Skip-Gram with negative-sampling

CentraleSupélec - NLP Execrcice 1

Omari-Betahi Yassine
Department of Applied Mathematics
CentraleSupélec
`yassine.omari-betahi@student.ecp.fr`

Daoudi Yasmina
Department of Applied Mathematics
CentraleSupélec
`yasmina.daoudi@supelec.fr`

Souaidi Omar
Department of Applied Mathematics
CentraleSupélec
`omar.souaidi@student.ecp.fr`

Jebbari Othmane
Department of Applied Mathematics
CentraleSupélec
`othmane.jebbari@student.ecp.fr`

1 Introduction

Traditional machine learning approaches (e.g. TF-IDF / Count Vectorizer) consider words and their combinations as unique dimensions of the feature space. Now, deep learning models embed words as vectors in a low-dimensional continuous space where dimensions represent shared latent concepts. The main advantages of this approach are :

- The ability to capture similarity between words and use that similarity for various tasks (topic modelling, auto-completion ...)
- Escaping the curse of dimensionality

Word embeddings can be initialized randomly and learned during training, or can be pre-trained. Pre-trained word vectors obtained with word2vec from very large corpora (wikipedia, twitter ...) are often used, one famous word2vec is GloVe [1].

2 Related Work

The original paper introducing skip-gram is [2], they also introduce another word2vec variant : CBOW. Negative sampling was proposed in a follow-up paper [3].
The Skip-gram model attempts to predict the source context words (surrounding words) given a target word (the center word). Let's consider a simple sentence like **"The quick brown fox jumps over the lazy dog"**, the skip-gram model's aim is to predict the context words from the target word. Hence the task becomes to predict the context **[quick, fox]** given target word **"brown"** or predict the context **[the, brown]** given target word **"quick"**. Therefore, the model tries to predict the context window words based on the target word (which is the reverse of what CBOW model does).

3 Project Objectives

In this small project, we will learn about the skip-gram with negative sampling method. First, we will derive the underlying formulas (computing the gradient, updating the word vectors , ...) and implement the model from scratch, then we will train the model and evaluate it. We have two main evaluations: **Similarity Estimation** and **IMDB Review Classification**.

3.1 Similarity Estimation

First, we will train (pre-train) it on a subset (10 %) of the "The One Billion Word dataset"[4], which is a well-know dataset that serves as a benchmark for language models. The training data was produced from the WMT 2011 News Crawl data using a combination of Bash shell and Perl scripts link to the project. Hence, we will evaluate it using the Simlex999 method[5], which provides a way of measuring how well models capture similarity, rather than relatedness or association. We will compute the cosine similarity between each pairs of words given by the simlex dataset using our pre-trained word2vec and compare it with the one of Simlex by using the spearman correlation. According to their website, The well-known Skipgram (Word2Vec) model trained on 1bn words of Wikipedia text achieves a Spearman Correlation of 0.37 with SimLex-999, we will attempt to reach a similar performance.

3.2 IMDB Review Classification

Secondly, we will see if it benefits to use word2vec pre-trained models compared to traditional machine learning approaches (TF-IDF). To do so, we will take 25k review from the IMDB Review Large Dataset [6] (25k represents the train set of the dataset in kaggle Kaggle Link. We will split randomly these 25k reviews into three subsets (train $\simeq$ 21k reviews / Dev $\simeq$ 2k reviews / test $\simeq$ 2k reviews). First we will set a baseline using TF-IDF and Xgboost as a classifier and then use a deep learning architecture (Bi-lstm) so that we can use word embedding. We will make our benchmark on four word representations :

- One that we obtain by training our skipgram model directly on the 25k reviews
- One that we obtain by training our skipgram model on the subset (10 %) of The One Billion Word Dataset (3.1)
- GloVe : Common Crawl => 42B tokens, 1.9M vocab, uncased.
- One that we obtain by adding a trainable embedding layer (with random initialization) to the deep learning architecture's base.

4 Skip Gram

4.1 Main Objective

Skip-gram is trained on the task of context prediction. The word vectors are learned as an artifact of training (they are the parameters of the model). More precisely, pairs of target and context word (t, C_t) are sampled by sliding a window over the corpus $0, 1, ..., T$. For a given instantiation of the window, the word in the middle is the target word t, and the words surrounding it compose the set of positive examples C_t. Skip-gram is then trained to assign high probabilities to the words in C_t, given t,i.e, to predict well the context of a given target word. This translates into the following log-likelihood:

$$\underset{\theta}{\operatorname{argmax}} \sum_{t=0}^{T} \sum_{c \in C_t} \log p(c \mid t; \theta) \tag{1}$$

The set of parameters of the model, θ, contains two matrices of word vectors, W_t and W_c, from which are drawn the vectors of the words when they are used as targets and contexts, respectively. If we denote by V the vocabulary (unique words) and by d the dimensionality of the word embedding space, then W_t and W_c live respectively in $\mathbb{R}^{|V| \times d}$ and $\mathbb{R}^{d \times |V|}$. These two matrices are often referred to as input and output matrices, and it is common practice to use the input matrix, after training, as the final word embeddings.
The predictions $p(c \mid t; \theta) \quad \forall c \in C_t$ are given by looking at the entries of the vector in $\mathbb{R}^{1 \times |V|}$ obtained by passing the vector $w_t \in \mathbb{R}^d$ of the target word to a simple linear layer parameterized by W_c, That is, by multiplying w_t with matrix W_c, which is also equivalent to computing the dot product between w_t and each column w_c of W_c representing a context word. To make the calculus as a probabilities, we normalize with a softmax :

$$p(c \mid t; \theta) = \frac{e^{w_c \cdot w_t}}{\sum_{v \in V} e^{w_v \cdot w_t}} \tag{2}$$

4.2 Negative Sampling Trick

This approach being expensive because of the size of the vocabulary, the author in [3] proposes a trick called Negative Sampling.
Instead of computing the softmax over all the words in the vocabulary, negative sampling consists on computing the probability over a subset of the whole vocabulary, containing the context words C_t^+ and randomly sampled words from the remaining vocabulary C_t^- . Hence, we teach the model to distinguish between words from the true context C_t^+ of a given target word and negative examples C_t^-:

$$\underset{\theta}{\operatorname{argmax}} \sum_{t=0}^{T} \left[\sum_{c \in C_t^+} \log p(c \mid t; \theta) + \sum_{c \in C_t^-} \log(1 - p(c \mid t; \theta)) \right] \tag{3}$$

We now have independent binary classification tasks to perform. If we assume that the labels (pos/neg) are indicated by ± 1, the individual predictions can be obtained with the sigmoid function $\sigma(x) = \frac{1}{1+e^{-w_c \cdot w_t}}$. If we put this in Eq.3, and using the fact that $1 - \sigma(x) = \sigma(-x)$, we have our final loss :

$$\underset{\theta}{\operatorname{argmax}} \sum_{t=0}^{T} \left[\sum_{c \in C_t^+} \log(1 + e^{-w_c \cdot w_t}) + \sum_{c \in C_t^-} \log(1 + e^{w_c \cdot w_t})) \right] \tag{4}$$

Note: The negative examples are sampled according to their dampened frequency (square root of their frequency). This is what can be done in practice according to [7].

4.3 Gradient Descent

We are going to train our Skip-Gram with Stochastic Gradient Descent. We thus need to compute the gradient to perform updates at each training iteration.
For a given training example (t, C_t^+, C_t^-), we have :

$$L(t, C_t^+, C_t^-) = \sum_{c \in C_t^+} \log(1 + e^{-w_c \cdot w_t}) + \sum_{c \in C_t^-} \log(1 + e^{w_c \cdot w_t}) \tag{5}$$

Where L is the loss.
Let's compute all the partial derivatives that we need to compute the gradient update.

$$\frac{\partial L}{\partial w_{c^+}} = - \sum_{c \in C_t^+} \frac{w_t}{e^{w_c \cdot w_t} + 1} \tag{6}$$

$$\frac{\partial L}{\partial w_{c^-}} = \sum_{c \in C_t^-} \frac{w_t}{e^{-w_c \cdot w_t} + 1} \tag{7}$$

$$\frac{\partial L}{\partial w_t} = - \sum_{c \in C_t^+} \frac{w_c}{e^{w_c \cdot w_t} + 1} + \sum_{c \in C_t^-} \frac{w_c}{e^{-w_c \cdot w_t} + 1} \tag{8}$$

Hence the representation of each word is updated based on its corresponding partial derivative. For instance, for a positive word at iteration n, we perform the following update:

$$w_{c_{n+1}^+} = w_{c_n^+} - \gamma_n \frac{\partial L}{\partial w_{c_n^+}} \tag{9}$$

5 Experimental Details

5.1 Skip Gram

As you can see, we use a scheduled learning rate (decay at each iteration) where $\gamma_n = \frac{\gamma}{1+decayxn}$ where $decay = 10^{-6}$ and $\gamma = 0.03$. We use 5 as maximum window size, 10^6 as the number of windows to sample at each epoch, 5 as negative examples to sample for each positive and d = 64 the dimension of the embedding space.

More details about the code are provided in Appendix A. For both Dataset (IMDB 25k reviews + subset of 1b), we trained the skipgram for 25 epochs, which makes a total of 15 hours of training per dataset.

NOTE: We did not use the script to do the training: for some reason, training lasted too long (1500 hours per epoch). So, we decided to do it directly on a Google Colab notebook (writing functions on cells ...) without any script and surprisingly it took much less time. All the materials (notebook, files ...) can be found on our github.

5.2 IMDB Classification

For this task, we use as a baseline a TF-IDF vectorizer the sentence and an Xgboost classifier. Regarding the deep learning model, it is a simple Bi-LSTM with a linear layer at the top, the architecture and main components are represented in Figure 1 and Figure 2.

We used a checkpoint where we evaluate the model on the dev set at each epoch and save only the weight of the model that performed best on the dev set. We also reduce learning rate (by a factor of 0.5) when the accuracy on the dev set has stopped improving (patience = 2 epochs). The training last 20 epochs.

5.3 Results and Analysis

5.3.1 Skip Gram Spearman Autocorrelation

Using the similarities given by pairs of words in the Simlex-999 dataset and the one obtained by our skipgram, we got a spearman correlation of **0.04** with the skip-gram trained on IMDB and **0.10** with the skip-gram trained on the subset (10%) of [4].

We are far from the state of the art (Spearman correlation = 0.37), nevertheless we can notice that training on larger dataset (while mainting all the other parameters fixed) improve the correlation. We think that using more than 50% of the dataset can help us improve the autocorrelation, unfortunately we didn't manage to test that because of our limited computational ressources.

5.3.2 IMDB Review Classification

The result on the test set can be found in Table 1:

- **Fixed** means that we initialize the embedding layer with the skipgram output matrix and we freeze the layer
- **Trainable** means that we initialize the embedding layer with the skipgram output matrix and the layer remains trainable
- **Trainable layer** means that we initialize randomly the embedding layer and the layer remains trainable

We can see an improvement with the GloVe Embedding compared to the baseline score (TF-IDF + Xgboost) and the trainable layer configuration.

Unfortunately, the embedding matrix that we obtained by training skipgram on the IMDB dataset did not improve the score, in fact, we got a lower accuracy score using this dataset. This is probably due to the fact that even if we train it on the IMDB reviews, the dataset is not large enough for training a skip-gram model (plus our code isn't really optimized, we could have tried different hyperparameters).

The embedding matrix that we obtained by training skip-gram on the 10% of the one billion word dataset has performed much worse than the one with the IMDB skip-gram matrix.

6 Conclusion

According to our experimentations on the IMDB review classification, we can assume that we can benefit from using word2vec pretrained model compared to traditional machine learning.

We use the same skipgram algorithm (model + hyperparameters), but the intra skip-gram model (intra means that we train it on the dataset used to evaluate the classification task) outperforms the extra skip-gram model. Hence, we can assume that if we are given a certain task on a certain dataset X, it would be better to use a word2vec model pretrained on this same dataset X.

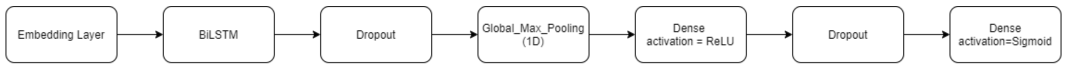

Figure 1: The detailed architecture of our classifier

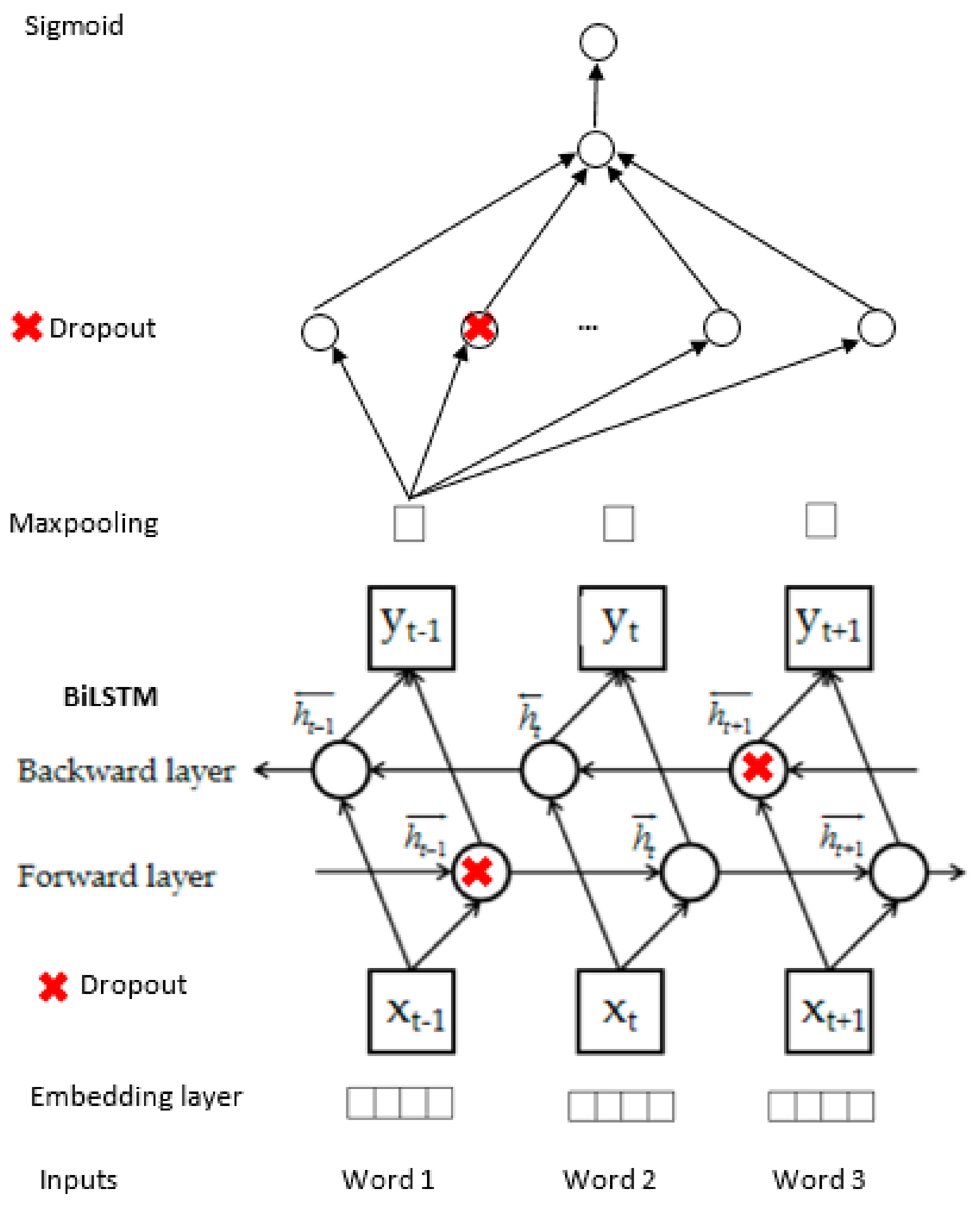

Figure 2: The architecture of our classifier

Model	Accuracy
TF-IDF + Xgboost	0.8650
Trainable Layer + BiLSTM	0.8775
Fixed GloVe + BiLSTM	0.8785
Trainable GloVe + BiLSTM	0.8820
Fixed IMDB (ours) + BiLSTM	0.7850
Trainable IMDB (ours) + BiLSTM	0.8425
Fixed 1B dataset (ours) + BiLSTM	0.5720
Trainable 1B dataset (ours) + BiLSTM	0.7500

Table 1: Accuracy score on test set.

A Appendix

First step is to preprocess the text :

1. Cleaning : strip extra white space, lowercase
2. Tokenize : using TweetTokenizer from nltk
3. Filter : keep tokens with a frequency larger than 5 in the corpus

Second step consists in creating the vocabulary :

1. Assign to each word an index based on its frequency in the corpus
2. Convert each sentence to integers using the index created

For the training, we initialize randomly following a normal distribution the matrices W_t and W_c.
At each epoch, we generate :

1. Windows (containing the context pairs and target)
2. Negative examples

In each window, we provide our model with the context pairs and negative examples. We compute the gradient using the equations (6), (7) and (8). The update via gradient descent is made using equation (9). The loss is computed using equation (5).
To compute the similarity between two words we extract their embedding using their index in the matrix W_t and use the cosine similarity from scipy.spatial.cosine.

References

[1] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[3] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[4] Ciprian Chelba, Tomás Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005, 2013.

[5] Felix Hill, Roi Reichart, and Anna Korhonen. SimLex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695, December 2015.

[6] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[7] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information, 2017.