Natural Language Processing (NLP)

RNN/LSTM for Text Processing

Instructor: Santosh Chapaneri

Jun 2021

- Reuters dataset is a dataset of 11,228 newswires from Reuters, labeled over 46 topics.
- The problem is to classify the topic.
- The words have been replaced by integers that indicate the ordered frequency of each word in the dataset. The sentences in each sample are therefore comprised of a sequence of integers.

- **Word Embedding**
- We will map each data sample into a real vector domain, a popular technique when working with text called word embedding. This is a technique where words are encoded as real-valued vectors in a high dimensional space, where the similarity between words in terms of meaning translates to closeness in the vector space.
- Keras provides a convenient way to convert positive integer representations of words into a word embedding by an Embedding layer.
- 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 [ ]:
```
import numpy as np
from keras.datasets import reuters
from keras.models import Sequential
from keras.layers import Dense, LSTM, SimpleRNN, GRU
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
```

In [ ]:
```
# Load the dataset but only keep the top n words, zero the rest
top_words = 5000

(X_train, y_train), (X_test, y_test) = reuters.load_data(num_words = top_words)
```

```
/usr/local/lib/python3.7/dist-packages/keras/datasets/reuters.py:143: VisibleDeprecationWarning: C
reating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or nd
arrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify
'dtype=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/reuters.py:144: VisibleDeprecationWarning: C
reating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or nd
arrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

In [ ]:
```
X_train.shape
```

Out[ ]:
```
(8982,)
```

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

Out[ ]:
```
139
```

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

Out[ ]:
```
[1, 53, 12, 284, 15, 14, 272, 26, 53, 959]
```

- 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.reuters.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
```

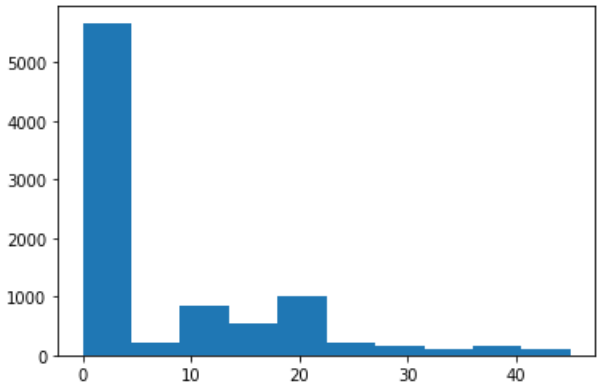
Out[ ]:
```
'the 25 oil body profits been said has would bpd directly and in crop part a in 30 gain than deale
rs conference and in has would seven jointly buffer companies may further 25 oil 50 national 7 fri
day manufacturing 28 in 30 in appears as 05 billion brazil vs been said has would bpd loss said th
an dealers it believe a in british 2 said in has would appears program note rose manufacturing a i
n public 2 said in has would appears am note shr gain development foreign 134 and regulations effe
ct been said national were speculators before been said bpd 3 from questioned expect been said nor
mal pct dlrs'
```

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 [ ]:
```
np.unique(y_train)
```

Out[ ]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45])
```

In [ ]:
```
import matplotlib.pyplot as plt

plt.hist(y_train);
```



In [ ]:
```
X_test_pad.shape
```

Out[ ]:
```
(2246, 500)
```

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

Out[ ]:
```
(8982, 500)
```

In [ ]:
```
X_train_pad[0]
```

Out[ ]:
```
array([   1,    2,    2,    8,   43,   10,  447,    5,   25,  207,  270,
          5, 3095,  111,   16,  369,  186,   90,   67,    7,   89,    5,
         19,  102,    6,   19,  124,   15,   90,   67,   84,   22,  482,
         26,    7,   48,    4,   49,    8,  864,   39,  209,  154,    6,
        151,    6,   83,   11,   15,   22,  155,   11,   15,    7,   48,
          9, 4579, 1005,  504,    6,  258,    6,  272,   11,   15,   22,
        134,   44,   11,   15,   16,    8,  197, 1245,   90,   67,   52,
         29,  209,   30,   32,  132,    6,  109,   15,   17,   12,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    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[:6000]
y_tr = y_train[:6000]

X_val = X_train_pad[6000:]
y_val = y_train[6000:]
```

In [ ]:
```
y_val.shape
```

Out[ ]:
```
(2982,)
```

- 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.
- 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(46, activation='softmax'))

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

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 500, 32)           160000
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 100)               13300
_________________________________________________________________
dense (Dense)                (None, 46)                4646
=================================================================
Total params: 177,946
Trainable params: 177,946
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
94/94 [==============================] - 57s 419ms/step - loss: 2.8499 - accuracy: 0.2863 - val_lo
ss: 2.3980 - val_accuracy: 0.3498
Epoch 2/3
94/94 [==============================] - 38s 401ms/step - loss: 2.4235 - accuracy: 0.3521 - val_lo
ss: 2.4125 - val_accuracy: 0.3498
Epoch 3/3
94/94 [==============================] - 36s 387ms/step - loss: 2.4081 - accuracy: 0.3523 - val_lo
ss: 2.3981 - val_accuracy: 0.3498
```

Out[ ]:
```
<keras.callbacks.History at 0x7f187b5b3750>
```

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

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

```
71/71 [==============================] - 2s 31ms/step - loss: 2.4085 - accuracy: 0.3620
```

Out[ ]:
```
[2.408541679382324, 0.36197686195373535]
```

- **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(46, activation='softmax'))

model_st_rnn.compile(loss = 'sparse_categorical_crossentropy',
              optimizer = 'adam',
              metrics = ['accuracy'])
model_st_rnn.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 500, 32)           160000
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 500, 100)          13300
_________________________________________________________________
simple_rnn_2 (SimpleRNN)     (None, 100)               20100
_________________________________________________________________
dense_1 (Dense)              (None, 46)                4646
=================================================================
Total params: 198,046
Trainable params: 198,046
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
94/94 [==============================] - 74s 767ms/step - loss: 2.7055 - accuracy: 0.3177 - val_lo
ss: 2.4022 - val_accuracy: 0.3498
Epoch 2/3
94/94 [==============================] - 72s 764ms/step - loss: 2.4535 - accuracy: 0.3437 - val_lo
ss: 2.4179 - val_accuracy: 0.3498
Epoch 3/3
94/94 [==============================] - 74s 789ms/step - loss: 2.3789 - accuracy: 0.3622 - val_lo
ss: 2.4357 - val_accuracy: 0.2582
```

Out[ ]:
```
<keras.callbacks.History at 0x7f187c4ced90>
```

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

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

```
71/71 [==============================] - 4s 59ms/step - loss: 2.4710 - accuracy: 0.2480
```

Out[ ]:
```
[2.470998764038086, 0.24799643456935883]
```

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(46, activation='softmax'))

model_lstm.compile(loss = 'sparse_categorical_crossentropy',
              optimizer = 'adam',
              metrics = ['accuracy'])
model_lstm.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 500, 32)           160000
_________________________________________________________________
lstm (LSTM)                  (None, 100)               53200
_________________________________________________________________
dense_2 (Dense)              (None, 46)                4646
=================================================================
Total params: 217,846
Trainable params: 217,846
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
94/94 [==============================] - 8s 50ms/step - loss: 2.9681 - accuracy: 0.3100 - val_los
s: 2.4091 - val_accuracy: 0.3498
Epoch 2/3
94/94 [==============================] - 4s 45ms/step - loss: 2.4309 - accuracy: 0.3533 - val_los
s: 2.3917 - val_accuracy: 0.3498
Epoch 3/3
94/94 [==============================] - 4s 45ms/step - loss: 2.4161 - accuracy: 0.3528 - val_los
s: 2.3994 - val_accuracy: 0.3494
```

Out[ ]:
```
<keras.callbacks.History at 0x7f187b744510>
```

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

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

```
71/71 [==============================] - 1s 13ms/step - loss: 2.4106 - accuracy: 0.3620
```

Out[ ]:
```
[2.4106383323669434, 0.36197686195373535]
```

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(46, activation='softmax'))

model_st_lstm.compile(loss = 'sparse_categorical_crossentropy',
              optimizer = 'adam',
              metrics = ['accuracy'])
model_st_lstm.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 500, 32)           160000
_________________________________________________________________
lstm_1 (LSTM)                (None, 500, 100)          53200
_________________________________________________________________
lstm_2 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense_3 (Dense)              (None, 46)                4646
=================================================================
Total params: 298,246
Trainable params: 298,246
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
94/94 [==============================] - 12s 97ms/step - loss: 2.8394 - accuracy: 0.3324 - val_los
s: 2.4068 - val_accuracy: 0.3498
Epoch 2/3
94/94 [==============================] - 8s 89ms/step - loss: 2.4252 - accuracy: 0.3515 - val_los
s: 2.4047 - val_accuracy: 0.3498
Epoch 3/3
94/94 [==============================] - 8s 89ms/step - loss: 2.4244 - accuracy: 0.3249 - val_los
s: 2.4082 - val_accuracy: 0.3498
```

Out[ ]:
```
<keras.callbacks.History at 0x7f1820d733d0>
```

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

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

```
71/71 [==============================] - 2s 27ms/step - loss: 2.4249 - accuracy: 0.3620
```

Out[ ]:
```
[2.4249348640441895, 0.36197686195373535]
```

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(46, activation='softmax'))

model_gru.compile(loss = 'sparse_categorical_crossentropy',
              optimizer = 'adam',
              metrics = ['accuracy'])
model_gru.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 500, 32)           160000
_________________________________________________________________
gru (GRU)                    (None, 100)               40200
_________________________________________________________________
dense_4 (Dense)              (None, 46)                4646
=================================================================
Total params: 204,846
Trainable params: 204,846
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
94/94 [==============================] - 7s 45ms/step - loss: 3.0790 - accuracy: 0.3078 - val_los
s: 2.3926 - val_accuracy: 0.3498
Epoch 2/3
94/94 [==============================] - 4s 41ms/step - loss: 2.3908 - accuracy: 0.3648 - val_los
s: 2.3872 - val_accuracy: 0.3498
Epoch 3/3
94/94 [==============================] - 4s 41ms/step - loss: 2.4392 - accuracy: 0.3465 - val_los
s: 2.3059 - val_accuracy: 0.2193
```

Out[ ]:
```
<keras.callbacks.History at 0x7f1818341690>
```

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

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

```
71/71 [==============================] - 1s 11ms/step - loss: 2.3365 - accuracy: 0.2110
```

Out[ ]:
```
[2.3365354537963867, 0.21104185283184052]
```