Welcome to the Python_and_Deep_Learning_Course-CSEE5590 Deep Learning Part Lab-3 submission

`By: Mani Sai Srinivas, Kandukuri`

`Class-id: 20`

Objective

To take a data set and do text classification on three models: RNN, LSTM, CNN and anaylzie and pick the best, the next best among those three.

Details

All of the code is commented for understanding. Code can be found here

Configuration

I used Anaconda for this project

- Anaconda
- Python 3.6

Approaches/Methods

- First we import the required functions and classes.
- Now we import the IMDB movie data set which consists of movie reviews. The data set have both positive and negative reviews.
- We set the training parameters
- We define an embedding layer and proceed with it
- Now we apply CNN/RNN/LSTM based on the the model we require
- Now the model will be trained for 2 epochs as applying more may lead to over fitting
- 'Adam' which is an efficient optimization algorithm is used
- We calculate the performance based on accuracy
- We can determine the best one based on the accuracy.

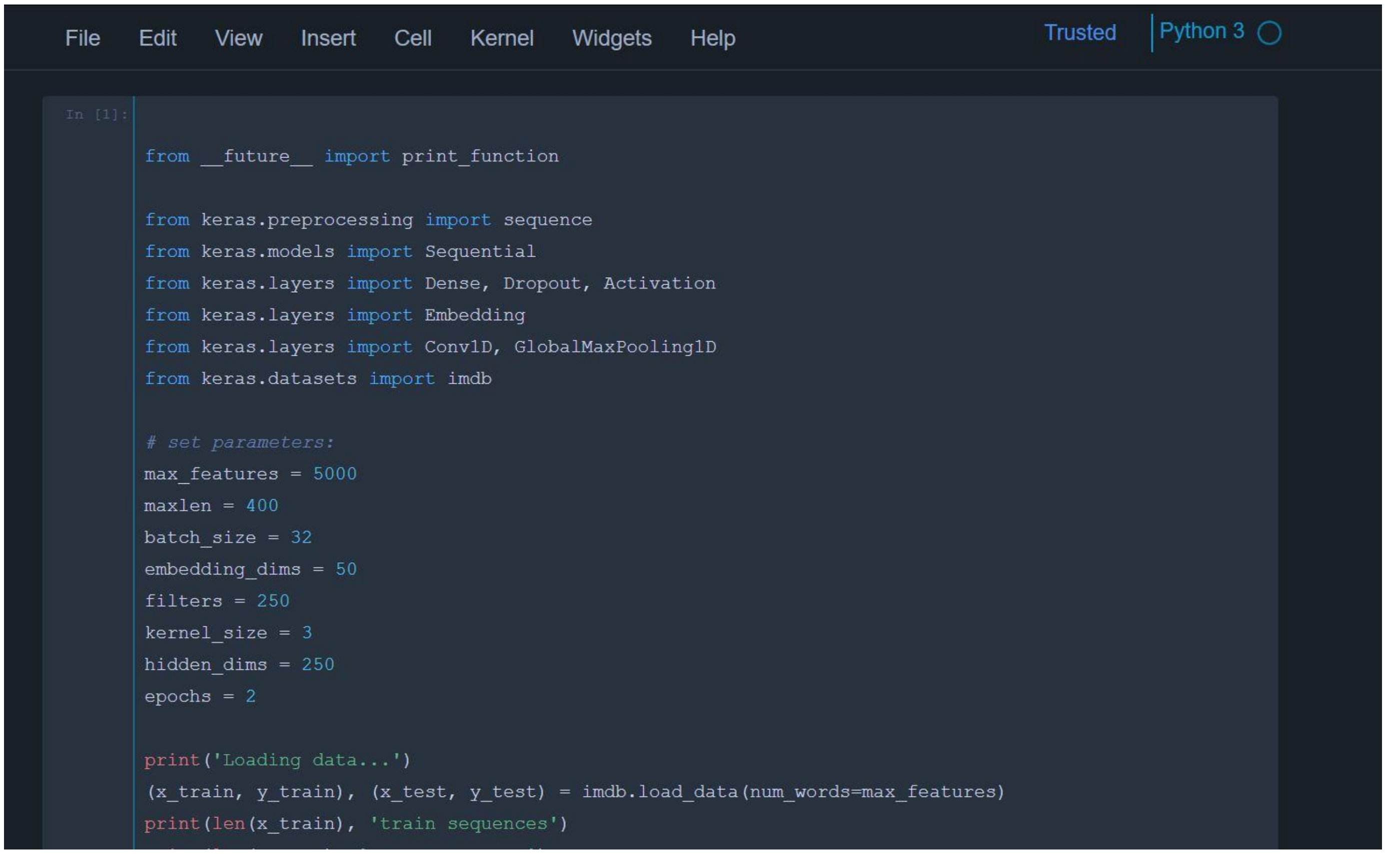
Dataset

We import the IMDB movie data set which consists of movie reviews. The data set have both positive and negative reviews, where training and test data are 25000 samples each.

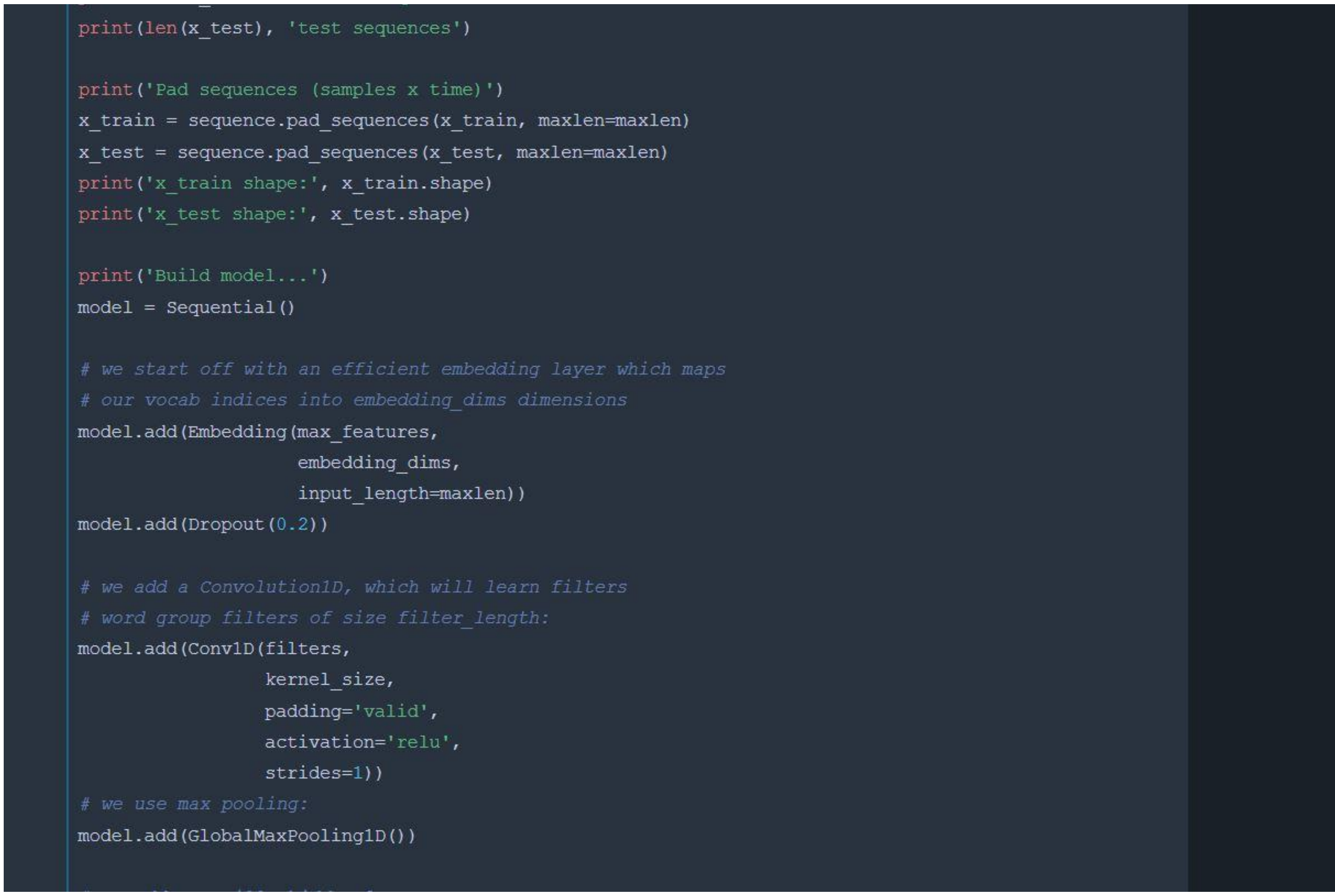
Code

Here are the screenshots of the code:

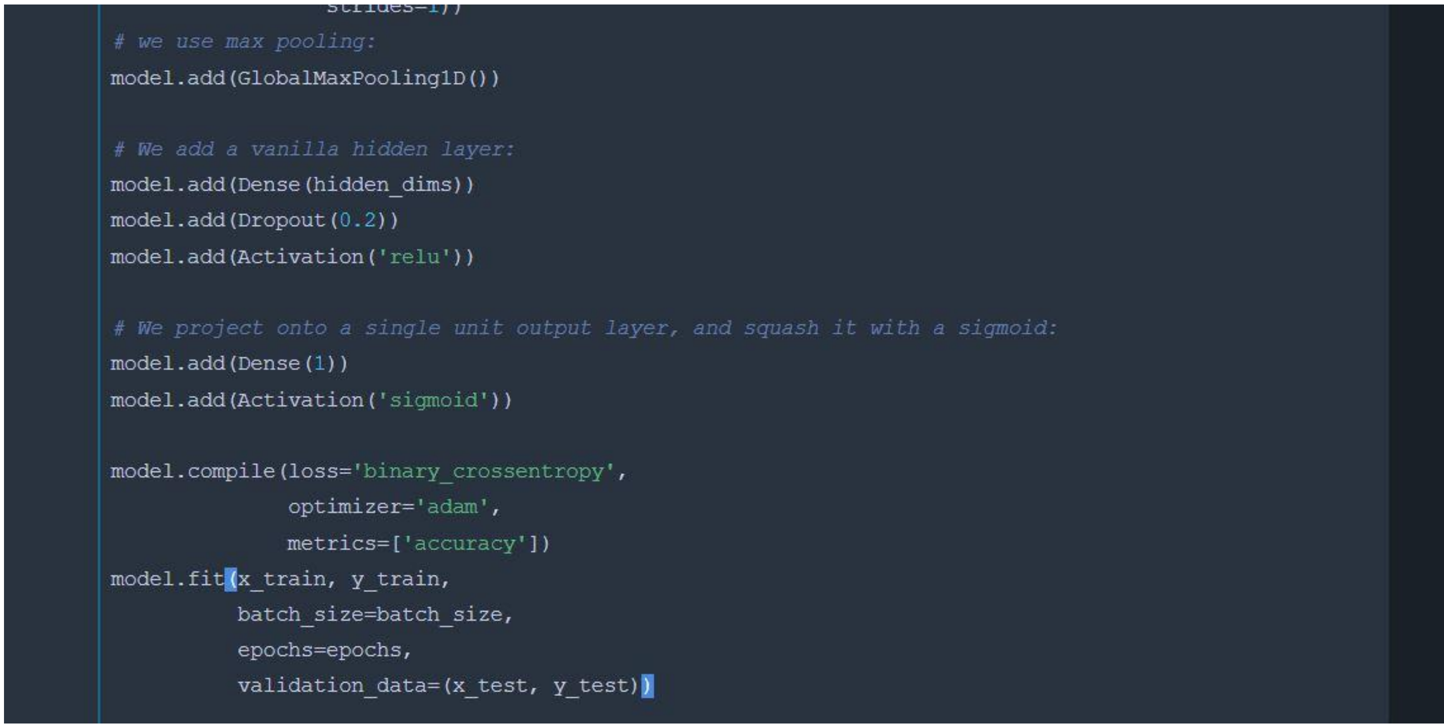
CNN:



```
from __future__ import print_function

from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.layers import Embedding
from keras.layers import Conv1D, GlobalMaxPooling1D
from keras.datasets import imdb

# set parameters:
max_features = 5000
maxlen = 400
batch_size = 32
embedding_dims = 50
filters = 250
kernel_size = 3
hidden_dims = 250
epochs = 2

print('Loading data...')
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
print(len(x_train), 'train sequences')
```



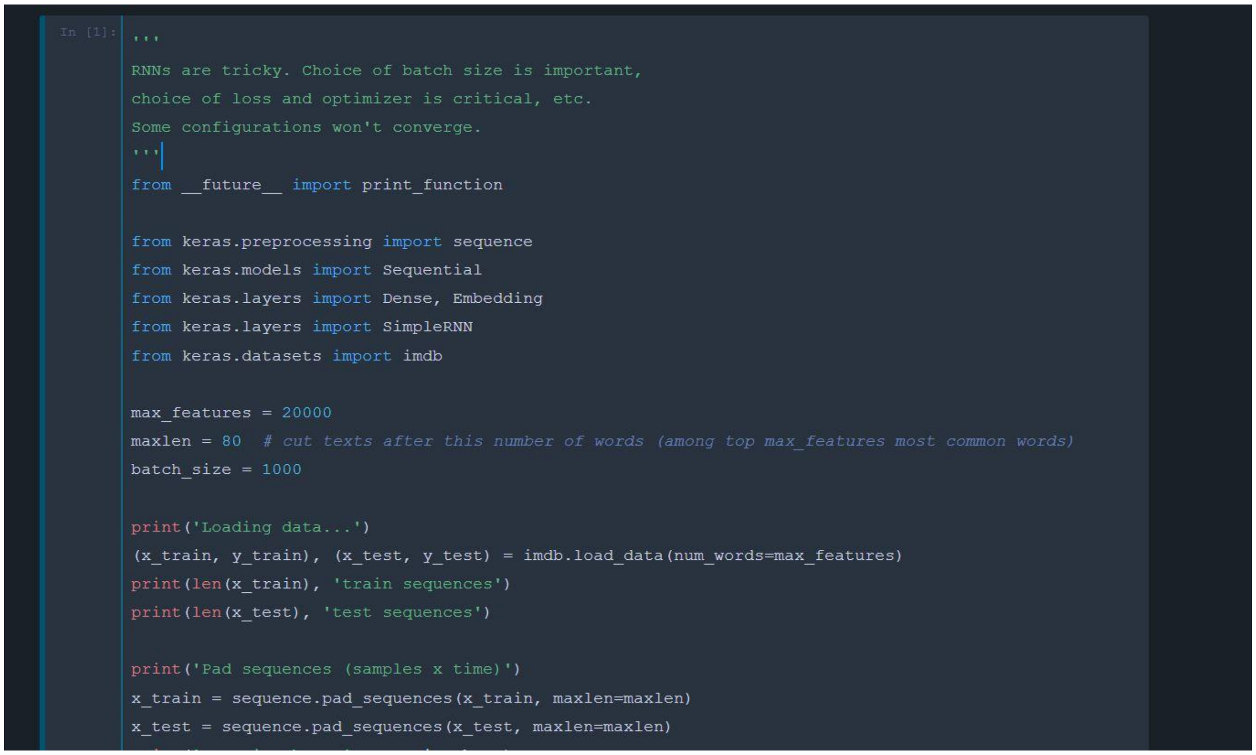
```
print(len(x_test), 'test sequences')

print('Pad sequences (samples x time)')
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)

print('Build model...')
model = Sequential()

# we start off with an efficient embedding layer which maps
# our vocab indices into embedding_dims dimensions
model.add(Embedding(max_features,
                    embedding_dims,
                    input_length=maxlen))
model.add(Dropout(0.2))

# we add a Convolution1D, which will learn filters
# word group filters of size filter_length:
model.add(Conv1D(filters,
                 kernel_size,
                 padding='valid',
                 activation='relu',
                 strides=1))
# we use max pooling:
model.add(GlobalMaxPooling1D())
```



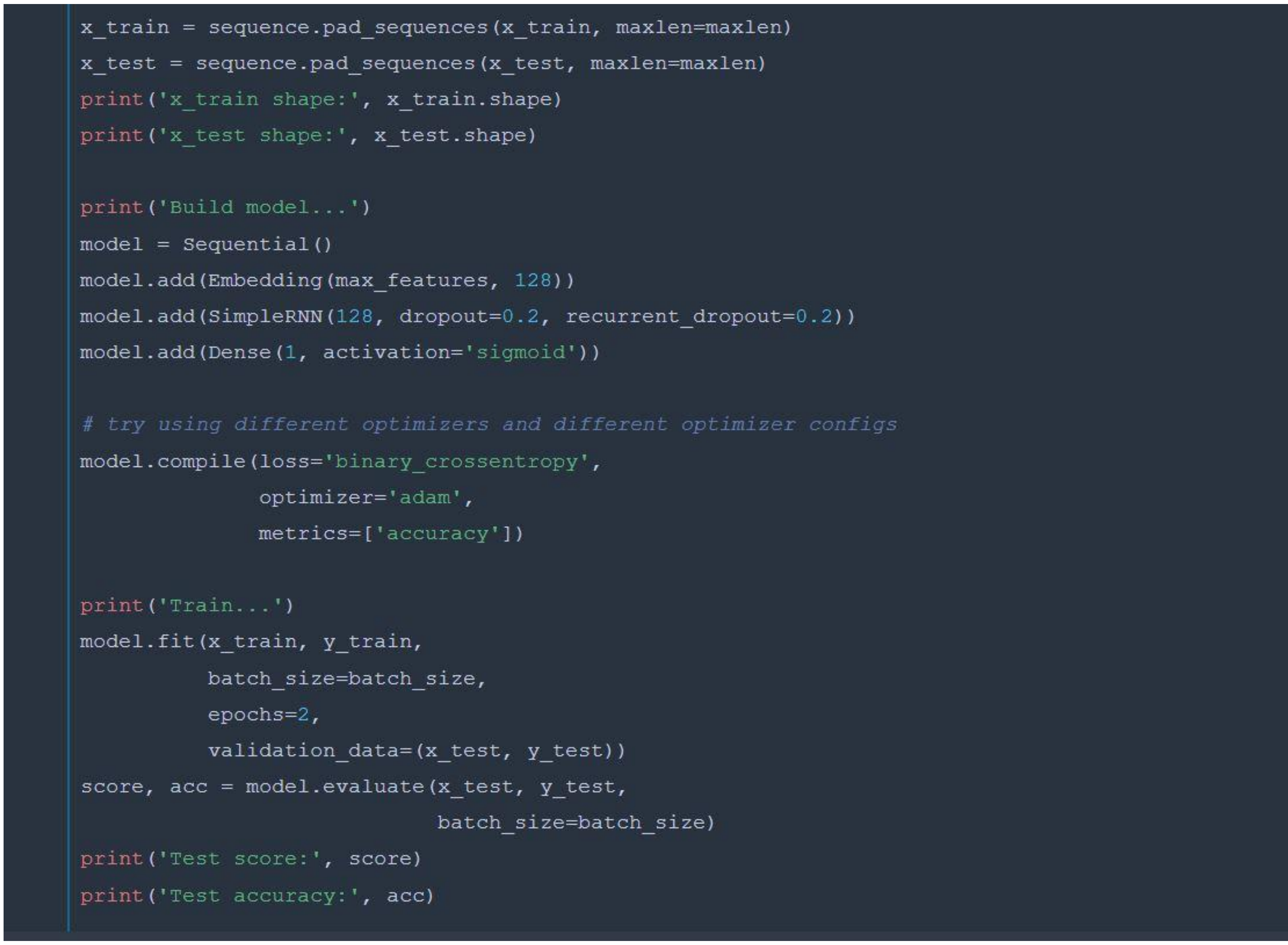
```
# we use max pooling:
model.add(GlobalMaxPooling1D())
```

```
# We add a vanilla hidden layer:
model.add(Dense(hidden_dims))
model.add(Dropout(0.2))
model.add(Activation('relu'))

# We project onto a single unit output layer, and squash it with a sigmoid:
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          validation_data=(x_test, y_test))
```

RNN:

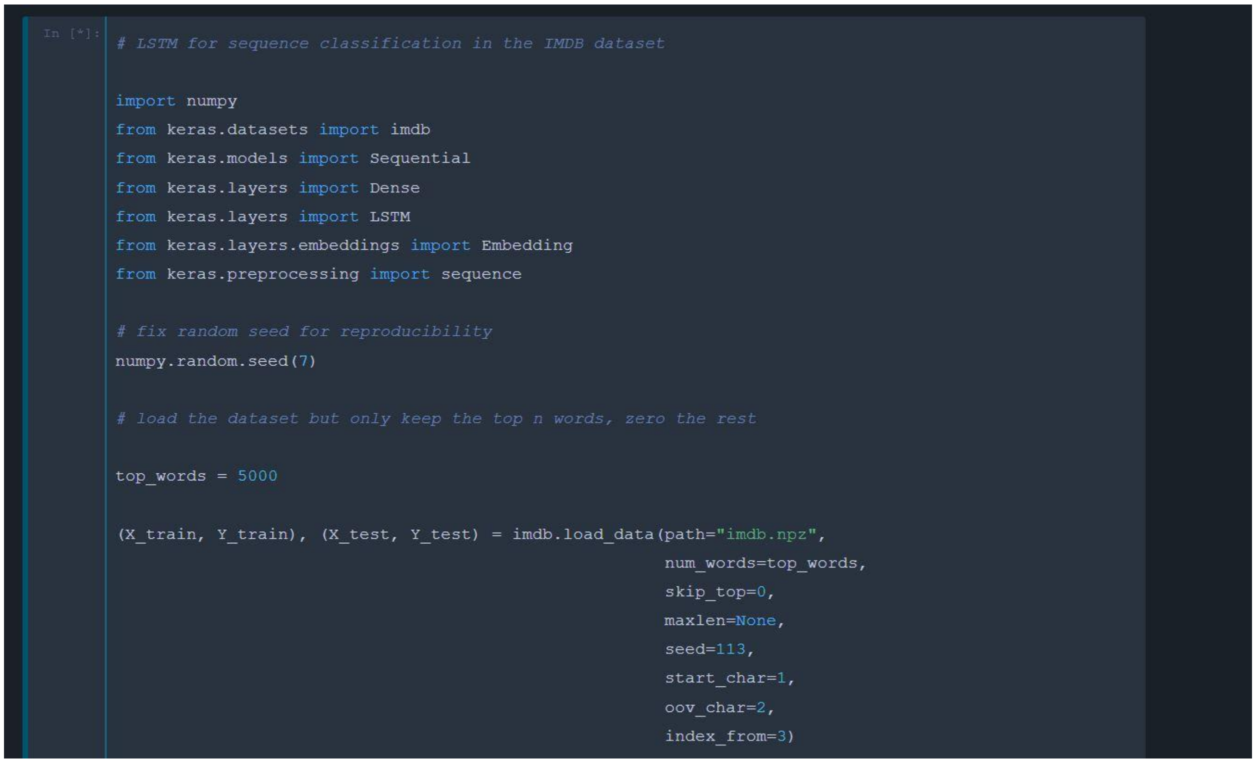


```
In [1]: '''
        RNNs are tricky. Choice of batch size is important,
        choice of loss and optimizer is critical, etc.
        Some configurations won't converge.
        '''
        from __future__ import print_function

        from keras.preprocessing import sequence
        from keras.models import Sequential
        from keras.layers import Dense, Embedding
        from keras.layers import SimpleRNN
        from keras.datasets import imdb

        max_features = 20000
        maxlen = 80  # cut texts after this number of words (among top max_features most common words)
        batch_size = 1000

        print('Loading data...')
        (x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
        print(len(x_train), 'train sequences')
        print(len(x_test), 'test sequences')

        print('Pad sequences (samples x time)')
        x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
        x_test = sequence.pad_sequences(x_test, maxlen=maxlen)
```

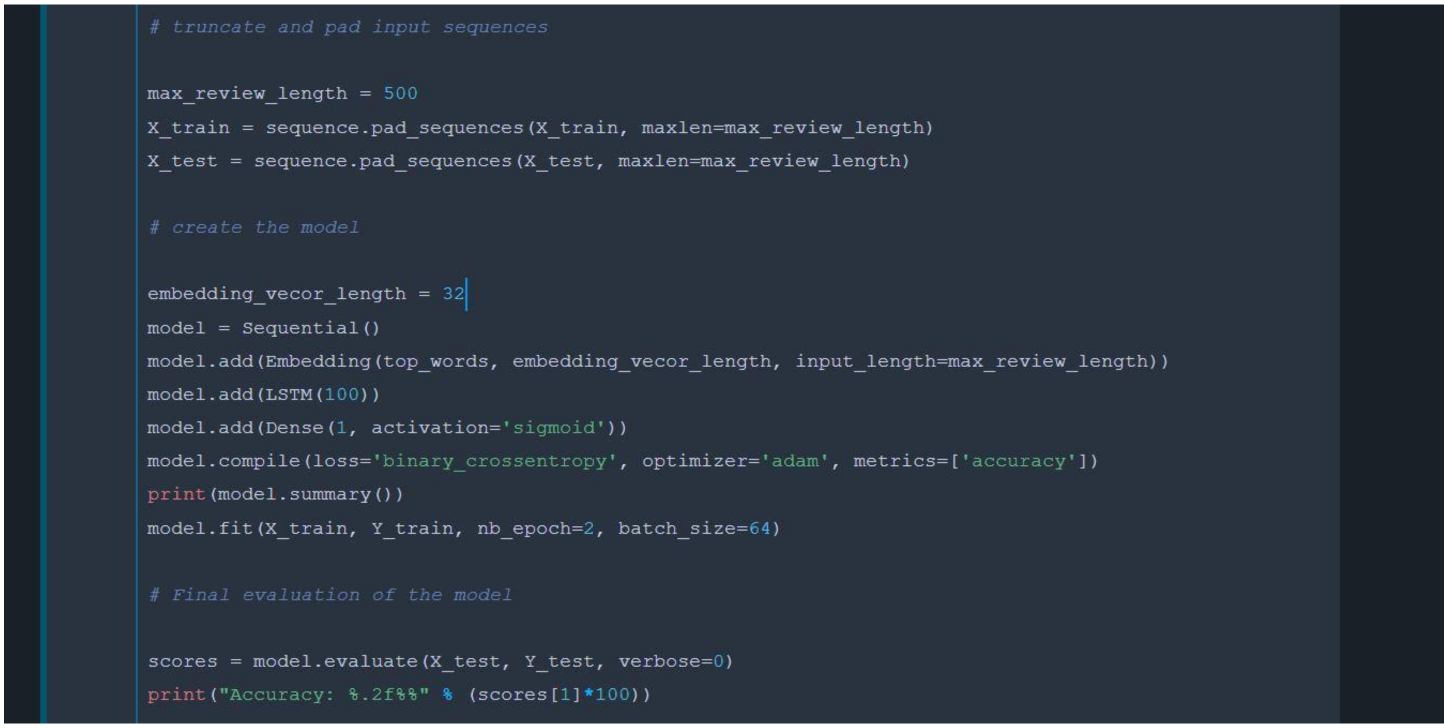


```python
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)

print('Build model...')
model = Sequential()
model.add(Embedding(max_features, 128))
model.add(SimpleRNN(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

# try using different optimizers and different optimizer configs
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

print('Train...')
model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=2,
          validation_data=(x_test, y_test))
score, acc = model.evaluate(x_test, y_test,
                            batch_size=batch_size)
print('Test score:', score)
print('Test accuracy:', acc)
```

LSTM:



```
In [*]:
# LSTM for sequence classification in the IMDB dataset

import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence

# fix random seed for reproducibility
numpy.random.seed(7)

# load the dataset but only keep the top n words, zero the rest

top_words = 5000

(X_train, Y_train), (X_test, Y_test) = imdb.load_data(path="imdb.npz",
                                                       num_words=top_words,
                                                       skip_top=0,
                                                       maxlen=None,
                                                       seed=113,
                                                       start_char=1,
                                                       oov_char=2,
                                                       index_from=3)
```

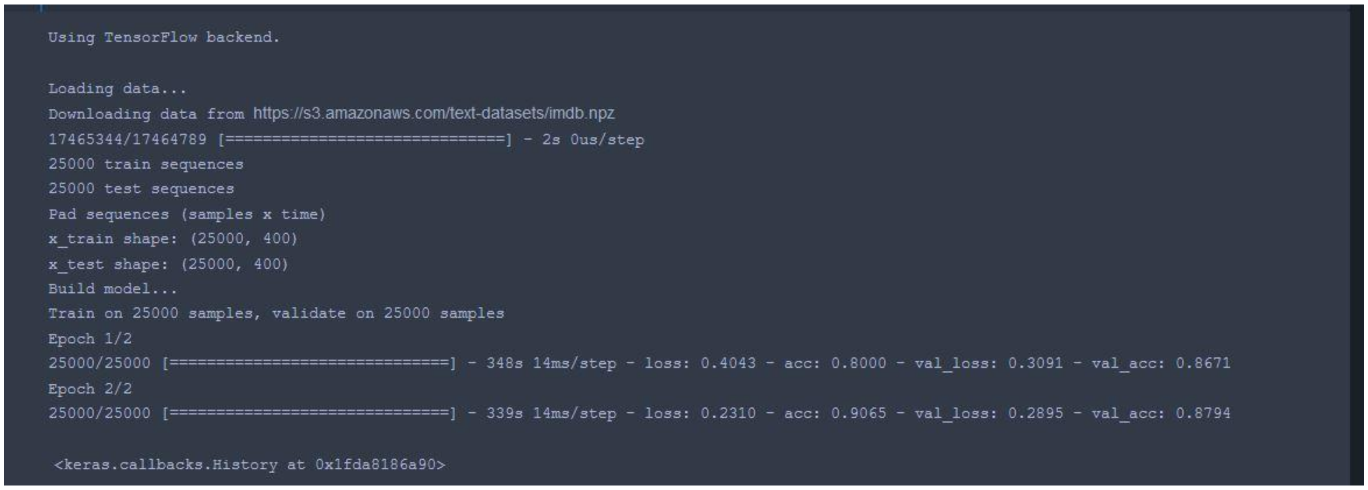


```
# truncate and pad input sequences

max_review_length = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)

# create the model

embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length, input_length=max_review_length))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
model.fit(X_train, Y_train, nb_epoch=2, batch_size=64)

# Final evaluation of the model

scores = model.evaluate(X_test, Y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

Output:

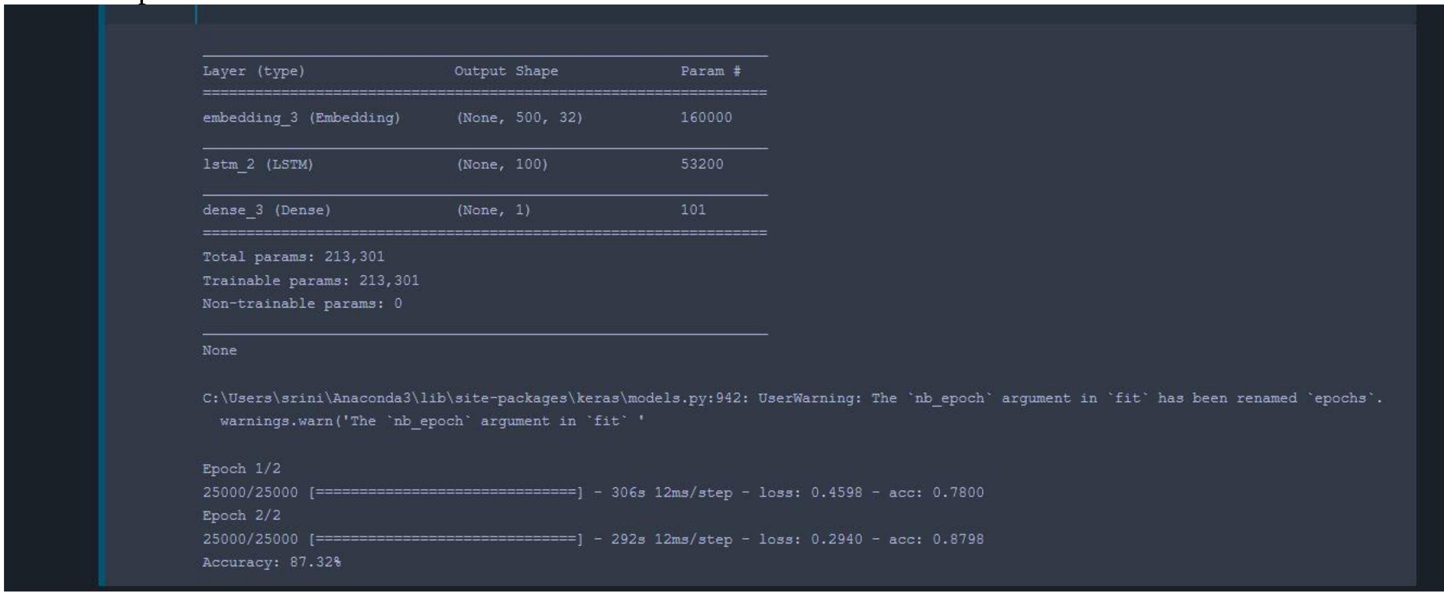
The result is:

CNN output:



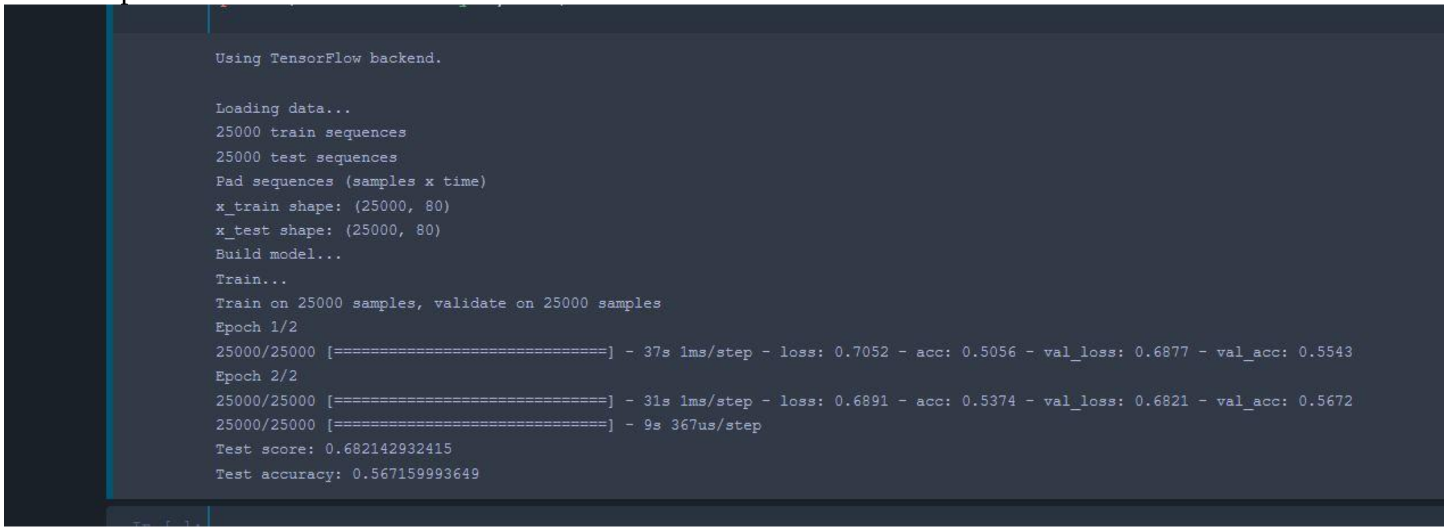
```
Using TensorFlow backend.

Loading data...
Downloading data from https://s3.amazonaws.com/text-datasets/imdb.npz
17465344/17464789 [==============================] - 2s 0us/step
25000 train sequences
25000 test sequences
Pad sequences (samples x time)
x_train shape: (25000, 400)
x_test shape: (25000, 400)
Build model...
Train on 25000 samples, validate on 25000 samples
Epoch 1/2
25000/25000 [==============================] - 348s 14ms/step - loss: 0.4043 - acc: 0.8000 - val_loss: 0.3091 - val_acc: 0.8671
Epoch 2/2
25000/25000 [==============================] - 339s 14ms/step - loss: 0.2310 - acc: 0.9065 - val_loss: 0.2895 - val_acc: 0.8794

 <keras.callbacks.History at 0x1fda8186a90>
```

LSTM output:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 500, 32)           160000
_________________________________________________________________
lstm_2 (LSTM)                (None, 100)               53200
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 101
=================================================================
Total params: 213,301
Trainable params: 213,301
Non-trainable params: 0
_________________________________________________________________
None

C:\Users\srini\Anaconda3\lib\site-packages\keras\models.py:942: UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
  warnings.warn('The `nb_epoch` argument in `fit` '

Epoch 1/2
25000/25000 [==============================] - 306s 12ms/step - loss: 0.4598 - acc: 0.7800
Epoch 2/2
25000/25000 [==============================] - 292s 12ms/step - loss: 0.2940 - acc: 0.8798
Accuracy: 87.32%
```

RNN output:



```
Using TensorFlow backend.

Loading data...
25000 train sequences
25000 test sequences
Pad sequences (samples x time)
x_train shape: (25000, 80)
x_test shape: (25000, 80)
Build model...
Train...
Train on 25000 samples, validate on 25000 samples
Epoch 1/2
25000/25000 [==============================] - 37s 1ms/step - loss: 0.7052 - acc: 0.5056 - val_loss: 0.6877 - val_acc: 0.5543
Epoch 2/2
25000/25000 [==============================] - 31s 1ms/step - loss: 0.6891 - acc: 0.5374 - val_loss: 0.6821 - val_acc: 0.5672
25000/25000 [==============================] - 9s 367us/step
Test score: 0.682142932415
Test accuracy: 0.567159993649
```

Observation

Based on the observation, (that i took 2 epochs and imdb movie review data set)

- I found that LSTM performed best whith accuracy of 87.98% in the 2nd epoch among three.
- Next comes CNN
- Rnn performed badly with accuracy of 56.72%

References:

https://stackoverflow.com/

https://www.tensorflow.org/