Word2vec implemention's report

Author: Guénolé Joubioux and Gian-Mario Sangiovanni
Date : 5/11/2023

---

1 Introduction

The goal of this homework is to implement a **Word2Vec** model from scratch using PyTorch following the imposed guidelines.

2 Preliminary questions

1. Given a word w and a word c from the positive (resp. negative) context $c \in \mathcal{C}^+$ (resp. $c \in \mathcal{C}^-$), we want to minimize the following loss function:

$$L\left(M(w,c)\right) = -\mathbb{1}_{c\in\mathcal{C}^+} log(\sigma(\mathbf{c}\cdot\mathbf{w})) - \mathbb{1}_{c\in\mathcal{C}^-} log(1-\sigma(\mathbf{c}\cdot\mathbf{w})) \tag{1}$$

   In order to minimize (1):

   - If $c \in \mathcal{C}^+$, we want to maximize the $log(\sigma(\mathbf{c}\cdot\mathbf{w}))$ (or minimize $-log(\sigma(\mathbf{c}\cdot\mathbf{w}))$). As the *Sigmoid* function (the argument of the logarithm) is non-decreasing and it assumes values between 0 and 1, $log(\sigma(\mathbf{c}\cdot\mathbf{w}))$ will increase from $-\infty$ to 0. We then want to maximize $\sigma(\mathbf{c}\cdot\mathbf{w})$ in order to have a small value of the log in absolute value;
   - If $c \in \mathcal{C}^-$, we want to maximize to $log(1-\sigma(\mathbf{c}\cdot\mathbf{w}))$. By the same reasoning as before we want to minimize $\sigma(\mathbf{c}\cdot\mathbf{w})$.

   As we want to minimize the loss and the *Sigmoid* function computes the similarity between c and w, it is pertinent to maximize the similarity between a word and its context and minimize between a random word.

2. - The contrastive learning is an un-supervised learning method which is going to search for similarities inside our data. Yet, we still need to specify which datas are similar and whose who are not. Through a similarity metric, the model is going to be trained to retrieve the underlying information of our data. This type of learning is going to approximate a classification of our data, with possibly hundreds of thousands or more of classes.
   - In the paper Y corresponds to a binary label. In our loss function, the analog is the indicator function $\mathbb{1}_{c\in\mathcal{C}^-}$ which takes 0 as value when c is a positive context.
   - Here, the analog of E_W is $(w,c) \mapsto \sigma(w\cdot c)$. They both measure the similarity of the two vectors in vector spaces but this loss is going to grow very fast for non-similar words.
   - L_G must increase because it is evaluating the loss of similar category. So it corresponds to $x \mapsto -log(x)$ function. L_I must decrease so it corresponds to $x \mapsto -log(1-x)$ function.

3 Word2Vec model

The **Word2Vec** model searches links and similarity towards words and then it develops a better vector representation of the meaning of each word through *word embeddings*.
Each word w is going to be associated to a positive context $\mathcal{C}_w^+$ and a negative context $\mathcal{C}_w^-$. The positive context are the associated words to w i.e. $\mathcal{C}_w^+ = [w_{i-R}...w_{i-1}w_{i+1}...w_{i+R}]$ and it is parametrized by the radius $\mathbf{R}$. The negative context are random words parametrized by the scale $\mathbf{K}$. In particular, for each word we have $2\mathbf{R}$ words inside the positive context and $2\mathbf{KR}$ inside the negative context. Note that not for all w we are able to find $2\mathbf{R}$ words for the positive context, e.g if the word is the first in the sentence we can just grab $\mathbf{R}$. Our basic solution is to add zeros in order to have vectors

of the same length (this can be seen as a sort of padding). We set differents values of the radius $\mathbf{R} = \{4, 6\}$, the scale $\mathbf{K} = \{2, 4\}$ and embedding dimensions $\{25, 100\}$ in order to find which is the best parametrization in our small study case . We have also trained a model with more data (25.000 sample), obtaining more less the same results .
Next step is to map each word w to a vector of $\mathbb{R}^d$ through an embedding space. Each words of positive and negative contexts will also be mapped to a d-dimensional real vector through a different embedding space. Our Word2Vec model is going to be trained to get the best embeddings which are able to maximize the similarity between a word w and a word of its positive context and minimize the similarity with a word of its negative context. The last sentence comes from the idea that a word w should be somehow related/linked to a word which is near to it while it should not share any relationship with a *distant* word. As loss, we are going to use a slighlty modified version of (1). More formally:

$$L\left(M(w,c)\right) = -\mathbb{1}_{c\in\mathcal{C}^+} log(\sigma(\mathbf{c}\cdot\mathbf{w}) + \epsilon) - \mathbb{1}_{c\in\mathcal{C}^-} log(1 - \sigma(\mathbf{c}\cdot\mathbf{w}) + \epsilon)$$

ϵ is added to the loss in order to avoid an infinity loss (namely when we are computing $log(0)$). We fix $\epsilon = 0.05$.
We evaluate the accuracy of the model with a sort of threshold classification, i.e assigning to words inside the positive context the label 1 while assigning 0 to words inside the negative context. The threshold is used to convert a probability to either 0 or 1:

$$\hat{y} = \begin{cases} 1 & \sigma(\mathbf{c}\cdot\mathbf{w}) \geq \mathbf{thresh} \\ 0 & otherwise \end{cases}$$

After an empirical study, we decide to fix the value of the threshold to 0.3.
Remark, we `choose` that $\mathcal{C}_w^+ \bigcap \mathcal{C}_w^- = \emptyset$.

4 Classification of positive or negative Reviews

We are now going to do a classification task of positive of bad reviews from **IMDB** database. We already have trained our Word2Vec model and so we get trained embeddings. We have decided to use the simple convolutional model made of a convolutional layer, a relu activation funtion, an adaptive max pooling layer (on the embeddings), a dropout layer (with $p = 0.3$) and a final linear layer. As loss we have decided to use a **Crossentropy** and as optimizer **Adam** with *learning rate* equal to 0.001 and the default value for *epsilon*. We have decided to use 5000 samples.
Then we have compared two models which share the same architecture but one of them is initialized with the trained embeddings (fixed) while the other one has embeddings as parameters to learn. Note that we do not want to train our embeddings in the pre-trained model.

5 Result

We set the number of epochs in our classification training to 5, getting a validation accuracy (our best model with 100 embeddings dimension) around 0.72 which is better than the model without the initialized embeddings which has 0.57 (0.53 with 25 embedding dimensions). It is an improvement and we could maybe improve it a bit more with a better optimization of the Word2Vec model. Unsurprisingly, we can observe that when the Word2vec model is trained with a low number of embedding dimensions, the situation changes as well as when the Word2vec model is trained with an higher number of dimension (100), we have an improvement for our task. This is consistent with the theory.

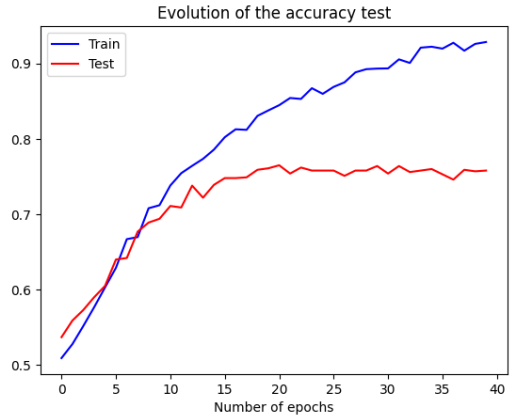
radius/scale	2	4
4	0.59	0.587
6	0.589	0.614

Table 1: Accuracy score with dimension 25

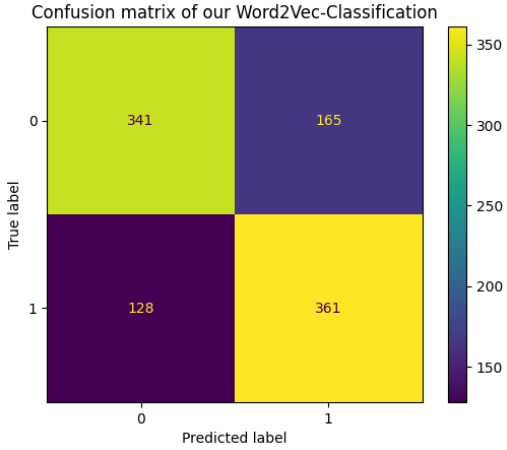
radius/scale	2	4
4	0.72	0.662
6	0.685	0.67

Table 2: Accuracy score with dimension 100

After having assessed the power of word embeddings, we have studied the number of epochs needed for the training of the model in the classification setup. We choose to analyze the performance of our best model to understand if allowing a greater number of training epochs it is possible to get a better score (we decrease the learning rate to $1e^{-04}$). As we can see from the plot, the training accuracy tends to linearly grow along the number of epochs. Moreover, giving a look to the evolution of the testing accuracy, we can see that it begins to stabilizes (0.75) quite soon and so a low number of epochs is needed. Note that it needs a lot of different epochs to cover the distance between 0.7 and 0.75.

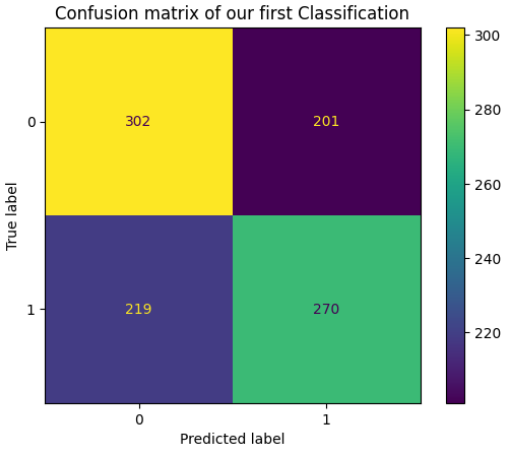


(a) Trace plot for the training and testing accuracy



(b) Confusion matrix after 20 epochs

We have also implemented a small code in order to achieve the **classification matrix** of a model. Here we show the confusion matrices of our best model and its equivalent without trained embeddings. Comparing those two plots it is possible to state that their behaviour is quite different after all, moreover the pre-initialized model performs better in the prediction of positive labels (good reviews). The two models seem balanced between the negative labels (bad reviews). As a general reminder, all the models tends to fail in the estimation of the positive labels.



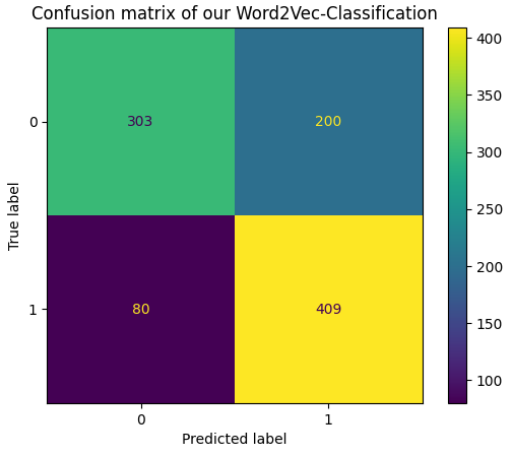


Figure 2: Confusion matrices for the two models, 5 epochs and a learning rate of 10^{-3}

Just a final remark. We have also used the model which takes into account more samples for the training of the embeddings and we get a slight decrease in the accuracy terms (around 0.73). Overall, we have gained valuable knowledge concerning the usage of static word embeddings and the implementation of Convolutional neural networks using pytorch.