Miniproject 2: Deep Learning for Natural Language Processing

Pirashanth RATNAMOGAN

pirashanth.ratnamogan@ens-paris-saclay.fr

In the following report one can find answers to the assignement paper questions but one can also find extra explanations and outcomes for what has been implemented. Answers to specific questions of the assignement will be preceded by the symbol ***Answer***

1. Monolingual embeddings

1.1 `Word2Vec`

1.1.1 Implementation :

In that section it is first asked to fill the class `Word2Vec`. We had first to add a function `score` that takes as input two words and return the cosine similarity between this two words using a pretrained loaded embedding. This function was created using some basic tools. Filling this function, one can wonder what to do if one of the two words given as input is not in the loaded lookup table. I have decided to arbitrarily put the output of the `score` function to 0 in this case that means that no similar words can be found using this method (0 is the worse score). Based on this `score` function, we then had to implement the function `most_similar` that returns the most similar words in the loaded pretrained lookup table.

1.1.2 Outcome :

We had first to compute the score between some pairs of words. I obtained the following score using the previously implemented function.

Word 1	Word 2	Score
cat	dog	0.67168
dog	pet	0.68421
dogs	cats	0.70744
paris	france	0
germany	berlin	0

As we can see we have the score between "paris"/"france" and the one between "germany"/"berlin" that is equal to 0. That is explained by the fact that neither "paris" nor "germany" are in the vocabulary. It is impossible to compute a score that involves this words. We can observe that the pairs "cat"/"dog", "dog"/"pet"

and "dogs"/"cats" are relatively close using the cosine similarity based score. That's what we were expecting. Then we had to give the most similar words for a given set of word using our `most_similar` function.

Input Word	Most Similar 1	Most Similar 2	Most Similar 3	Most Similar 4	Most Similar 5
cat	cat	cats	kitty	kitten	Cat
dog	dog	dogs	puppy	Dog	canine
dogs	dogs	dog	Dogs	puppies	cats
paris	-	-	-	-	-
germany	-	-	-	-	-

The final outcome seems really fair. As it has been said previously, we are unable to use the algorithm with the words that are not in the vocabulary : "paris" and "germany".

1.2 `BoV`

In this section the goal was to compare sentences using some simple sentence embeddings (mean or idf-weighted mean of the word embeddings present in the sentence).

1.2.1 Implementation :

We had to fill the class `BoV`. First, we simply create the function that compute and returns a dictionary that gives the *idf* score for all the words in our vocabulary given a list of sentences. I have computed the function that encode a sentence by taking the mean (or idf-weighted mean given the idf dictionary) of the words **that are in the lookup table**. If any of the words from a given sentence are in the lookup table we return a vector that as the dimension of the embedding vectors filled with zeros. Then using the same method described in the part about `Word2Vec` we compute a `score` function and a `most_similar` function that take as input a sentence s and a set of sentences S and returns the most similar sentences of s present in S.

1.1.2 Outcome :

We are trying to find the most similar sentences to the sentence "1 smiling african american boy ."

Using mean We have obtained this top returned outcome

1) "1 smiling african american boy ."
2) "blond boy waterskiing ."
3) "a boy jumps ."
4) "a boy jumps ."
5) "a boy smiles underwater ."

Using idf-weighted mean We have obtained this top returned outcome

1) "1 smiling african american boy ."
2) "5 women and 1 man are smiling for the camera ."
3) "a man rides a 4 wheeler in the desert ."
4) "3 males and 1 woman enjoying a sporting event"
5) "a man in black is juggling 3 flamed bottles ."

As we can see we have obtained different outcomes by changing the way that we are aggregating the word embedding in the sentence. Using the basic mean averaging of the word embedding, all the outcomes contains

the word "boy" that is actually present in quiet a lot of sentences. The context is not taken into account and in my opinion the most relevant outcome is "a boy smiles underwater ." that contains the information about "someone smilling" and the appearance of "a boy". However this outcome is only ranked in fifth position.
Using idf-weighted mean we can observe that almost all the outcomes doesn't contains the word "boy" that appears in a lot of sentences and hence doesn't have a heavy weight in the idf-weighted mean. The sentence "5 women and 1 man are smiling for the camera ." contains the information about "a man" and "smilling" seems a relevant outcome but other outcomes are not that relevant.
To conclude, the task to find sentence similarity is quiet complicated and by using this approach of averaging word embeddings is not that good because we are forgetting a lot of information. A good way to compare two sentences using deep learning is given by the so-called Siamese Network architecture.

2. Multilingual word embeddings

2.1. Theoretical part :

Answer In order to find a good mapping between a source language and a target language, one can solve the problem :

$$W^* = \text{argmin}_{W \in \mathcal{O}_d(\mathcal{R})} ||WX - Y||_F$$

The problem is equivalent to solving :

$$\begin{aligned} W^* &= \text{argmin}_{W \in \mathcal{O}_d(\mathcal{R})} ||WX - Y||_F^2 \\ &= \text{argmin}_{W \in \mathcal{O}_d(\mathcal{R})} ||X||_F^2 + ||Y||_F^2 - 2 < WX, Y > \\ &= \text{argmin}_{W \in \mathcal{O}_d(\mathcal{R})} - 2 < WX, Y > \\ &= \text{argmax}_{W \in \mathcal{O}_d(\mathcal{R})} trace(WXY^T) \end{aligned}$$

We can first reming the Von Neumann theoreme that states the following :
For any mxn real-valued matrices F and G, let $\sigma_1(F) \geq \sigma_2(F) \geq ... \geq 0$ and $\sigma_1(G) \geq \sigma_2(G) \geq ... \geq 0$ be the descneding singular values of F and G respectively. Then

$$trace(F^T G) \leq \sum_{i=1}^{n} \sigma_i(F)\sigma_i(G)$$

One can see more details about that in R.Vidal's book *Generalized Principal Component Analysis*.

We will introduce two SVD decompositons.

$$\begin{aligned} SVD(W) &= U_w \Sigma_w V_w \\ SVD(YX^T) &= U \Sigma V \end{aligned}$$

Then we can rewrite the quantity that we want to maximize for any $W \in \mathcal{O}_d(\mathcal{R})$ by :

$$
\begin{aligned}
trace(WXY^T) &= trace(U_w\Sigma_w V_w(U\Sigma V)^T) \\
&= trace(U_w\Sigma_w V_w V\Sigma U^T) \\
&= trace(\hat{U}\Sigma_w\hat{V}\Sigma) \qquad \text{with } \hat{U} = U^T U_w \quad \hat{V} = VV_w \\
&\leq trace(\Sigma_w\Sigma) \quad \text{by applying the Von Neumann theorem to } F = W \text{ and } G = XY^T
\end{aligned}
$$

Hence the quantity is maximized for $\hat{U} = U^T U_w = I$ and $\hat{V} = VV_w = I$. Hence $V_w = V^T$ and $U_w = U$. We hence have $W = U\Sigma_w V^T$. Finally using the fact that $W \in \mathcal{O}_d(\mathcal{R})$ we have $W^T W = I$. That leads to $U\Sigma^2 U^T = I$ that means that $\Sigma^2 = \Sigma = I$.
Finally we have obtained $W = UV^T$.

2.2. Implementation :

In order to use the previous part we have built a source input X and theirs associated target Y using the words that are common in the two languages. Then I built a class `BilingualWord2Vec` that allows to compute the nearest neighbours in the target domain for a word in the source input given the alignment using an approach close to the one used in the previous part.

2.2. Outcome :

Here are some outcomes that I have obtained using this implementation.

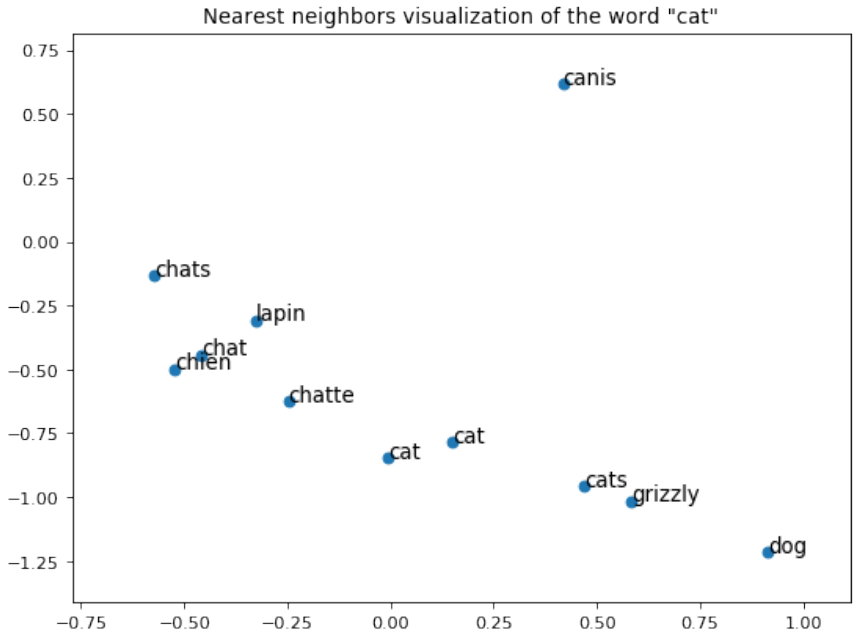

Figure 1: Exemple of embedding in the french space for the word "cat"

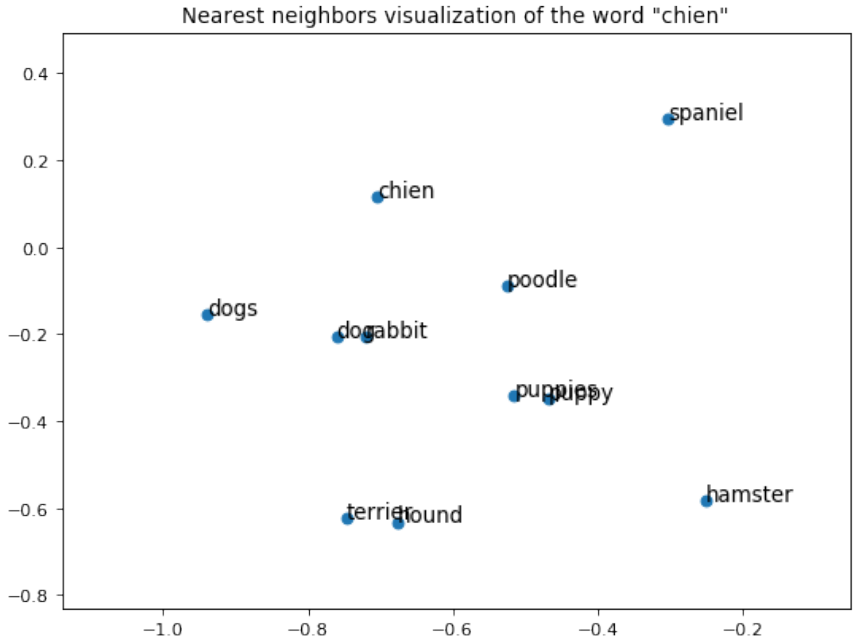

Figure 2: Exemple of embedding in the english space for the word "chien"

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. Implementation :

Using the encoder developed in the section **2** we have a simple way to create an embedding for each sentence, we can directly feed a simple classifier (based on the training set) on that embedding in order to classify each sentences. The goal of this part was to trained a Logistic Regression Classifier based on the embeddings developed in **2**.

3.2. Outcome :

Answer

Method	c value	Train Score	Dev Score
Mean	1	0.46102	0.40418
idf-weighted-mean	0.5	0.46184	0.38419

The score is pretty fair knowing as complex the task is.

3.3. Using other classifiers :

`LogisiticRegression` is a really simple classifier we can try to use other classifiers in order to see if we could obtain better outcomes using some other classifier.

Here is a table of the outcome that I have obtained using various complex and powerfull classifiers based on decision trees (`XGBoost` and LGBM are used with early stopping) :

Method	Encoding	parameter	Train Score	Dev Score
Logistic Regression	Mean	c=1	0.46102	0.40418
	idf-weighted-mean	c=0.5	0.46184	0.38419
Random Forest	Mean	n_estimator =100	0.99859	0.36421
	idf-weighted-mean	n_estimator =100	0.99859	0.34877
XGBoost	Mean	n_estimators=300, max_depth=3	0.81835	0.38056
	idf-weighted-mean	n_estimators=300, max_depth=3	0.67720	0.36330
LGBM	Mean	n_estimators=300, max_depth=3	0.69405	0.36966
	idf-weighted-mean	n_estimators=300, max_depth=3	0.69405	0.36966

Using all this classifiers doesn't allow to significatly outperform the Logistic Regression.

4. Deep Learning models for classification

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).

4.1. Implementation :

Answer First we use the a really simple network in order to perform our sentiment analysis task. 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 softmax 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. Considering a sentence x and its associated label y we will note $p(x) = (p(x)_i)_{i\in[[0;4]]} = (\delta_{iy})_{i\in[[0;4]]}$. Given a new input sentence x a neural network is trained to predict $q(x)$ that is a vector of the same size of $p(x)$, our neural network tries to approach p and gives q. The cross entropy quantify the difference between the output q with respect to the ground truth p. The loss is given for a set of input X by

$$L = -\sum_{x\in X} p(x)\log(q(x))$$

4.2. Outcome :

Answer For this really simple network, we can plot the evolution of train/dev results :

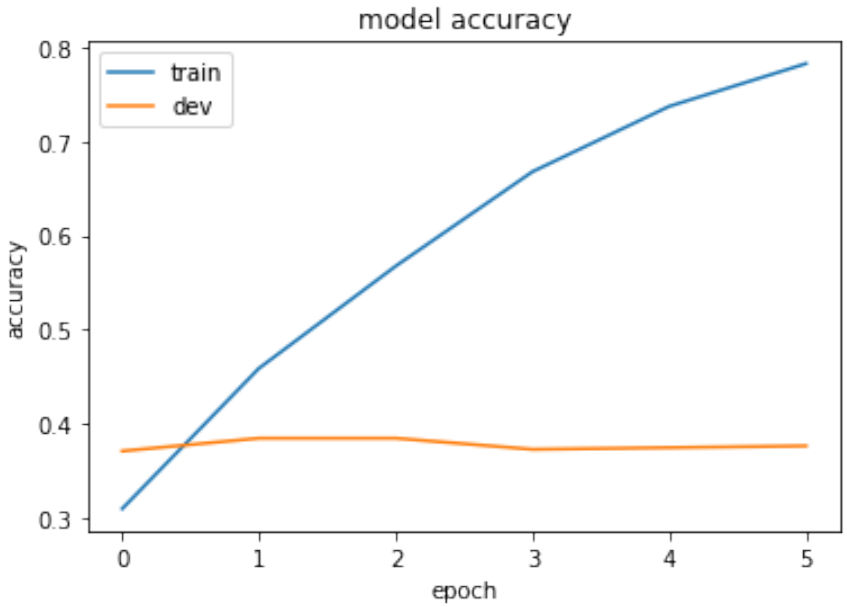

Figure 3: Evolution of the accuracy of the model through epochs

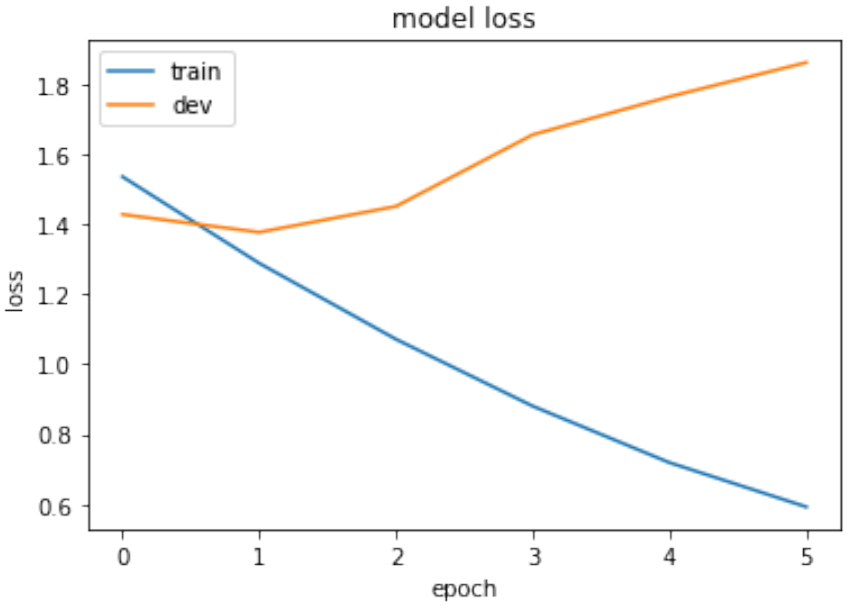

Figure 4: Evolution of the loss of the model through epochs

As we can see the outcome is close to the one obtained with the other classfiers previously. We can see that from the epoch 2, the model is overfitting the training set and the dev loss is largely decreasing.

4.3. Use a more complex model :

Answer It is a really complicated task to build a model that performs significantly better than the basic model with a simple LSTM (that obtain a maximum score about 39.5 in the dev set during the best

epoch). 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 transformed the LSTM layer as a Bidirectional LSTM layer
(iii) I tuned a bit the RMSprop optimizer that I used
(iv) I have extracted the outcome of the Bidirectional LSTM layer and I feed a linear SVM classifier that should produce the final outcome

I thought that using Bidirectional LSTM was a good idea because using LSTM layer just allows to capture the information in the forward way but going backward could bring some new informations about the semantic of the sentence. I tried to build deeper networks but any network was performing really well. I think that this could be explained by the small amount of data that we have for the training process. Using the previously described architecture I have obtained a final score of 39.69 on the dev set that is really close to the one obtained using the simple LSTM model. However, I think that this model will generalize more to the test set. The submission for this part is given in the file `bidirectional_lstm_svm_y_test_sst.txt`