Words Embedding and News Topic Classification

Jean Lucien Randrianantenaina
Applied Mathematics, Machine Learning and Artificial Intelligence
Stellenbosch University

***Abstract*—Processing text documents is crucial in Natural Language Processing; however, computers cannot process text directly in its raw form. To handle this problem, one may use one-hot encoding, which misses a lot of useful information and word relationships from the document. In this work, we explore a powerful word embedding technique called Word2Vec, specifically the Skip-Gram and the Skip-Gram with negative sampling variant, trained on the AG News dataset. We also build a text classification model on top of the word embeddings that achieves an accuracy of 90%.**

I. Introduction

Natural Language Processing (NLP) is a field of artificial intelligence that focuses on the interaction between computers and human languages. One fundamental task in NLP is transforming raw text into a format that computers can understand and analyze, essential for applications like text classification, sentiment analysis, and machine translation.

One-hot encoding is a simple method to represent text, assigning a unique vector to each word. However, it leads to sparse vectors that fail to capture semantic similarities between words, resulting in a loss of valuable information.

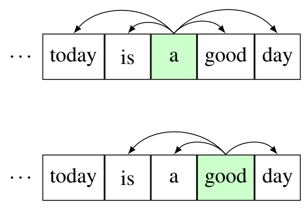
To address these limitations, more sophisticated techniques like word embeddings have been developed. Word embeddings are dense vector representations that capture word meanings by placing similar words closer together in the vector space. One influential technique is Word2Vec, introduced by Mikolov et al., which uses neural networks to learn word associations from large corpora. To see how it works, we discuss some generaly about Skip-Gram and text classification in Section II, theb we detail our data preparation and implementation in Section III Finally, we present our results in the SectionIV.

II. Word Embeddings and Skip-Gram Model

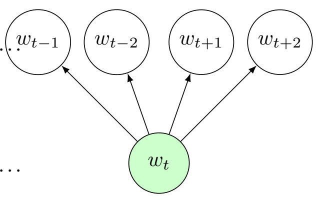
This section explores word embeddings and the Skip-Gram model, a popular method for learning word representations. The Skip-Gram model, proposed by Mikolov et al. [2]. It predicts surrounding context words given a central word. This approach allows the model to learn the semantic meaning of words based on their co-occurrence in text.

A. Skip-Gram

The Skip-Gram model models the word representation by considering a centre word c and predicting $2M$ (can be less) words that are around c. The M is referred to as window size, so we have at most M words on the left and M words at the right of the centre words. Figure 1 illustrates a skipgram model with a window size of 2.



(a) Skip-Gram Illustraion



(b) Skip-Gram Model

Fig. 1: The skip-gram model

Since, these context words are predicted using the probability $p(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}|w_t)$, which is simplified by the following assumption:

- Each window is an identically and independently distributed sample.
- Within each window, each context word is conditionally independent given the centre word.
- Each context word can be predicted from the centre word in the same way, irrespective of its position.

Considering the sequence $w_{1:T}$ and a window size M, we can derive the negative likelihood of our model written as follows, with the two first assumptions:

$$J(\theta) = -\log \prod_{t=1}^{T} P_\theta(w_{t-M:t-1}, w_{t+1:t+M}|w_t) \tag{1}$$

$$= -\sum_{t=1}^{T} \sum_{\substack{-M \le j \le M, \\ j \ne 0}} \log P_\theta(w_{t+j}|w_t) \tag{2}$$

Where θ is the parameter of the model. Now, to obtain $P_\theta(w_{t+j}|w_t)$ we consider the third assumption. We also denote the vector representation of a context word w by $\mathbf{u}_w$, and $\mathbf{v}_w$ for a centre word w. Thus, probability for a context words o given a centre word c is given by:

$$P_\theta(w_{t+j} = o|w_t = c) = P(o|c) = \frac{e^{\mathbf{u}_o^\top \mathbf{v}_c}}{\sum_{w\in\mathcal{V}} e^{\mathbf{u}_w^\top \mathbf{v}_c}} \tag{3}$$

where $\mathcal{V}$ is the set of vocabulary. More generally we have

$$\mathbf{f}_\theta(w_t = c) = \frac{1}{\sum_{w\in\mathcal{V}} e^{\mathbf{u}_w^\top \mathbf{v}_c}} \begin{bmatrix} e^{\mathbf{u}_o^\top \mathbf{v}_c} \\ e^{\mathbf{u}_o^\top \mathbf{v}_c} \\ \vdots \\ e^{\mathbf{u}_o^\top \mathbf{v}_c} \end{bmatrix} = \operatorname{softmax}(\mathbf{U}\mathbf{v}_c) \tag{4}$$

where $\mathbf{f}_\theta(\cdot)$ is our model.

Furthermore, the parameter of the model can be viewed as two matrices $\mathbf{V}$ and $\mathbf{U}$, both with a size of $D \times |\mathcal{V}|$. Each column represents the vector embedding of a centre word for $\mathbf{V}$, and a context word for $\mathbf{U}$. The optimal parameter can be found using the gradient-based optimisation method. After training the model we use $\mathbf{V}$ as the embedding and discard U.

One issue with this approach is the computational complexity induced by the softmax when we deal with a large dataset. A solution to this is proposed in [3] that we will present in the next section.

B. Improving skip-gram with negative sampling

Before discussing the negative sampling concepts. In the same paper, the author discussed the usage of a subsampling method. This method consists of discarding some frequent words based on a heuristic probability:

$$p(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{5}$$

Where $f(w_i)$ if the frequency of the word w_i, and t a choosen threshold suggested to be 10^{-5}. By doing this, we reduce the size of our vocabulary while keeping more informative words. However, instead of applying this approach, we will just remove some of the common words known as stopwords from our data for the sake of simplicity.

On the other hand, the Noise Contrastive Estimation says that a good model should be able to differentiate data from noise by the mean of a binary logistic regression [3]. Based on that, for each pair (c, o) (positive) we sample K pairs (c, w_i) (negatives) then, we can create a model that predicts if a given pair is a positive pair or not. This approach yields the following negative log-likelihood for a single pair with the corresponding negative sample.

$$J_{c,o}(\theta) = -\log \sigma(\mathbf{u}_o^\top \mathbf{v}_c) - \sum_{i=1}^{k} \log \sigma(\mathbf{u}_{w_i}^\top \mathbf{v}_c) \tag{6}$$

This approach reduces considerably the computational complexity because instead of computing a huge softmax, we compute some simple sigmoid function.

C. Words embedding evaluation

This section discusses how to evaluate word embeddings and how to use them in text classification. A common way to see if the learned embedding makes sense is to see if similar words or related words are located in a specific region of the embedding space (close). This is achieved by computing the $\cos$ine of their embedding vectors. So for two vectors $\mathbf{u}$ and $\mathbf{v}$ we have:

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}^\top \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|} \in [-1, 1] \tag{7}$$

Words with similar meanings will have a cosine similarity close to 1, while dissimilar words will have a value closer to -1. Additionally, the distance between two embeddings is calculated using:

$$\mathrm{d}(\mathbf{u}, \mathbf{v}) = 1 - \cos(\mathbf{u}, \mathbf{v}) \in [0, 2] \tag{8}$$

However, evaluating the overall quality of word embeddings is a complex task. So, in addition the the cosine similarity and distance we can do the following approach:

- **Visualization:** techniques like PCA, t-SNE, and UMAP can be used to project word embeddings into a lower-dimensional space for visual inspection. By looking at known similar words and how they cluster together, we can gain insights into the quality of the embeddings.
- **Extrinsic evaluation:** generally using the words embedding in a downstream task like text classification, or other real-world problems.
- **Intrinsic evaluation:** This method uses words analogy or asses the correlation with human word similarity judgement.

As mentioned, a downstream task can be used to evaluate the word embedding, so in the next subsection, we use it for a text classification task.

D. Text Classification

To perform a text classification using the embedded words, we compute the average embedding of the text/sentence:

$$\mathbf{w}_{\text{sentence}} = \frac{1}{T} \sum_{T}^{t=1} \mathbf{w}_t \tag{9}$$

where $\mathbf{w}_t$ is the embedding of the t$^{\text{th}}$ word and T is the total number of words. Then, we can feed it into a Simple multilayer perceptron to predict the associated class.

III. Methodology and implementation

This section presents the implementation of the word2vec model with negative sampling on the AG news dataset. The dataset contains news articles from four classes (World, Sports, Business, and Sci/Tech), with a title and description field for each article. The dataset is split into separate training and test partitions.

A. Data Preparation

The first task is to normalize the text, to be uniform and easy to treat. To achieve that we apply the following process:

- Split the dataset into three separate files: title, description, and class label.
- Remove common stop words using a list from [4]
- Remove HTML tags that can be handled.
- Replace all numbers (either preceded or followed by letters) with the token $< \text{num} >$.
- Replace all diacritics by their simple form.
- Remove all non-alphanumeric characters by a space.
- Convert all text to lowercase.
- Remove all trailing spaces and multiple spaces.

We also introduce two special tokens $< \text{unk} >$ to represent an unseen word of the test set and $< \text{pad} >$ to complete a sentence to have the same length in a batch. These two will be useful for the text classification task.

B. Word2vec Implementation

The word2vec model is implemented as a neural network with two layers. The first one represents $\mathbf{V}$ and the second one $\mathbf{U}$, and there is no activation function between these two layers. The softmax function is already included in the `CrosEntropyLoss` of `PyTorch` so we do not need to include it. We use the module `nn.Embedding` of `PyTorch` for $\mathbf{V}$ and $\mathbf{U}$[1] to be more efficient instead of using a one-hot vector to extract the corresponding embedding vector. These layers act like a look-up table that maps each index of the words into the corresponding vector representation.

C. Negative sampling

To sample the negative sample follow the suggestion in [3], to use the heuristic probability

$$P_n(w) = \frac{U(w)^{3/4}}{Z} \tag{10}$$

when sampling, where U is the unigram distribution. We do not handle if some of the contexts of the centre word are among the negative sample, as it can be rare, and later re-adjuster when the pair is considered as a positive pair.

These negative sample are sampled during after constructing the positive pairs, it avoids sampling at each batch which can slow down the training process.

D. Embedding Visualization

For the visualization, we use TensorBoard, which allows us to have an interactive visualization and access to UMAP, t-SNE and PCA for the projection.

IV. Results and Discussion

After setting the two models and data set, we discuss and present the results of our experimentation in this section.

A. Words embedding parameter

The Table I list the hyper parameters related to the two words embedding models, where SGNS means Skip-Gram with Negative Sampling.

	Skip-gram	SGNS
Window size	3	3
Negative samples	10	N/A
Vocabulary size	56 465	56 465
Number of words	1 944 289	1 944 289
Number of pairs	4 637 822	4 637 822
Batch size	4096	4096
Embedding dimension	256	256
Learning rate	10^{-3}	10^{-3}
Number of iterations	30	5
Sec./epoch	60	300

TABLE I: Hyperparameter for the Skip-gram and SGNS models.

We observe that SGNS is very fast compared to the vanilla skip-gram, due to that fact, we train it for only five epoch.

[1]`nn.Linear` for $\mathbf{U}$ with the vanilla Skipgram

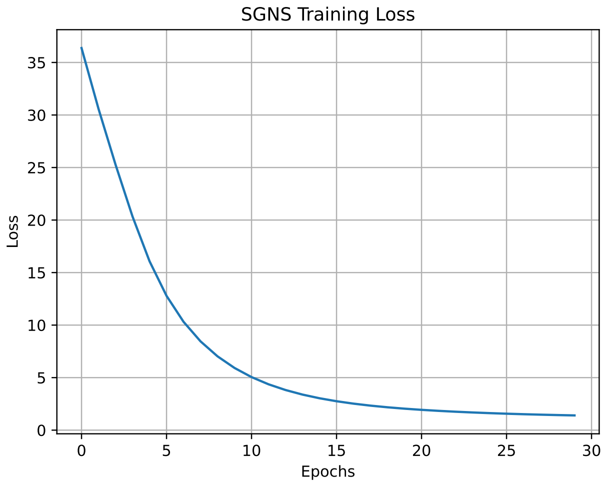


Fig. 2: SGNS Loss

The figure 2 show the loss function of the SGNS, whis is monotonically decreasing.

Word	Similarity	Word	Similarity
google		**football**	
search	0.583	game	0.626
online	0.570	team	0.595
data	0.544	players	0.556
internet	0.525	time	0.555
linux		**food**	
software	0.600	business	0.392
internet	0.582	giant	0.383
technology	0.557	industry	0.380
products	0.538	customers	0.372
windows		**money**	
software	0.605	business	0.461
users	0.600	service	0.460
market	0.548	corp	0.457
search	0.547	management	0.453
war		**police**	
country	0.649	killed	0.627
government	0.626	military	0.603
iraq	0.623	people	0.601
officials	0.594	country	0.597

TABLE II: Word Similarities

We can effectively observe it the table II that most of these words are related, linked and similar toi the query, whic confirm the effectivenef of our embding method with SGNS. We may also got the similar results with the Vanilla Skip-gram but, due to time and space we are not able to provide the results here.

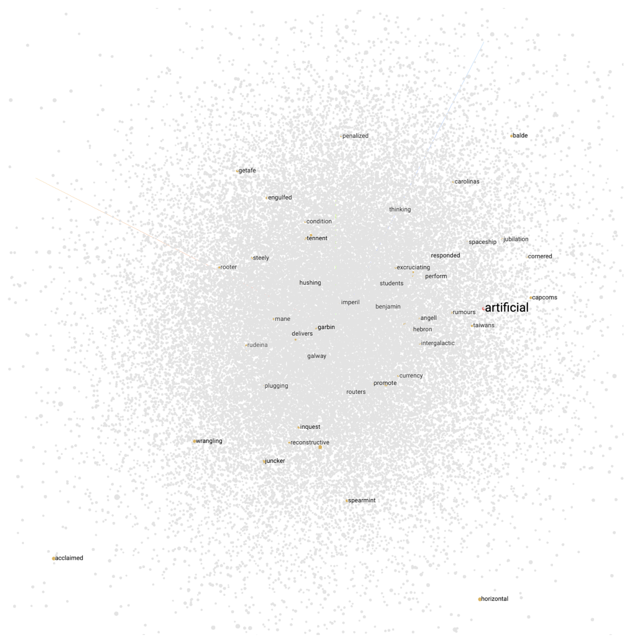


Fig. 3: Tensorboard UMAP visualization

The Figure 3 show a visualization with TensorBoard wher the query is 'artificial'.

B. Text classification

With two layers, we obtain an accuracy of 90% on the test set. The training was fast, We observe that SGNS is very fast compared to the vanilla skip-gram, due to that fact, we train it for only five epoch, so the results is not yet good enought.

C. Text Classification

After traingin the text calssifier for 70 epocs, we observe its loss and accuracy in the Figure 4.

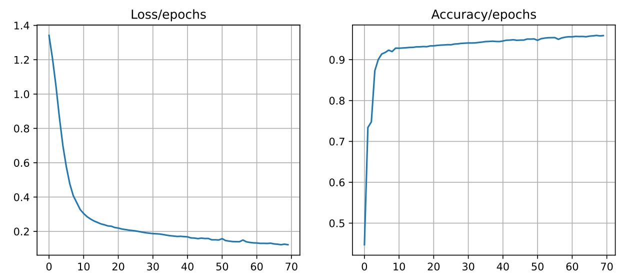


Fig. 4: Loss and accuracy training of the text classiifiers

On the test set we obtain ana ccuracy of 90%, and the Figure 5 show it confusion matrix.

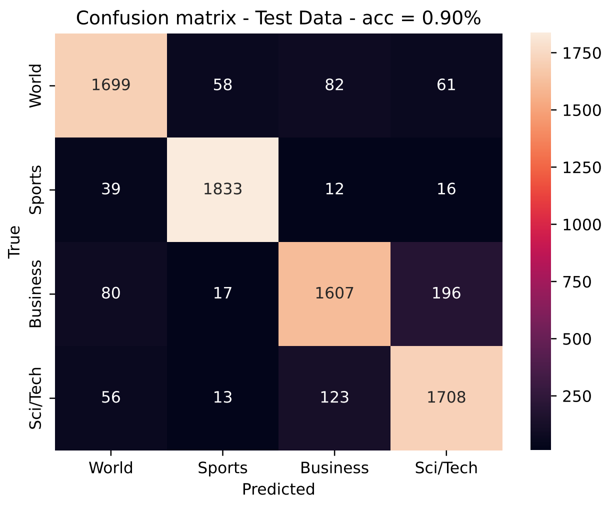


Fig. 5: Confusion matrix of the text classifier on the test set

We see that it susffer on the Business and Sci/Tech, it may be due to some similarity between thes class.

V. Conclusion

References

[1] Herman Kamper. *NLP817*. https://www.kamperh.com/nlp817/, 2022–2024.

[2] Mikolov, T., Chen, K., Corrado, G. and Dean, J., 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[3] Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S. and Dean, J., 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems, 26*.

[4] English stopwords list https://github.com/stopwords-iso/stopwords-en/blob/master/stopwords-en.txt

[5] Andrew Ng, Negative Sampling https://www.youtube.com/watch?v=4PXILCmVK4Q