Semi supervised learning by incorporating topic model and word embedding in neural network

Quan Duong
CS Department
University of helsinki
quan.duong@helsinki.fi

Miikka Silfverberg
DH Department
University of Helsinki
miikka.silfverberg@helsinki.fi

Hande Celikkanat
DH Department
University of Helsinki
hande.celikkanat@helsinki.fi

Abstract

Word embedding has been widely used for word's representation in natural language processing. In this study, we will implement skip-gram model with negative sampling by Pytorch framework and compare it with other embedding models. In addition to that, this article also explore *Latent Dirichlet allocation* (LDA) topic modeling algorithm as an unsupervised task for making gold label to the supervised task. The application implemented for this study is about grouping documents in different topics, then allow to query relevant documents in topics by keywords. To achieve that, we create a hybridge language model which combines word embedding and LDA methods to for a supervised training in neural network.

1 Introduction

In media industry, there are ton of articles are published every day. The goal for the publishing companies is to reach a highest number of views when possible, but the quality feedbacks from the users are also important. For instance, there are articles in different categories of sport, politic and fashion. By mapping users interest with the article's topic, publisher could delivery more relevant articles to user or plan a efficient marketing strategy. A good recommender system could gains many traffic to the website and turn to more conversion on advertisement.
If an user who prefers to read more about fashion news, the system should suggest more articles and ads about fashion products.
There are many ways to identify users favour like click or review. Ather approach is based on the keywords of users in searching, companies want to match these keywords to the same categories of articles. With the keyword as a signal, perhaps, it could bring more broad possibility to understand user's characteristic than clicks which need to be collected in a massive number.
The difficulty is that, there is no labeled topics for the articles. Or if any user enter the keywords not existing yet on the vocabulary of docs before, the result could be empty. To solve these problems, we suggest to use topcic modeling algorithm LDA and word2vec embedding. The idea is that we train the word2vec model in the larger vocabs. After that, we use LDA to cluster articles in different groups, which we called topics. In each topic, there are keywords which

contributed for the topic ordered by probability. By taking the topic as label, the keywords as the input, we can train a neural network with embedding layer to have a topic-keywords model. The following section will go deeper on the implementation of the methods.

2 Word2vec skip-gram

Word embedding is described as feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers.
The simplest way to encode a word in a vectors space is using one hot encoding, which generate a huge numbers of dimension and depend on the length of vocab and cannot present the relationship between the words. Conceptually, to have a better vector representation of word, we need a method to convert it to lower dimension vector, a group of related models to this called word2vec. There are two typical methods for the implementation in word2vec: *Skip-gram* and *Continuous Bag of Words* (CBOW) with their various optimizations such as hierarchical softmax and negative sampling. [2]

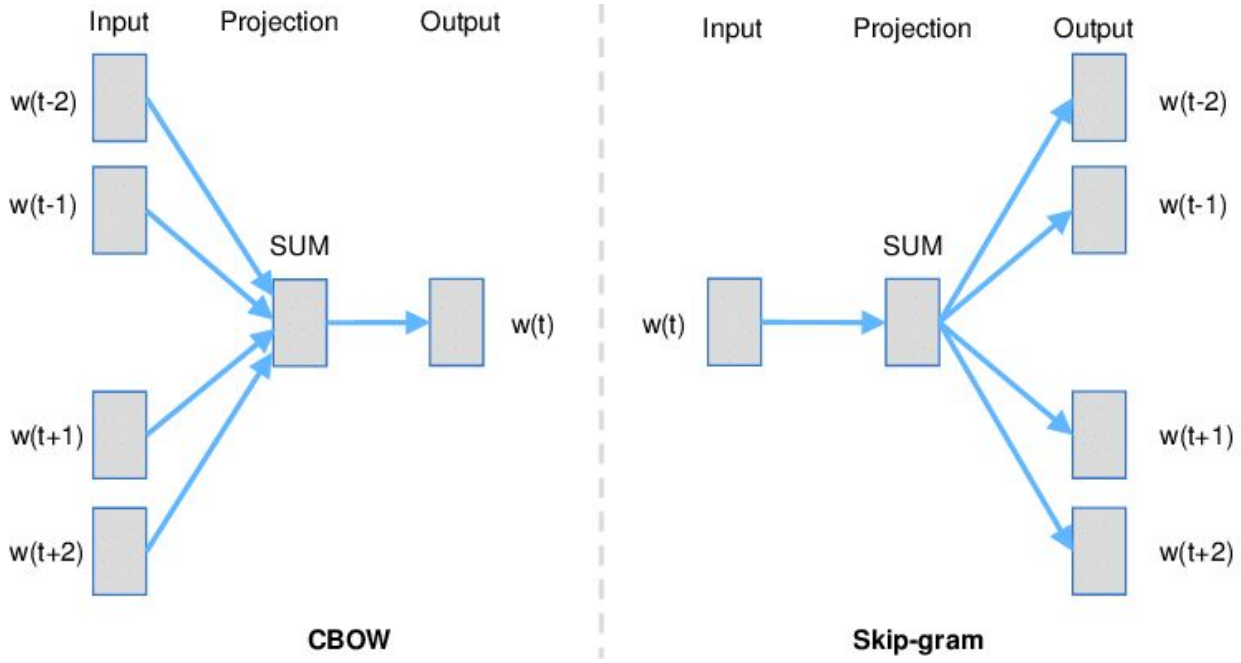

Figure 1: Continuous Bag-of-words (CBOW) and Skip-gram (SG) training model illustrations [1]

In the continuous bag-of-words architecture, the model predicts the current word from a window of surrounding context words. In the continuous skip-gram architecture, the model uses the center word to predict the context words. [1]

Basic assumptions is that similar words will share the similar context. For example, the context of “happy” and “excited” may be similar because we can easily replace a word with the other and get meaningful sentences.

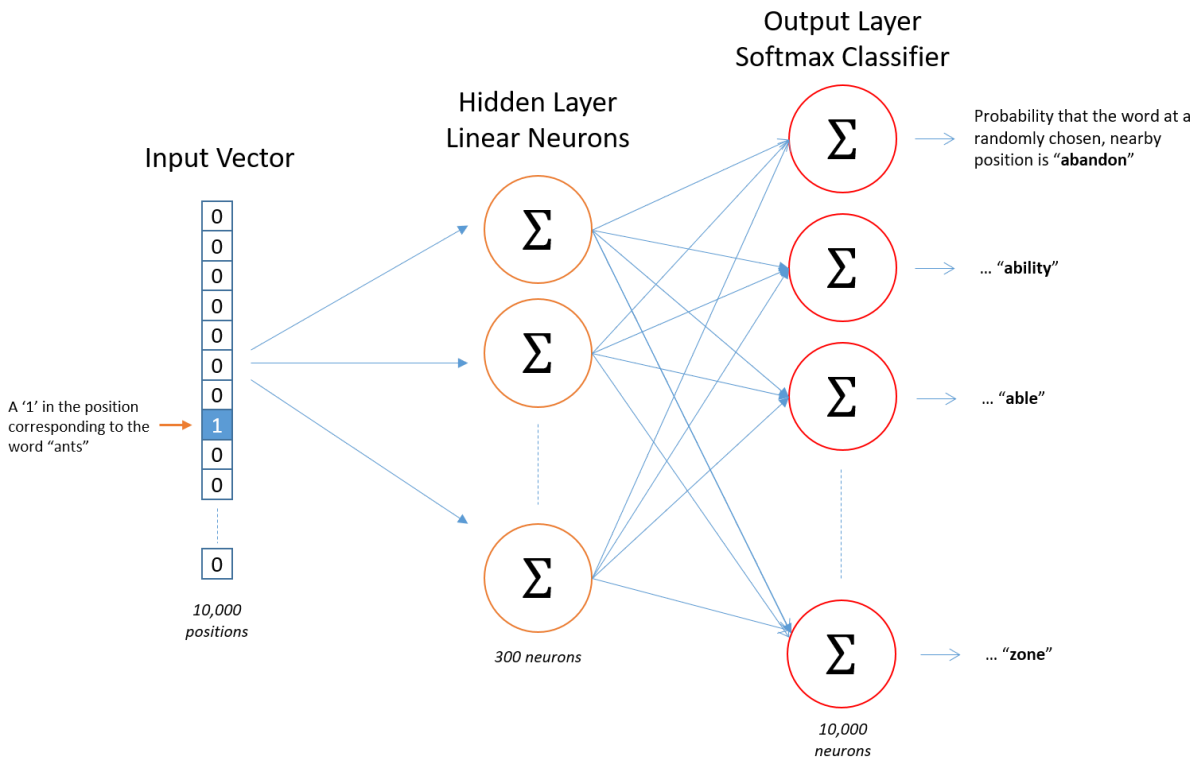

Figure 2: Skip-gram network architecture [3]

The input for network is one hot encoded vectors, the output is probability of the word chosen is nearby to the input and hidden layer acting as the lookup table for the input word.

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

Figure 3: Input layer and hidden layer [3]

In the training, to find the position of a word in the vectors table, we just need to multiply the input vector with the hidden table. This vector will be transferred to the output layer.

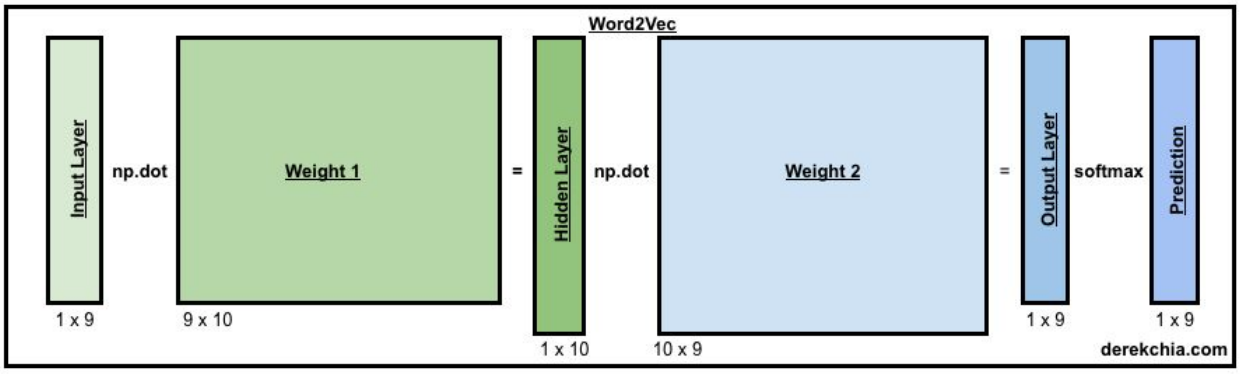

Figure 4: Network architecture [4]

For example, the word “ants” after go through hidden layer will have presentation of a vector 300 size, it is multiplied with others word in output layer to produce the probability for each word in output layer. The softmax function will calculate the output’s probability and the sum of all probabilities is 1.

3 Negative Sampling

In the training phase, for each iteration, the weight of all the words need to be updated. However, passing updating the weight for billion samples is a tough task or impossible with normal computation resource. There is one technique to solve this problem called negative sampling. The idea is, whenever the center word and context words need to be evaluated as 1, we also take some words which be considered as negative and flag them as 0. Thus, each time of updating the weights, only few words need to be adjusted. Let says if we have 1 millions words and 300 features, and each time, we only need to update 1 positive word1 and 5 negative words, the number of weight is lowered down a lots to 6 x 300 instead of 300 millions.

To decide what should be the words used as negative samples, the paper suggest that the words which has more frequency. The probability for a words be selected as negative sample is calculated as the formula below [2]:

$$P(w_i) = \frac{f(w_i)}{\sum_{j=0}^{n} f(w_j)} \qquad (1)$$

The idea implemented on the paper [2] is using a table called “unigram distribution” with the size of 100 millions elements. Then multiple the probability of each words with the table size as the number of the words will appear in the table. For example, the word “and” has probability is 0.001, it will be filled 0.001 x 100.000000 times in the table.
However, in the implementation code, the formula has been changed a bit. The authors give a threshold as power of 0.75 for the frequency of the analyzing word and other words [2].

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} f(w_j)^{3/4}} \quad (2)$$

After having the negative table, for each word as a target, we randomly select N words as the negative samples pair and feed all of them to the network. The loss function for this is calculated as follow:

$$\begin{aligned} E &= -\log p(w_{O,1}, w_{O,2}, \cdots, w_{O,C} | w_I) \\ &= -\log \prod_{c=1}^{C} \frac{\exp(u_{c,j_c^*})}{\sum_{j'=1}^{V} \exp(u_{j'})} \\ &= -\sum_{c=1}^{C} u_{j_c^*} + C \cdot \log \sum_{j'=1}^{V} \exp(u_{j'}) \end{aligned} \quad (3)$$

where j_c^* is the index of the actual c-th output context word in the vocabulary. The loss function comprises of 2 parts. The first part is the negative of the sum for all the elements in the output layer (before softmax). The second part takes the number of the context words and multiplies the log of sum for all elements (after exponential) in the output layer. [5]

4 Implementation for word2vec

4.1 Dataset

We implement the skip-gram with negative sampling model in Pytorch - a python framework for deep learning. The dataset is from https://github.com/ufal/rh_nntagging project, which provides the dataset for 37 languages included English and Finnish. The dataset for one language divided into train, dev and test set. However, our word2vec model is unsupervised learning, so we only need to deal with training set. The data is in conllu format, which need to be parsed to the normal list type of python. In the implemented code, there are the function to handle this automatically. You only need to select the language to download and get the cleaned tokens for the result.

In the English training set, there are 12543 sentences, around 450500 words with 3185 unique words.

In the Finnish training set, there are 12217 sentences, around 708000 words with 3833 unique words.

4.2 Experiments

The code are divided into 4 modules:

- Utils: handle the task related dataset like download, clean, read dataset
- Preprocessing: prepare the input for the model. For example, create negative sample, create pair of input and context words, and generate batches for training.
- Model: This is where the skip-gram model implemented.
- Main: In this main module, the training phase be called. The Word2Vec class used as a pipeline for all the steps, from getting data to preprocessing and training, saving the model.

Running the code:

- Install all dependencies from requirements.txt
- From terminal, cd to the "project" folder, run the command "`python src/main.py`"
- This will ask you the language you want to train, for example "english", "finnish", "swedish"
- Next, this will ask you how many epochs you want to run, for quickly test, you can run 1 epoch.
- Finally, the model will be saved to the "`models/w2v.txt`"

Features:

- Download the dataset from repository based on the language we enter. If the data is already downloaded, it will ignore this step.
- Parse conllu format to the tokenized text
- Read normal corpus in text format (train normal text, not conllu)
- Read gensim dataset
- Negative sampling with unigram samples table
- Sub sampling by simple removing stopwords (the proper function can be implemented later)
- Save the model after train

The Word2vec class has following arguments with default values:

- lang = "english"
- n_epoch = 20
- batch_size = 500
- embed_dim = 300
- window_size = 5
- neg_sample = 10
- min_count = 5
- lr = 0.01
- report_every = 1

As mentioned in the paper by Mikolov et al [2], the optimal for negative sample number is 5 to 20 for small training set, while 2 - 5 is good for large training set. Because our training set is quite small, so neg_sample = 10 is selected.
You can find the comment in the code and the notebooks.

Pipeline

Firstly, the function get_data() will be called, the data will be downloaded and cleaned here.
After that, we will process the tokenized data by following steps:

- Generate vocab, word to index, index to word
- Generate the pair of input and context word vectors based on the window_size
  For example, the word C with window_size = 2 will have vectors of A, B and D, E as context words, and there are 4 pairs in total.
- Generate negative sample table with the size of 100M words.
- Generate batch size by combining pairs of words with negative sample. For example, with the sample_size = 5, for each pair of words, we will random choose 5 negative words in the negative samples table.
- The batches list has the shape [[u, u, u], [v, v, v], [[neg, neg], [neg, neg], [neg, neg]]]
  Where “u” is the input word, “v” is the context word, and “neg” is the negative samples.
- The training process take u, v, and neg as the input for forward function
- After the training done, the vectors will be saved to a file as the model

4.3 Evaluation

To make the comparison, we create another 3 different word2vec models from gensim library.

- Gensim skip-gram negative sampling (G-SGN)
- Gensim continuous bag of words (G-CBOW)
- Gensim fasttext (G-FT)
- Our implemented skip-gram negative sampling model (W2V)

Similarity score
The language for the first round is English. Gensim provided the datasets for evaluating similarity score of word2vec model called “wordsim353” and “simlex999”. These 2 datasets was created for measuring word similarity or relatedness, however, only available for English.
Using *evaluate_word_pairs* function from gensim, the result return in this tuple of 3 elements format:

- `(0.311748743, 0.00011478623)`: **pearson (tuple of (float, float)) – Pearson correlation coefficient with 2-tailed p-value.**
- `SpearmanrResult(correlation=0.31643317995983,pvalue=8.9249064003694)`: **spearman (tuple of (float, float)) – Spearman rank-order correlation coefficient between the similarities from the dataset**

and the similarities produced by the model itself, with 2-tailed p-value.

- 58.07365439093485: **oov_ratio (float) – The ratio of pairs with unknown words.**

Table 1: The Spearman correlation coefficient between the similarities from dataset and the similarities produced by the model. (-1, 1 farther away from 0 is better)

	G-SGN	G-CBOW	G-FT	W2V
wordsim353	0.290	-0.036	-0.134	0.316
simlex999	0.056	0.034	-0.018	0.035

Based on the spearman similarities score in the table 1, we can see our model outperform gensim's model in wordsim353 dataset. In simlex999 dataset, gensim skip-gram model takes the first place, but our model gives a better performance compared to the others G-CBOW and G-FT. Interestingly, G-CBOW and G-FT is showing relatively low scores compared to the others two.

Analogies score

Next evaluation is analogy. Gensim provides *evaluate_word_analogies* function which has the questions and evaluates based on the correct answers.

- G-SGN: 0.013084112149532711 score

```
Correct :   ('BAGHDAD', 'IRAQ', 'KABUL', 'AFGHANISTAN'),
            ('BAGHDAD', 'IRAQ', 'KABUL', 'AFGHANISTAN'),
            ('EASY', 'EASIER', 'SMALL', 'SMALLER'),
            ('HIGH', 'HIGHER', 'SAFE', 'SAFER'),
            ('SMALL', 'SMALLER', 'EASY', 'EASIER'),
            ('BAD', 'WORST', 'GOOD', 'BEST'),
            ('GOOD', 'BEST', 'BAD', 'WORST')
```

- G-CBOW: 0.009345794392523364 score

```
Correct :   ('BAD', 'WORST', 'GOOD', 'BEST'),
            ('EASY', 'EASIEST', 'GOOD', 'BEST'),
            ('EASY', 'EASIEST', 'BAD', 'WORST'),
            ('ITALY', 'ITALIAN', 'INDIA', 'INDIAN'),
            ('RUSSIA', 'RUSSIAN', 'INDIA', 'INDIAN')
```

- G-FT: 0.022429906542056073 score

```
Correct :   ('BOY', 'GIRL', 'MAN', 'WOMAN'),
            ('COMPLETE', 'COMPLETELY', 'IMMEDIATE', 'IMMEDIATELY'),
            ('IMMEDIATE', 'IMMEDIATELY', 'COMPLETE', 'COMPLETELY'),
            ('SERIOUS', 'SERIOUSLY', 'IMMEDIATE', 'IMMEDIATELY'),
```

```
                ('SERIOUS', 'SERIOUSLY', 'OBVIOUS', 'OBVIOUSLY'),
                ('BIG', 'BIGGER', 'LONG', 'LONGER'),
                ('GOOD', 'BETTER', 'GREAT', 'GREATER'),
                ('LONG', 'LONGER', 'BIG', 'BIGGER'),
                ('LOW', 'LOWER', 'LONG', 'LONGER'),
                ('BAD', 'WORST', 'HIGH', 'HIGHEST'),
                ('RUSSIA', 'RUSSIAN', 'EGYPT', 'EGYPTIAN'),
                ('SPAIN', 'SPANISH', 'INDIA', 'INDIAN')
```

- W2V: 0.009345794392523364 score

```
Correct :   ('PARIS', 'FRANCE', 'ROME', 'ITALY'),
            ('PARIS', 'FRANCE', 'ROME', 'ITALY'),
            ('CHEAP', 'CHEAPER', 'EASY', 'EASIER'),
            ('LARGE', 'LARGER', 'CHEAP', 'CHEAPER'),
            ('GOOD', 'BEST', 'BAD', 'WORST')
```

Both our model and G-CBOW answered 5 question correctly. The G-SGN model is better with 7 correct times. Surprisingly, the G-FT show the best result with 12 correct times even give the worst score in similarity check above.

Re-checking

Now we want to see how the model acting when select a random words and try to validate by human knowledge. The word has been chosen here is “soup”. We will get the similarity words to this one.

Table 2: Similar words to “soup” in different models

G-SGN:	***G-CBOW:***
[('cuban', 0.9674544334411621), ('willow', 0.9575533866882324), ('fruit', 0.9544236063957214), ('burger', 0.9542075991630554), ('authentic', 0.950620174407959), ('salad', 0.9498319625854492), ('gem', 0.9488934278488159), ('pie', 0.942488431930542), ('spa', 0.9398939609527588), ('enjoyable', 0.9398787021636963)]	[('garden', 0.9993675947189331), ('chocolate', 0.9992362260818481), ('dessert', 0.9992275238037109), ('factor', 0.9991266131401062), ('cake', 0.9989535808563232), ('bradley', 0.9989365935325623), ('urine', 0.9988648295402527), ('lovely', 0.9988009929656982), ('guinea', 0.9987917542457581), ('recieve', 0.9987843632698059)]
G-FT:	***W2V:***
[('obvious', 0.9979274272918701), ('hide', 0.9978873133659363), ('temperature', 0.99689555168151), ('hindu', 0.9967877268791199),	[('salad', 0.8774807453155518), ('chocolate', 0.8551594614982605), ('gem', 0.8508447408676147), ('sandwich', 0.8381732702255249),

('linda', 0.9964937567710876), ('powerful', 0.9964475631713867), ('funding', 0.9963701963424683), ('music', 0.9963683485984802), ('chinese', 0.9963516592979431), ('travel', 0.9962276220321655)]	('ham', 0.8251281976699829), ('waiter', 0.821048378944397), ('cuban', 0.8197777271270752), ('sauce', 0.8185526132583618), ('authentic', 0.8137593865394592), ('luna', 0.812670111656189)]

Obviously, the W2V and G-SGN give the most reasonable result for the world “soup”. The G-CBOW gives 3 words “chocolate, dessert and cake” is related but the rest is wrong. The G-FT model shows irrelevant result. This test reflects the similarity score we evaluated above is correct.

In Finnish corpus, we don’t have dataset for evaluating similarity and analogy score. We can use the same method which evaluate by a random word. In this case, we continue select the word “soup” in Finnish is “keitto”.

Table 3: Similar words to “keitto” in different models

G-SGN: [('kasvis', 0.9235570430755615), ('kera', 0.9183982610702515), ('sekaan', 0.913447380065918), ('keittää', 0.9067599177360535), ('pekoni', 0.9018987417221069), ('juures', 0.8989263772964478), ('herne', 0.8963477611541748), ('suklaa', 0.8957052826881409), ('tomaatti', 0.8957021832466125), ('pannu', 0.8946678042411804)]	***G-CBOW:*** [('järistys', 0.36036741733551025), ('liisa', 0.33173853158950806), ('koulu', 0.3075469732284546), ('rengas', 0.28496959805488586), ('puuha', 0.2849036753177643), ('sääty', 0.28404876589775085), ('tomaatti', 0.2806519865989685), ('luonto', 0.27573275566101074), ('kehottaa', 0.2743917405605316), ('kunnia', 0.27247995138168335)]
G-FT: [('peitto', 0.9854873418807983), ('kenties', 0.9183449149131775), ('kenttä', 0.9035232663154602), ('keittiö', 0.8903461694717407), ('lomake', 0.8643661737442017), ('pelto', 0.8629583120346069), ('kierto', 0.8625518083572388), ('accenture', 0.8618196845054626), ('hiihto', 0.8604219555854797), ('baker', 0.8582370281219482)]	***W2V:*** [('kattila', 0.8759456872940063), ('kasvis', 0.8713175058364868), ('suklaa', 0.8412356376647949), ('herne', 0.8211257457733154), ('pekoni', 0.8163641691207886), ('pitsa', 0.815356969833374), ('keittää', 0.8106091022491455), ('lohi', 0.8072177171707153), ('jauhe', 0.8058486580848694), ('mureke', 0.8032611608505249)]

The W2V model is still showing the best relevant result following by G-SGN. The G-CBOW and G-FT remain the poor performance in this case.

4.4 Visualization

We use TSNE to visualize 200 words vector in the implemented model

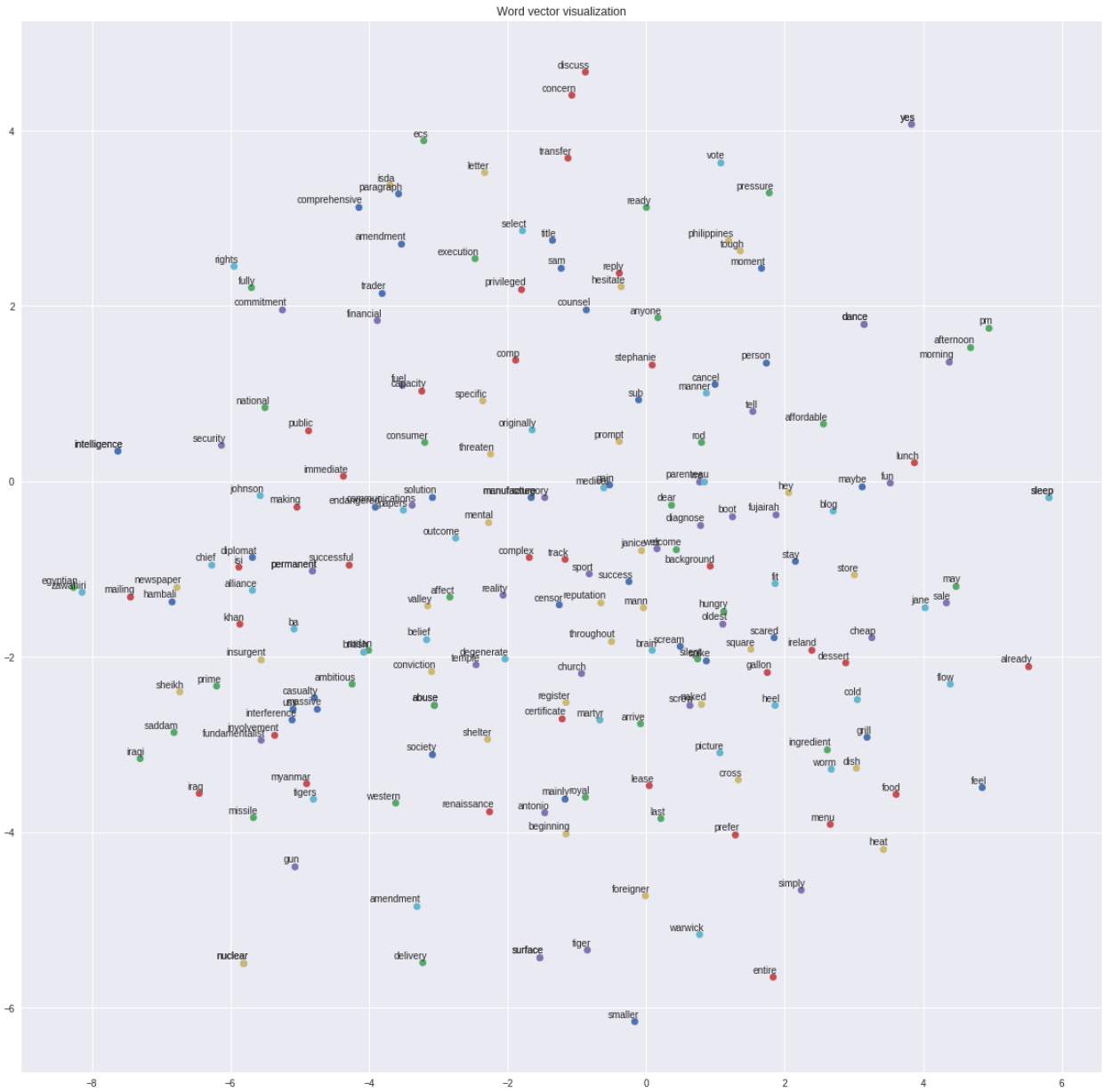

Figure 5: Word2vec visualization by TSNE

To get the visualization, run the function tsne_plot in notebook. Notice when plot gensim model that the input is the vectors of word from gensim model, not the gensim model itself.

5 Implementation for topic modeling

5.1 Dataset

The dataset includes 7303 articles in Finnish which having “body, title, description and category” as the text features.

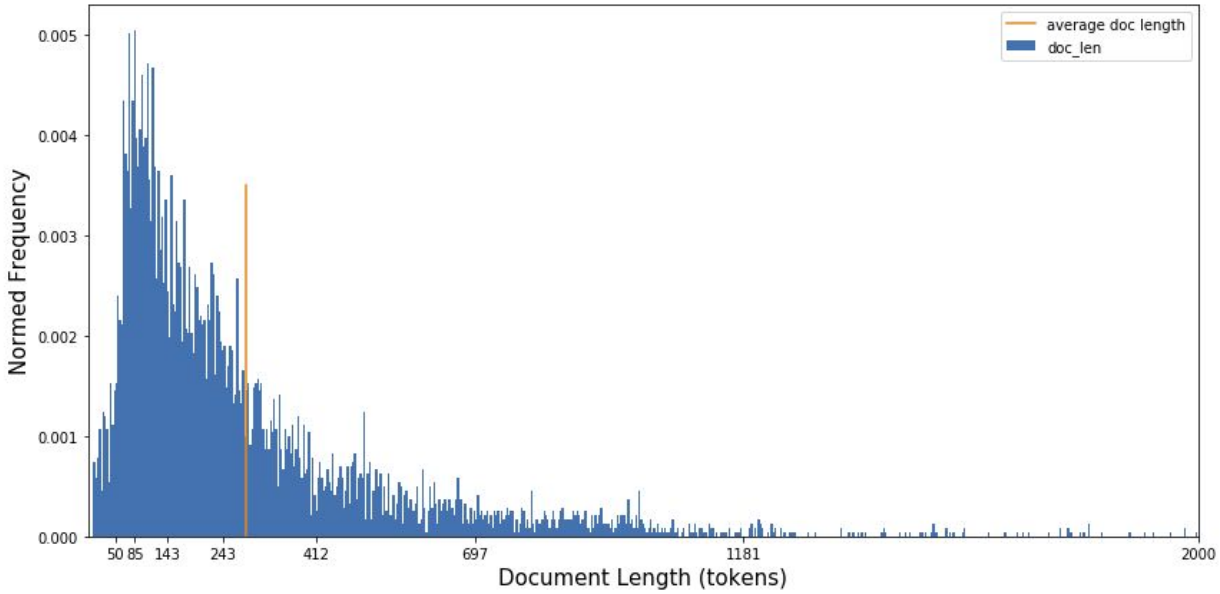

Figure 6: Number of words in documents distribution

- Average document length 284.06791729426266
- Minimum document length 10
- Maximum document length 3316
- Number document less than 50 words: 285

After cleaning, removing stop words, we have 7005 articles left which have more than 50 words for training.

5.2 Topic modeling

Topic modeling is the process of identifying topics in a set of documents. This can be useful for search engines, customer service automation, and any other instance where knowing the topics of documents is important. Latent Dirichlet Allocation (LDA) is an algorithm used widely in topic model. The idea of LDA is the way a document was generated was by picking a set of topics

and then for each topic picking a set of words [6]. We will not go into the theoretical detail of this algorithm. Instead, we will implement the topic modeling for our dataset by Gensim library.
Gensim library provides ready solution to use LDA in easy way. There are couple of parameters we need to consider. Firstly, we need to find a suitable the number of topics. Because this task is unsupervised learning, so we have to try some number to see which one is the best fit. To measure the how good the topics are separable from each others, gensim gives us "CoherenceModel" which receive LDA model as input and return coherence score.

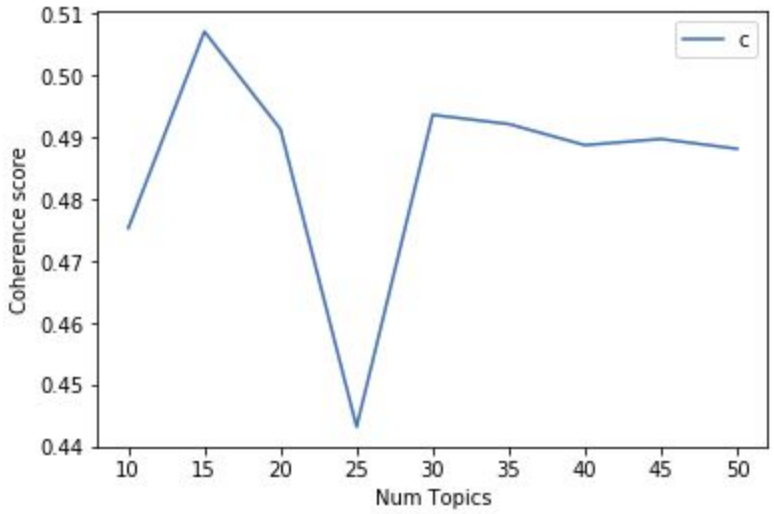

Figure 7: Number of topic corresponding to coherence score

After plotting the number of topics with its coherence score, we can see the coherence has a peak at 15, drop at number 25, however remain stable after 30. We could select 30 as the number of topics for the training. The reason why we don't choose 15 because of to have more topics for the documents but keeping the coherence score at average.

After training, we got the LDA model with 30 topics. Each topic is defined by its keywords and probabilities of contribution. Using pyLDAvis package, we can plot the topics and keywords as shown in the figure x and y.

5.3 Evaluation

To get measure of how good the topics cluster are, we again calculate the coherence and perplexity score from gensim function. The result shows
Perplexity: -10.107144777288838 (lower is better)
Coherence Score: 0.5180880042662807 (higher is better)
The coherence score is appropriately good enough. Perplexity is difficult to judge because there is no limitation on that. In this case it is already small, but we would expect a smaller number.

Figure 8: Topic distribution map

The perplexity is reflexed in the figure x, when many topics are overlapping. However, our purpose is not trying to separate articles clearly, the variant in the result could be a good thing. For example, if someone search for keyword “fruit”, the results can contain the “food” topic or “healthy” topic. So overlapping is acceptable.

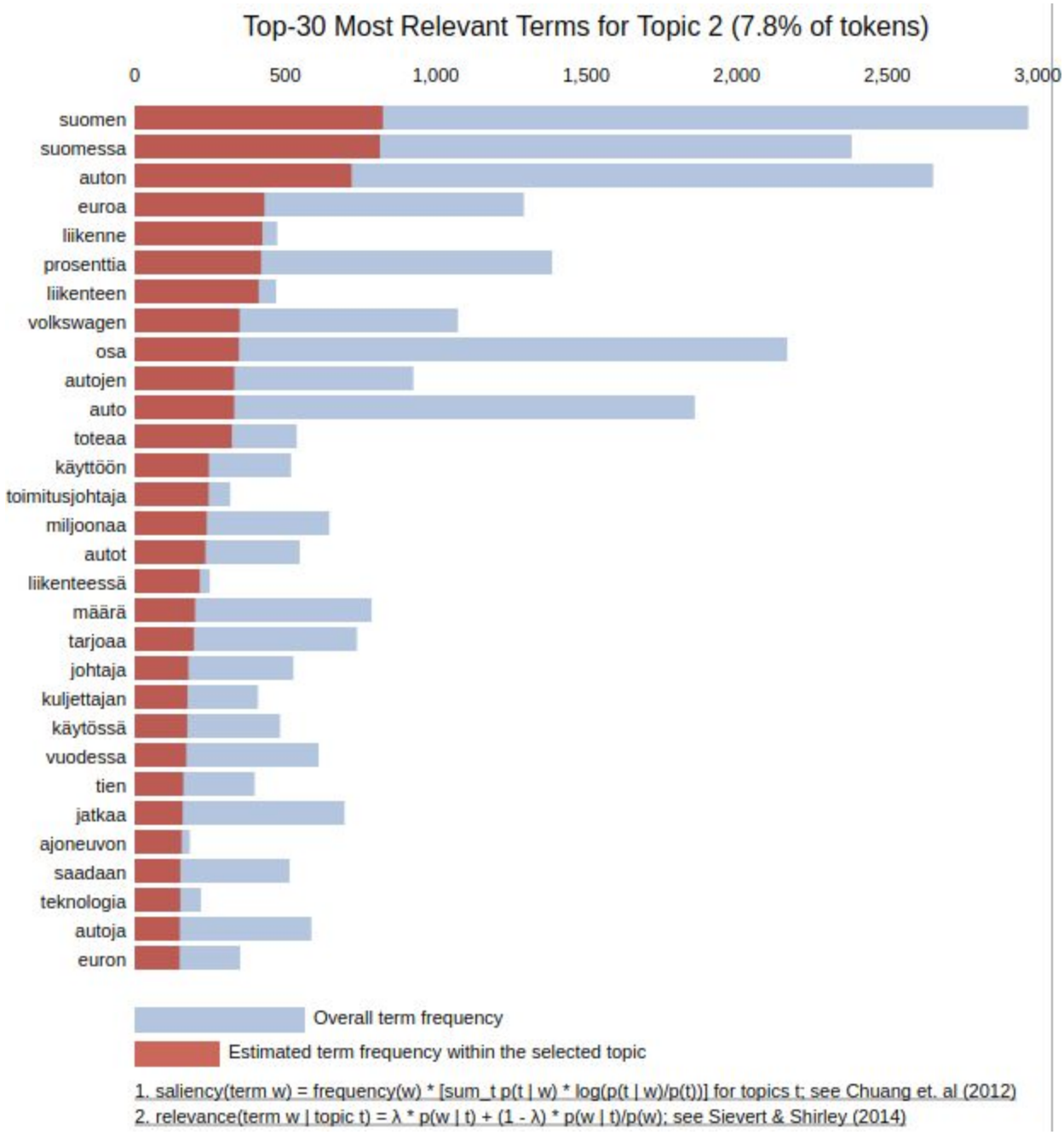

Figure 9: The words contribute to the topic 2

In figure y, we could guess this topic is about car. After training topic model by LDA, we saved the topics and contributed keywords as json format. We also saved the articles with the list of topics probability along to them.

6 Combine Word2vec and LDA

6.1 Dataset

After word2vec model, we can use this for embedding layer in neural network. However, our dataset for word2vec model is too small, we decided to use pre-trained word2vec model from BPEmb [8]. The BPEmb has pre-trained word vectors for 275 languages. The algorithms used to train these models is subword information as described in the “Enriching Word Vectors with Subword Information” paper [7]. The model for Finnish includes 200000 words x 300 dimension.

In the training set, the input for the network created from the topic and its contributed keywords. We only take top 50 keywords for each topic into account. For example, the word “great” has index 729 and the topic number 2 will be formed as “[729]” for input tensor and “[2]” for the gold label tensor. However, due the the format of Finnish language, and we don’t have a good library for lemmatizing, there are 611 missing words in the pre-trained model and only 889 words is available for training.

Making a test set is tricky. We assume the pre-trained word2vec and topic model is good enough in our experiment. Thus, for each keyword in the topic, we find N most similar words from word2vec model and give them the label is the number of topic.

6.2 Training

For the training section, we use a simple feedforward network model with one hidden layer. The input is embedded layer from pre-trained word2vec model and the output is passed through a softmax function to return the probabilities. After training, we can start making prediction for keyword entered, it will return the list of relevant topics which can be used to get the articles.

6.3 Evaluation

Firstly, we set a case, given a keyword if randomly select N topics as a target, what is the probability of correction? Let say *N = 5* and we have 30 topics, the result is *5/30*, around ***16.6%*.**

Now we make a test function for our model which use the same set of parameter N=5. The result showing that the average accuracy for keywords in the testset is ***53.9%*** which is much higher than the random choice.

7 Conclusion and future work

In summary, we have trained Skip-gram negative sampling model which give the better results in overall compared to gensim skip-gram, cbow and fasttext model in English and Finnish. Furthermore, we implemented LDA topic modeling for Finnish docs and use the result as gold label for an supervised learning network. Based on the result, we confirm word embedding is useful to keyword query application. The current work showing the accuracy is 53.9% instead of 16.7% as a random choice. However, this can be improved in the futures with a better dataset.

For the future work, we could improve the Skip-gram model by implementing sub sampling technique and subwords information. For the finnish corpus, we need to find a good lemmatizing method or continuous training on pre-trained word2vec model. By that, there is more words available for supervised training task, thus improve the performance.

8 References

1. T. Mikolov, K. Chen, G. Corrado, and J. Dean.Efficient estimation of word representations in vectorspace. CoRR, abs/1301.3781, 2013
2. Mikolov, Tomas; Sutskever, Ilya; Chen, Kai; Corrado, Greg S.; Dean, Jeff (2013). Distributed representations of words and phrases and their compositionality. Advances in Neural Information Processing Systems.
3. Chris McCormick. Word2Vec Tutorial Part 2 - Negative Sampling http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/
4. Derek Chia. An implementation guide to Word2Vec using NumPy and Google Sheets https://towardsdatascience.com/an-implementation-guide-to-word2vec-using-numpy-and-google-sheets-13445eebd281
5. X. Rong word2vec Parameter Learning Explained. arXiv:1411.2738v4 [cs.CL] 5 Jun 2016
6. DM Blei, AY Ng, MI Jordan. Latent dirichlet allocation - Journal of machine Learning research, 2003
7. P Bojanowski, E Grave, A Joulin, T Mikolov. Enriching word vectors with subword information - TACL 2017
8. https://nlp.h-its.org/bpemb/
9. https://radimrehurek.com/gensim/