Natural Language Processing 1 - Neural Word Embedding

Arthur Bražinskas
MSc Artificial Intelligence
University of Amsterdam
arthuras.brazinskas@student.uva.nl

Bryan Eikema[1]
MSc Artificial Intelligence
University of Amsterdam
bryan.eikema@student.uva.nl

Sander Lijbrink[1]
MSc Artificial Intelligence
University of Amsterdam
sander.lijbrink@student.uva.nl

Minh Ngo
MSc Artificial Intelligence
University of Amsterdam
minh.ngole@student.uva.nl

Abstract

In this report we present our study on neural network models for word vector representations learning. We considered three types of models: CBOW, Skip-gram, and RNN, and compared them against each other on similarity tasks. In addition, we performed individual experiments for each model to find the best configuration with softmax layer optimisations and learning rate adaptation methods.

1 Introduction

Word vector representations have shown to carry useful semantic and syntactic information [Andreas and Klein, 2014], and they can be used to significantly improve and simplify many natural language processing applications like POS-tagging and name entities annotation [Collobert et al., 2011]. The key reasons is in an integrated feature extraction process to the learning pipeline, which releases developers from manual feature extraction duty.

In this report we describe our experiments with three different models for word embedding (representations) learning: continues bag of words, Skip-gram, and recursive neural network. The goal of our project was to compare those models against each other, and for each model find the best configuration. Configurations mainly consist of softmax layer optimization (negative sampling, hierarchical softmax, output layer factorization), activation function preference (sigmoid, ReLU) and learning rate adaptation algorithms (e.g. adaGrad, adaDelta).

In the first experiment models were trained on the same machine, such that it would be possible to compare their run-time, and the model configurations experiments on different machines for every model without attempt to do a global comparison.

The document is structured as follows: we present the background information for a reader to get an insight into the models and optimisations, then we will present our experimental setup and the results, afterwards we will discuss some open issues and interesting aspects of the project, and finally wrap everything with conclusions.

[1]These authors have decided to continue the course next year

2 Background

In this section we describe our models, and present some underlying mathematical derivations. We start by presenting all three models, then softmax optimisations, and end by learning rate adaptation methods.

2.1 Continues bag of words

Continues bag of words (CBOW) is a neural network alike model that permits efficient learning of word-vector representations. Its advantage over Skip-gram is in speed, the model can be three times faster as is shown in [Mikolov et al., 2013b].

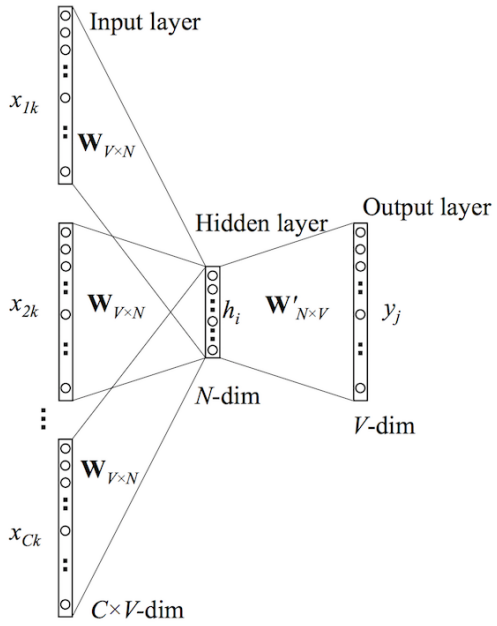


Figure 1: CBOW architecture

The architecture of CBOW is shown in fig 1. Input layers are vocabulary size one-hot vectors where a component with the value of 1 corresponds to the actual context word. The model has a shared weights input matrix **W** that connects input and hidden layers. The hidden layer has a linear activation function which is different from traditional neural networks that have sigmoid, rectified linear, or hyperbolic tangent activation functions. The output layer has softmax activation functions, and therefore the output of the layer is a valid probability distribution vector, which encodes probability of each word in a vocabulary being in a center of a context.

We compute probability of a word j to be in a context of words encoded in a vector **h** via softmax as show in eq. 1. The hidden layer's output vector is obtained via eq. 2. Notice that we average input projections to the hidden layer.

$$p(w_j|\boldsymbol{h}) = y_j = \frac{\exp(\boldsymbol{v}'^{\boldsymbol{T}}_{\boldsymbol{w_j}}\boldsymbol{h})}{\sum_{j'}^{V}\exp(\boldsymbol{v}'^{\boldsymbol{T}}_{\boldsymbol{w_{j'}}}\boldsymbol{h})} \tag{1}$$

$$\boldsymbol{h} = \frac{1}{C}\boldsymbol{W}(\boldsymbol{w}_1 + \cdots + \boldsymbol{w}_C) = \frac{1}{C}(\boldsymbol{v}_{w_1} + \cdots + \boldsymbol{v}_{w_C}) \tag{2}$$

where C is the number of context words, $\boldsymbol{v}'_{w_i}$ is an output representation of a word w_i, i.e. it's the column i in the output matrix $\boldsymbol{W}'$, and $\boldsymbol{v}_{w_i}$ is an input representation of a word w_i i.e. the row i of the input matrix $\mathbf{W}$.

And we are interested in minimizing the following negative log-likelihood objective function for each word show in eq. 3, partial derivatives of which are used in stochastic gradient descent (SGD) optimization.

$$E_j = -\log p(w_j|\boldsymbol{h}) \tag{3}$$

To learn input and output representations of words we take partial derivatives of the equation 3, and finally we use SGD with update eq. 4 and 5.

$$\boldsymbol{v}'_{w_j} = \boldsymbol{v}'_{w_j} - \eta\, e_j\, \boldsymbol{h} \qquad for \quad j = 1, 2, \ldots, V \tag{4}$$

$$\boldsymbol{v}_{w_i} = \boldsymbol{v}_{w_i} - \eta\, \mathrm{EH} \qquad for \quad i = 1, \ldots, C \tag{5}$$

$$\mathrm{EH} = \sum_{j=1}^{V} e_j \boldsymbol{v}'_{w_j} \tag{6}$$

$$e_j = y_j - t_j \tag{7}$$

where $t_j = (j = j^*)$, i.e., t_j will only be 1 when the j-th node is the actual output word, otherwise t_j = 0.

2.2 Skip-gram

The Skip-gram model is another model introduced in Mikolov et al. [2013b] and elaborated upon in Mikolov et al. [2013c]. The Skip-gram model does the opposite of the CBOW model. Whereas the CBOW model predicts the center word from a set of context words, the Skip-gram model predicts the context words from a single center word. This is reflected in the architecture of the Skip-gram model as shown in figure 2 below. Despite the fact that Skip-gram is slower in practice, it is better for infrequent words, and as has been shown in [Mikolov et al., 2013b] it produces more accurate results.

There are, again, two weight matrices $\boldsymbol{W}$ and $\boldsymbol{W'}$. Note that there is only a single hidden to output matrix $\boldsymbol{W'}$, and thus the model will not actually be able to predict multiple different context words. The architecture and objective function (as will be shown later), however, do reflect this goal. Our actual objective is to find good word representations, and thus we do not have a need for multiple hidden to output matrices. It is shown in Mikolov et al. [2013b] that this does lead to good word representations.

The objective function for the Skip-gram model is similar to the one for the CBOW model and is shown in eq. 8, 9 and 10.

$$E = -\log p(w_{O,1}, \ldots, w_{O,C}|w_I) \tag{8}$$

$$= -\log \prod_{c=1}^{C} \frac{\exp(\boldsymbol{v}'^{\boldsymbol{T}}_{\boldsymbol{w}_{\boldsymbol{j}^*_{\boldsymbol{c}}}} \boldsymbol{h})}{\sum_{j'=1}^{V} \exp(\boldsymbol{v}'^{\boldsymbol{T}}_{\boldsymbol{w}_{\boldsymbol{j}'}} \boldsymbol{h})} \tag{9}$$

$$= -\sum_{c=1}^{C} \exp(\boldsymbol{v}'^{\boldsymbol{T}}_{\boldsymbol{w}_{\boldsymbol{j}^*_{\boldsymbol{c}}}} \boldsymbol{h}) + C \log \sum_{j'=1}^{V} \exp(\boldsymbol{v}'^{\boldsymbol{T}}_{\boldsymbol{w}_{\boldsymbol{j}'}} \boldsymbol{h}) \tag{10}$$

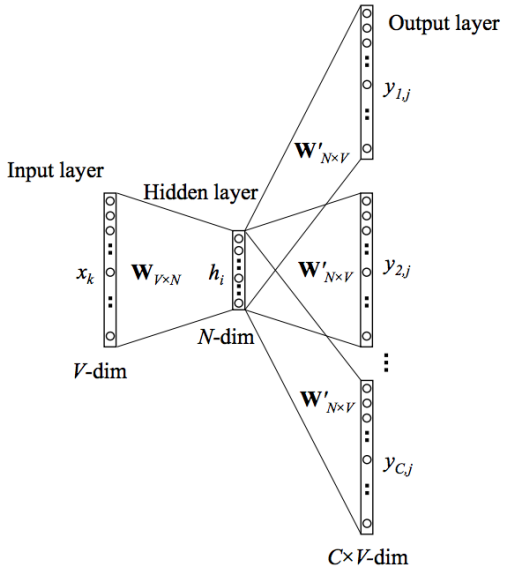


Figure 2: The Skip-gram model

The notation here is borrowed from Rong [2014]. Here j^* is the correct c^{th} context word, C is the number of context words and V is the vocabulary size. For Skip-gram $\boldsymbol{h}$ is simply the input representation of the center word, i.e., the j^{th} row of $\boldsymbol{W}$. v'_{w_j} is still the output representation for word j, i.e., the j^{th} column of $\boldsymbol{W'}$. From this objective function the update equations for SGD can easily be derived, they are shown below.

$$\boldsymbol{v}'^{(new)}_{w_j} = \boldsymbol{v}'^{(old)}_{w_j} - \eta\, EI_j\, \boldsymbol{h} \qquad for \quad j = 1, 2, \ldots, V \tag{11}$$

$$EI_j = \sum_{c=1}^{C} e_{c,j} \tag{12}$$

$$e_{c,j} = y_j - t_{c,j} \tag{13}$$

$$y_j = \frac{\exp(\boldsymbol{v}'^{\boldsymbol{T}}_{\boldsymbol{w_j}} \boldsymbol{h})}{\sum_{j'=1}^{V} \exp(\boldsymbol{v}'^{\boldsymbol{T}}_{\boldsymbol{w_{j'}}} \boldsymbol{h})} \tag{14}$$

$$\boldsymbol{v}^{(new)}_{w_I} = \boldsymbol{v}^{(old)}_{w_i} - \eta\, \mathrm{EH} \tag{15}$$

$$EH = \sum_{j=1}^{V} EI_j \boldsymbol{v}'_{w_j} \qquad for \quad i = 1, \ldots, N \tag{16}$$

In these equations η is the learning rate and $t_{c,j}$ is the target value. Thus $t_{c,j}$ is 1 if the j^{th} word is the expected c^{th} context word and 0 otherwise.

2.3 Recurrent Neural Network

In the Recurrent Neural Network Language Models (RNN LM) the next word is predicted, given a particular word in the sentence. In addition, the existence of the recurrent connection for the hidden layer helps it to capture long relationships.

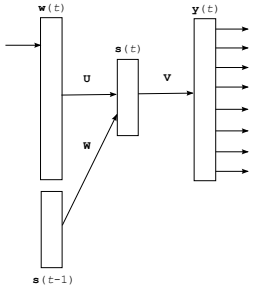


Figure 3: The standard RNN LM

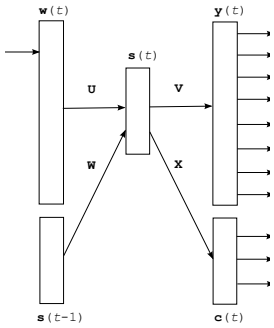


Figure 4: The RNN LM with factorization of the output layer

The standard baseline model has been introduced by Mikolov et al. [2010b]. In this model there is one single hidden layer $\mathbf{s(t)}$ of size $\mathbf{H}$ with a recurrent connection between its previous state and the current state [Fig 3]. An input vector $\mathbf{w(t)}$ is represented as a one-hot vector of the size $\mathbf{N}$ of the vocabulary. As an output one-hot vector $\mathbf{y(t)}$ of the same size that represents the next word of the sequence is expected. **U**, **W**, **V** are weight matrices of size $[H \times N]$, $[H \times H]$, $[N \times H]$ respectively. For the hidden layer, the sigmoid activation function is used, for the output layer the softmax function is used.

Forward propagation is computed as following:

$$\begin{aligned} \mathbf{y}(t) &= softmax(\mathbf{V}\mathbf{s}(t)) \\ \mathbf{s}(t) &= \sigma(\mathbf{U}\mathbf{w}(t) + \mathbf{W}\mathbf{s}(t-1)) \end{aligned} \tag{17}$$

The cross entropy error function is used for training:

$$\begin{aligned} E &= -\sum_i^N t_i \log(y_i) = \sum_i^N t_i \log \frac{\exp(\mathbf{V}_i^T \mathbf{s}(t))}{\sum_j^N \exp(\mathbf{V}_j^T \mathbf{s}(t))} \\ &= -\sum_i^N t_i \left({\mathbf{V_i}}^T \mathbf{s}(t) - \log \sum_j^N \exp(\mathbf{V}_j^T \mathbf{s}(t)) \right) = -(q_k - \log Z) \end{aligned} \tag{18}$$

where y_i, t_i are an output and a correct label for the i-th element of the output vector, k is an index where $t_k = 1$, $q_i = \mathbf{V}_i^T \mathbf{s}(t)$ is an element of the vector $\mathbf{q}$ of size N, $Z = \sum_j^N \exp(\mathbf{V}_j^T \mathbf{s}(t))$. Later, derivatives for the gradient descent procedure can be easily computed:

$$\frac{\partial E}{\partial q_i} = \begin{cases} -(1 - y_i(t)), & \text{if } i = k \\ y_i(t), & \text{if } i \neq k \end{cases} \Leftrightarrow \frac{\partial E}{\partial \mathbf{q}} = -(\mathbf{t} - \mathbf{y}(t)) = -\mathbf{e_o} \tag{19}$$

$$\frac{\partial E}{\partial V} = \frac{\partial E}{\partial \mathbf{q}} \frac{\partial \mathbf{q}}{\partial \mathbf{V}} = -\mathbf{e_o} \mathbf{s}^T(t) \tag{20}$$

$$\frac{\partial E}{\partial U} = \frac{\partial E}{\partial \mathbf{q}} \frac{\partial \mathbf{q}}{\partial \mathbf{s}(t)} \frac{\partial \mathbf{s}(t)}{\partial \mathbf{U}} = -\left[\mathbf{V}^T \mathbf{e_o} \cdot \mathbf{s}'(t) \right] \mathbf{w}^T(t) = \mathbf{e_h}(t) \mathbf{w}^T(t) \tag{21}$$

where $\cdot$ is a element-wise multiplication operator, $\mathbf{s}'(t)$ is a vector of size H with sigmoid function derivatives $s_i(t)(1 - s_i(t))$ as elements[1].

[1] For more detailed derivations `http://imgur.com/a/3JXvf`

A derivative $\frac{\partial E}{\partial W}$ is computed using back-propagation through time:

$$\begin{aligned} \mathbf{e_h}(t-1) &= \mathbf{W}^T \mathbf{e_h}(t) \cdot \mathbf{s}'(t-1) \\ \frac{\partial E}{\partial W} &= -\sum_{i=1}^{t-1} \mathbf{s}(i) {\mathbf{e_h}}^T(i+1) \end{aligned} \tag{22}$$

Error gradients quickly vanish as they get propagated in time, therefore several steps of unfolding are sufficient. In our experiments i is chosen from (t-3) to (t-1) [Mikolov et al., 2010a].

Two types of word vector representations in this model can be considered:

- **Inner word representation** of the i-th word from the vocabulary is a column vector $\mathbf{U}_i$ [Mikolov et al., 2013d, Rong, 2014]
- **Outer word representation** of the i-th word from the vocabulary is a row vector $\mathbf{V}_i$ [Rong, 2014]

In the case a model doesn't have a transition matrix from the hidden layer to output layer (like in Hierarchical softmax), only Inner word representation can be counted.

Bengio et al. [1994] have pointed out about several difficulties with training Recurrent Neural Networks:

- the **gradient exploding** problem refers to the large increase in the norm of the gradient during training;
- the **vanishing gradients** problem refers to the opposite behavior, when long term components go exponentially fast to norm 0;
- in addition, the baseline model is **slow**, mostly because of the softmax layer computation that is proportional to the size of vocabulary [Elsen, 2015].

In our work we use **gradient clipping** (GC) [Pascanu et al., 2012] to solve the gradient exploding problem by scaling a gradient in the case if its norm is greater than a threshold:

$$grad = \frac{threshold}{|grad|} \, grad \tag{23}$$

For solving the gradient vanishing problem we use **ReLU activation functions** instead of sigmoids. Le et al. [2015] have shown that with initialization of the weight matrix $\mathbf{W}$ by an identity matrix RNN with ReLU has outperformed the state-of-the-art Long-Short-Term Memory model introduced by Hochreiter and Schmidhuber [1997] with less model complexity. Derivations will stay almost the same:

$$\begin{aligned} \sigma(x) &\text{ will become } max(0, x) \\ \sigma'(x) &\text{ will become } I\{x > 0\} \end{aligned} \tag{24}$$

We have performed several experiments that focus on the performance improvement:

- **Hierarchical softmax** that can compute softmax in the logarithmic time [Section 2.4].
- **RNN with classes** that divides a vocabulary into **K** classes and tries to predict a class of the output word and an index of it in the particular class with the help of two softmax layers [Fig 4]. In this case the size of the softmax layer for predicting a word index will decrease in K times, we will have K small vocabularies instead of 1 big one [Mikolov et al., 2011, Shi et al., 2013].

  Let $\mathbf{X}$ be a weight matrix of size $K \times H$ from the hidden layer to the softmax layer for class prediction, a class of an output word can be computed as following:

$$\mathbf{c}(t) = softmax(\mathbf{X}\mathbf{s}(t)) \tag{25}$$

A cross-entropy error will become:

$$E = -\sum_{i}^{class\ size} t_i \log(y_i) + \sum_{i}^{K} l_i \log(c_i) \tag{26}$$

where l_i, c_i are an output and a correct class for the i-th element of the class output vector. Formulas for back-propagation should be changed respectively:

$$\begin{aligned} \frac{\partial E}{\partial \mathbf{c}} &= -(\mathbf{l} - \mathbf{c}(t)) = -\mathbf{e_c} \\ \frac{\partial E}{\partial \mathbf{X}} &= -\mathbf{e_c}\mathbf{s}^T(t) \\ \mathbf{e_h}(t) &= (\mathbf{V}^T\mathbf{e_o} + \mathbf{X}^T\mathbf{e_c}) \cdot \mathbf{s}'(t) \end{aligned} \tag{27}$$

2.4 Hierarchical softmax

Hierarchical softmax (HSM) [Mnih and Hinton, 2009, Morin and Bengio, 2005] is an efficient method to compute softmax in logarithmic time $O(\log(V))$. As one can notice eq. 1 is linear in the vocabulary size V and can become a bottleneck for training on massive datasets. It is common to use Huffman's tree where leafs are words of the vocabulary and the softmax eq. 1 is changed to eq. 28.

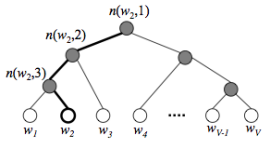


Figure 5: Hierarchical softmax binary tree illustration

$$p(w) = \prod_{j=1}^{L(w)-1} \sigma([n(w, j+1) = ch(n(w, j)]\boldsymbol{v}'^T_{n(w,j)}\boldsymbol{h}) \tag{28}$$

where $ch(n)$ is the left child of node n; $v'^T_{n(w,j)}$ is the vector representation ("output vector") of the inner node $n(w, j)$; $[x]$ is a special function that yields 1 if x is true, 0 otherwise. Notice that in the model there are no output vector representations for words.

The basic idea behind HSM is to travel from the root to a leaf corresponding to a word of interest for which probability is computed, and take a product of sigmoid outputs of each node along the path.

In practice the modification will require to derive different update equations for output and input matrices, which can be found at [Rong, 2014] for CBOW and Skip-gram. However, HSM is applicable and has been used for three models including RNN.

2.5 Negative Sampling

Negative sampling is an alternative to the HSM method that addresses softmax's linear computation cost [Mikolov et al., 2013c]. In negative sampling, instead of linear in vocabulary size denominator computation, we randomly sample a number of words from a noise distribution $p_n(w)$, and our goal is to distinguish those negative samples from the actual words of interest (positive sample). For example, the distribution can be unigram based on frequency of words.

For each training instance we have 1 or C positive samples, depending on whether we are using CBOW or Skip-gram. On top of this we draw K negative samples from our noise distribution, which excludes the target word from being selected. Then we optimize the output vector representations for these positive and negative samples to make our model yield lower probability for negative samples

and higher for positive ones. The authors of [Mikolov et al., 2013c] suggest a simplified objective function as shown in eq. 29.

$$E = -\sum_{c=1}^{C} \log \sigma(\boldsymbol{v}'^{\boldsymbol{T}}_{\boldsymbol{w_{O,c}}} \boldsymbol{h}) - \sum_{i=1}^{K} \log \sigma(-\boldsymbol{v}'^{\boldsymbol{T}}_{\boldsymbol{w_i}} \boldsymbol{h}) \tag{29}$$

where $w_i \sim p_n(w)$ for $i = 1, \ldots, K$.

Here $\boldsymbol{h}$ is defined as before for each model and the sum over C should be dropped for the CBOW model. From this objective function we can derive the update equations as shown below.

$$\boldsymbol{v}'^{(new)}_{w_j} = \boldsymbol{v}'^{(old)}_{w_j} - \eta \left(\sigma(\boldsymbol{v}'^{\boldsymbol{T}}_{\boldsymbol{w_j}} \boldsymbol{h}) - t_j\right) \boldsymbol{h} \tag{30}$$

$$EH = \sum_{j} (\sigma(\boldsymbol{v}'^{\boldsymbol{T}}_{\boldsymbol{w_j}} \boldsymbol{h}) - t_j) \, \boldsymbol{v}'_{\boldsymbol{w_j}} \tag{31}$$

where $w_j \in \{w_{O,1}, \ldots, w_{O,C}\} \cup \{w_i \sim p_n(w) | i = 1, \ldots, K\}$.

In these equations $t_j = 1$ for positive samples and $t_j = 0$ for negative samples. Also note that there is only a single output word for the CBOW model, and thus only a single positive sample per training instance. EH can be inserted in eq. 15 for Skip-gram or eq. 5 for CBOW to receive the update equations for the input vectors.

2.6 AdaGrad

AdaGrad [Duchi et al., 2011] is a method that permit adaptation of a learning rate during training phase. A new abstract update equation is presented in eq. 32.

$$x_{t+1} = x_t - \frac{\eta}{\sqrt{\sum_{j=1}^{t} g_j^2}} g_t \tag{32}$$

where η is a scalar global learning rate that has to be defined beforehand, and g_t is a gradient computed for a feature x. This method relies on only first order information and by accumulating previously computed gradients it allows frequently occurring features to get a small learning rate and infrequent features get a higher one.

2.7 AdaDelta

AdaDelta (Zeiler [2012]) is a modification of adaGrad in attempt to fix several associated problems: 1) the continual decay of learning rates throughout training, and 2) the need for a manually selected global learning rate. The main difference between adaGrad and adaDelta is that in latter we accumulate the sum of squared gradients over a window of time, which is not infinity as in adaGrad. This makes the accumulated denominator of adaGrad local, and ensures that learning continues to make progress even after many iterations of updates have been done. Finally, adaGrad has two hyper-parameters (decay rate and noise) instead of one.

3 Experiments

The following section starts discussion about the experimental setup and data, then we present our experiments with different model configurations, please note that those are not meant to be compared against each other. And finally, we present three models comparison.

3.1 Configuration and data

We use English sentences obtained from the *1 Billion Word Language Model Benchmark* [1]. This dataset consists of randomly shuffled sentences, crawled from news on the internet. As recommended by Mikolov et al. [2013c], we apply subsampling, i.e., we discard words with a probability computed with the formula:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{33}$$

where $f(w_i)$ is the frequency of word w_i in our corpus and t is a suitable threshold, in our case we used 10^{-4}. This filtering process counters the imbalance between rare and frequent words in our dataset, as it aggressively subsamples frequent words while preserving rare words.

Furthermore, we use the following steps for pre-processing:

- We have constructed the vocabulary with occurrence counts for the words and removed words from the vocabulary that occur fewer than 5 times in the total corpus.
- We process the data per sentence, as provided in the dataset, and tokenize words using the Python library `nltk` [2].
- Words are converted to lowercase and words that are not in the vocabulary are replaced with a common 'out-of-vocabulary' tag.

We take two subsets, consisting of roughly 1M and 10M words. We have experimented with 100M words but this turned out to be unfeasible. And for tuning hyper-parameters, such as learning rate, we use a separate subsample of words. Whenever was necessary to observe the log-likelihood changes over training time, a separate test subset has been used.

For the final evaluation of the models quality we extract input representations of words from trained models, and use two tests based on word similarities:

- Semantic and syntactic accuracy in reasoning tasks as provided by the GloVe project[3]. This benchmark contains many tasks such as finding the capital city given a country name, using the word vectors produced by the model.
- Spearman's rank correlation coefficient. We use the benchmark code from `www.wordvectors.org` [4] which computes the ρ coefficient between our word vector similarity and human-judged similarity values.

3.2 CBOW configurations

In table 1 we present results that were obtained with different CBOW configurations and a constant 0.09 learning rate , 20 context words (10 context window's size), and 17 negative samples for NS method.

Configuration	Dim	Words	Vocab	Sem (%)	Synt (%)	ρ	Time (h)
HSM	100	10M	118K	1.53	2.48	0.1398	**2.1**
HSM + adaGrad	100	10M	118K	1.84	1.3	0.1772	3.8
HSM + adaDelta	100	10M	118K	2.14	2.88	0.1617	5.4
NS	100	10M	118K	**4.04**	**3.93**	0.1905	4.7
NS + adaGrad	100	10M	118K	1.06	0.44	**0.2053**	6.4
NS + adaDelta	100	10M	118K	1.1	0.59	0.1953	8.6

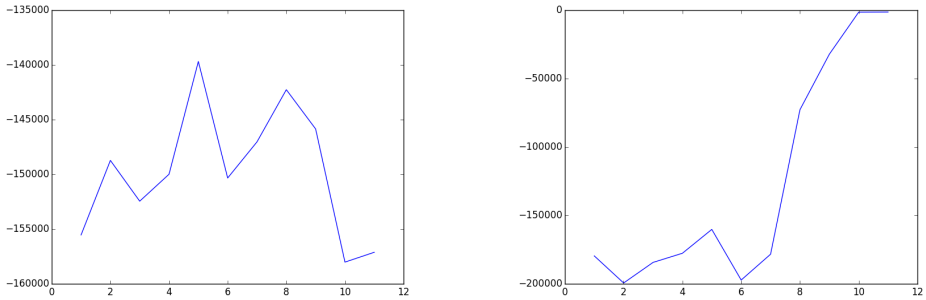
Table 1: CBOW different configuration results based on accuracy (semantic and syntactic) and Spearman coefficient ρ

[1] `http://www.statmt.org/lm-benchmark/`

[2] `http://www.nltk.org/`

[3] `https://github.com/stanfordnlp/GloVe`

[4] `https://github.com/mfaruqui/eval-word-vectors`



(a) 0.5 decay rate

(b) 1 decay rate

Figure 6: HSM+adaDelta with a constant noise (1e-4)

First of all notice that CBOW with negative sampling produced the best result accuracy-wise, while additional learning rate modifiers only decreased the accuracy result. On the other hand, Spearman coefficient did not decrease when the modifiers were used. A similar situation can be observed in case of HSM and HSM+adaGrad. However, adaGrad and adaDelta add extra computational overhead that increase the training time as we can see from the table.

Despite the fact that NS is associated with the best results, it appears to be more computationally expensive than HSM. The main reason is that the number of negative samples was set to $\log V$, where V is a vocabulary size, which appeared to be rounded to 17. But in case of HSM $\log(V)$ is the worse-case path from root to a word-leaf in Huffman's tree, and on average it will be smaller. Therefore, HSM on average will be faster, but we could always reduce the number of negative samples, but the results would also decrease.

Finally, we would like to emphasize that adaGrad and adaDelta are sensitive to initial hyper-parameters [Zeiler, 2012]. Unfortunately, with adaDelta the wrong set of hyper-parameters can lead to divergence, fig. 6 illustrates adaDelta log likelihood obtained in 1 epoch on a 1M words subset. Notice that on the left figure in the end of training we've got the log-likelihood lower that the initial one.

3.3 Skip-gram configurations

Skip-gram configuration	Words	Vocab	Sem (%)	Synt (%)	ρ	Time (h)
HSM	1M	59K	0.00	0.00	0.0383	**0.21**
HSM + AdaGrad	1M	59K	**0.01**	**0.01**	-0.0266	0.41
HSM + AdaDelta	1M	59K	0.01	0.01	0.0017	0.84
NS	1M	59K	0.00	0.00	0.0655	0.25
NS + AdaGrad	1M	59K	0.00	0.00	0.0004	0.48
NS + AdaDelta	1M	59K	0.00	0.00	**0.1117**	1.91

Table 2: Model comparison for Skip-gram with accuracy (semantic and syntactic) and Spearman coefficient ρ

The model configurations have not indicated any decent accuracy-wise results, which could be caused by the lack of data that the models have been trained on. On the other hand, we can see that negative sampling for all configurations shows better Spearman coefficient. Finally, we could like emphasize once more that learning adaptation methods come with a computational cost as we already discussed in sec. 3.2

RNN Configuration	Words	Iter	Sem (%)	Synt (%)	ρ	Time (h)
σ	1M	1	0.02	0.00	0.0095	38.50
ReLU + GC	1M	1	0.01	0.02	0.0112	46.63
σ + classes	1M	1	0.00	0.01	-0.0028	5.54
ReLU + GC + classes	1M	1	0.00	0.00	-0.0365	0.43
σ + HSM	1M	1	0.00	0.07	-0.0136	0.81
ReLU + GC + HSM	1M	1	0.00	0.00	0.0592	0.98
σ (outer representation)	1M	1	0.00	0.01	-0.0778	38.50
σ + HSM	1M	10	0.00	0.31	-0.1215	8.00
ReLU + GC + HSM	1M	10	**0.31**	0.00	**0.1905**	9.76
σ + HSM	10M	1	0.00	0.25	-0.0772	6.85
ReLU + GC + HSM	10M	1	0.00	**0.36**	-0.0409	19.57

Table 3: Model comparison with accuracy (semantic and syntactic) and Spearman coefficient ρ

3.4 RNN configurations

6 experiments have been performed to compare introduced optimization with the baseline model on the 1 million words dataset. Bigger dataset has not been considered for comparison because of the big time consumption of the standard RNN LM model.

As before we use a hidden layer of size 100. For gradient clipping we experimented with a gradient norm threshold equaled to 6.0 [Pascanu et al., 2012] and 1.0 [Le et al., 2015], the last turned to provided better results. For models with sigmoid we used initial learning rate = 0.005, for ReLU in combination with Gradient Clipping we used learning rate of 0.001 as suggested by Le et al. [2015].

Next, we have run RNN models with Hierarchical Softmax for 10 iterations and on more data (10 million) to investigate how these factors influence into the accuracy of word embedding. Furthermore, we have compared a quality of outer and inner word representation in the standard RNN LM model. Results of our experiments can be found in the table 3.

Finally, we have run RNN with Hierarchical Softmax on two configurations (with sigmoid and ReLU activations) with the same learning rate (0.001) to compare influence of ReLU into the training process [Fig 7]. It's valuable to point that the difference between configurations with ReLU and sigmoid is significant based on the fact that log-likelihood is computed from a sample data of 5000 sentences (not from the full dataset of 80K sentences) and the first epoch has been started from the huge negative value.

It can be seen that RNN with ReLU activation function has provided better results in most of cases. In the word similarity test they have outperformed the sigmoid configuration. With an increase of data/number of iterations performance growths (in our case from 0.07 % on the GloVe syntactic test to 0.25% and 0.36%).

Factorization of the output layer and hierarchical softmax reduce significantly time consumption. The different amount of time spent on RNN with HSM on 1M words for 10 iter. and for the same model but on 10M for 1 iteration can be explained by the different size of the Hierarchical softmax layer that is dependent on the number of words in the vocabulary. Because of the small size of the dataset (1M) obtained results aren't diverse enough for deeper analysis.

3.5 CBOW, SkipGram and RNN comparison

To compare CBOW, SkipGram and RNN we have run them on the same machine with the same configuration. Results are presented in the table 4.

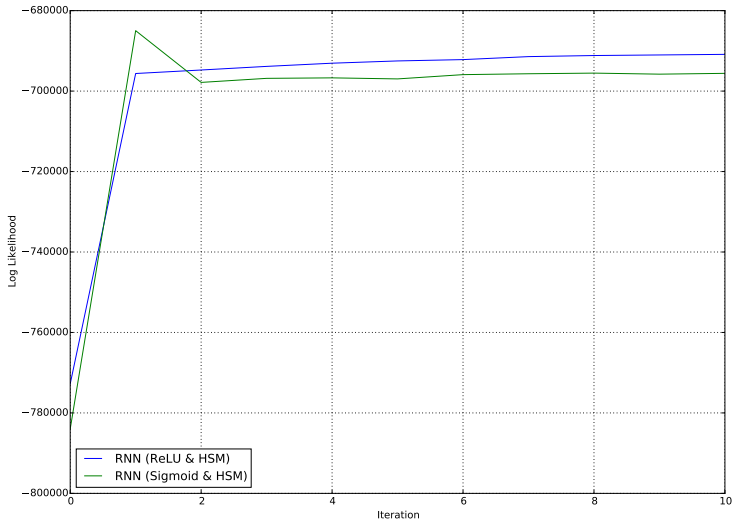


Figure 7: Subsampled log-likelihood for RNN LM with Hierarchical Softmax on two configuration with the same learning rate (0.001).

Model	**Words**	**Vocab**	**Sem (%)**	**Synt (%)**	ρ	**Time (h)**
SkipGram HSM	10M	59K	0.00	0.00	0.26	2.85
CBOW HSM	10M	59K	**1.36**	**0.9**	**0.31**	**0.40**
RNN HSM	10M	59K	0.14	0.16	0.19	0.70

Table 4: Model comparison with accuracy (semantic and syntactic) and Spearman coefficient ρ. Vector dimension = 100. Measured on a quad-core Core i5 system

The CBOW model indicated to have the best accuracy and Spearman coefficient results, which is different from results obtained in [Mikolov et al., 2013b]. One explanation could be that Skip-gram requires more data to start outperforming CBOW, and in our case that was unfeasible. The training speed difference between Skip-gram and CBOW is caused by the update equations for word-representations. RNN performed worse as well as in the original paper.

4 Discussion

The learning principle of word-vectors for Skip-gram and CBOW is based on the assumption that words that occur in similar contexts tend to have similar meanings [Wittgenstein, 1967, Harris, 1954, Jones and Furnas, 1987], and while learned vectors based on context have interesting results associated with similarity tasks shown in the original paper, they also have their own drawbacks. As was indicated in [Levy et al., 2015], word-vectors learned via Skip-gram does not capture connections between words. For example, antonyms tend to appear in a same context and therefore get similar representations, but they have different semantics. Therefore, the authors of [Levy et al., 2015] indicate that those word-vectors can't be used as inputs to simple models, such as logistic regression to perform entailment tasks. On the other hand, [Rocktäschel et al., 2015] indicated that word-vectors can be used as inputs to more complex models, such as LSTM, to perform the task.

AdaDelta is positioned as a method to overcome an initial learning rate problems associated with adaGrad, but as we have presented in fig. 6, the method is sensitive to the initial hyper-parameters, which are now two instead of one, and both require tuning. In addition, both adaGrad and adaDelta come with a cost of an extra computational overhead. However, it's unclear why learning rate adaptation methods cause worse accuracy-wise results.

Hyper-parameters such as learning rate and number of hidden units play a crucial role in our models, and so far those hyper-parameters have been chosen manually, and they can be non-optimal. We have made an attempt to select them on a data subset, however this attempt was unsuccessful, and obtained hyper-parameters were non-optimal or lead to divergence on a bigger dataset. On the other hand, it was not possible to perform model selection on a bigger dataset due to time constraints. Therefore, this problem remains an open question and will be addressed in the future work.

Finally, we are expecting that results obtained by SkipGram will outperform CBOW as reported by Mikolov et al. [2013a] with bigger amount of data. RNN also requires more data to obtain more distinguishable results.

5 Conclusion

In models comparison CBOW indicated dominating results based on accuracy, Spearman coefficient and run-time. However, given more data Skip-gram could show better than CBOW results. In CBOW configurations comparison we realized that negative sampling with $K = 17$ produces the best results, thought run-time overhead is significant.

RNN models with ReLU activation unit have produced promising results against sigmoid configurations that confirm experiments reported by Le et al. [2015]. Hierarchical softmax allows us to obtain significantly better results in comparison to standard RNN LM models with less time consumption.

6 Future work

1. Learning of word representations based on context is not the only way, an alternative could be use all words that share a syntactic connection with the target word [Levy and Goldberg, 2014].
2. In the experiments we used inner representations of words, while there are more options we could use, e.g. a combination of inner and output representations [Garten et al., 2015]
3. Efficient selection of hyper-parameters, such as learning rate and a number of hidden units.
4. Perform experiments on bigger datasets.

References

J. Andreas and D. Klein. How much do word embeddings encode about syntax? In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 822–827, Baltimore, Maryland, June 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P14-2133.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. URL http://www.iro.umontreal.ca/~lisa/pointeurs/ieeetrnn94.pdf.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. P. Kuksa. Natural language processing (almost) from scratch. *CoRR*, abs/1103.0398, 2011. URL http://arxiv.org/abs/1103.0398.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

E. Elsen. Optimizing RNN performance, 2015. URL http://svail.github.io/rnn_perf/.

J. Garten, K. Sagae, V. Ustun, and M. Dehghani. Combining distributed vector representations for words. In *Proceedings of NAACL-HLT*, pages 95–101, 2015.

Z. S. Harris. Distributional structure. *Word*, 1954.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

W. P. Jones and G. W. Furnas. Pictures of relevance: A geometric analysis of similarity measures. *Journal of the American society for information science*, 38(6):420–442, 1987.

Q. V. Le, N. Jaitly, and G. E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *CoRR*, abs/1504.00941, 2015. URL http://arxiv.org/abs/1504.00941.

O. Levy and Y. Goldberg. Dependencybased word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, volume 2, pages 302–308, 2014.

O. Levy, S. Remus, C. Biemann, I. Dagan, and I. Ramat-Gan. Do supervised distributional methods really learn lexical inference relations? In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics â AS Human Language Technologies (NAACL HLT 2015), Denver, CO*, 2015.

T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 1045–1048, 2010a. URL http://www.isca-speech.org/archive/interspeech_2010/i10_1045.html.

T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur. Recurrent neural network based language model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH 2010)*, volume 2010, pages 1045–1048. International Speech Communication Association, 2010b. ISBN 978-1-61782-123-3. URL http://www.fit.vutbr.cz/research/view_pub.php?id=9362.

T. Mikolov, S. Kombrink, L. Burget, J. Cernocký, and S. Khudanpur. Extensions of recurrent neural network language model. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2011, May 22-27, 2011, Prague Congress Center, Prague, Czech Republic*, pages 5528–5531, 2011. doi: 10.1109/ICASSP.2011.5947611. URL http://dx.doi.org/10.1109/ICASSP.2011.5947611.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013a. URL http://arxiv.org/abs/1301.3781.

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

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013c.

T. Mikolov, W. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA*, pages 746–751, 2013d. URL http://aclweb.org/anthology/N/N13/N13-1090.pdf.

A. Mnih and G. E. Hinton. A scalable hierarchical distributed language model. In *Advances in neural information processing systems*, pages 1081–1088, 2009.

F. Morin and Y. Bengio. Hierarchical probabilistic neural network language model. In *Proceedings of the international workshop on artificial intelligence and statistics*, pages 246–252. Citeseer, 2005.

R. Pascanu, T. Mikolov, and Y. Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012. URL http://arxiv.org/abs/1211.5063.

T. Rocktäschel, E. Grefenstette, K. M. Hermann, T. Kočiskỳ, and P. Blunsom. Reasoning about entailment with neural attention. *arXiv preprint arXiv:1509.06664*, 2015.

X. Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

Y. Shi, W. Zhang, J. Liu, and M. T. Johnson. RNN language model with word clustering and class-based output layer. *EURASIP J. Audio, Speech and Music Processing*, 2013: 22, 2013. doi: 10.1186/1687-4722-2013-22. URL http://dx.doi.org/10.1186/1687-4722-2013-22.

L. Wittgenstein. *Philosophische Untersuchungen. Philosophical Investigations; Translated by GEM Anscombe. Reprinted*. Blackwell, 1967.

M. D. Zeiler. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.