Contents

1 Research Questions

This paper will address the following questions:

1. Can the throughput of the skip Gram Model be optimized by the use of advanced optimizers, while at the same time maintaining it's accuracy?
2. Can the throughput of the skip Gram Model be optimized by the use of input shuffling, while at the same time maintaining it's accuracy?
3. Can those, previously mentioned optimization ideas be combined?

2 Background

2.1 The Skip-Gram Model

The Skip Gram Model is a model used to embed words into vectors, by analyzing the context in which the words happen. It's acheiving this by maximizing the following equation:

$$\prod_{t=1}^{T} \prod_{-m<j<m} p(w_{t+j}|w_t) \tag{2.1}$$

Where T is the number of words in the dataset, and m is the context window. This means that the m nearest words to w are considered as context words. This equation can be transformed quite easily into sums by using log probabilities:

$$\sum_{t=1}^{T} \sum_{-m<j<m} p(w_{t+j}|w_t) \tag{2.2}$$

where the parameters are the same as in 2.1. The basic Skip-Gram Model uses a classical softmax to calculate the conditional probability $p(w_{t+j}|w_t)$:

$$p(w_{t+j}|w_t) = \frac{exp(w_{t+j}^T w_t)}{\sum_{w=1}^{v} exp(v_w^T w_t)} \tag{2.3}$$

There lies a problem in this approach. As a matter of fact it is not feasible to compute the softmax. For the computation of $\sum_{w=1}^{v} exp(v_w^T w_t)$ one has to go over the complete dataset. As very big data sets are needed to train the model, this is not a solution. But different solutions were proposed by [**mikolov2**]. The first one is to use a Hierarchical soft max introduced by [**hsoftmax**]. The probability distribution is saved in a binary tree which gives one a logarithmic computation time, and which makes it feasable to compute the softmax. Another possibility is the use of negative sampling which is used in the original word2vec implementation [**mikolov2**], which I shall discuss in the next section.

2.2 Negative Sampling

The idea behind negative sampling is to only update the output nodes of certain words. This will obviously save an enormous amount of computation time. The idea is that given a pair (w, c) we will set $p(x|c) = 1$ select K random words k_i from the dataset and set $p(x|k_i) = 0$. We will denote the probability that the (w, c) exists in the data set the following way: $p(y = 1|w, c)$, and if (w, k) is chosen at random this way: $p(y = 0|x, k)$. Know we will use logistic regression to update the weights of the k selected context words and c. By doing so we will only have to update $k + 1$ output nodes.

Let's look at how we construct our objective function for a given word w and one of its context words c:

$$\begin{aligned}
p(x|c) &= p(y=1|w,c) + \prod_{k \in K} p(y=0|k,c) \\
&= p(y=1|w,c) + \prod_{k \in K} 1 - p(y=1|k,c) && \text{Use of log probabilities)} \\
&= log((p(y=1|w,c)) + \sum_{k \in K} log(1 - p(y=1|k,c)) && \text{Use of softmax} \\
&= log(\frac{1}{1+e^{-v_c v_x}}) + \sum_{k \in K} log(1 - \frac{1}{1+e^{-v_c v_k}}) \\
&= log(\frac{1}{1+e^{-v_c v_x}}) + \sum_{k \in K} log(\frac{1}{1+e^{v_c v_k}}) \\
&= log(\sigma(v_c v_x) + \sum_{k \in K} \sigma(log(-v_c v_k)) && where, \sigma = \frac{1}{1+e^{-x}}
\end{aligned}$$

We see that to compute our objective function we will only have to compute the sum over K. Which in practice is very small (2-20). Too put things in perspective lets imagine our data set consits of 10000 words, we set $K = 5$ and let's say that each output neuron has weight vector v with $|v| = 300$. When updating our wheights we would only update 0.06% of the 3 million weights in the output layer.

One question remains: how do we choose our random words? [**mikolov2**] used the following random distribution:

$$P(w) = \frac{f(w)^{\frac{3}{4}}}{\sum_{w_k \in W} f(w_k)^{\frac{3}{4}}} \tag{2.4}$$

where $f(w)$ is the frequency of w in the Vocabulary W. The value of $\frac{3}{4}$ is set empirically.

It's quite easily observable that this approach will outperform the classical softmax in computation time. Now the question arises if the accuracy is good enough enough but according to [**mikolov2**] the negative sampling method "is an extremely simple training method that learns accurate representations". We know have enough background knowledge about word2vec and the skip gram model to look at how it can be optimized. In the next section we are going to cover what has already be done.

2.3 Optimization of the Skip Gram Model

Due to the popularity of the skip gram model, a lot of research went into optimizing it. This research can actually be divided into three categories, parallelization, restating the problem as matrix factorization, optimize the accuracy of the alogirithm by aloying words to have multiple meanings. In this section we are gonna cover all of those and try to compare them, inasmuch as it is possible.

2.3.1 Parallelization

In the original model the optimization is done with Stochastic Gradient Descent (SGD), which in it's most basic form is sequential. This does not favor a paralization process. To deal with this problem [**mikolov2**] used a Hogwild tree. The approach is to allow multiple threads to acces a shared memory, in this case the single model. This can obviously lead to overwriting errors. But according to [**hogwild**] this doesn't lead to a significant accuracy loss if the data access isn't too frequent. But this maybe inaccurate for NLP, and especially for word embedding, as many words share the same context words. There were several attempts at solving this issue, and we are going to cover a few of them in the following subsections.

Parallization in shared and Distributed Memory

The first parallization solution which was proposed by [**intel**], is to try to reduce the cost of our vector multiplication. The main idea in this paper is to convert the level 1-BLAS vector to vector operations to a level-3 BLAS matrix multiplication operation. This is achieved, buy using the same negative samples for each context word of a given word w. Instead of using at each context word a vector to vector multiplication we can transform this, under the assumption that we will not loose accuracy by sharing the same negative samples, into a matrix multiplication. The matrix multiplication can be represented the following way.

$$\begin{bmatrix} w \\ w_{n_1} \\ \vdots \\ w_{n_k} \end{bmatrix} * \begin{bmatrix} w_{c_1} \\ \vdots \\ w_{c_{2m}} \end{bmatrix}$$

where w is our given word, $w_{n_1}...w_{n_k}$ are the shared negative samples, with $k \in [5, 20]$, and $w_{c_1}...w_{c_{2m}}$ are the words inside of the context window m of w, with $m \in [10, 20]$. [**intel**] called $w_{c_1}...w_{c_{2m}}$ a batch of input context words. After each batch the model updates the weights of the used vectors. This model achieves a 3.6 fold increase in throughput, by only losing 1% of accuracy. This model though is only applicable on architectures that are capable of such level-3 BLAS operations.

Parallelization by the use of caching

This idea was proposed by [**efficient**]. The architecture used here is the basic skip gram model with an hierarchical soft max. The general idea is to cache the most frequent used nodes of the binary tree used to memorize the probability distribution, and update them on the shared single model after a certain amounts of seen words (the paper used the number 10). The paper produced interesting results as they managed to increase efficiency by increasing the number of cores used for the calculation. This is very powerful because the original implementation regressed in it's accuracy after 8 cores, this seems to indicate that too much overwriting is happening, as the number of concurrent threads surpasses a certain threshold. This can be seen in 2.1, where c31 is the model proposed by [**efficient**]

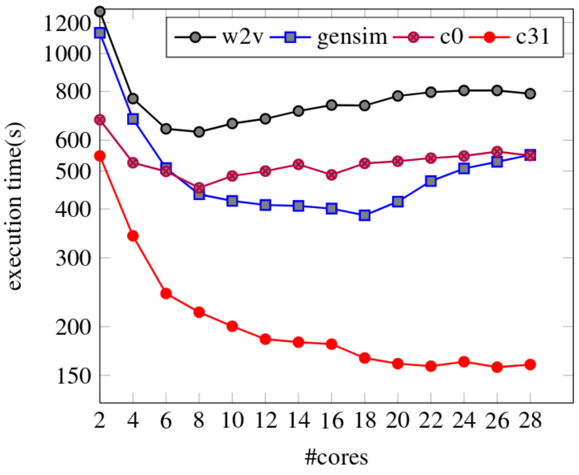


Figure 2.1: Comparasion of the execution time in relation with the number of used cores [**efeficient**]

2.3.2 Restating the problem as Matrix factorization

It was also shown in [**goldenberger**] that the optimization problem solved by the sgd can alos be restated as a matrix factorization. In fact the partial derivative Matrix W*C where w is the word embedding matrix and C the context matrix, is nothing else then PPMI - log k. which states the optimizing problem as a pure matrix factorization problem. Also this first step showed similar results to the original word2vec. Then this was picked up by, to do it with riemman optimization and outperformed the original word2vec in accuracy.

2.3.3 Context sensitive word embedding

A word does not always have the same meaning according to it's context. This is a problem that is not addressed by word2vec and the general skipGram model. Some new models, that would take this issue into consideration, were proposed. A lot of work has be done in this direction, [**topicalWE**], [**breaking**] but the the one reporting the best results is [**contextWithTensor**]. The main idea is to change the way and variables in our conditional probability. The idea is to look if a word given a certain context word matches to a topic. Bank would match too Economic given the context word Money. Bank would also match too Nature if river was the given context word.

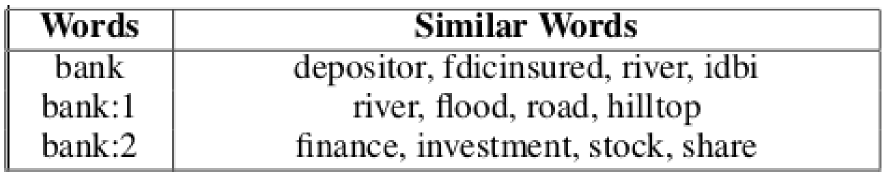
But bank would not match too Nature with the context word money. As said before [**contextWithTensor**] had the best results, let's look at how they achieved it: First let's take a look at the objective function:

$$J(\Omega) = \sum_{(w,t,c)\in D} \sum_{(w,\tilde{t},\tilde{c}\in\tilde{D})} max(0, 1 - g(w,t,c) + g(w,\tilde{t},\tilde{c}))\lambda||\Omega||_2^2 \tag{2.5}$$

This approach uses the same negative sample technique as described int the previous sections, D is the corpus data and $\tilde{D}$ is the set of negative samples. What is interesting her is the function g, which is defined as follows:

$$g(w,c,t) = u^T\sigma(w^T M^{[1:k]}t + V_c^T(w \oplus t) + b_c) \tag{2.6}$$

where, u, V_c, b_c are standard parameters for a neural network. $\oplus$ is the concatenation, $\lambda$is the hyperparameter used for the standard L_2 and the most important parameter is $M^{[1:k]}$, which a tensor layer, the tensor layer is used because of its ability to model multiple interactions in the data, as this will be useful for multiple contexts. They used SGD for the optimization of this objective function. They acheived really interesting results as shown in 2.2.



Words	Similar Words
bank	depositor, fdicinsured, river, idbi
bank:1	river, flood, road, hilltop
bank:2	finance, investment, stock, share

Figure 2.2: "Nearest neighbor words by our model and Skip- Gram. The first line in each block is the results of Skip-Gram; and the rest lines are the results of our model" [**contextWithTensor**]

3 Project Plan

This section will cover the main project plan, we will discuss exactly what we wish to implement and how we are going to test our implementation. The whole process will be done the following way:

- `Phase 1: Research`
  In this phase we will become a broad overlook on the subject, research possible libraries that we can use and start planing the following phases.
- `Phase 2: Implementation`
  This phase will be our main work, as we will implement our own word2vec version.
- `Phase 3: Testing`
  Here we will first test if our optimization ideas were succesfull, and if they were we will test the accuracy of our Model.
- `Phase 4: Writing`
  In this phase we will summarize Phase 1-3 in our thesis.

More details on phase 2 and 3:
`Phase2` First we will implement our own version of the skip gram model. We will implement the optimization techniques stated in 1, this means we will use input shuffling and advanced optimizers. There exists a python implementation of the original word2vec mode, that is called Gensim [**gensim**]maybe it's possible to tweak it to fit our needs, if not we are going to implement our own version. `Phase3` First we compare our model against the original gensim word2vec implementation. For this we wil use the dataset *text8*, that was created by Matt Mahoney [1]. We will first compare the throughput, this means the number of words our model is able to analyze per second. If we see promising results we will then test the accuracy of our model. This is quite difficult as the quality

[1]mattmahoney.net

of word embeddings are often task dependent, but [**mikolov2**] presented a word analogy task and [**wSimilarity**] presented a word similarity evaluation. We will test our model on both on these task if we achieve a significant optimization.