Deep Learning

Bastien BRIER
bastien.brier@student.ecp.fr

March 5, 2017

Final Assignment NLP

1 Word and sentences embeddings with word2vec

To begin, just a precision about the versions of gensim and keras we used.
- gensim : 0.13.3
- keras : 1.2.0

1.1 Question 1

From the info shown while training, we can see that:
1. There are 17,005,207 raw words in the corpus.
2. 16,718,844 of them are retained in the word2vec vocabulary after min_count = 5.

1.2 Question 2

1. We load the two given models, and compute the similarities between different words:
   - 'apple' and 'mac' : 0.5679
   - 'apple' and 'peach' : 0.1784
   - 'banana' and 'peach': 0.6887

In my opinion, we are asked about these words to show that context is really important for the algorithm to understand the word. The word 'apple' is both a fruit and the brand that has created MacBooks. And, as this word is more used as the brand name, the algorithm does not seem to acknowledge the fact that this is also a fruit, the similarity score between 'apple' and 'peach' being very low.

2. We compute the closest word of 'difficult' with the two models:
   - model: 'easy' : 0.7629
   - model_phrase: 'very difficult' : 0.8704

From these similarities, we can perceive the difference between the models. Model gives the most similar words, whereas model_phrase computes the most similar group of words (phrases) to the word we are looking for.
Then, we can compute the three most similar phrases to the word 'clinton':
- 'bush' : 0.8667
- 'reagan' : 0.8594
- 'gore' : 0.8575

3. We now look for the closest word to the vector "vect(france) - vect(germany) + vect(berlin)":
- 'paris' : 0.7577

4. We explored a bit the embedding space and found some interesting behaviors.
For instance, hollywood is here much more correlated with the movie industry and the algorithm has trouble relating it as a neighborhood of Los Angeles:
- 'hollywood' and 'batman' : 0.8004
- 'hollywood' and 'los_angeles' : 0.5804

In another subject, the vector "vect(napoleon) - vect(france) + vect(germany)" outputs:
- 'hitler' : 0.6917

I also explored if relationships between music bands were known and the result is not that bad:
- 'rolling_stones' and 'pink_floyd' : 0.8257
- 'rolling_stones' and 'beatles' : 0.8029
- 'rolling_stones' and 'led_zeppelin' : 0.7991

1.3 Question 3

We now want to construct sentence embeddings from word embeddings, by simply taking the mean of the word embeddings. The vector is computed as :

$$avgword2vec(s) = \frac{\sum_{i=1}^{S} \alpha_i word2vec(\omega_i)}{\sum_{i=1}^{S} \alpha_i}$$

1. We created the function cosine_similarity that we define as :

$$cosine\_similarity(a, b) = \frac{\sum_{i=1}^{S} a_i b_i}{\sqrt{\sum_{i=1}^{S} a_i^2}\sqrt{\sum_{i=1}^{S} b_i^2}}$$

With this definition, we computed the closest sentence to the sentence with idx "777" : "a girl trying to get fit for a bodybuilding competition". And the result is:
- "gymnasts get ready for a competition" : 0.9029

2. We filled the blanks of the most_5_similar function and computed then the 5 closest sentences to the same sentence as the question above :

- "gymnasts get ready for a competition" : 0.9029
- "a woman is getting ready to perform a song for the audience" : 0.8901
- "a runner in a competition want to go to the finish line" : 0.8555
- "men working to build a fence for customers" : 0.8515
- "a man prepares to give a speech for a television audience" : 0.8495

1.4 Question 4

1. We now use a new technique and define the weights α_i as the IDF (Inverse Document Frequency) of each word. IDF is defined as:

$$idf(t, D) = log \frac{N}{|\{d \in D : t \in d\}|}$$

N is the total number of documents, and $|\{d \in D : t \in d\}|$ is the total number of documents where t appears.
We compute the IDF scores of different words:

- "the" : 0.6631
- "a" : 0.1018
- "clinton" : this word is not in the corpus

2. We now compute a new function taking as weights the IDF we just calculated:

$$IDF\_avgword2vec(s) = \frac{\sum_{i=1}^{S} idf(\omega_i) \cdot word2vec(\omega_i)}{\sum_{i=1}^{S} idf(\omega_i)}$$

The closest sentence to sentence with idx 777 is now:

- "gymnasts get ready for a competition" : 0.8961

2 Simple LSTM for sequence classification

2.1 Question 1

The objective of this part is to train an LSTM for sequence classification. Here are the different shapes asked:

1. input of the embedding layer : 2D tensor with shape (nb_samples, sequence_length), here (32, 80).

2. input of the LSTM layer : 3D tensor with shape (nb_samples, timesteps, input_dim), here (32, 80, 32).

3. output of the LSTM layer : 2D tensor with shape (nb_samples, output_dim), in our case (32, 64).

2.2 Question 2

From the output of the lstm_imdb.py script, we can see that there are:

- 504,897 total parameters
- 35,000 sentences in the training set

We have in this case much more parameters than number of samples. For our analysis to be possible, we have to bet on sparsity, which means that in high dimension, most of the variables will not be significant. We then have to regularize and chose variables, using techniques like lasso (L1-regularization) for instance.

2.3 Question 3

The exact form of $f(h_1, ..., h_T)$ in the python script is an average pooling of the outputs of the T memory cells of the LSTM.
The output is computed as follows:

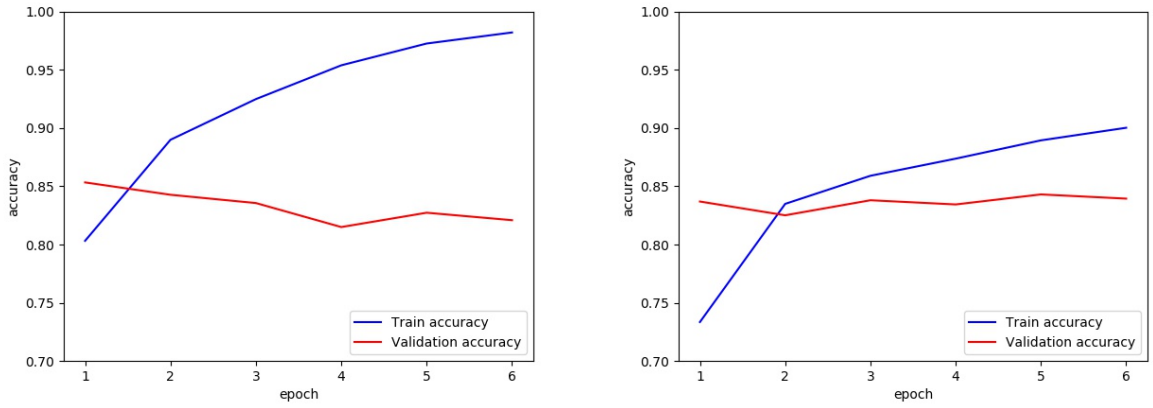
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$
$$h_t = o_t * tanh(C_t)$$

With C_t being the memory cell state at time t, x_t being t^{th} input, W_o and U_o are weight matrices respectively associated to the current input and the previous output.
Source : Keras documentation.

2.4 Question 4

We run the model with and without dropout and computed the various training and validation accuracy by epoch.

Figure 1: Accuracy per epoch: without dropout (left) and with dropout (right)



We noted also the test accuracy and loss of both models:

	Accuracy	Loss
Without dropout	0.8229	0.6162
With dropout	0.8395	0.3671

We see that without dropout, the accuracy is much higher in the training set (goes up to 0.98), but when it comes to validation or test, the result is worst than with dropout. We can then say that dropouts prevent overfitting and give better results when it comes to validating and testing our model.

2.5 Question 5

Stochastic Gradient Descent (SGD) updates each parameter by subtracting the gradient of the loss multiplied by learning rate. Adaptative Moment Estimation (Adam) calculates a moving average of the first and second order moments that are also exponentially decayed. The main difference resides here, as Adam takes into account more information and keeps memory of previous states.

3 Simple ConvNet for sequence classification

3.1 Question 1

We ran the script cnn_imdb.py and here are the results obtained:

- Test accuracy : 0.8389
- Test score : 0.3638

3.2 Question 2

The difference with the previous section is that here we trained a Convolution1D. We are asked to give the shapes of the input and output:

- input of Convolution1D : 3D tensor with shape (samples, steps, input_dim), in this case (32, 80, 16).
- output of Convolution1D : 3D tensor with shape (samples, new_steps, nb_filter), here (32, 78, 250).

3.3 Question 3

We created a new script, cnn_lstm_imdb.py, where we feed the LSTM with the output of our ConvNet (see code for more details). We then tried different combinations of parameters and here are the best results we obtained:

- Parameters : embed_dim = 16, nhid = 396, nb_filter = 250, filter_length = 3, hidden_dims = 750, epoch = 6, dropout = 0.3
- Test accuracy : 0.8555
- Test score : 0.3321

We improved our accuracy by nearly 0.02 and score by 0.03, which is clearly not negligible, as a slight progress in the performance of the algorithm can yield much better results when used with a lot of data.