**Large Language Model**
November $8^{th}$, 2023
Mathilde KRETZ

# Homework 2

**Word2Vec**

## 1 Preliminary questions

1. We want to minimize the loss whose computation is given by:

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

$\mathcal{L}$ is a decreasing function of $\sigma(\mathbf{c}\cdot\mathbf{w})$ when $c \in C^+$ and a increasing function of $\sigma(\mathbf{c}\cdot\mathbf{w})$ when $c \in C^-$. Since $\sigma$ is a increasing function of $c$, by minimizing $\mathcal{L}$, we want to maximize $\sigma(\mathbf{c}\cdot\mathbf{w})$ for $c \in C^+$ and minimize it for $c \in C^-$. Minimizing $\sigma(\mathbf{c}\cdot\mathbf{w})$ means bringing it closer to zero, which is equivalent to making $c\cdot w$ tend toward minus infinity. On the contrary maximizing $\sigma(\mathbf{c}\cdot\mathbf{w})$ is equivalent to making $c\cdot w$ tend toward plus infinity (as $\sigma$ tends toward one). Geometrically, a $w$ and a $c$ whose dot product tends toward plus infinity (as it is for positive context) are vectors pointing to the same direction with a slight angle: points are in the same area for a common origin, and a $w$ and $c$ whose dot product tends toward minus infinity are vector pointing to opposite direction (and maximal angle).

2. Contrastive learning is a technique proposed as a solution for classification problem where the dataset used during the training implicates a large number of categories which may not be all known (eg. classifying people's face within an unknown population) and/or which contain a limited number of information. The idea is to train a similarity metric, from the training data, that will map into a target space the data from input space such that similar samples end up close (in the Euclidean sense) in the target space. Classification is then made by a kind of clustering where classes are built with close data points.

   The paper gives a form for this similarity metric:

$$L(W,(Y,X_1,X_2)^i) = (1-Y)L_G(E_W(X_1,X_2)^i) + YL_I(E_W(X_1,X_2)^i). \tag{2}$$

   Our similarity metric is of this form with:

   - $Y = \mathbb{1}_{\{c\in C^-\}}$
   - $L_G = -\log(x)$
   - $L_I = -\log(1-x)$
   - $E_W = \sigma(\mathbf{X_1}\cdot\mathbf{X_2})$

## 2 Word2Vec implementation

While the first part of data preprocessing is pretty similar to what have been done on previous practical sessions, the major differences of preprocessing can be summarized in the `collate_fn` function. It returns for a batch (here a list of tokens representing a text) as input, a dictionary composed of:

- a `word_id` which is the token of all word in the text,
- a list `positive_context_ids` that is of length `2*K` and that contains the tokens of the words composing the positive context, ie words inside the circle of radius `R` around `word_id`. It is important to note that for word in the border of the text, the token `0` has been used for context words that don't exist, as it usually represents the sequence `<PAD>`.
- finally a list `negative_context_ids` that is of length `2*K*R` and that contains the tokens of the words representing negative context. They are drawn randomly from the vocabulary given by the tokenizer. Parameters `R` and `K` have to be chosen.

The model has then been implemented as a class inherited from `torch.nn.Module`. It defined the two embedding tables described in the instructions: one for the actual words and one that embeds the context words. The model returns for a word $\mathbf{w}$ and a single context word $\mathbf{c}$, the similarity score $\sigma(\mathbf{w} \cdot \mathbf{c})$. The embedding size has also been chosen, it should be between 100 and 300.

A training function has then been implemented consequently. To correspond with the modeling described in part 1, the `nn.BCEloss()` has been used to operate the binary classification loss since the sigmoid is computed inside the model. Additionally, as every context word has to be modeled with its own central word, the training computes the mean of the loss of all similarity scores returned by the model (`batch_size * 2*R` similarity scores for `positive_context_ids` for example), and sums the losses given by positive and negative context (it could have been the mean too). The model is then backpropagated given this loss and an accuracy metric is computed to be able to visualize learning process. It has arbitrarily chosen that a word is considered as positive context it its similarity with the central word is higher that 0,5 and from negative context if it is lower than 0,5; however an other threshold could have given a more preciser idea of the accuracy.

Concerning the training of the model, the number of samples used has been maintained to `5000`, `R` and `K` chosen arbitrary at respectively `4` and `6`, and for computational reason, the embedding dimension has been lowered to `100`.

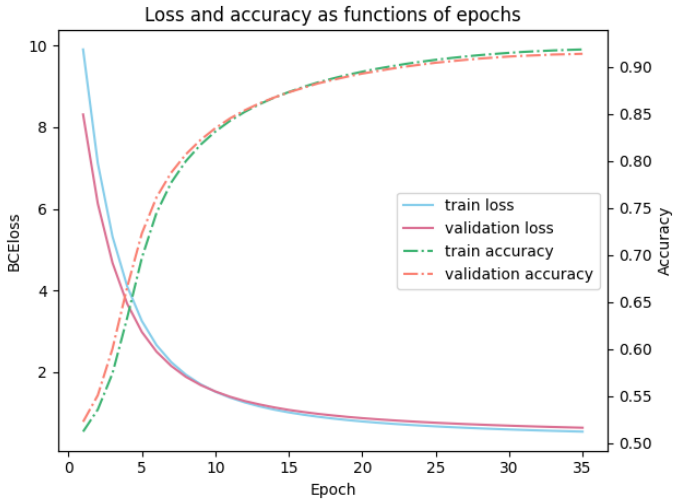


Figure 1: Loss and accuracy functions of epoch - first setting

No overfitting is observed on figure 1 in this setting, even after 35 epochs. Accuracy reaches an acceptable level while the training loss is still decreasing followed by the validation. The computation time is important even when computed with cuda (more than an hour and a half for 35 epochs).

In a second time, the embedding dimension was set to `200` to observe its contribution to the results. Alongside, `R` and `K` have been lowered to `2` and `3`. It is important to note that this is absolutely not a good practice to observe influence of parameters on the results, they indeed need to be changed independently, the idea here was to keep a comparable number of values to compute.

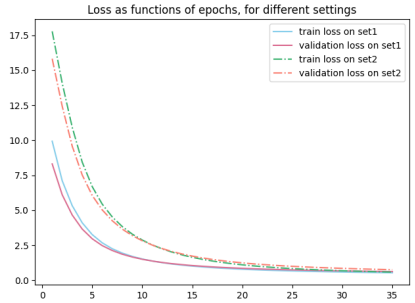


Figure 2: Loss as function of epoch - comparison of first and second setting.

We observe that in the second setting, the model takes more time to converge to the same value of loss and accuracy. Even if the model is supposed to be more precise due to the high dimension of the embedding, it seems that having lowered R and K created a balance in the resulting accuracy.

# 3 Classifier implementation

The classifier part implements parts of code that have been made during previous labs, especially preprocessing and dataloader parts. A convolutional model with a convolution layer with a single filter dimension has been implemented based on the idea of (1) in its simplest setting. Two models can be set with the implementation of the notebook: a first one using the `load_model` function which initializes the embedding layer with the weights coming from the checkpoints saved by the Word2Vec model, and a second initializing randomly the embedding weights of embedding layer as usual. By training this model, we can compare the results with or without the Word2Vec embedding initialization.

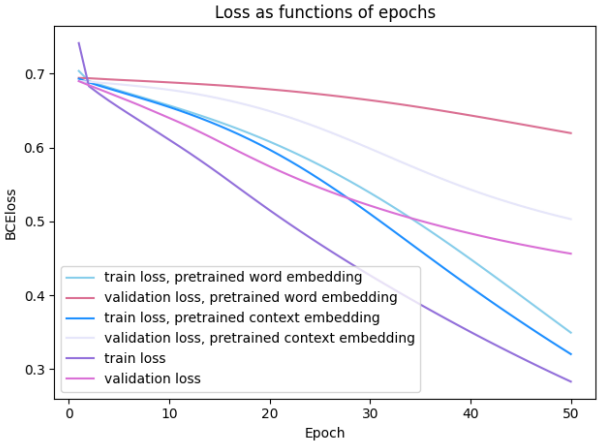


Figure 3: Loss as a function of epochs for different settings of classifiers.

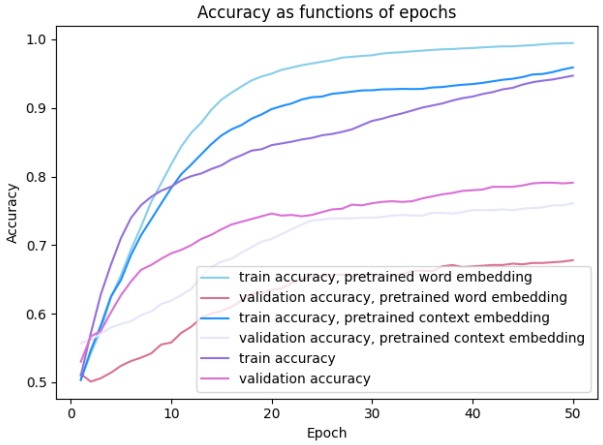


Figure 4: Accuracy as a function of epochs for different settings of classifiers.

The observed results on figures 3 and 4 which was what was expected is that, with the pretrained embedding layer, the classification model converges faster. Nevertheless, in the pretrained model, the validation follows the training values further back than without embedding initialization, especially in term of accuracy. An interesting point can be observed too: the difference between the use of both embedding tables from Word2Vec. The embedding of "central" words is supposed to connect together word that are semantically close ie that have the same context such as synonyms etc.., while the embedding of context words connect together words that have a linked meaning because they can be found inside the same context. It seems logical for our task to initialize our model with the (central) word embedding which would allowed our model to understand positive or negative semantic of words to better classify. It is well observed on figure 4 where the word embedding enabled the training accuracy to converge faster and higher than the context accuracy and the non initialized model. However this is inversely followed by the validation accuracy, which may be due to the basic setting of our convolutionnal layers.

# 4 Conclusion

An more in-depth ablation study could have been carried out by observing accuracy results depend on R, K, the embedding dimension and the number of samples, each of them independently modified.

In conclusion, the homework enabled me to understand the implementation of neural networks in great depth, as well as the points to be taken into account for training. It was very instructive for the overall understanding of the course although the computing power issues were very frustrating.

# References

[1] Y. Kim, "Convolutional neural networks for sentence classification," *CoRR*, vol. abs/1408.5882, 2014.