Miniproject 2: Deep Learning for Natural Language Processing

Yosri SAKLY

January 3, 2019

1 Monolingual embeddings

1.1 Word2Vec

I completed the code of the class Word2Vec. I had first to add a function score taking as input two words and returning the cosine similarity between this two words using a pretrained loaded embedding. Then I implemented the function most similar that returns the most similar words in the loaded pretrained lookup table. Then, I computed the score between the given pairs of words. The scores are in the following table :

Word 1	Word 2	Score
cat	dog	0.671683666279249
dog	pet	0.6842064029669219
dogs	cats	0.7074389328052403
paris	france	0.7775108541288561
germany	berlin	0.7420295235998392

Table 1: scores.

We can observe that the pairs are relatively close using the cosine similarity based score. That's expected.

We give below the most similar words for a given set of word:

The results are logical and fair.

1.2 BoV

Here we compare sentences using some simple sentence embeddings (mean or idf-weighted mean of the word embeddings present in the sentence).

I completed the code of the class BoV. I coded a function computing the idf score as a dictionary for all the words in our vocabulary given a list of sentences. I coded a function encoding a sentence by taking the mean (or idf-weighted mean given the idf dictionary) of the words that are in the lookup table. I coded also a function that take as input a sentence s and a set of sentences and returns the most similar sentences of s present in the set.

We find the following :

The 5 most similar of "1 smiling african american boy ." using mean are :

1) 1 smiling african american boy .

2) girl smiling on roller coaster .

Word	5 similar words
cat	'cat', 'cats', 'kitty', 'kitten', 'feline'
dog	'dog', 'dogs', 'puppy', 'Dog', 'doggie'
dogs	'dogs', 'dog', 'pooches', 'Dogs', 'doggies'
paris	'paris', 'france', 'Paris', 'parisian', 'london'
germany	'germany', 'austria', 'europe', 'german', 'berlin'

Table 2: 5 similar words.

3) a boy smiles underwater .

4) two girlfriends smiling .

5) a smiling child swims .

The 5 most similar of "1 smiling african american boy ." using idf-weighted mean are :

1) 1 smiling african american boy .

2) 5 women and 1 man are smiling for the camera .

3) 2 guys facing away from camera , 1 girl smiling at camera with blue shirt , 1 guy with a beverage with a jacket on .

4) two girlfriends smiling .

5) 1 man singing and 1 man playing a saxophone in a concert .

The results depend on the way we aggregated the words embeddings in the sentence. Using the basic mean averaging of the word embedding, 4 out of 5 of the outcomes contain the word "smiling" that is actually present in quiet a lot of sentences. The context is not taken into account.

Using idf-weighted mean we can notice that the outcomes are not very relevant.

The word embeddings averaging method is not that performant because we are forgetting a lot of information.

2 Multilingual word embeddings

2.1 Theory

We have to solve the optimization problem :

$$W^* = argmin_{W \in O_d(\mathbb{R})} \|WX - Y\|_F$$

This is equivalent to :

$$\begin{aligned}
&= argmin_{W \in O_d(\mathbb{R})} \|WX - Y\|_F^2 \\
&= argmin_{W \in O_d(\mathbb{R})} \|WX\|_F^2 + \|Y\|_F^2 - 2 < WX, Y > \\
&= argmin_{W \in O_d(\mathbb{R})} \|X\|_F^2 + \|Y\|_F^2 - 2 < WX, Y > \\
&= argmin_{W \in O_d(\mathbb{R})} - 2 < WX, Y > \\
&= argmax_{W \in O_d(\mathbb{R})} tr(WXY^T)
\end{aligned}$$

We introduce the SVD decompositions :

$$SVD(W) = U_w \Sigma_w V_w$$

$$SVD(YX^T) = U \Sigma V$$

The optimization problem becomes maximizing over $W \in O_d(\mathbb{R})$ the quantity:

$$\begin{aligned}
tr(WXY^T) &= tr(U_w \Sigma_w V_w (U \Sigma V)^T) \\
&= tr(U_w \Sigma_w V_w V \Sigma U^T) \\
&= tr((U^T U_w) \Sigma_w (V_w V) \Sigma) \\
&\leq tr(\Sigma_w \Sigma)
\end{aligned}$$

by applying the Von Neumann Theorem which states that for any real matrices $A = W$ and $B = XY^T$ with descending singular values $\sigma_i(A)$ and $\sigma_i(B)$, we have:

$$tr(A^T B) \leq \sum_i \sigma_i(A) \sigma_i(B)$$

The quantity is maximized for $U^T U_w = I$ and $VV_w = I$ which yields $U_w = U$ and $V_w = V^T$ hence $W = U \Sigma_w V_T$ with the contraint $W^T W = I$. The maximizer is then $W = UV^T$.

2.2 Code

We find the following results:

From French to English

Most similar words for fort are : ['fort', 'forts', 'fortification', 'blockhouse', 'fortifications']

Most similar words for chat are : ['cat', 'rabbit', 'hamster', 'feline', 'poodle']

Most similar words for manger are : ['eat', 'meal', 'eating', 'eaten', 'ate']

Most similar words for chien are : ['dog', 'poodle', 'terrier', 'dogs', 'spaniel']

Most similar words for toit are : ['roof', 'roofed', 'roofs', 'mansard', 'balconies']

Most similar words for après are : ['after', 'shortly', 'afterward', 'afterwards', 'ensuing']

From English to French

Most similar words for real are : ['real', 'reality', 'personal', 'réel', 'property']

Most similar words for learning are : ['learning', 'education', 'apprentissage', 'apprentissages', 'educational']

Most similar words for dog are : ['dog', 'chien', 'hound', 'chiens', 'chienne']

Most similar words for feeling are : ['sentiment', 'feeling', 'mélancolie', 'sentiments', 'frustration']

Most similar words for tedious are : ['fastidieux', 'pénible', 'compliqué', 'pénibles', 'inutilement']

Most similar words for actually are : ['would', 'should', 'could', 'that', 'really']

The outcome is quite good and could be really usefull.

3 Sentence classification with BoV

In this section the goal is to perform fine-grained sentiment analysis based on the Stanford Sentiment Treebank. We have to classify each sentence into 5 classes.

3.1 code

We create an embedding for each sentence using the encoder developed before, we can directly feed a simple classifier on that embedding in order to classify each sentence. The goal of this part was to trained a Logistic Regression Classifier based on the embeddings developed in 2.

We find the following scores:

train set without idf precision 0.48408239700374533

dev set without idf precision 0.407811080835604

train set with idf precision 0.47120786516853935

dev set with idf precision 0.4014532243415077

LogisiticRegression is a really simple classifier. I tried many other classifiers in order to see if we could obtain better outcomes using some other classifier. But I noticed that almost all classifiers don't allow to significatly outperform the Logistic Regression. We give here the example of LGBM:

train set without idf precision 0.9774110486891385

dev set without idf precision 0.4069028156221617

train set with idf precision 0.9465121722846442

dev set with idf precision 0.36966394187102636

4 Sentence classification with LSTMs in Keras

In this section we are trying to solve the same problem as the one described previously but now we will try to build a Deep Neural network architecture. In this Deep Neural Network we will learn an embedding (a lookup table) to embed the words that are in the sentences (instead of using the mean of the pretrained word embeddings).

First we use a really simple NN. The network takes as input the lookup table in order to transform words in their vectors, then we use a LSTM in order to concatenate the information present in all the embeddings of the words present in the sentence. We will then feed a dense layer with a sigmoid activation function. In order to perform the optimization, I will use the $categroical_crossentropy$ loss and rmsprop optimizer that appears to performs quite properly with LSTM.

```
Train on 8544 samples, validate on 1101 samples
Epoch 1/6
8544/8544 [==============================] - 41s 5ms/step - loss: 1.4875 - acc: 0.3385 - val_loss: 1.3790 - val_acc: 0.3860
Epoch 2/6
8544/8544 [==============================] - 37s 4ms/step - loss: 1.1501 - acc: 0.4950 - val_loss: 1.4348 - val_acc: 0.4087
Epoch 3/6
8544/8544 [==============================] - 37s 4ms/step - loss: 0.7951 - acc: 0.6879 - val_loss: 1.7132 - val_acc: 0.3733
Epoch 4/6
8544/8544 [==============================] - 37s 4ms/step - loss: 0.4915 - acc: 0.8159 - val_loss: 2.1568 - val_acc: 0.3678
Epoch 5/6
8544/8544 [==============================] - 37s 4ms/step - loss: 0.2946 - acc: 0.8943 - val_loss: 2.5684 - val_acc: 0.3579
Epoch 6/6
8544/8544 [==============================] - 37s 4ms/step - loss: 0.1696 - acc: 0.9393 - val_loss: 3.0256 - val_acc: 0.3669

<Figure size 432x288 with 0 Axes>
```

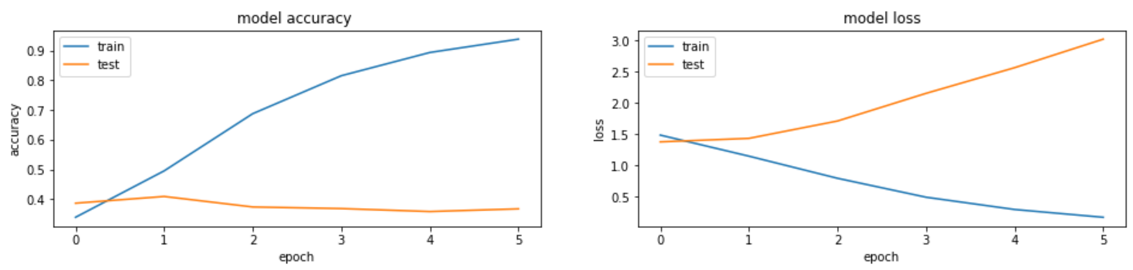

Figure 1: Loss and accuracy

We can see that from the epoch 2, the model is overfitting the training set and the dev loss is largely decreasing.

5 Innovate

In order to outperform the basic model with a LSTM encoder I did the following modifications :
(i) I pretrained my lookup table using the crawl-300d-200k.vec fastText word embedding
(ii) I used a conv1D layer to extract features
(iii) I used a maxpooling layer
(iv)I added the LSTM layer
(v) I tuned the RMSprop optimizer that I used.

I thought it is relevant to use a 1d convolutinal layer . A 1d CNN works well for identifying simple patterns within the data which will then be used to form more complex patterns within higher layers. A 1D CNN is very effective when we expect to derive interesting features from shorter (fixed-length) segments of the overall data set and where the location of the feature within the segment is not of high relevance.

Here however, we notice that this complex network do not outperform the simple LSTM network. I tried many other netorks but LSTM outperforms always.