# Skipgram from scratch

Matthias Gallé

Mohamed Alami Chehboune
CentraleSupelec
Paris, France
m.alamichehboune@student-cs.fr

Louis De Vitry
CentraleSupelec
Paris, France
Louis.devitry@student-cs.fr

## 1 SKIPGRAM ARCHITECTURE

First introduced by Mikolov and al in [3], Skipgram is an efficient method for learning high quality vector representations of words from a large amount of unstructured text data. its architecture is simple and dos not require heavy calculations as it only necessitates one hidden layer, while its input a unique vector. Its aim is to learn vectors or embeddings associated with every word of a given vocabulary, that should encode linguistic regularities and patterns. Therefore, the vectors of similar words should be close to each other. This latent representation of words thus allows to learn how to predict the context of a given word.

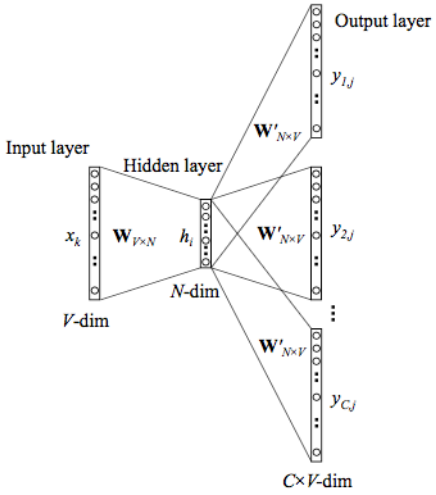


***Figure 1: Skipgram Architecture***

As displayed above, the architecture itself is quite simple with a single input vector, one hidden layer, and C output vectors.

The input vectors is a one hot encoded vector with dimension V. W is the weight matrix of dimension V*N. Therefore: $W^T X = h$

$$\begin{bmatrix} . & . & . & w_{1,k} & . & . & . \\ . & . & . & w_{2,k} & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & w_{N,k} & . & . & . \end{bmatrix} * \begin{bmatrix} . \\ . \\ 1 \\ . \\ . \end{bmatrix} = \begin{bmatrix} h_1 \\ . \\ h_i \\ . \\ h_N \end{bmatrix}$$

It turns out that $h$ is just the j-th column of $W^T$ or the j-th line of $W$. $h$ is just copying and transposing a row of the input → hidden weight matrix.

Through the same process, the output units are obtained by multiplying the hidden units by $W'^T$, $V * N$ weights matrix. $U = W'^T * h$

$$\begin{bmatrix} . & . & . & w'_{1,k} & . & . & . \\ . & . & . & w'_{2,k} & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & w'_{V,k} & . & . & . \end{bmatrix} * \begin{bmatrix} h_1 \\ . \\ h_i \\ . \\ h_N \end{bmatrix} = \begin{bmatrix} U_{c,1} \\ . \\ U_{c,i} \\ . \\ U_{c,V} \end{bmatrix}$$

$U_{c,i}$ is the score of the i-th output word in c-th panel. Therefore, the softmax function is needed to transform all these scores into probabilities.

$$p(w_{c,j} = w_{O,c}|w_I) = y_{c,j} = \frac{exp(u_{c,j})}{\sum_{j'=1}^{V} exp(u_{j'})}$$

* $w_{c,j}$ is the j-th word on the c-th panel
* $w_{O,c}$ is the actual c-th word in the output context words
* $w_I$ is the input word
* $y_{c,j}$ is the output of the j-th unit of the c-th panel
* $u_{c,j}$ is the input of the j-th unit on c-th panel.

However, as the output layer panels share the same weights: $u_{c,j} = u_j$ for $c = 1, 2, ..., C$

For Skipgram the loss function takes this form:

$$\begin{aligned} E &= -\log p(w_{O,1}, w_{O,2}, ..., w_{O,C}|w_I) \\ &= -\log \prod_{c=1}^{C} \frac{exp(u_{c,j_c^*})}{\sum_{j'=1}^{V} exp(u_{j'})} \\ &= -\sum_{c=1}^{C} u_{j_c^*} + C.\log \sum_{j'=1}^{V} exp(u_{j'}) \end{aligned}$$

$j_c^*$: the index of the actual c-th context word

## 1.1 Backpropagation

We first take the derivative of E regarding the input of every unit in every panel of the output layer $u_{c,j}$:

$$\frac{\partial E}{\partial u_{c,j}} = y_{c,j} - t_{c,j} = e_{c,j}$$

$t_{c,j} = 1\ if\ j = j^*\ and\ 0\ otherwise.$

This derivative is in fact the prediction error on the unit. Thus a $V$ dimensional vector $EI{=}EI_1, ..., EI_V$ is defined such that it is the sum of all prediction errors over all context words: $EI_j = \sum_{c=1}^{C} e_{c,j}$

### 1.1.1 *Hidden → output weights.*

The next step is to take the derivative of $E$ regarding to the hidden → output matrix W'.

$$\frac{\partial E}{\partial w'_{i,j}} = \sum_{c=1}^{C} \frac{\partial E}{\partial u_{c,j}} . \frac{\partial u_{c,j}}{\partial w'_{i,j}} = EI_j . h_i$$

The update equation for the hidden → output matrix $W'$ appears to be:

$$w'^{(new)}_{i,j} = w'^{(old)}_{i,j} - \eta . EI_j . h_i$$

This update equation needs to be applied to every element of the hidden → output matrix.

### 1.1.2 *Input → Hidden weights.*

Having obtained the update equations for $W'$, it is now possible to move on to $W$. The derivative of E on the output of the hidden layer is given by:

$$\frac{\partial E}{\partial h_i} = \frac{\partial E}{\partial u_{c,j}} . \frac{\partial u_{c,j}}{\partial h_i} = EI_j . w'_{i,j} = EH_i$$

The final step is to take the derivative of $E$ regarding $W$.

$$\frac{\partial E}{\partial w_{k,i}} = \frac{\partial E}{\partial h_i} . \frac{\partial h_i}{\partial w_{k,i}}$$

However,$h_i = \sum_{k=1} V x_k w_{k,i}$

Therefore, the final update equation becomes: And the update equation becomes:

$$v^{(new)}_{wI} = v^{(old)}_{wI} - \eta . EH^T$$

where $EH$ is an $N$ dimension vector : $EH_i = \sum_{j=1}^{V} EI_j . w'_{i,j}$

# 2 NEGATIVE SAMPLING

The procedure described above while being robust, remains costly. Indeed, the backpropagation requires to iterate through all the words in the vocabulary at every step. Doing so many iterations for all the worlds at every training step makes the training very slow, making skipgram impractical for handling large datasets. To solve this issue, an idea is to update only a small sample of outputs vectors for each training step.

The main task is to choose the vectors that have to be updated. For this, two subsamples are defined. The first one, called the positive sample, is the ground truth or the output word (the embedding of the input word). the other one, called negative sample is a subset of few randomly chosen words.

## 2.1 Selecting the negative samples:

For $k$ negative samples, the generated dataset $D$ (i.e the vocabulary) is $k$ times larger than $D$, and for each $(w, c)$ in $D$ we construct $k$ samples $(w, c_1), ..., (w, c_k)$, where each $c_j$ is drawn according to its unigram distribution raised to the 3/4 power.

Indeed, the probability to randomly pick a word from a dataset is the count of the word's occurrence divided by the total number of words in the dataset.

$$P(w_i) = \frac{f(w_i)}{\sum_{j=0}^{n} f(w_j)}$$

where $P(w_i)$ is the probability to draw the word $w_i$ and $f$ is the functions that counts the number of occurrences of a given word in the dataset.

Mikolov et.al. state in [2] that they tried a number of variations on this equation, and the one which performed best was to raise the word counts to the 3/4 power. Therefore, the probability distribution becomes:

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} f(w_j)^{3/4}}$$

This distribution has the tendency to increase the probability for less frequent words and decrease the probability for more frequent words.

## 2.2 Finding the loss function

In [2], Mikolov et. al. introduce an optimization trick for the objective function which is that they instead aim to maximize the probabilities that all of the observations indeed came from the data:

$$\max_{\theta} \prod_{(w,c)\in D} p(D = 1 \mid c, w; \theta)$$

$$\Leftrightarrow \max_{\theta} \sum_{(w,c)\in D} \log \frac{1}{1 + e^{-v_c \cdot v_w}}$$

In this setting, a trivial solution can be obtained and leads to a non-injective representation (two identical vectors for two distinct words). Set $\theta$ such that $p(D = 1|w, c; \theta) = 1$ for every pair $(w, c)$. This can be easily achieved by setting $\theta$ such that $v_c = v_w$ and $v_c v_w = K$ for all $v_c, v_w$, where $K$ is a large enough number (practically, we get a probability of 1 as soon as $K \approx 40$). Afterwards, a generated dataset $D'$ made of incorrect pairs is introduced to "balance" the probabilities. This results in the negative sampling loss:

$$max \prod_{(w,c)\in D} p(D = 0 \mid c, w; \theta) \Leftrightarrow max \prod_{(w,c)\in D} p(D = 1 \mid c, w; \theta)$$

The loss function is therefore:

$$- \sum_{(w,c)\in D} \log \sigma(-v_c \cdot v_w) - \sum_{(w,c)\in D'} \log \sigma(v_c \cdot v_w)$$

## 2.3 Backpropagation

The first step is to calculate the derivative of the loss regarding the output:

$$\frac{\partial E}{\partial v'^{T}_{w_j}.h} = \sigma(v'^{T}_{w_j \ldots h}) - t_j$$

* $v'_{w_j}$ is the output vector of the word $w_j$
* h is the output value of the hidden layer
* $t_j$ is the label of the word $w_j$: t=1 if $w_j$ is the positive sample, and $t = 0$ otherwise

Practically, this is just the error on the algorithm's output.

### 2.3.1 *Hidden → output weights*.

The next step is the derivative of the loss regarding the output vector of the word $w_j$

$$\frac{\partial E}{\partial v'_{w_j}} = \frac{\partial E}{\partial v'^{T}_{w_j}.h} * \frac{\partial v'^{T}_{w_j}.h}{\partial v'_{w_j}}$$

Therefore the update equation for the output vector is:

$$v'^{(new)}_{w_j} = v'^{(old)}_{w_j} - \eta \frac{\partial E}{\partial v'_{w_j}}$$

Compared to the update equation given on the simple skipgram version, this update equation is much computationally efficient as it only has to be applied on the set of negative samples and the positive sample.

### 2.3.2 *Input → Hidden weights*.

To backpropagate the error to the hidden layer and thus update the input vectors of words, we need to take the derivative of $E$ with regard to the hidden layer's output.

$$\begin{aligned}\frac{\partial E}{\partial h} &= \sum_{\mathcal{W}} \frac{\partial E}{\partial v'^{T}_{w_j}.h} * \frac{\partial v'^{T}_{w_j}.h}{\partial h} \\ &= \sum_{\mathcal{W}} \sigma(v'^{T}_{w_j \ldots h}) v'_{w_j} \qquad = EH\end{aligned}$$

Where $\mathcal{W}$ is the set of words in the negative and positive samples.

Finally we use the same w update equation as for the classic skipgram architecture.

$$v^{(new)}_{wI} = v^{(old)}_{wI} - \eta . EH^T$$

# 3 EFFECT OF SUBSAMPLING AND RARE-WORD PRUNING

## 3.1 Rare-word pruning

Trying to learn contexts of rare words in the corpus is problematic for SkipGram, as there is not enough samples to train properly. And because there are less contexts to learn, this boosts the overall training speed. We retain words with at least 5 occurrences.

## 3.2 Effect of subsampling

Implemented as is, the frequent words such as 'the', 'is' undermines the ability of SkipGram to learn great embeddings. To tackle this issue, frequent words are down-sampled. The motivation behind this variation is that frequently occurring words hold less discriminative power. The underlying motivation is the effect of increasing both the effective window size and its quality for certain words. According to Mikolov et al. (2013), sub-sampling of frequent words improves the quality of the resulting embedding on some benchmarks.

For a given threshold $t$, we remove a word with frequency $f$ with probability:

$$p = \frac{f - t}{f} - \sqrt{\frac{t}{f}}$$

Implementation note: Importantly, these words are removed from the text before generating the contexts.

# 4 USING PRINCIPAL COMPONENTS ANALYSIS TO IMPROVE OUR RESULTS

In [1], Raunak states that here has been recently an emphasis on further improving the pre-trained word vectors through post-processing algorithms. One such area of improvement is the dimensionality reduction of the word embeddings. Reducing the size of word embeddings through dimensionality reduction can improve their utility in memory constrained devices, benefiting several real-world applications. In his work, he presented a novel algorithm that effectively combines PCA

based dimensionality reduction with a recently proposed post-processing algorithm, to construct word embeddings of lower dimensions.Empirical evaluations on 12 standard word similarity benchmarks showed that his algorithm reduces the embedding dimensionality by 50%, while achieving similar or (more often) better performance than the higher dimension embeddings.

We proposed to replicate these results and implement the PCA on the trained vector outputs of our Skipgram model.

## 4.1 A brief explanation of PCA

PCA is an unsupervised method, which aim is to find a low dimensional space such that information is maximized (i.e variance) when data is projected on that space. In fact, the aim is to project the data on a space where the data's variance is maximized, i.e a space that discriminates well between the different instances and where differences are visible and explainable as shown in the figure below.

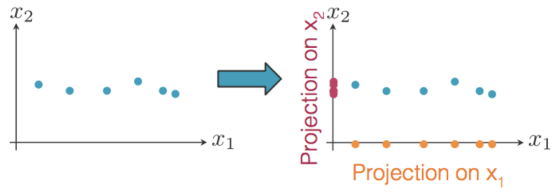


***Figure 2: Data projection on new spaces***

Given a dataset $\mathcal{D} = \{x^1, x^2, ...x^m\}$, the variance and the mean are defined by:

$$\mu_j = \frac{1}{m}\sum_{i=1}^{j} x_j^i$$

$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^i - \mu_j)^2$$

features that take large values will have a large variance, therefore, PCA requires to first standardize the features: *__Mean centering__: give each feature a mean of 0 *__Variance Scaling__: give each feature a variance of 1

$$x_j^i \leftarrow \frac{x_j^i - \mu_j}{\sigma_j}$$

Let's consider a matrix $X$, whose components are $x_i^j$. The aim is to find a low dimensional space such that the variance is maximized when $X$ is projected on that space. Therefore the projection of $X$ on the wanted new space can be written as: $Z_{m,1} = X_{m,n}.w_{n,1}$

Next, the variance of $Z$ is computed as a function of $w$ and $X$:

$$\begin{aligned} Var(Z) &= Var(Xw) \\ &= Var(w^T.X^T \\ &= E\Big[((w^T.X^T) - E\big[(w^T.X^T)\big])^2\Big] \\ &= E\Big[((w^T.X^T) - w^T E[X])^2\Big] \\ &= E\Big[w^T X^T X w\Big] \\ &= w^T E\Big[X^T X\Big] w \end{aligned}$$

The goal is to find $w_1 = \underset{w \in R^n}{argmax} Var(Xw)$ s.t ||w||=1

#### 4.1.1 ***PCA Algorithm.***

Input: data matrix $X$, $k$ principal axes * Suppose that the data is organized as an $m * n$ matrix
* Subtract mean values from columns: $C = X - M$
* Compute the covariance Matrix: $\sum = C^T C$
* **Principal Axes:** the $k$ eigenvectors $U[1, ..., k]$ of $\sum$ that correspond to the $k$ largest eigenvalues
* **Principal Components:** project the data to the new space C U[1,...,k]

The total variance explained in the data is $Trace(\sum) = \lambda_1 + \lambda_2 + ... + \lambda_n$ where $\lambda_i$ are the eigenvalues of $\sum$.

Finally the $k$ principal components account for $\frac{\lambda_1+\lambda_2+...+\Lambda_k}{\lambda_1+\lambda_2+...+\lambda_m}$ of the total variance explained.

## 5 EXPERIMENTS AND RESULTS

Beyond playing with the most similar words of a given word, our experiments were in large part related to integrating additional techniques to SkipGram and Negative sampling (i.e. subsampling, PCA, and dynamic window size). As to measure the effect of such variations, we used SimLex 999 dataset. In this dataset, for every pair of words, a similarity measure between 0 and 10 is given. This correspond to a ground truth labels. We use our word embeddings to compute similarities and confront them to the ground truth using the Mean Absolute Error. For different configurations, we obtain:

where NS stands for Negative sampling, SS stands for Sub sampling and DW is dynamic window. We notice in these plots, that for the task of computing similarities, w2v performance is enhanced by adding Sub sampling and PCA. In this particular setting, we set the number of components for our embedding to 75 (out of 100).

## REFERENCES


[1] Vikas Raunak. Simple and effective dimensionality reduction forword embeddings. In *NIPS*, 2017.
[2] Greg Corrado Tomas Mikolov, Kai Chen and Jeffrey Dean. Distributed representations of words and phrasesand their compositionality. In *NIPS*, 2013.

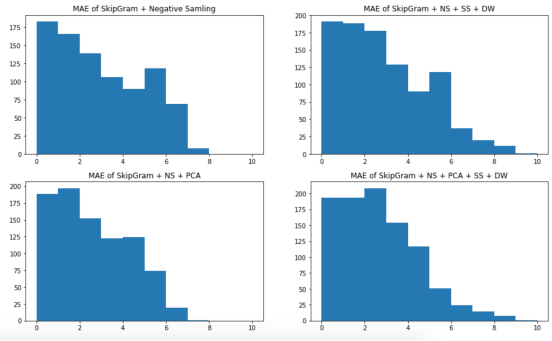


***Figure 3: MAE of various word2vec for SimLex 999***

[3] Greg Corrado Tomas Mikolov, Kai Chen and Jeffrey Dean. Efficient estimation of word representations in vector space. In *ICLR Workshop*, 2013.