Homework: Word2Vec Model

Jules Merigot[1]

[1]PSL Research University - Large Language Models

November 8, 2023

1 Introduction

The following homework report will detail a summary of the implementation of a Word2Vec model, the justification for its manner of implementation, as well as the answers to a series of important questions related to the model and its usage. This will be accompanied by plots and graphs to analyze results and showcase the performance of the model, as well as the analysis of a classification task utilizing a convolutional model.

2 Preliminary Questions

1. To compute the similarity between a word and a context, we use the similarity $\sigma(\mathbf{c} \cdot \mathbf{w})$. Since we want to minimize the loss:

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

   (a) For $c \in C+$, we maximize $\sigma(\mathbf{c} \cdot \mathbf{w})$. Since we want to minimize the loss, we also want to minimize the negative log likelihood of $\sigma(c \cdot w)$. The log function increases as its input increases, so to minimize the negative log likelihood, we maximize its input of $\sigma(\mathbf{c} \cdot \mathbf{w})$.

   (b) For $c \in C-$, we minimize $\sigma(\mathbf{c} \cdot \mathbf{w})$. Since we want to minimize the loss, we also want to minimize the negative log likelihood of $1 - \sigma(c \cdot w)$. The log function increases as its input increases, so to minimize the negative log likelihood, we maximize $1 - \sigma(c \cdot w)$ and minimize $\sigma(\mathbf{c} \cdot \mathbf{w})$.

To give a geometrical interpretation, we can imagine each word w and context c as vectors in a high-dimensional space, and $c \cdot w$ represents a measure of how these vectors align. For $c \in C+$, maximizing $\sigma(\mathbf{c} \cdot \mathbf{w})$ would be like pushing the vectors toward each other to indicate a positive association between the word and the context. On the other hand, for $c \in C-$, minimizing $\sigma(\mathbf{c} \cdot \mathbf{w})$ would be like pushing the vectors away from each other to indicate a lack of association between the word and the context.

2. One of the first application to contrastive learning has been presented by Chopra, Hadsell, and LeCun [1].

   (a) Constrastive learning is a method of learning patterns in data in order to identify patterns in unknown data. This would be like meeting new family members at a reunion, and identifying them based on their resemblance to another family member. You have learned that a family member that you know well has certain features, which enables you to classify a new unknown family member, maybe a distant cousin, based on the same feature patterns that you have learned.

   Page 3 of this article, we find the expression:

$$L(W, (Y, X_1, X_2)^i) = (1 - Y)L_G(E_W(X_1, X_2)^i) + Y L_I(E_W(X_1, X_2)^i), \qquad (2)$$

   (b) The analog of Y is a binary label of whether the context of a given word is compatible or not with the word. If Y = 0 for the pair of a word and context, then they are compatible – a "genuine pair". However, if Y = 1 for the pair, then they are incompatible and we would not be close in common language usage – an "impostor pair".

   (c) The analog of E_W is an "energy" function that measures the compatibility of the word and context inputs. This is not the same as Y which is only a label scalar that indicates the similarity between the two inputs – the word and the context. On the other hand, E_W is an embedding function that transforms the input data, maps the words and contexts to their vector representations, and processes it.

   (d) The analogs of L_G and L_I are the partial loss functions for a "genuine" pair of word and context inputs and for an "impostor" pair of word and context inputs, respectively. They are specialized loss functions for handling similar and dissimilar pairs respectively.

3 Implementation of the Word2Vec Model

3.1 Pre-processing

For the pre-processing of my model, I followed the instructions laid out by the report guidelines. Therefore, I will only be justifying the steps where I made decisions that were not imposed.

When extracting the contexts, to handle the borders and ensure that every context has the same size, I used a padding token that allows the context of words at the beginning and end to still be extracted. By padding at both ends using `padding_token` in my function, I ensure that even the first and last words of the document have a full context window. I have chosen a padding token ID of `0` because it does not clash with any word IDs in my dataset.

For my two parameters R (the radius of context) and K (the scaling factor), I chose values of 4 and 6, respectively, in order to have more context in my final model as well as proper scaling. For sampling the negative context, I randomly sampled from the vocabulary set of the `BERT` tokenizer. I finally wrapped everything in DataLoader with a batch size of 16 due to the smaller size of my dataset.

3.2 Model

For the implementation of my actual `Word2Vec` model, I followed the formulation of the algorithm provided in the guidelines of the report. This includes sampling a word w from a document, taking the words surrounding w in a local window of radius R in the document to make the positive context C^+, and sampling $2KR$ random words in the vocabulary V to make the negative examples, C^-. After computing the similarity scores between the contexts and w using the sigmoid function, I have a model M that maps a word w and another word c from a context $c \in C^+$ or $c \in C^-$) to a scalar $\sigma(c \cdot w)$. This allows me to compute the binary classification loss of my model in my training function. For my model, I chose an embedding dimension of 100 to start.

For the training of my model, I am using certain hyperparameters to increase the performance of my model. I use the Adam optimizer with the model's parameters, as well as a learning rate of $5 \times (10)^{-5}$, as recommended by the authors of the BERT model. I chose to test different numbers of epochs E, starting with 20, and increment by 5 until 40. During training and testing, my model produced the best results when run for a duration of 40 epochs.

In my training and validation function, I compute for each epoch the training loss and accuracy as well as the validation loss and accuracy, respectively. The binary classification loss is calculated by applying the `nn.BCELoss()` module of torch on the similarity predictions outputted by the `forward()` function of our model. The accuracy is calculated by counting the number of outputted similarity predictions larger than 0.5 that equal the labels of the words in the dataset, and then dividing by the total predictions before multiplying by 100 to get a percentage value. These calculations are done for both the training and validation data in each epoch, and are printed at the end of each epoch, with a final list compiled at the end to enable graphic representation of the model's performance.

4 Analyzing Results

Below is a graphical representation of the metric results of my model when run for 40 epochs. I also trained my model over less epochs, but as the epochs increased, so did my model performance. Due to the time it takes to train my model per epoch and GPU limits, I was not able to train my model for more epochs, but that is an extension worth pursuing.

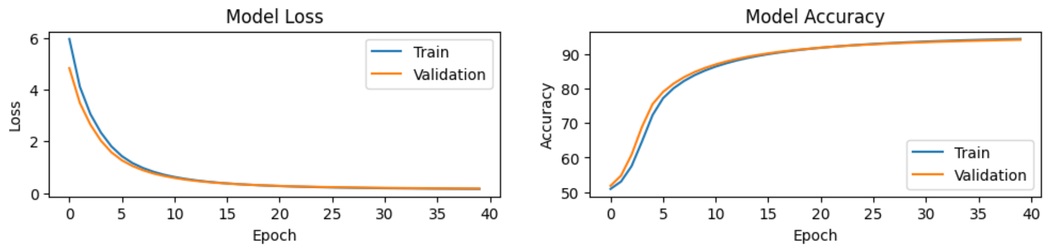

Figure 1: Model loss and accuracy results for Train vs. Validation

As we can see, both the train and validation loss of the model decrease exponentially over time and plateau around a loss value of 0.18. While the validation loss is consistently lower than the train loss at first, after the 23rd epoch, the train loss becomes lower. With a final validation loss value of 0.16, I can say that my trained model performed well as expected. For the model accuracy, both the train and validation accuracy increase over the epochs, reaching values above 90% accuracy. After 40 epochs, the final train accuracy is 96.4% while the final validation accuracy is 95.1%.

5 Classification Task

5.1 Augmented with Word2Vec Embeddings

For the classification task, I augmented a classification model with my Word2Vec model. This task utilizes a Convolutional model that I first trained with an initialization using the embeddings saved from my Word2Vec model. After training this model, I compared it to a simple model, trained with some randomly chosen intialization parameters. Comparing the results of the two models will give some insight into how performant the embeddings from my first model are.

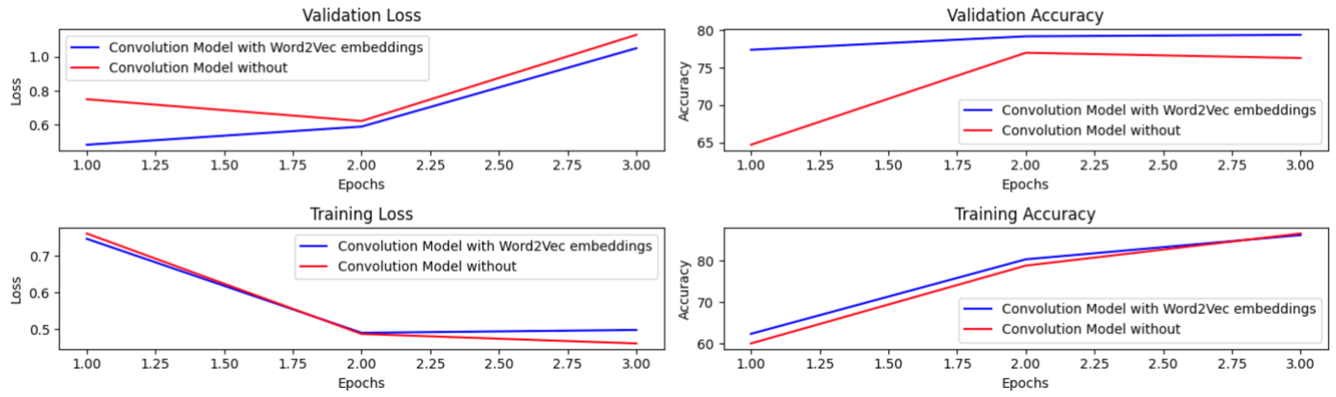

Figure 2: Model Training and Validation Metrics Comparison

From the plots above, it can be observed that after training for 3 epochs, using the embeddings of Word2Vec as initialization for the convolutional model does not improve performance, and in fact becomes worse than a model with simple initialization. This is not expected, but shows that the Word2Vec embeddings are not appropriate for a classification model. Aside from the validation loss, the other metrics are behaving as expected, with loss decreasing and accuracy increasing over the epochs. After multiple different experiments with a consistent random seed value, I noticed no change in the performance trend of these models.

5.2 Ablation Study to Compare Embedding Configurations

In my ablation study, I decided to test the influence of some parameters of the Word2Vec model on the classification task. For this, I set up various configurations for my Word2Vec model, and decided to test different embedding dimensions, batch sizes, context radiuses, and ratios. The results of these tests can be seen in the table below.

Embedding Dim (d)	Batch Size (B)	Radius (R)	Ratio (K)	Train Loss	Valid Loss	Train Acc. (%)	Valid Acc. (%)
50	16	4	6	0.4808	0.9847	87.35	78.50
50	32	4	10	0.3978	1.0538	86.67	71.20
100	32	6	8	0.4901	0.8841	86.67	75.80
100	64	6	10	0.5182	0.5757	83.38	77.50
200	64	8	10	0.3405	0.6911	87.72	80.40

Table 1: Metric Results of Different Configurations in the Ablation Study

In the table above I included only the configurations that provided good metric results when trained for 3 epochs. From these results, it can be observed that an embedding size of 100, batch size of 64, radius of 6, and ratio of 10 seems to be the best configuration of parameters. However, it can be concluded from this experiment that the batch size and ratio values seem to have the most influence on the classification task, as they are the two parameters that caused the most improvement when increased.

6 Conclusion

By completing this homework, I have deepened my understanding of the Word2Vec model and its practical application in natural language processing. Through the process of fine-tuning hyperparameters and utilizing pre-processing techniques, I've learned how to effectively shape and influence model performance. This hands-on experience has highlighted the significance of embedding quality in enhancing downstream tasks, such as classification using a convolutional model. This task has been a concise yet comprehensive exercise in applying theoretical concepts to real-world machine learning challenges.