UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Facultat d'Informàtica de Barcelona

Word Embeddings and Topic Detection for Contextual LSTM NLP tasks: Part I

Authors:
Pol Alvarez Vecino

May 24th 2017, Barcelona

Contents

1 Introduction

The basic objective of this project is to prepare a dataset to train a Contextual Long-Short Term Memory (CLSTM) neural network as described by Ghost et al. [1]. In order to be used with CLSTM, the dataset needs the words' embeddings and be able to produce a topic/context for a subset words. To give more insight to the decisions made throughout this deliverable, the work required to actually train the CLSTM network has been divided into the following objectives:

1. **Preprocess dataset for word embeddings creation**
2. **Train the word embeddings**
3. **Use word embeddings IDs to transform wikipedia articles into lists of embeddings' indices**
4. Train a LSTM network for perform word prediction (and act as a result baseline)
5. Find a way to provide the some kind of context the CLSTM
6. Develop an efficient mechanism for querying the context during network training
7. Train CLSTM network for word prediction

This assignment will address the first three goals only. However, it will explore the embeddings obtained to make sure the training done produces good and sensible word representations.

To evaluate the quality we will use t-distributed Stochastic Neighbor Embedding (t-SNE) [2] to visualize some subsets of the embeddings.

We will also use embeddings' distance measures to get the most similar items to a random subset of words. The most similar/closer words to a given one should have an strong relation with it (whether it is semantic, syntactic or even grammatical).

Finally, we will also use the semantic and grammatical relationships accuracy metric introduced by Mikolov et al. [3] when they published the CBOW and Skip-gram models (see Subsection **??**) to see how good they perform and have a reference metric when comparing different embedding's size.

2 Code structure

Processing and training the required models large datasets requires a lot of time. If all tasks are performed by a single application, any error or unexpected problem can waste lots of computation hours. In order to minimize these risks while enabling fast prototyping and development iterations, the whole task at hand was divided into smaller blocks.

Blocks are structured in such a way that can be called as a function inside a pipeline defined by an orchestration file or as a standalone program. For each block, its results are stored to disk (even when they are in a pipeline) in order to be able to resume executions from the last failed block or to modularly modify and test each functionality.

Figure 1 shows the execution flow and generated data defined by the orchestrator file *preprocess.py*.

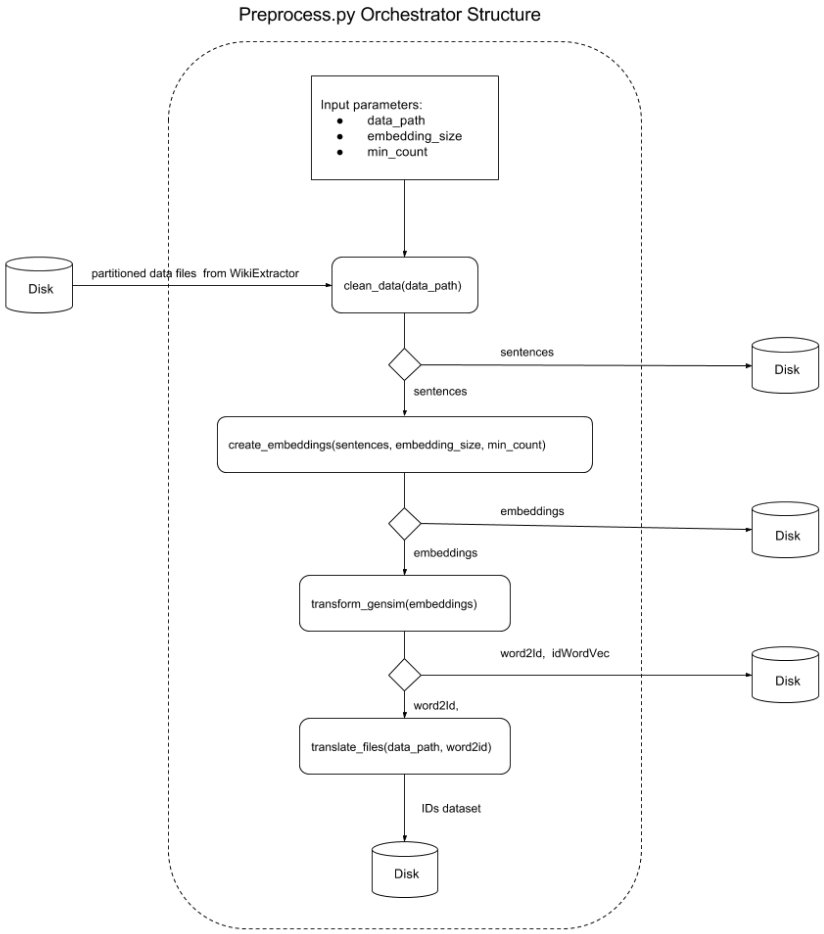

Figure 1: Structure of the processing pipeline.

.

Unfortunately, due to the models and input data sizes, the code attached to this report can not be run (only the embeddings with size 200 are already 231 Mb) unless a proper Wikipedia dump is download into the */data* folder. In that case, executables under *./bin* can be used to run the wikiExtractor (*wiki_extractor_launch*), the whole processing pipelin (*preprocess*), and the semantic checks, accuracies, and plot (*plot_tsne_and_accuracies*. The only requirement is to change their paths to point to the downloaded data. To launch each component independently, change dir to *./src* and issue either *python src/preprocess/[embeddings.py—transform_from_gensim.py]* or *python src/postprocess/[tsne.py—semantics_check.py]* with their according parameters (all blocks have built-in command line help dialogs).

I am also available for a live demo which is the easiest way to explore the embeddings results.

2.1 Extraction

Associated files:

· **WikiExtractor,** python script that extracts and cleans Wikipedia dumps.

Prior to processing the wikipedia dump must be extracted. The data dump contains many XML tags and other useless (for our purposes) data that we would like to strip. Also, it is desirable to split the dataset into smaller files in order to parallelize the computation of next steps.

WikiExtractor project [1] is designed to do exactly that. It takes a wikipedia dump as input, cleans it of XML tags and links, and partitions it into 100 Mb files containing many articles. It is worth noting that it does not process the actual text in any way. Thanks to WikiExtractor the resulting files' size is 12 GB (which is less than the compressed dump file) instead of the 57 GB of the uncompressed data (see Dataset 3 for more details).

2.2 Preprocessing

Associated files:

· **embeddings.py,** handles the preprocessing of the "wikiExtracted" data and creates the embeddings with it.

Once the data is clean and in text format, the next step is to parse it into words. To choose the format I took into account two requirements: train the word embeddings with the resulting data and being able to feed the data into a neural network implemented in TensorFlow.

For the network training, we need to keep the article-paragraph-sentence structure. After evaluating many alternatives, the chosen format was a 4D list structure. For each file (which contains multiple articles), the outermost list indices is the documents, next list is the paragraphs, then sentences, and finally words.

The processing time of this section in sequential mode was roughly 22 hours. Using python's multiprocessing module [2], it was parallelized to use four times the number of available CPUs of the running computer thus allowing to switch context when threads are blocked in I/O (which is the most part of the time due to the read/write performed in each thread). The parallelization version took only 45 minutes using 16 CPUs.

As a side note, multiprocessing's Pool was not able to manage memory efficiently so it crashed after some files were processed (with 120 GB available RAM memory). Simple *Processes* were used and their memory unallocated for each file.

With the data divided into lists, the next step is to train the word embeddings.

2.3 Embeddings

Associated files:

· **embeddings.py,** handles the preprocessing of the "wikiExtracted" data and creates the embeddings with it

Word embeddings are a group of NLP techniques that map words to numerical vectors. They can be used for language modeling, feature learning, and dimensionality reduction. There are many methods to construct this mappings but we used the Word2Vec implementation of *gensim* python package [3].

Word2Vec algorithm [3] trains a neural network and uses the weight of each word as embedding. Depending on the model, the task is different. For Continuous Bag-of-Words (CBOW), the task is to predict the current word based on the context. When using Continuous Skip-gram architecture, shown in **??**, the goal is to predict the surrounding words of the current word. *Gensim* implementation is based on [3] with the improvements introduced by [4] which obtains better performance and more regular representations with sub-sampling techniques and Negative Sampling.

This project uses the Skip-gram model because general opinion is that it usually works better with big datasets and Prof. Lluïs Padró recommended it over the CBOW model.

[1] https://github.com/attardi/wikiextractor
[2] https://docs.python.org/2/library/multiprocessing.html
[3] https://radimrehurek.com/gensim/models/word2vec.html

The method used takes a list a sentences because word embeddings are done at sentence level (i.e. the sliding window starts and ends in each sentence). In order to train it, three parameters need to be set:

- Embedding size, number of weights per word during the training (i.e. final size of the word embeddings)
- Min. word occurrences, number of minimum occurrences per word to be included in the Skip-gram training.
- Window size, maximum distance between the current word and the word to be predicted.

The bigger the embeddings are the better because they have more expressive power. However, too large sizes will lead to long training times. The actual sizes is often decided empirically. For this project, three embeddings were computed with size 200, 500, and 1000.

The minimum occurrences per word was set to 200 because it is the value used by Ghosh et al. [1]. The window value was set to 10 (again this parameter is tuned empirically and general opinions regard 10 as an expressive enough value for posterior deep learning).

The embeddings training time was 3.5 hours when using vectors of length 200 and all the available CPUs.

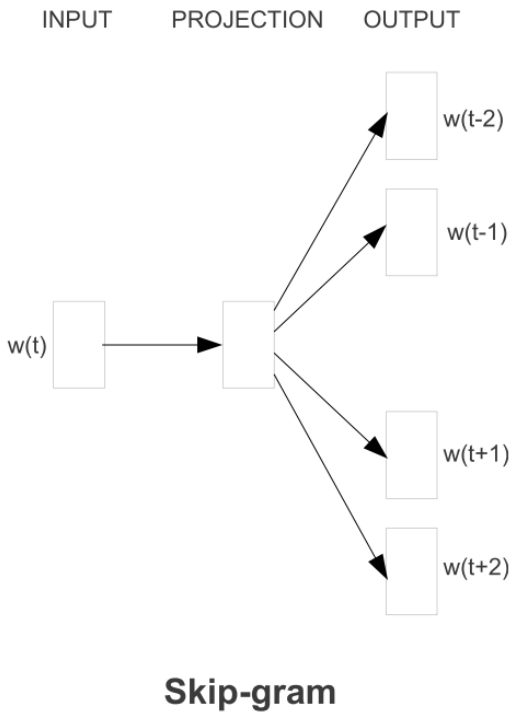

Figure 2: Skip-gram model architecture where the neural networks needs to predict the surrounding neighbors (Image extracted from Mikolov et al. [3])

.

2.4 Words to IDs

Last step of the pipeline is to translate the lists of words into lists of IDs. The objective of this translation is three-fold.

First, integers' size is smaller than words' thus greatly decreasing the disk size and memory footprint required to parse and store them.

Second, the ID used is the index of the corresponding embedding in a vector. Using a vector instead of a dictionary also reduces memory footprint (compared to a dictionary) and makes the access to the elements $\Theta(1)$ (without any clash possibility as in some Hashmaps).

Finally, there are words which do not have embedding because of the min. occurrences filter. During the translation, this words are assigned to index 0. Then, this first position in the embeddings is manually associated with a 0-valued vector. This way, we set all unknown-labeled words to the 0 information vector without any need to special processing when encountering them during NN training.

Execution time of this block was a bit lower than the preprocessing ones, about 37 minutes using also 16 CPUs but launching, again, 64 processes (due to the same I/O reasons already mentioned). It was directly implemented in parallel so no sequential comparison is available.

3 Dataset

The dataset used is the Wikipedia English corpora as of 20th of February 2017. It is a full raw dump so it contains HTML tags, links and many other useless informations that need to be stripped.

- Compressed data size: 13 GB
- Uncompressed data size: 57 GB
- wikiExtracted data size[4]: 12 GB
- Number of documents: 5.339.722

4 Results

Listing 1 shows some semantic tests and performs the semantic/grammatical analysis presented in [4] and [3]. The paper tests are divided into two big sections: semantic relations, like country-capital; and grammatical relations, like present-participle. Each of the categories is subdivided into smaller ones. Mikolov et al. report 66% and 72% with Google News dataset and embeddings of size 1000 against our 74% using a size of 200. However, their results use larger training and tests (approximately 10k more questions for the testing) and our model uses both their optimizations.

The model performs quite bad for antonyms category (0.44%) and is absolutely awful finding countries' currency (0.06%). First issue is probably due to the fact that Wikipedia is an encyclopedia (whose purpose is to define concepts) and antonyms are not normally used when defining a term (so noticing that relations may be harder). For the currency issue, one option is that the country-currency relation appears only in the country and the currency articles and almost nowhere else so the embeddings have few examples to learn from.

Incrementing the size of the embeddings to 500 and 1000 yielded slightly better results with an accuracy of 76-77%.

Listing 1: Semantic checks and accuracy testing performed with the 200 word embeddings. First two lines show the result of performing arithmetic operations using multiplicative combination objective (cosmul) described by Levy et al. [5]. Intuitively, if we remove the male sex from the word king and add the female sex, the result is queen. Next line, captures the relation Country-Capital (England, London) and transfers it to Baghdad, resulting in country Syria. Next three semantic checks show the 10 most similar words to a given one. Similarity is against with cosmul. Results are sensible: Paris yields other Europe capitals and French locations; Jupiter results in his moons and other solar system planets; and Zeus returns Greek gods' names starting with his wife, child, father, child, brother, etc... Next results show the accuracy of the embeddings capturing 5 and 9 semantic and grammatical relations respectively.

```
Loading model...
```

[4] See Section **??**

```
Operations using multiplicative combination objective:
 * King + Woman - Man = queen [0.973350346088]
 * Baghdad + England - London = syria [0.97702473402]

 * Most similar words to Paris:
[(u'brussels', 0.8833572268486023),
 (u'strasbourg', 0.8460177183151245),
 (u'marseille', 0.8442000150680542),
 (u'vienna', 0.8400871753692627),
 (u'france', 0.8300738334655762),
 (u'parisian', 0.8294538855552673),
 (u'lyon', 0.8263410925865173),
 (u'marseilles', 0.8226707577705383),
 (u'bordeaux', 0.8218927979469299),
 (u'berlin', 0.820243239402771)]

 * Most similar words to Jupiter:
[(u'uranus', 0.8909528255462646),
 (u'neptune', 0.8783587217330933),
 (u'amalthea', 0.8696370124816895),
 (u'pluto', 0.8651199340820312),
 (u'mimas', 0.8646141290664673),
 (u'ganymede', 0.8606989979743958),
 (u'saturn', 0.8543949723243713),
 (u'venus', 0.8473851084709167),
 (u'jovian', 0.8456070423126221),
 (u'moons', 0.8444883823394775)]

 * Most similar words to Zeus:
[(u'hera', 0.9136345982551575),
 (u'athena', 0.9080315232276917),
 (u'cronus', 0.8982190489768982),
 (u'aphrodite', 0.8967347145080566),
 (u'poseidon', 0.8967235684394836),
 (u'heracles', 0.8964233994483948),
 (u'hephaestus', 0.8927215933799744),
 (u'artemis', 0.8859370946884155),
 (u'dionysus', 0.8765401244163513),
 (u'helios', 0.8758820295333862)]

[Capital-Common-Countries]
        Accuracy [0.86] 437/506

[Capital-World]
        Accuracy [0.89] 1849/2085

[Currency]
        Accuracy [0.06] 3/54
```

```
[City-In-State]
        Accuracy [0.65] 1438/2203

[Family]
        Accuracy [0.94] 323/342

[Gram1-Adjective-To-Adverb]
        Accuracy [0.31] 236/756

[Gram2-Opposite]
        Accuracy [0.44] 121/272

[Gram3-Comparative]
        Accuracy [0.87] 1038/1190

[Gram4-Superlative]
        Accuracy [0.7] 420/600

[Gram5-Present-Participle]
        Accuracy [0.68] 636/930

[Gram6-Nationality-Adjective]
        Accuracy [0.96] 1322/1371

[Gram7-Past-Tense]
        Accuracy [0.64] 898/1406

[Gram8-Plural]
        Accuracy [0.78] 823/1056

[Gram9-Plural-Verbs]
        Accuracy [0.68] 516/756

[Total]
        Accuracy [0.74] 10060/13527
W
```

Next, we will explore the visual representation of a subset of 500 words after applying t-SNE dimensionality reduction. Words were chosen as follows: an initial word was selected and added to the words to be plotted, then the 10 nearest words (using cosmul) were added, for each of these words the 10 nearest were added (discarding repetitions) until the set size was 500.

The initial word was "Jupiter" because it was expected to lead to diverse areas (Greek mythology, planets and astronomy, possibly to music through operas based on ancient Greek myths, etc...). Figure 2 shows in fact 3-4 clusters of meaning. Top right are celestial bodies, study-related topics, and technical-scientific words on the right. Bottom cluster is related to music and operas. Left cluster is composed of mostly of French relevant people. I did not spend more time checking out their relations, but a good deal of them are dramaturgy and poets so this group could be between the music/opera one and another more related to prosaic literature (not

present in the plot).

Figure 3 shows the nearest words to Jupiter which are all planets and celestial bodies. Figure 4 shows another zoomed area in which the meaning has leapt from astronomy and physics to gradually more mathematically related topics (like optimization and computation) passing through waves' topics (wave-function or superposition).

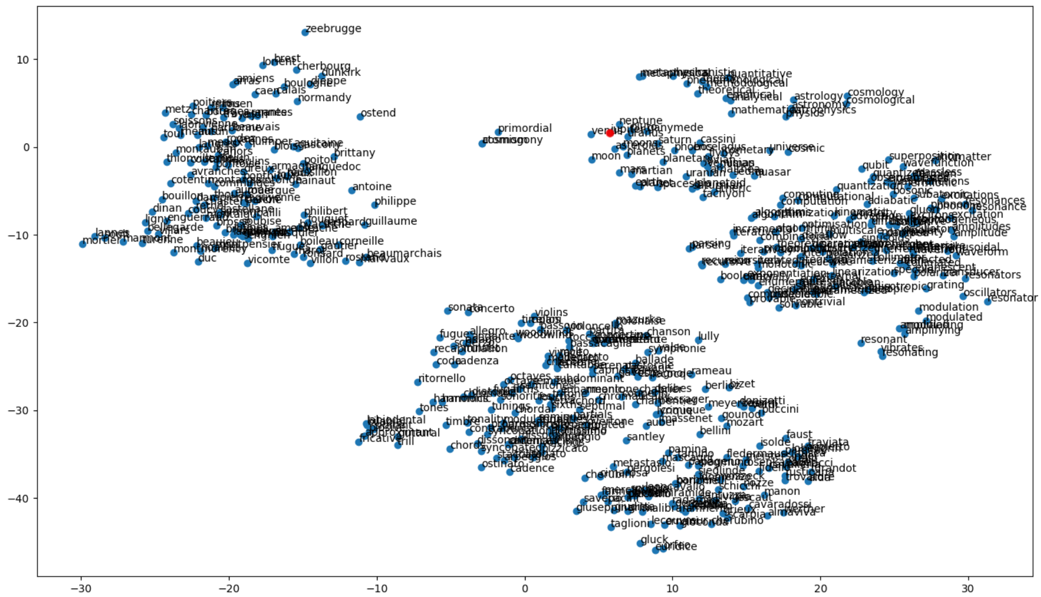

Figure 3: Scatter plot resulting of embedding the word representations into 2D with t-SNE dimensionality reduction. The model used had an embedding size of 200, a max-length window of 10, and was trained with the whole Wikipedia (thresholded at 200 occurrences per word). Jupiter is marked in red and its closest neighbors are celestial bodies.

.

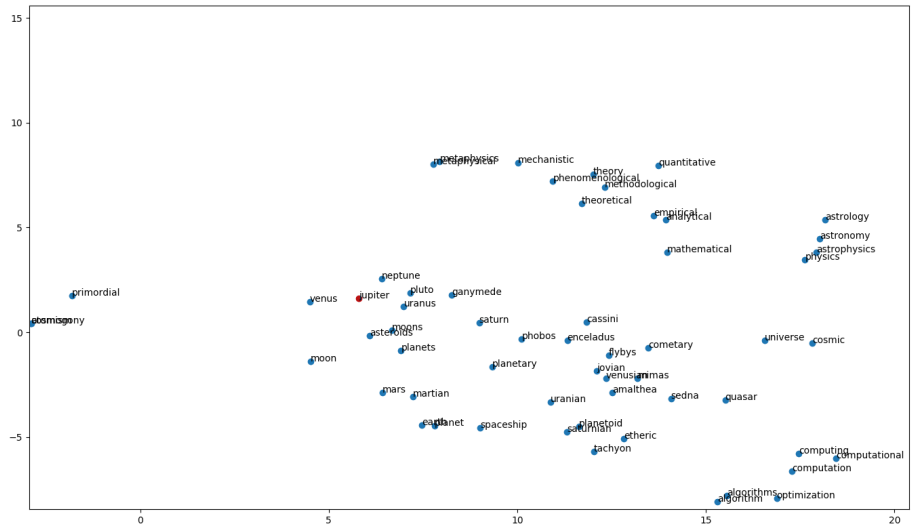

Figure 4: Detail of Figure 2. It shows Jupiter in red and its nearest neighbors: the solar system planets and its moons (even NASA's probe Cassini which is orbiting Saturn). Bottom-right, topics become more statistics-computation related. Top-right meanings are more related to physics, mechanics and experimental setups.

.

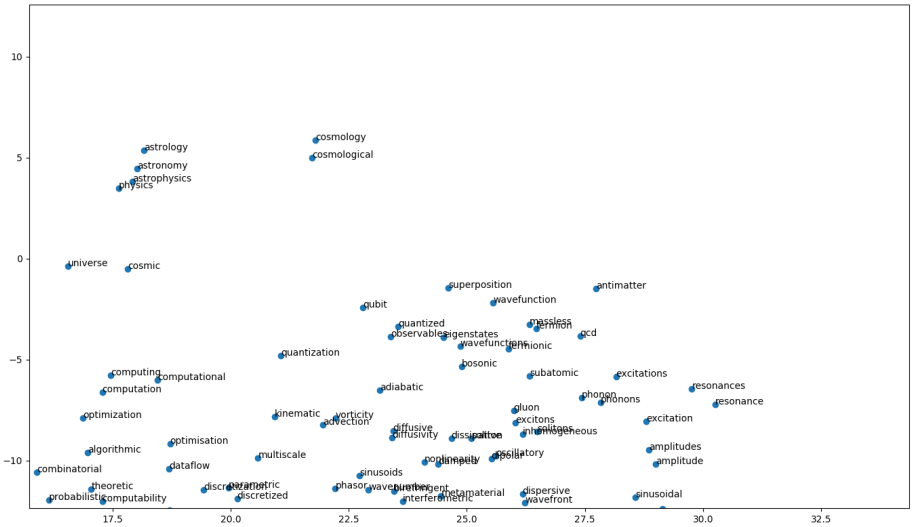

Figure 5: Detail of Figure 2 (it is located bottom-right of Figure 3). The representation captures the heavy relations between optimization, computing, statistics, and data in the bottom-left corner. Further to the right, we notice relations between acoustic waves, light, matter and its respective attributes.

.

5 Conclusions

The objective of processing the input data has been achieved. The developed processing pipeline is solid and can handle huge datasets in a reasonable time with certain amount of fail-safety thanks to the check-pointing

files of each block. The modularity of its components allows to easily retrain the word embeddings with different sizes without needing to execute the whole pipeline again. This is facilitates the production of a number of different embeddings for posterior evaluation during the LSTM network training.

The embeddings obtained show good accuracy and the visual and semantic checks show no big issues. The possibility to represent them visually and use arithmetic operations will be very valuable for the posterior step in which a technique needs to be developed to provide context/topic of word segments.

The translation of the files to numeric IDs was successful: they require far less memory than the actual text dataset and their indices can be directly used to get the corresponding embedding is constant time (see Subsection 2.1. The correctness of the translation has only been inspected visually for a subset of all the files and should be automatized.

6 Future Work

Future work includes the objectives not addressed in this project. Specifically, next assignment will explore how to provide context/topic for an arbitrary long sentence or paragraph fragment. LSTM and CLSTM implementations and their evaluations will be done in a separate project built on the foundations laid by this and next assignment.

For the word to ID translation, it is mandatory to build an automatized validation tool to make sure no errors are present in the translated files. The design line will be to translate back the IDs to words and match them to the original file. This comparison should yield very few different words (the words with not enough occurrences to be present in the word2Id dictionary will be different from the original ones), so it is feasible to compare all files and log an alert for those with a percentage of different words higher than a predefined threshold.

Finally, regarding the word representations, it would be interesting to explore the phrase embeddings [5] in which many terms can become a single entity allowing to model multi-word concepts such as *New York*. However, the possible improvements should be evaluated when used as input for the networks, which is the ultimate goal. Thanks to the processing pipeline, generating phrase instead of word embeddings only requires to change a line of code (Word2Vec call to Phrases).

References

[1] S. Ghosh, O. Vinyals, B. Strope, S. Roy, T. Dean, and L. Heck, "Contextual LSTM (CLSTM) models for Large scale NLP tasks,"

[2] L. Van Der Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[3] T. Mikolov, G. Corrado, K. Chen, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *Proceedings of the International Conference on Learning Representations (ICLR 2013)*, pp. 1–12, jan 2013.

[4] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," oct 2013.

[5] O. Levy and Y. Goldberg, "Linguistic regularities in sparse and explicit word representations," *CoNLL*, pp. 171–180, 2014.

[5] https://radimrehurek.com/gensim/models/phrases.html#module-gensim.models.phrases