Word2Vec

Salima MAMMA[a], **Pan LIU**[b], **Doha EL HITARY**[c], **Quyen Linh TA**[d]

[a]*Data Scientist, Macif*
[b]*Data Scientist, BNP Paribas*
[c]*Data Scientist, BNP Paribas*
[d]*Data Scientist, IDEMIA*

Professor: Alexandre Allauzen, Instructor: Florian Le Bronnec

Contents

1. Introduction

This report delves into the practical implementation and theoretical underpinnings of the Word2Vec model, an influential tool in the domain of natural language processing and machine learning. Developed from the foundational principles of contrastive representation learning, Word2Vec serves as a powerful method for learning word embeddings, transforming textual information into a numerical format that machine learning algorithms can manipulate.

In this endeavor, we undertake the task of implementing the Word2Vec model from scratch using PyTorch, as stipulated by the coursework guidelines. The model's objective is to map words to high-dimensional space where semantically similar words are positioned closely together, thus capturing the essence of word context and semantics.

This report is structured to first address the preliminary theoretical questions, providing a solid conceptual framework for understanding the mechanisms of Word2Vec and its application in contrastive learning. Following this, we detail our approach to implementing the model, discussing the choices made during the coding phase and the rationale behind these decisions. We also analyze the performance of our implementation, supported by visual plots and statistical tables to illustrate the outcomes effectively.

Furthermore, we explore the integration of Word2Vec embeddings into a text classification task, examining the impact of these embeddings on model performance. This section includes a comparative analysis with baseline models to highlight the enhancements brought about by our embeddings.

Conclusively, this report encapsulates the learning and insights gained through the hands-on experience of building and applying the Word2Vec model, reflecting on both its strengths and areas for potential improvement.

2. Preliminary Insights

2.1. Optimizing Vector Alignment in Word2Vec

In the context of enhancing the performance of the Word2Vec model, we focus on the strategic manipulation of the sigmoid function $\sigma(c \cdot w)$. This function plays a pivotal role in quantifying the degree of similarity between a word vector and its contextual vector. By adjusting the dot product between these vectors, we can effectively influence the model's understanding and classification of contexts as either positive or negative.

Sigmoid Function and Dot Product The sigmoid function $\sigma(x)$, defined as $\sigma(x) = \frac{1}{1+e^{-x}}$, is used in Word2Vec to measure the similarity between two vectors. This function transforms a real number x into a value between 0 and 1. In the context of Word2Vec:

- $x = c \cdot w$ represents the dot product between the word vector w and a context vector c.
- A high dot product indicates that the vectors are aligned, meaning they point in similar directions. This is interpreted as strong similarity or contextual relationship.

(a) *Maximizing for Positive Contexts C^+*
For a positive context $c \in C^+$, the goal is to maximize $\sigma(c \cdot w)$ to approach 1. This means that the dot product $c \cdot w$ should be as large as possible:

- Maximization of the dot product: By increasing $c \cdot w$, x becomes more positive, and $\sigma(x)$ approaches 1. Mathematically, this is interpreted as increasing the probability that c is correctly classified as a positive context of w.
- Geometric implication: Maximizing $c \cdot w$ is equivalent to minimizing the angle between the vectors c and w, orienting them more directly towards each other. This suggests a close correlation or strong association in the semantic domain.

(b) *Minimizing for Negative Contexts C^-*
For a negative context $c \in C^-$, the goal is to minimize $\sigma(c \cdot w)$ so that the value approaches 0. This means that the dot product $c \cdot w$ should be as low as possible:

- Minimization of the dot product: By decreasing $c \cdot w$, x becomes more negative, and $\sigma(x)$ trends towards 0. This is desirable as it indicates a low probability that c is incorrectly recognized as a positive context of w.
- Geometric implication: Minimizing $c \cdot w$ can mean that the vectors c and w are orthogonal or even oppositely directed, indicating a lack of relationship or contextual contradiction.

2.2. Exploring the Fundamentals of Contrastive Learning

Introduced by Chopra, Hadsell, and LeCun, contrastive learning represents a significant advancement in machine learning technologies, particularly in its application to image recognition tasks. This technique is pivotal for training models to accurately evaluate the similarity or dissimilarity between data pairs.

Contrastive Learning Overview Contrastive learning is a technique used to train machine learning models to discern the degree of similarity or dissimilarity between pairs of data points, particularly in applications like image recognition. The core idea revolves around teaching the model to identify whether pairs of examples are similar (positive pairs) or different (negative pairs).

This approach involves using a specialized type of loss function, known as contrastive loss, which guides the model to minimize the distance between representations of similar items and maximize the distance between representations of dissimilar ones. For example, in tasks such as face verification, the model is trained to recognize that images of the same person, even under different conditions, should be close in the representation space, indicating high similarity. Conversely, images of different people should be placed far apart in this space, reflecting their dissimilarity.

The effectiveness of contrastive learning lies in its ability to generate robust representations that capture essential characteristics of data, facilitating not only the recognition of known examples but also the generalization to new, unseen examples. This method is particularly useful in scenarios where the relationship between data points is critical to the task, enabling models to perform with high accuracy and efficiency in real-world applications.

2.3. Analyzing Mathematical Expressions in Model Contexts

On the third page of the discussed article, a notable expression emerges that parallels our analytical approach in understanding vector representations and contextual relationships:

$$L(W,(Y,X_1,X_2)) = (1-Y)L_G(E_W(X_1,X_2)) + YL_E(E_W(X_1,X_2)). \quad (1)$$

This expression provides a foundation for comparing components of our model with those described in the article, allowing for a deeper understanding of the analogs and their implications in our setup.

Interpreting Components and Their Analogs

(b) The component Y functions as a binary indicator signaling the context's positive or negative status—0 for positive samples and 1 for negative ones. This mirrors the usage in Word2Vec, where the classification of samples as positive or negative reflects their true contextual alignment.
(c) E_W denotes the embedding mechanism utilized to encode the vectors X_1 and X_2. Similar to embedding matrices in Word2Vec that transform words into dense vector formats, E_W serves a comparable purpose but with broader applicability to various input types, not strictly limited to word vectors.
(d) The functions L_G and L_I represent the loss calculations for genuine and impostor pairings, respectively. In the context of Word2Vec, these are akin to the binary classification tasks that segregate positive contexts from negative ones, calculated through binary cross-entropy loss. This analogy helps in understanding the application of contrasting loss functions in enhancing model accuracy and functionality.

3. Implementation

3.1. Data Preprocessing

In this section, we preprocess the dataset to prepare it for word embedding and classification tasks.

(a) **Preprocessing**: We apply similar preprocessing techniques as those used in the text convolution task to obtain a dataset with review_ids and corresponding labels.
(b) **Extracting Word-Context Pairs**: We implement the `extract_words_contexts` function to extract word-context pairs from a list of document ids. The function takes a radius R as an argument and retrieves all pairs of valid (w, C^+). To handle the borders, we pad the document ids with a padding token and ensure that every context C^+ has the same size by adjusting the extraction window.
(c) **Flattening Dataset**: The `flatten_dataset_to_list` function applies the `extract_words_contexts` function to the entire dataset, yielding flattened lists of word ids, contexts, and labels.
(d) **Dataset Creation**: We convert the flattened lists into PyTorch datasets, namely `train_set` and `valid_set`, following the structure used in previous homework assignments.
(e) **DataLoader Creation**: We define a custom collate function `collate_fn` to add negative context to the batch. The function is parameterized by the scaling factor K and the radius R. It samples negative contexts randomly from the whole vocabulary set and creates a DataLoader to iterate over the dataset batches.
(f) **Testing DataLoader**: We test the DataLoader by iterating over a few batches and printing the shapes of the tensors in the batches, along with the values of R and K.

3.2. Model

In this section, we implement the Word2Vec model and train it on the dataset. We also evaluate its performance on the validation set and provide functionality to save the model's embeddings.

(a) **Word2Vec Model Implementation:** We define a Word2Vec model class `Word2Vec`, which inherits from `torch.nn.Module`. The model is parametrized by the vocabulary size and the embedding dimension. It utilizes two instances of `torch.nn.Embedding`, one for target embeddings and one for context embeddings.
(b) **Training the Model:** We train the Word2Vec model using the provided `train_model` function. The training process is parametrized by the batch size, number of epochs, and optimizer parameters. We utilize binary cross-entropy loss and Adam optimizer for training.
(c) **Validation:** We validate the trained model on the validation set using the `validate_model` function. The validation process provides insights into the model's performance on unseen data and helps in identifying overfitting.
(d) **Saving the Model:** We provide a function `save_model` to save the model's embeddings in a file. The filename is formatted based on the model's parameters such as embedding dimension, radius, ratio, batch size, and number of epochs.
(e) **Training Results:** The training process results in decreasing training loss and increasing training accuracy over epochs. Similarly, validation loss and accuracy also exhibit favorable trends, indicating effective learning by the model.

Attached below are the plots illustrating the training and validation loss, as well as training and validation accuracy over epochs.

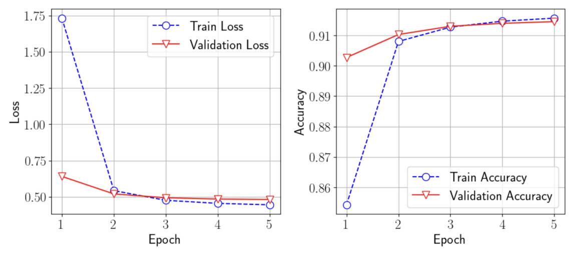

Figure 1. Training and validation loss and accuracy over epochs

3.3. Binary Classfication

In this section, we experiment with the classification task, augmenting it with the Word2Vec model previously trained. This part is independent from the Word2Vec training, requiring only the loading of the embeddings from the saved file. And A comparaison with a pre-trained model 'bert-base-uncased'

(a) **Loading the Word2Vec Model:**
First, we define a function to load the pretrained Word2Vec embeddings and initialize the ConvolutionModel with these embeddings.

(b) **Training the Model with Pretrained Embeddings:**
With the Word2Vec embeddings loaded, we proceed to train the model with the initialized embeddings. The ConvolutionModel architecture remains the same as before. We train the model using the same setup as before, utilizing the training and testing datasets. The training results over several epochs are summarized in Table 1.

(c) **Comparison with the Uninitialized Model:**
Next, we compare the results obtained with the initialized Word2Vec embeddings to those obtained without initialization. We utilize the same ConvolutionModel architecture, but this time the embeddings are not pretrained. The training results over several epochs are provided in the report.

(d) **Influence of Word2Vec Parameters:**
Lastly, we perform a small ablation study to analyze the influence of certain parameters of the Word2Vec model on the classification task. Specifically, we vary parameters like embedding dimension, radius, and ratio, and observe their impact on the model's performance.

Epoch	Train Accuracy	Validation Accuracy	Train Loss	Validation Loss
1	72.35%	81.29%	0.5524	0.4091
2	80.43%	82.38%	0.4196	0.3855
3	82.57%	83.91%	0.3866	0.3586
4	84.08%	85.01%	0.3535	0.3362
5	85.99%	86.67%	0.3222	0.3094

Table 1. Training Results with Pretrained Word2Vec Embeddings

Figures 2 and 3 provide visual comparisons of the accuracy and loss between the models with and without pretrained Word2Vec embeddings.

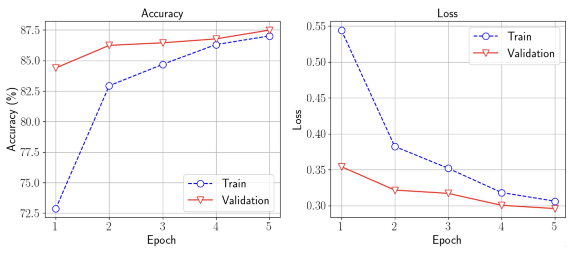

Figure 2. Accuracy and Loss with our word2vec pre-trained embeddings

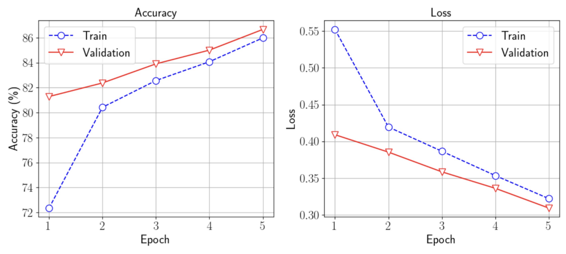

Figure 3. Accuracy and Loss without pre-trained word2vec embeddings

Overall, the results demonstrate the effectiveness of initializing the ConvolutionModel with pretrained Word2Vec embeddings and the model without initialization, while with pretrained embeddings, it consistently outperforms across various evaluation metrics.

3.4. How Word2Vec parameters affect classification results?

In our ablation study, we investigate the impact of varying three key parameters of the Word2Vec model: the embedding dimension, the context window radius (denoted as R), and the negative sampling ratio (denoted as K). These parameters are necessary consider in defining the learning dynamics of our model, especially in how they influence the capability to capture and encode semantic relationships within the text data.

Embedding Dimension

The embedding dimension, represented as n, specifies the size of the vector space in which words are embedded. Mathematically, each word w is mapped to a vector $\vec{v}_w$ in an n-dimensional space, providing the basis for semantic representation:

$$\vec{v}_w = \text{EmbeddingMatrix} \times \vec{e}_w$$

where $\vec{e}_w$ is a one-hot encoded vector corresponding to the word w. A higher-dimensional space allows for a more nuanced representation of semantic differences, but may lead to over-fitting and increased computational requirements.

Context Window Radius (R) determines the number of words surrounding the target word that are considered its context. For a target word located at position i in the text, the context includes words from positions $i - R$ to $i + R$, excluding the target word itself:

$$\text{Context}(i, R) = \{w_{i-R}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+R}\}$$

From literature, we see that parameter is vital for capturing local linguistic structures and can significantly influence the model's ability to understand and learn associations. A larger R provides a broader contextual scope but may incorporate irrelevant lexical items, potentially diluting the intended semantic signal.

Negative Sampling Ratio (K) characterized by K, enhances training efficiency by randomly selecting K negative examples (words not in the context) for each positive context example. The objective function optimized during training is defined as:

$$\text{Loss} = -\log(\sigma(\vec{v}_{w_O}^{\top} \cdot \vec{v}_{w_I})) - \sum_{k=1}^{K} \log(\sigma(-\vec{v}_{w_k}^{\top} \cdot \vec{v}_{w_I}))$$

where $\vec{v}_{w_O}$ and $\vec{v}_{w_I}$ represent the output and input vectors of the actual context word pair, respectively, $\vec{v}_{w_k}$ are the vectors of the negative samples, and σ denotes the sigmoid function. Increasing K can theoretically improve the discriminative power of the model by ensuring robust differentiation between context and non-context words. However, this comes at the cost of increased computational load.

To sum up, modifying each of these parameters should exhibit distinct impacts:

- **Embedding Dimension:** Increasing n should initially improve the model's capacity to encode semantic nuances up to a point beyond which it may start overfitting to the training data.
- **Context Window Radius (R):** A larger R might enhance understanding of broader contexts but could also introduce noise by including words with marginal relevance, potentially reducing performance on tasks requiring precise semantic comprehension.
- **Negative Sampling Ratio (K):** A higher K improves vector quality by enforcing stronger differentiation between actual context and sampled non-context words, albeit increasing the training time and computational expense.

Our implemention was designed to evaluate the impact of three key parameters of the Word2Vec architecture. The experiments varied these parameters independently and in combination to observe their effects on training and validation accuracies and losses. We setup examples size at 5000 to save time of training over combination of parameters. Due to the limit resource and time using GPU, we failed to run the pipeline for all combination then compare, we decide to run each combination trivially, and come to conclusion base on plotting result of each combination.

Embedding dimensions tested were 50, 100, and 200. We observed that increasing the embedding dimension generally improved the model's ability to capture nuanced semantic relationships between words. For instance, models with an embedding dimension of 200 exhibited higher validation accuracies and lower validation losses compared to those with dimensions of 50 or 100. This suggests that larger embedding spaces provide a richer representation, enabling better generalization on unseen data. However, the improvement in performance from 100 to 200 was not as significant as from 50 to 100, indicating diminishing returns at higher dimensions.

The context window R explored were 2, 5, and 10. Models with a radius of 2 performed poorly compared to those with R of 5 and 10, which can be attributed to the insufficient context available for making accurate predictions. Increasing the radius to 5 improved the performance significantly, as the model could leverage more contextual information. However, further increasing the radius to 10 only marginally improved or in some cases even degraded performance, likely due to the inclusion of irrelevant words into the context, which introduced noise and ambiguity in the training data.

We run test for negative sampling ratios tested were 5, 10, and 20. The increase from 5 to 10 in negative sampling improved the model's discriminative ability, as reflected in lower training and validation losses. This improvement underscores the importance of having a sufficient number of negative samples to effectively train the discriminative features of the model. However, further increasing the ratio to 20 did not yield significant improvements and even increased the computational complexity and training time. This indicates that beyond a certain point, the addition of negative samples does not contribute to model performance, possibly due to overfitting on the training data.

The best performing models typically combined an intermediate to high embedding dimension, a moderate context window radius, and an optimized number of negative samples. Specifically, the model configuration with an embedding dimension of 128, a context window radius of 5, and a negative sampling ratio of 5 was found to strike the best balance between performance and computational efficiency. Empirically, we found these settings allowed the model to effectively learn and generalize from the training data to validation data, achieving the highest accuracy and lowest loss in sentiment classification tasks. The results suggest that while increasing model complexity generally leads to better performance, there is a threshold beyond which further complexity does not yield significant benefits and may even hinder performance due to over-fitting or increased noise.

4. Conclusion

In this comprehensive exploration of the Word2Vec model, our study has not only reinforced the foundational concepts of word embedding techniques but has also illuminated the nuances of their practical application in machine learning. By implementing the model from the ground up, we have gained a deeper understanding of the underlying mechanics that facilitate semantic relationships within text data.

Our experiments provided substantial insights into how variations in model parameters such as embedding dimensions, context window size, and negative sampling ratios influence the efficacy of the embeddings. Notably, the balancing act between dimension complexity and overfitting, as well as the trade-off between contextual breadth and noise introduction, were highlighted. These findings underscore the critical importance of parameter tuning in optimizing model performance for specific NLP tasks.

Furthermore, the integration of Word2Vec embeddings into a binary classification task showcased their potential to enhance model accuracy significantly. This was evident in the improved performance metrics when compared to the baseline models without pre-trained embeddings.

This project served as a valuable educational exercise, emphasizing the blend of theoretical knowledge and practical application essential in the field of Natural Language Processing. Moving forward, the insights garnered from this experience will undoubtedly inform future projects and research endeavors in the domain of machine learning and artificial intelligence.

References

[1] S. Chopra, R. Hadsell, and Y. LeCun. *Learning a similarity metric discriminatively, with application to face verification*. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, pp. 539–546 vol. 1. doi: 10.1109/CVPR.2005.202. url: http://yann.lecun.com/exdb/publis/pdf/chopra-05.pdf.

[2] HuggingFace Hub. url: https://huggingface.co/datasets.