Natural Language Processing (NLP)

RNN/LSTM for Text Processing

Instructor: Santosh Chapaneri

Sep 2021

- The IMDB dataset) contains 25,000 highly-polar movie reviews (good or bad) for training and the same amount again for testing.
- The problem is to determine whether a given movie review has a positive or negative sentiment.
- The words have been replaced by integers that indicate the ordered frequency of each word in the dataset. The sentences in each review are therefore comprised of a sequence of integers.

- **Word Embedding**
- We will map each word onto a 32 length real valued vector. We will also limit the total number of words that we are interested in modeling to the 5000 most frequent words, and zero out the rest. Finally, the sequence length (number of words) in each review varies, so we will constrain each review to be 500 words, truncating long reviews and pad the shorter reviews with zero values.

In [ ]:

```python
import numpy as np
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense, LSTM, SimpleRNN, GRU
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
np.random.seed(2021)
```

- We are constraining the dataset to the top 5,000 words. We also split the dataset into train (50%) and test (50%) sets.

In [ ]:

```python
# 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(num_words = top_words)
```

```
<string>:6: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is
a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If
you meant to do this, you must specify 'dtype=object' when creating the ndarray
/usr/local/lib/python3.7/dist-packages/keras/datasets/imdb.py:155: VisibleDeprecationWarning: Crea
ting an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarr
ays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dt
ype=object' when creating the ndarray
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
/usr/local/lib/python3.7/dist-packages/keras/datasets/imdb.py:156: VisibleDeprecationWarning: Crea
ting an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarr
ays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dt
ype=object' when creating the ndarray
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

In [ ]:

```python
X_train.shape
```

Out[ ]:

```
(25000,)
```

In [ ]:

```python
len(X_train[2]) # vary number to see length of different samples
```

Out[ ]:

```
141
```

In [ ]:
```
X_train[2][:10]
```

Out[ ]:
```
[1, 14, 47, 8, 30, 31, 7, 4, 249, 108]
```

- We need to truncate and pad the input sequences so that they are all the same length for modeling.
- The model will learn the zero values carry no information so indeed the sequences are not the same length in terms of content, but same length vectors is required to perform the computation in Keras.

In [ ]:
```
import keras

# Retrieve the word index file mapping words to indices
word_index = keras.datasets.imdb.get_word_index()

# Reverse the word index to obtain a dict mapping indices to words
inverted_word_index = dict((i, word) for (word, i) in word_index.items())

# Decode the first sequence in the dataset
idx = 25
decoded_sequence = " ".join(inverted_word_index[i] for i in X_train[idx])
decoded_sequence
```

Out[ ]:
```
"the as it is time usual basis must has is small whole for there is works oh and most all low in a
nd they be martial and developed in long an friendly br appeal br of great this is playing and br
and and to recently in also of clearly br is save br specially past mixed or actually french mysel
f and there is copy editing like book else damon show and to it look so and finds and br and or is
dislike more he something br budget what's better of and this were and film and dave to and early
around get of every that it girl each in perfect man second some br of and film as you not like dr
ew that it see is you in own have is again older they is hell certainly way this and"
```

In [ ]:
```
# truncate and pad input sequences
max_review_length = 500

X_train_pad = sequence.pad_sequences(X_train, maxlen = max_review_length, padding = 'post')

X_test_pad = sequence.pad_sequences(X_test, maxlen = max_review_length, padding = 'post')
```

In [ ]:
```
X_train_pad.shape
```

Out[ ]:
```
(25000, 500)
```

In [ ]:

```
X_train_pad[0]
```

Out[ ]:

```
array([   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,    2,    9,   35,  480,  284,    5,  150,
          4,  172,  112,  167,    2,  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,    2,    5,
         62,  386,   12,    8,  316,    8,  106,    5,    4, 2223,    2,
         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,
          2,    8,    4,  107,  117,    2,   15,  256,    4,    2,    7,
       3766,    5,  723,   36,   71,   43,  530,  476,   26,  400,  317,
         46,    7,    4,    2, 1029,   13,  104,   88,    4,  381,   15,
        297,   98,   32, 2071,   56,   26,  141,    6,  194,    2,   18,
          4,  226,   22,   21,  134,  476,   26,  480,    5,  144,   30,
          2,   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,    2,   19,  178,   32,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0], dtype=int32)
```

In [ ]:

```
X_tr = X_train_pad[:1000]
y_tr = y_train[:1000]

X_val = X_train_pad[1000:1500]
y_val = y_train[1000:1500]

X_test = X_test_pad[:500]
y_test = y_test[:500]
```

In [ ]:

```
y_val.shape
```

Out[ ]:

```
(500,)
```

- Define, compile and fit our RNN model
- The first layer is the Embedding layer that uses 32 length vectors to represent each word.
- The next layer is the RNN layer with 100 memory units (smart neurons).
- Finally, because this is a classification problem we use a Dense output layer with a single neuron and a sigmoid activation function to make 0 or 1 predictions for the two classes (good and bad) in the problem.
- Because it is a binary classification problem, log loss is used as the loss function (binary_crossentropy in Keras).
- The efficient ADAM optimization algorithm is used.
- The model is fit for only 2 epochs because it quickly overfits the problem. A large batch size of 64 reviews is used to space out weight updates.

In [ ]:

```
from keras.layers import SimpleRNN

# create the model
embedding_veclen = 32

model = Sequential()
model.add(Embedding(top_words, embedding_veclen, input_length = max_review_length))
model.add(SimpleRNN(100))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss = 'binary_crossentropy',
              optimizer = 'adam',
              metrics = ['accuracy'])

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 500, 32)           160000
_________________________________________________________________
simple_rnn_2 (SimpleRNN)     (None, 100)               13300
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 173,401
Trainable params: 173,401
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
model.fit(X_tr, y_tr,
          validation_data=(X_val, y_val),
          epochs=3, batch_size=64)
```

```
Epoch 1/3
16/16 [==============================] - 8s 452ms/step - loss: 0.6984 - accuracy: 0.5152 - val_los
s: 0.6922 - val_accuracy: 0.5240
Epoch 2/3
16/16 [==============================] - 6s 389ms/step - loss: 0.6944 - accuracy: 0.5108 - val_los
s: 0.6930 - val_accuracy: 0.5160
Epoch 3/3
16/16 [==============================] - 6s 404ms/step - loss: 0.6814 - accuracy: 0.5676 - val_los
s: 0.6938 - val_accuracy: 0.5000
```

Out[ ]:

```
<keras.callbacks.History at 0x7ff916dd5fd0>
```

In [ ]:

```
# Final evaluation of the model

scores = model.evaluate(X_test, y_test)
scores
```

```
16/16 [==============================] - 1s 33ms/step - loss: 0.6937 - accuracy: 0.5180
```

Out[ ]:

```
[0.6936662793159485, 0.5180000066757202]
```

- **Stacked RNN**

In [ ]:

```
model_st_rnn = Sequential()
model_st_rnn.add(Embedding(top_words, embedding_veclen, input_length = max_review_length))
model_st_rnn.add(SimpleRNN(100, return_sequences = True))
model_st_rnn.add(SimpleRNN(100, return_sequences = False))
model_st_rnn.add(Dense(1, activation='sigmoid'))
model_st_rnn.compile(loss = 'binary_crossentropy',
                     optimizer = 'adam',
                     metrics = ['accuracy'])
model_st_rnn.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                Output Shape              Param #   
=================================================================
embedding_5 (Embedding)     (None, 500, 32)           160000    
                                                                 
_________________________________________________________________
simple_rnn_7 (SimpleRNN)    (None, 500, 100)          13300     
                                                                 
_________________________________________________________________
simple_rnn_8 (SimpleRNN)    (None, 100)               20100     
                                                                 
_________________________________________________________________
dense_5 (Dense)             (None, 1)                 101       
                                                                 
=================================================================
Total params: 193,501
Trainable params: 193,501
Non-trainable params: 0
                                                                 
_________________________________________________________________
```

In [ ]:

```
model_st_rnn.fit(X_tr, y_tr,
                 validation_data=(X_val, y_val),
                 epochs=3, batch_size=64)
```

```
Epoch 1/3
16/16 [==============================] - 24s 834ms/step - loss: 0.7176 - accuracy: 0.5054 - val_lo
ss: 0.7161 - val_accuracy: 0.5240
Epoch 2/3
16/16 [==============================] - 12s 777ms/step - loss: 0.6998 - accuracy: 0.5234 - val_lo
ss: 0.6947 - val_accuracy: 0.4880
Epoch 3/3
16/16 [==============================] - 12s 781ms/step - loss: 0.6884 - accuracy: 0.5279 - val_lo
ss: 0.6938 - val_accuracy: 0.4900
```

Out[ ]:

```
<keras.callbacks.History at 0x7ff91402e690>
```

In [ ]:

```
# Final evaluation of the model

scores = model_st_rnn.evaluate(X_test, y_test)
scores
```

```
16/16 [==============================] - 1s 64ms/step - loss: 0.6921 - accuracy: 0.5260
```

Out[ ]:

```
[0.6921359300613403, 0.5260000228881836]
```

LSTM

```
In [ ]: model_lstm = Sequential()
        model_lstm.add(Embedding(top_words, embedding_veclen, input_length = max_review_length))
        model_lstm.add(LSTM(100, return_sequences = False))
        model_lstm.add(Dense(1, activation='sigmoid'))
        model_lstm.compile(loss = 'binary_crossentropy',
                           optimizer = 'adam',
                           metrics = ['accuracy'])
        model_lstm.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 500, 32)           160000
_________________________________________________________________
lstm (LSTM)                  (None, 100)               53200
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 101
=================================================================
Total params: 213,301
Trainable params: 213,301
Non-trainable params: 0
_________________________________________________________________
```

```
In [ ]: model_lstm.fit(X_tr, y_tr,
                       validation_data=(X_val, y_val),
                       epochs=3, batch_size=64)
```

```
Epoch 1/3
16/16 [==============================] - 7s 70ms/step - loss: 0.6931 - accuracy: 0.5136 - val_los
s: 0.6937 - val_accuracy: 0.4940
Epoch 2/3
16/16 [==============================] - 1s 36ms/step - loss: 0.6928 - accuracy: 0.4918 - val_los
s: 0.6930 - val_accuracy: 0.4920
Epoch 3/3
16/16 [==============================] - 1s 34ms/step - loss: 0.6920 - accuracy: 0.5478 - val_los
s: 0.6938 - val_accuracy: 0.4920
```

```
Out[ ]: <keras.callbacks.History at 0x7ff90e656490>
```

```
In [ ]: # Final evaluation of the model

        scores = model_lstm.evaluate(X_test, y_test)
        scores
```

```
16/16 [==============================] - 0s 16ms/step - loss: 0.6918 - accuracy: 0.5360
```

```
Out[ ]: [0.6918402314186096, 0.5360000133514404]
```

Deep LSTM

In [ ]:

```
model_st_lstm = Sequential()
model_st_lstm.add(Embedding(top_words, embedding_veclen, input_length = max_review_length))
model_st_lstm.add(LSTM(100, return_sequences = True))
model_st_lstm.add(LSTM(100, return_sequences = False))
model_st_lstm.add(Dense(1, activation='sigmoid'))
model_st_lstm.compile(loss = 'binary_crossentropy',
                      optimizer = 'adam',
                      metrics = ['accuracy'])
model_st_lstm.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_7 (Embedding)      (None, 500, 32)           160000    
_________________________________________________________________
lstm_1 (LSTM)                (None, 500, 100)          53200     
_________________________________________________________________
lstm_2 (LSTM)                (None, 100)               80400     
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 101       
=================================================================
Total params: 293,701
Trainable params: 293,701
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
model_st_lstm.fit(X_tr, y_tr,
                  validation_data=(X_val, y_val),
                  epochs=3, batch_size=64)
```

```
Epoch 1/3
16/16 [==============================] - 5s 114ms/step - loss: 0.6939 - accuracy: 0.5086 - val_los
s: 0.6951 - val_accuracy: 0.4760
Epoch 2/3
16/16 [==============================] - 1s 58ms/step - loss: 0.6936 - accuracy: 0.5029 - val_los
s: 0.6939 - val_accuracy: 0.4760
Epoch 3/3
16/16 [==============================] - 1s 61ms/step - loss: 0.6924 - accuracy: 0.5253 - val_los
s: 0.6936 - val_accuracy: 0.4880
```

Out[ ]:

```
<keras.callbacks.History at 0x7ff9198ffd50>
```

In [ ]:

```
# Final evaluation of the model

scores = model_st_lstm.evaluate(X_test, y_test)
scores
```

```
16/16 [==============================] - 0s 26ms/step - loss: 0.6925 - accuracy: 0.5380
```

Out[ ]:

```
[0.6924556493759155, 0.5379999876022339]
```

GRU

In [ ]:

```
model_gru = Sequential()
model_gru.add(Embedding(top_words, embedding_veclen, input_length = max_review_length))
model_gru.add(GRU(100, return_sequences = False))
model_gru.add(Dense(1, activation='sigmoid'))
model_gru.compile(loss = 'binary_crossentropy',
                  optimizer = 'adam',
                  metrics = ['accuracy'])
model_gru.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_8 (Embedding)      (None, 500, 32)           160000
_________________________________________________________________
gru (GRU)                    (None, 100)               40200
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 101
=================================================================
Total params: 200,301
Trainable params: 200,301
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
model_gru.fit(X_tr, y_tr,
              validation_data=(X_val, y_val),
              epochs=3, batch_size=64)
```

```
Epoch 1/3
16/16 [==============================] - 3s 69ms/step - loss: 0.6936 - accuracy: 0.4780 - val_los
s: 0.6941 - val_accuracy: 0.4780
Epoch 2/3
16/16 [==============================] - 0s 31ms/step - loss: 0.6925 - accuracy: 0.5223 - val_los
s: 0.6942 - val_accuracy: 0.4820
Epoch 3/3
16/16 [==============================] - 0s 28ms/step - loss: 0.6909 - accuracy: 0.5617 - val_los
s: 0.6931 - val_accuracy: 0.4840
```

Out[ ]:

```
<keras.callbacks.History at 0x7ff91510c550>
```

In [ ]:

```
# Final evaluation of the model

scores = model_gru.evaluate(X_test, y_test)
scores
```

```
16/16 [==============================] - 0s 17ms/step - loss: 0.6926 - accuracy: 0.5420
```

Out[ ]:

```
[0.6925719380378723, 0.5419999957084656]
```