Optimizations of the Skip-Gram model with negative sampling

Milinaire Cédric

Chair of Data Science
University of Passau, Germany

1 April 2019

Overview of my thesis

- Word embeddings are vector representations of words
- Word embeddings are a powerful tool that facilitate NLP
- Skip Gram Model with negative sampling, is a simple and powerful algorithm (Mikolov et al.) [1]
- This work focused on optimizing the convergence time
- Techniques used:
    - Advanced optimizers
    - Input shuffling

Outline

1. Motivation
2. Background
   1. Skip Gram Model
   2. Skip Gram Model with negative Sampling
3. Implementation
4. Results
5. Discussion
6. Conclusion

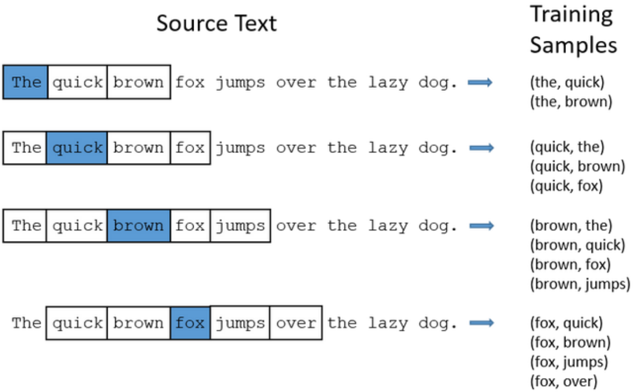
Background

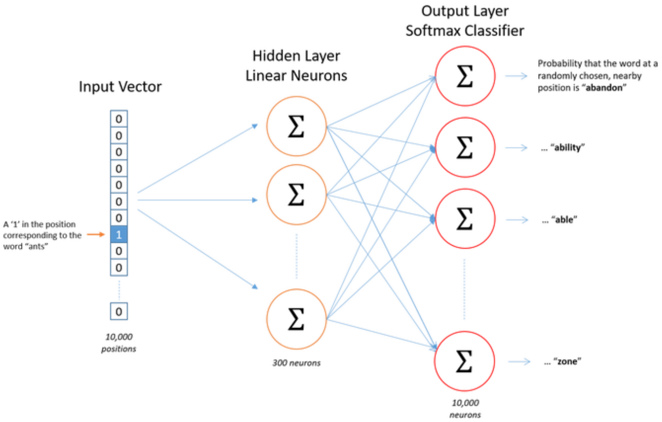
1. Skip Gram Model
2. Skip Gram with Negative Sampling (SGNS)

Background

Main idea: train a network on a "fake task" then use the weights as embedding.

- The fake task:
- Given a word w guess the context words.





(Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/)

Background

Softmax:
$$p(c|w) = \frac{exp({v_c'}^\intercal v_w)}{\sum_{i=1}^{T} exp({v_i'}^\intercal v_w)} \quad (1)$$

v' is the output layer vector v is the input layer vector

Negative Sampling

- Distinguish data from noise $\Rightarrow$ reduce problem to a logistic regression.
- Guess k random samples
- For each pair (w, c) we get:

$$\arg\max_{\theta}\ log(\sigma({v_c'}^\intercal v_w) + \sum_{k \in K} log(\sigma(-{v_k'}^\intercal v_w)) \quad (2)$$

- Uses SGD as an optimizer

State of the Art

- word2vec (Mikolov et al. 2013) [1]
- Parallelizing Word2Vec in Shared and Distributed Memory (Ji et al. 2016)[2]
- Acceleration of Word2vec Using GPUs (Seulki and Youngmin 2016) [3]
- Gensim (Řehůřek and Sojka) [4]

Research Questions:

Can the convergence time of the skip Gram Model be optimized by the use of:

- Advanced optimizers

and

- Input Shuffling

while at the same time maintaining it's accuracy?

Our Implementation

Main Idea:

- Create a large batch of training samples, i.e 2000 pairs
- Compute loss for each pair
- Use sum over all pairs as loss for batch

Our Implementation

Illustration of the batched Skip-Gram Model

$X = (v_1, c_1), (v_2, c_2), (v_3, c_3)$ Input:

$$v = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}, c = \begin{bmatrix} c1 \\ c2 \\ c3 \end{bmatrix} \text{ and } A = \begin{bmatrix} k_{1,1} & k_{2,1} & k_{3,1} \\ k_{1,2} & k_{2,2} & k_{3,2} \\ k_{1,3} & k_{2,3} & k_{3,3} \end{bmatrix}$$

We then concatenate c and A, resulting in:

$$\tilde{A} = \begin{bmatrix} c_1 & k_{1,1} & k_{2,1} & k_{3,1} \\ c_2 & k_{1,2} & k_{2,2} & k_{3,2} \\ c_3 & k_{1,3} & k_{2,3} & k_{3,3} \end{bmatrix}$$

Our Implementation

Embeddings:

$$E_v = \begin{bmatrix} \tilde{v}_{11} & \dots & \tilde{v}_{1d} \\ \tilde{v}_{21} & \dots & \tilde{v}_{2d} \\ \tilde{v}_{31} & \dots & \tilde{v}_{3d} \end{bmatrix}$$, where $\tilde{v}_i = \begin{bmatrix} \tilde{v}_{i1} & \dots & \tilde{v}_{id} \end{bmatrix}$ is the embedding of v_i.

$$E_c = \begin{bmatrix} \tilde{c}_1 & \tilde{k_{1,1}} & \tilde{k_{2,1}} \\ \tilde{c}_2 & \tilde{k_{1,2}} & \tilde{k_{2,2}} \\ \tilde{c}_3 & \tilde{k_{1,3}} & \tilde{k_{2,3}} \end{bmatrix}$$, where each entry of the matrix is a vector of dimension d

Batch multiplication and negation of samples:

$$S = \begin{bmatrix} \tilde{v}_1 \cdot \tilde{c}_1 & -\tilde{v}_1 \cdot \tilde{k_{1,1}} & -\tilde{v}_1 \cdot \tilde{k_{2,1}} & -\tilde{v}_1 \cdot \tilde{k_{3,1}} \\ \tilde{v}_2 \cdot \tilde{c}_2 & -\tilde{v}_2 \cdot \tilde{k_{1,2}} & -\tilde{v}_2 \cdot \tilde{k_{2,2}} & -\tilde{v}_2 \cdot \tilde{k_{3,2}} \\ \tilde{v}_3 \cdot \tilde{c}_3 & -\tilde{v}_3 \cdot c_3\tilde{k_{1,3}} & -\tilde{v}_3 \cdot c_3\tilde{k_{2,3}} & -\tilde{v}_3 \cdot \tilde{k_{3,3}} \end{bmatrix}$$

Loss computation:

$$\mathrm{L} = -\sum_{(i,j)\in k\times n} S(i,j)$$

Implementation

1. Setting
   - Dataset
   - Network Architecture
2. Optimization Process

Dataset

- Text8 dataset
- First 30MB of clean text from wikipedia
- Vocabulary $\approx$ 250k word (small)
- Subsampling $\implies$ 50% decrease of data set size

Optimization process

Optimization techniques:

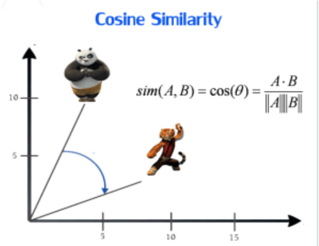
- Advanced Optimizers
  - Momentum
  - Nesterov accellerated Momentum
  - Adagrad
  - Adam
- Input Shuffling

Results

- Rating our work
  - Word similarity
  - Convergence time
- Results
  - Advanced Optimizers
  - Input Shuffling
- Discussion
  - Comparison to Gensim and other related work

What is word similarity?

- Two word embeddings are close to each to other if their cosine distance is small.
- Pairs of word rated between 1 and 10 on their similarity,
- `['FBI', 'investigation', '8.31', 'Mars', 'scientist', '5.63']`
- We are going to rank our model on the corelation between the distance of the word pairs and the human score.



Word Similarity

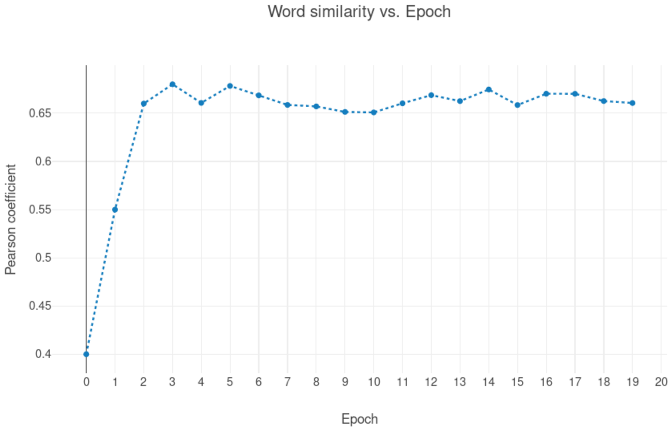
- Word Similarity vs. Related Work

model	Word Similarity
Gensim	**0.66**
w2vec (original)	0.63
our Work	**0.66**

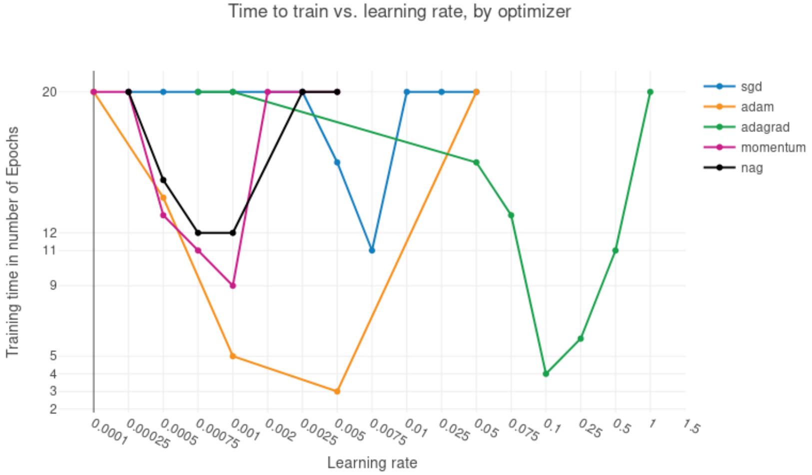
(Ji et al. 2016) [2]

Convergence time

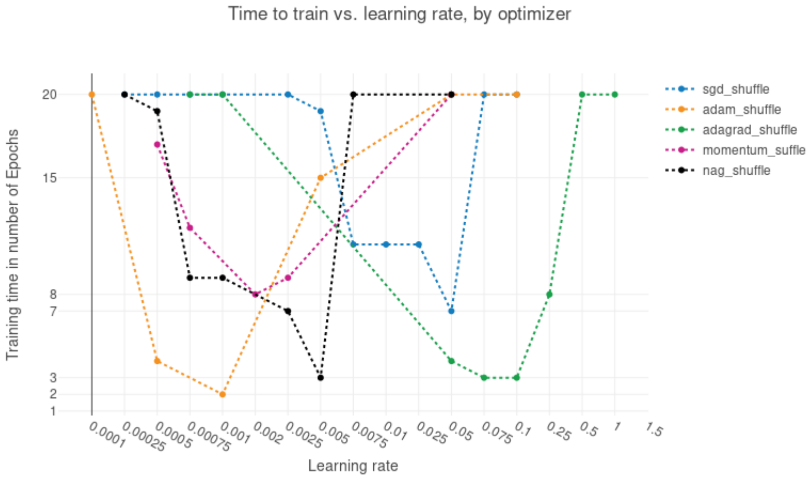
- Defined convergence time based on word similarity
- Early Stoppage if: $\rho - \rho_{prev} < 0.009 \vee \rho > 0.66$
- No more than 20 epochs.



Advanced Optimizers



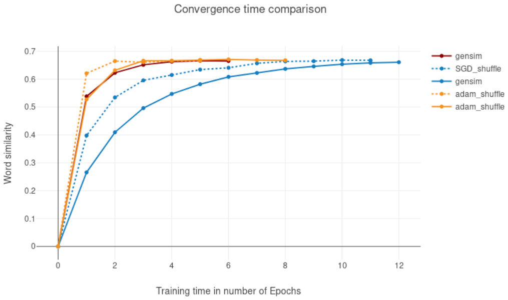
Input Shuffling



Discussion

Convergence time vs Gensim

Model	Convergence Time	Word Similarity
SGD	11	0.65
SGD w/shuffling	7	0.66
Adam	3	0.66
Adam w/ shuffling	**2**	0.66
Gensim	4	0.66



Questions that arises from the Thesis

- Can the results be replicated on other datasets?
- Can the results be replicated on other tasks?

Conclusion

- Skip Gram Model powerful yet simple tool to create word embedings
- Advanced optimizers especially Adagrad and Adam improve convergence time
- Improved convergence time, while maintaining accuracy

Solution to the batched Approach

How can we improve the batched approach?

- A batch without double appearing words?
- Analyze the distribution of words in the dataset?
- Creating the perfect batch?
- Delete frequent occuring words from the dataset?

Distribution of Words

Problem:

Words appear more than once in a batch $\rightarrow$ performance loss

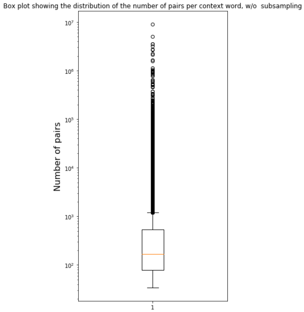
Solution:

Create batch of different sizes, each batch will hold at most one pair per context word

Problem of the Solution:

Average Batch Size = 200, i.e training takes too long

Distribution of Words



Results of the Distribution

- A few words are responsible for the majority of pairs.
- They almost have the same context words
- Have they the same representation?

Distribution of Words

Yes they have! blablalbab

First Results

Future Work

- Creating the perfect batch
- Analyze the deletion of outliers on other (bigger) datasets.

It's unsuitable to compute the softmax

$$p(c|w) = \frac{exp(v_c'^{\intercal} v_w)}{\sum_{i=1}^{T} exp(v_i'^{\intercal} v_w)} \tag{3}$$

- v_w and $v_c^{'}$ are the "input" and "output" representation of w
- For each pair we have to go over the whole training corpus. (Billions of word in practice)

Network Architecture

- Dimension of input and output vectors = 100
- Context window = 5
- Negative Samples = 10
- Coded in Pytorch 1.0

First 10 pairs of training:

```
[('anarchism', 'originated'),
 ('anarchism', 'abuse'),
 ('originated', 'abuse'),
 ('abuse', 'first'),
 ('abuse', 'originated'),
 ('abuse', 'working'),
 ('abuse', 'class'),
 ('abuse', 'radicals'),
 ('abuse', 'diggers'),
 ('first', 'working')]
```

Negative Samples:

```
['zero',
 'achieved',
 'doubts',
 'place',
 'nine',
 'york',
 'has',
 'zero',
 'while',
 'gunner']
```

Each parameter θ_i, at time step t will have it's own learning rate $\eta_{t,i}$

$$\eta_{t,i} = \frac{\eta_0}{\sqrt{\sum_{i=1}^{t} g_{t,i}^2 \epsilon}} \tag{4}$$

where

- $g_{t,i} = \nabla J(\theta_{t,i})$ is the partial derivative of the loss function with respect to the parameter θ_i at time step t.
- each parameter θ_i has it's one learning rate
- $\theta_{t+1,i} = \theta_{t_i} - \eta_{t,i} g_{t,i}$

We can now construct our global parameter update as follows:

$$\theta_{t+1,i} = \theta_{t_i} - \frac{\eta}{\sqrt{G_{t_{i,i}}} + \epsilon} g_{t,i}, \tag{5}$$

with $G_{t_{i,i}}$ being the diagonal Matrix of the sum of the squares of the graditents ($g_{t,i}$).

$$He = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad is = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad King = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Input Layer 3x3 **Output Layer 3x3**

$$\begin{pmatrix} 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{pmatrix} = \begin{pmatrix} 2 & 2 & 2 \end{pmatrix} \begin{pmatrix} 0.1 & 0.2 & 0.3 \\ 0.1 & 0.2 & 0.3 \\ 0.1 & 0.2 & 0.3 \end{pmatrix}$$

$$= \begin{pmatrix} 0.6 & 1.5 & 3 \end{pmatrix} \implies Softmax : \begin{pmatrix} 0.13 & 0.31 & 0.56 \end{pmatrix}$$

Probabilities: $p(v_{he}|v_{is})$ $p(v_{is}|v_{is})$ $p(v_{king}|v_{is})$

References

Mikolov, Tomas and Sutskever, Ilya and Chen, Kai and Corrado, Greg S and Dean, Jeff, 2013, *Distributed representations of words and phrases and their compositionality*

Ji, Shihao and Satish, Nadathur and Li, Sheng and Dubey, Pradeep, 2016, *Parallelizing word2vec in shared and distributed memory*

Bae, Seulki and Yi, Youngmin, 2016, *Acceleration of Word2vec Using GPUs*

Radim Řehůřek and Petr Sojka, 2010, *Software Framework for Topic Modelling with Large Corpora*