Deep Learning - CentraleSupelec
NLP Project

By JEHANNO Emmanuel
emmanuel.jehanno@student.ecp.fr

Abstract

This lab is part of the Deep Learning course at CentraleSupelec. Its purpose is to get acquainted with the NLP topic in Deep Learning on different aspects : monolingual word and sentence embeddings, multilingual word embeddings and sentence classification with Bag-of-Vectors (BoW) and LSTMs. You can find my entire code repository on github : https://github.com/EmmanuelJhno/Deep_Learning/tree/master/Deep_Learning_Labs/nlp_project

1 Monolingual embddings

This is mainly described in the jupyter notebook which contains the code, the main results and some conclusions.

2 Multilingual word embeddings

The goal is to find a mapping W that will map a source word space (e.g French) to a target word space (e.g English), such that the mapped source words are close to their translations in the target space. For this, we need a dictionary of "anchor points". Here, we will use the identical character strings in both languages. We can show that the solution of $\underset{W \in O_d(R)}{arg\,min} \|WX - Y\|_F$ has a closed form.

Using the orthogonality and the properties of the trace, prove that, for X and Y two matrices : $W^* = \underset{W \in O_d(R)}{arg\,min} \|WX - Y\|_F = UV^T$, with $U\Sigma V^T = \text{SVD}\left(YX^T\right)$

The problem is equivalent to solving :

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

We can first remind the Von Neumann theorem that states the following:
For any $(m * n)$ real-valued matrices F and G, let $\sigma_1(F) \geq \sigma_2(F) \geq \ldots \geq 0$ and $\sigma_1(G) \geq \sigma_2(G) \geq \ldots \geq 0$ be the descending singular values of F and G respectively. Then :

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

We will introduce two SVD decompositions :

$$W = U_w \Sigma_w V_w$$
$$YX^T = U \Sigma V$$

Then, one has :

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

Then one can understand that the upper boundary would be obtained for $\hat{U} = I$ and $\hat{V} = I$. This means that $U_w = U$ and $V_w = V^T$. This leads to $W^* = U \Sigma_w V^T$ and since $W^* \in \mathcal{O}_d(\mathcal{R})$ we have $W^{*T} W^* = I$. Thus, one has $U \Sigma^2 U^T = I$ which means that $\Sigma^2 = \Sigma = I$ and finally $W^* = UV^T$.

3 Sentence classification with BoW

3.1 Logistic Regression with L2 penalty

Using either the average of word vectors or the weighted-average (idf) I get the following scores using a Ridge Classifier with $\alpha = 0.6$. I tried using a PCA to improve the results while keeping 99% of the variance or 95%. This technique sadly doesn't improve the results but interestingly, when applying a PCA the idf technique performs better than the standard mean.

Method	Train Score	Dev Score
Mean	0.4088	0.4005
idf-weighted-mean	0.3848	0.3824

3.2 Support Vector Machine Classifier

Using either the average of word vectors or the weighted-average (idf) I get the following scores using a Linear SVC with $C = 0.6$:

Method	Train Score	Dev Score
Mean	0.4148	0.4069
idf-weighted-mean	0.3892	0.3833

I also implemented the following algorithms :

- Decision Tree Classifier : no great results, worse than with Random Forest Classifier (even with PCA).
- Random Forest Classifier : 36% accuracy on the dev set which is not great... PCA doesn't improve neither.
- AdaBoost Classifier : 35% accuracy on the dev set, this technique is very difficult to master and doesn't improve the results here...

4 Deep Learning models for classification

4.1 Loss

I used the categorical crossentropy loss for this section in order to predict the true one-hot vector. Considering a sentence x and its associated label y, we will note $p(x) = (p(x)_i)_{i\in[0,4]]} = (\delta_{iy})_{i\in[[0,4]]}$ the one hot vector associated with y of size (batch_size * 5). Then, given an input sentence x our model is trained to predict this one hot vector p(x). The cross entropy quantifies the difference between the output q of our model and the ground truth p. The loss is given for a set of input X by

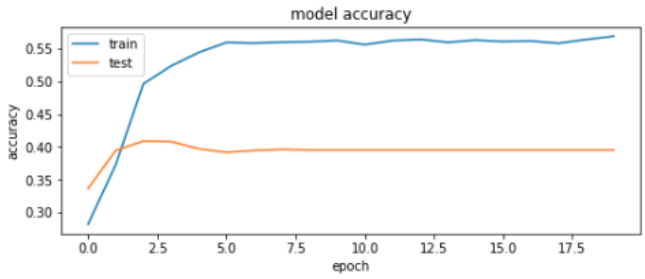
$$L = -\sum_{x\in X} p(x)\log(q(x))$$

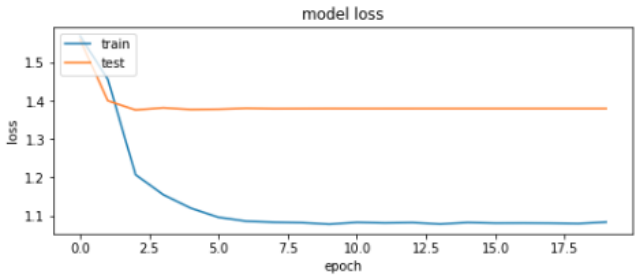
In this expression, since $p(x)$ is a one hot vector, then only the value $q(x)_i$ corresponding to the label will be considered.

4.2 Standard model

We add some callbacks to the model :

1. Learning Rate Scheduler : after some trainings we see that the model needs a high learning rate to overcome some local maxima at first but then it doesn't fit properly. Thus we implement a learning rate scheduler to decrease the learning rate value after a few epochs.
2. Model Checkpoint in order to save the best model with respect to validation loss.
3. Early Stopping in order to stop the training once the training has finished and the model starts overfitting. *However, I remove this in order to plot the training curves.*





Using this model with some properly chosen hyperparameters we reach in validation accuracy score after a very few epochs.

Method	Train Score	Dev Score
LSTM	0.5248	0.4142

The corresponding parameters are presented in the notebook (embedding of 50000 * 100, 256 hiddent units, batch size of 64, 20 epochs, learning rate of 0.0005, decay of 0.1 every 3 epochs...).

4.3 Innovate

Previously, we have used a LSTM network in order to predict using long term memory relations from our sentences. However, since the max length is 49, all our sentences are not that long and thus a standard Recurrent Neural Network might work ! It could have been interesting to investigate this approach even though it is somehow done by the previous Long Short Term Memory network (which has been developed to overcome the memory limitations of RNN).

I decided to go with Transformers which seem to produce the state of the art by using attention layers. In this case, we could have used transfer learning playing with pretrained networks from the Hugging Face library which gains much in importance in the NLP domain. However, since the idea here is to get the sense of how it works and to play a bit with the operations, I have developed my own Transformer network. I also use Keras API since it is simpler and more adaptable.

My original network is composed of an Embedding and a LSTM. The transformer part is thought to improve the LSTM part of the model, but one can also try to play with the embedding in order to improve the results. Thus, I also used both a trainable embedding and the GLOVE embedding in order to improve the results using other parts of the network. If needed you can download the weights on the following website : https://nlp.stanford.edu/projects/glove/ [file : glove.42B.300d.zip].

With this approach, we get the following results :

Method	Train Score	Dev Score
Transformer	0.9737	0.3815

These results are not great and the model is pretty hard to master and to make work properly. However, this approach is very promising ! The corresponding parameters are presented in the notebook :

- trainable embedding of 50000 * 300 in addition to GLOVE
- 256 hiddent units for the LSTM
- 256 neurons for the fully connected of the attention layer
- convolutions with 64 filters, a kernel size of 4, a pool size of 3
- classifier with 512 neurons for each of the 2 dense layers and finally 5 for the final classification
- batch size of 64
- 20 epochs
- learning rate of 0.0005
- decay of 0.1 every 3 epochs

NB : Thanks to attention layers we can't use Model Checkpoint callback in Keras to save the best weights so I used Early Stopping to stop the process and then restore the best weights.