IMDB_sent_an_Keras_LSTM

August 14, 2018

1 Reproducing previous architecture with Keras

I started my project from this blog which presents an implementation based on TensorFlow basic APIs. https://www.oreilly.com/learning/perform-sentiment-analysis-with-lstms-using-tensorflow

I will reproduce in this section the architecture presented there, i.e. a simple LSTM on top of GloVes embeddings, and reuse some of the data preprocessing, but using Keras APIs instead, to see how the two compares in terms of complexity of the code and performances.

The beginning is similar to what was done before, with TensorFlow. We transform the texts into vectors of indices to be used with GloVe look-up table.

1.1 Libraries

```
In [1]: import numpy as np
        import csv
        import io
        import keras
        from matplotlib import pyplot
        from keras.datasets import imdb
        from keras.models import Sequential
        from keras.layers import Dense
        from keras.layers import Flatten
        from keras.layers.embeddings import Embedding
        from keras.preprocessing import sequence
        import pickle
```

```
Using TensorFlow backend.
```

1.2 Preprocessing and Data exploration

The Data exploration part (measuring the average number of words in the reviews) and the data preprocessing, turning texts into sequence of indexes corresponding the GloVes word embeddings, are done in another notebook called IMDB_sent_an_data_preprocessing. The variable created there are then loaded in the next sections.

1.3 Loading matrices of embedding indexes and lists of labels

The pretrained embeddins from GloVe can be downloaded here: https://nlp.stanford.edu/projects/glove/

There are different **word embedding sizes**. The possibilities are 50, 100, 200, 300. We define the one we use next.

```
In [2]: word_emb_size = '100'
```

```
In [3]: prepr_dir = '/home/aritz/Documents/CS_Programming_Machine_Learning/Projects/IMDB_sentime
```

```
In [4]: ids_train = np.load(prepr_dir+'Saved_embeddings/idsMatrixTrain'+word_emb_size+'.npy')
        ids_test = np.load(prepr_dir+'Saved_embeddings/idsMatrixTest'+word_emb_size+'.npy')
```

```
In [5]: max_seq_len = ids_train.shape[1]
```

Next we load the **labels** with and without **one-hot-encoding** ([1, 0] for positive and [0, 1] for negative).

```
In [6]: with open(prepr_dir+"y_train_ord.txt", "rb") as fp:
            y_train_ord = pickle.load(fp)

        with open(prepr_dir+"y_test_ord.txt", "rb") as fp:
            y_test_ord = pickle.load(fp)

        with open(prepr_dir+"y_train.txt", "rb") as fp:
            y_train = pickle.load(fp)

        with open(prepr_dir+"y_test.txt", "rb") as fp:
            y_test = pickle.load(fp)
```

Next we load the **list of words in the GloVe table** and a numpy array containing the **GloVe look-up table**:

```
In [7]: with open(prepr_dir+"words_list.txt", "rb") as fp:
            words_list = pickle.load(fp)

        word_vectors = np.load(prepr_dir+'word_vectors.npy')
```

1.4 Definition of the model

```
In [8]: model = Sequential()
```

I have a little doubt about the `input_dim` argument of `Embedding`: I think that in our case it is equal to the length of `words_list` above because it is described on the official page as the length of the vocabulary:

but in this tutorial

they add one to the length of the vocabulary. I guess it is because in the later case, they have to add one token for the padding symbol.

```
In [9]: model.add(Embedding(input_dim=len(words_list),
                            output_dim=int(word_emb_size),
                            weights=[word_vectors],
                            input_length=max_seq_len,
                            trainable=False,
                            mask_zero=True))
```

```
In [10]: from keras.layers import LSTM
```

```
In [11]: lstm_units = 64
         keep_prob = 0.7
         model.add(LSTM(lstm_units, dropout=1-keep_prob, recurrent_dropout=0.1))
```

```
In [12]: model.add(Dense(1, activation='sigmoid'))
```

```
In [13]: model.compile(loss='binary_crossentropy',
                       optimizer='adagrad',
         metrics=['accuracy'])
```

1.5 Training and Evaluation

```
In [14]: batch_size = 100
```

```
In [15]: print('Train...')
         model.fit(ids_train, y_train_ord,
                   batch_size=batch_size,
                   epochs=1,
                   validation_data=(ids_test, y_test_ord))
         score, acc = model.evaluate(ids_test, y_test_ord,
                                     batch_size=batch_size)
```

```
Train...
Train on 25002 samples, validate on 25001 samples
Epoch 1/1
25002/25002 [==============================] - 134s 5ms/step - loss: 0.6174 - acc: 0.6546 - val_
25001/25001 [==============================] - 34s 1ms/step
```

```
In [16]: acc
```

```
Out[16]: 0.7572897064757821
```