LSTM network with TensorFlow

Muhammad Tamjid Rahman

Contents

Loading R packages

```
library(uuml)
```

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, 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)
```

1 Implementing an LSTM network

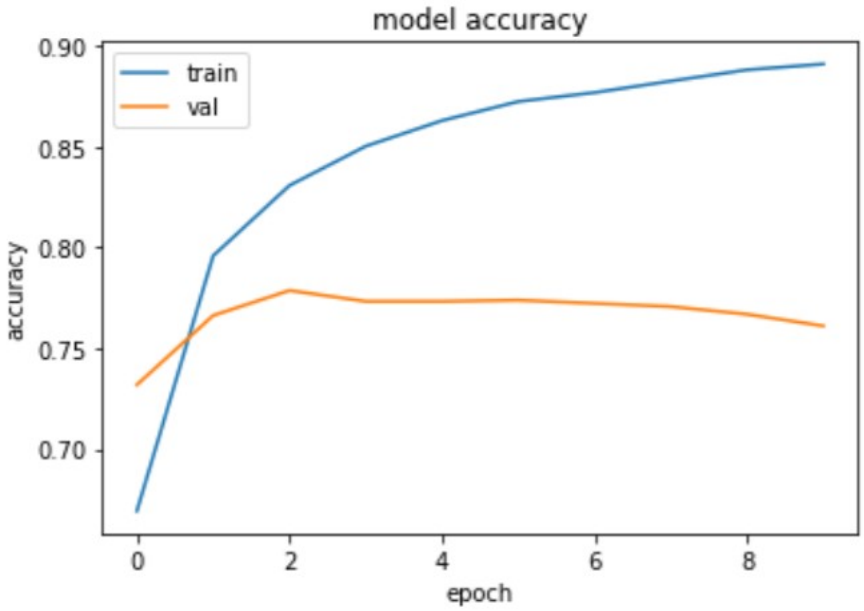
```
model = Sequential()
model.add(Embedding(10000, 16, input_length=maxlen))
model.add(layers.LSTM(32))

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()
```

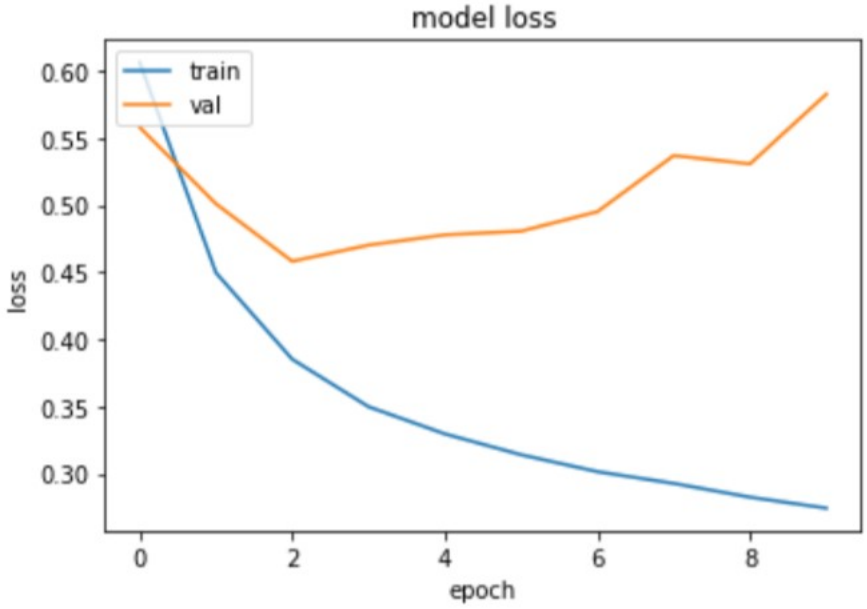
```
## Model: "sequential"
## _________________________________________________________________
##  Layer (type)                 Output Shape              Param #
## =================================================================
##  embedding (Embedding)        (None, 30, 16)            160000
##
##  lstm (LSTM)                  (None, 32)                6272
##
##  flatten (Flatten)            (None, 32)                0
##
##  dense (Dense)                (None, 32)                1056
##
##  dense_1 (Dense)              (None, 1)                 33
##
## =================================================================
## Total params: 167,361
## Trainable params: 167,361
## 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 [==============================] - 4s 5ms/step - loss: 0.5722 - acc: 0.7651
```

```
test_acc
```

```
0.7651200294494629
```

```
test_loss
```

```
0.5721971988677979
```

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

8

First Model

```
# Number of words to consider as features
max_features = 10000
maxlen = 50
(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)
```

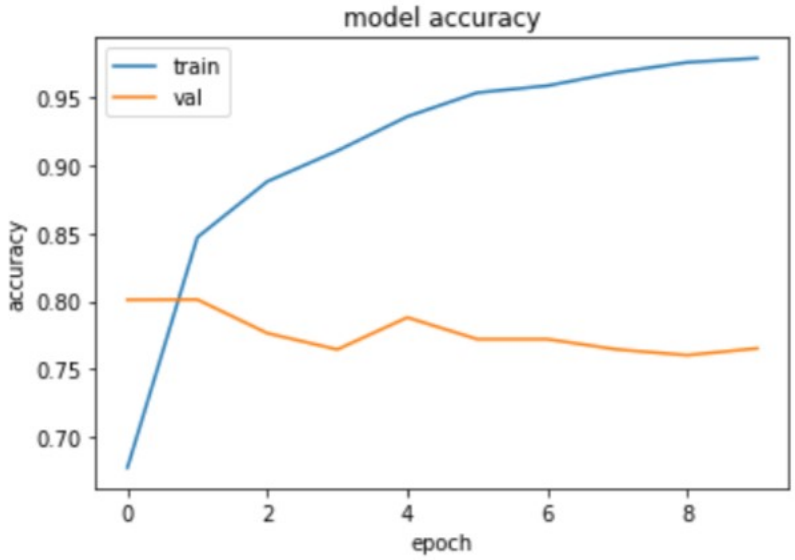
```
model = Sequential()
model.add(Embedding(10000, 32, input_length=maxlen))
model.add(layers.LSTM(64))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))
model.add(layers.BatchNormalization())
model.add(Dense(32, activation='sigmoid'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
model.summary()
```

```
## Model: "sequential_1"
## _________________________________________________________________
##  Layer (type)                 Output Shape              Param #
## =================================================================
##  embedding_1 (Embedding)      (None, 50, 32)            320000
##
##  lstm_1 (LSTM)                (None, 64)                24832
##
##  dropout (Dropout)            (None, 64)                0
##
##  flatten_1 (Flatten)          (None, 64)                0
##
##  dense_2 (Dense)              (None, 64)                4160
##
```

```
##  dropout_1 (Dropout)         (None, 64)                0
##
##  batch_normalization (BatchN  (None, 64)               256
##  ormalization)
##
##  dense_3 (Dense)             (None, 32)                2080
##
##  dropout_2 (Dropout)         (None, 32)                0
##
##  dense_4 (Dense)             (None, 1)                 33
##
## =================================================================
## Total params: 351,361
## Trainable params: 351,233
## Non-trainable params: 128
## _________________________________________________________________
```

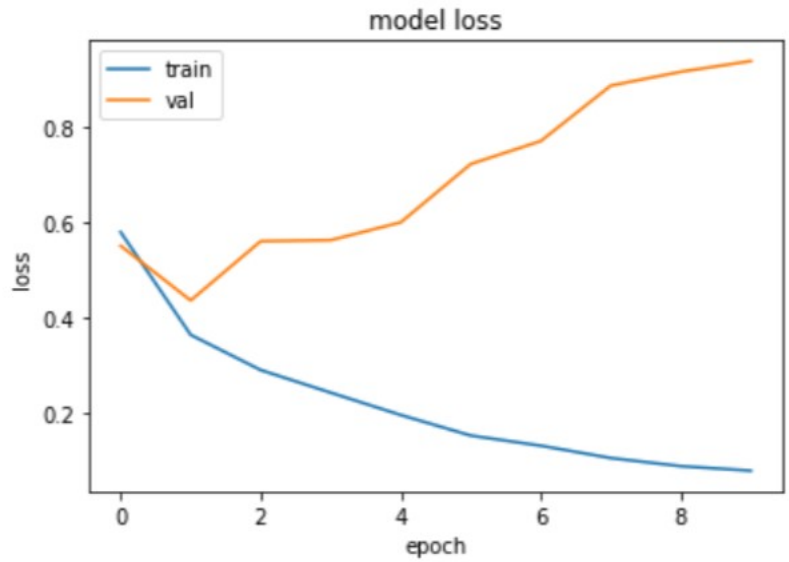
```
history = model.fit(x_train, y_train,
                    epochs=10,batch_size=128,
                    validation_split=0.2)
```

```
# 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 7ms/step - loss: 0.9155 - acc: 0.7737
```

```
test_acc
```

```
0.7736799716949463
```

```
test_loss
```

```
0.9154923558235168
```

For the first model we took maxlen=50 and we used 32 dimensional embedding and an LSTM layer with 64 units. WE also used couple of neural networks and 'adam' as optimizer. We used BatchNormalization in one neural network and use several Dropout layers. After 1 epoch the model is getting overfitted. The test accuracy is 77%.