Team_id: 16
Team name: Quattroporte
Name: Nagima Zhailau,
Shynggyskhan Turganbekov,
Fariza Zholdassova,
Yerkin Serik
Student_id: 201107043, 201107042, 201107006, 201107068

Report

This is the final report of our group Quattroporte. Here we will explain about our group project 2. Main idea of our project is to read and understand a research paper, and implement them. With the group's decision,we chose paper - “GloVe: Global Vectors for Word Representation”.

Firstly, we made a meeting(call) and divided papers that were given to us to implement. After this we made a second call and we selected a text paper with the thesis name “GloVe: Global Vectors for Word Representation”. In this project we want to represent how we can perform text classification using global vectors. So we use Keras to classify text with the help of GloVe Word Embeddings. The main intuition underlying the model is the simple observation that ratios of word-word co-occurrence probabilities have the potential for encoding some form of meaning.The training objective of GloVe is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence. And also divided tasks between us.

Team members: Nagima Zhailau - Managing reports, Monitoring progress
Shynggyskhan Turganbekov - Planning,organizing team meetings,Tester (Finding any kind of errors)
Fariza Zholdassova - Providing with ideas and way, Read Me of github
Yerkin Serik - Implement tasks, writing code.

There is main tasks are:

1. Select paper - “GloVe: Global Vectors for Word Representation”
2. Read and understand, and also discuss it with the team.
3. Collecting Data
4. Storing data in csv or excel
5. Downloading requests library
6. Data preprocessing in Python

7. Creating corpus code
8. Split the text and clear data(remove urls and emoji)

The main idea of this paper word embeddings with glove vectors, by using pre-trained word vectors we will get a lot better results. Why should we use a glove? Glove uses the frequency of co-occurrences to their context. Glove builds word embedding in a way that a combination of word vectors relates directly to the probability of these words' co-occurrence in the corpus. There we can give an example for word embeddings like "king is to queen as man is to woman" should be encoded in the vector space by the vector equation king − queen = man − woman. This evaluation scheme favors models that produce dimensions of meaning, thereby capturing the multi-clustering idea of distributed representations.

Words are usually considered as the smallest meaningful units of speech or writing in human languages. High-level structures in a language, such as phrases and sentences, are further composed of words. For human beings, to understand a language, it is crucial to understand the meanings of words. Therefore, it is essential to accurately represent words, which could help models better understand, categorize, or generate text in NLP tasks.

A word can be naturally represented as a sequence of several characters. However, it is very inefficient and ineffective only to use raw character sequences to represent words. First, the variable lengths of words make it hard to be processed and used in machine learning methods. Moreover, it is very sparse, because only a tiny proportion of arrangements are meaningful. For example, English words are usually character sequences which are composed of 1–20 characters in the English alphabet, but most of these character sequences such as "aaaaa" are meaningless.

GloVe(Global Vectors) is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

This work has analyzed the model properties necessary to produce linear directions of meaning and argue that global log-bilinear regression models are appropriate for doing so. They propose a specific weighted least squares model that trains on global word-word co-occurrence counts and thus makes efficient use of statistics. The model produces a word vector space with meaningful substructure, as evidenced by its state-of-the-art performance of 75% accuracy on the word analogy dataset. They also demonstrate that our methods outperform other current methods on several word similarity tasks, and also on a common named entity recognition (NER) benchmark.

In this paper also represented such algorithms like Matrix Factorization Methods and Shallow Window-Based Methods. Matrix factorization methods for generating low-dimensional word representations have roots stretching as far back as LSA. Another approach is to learn word representations that aid in making predictions within local context windows. The GloVe model performs significantly better than the other baselines, often with smaller vector sizes and smaller corpora. Paper results using the word2vec tool are somewhat better than most of the previously published results. This is due to a number of factors, including our choice to use negative sampling (which typically works better than the hierarchical softmax), the number of negative samples, and the choice of the corpus.

Implementation of this paper

Let's discuss the implementation of our project. Firstly, after reading and understanding the paper we realized that we will work with a huge amount of words and a big dataset. With this data we will teach the program to recognize words from text. So that the program can recognize words, we did the following:

1. After collecting the data will we have file with a text
2. Downloading requests library
3. Clear data(remove urls and emoji)
4. Number each word
5. Get text as arrays
6. We take a trained Glove dataset. The GloVe model is trained on non-zero entries in the global word-word match matrix, which indicates how often words occur with each other in a given corpus.

With the help of numerous runs, the matrices of words of the tested text are filled with numbers. As a result, we get a dataset with matrices of numbers for each word and with the help of them we can find out the connections of words in this text.
Collected data should be restored as numerical array. We indexed every unique word and get vocabular size of our text. With this array programm will work further. We indexed words in code part called Corpus.

```
from nltk.tokenize import word_tokenize
import nltk


def create_corpus_tk(df):
    corpus = []
    for text in train["text"]:
        words = [word.lower() for word in word_tokenize(text)]
        corpus.append(words)
    return corpus
```

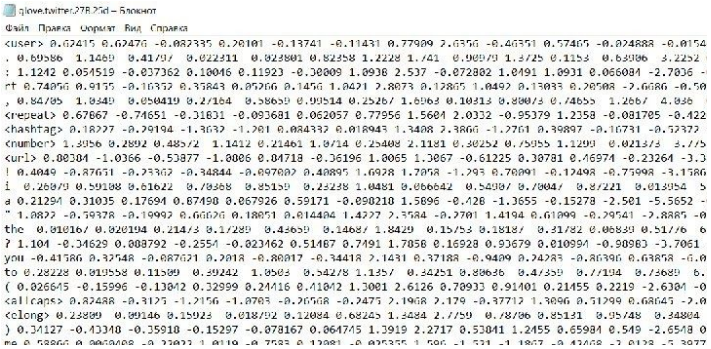
```
train.text
```

```
]:  0                       nursultan capital city kazakhstan
    1      known   astana akmola renamed nursultan march ...
    2      city lies banks ishim river northcentral part ...
    3       official estimate reported population  within...
    4      akmola became capital kazakhstan  since grown ...
                                ...
    671     nursultan attracted three trillion tenge us b...
    672    growth achieved due large number construction ...
    673
    674    tourism becomes one factors drive economic gro...
    675    nursultan among top ten attractive tourist cit...
    Name: text, Length: 676, dtype: object
```

```
[[1, 3, 2, 4],
 [79, 11, 7, 5, 1, 10, 80, 81, 6, 16, 1, 17],
 [2, 82, 83, 84, 85, 86, 87, 4, 34, 7, 35, 88, 89, 2, 19, 36, 90, 91, 35],
 [92, 93, 94, 95, 34, 2, 96, 97, 98, 2, 99, 42, 43, 3],
 [7, 18, 3, 4, 44, 100, 101, 102, 45, 37, 20, 46, 47],
 [10, 38, 103, 104, 105, 39, 2, 5, 1, 106, 4, 16, 1, 17],
 [37, 1, 40, 2, 38, 107, 40, 108],
 [18, 3, 4, 2, 109, 110, 111],
 [12, 112, 113, 114, 115, 116, 117],
 [118, 41, 4, 1, 119, 39, 120, 121, 122, 123, 124, 125, 126, 48, 41, 127, 128]
 [129, 130, 131, 132, 133, 134, 135],
 [136, 137, 75, 138, 76],
 [21, 139, 49, 50, 140, 141, 51, 142, 143, 144, 2, 145, 146, 147, 148],
 [149, 150, 36, 5, 151],
 [152, 153, 10, 5, 52, 33, 154],
 [2, 53, 54, 155, 156, 157, 158, 22, 53, 54, 159],
 [6, 160, 52, 161, 162, 163, 21, 7],
 [164, 7, 165, 42, 3, 4],
 [166, 5, 11, 51, 3, 2, 6],
 [10, 3, 5, 11, 167, 21, 1, 168, 169, 6, 16, 1, 17, 170, 171],
```

Comparing of text and its numerical form

We are going to use a GloVe pre trained corpus model to represent our words, trained on Twitter data (27B tokens). With trained 25 dimension glove vectors. Size of .csv file 250MB. We have collected data from Wikipedia as text format and .csv file. Every word in dataset provided as numerical array with 25 dimension.



There you can see a picture of glove.twitter data as an array.

After that, we get all needed information, so next part is training the word representation. We used Keras interface, which often uses for training neural networks, but its also can train our code, we just need several time pass through both arrays: bigger dataset 27B 25D and our simpler dataset with over 500 word arrays. By using keras methods and functions we create model for training, there we enter vocabular size of our training text and dimension of word array, we set trainable=False because we already have pretrained big dataset(glove.twitter.27B.25d.txt)

From result of run this part we can know that first pass often going slower than next and with lower accuracy, also tools bit more pass time. Other 9 passes similar to each other, everywhere high accuracy and similar pass time. It cause first time programm checking every relation and due to its non dynamic data further calculations based on first one.

```
Epoch 1/10
16/16 [==============================] - 4s 95ms/step - loss: 0.1713 - accuracy: 0.9620 - val_loss: 6.2505e-06 - val_accuracy:
1.0000
Epoch 2/10
16/16 [==============================] - 0s 12ms/step - loss: 4.7296e-06 - accuracy: 1.0000 - val_loss: 2.5167e-06 - val_accura
cy: 1.0000
Epoch 3/10
16/16 [==============================] - 0s 19ms/step - loss: 2.3361e-06 - accuracy: 1.0000 - val_loss: 2.0341e-06 - val_accura
cy: 1.0000
Epoch 4/10
16/16 [==============================] - 0s 10ms/step - loss: 1.9919e-06 - accuracy: 1.0000 - val_loss: 1.9166e-06 - val_accura
cy: 1.0000
Epoch 5/10
16/16 [==============================] - 0s 9ms/step - loss: 1.8937e-06 - accuracy: 1.0000 - val_loss: 1.8664e-06 - val_accurac
y: 1.0000
Epoch 6/10
16/16 [==============================] - 0s 9ms/step - loss: 1.8466e-06 - accuracy: 1.0000 - val_loss: 1.8287e-06 - val_accurac
y: 1.0000
Epoch 7/10
16/16 [==============================] - 0s 10ms/step - loss: 1.8259e-06 - accuracy: 1.0000 - val_loss: 1.7924e-06 - val_accura
cy: 1.0000
Epoch 8/10
16/16 [==============================] - 0s 10ms/step - loss: 1.7700e-06 - accuracy: 1.0000 - val_loss: 1.7558e-06 - val_accura
cy: 1.0000
Epoch 9/10
16/16 [==============================] - 0s 9ms/step - loss: 1.7455e-06 - accuracy: 1.0000 - val_loss: 1.7186e-06 - val_accurac
y: 1.0000
Epoch 10/10
16/16 [==============================] - 0s 9ms/step - loss: 1.7007e-06 - accuracy: 1.0000 - val_loss: 1.6812e-06 - val_accurac
y: 1.0000
```

So in result we have array which contains indexes of words with he highest accuracy ratio

```
padded
```

```
array([[  4,   4,  99, 176,  42,  43,  85,   0,   0,   0],
       [306, 288,  87,   0,   0,   0,   0,   0,   0,   0],
       [  4,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  4, 265,  99,  99, 265,   0,   0,   0,   0,   0],
       [ 95, 227,  40,  95, 265,  70, 209,   0,   0,   0],
       [ 39,   3,  79,  10,   0,   0,   0,   0,   0,   0],
       [ 37,  99,   2,   0,   0,   0,   0,   0,   0,   0]])
```

When we decoded those array against to words we can see that programm finished task simply correctly. Of course due to not big dataset size some words repeating

```
kazakhstan kazakhstan country mainly central asia river
also large part
kazakhstan
kazakhstan world country country world
population million one population world people per
since capital known astana
almaty country city
```

In process implementation of this work, there were problems. First problem was related to collecting data, so we collected data by ourself in wikipedia. Second was related to .csv file was not readable, so we use UTF-8 encoding, after this problem was solved.

Conclusion

This kind of experience was amazing and helpful for us. We have learned how to perform text classification using global vectors. Extracting structured data in Wikipedia, and using Global Vectors for Word Representation. In this work we argue that the two classes of methods are not dramatically different at a fundamental level since they both probe the underlying co-occurrence statistics of the corpus, but the efficiency with which the count-based methods capture global statistics can be advantageous.

The result, GloVe, is a new global log-bilinear regression model for the unsupervised learning of word representations that outperforms other models on word analogy, word similarity, and named entity recognition tasks**.**

Finally, we decide that the GloVe model performs significantly better than the other baselines, often with smaller vector sizes and smaller corpora.

References

[1] https://nlp.stanford.edu/projects/glove/

[2] Eric H. Huang, Richard Socher, Christopher D. Manning, and Andrew Y. Ng. 2012. Improving Word Representations via Global Context and Multiple Word Prototypes. In ACL.

[3] https://jhui.github.io/2018/02/11/Keras-tutorial/

[4] https://stackabuse.com/text-classification-with-python-and-scikit-learn/

[5]https://stackoverflow.com/questions/59866359/unicodedecodeerror-charmap-codec-cant-decode-byte-0x98-in-position-668-char

[6]https://www.kite.com/python/answers/how-to-split-text-into-sentences-in-python#:~:text=Use%20sent_tokenize()%20to%20split%20text%20into%20sentences&text=download(module)%20with%20%22punkt,into%20a%20list%20of%20sentences

Links

Github