Word2Vec and Classification

Maxence Lasbordes
PSL-Research University
MASH

Abstract

This report details the implementation of Word2Vec from scratch in PyTorch, as introduced by Mikolov et al. (2013) in the Large Language Models course at Université Paris Dauphine. Word2Vec is a neural network model that generates dense vector representations of words, capturing semantic relationships in text. This implementation explores key techniques of the architecture, including the use of positive and negative contexts, data preprocessing, the model class, and the training process. We will then use the pre-trained embeddings as a starting point for our classifier. Finally, we will compare the impact of various Word2Vec parameters on the model's performance.

1 Introduction

The Word2Vec algorithm samples a word from a dataset and identifies its surrounding words within a specified window radius R to create positive context examples. Additionally, the model utilizes negative sampling, where random words from the vocabulary are selected as negative examples. This implementation computes similarity scores between the target word and context words using the sigmoid function, with the objective of maximizing similarity for positive contexts and minimizing it for negative contexts. This process enables the model to learn effective word embeddings that capture the underlying semantic structure of the text. We will use a pre-built BERT tokenizer and Hugging Face's sentiment analysis dataset 'imdb'.

2 Model architecture

Our implementation of Word2Vec slightly differs from the original, as our goal is to predict whether given context words match a specific target word. Specifically, we use a sigmoid function to evaluate whether the given words are actually part of the context for the target word.

During pre-processing, one issue was the varying lengths of the contexts. When the selected word was close to the beginning or end of a sentence, the surrounding context would be cropped due to the lack of available words. To address this, we used padding tokens (the 0 token from the pre-trained BERT tokenizer by Hugging Face) to equalize the length of the context lists. Since token positions are not considered, the necessary padding tokens were simply added to the end of each context list.

During the training process, for each target word we provide the model with positive examples of actual context words and negative examples of random words from the vocabulary. The models outputs a probability for each word in the vocabulary, indicating how likely it is to be a context word. So for positive context words, we want the model to output probabilities close to 1, while for negative context words, we want the model to output probabilities close to 0.

3 Word2Vec Training

We train our model using the hyperparameters described in the Table 1. Furthermore, the accuracy was computed by considering a correct prediction as one where the model predicts a probability greater than 0.5 for positive context words and less than 0.5 for negative context words. Naturally, we do not take into consideration the padding tokens added to equalize the length of the positive context.

We feed our model with both positive and negative context words. The loss is the one for binary classification for a word w and a context c.

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

Table 1: Hyperparameters used for training.

Parameter	Value
Context Size	10
Negative Samples Factor	5
Embedding Dimension	100
Learning Rate	1×10^{-3}
Batch Size	256

Table 2: Training Loss & Test Accuracy per Epoch for the Word2Vec model.

Epoch	Training Loss	Test Accuracy (%)
1	3.6660	80.78
2	1.3400	86.79
3	0.9324	88.58
4	0.7651	89.35
5	0.6718	89.77
6	0.6110	90.01
7	0.5679	90.17
8	0.5367	90.29
9	0.5135	90.37
10	0.4960	90.46

4 Classification task

4.1 Training

We now proceed to the classification task using a basic convolutional model. First, we train it with the pre-trained context embeddings from our previous Word2Vec model, and then without loading the embeddings. We observe that the losses are generally lower when using the pre-trained embeddings from Word2Vec. This result makes sense, as starting the training with embeddings that already capture some meaning of the tokens provides an advantage compared to randomly initializing them.

Table 3: Classifier training with and without Checkpoint Loading from Word2Vec. With model_dim-200_radius-10_ratio-2-batch-256-epoch-2.ckpt

	Checkpoint Loading			No Checkpoint Loading		
Epoch	**Train Loss**	**Val Loss**	**Val Acc (%)**	**Train Loss**	**Val Loss**	**Val Acc (%)**
1	0.0383	0.0310	78.30	0.0410	0.0378	68.80
2	0.0195	0.0288	78.50	0.0306	0.0337	72.60
3	0.0063	0.0264	80.40	0.0227	0.0318	73.80
4	0.0014	0.0268	80.90	0.0152	0.0309	75.80
5	0.0006	0.0275	80.90	0.0090	0.0318	76.00

4.2 Influence of Word2Vec parameters

When using various embedding sizes, we observe that larger embedding sizes enhance the effectiveness of the Word2Vec pre-trained embeddings as a starting point for the classifier, leading to improved validation accuracy throughout training. Furthermore, the accuracy does not vary significantly after five full epochs, regardless of the embedding size used. This lack of variance may be attributed to the limited size of the dataset or the relatively simple architecture of the model, which has few layers and parameters. Regarding the context sizes (R), the results appear to be similar no matter the context size, which is surprising, as we would expect the model to perform significantly worse with embeddings trained on smaller context sizes. It is possible that a much higher context size might reveal a more noticeable difference in performance.

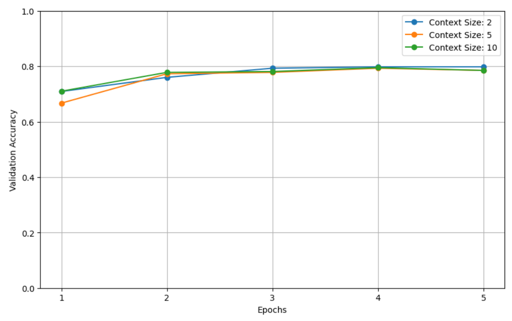

Figure 1: Validation Accuracy for Different Context Sizes

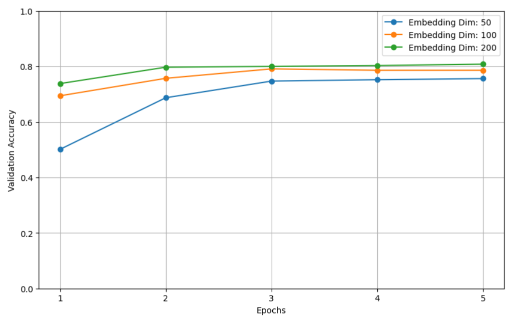

Figure 2: Validation Accuracy for Different Embedding Dimensions

5 Preliminary question

[1]
[a] For positive contexts $c \in C^+$, where the context and the word are related, we want to maximize $\sigma(c \cdot w)$ so that the probability is close to 1 which means the two words have a strong similarity.

[b] For negative contexts $c \in C^-$, we want to minimize $\sigma(c \cdot w)$ so that the probability is close to 0, because the context and the word are unrelated.

When we maximize similarity for positive contexts, we are making the vector for the word w point closer to the vector for its true context c^+ in vector space.
When we minimize similarity for negative contexts, we are pushing the word vector w and the negative context vector c^- farther apart.

[2]
[a] Contrastive learning is a method where the model learns to differentiate between similar and dissimilar data points by grouping similar examples together and pushing dissimilar examples apart.

[b] In the equation presented by Chopra, Hadsell, and LeCun:

$$L = (1 - Y)L_G + Y L_I$$

The analog of Y is the context c, from C^+ (positive context) or C^- (negative context). Which means, $Y = 1$ if the context is positive and $Y = 0$ if it is negative.

[c] The analog of E_W would be the dot product $c \cdot w$ between the word and context vectors. But it is not exactly the same as in our case we apply a sigmoid function to this dot product.

[d] L_G represents the loss for the negative context. Which is the analog of the loss associated with C^-. And L_I represents the loss for positive contexts. Which is the analog of the loss associated with C^+.