Word2vec implementation for Russian text

EDHC 2018, Tartu

Olha Kaminska
olha.kaminska@ut.ee
University of Tartu

Outline

1. **Introduction**
   i. Abstract
   ii. Tasks
   iii. Theory
   iv. Train data
2. **Algorithm**
   i. Corpora
   ii. Lemmatization
   iii. Encoding
   iv. Neural Networks
   v. Weights
3. **Test**
   i. Test data
   ii. Classification task description
4. **Summary**
   i. Conclusion
   ii. Further work
   iii. References
   iv. Acknowledgements

INTRODUCTION

Abstract

- Machine Learning algorithms work with **numbers, not letters**, so developing a correct system for converting words to numbers is an important and relevant task.
- In this work the implementation of the algorithm is considered for converting words into vectors, called Word2Vec. The implemented algorithm turns the given **word into a 300-dimensional vector**, such that the words close in meaning have close coordinates.
- The algorithm is implemented for the Russian text, because, unlike for English, for this language exist **not so many solutions** in the NLP field.
- In general, the described algorithm **can be applied to other languages**, which can provide enough input text data for processing and vocabulary building.

Tasks

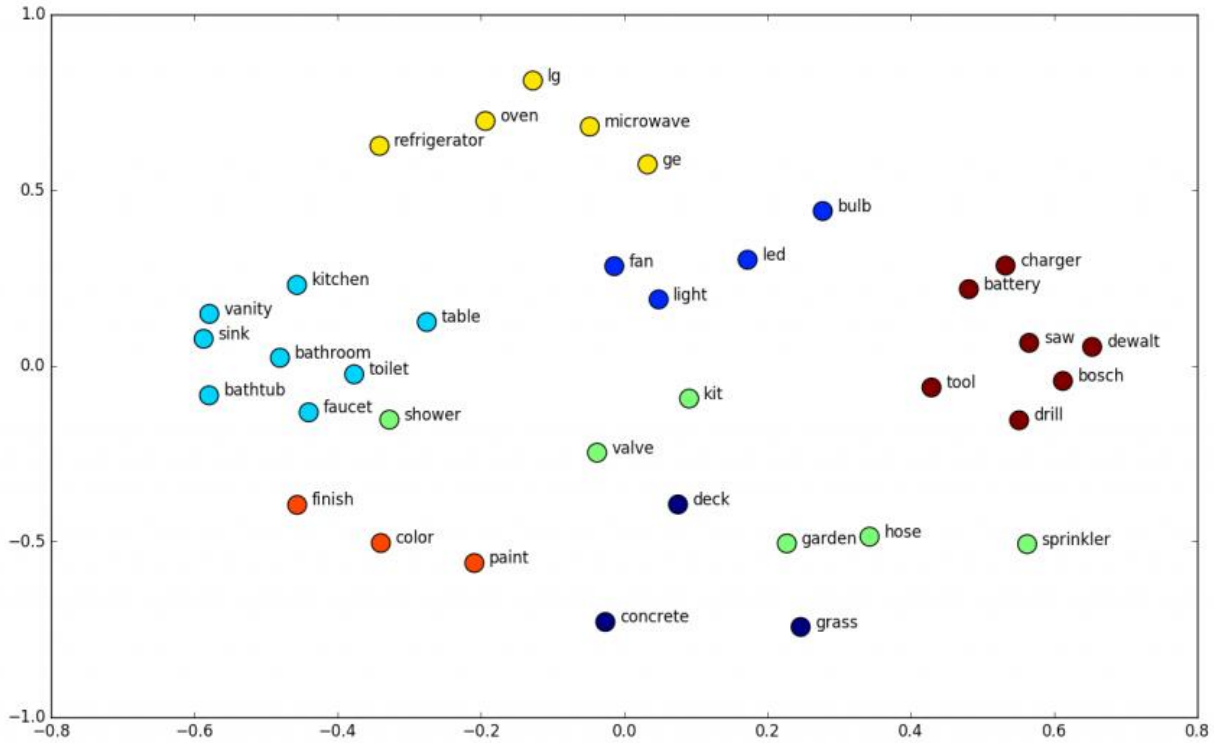
- Collect Russian text data.
- Implement Word2Vec algorithm on obtained data.
- Test obtained results.

Theory

- **Word2vec** is a group of related models that are used to produce word embedding.
- Word2vec takes as its **input** a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space.
- Word vectors are positioned in the vector space such that words that share **common contexts** in the corpus are located in close proximity to one another in the space.

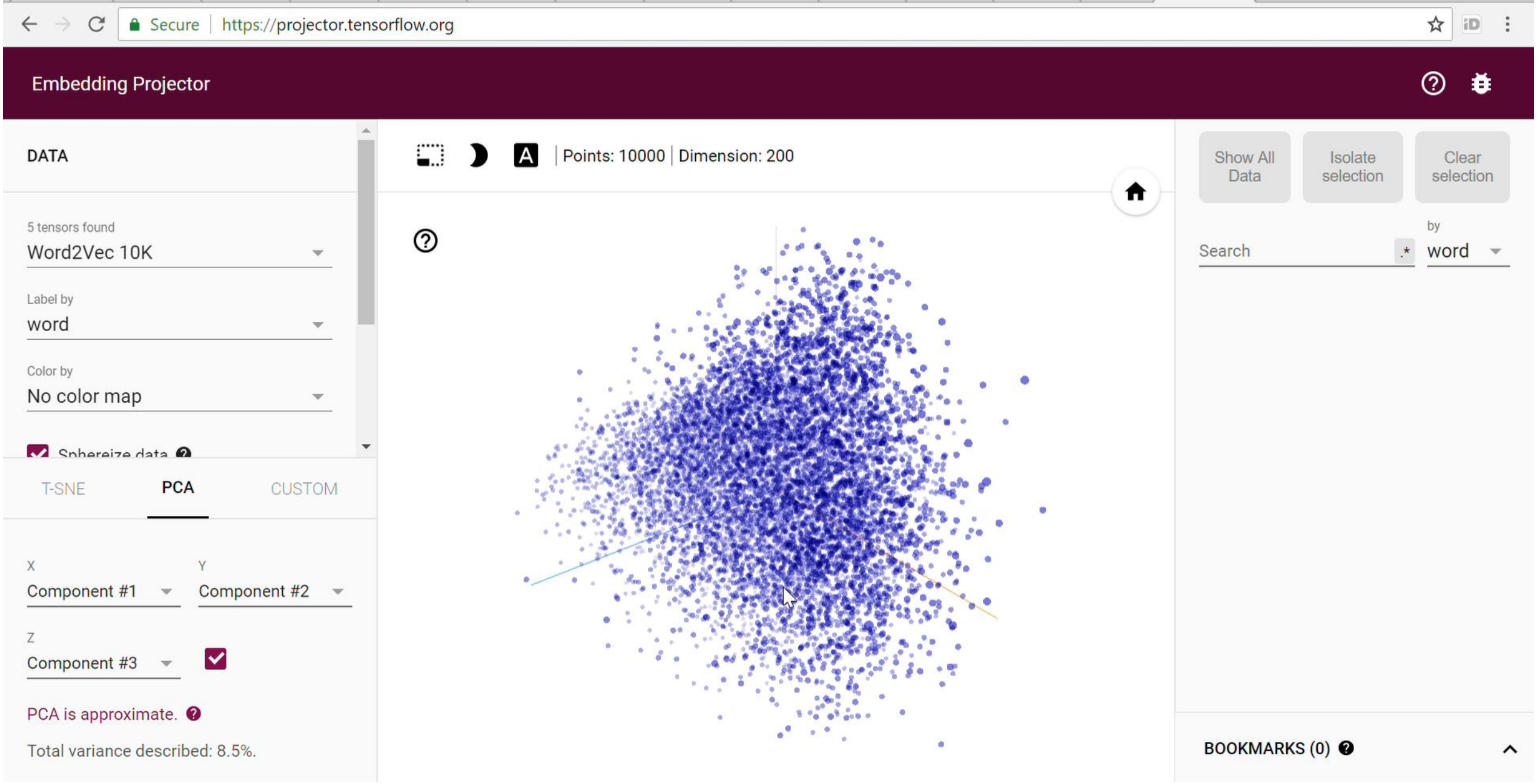
Examples

2D word embedding space



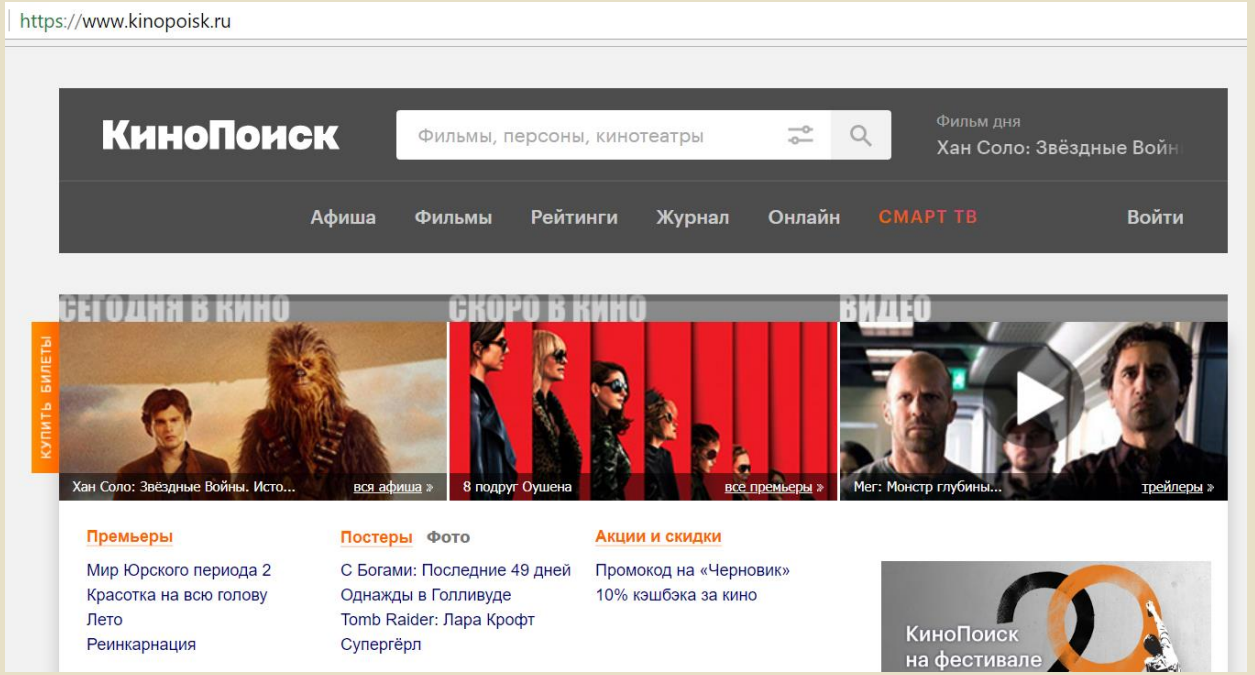
Software for word embedding

1. **Word2vec** (by Google)
2. **GloVe** (by Stanford University)
3. **fastText** (by Facebook)
4. **Gensim** (Python library)


Secure
https://projector.tensorflow.org
Embedding Projector
DATA
5 tensors found
Word2Vec 10K
Label by
word
Color by
No color map
Sphereize data
T-SNE
PCA
CUSTOM
X
Component #1
Y
Component #2
Z
Component #3
PCA is approximate.
Total variance described: 8.5%.
Points: 10000 | Dimension: 200
Show All Data
Isolate selection
Clear selection
Search
.*
by
word
BOOKMARKS (0)

Data

- "**KinoPoisk.ru**" webpage
- From reviews to top-250 movies – collected **positive reviews**
- From reviews to worse-250 movies – collected **negative reviews**
- Saved as text files



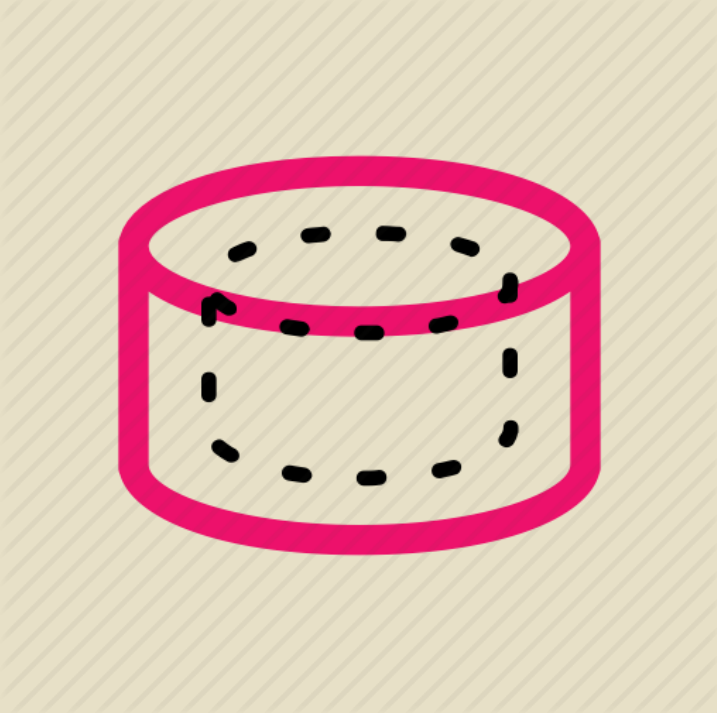
ALGORITHM

Steps

1. Gather **corpora** of Russian text
2. **Lemmatize** words from corpora
3. Encode each lemma as **one-hot vector**
4. Train **Neural Network** to predict words from the context
5. From NN take the **transition weights** from the input layer to the hidden layer, this will be **embedding**

1. Corpora

- Take Russian reviews **data**.
- **Preprocess** data:
  - Delete punctuation;
  - Delete numbers;
  - Delete special symbols;
  - Lowercase letters.



2. Lemmatization

Definition

- **Lemmatization** is the algorithmic process of determining the lemma of a word based on its intended meaning.
- Lemmatization depends on correctly identifying the intended part of speech and meaning of a **word in a sentence**, as well as within the larger context surrounding that sentence, such as neighboring sentences or even an entire document.

How was used

- Take **lemmas** from preprocessed data.
- For Russian text use library "**pymorphy2**".

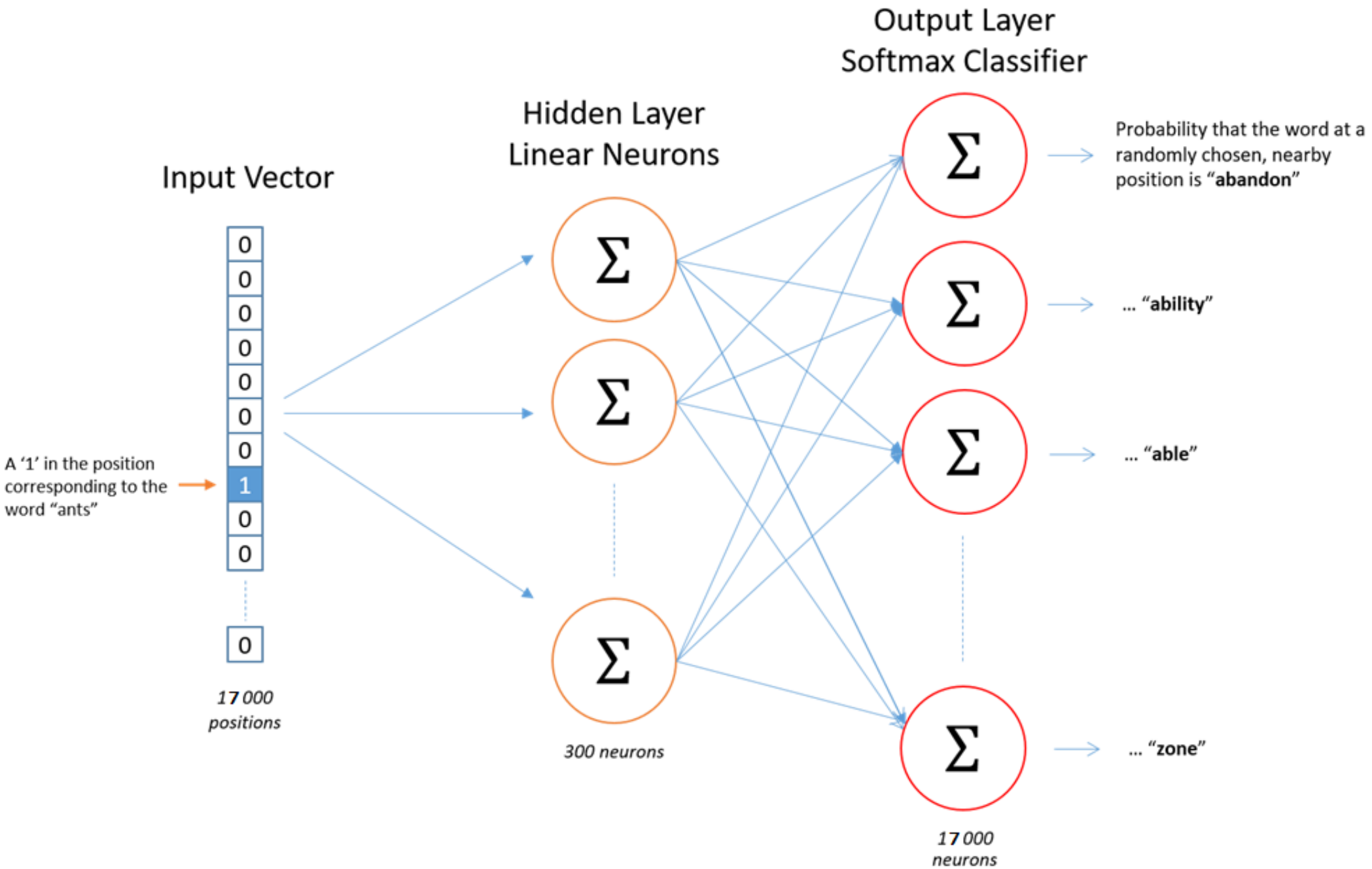
3. Encoding

- Encode each lemma as **one-hot vector.**
  - A one-hot encoding is a representation of categorical variables as binary vectors.
- Obtain **matrix**, with size of rows equal to number of unique lemmas and columns – number of all words in dataset.

4. Neural Network

- Train the Neural Network **to predict words from the context**, considering the several previous and following words.
- There are as many **input neurons** as there are unique lemmas.
- One-hot vector is fed to **input**.
- In the **hidden layer** there are so many neurons, as we want to have dimensionality for embedding.

- **Spoiler:** 300 dimensional vectors.


Output Layer
Softmax Classifier
Hidden Layer
Linear Neurons
Input Vector
Probability that the word at a randomly chosen, nearby position is "abandon"
... "ability"
... "able"
... "zone"
A '1' in the position corresponding to the word "ants"
0
0
0
0
0
0
0
1
0
0
0
17 000 positions
300 neurons
17 000 neurons

5. Weights

- From NN take the **transition weights** from the input layer to the hidden layer.
- This is **embedding** of input words.

```
In [117]: word2vec('разделять')

Out[117]: array([-0.00639411, -0.04588562,  0.0369455 ,  0.03193153,  0.0159332 ,
                  0.00820546, -0.02958933, -0.00914028, -0.03050431,  0.01597221,
                  0.04575444, -0.00564439,  0.0391799 ,  0.02425341,  0.03255101,
                  0.00367632,  0.00501155,  0.01181419, -0.01500569,  0.04380527,
                 -0.03364797,  0.0233223 , -0.02690406, -0.02575034,  0.04332725,
                 -0.03481043,  0.00775111, -0.01910125,  0.00395717,  0.03562836,
                 -0.0340369 ,  0.00250802, -0.04525683,  0.00473332, -0.0064028 ,
                  0.00835174,  0.04644747,  0.00252277, -0.02215388,  0.04157735,
                  0.03594367,  0.00420265,  0.00365066,  0.04478629,  0.03465109,
                 -0.00748581,  0.01434831, -0.03768604,  0.02885463,  0.04136123,
                 -0.00938225, -0.02604549, -0.02458672,  0.00798132,  0.02697713,
                 -0.02496132,  0.03379746,  0.01614414,  0.03414029,  0.04203514,
                 -0.02488135,  0.00391269,  0.01361063, -0.00125268,  0.0096677 ,
                 -0.033234  , -0.03777047,  0.01222209,  0.03590537, -0.04351985,
                  0.03340068, -0.03317507,  0.04392919, -0.02210417,  0.03605293,
                  0.03787395,  0.00215704, -0.02471151, -0.03443055, -0.03110588,
                  0.02536169,  0.03612708,  0.02934559, -0.0138054 , -0.02079163,
                 -0.01079961, -0.02632268, -0.02710073,  0.03670942,  0.02447869,
                 -0.03197314, -0.02857919, -0.01888352, -0.04077089, -0.04441738,
                  0.01260424,  0.00739974, -0.00906618,  0.00228496,  0.02333153,
                 -0.04668694,  0.01110164, -0.01197695,  0.01885737,  0.01484401,
                 -0.02616754,  0.03828255, -0.0257233 ,  0.03302716, -0.02413914,
                 -0.01160299, -0.03155842,  0.0091939 , -0.00830918,  0.00849971,
                  0.03284189,  0.01926753, -0.00436024, -0.00612164, -0.04172376,
                 -0.03947359,  0.03658891, -0.04036014, -0.00943488, -0.01623476,
                  0.02539685, -0.00507327,  0.02575494, -0.03100345, -0.02568106,
                  0.000123  ,  0.00656615,  0.0006848 , -0.01559029, -0.03055143,
                  0.00730261, -0.04352391,  0.0392156 , -0.00276283, -0.01000359,
                  0.00387604,  0.03028967, -0.00258934, -0.00610403,  0.02041425,
                  0.03307925,  0.02209498, -0.00054471, -0.0147761 ,  0.02542557,
                  0.0002073 , -0.01024033, -0.03745475,  0.04039321, -0.03179872,
                 -0.00854158,  0.02731036,  0.04003964,  0.04188787, -0.03920196,
                  0.03671835,  0.01075296,  0.01271475, -0.00200171, -0.02287095,
                  0.0141558 , -0.02906163,  0.03200147,  0.03585149, -0.0279744 ,
                  0.00107983, -0.00406467, -0.0259136 , -0.01715655,  0.03689127,
                  0.04382261,  0.04091699, -0.03605505, -0.01729221, -0.00817823,
                  0.01880536,  0.00102464,  0.03631086, -0.0233483 ,  0.0232889 ,
                  0.01193541,  0.03636067,  0.00774051, -0.0146293 , -0.01894907,
                  0.03996055,  0.02045731,  0.02348446, -0.04191395,  0.01920758,
                 -0.01000123, -0.02458888, -0.03473673,  0.02174515, -0.02704981,
                  0.03334304,  0.03390664, -0.03598229,  0.0183765 , -0.02050221,
                  0.03323628, -0.02625937, -0.00030264, -0.03888242, -0.00465083,
                  0.02688103,  0.02550639, -0.03450549, -0.02571611,  0.01002207,
                  0.02529385,  0.03346233, -0.02259541,  0.02034368,  0.04613729,
                  0.0198002 ,  0.01173888, -0.04225231,  0.03102531,  0.03782723,
                 -0.00669488, -0.01442955, -0.02713644,  0.04186819, -0.02459098,
                  0.0461633 ,  0.03263813, -0.00840959,  0.03133181, -0.01732666,
                 -0.00613568,  0.02479468, -0.01206202,  0.036767  , -0.02026358,
                 -0.02448322,  0.03520351, -0.01520187,  0.0001885 ,  0.02405567,
                  0.04200333,  0.04547734, -0.01746451,  0.02202057, -0.04177645,
                  0.01929124, -0.03189236, -0.02967607, -0.02467472, -0.0056316 ,
                 -0.01846931,  0.01992805, -0.04041422, -0.01565258,  0.03517888,
                 -0.00715944,  0.00167413, -0.03111748,  0.03131553,  0.03298446,
                  0.02644238, -0.01776054, -0.01509872, -0.02807837, -0.00119478,
                 -0.03365725, -0.02628119,  0.04075695,  0.02126157,  0.03677196,
                 -0.00529721,  0.03221723, -0.01531554, -0.03660807, -0.01496948,
                  0.00281821, -0.01283264, -0.03264076, -0.04187772, -0.01793291,
                  0.04386906,  0.04135433, -0.01557107,  0.02048864, -0.01846153,
                  0.02122997, -0.00450943,  0.03074425,  0.01091178,  0.03923343,
                  0.01989278,  0.00319333,  0.02504788,  0.03942253, -0.03279759], dtype=float32)
```

TEST

Using in classification task

- **Task:** classify review as positive or negative.
- **Train data:** 90% of positive reviews + 90% of negative reviews.
- **Test data:** 10% of positive reviews + 10% of negative reviews.
- **Model:** Logistic Regression.
- **Test set accuracy:** 70%*****

***** Logistic Regression with Google Word2Vec for the same amount of English reviews gave 76% test set accuracy.

SUMMARY

Conclusion

- This algorithm turns word into **a 300-dimensional vector**. Main idea of this 300-dimensional space, is that words which are close in meaning, have close coordinates in it.
- Provided Word2Vec algorithm can be applied to **any language**, with appropriate size of train data.
- Using obtained outputs of this algorithm gave **high enough accuracy** in classification task, even comparing with the best approach for English language in the similar task.

Further work

- **Increase** size of obtained corpora, use more different sources.
- Find more approaches for corpora **preprocessing**.
- Try other approaches, for example, Bag of Words.
- Optimize model for big amount of data.
- Try to implement this algorithm for **Ukrainian language**, that is close, but less popular, and compare obtained results for classification tasks.

References

1. McCormick, C. (2016, April 19).
   Word2Vec Tutorial - The Skip-Gram Model.
   Retrieved from http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

2. **Lecture 2 | Word Vector Representations: word2vec**
   by Stanford University School of Engineering YouTube channel.
   Retrieved from https://www.youtube.com/watch?v=ERibwqs9p38

Acknowledgements

I would like to express my very great appreciation to

- Supervisor Benson Muite (benson.muite@ut.ee)
- Neural Network expert Viacheslav Komisarenko (viacheslav.komisarenko@ut.ee)
- Natural Language Processing expert Elizaveta Korotkova (elizaveta.korotkova@gmail.com).

THANK YOU FOR ATTENTION!

QUESTIONS?