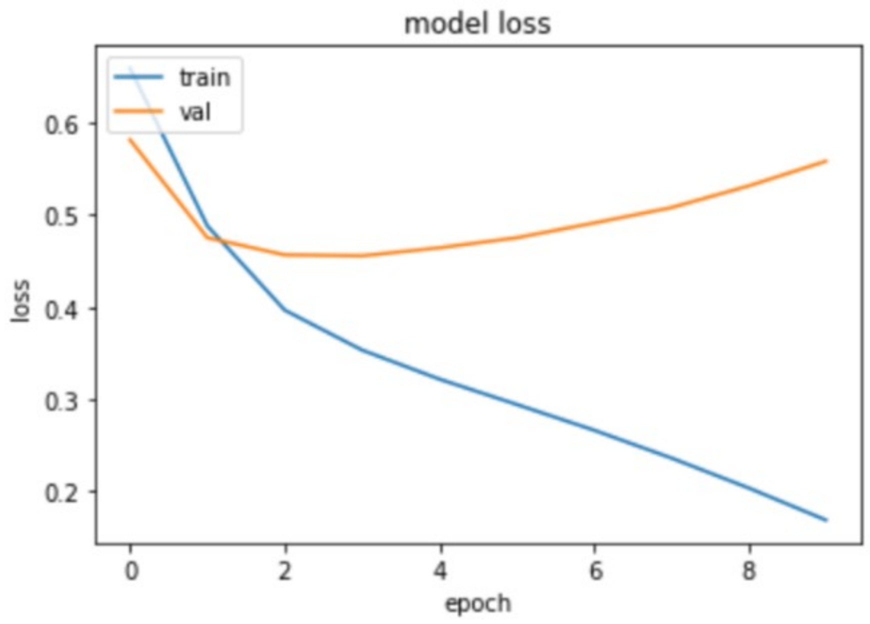
```
## Model: "sequential"
## _________________________________________________________________
##  Layer (type)                Output Shape              Param #
## =================================================================
##  embedding (Embedding)       (None, 30, 16)            160000
##
##  flatten (Flatten)           (None, 480)               0
##
##  dense (Dense)               (None, 32)                15392
##
##  dense_1 (Dense)             (None, 1)                 33
##
## =================================================================
## Total params: 175,425
## Trainable params: 175,425
## Non-trainable params: 0
## _________________________________________________________________
```

```
history = model.fit(x_train, y_train,
                    epochs=10,batch_size=128,
                    validation_split=0.2)
```

```
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



```
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



```
test_loss, test_acc = model.evaluate(x_test, y_test)
```

```
782/782 [==============================] - 2s 3ms/step - loss: 0.5501 - acc: 0.7702
```

```
test_acc
```

```
0.7702400088310242
```

```
test_loss
```

```
0.5500849485397339
```

After 1 epoch the model got overfitted and we got 77% accuracy on the test data.

2 The embedding layer

The embedding layer turns the inputs into vectors. In our case we have 16 dimensional embedding layer. Our maximum features(top most frequent words) is 10,000 and among them we took first 30 word in a single review. This layer learn 16-dimensional vector for 10,000 words. Then the 30 words of a review is mapped. So, for the embedding layer we have 16x10000 parameters and the output shape is (30,16).

3

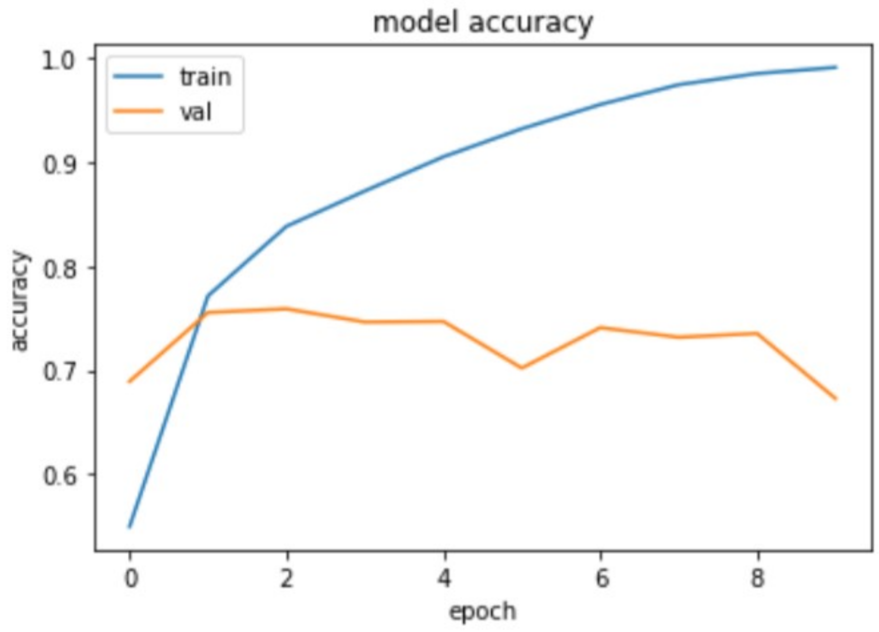
```
model = Sequential()
model.add(Embedding(10000, 16, input_length=maxlen))
model.add(layers.SimpleRNN(32))

model.add(Flatten())
model.add(Dense(32, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()
```

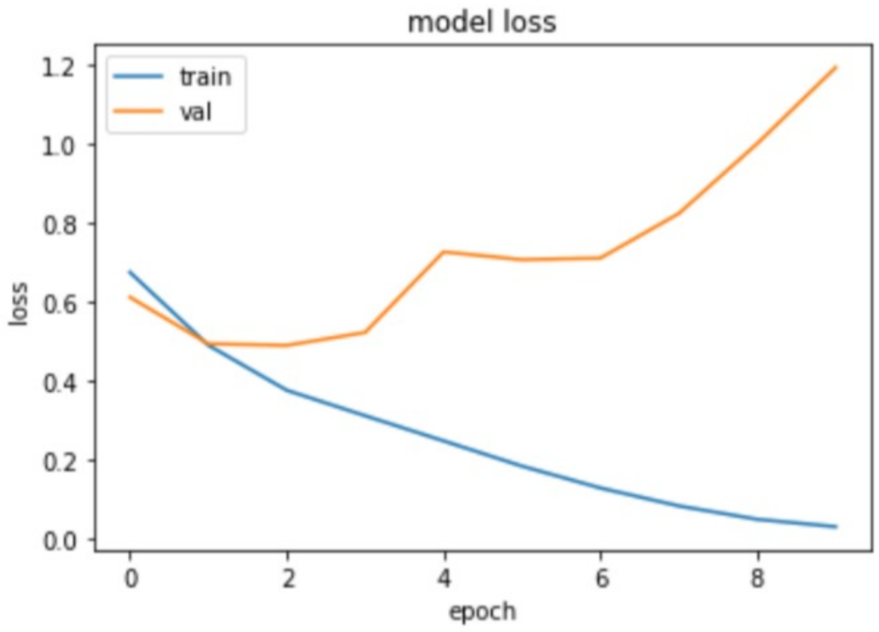
```
## Model: "sequential_1"
## _________________________________________________________________
##  Layer (type)                Output Shape              Param #
## =================================================================
##  embedding_1 (Embedding)     (None, 30, 16)            160000
##
##  simple_rnn (SimpleRNN)      (None, 32)                1568
##
##  flatten_1 (Flatten)         (None, 32)                0
##
##  dense_2 (Dense)             (None, 32)                1056
##
##  dense_3 (Dense)             (None, 1)                 33
##
## =================================================================
## Total params: 162,657
## Trainable params: 162,657
## Non-trainable params: 0
## _________________________________________________________________
```

```
history = model.fit(x_train, y_train,
                    epochs=10,batch_size=128,
                    validation_split=0.2)
```

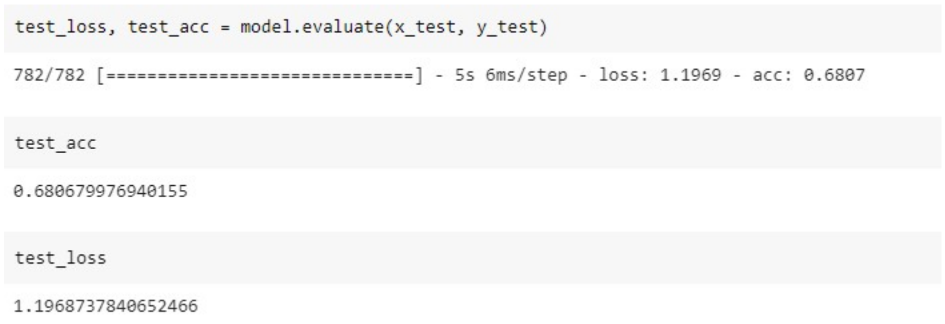
```
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



```
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



```
test_loss, test_acc = model.evaluate(x_test, y_test)
```



```
test_loss, test_acc = model.evaluate(x_test, y_test)

782/782 [==============================] - 5s 6ms/step - loss: 1.1969 - acc: 0.6807

test_acc

0.680679976940155

test_loss

1.1968737840652466
```

After 1 epoch the model got overfitted and we got 68% accuracy on the test data.

4 The recurrent network describes best

In our keras.layers.SimpleRNN, the parameter return_sequences is set False. I think Fig. 10.5 describes my implemented model best. It takes the input and produce a single output at the end of the sequence. It produces a fixed size output that is used as input for the next processing. If return_sequences is set to True, it would produce output for each time step(Fig. 10.3).[1,4]