Unsupervised Word Translation with Dutch and Russian

Svetlana Tchistiakova & Guido Linders

February 7, 2018

Introduction

In the field of machine translation (MT), historically, high quality translation models are built using parallel corpora in a supervised way for parameter estimation, most recently and successfully with the help of neural networks (e.g. Wu et al. (2016)). However, certain language pairs (e.g. those involving low-resource languages), suffer from lack of parallel corpus data, making this infeasible (Östling and Tiedemann, 2017). Recent approaches have begun to use semi-supervised methods with combined parallel and monolingual corpora (e.g. Cheng et al. (2016)). Those approaches only need a small amount of annotated data for training and thus are especially suited for language pairs for which very little parallel data is available.

Another more recent approach in neural machine translation uses no parallel corpora at all but relies solely on monolingual embedding spaces that are then mapped to each other. This work can serve as the first step towards creating a larger MT system, not limited to only word translation (e.g. as produced by Lample et al. (2017)). In this project we create a word translation system for Russian and Dutch, following the method presented in Conneau et al. (2017). This method follows a four-step process:

1. Train monolingual word embeddings for each language from distributional data
2. Train a generative adversarial neural network (GAN) to find a mapping between source language embeddings as target language embeddings
3. Refine the parameters trained by the GAN using the Procrustus algorithm (symmetrically for source-to-target, and then target-to-source)
4. Calculate a similarity metric for a source language word to each target language word, and choose the best one as the translation

The word embedding (Řehůřek and Sojka, 2010), GAN (Goodfellow et al., 2014), and refinement/similarity calculation algorithms (Conneau et al., 2017) that we use have all been described in detail by their source authors. We now seek to apply these algorithms to a new Dutch/Russian corpora. Where we have deviated from the method in Conneau et al. (2017) is primarily in the acquisition and processing of our corpus and the use of `gensim` (Řehůřek and Sojka, 2010) in training our word embeddings.

In the rest of this report, we will cover (1) a slightly more in-depth explanation of the algorithm, (2) those implementation details which are particular to our approach, (3) an evaluation of the results, and (4) conclusion with suggestions for future work.

1 Explanation of the algorithm

The model as proposed by Conneau et al. (2017) consists of 3 steps, which are step 2-4 in the previous section. This algorithm starts with monolingual word embedding spaces in both the source and target language. In the first step a rough mapping from the the source language space to the target language space is estimated using a generative adversarial network. Effectively a rotation matrix is learned which maps the source language embedding space onto the target language embedding space. This network consists of a discriminator that tries to discriminate between the target language embedding and the source language embedding that is modified by the rotation matrix. The rotation matrix is trained to “fool” the discriminator by trying to make the source and target embedding spaces as similar as possible so that the discriminator cannot distinguish between the two spaces. The training of the discriminator and rotation matrix is an iterative process.

In the next step this rotation matrix is refined by using the Procrustus algorithm on a subset of the embedding space, more specifically the most frequent words. The intuition is that the most frequent words have a more reliable representation in space because they have been seen more often in the training of the word embeddings. The Procrustus algorithm uses these most frequent words as anchor points and tries to minimize the distance between the closest anchor points using singular-value decomposition (SVD). This step can also be repeated for a certain number of iterations, but will have a very limited impact on the final performance after just one or two iterations

In the final step, actual word pairs are constructed. The most naive way of doing this is to match each source word with the K nearest neighbors in the target language space. This method, however, has the problem of being asymmetric in the sense that if a word in the source language is a nearest neighbor (NN) of a word in the target language space, the reverse does not have to be true. This leads to some source words being hubs, and having many NN’s in the target language space while other source words have no NN’s in the target language space. The authors propose another method where they compute, for each word in both the source and target language a mean similarity between the word and its K NN’s in the other language space. These mean similarities are then used to compute a similarity score between a word in the source language and a word in the target language. This final similarity score is called cross-domain similarity local scaling (CSLS).

2 Implementation details

2.1 Downloading comparable corpora

For the training data, we wanted to create a comparable Russian/Dutch corpus, where each corpus covers similar topics, so that we can more effectively train word translations. Wikipedia contains a lot of data and is easily accessible. For it, we used the MediaWiki API to scrape Wikipedia. That is, we searched for random Wikipedia pages in the source language, and if that page contained a language link to the target language, we downloaded both pages. We alternated searching for Dutch pages first, and then searching for Russian pages first. In this way, we built up a comparable corpus consisting of the same number of documents on the same topics in each of the two languages.

One problem with this approach is that each downloaded document might vary in number of words, which could happen if the Wikipedia articles in one language are generally less descriptive than in the other. This was one problem we did not address in our work. Another potential problem with data from Wikipedia is that it contains a relatively high number of proper nouns. We can work around this problem by increasing the number of documents and by using a cut-off threshold for the number of types as well as a minimum number of occurrences of a word to be included in the word embeddings.

2.2 Data processing

We processed the data by splitting the text documents into sentences, normalizing the sentences, and splitting them into lists of words. Normalization included removing punctuation, converting each sentence into lower case, simplifying numeric expressions, and lemmatization.

The text data was split into sentences using the Python3 `NLTK` package (Bird et al., 2009), and then converted into lower case. Numeric expressions were replaced with a single type called `NUM`. This was done in order to avoid having a potentially infinite number of word embeddings representing numeric expressions. Note that we did not replace any numbers that were already written out in words with `NUM`, though this is another normalization that could be tried.

Since we are focused on word translation, we are particularly interested in the semantic space, and less interested in syntactic differences of word forms. (The opposite may be true in a larger project focused on sentence translation.) Russian, in particular, exhibits a rich inflectional morphology, which results in many different word forms for the same semantic concept, and could needlessly increase the number of word embeddings in our model. For these reasons we chose to try experiments in which we lemmatize the corpus before training word embeddings, in addition to experiments without lemmatization. Lemmatization was performed both before training of the language-specific word embeddings, as well as to create the Dutch-Russian/Russian-Dutch evaluation dictionaries. One potential problem with lemmatization is that it introduces another source for errors, in case a wrong lemma is chosen. This could in turn negatively impact the performance of the model.

To perform Russian lemmatization, we used the `pymystem3` tool[1], which is a wrapper around the Yandex Mystem 3.0 tool (Segalovich, 2003). Lemmatization was performed on a per-sentence level.

For Dutch, we used the freely available CSTLemmatizer tool[2] (Jongejan and Dalianis, 2009). Because of the slow nature of invoking this tool from our program, lemmatization of Dutch was performed on whole documents, before sentence splitting. For some Dutch words the lemmatizer generates more than one lemma. In this case, the first lemma was chosen for normalization to train word embeddings, and all lemmas produced were used as entries in the evaluation dictionary. Ideally we would like to use the context of the word for deciding which lemma to use. A part-of-speech (POS) tagger might help with this as the different lemmas often have a different POS tag. However the Dutch lemmatizer does not provide the POS tag with the different lemmas. For this reason we have left the step of inferring the correct lemma for Dutch from the context and its POS for future work. When creating the evaluation dictionaries, we do not have any information on the POS tag because there are no surrounding words for context. In future work, the POS tag might be able to be inferred from the English word in the original dictionaries, or using a morphological analysis tool.

2.3 Generating word embeddings

The Python3 `gensim` (Řehůřek and Sojka, 2010) library was used to generate word embeddings. The normalized, lemmatized sentences were fed directly into the `gensim Word2Vec` model, which trained language-specific word vectors for Dutch and for Russian. The number of word tokens (i.e. length of the corpus in words) and the vocabulary size (unique word tokens) for each language can be found in Table 1. We did not limit the vocabulary size for the word vectors as we could limit this at a later stage in the model of Conneau et al. (2017).

		Documents	**Word tokens**	**Vocab size**
Dutch	Lemma	70275	127,320,984	29,786
	No lemma	70275	127,627,062	30,251
Russian	lemma	70275	198,561,330	38,681
	No lemma	70275	190,516,290	27,714

Table 1: Number of words used for training language-specific word vectors. The "lemma" row indicates that the input text was lemmatized prior to training word embeddings, whereas the "no lemma" row was not.

For the generation of the word vectors we used continuous bag-of-words (CBOW). This technique uses a limited number of context words in generating word representations. In our case we used the default window of 5. We chose CBOW over Skip-gram because it has a faster training time and the

[1]Code retrieved from: https://github.com/nlpub/pymystem3
[2]Code retrieved from: https://github.com/kuhumcst/cstlemma.

accuracy for most frequent words is slightly better (Mikolov et al., 2013). In line with Conneau et al. (2017) we used 300 as our word embedding size. Finally, we sorted the word embeddings on frequency so that we could later limit our vocabulary size by cutting off the least frequent words. For the other parameters we mostly used default values and for these details we refer to our implementation[3].

2.4 Training the model

The word vectors were then directly fed into the model of Conneau et al. (2017)[4]. As this model is fully unsupervised, it does not need any gold-standard dictionaries for creating the translation pairs. However the actual implementation needs a small dictionary which is used for intermediate evaluation and, after analyzing the code, it does not seem that the implementation uses these dictionaries to facilitate learning of the translation pairs, for example in choosing the best rotation matrix from a set of rotation matrices.

To choose the hyperparameters, we did some experiments with different settings. However as the model takes some time to finish we were limited in doing a lot of experimenting. Therefore most parameters were set to values as described by Conneau et al. (2017) or were kept to default values. We'll discuss the most important settings below. For other parameter settings we refer to our implementation[5].

We used a vocabulary size of 35000. This was about the maximum we could use as our own word embeddings generated around 35000 lemmas. This means we hardly removed low frequent words which could have a negative impact on the results as those words have not been seen a lot in training the word embeddings and might be wrongly represented. Please note that we did exclude words with a frequency of less than 5 in the training of the word embeddings However, since we use data from Wikipedia, probably there are still a lot of proper nouns in these embeddings. The reason for not filtering out more words is that our vocabulary size is already relatively small and we did not want to make it even smaller.

For the discriminator, we used the details as described by Conneau et al. (2017). The discriminator has 2 hidden layers of size 2048. In contrast to Conneau et al. (2017), all of the words are used in the discriminator. Adversarial learning is done with 5 epochs and each epoch consists of 100,000 iterations as around those numbers the discriminator loss stabilizes and we want to prevent the model from overfitting the data, which could have a negative impact on the results. Finally, a batch size of 32 is used and optimization is done using stochastic gradient descent.

3 Evaluation

3.1 Method

For language pairs with little to no parallel corpora there are often also no word translation dictionaries available. Indeed, for our language pair we could not find a high quality, freely available dictionary. As such, we bootstrapped from the English/Dutch and English/Russian dictionaries provided by Conneau et al. (2017) to create a two-way dictionary of Dutch and Russian in order to evaluate the words translations that come out of our model. We thus use English as an intermediate language to generate word translation pairs of Dutch/Russian. These dictionaries are of high quality, and sorted by word frequency. However, they contain many polysemous words, that is, an English word typically has a few different translations into the target language, some of which express very different semantic senses. We matched the two dictionaries up by creating a many-to-many mapping between the Dutch and Russian words which had the same English translation. This means we get the same word pairs for both directions of translating. See Figure 1 one for a visualization.

[3]See `w2vconfig.py` in our code for these details.
[4]Code retrieved from: https://github.com/facebookresearch/MUSE.
[5]See train.sh as well as `unsupervised.py` in the folder `MUSE` for details on the parameter settings

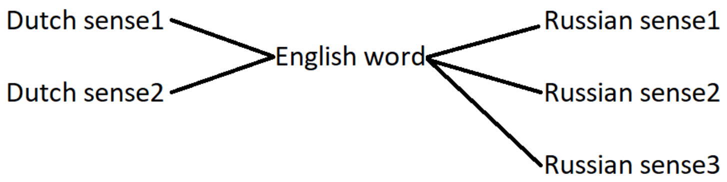

Figure 1: We generate a many-to-many mapping between each Dutch translation of an English word and each Russian translation of the same English word. In this example the English word has two different senses in Dutch and three different senses in Russian.

This method causes difficulties in the case of English words which may contain multiple unrelated senses in Dutch and/or Russian. For example, the English word "iron" out of context is ambiguous in the meaning of "object to straighten clothes" and "the metal." Dutch and Russian both use two separate words in these contexts, as depicted in Table 2. Using the English word as a base, and without any other context, we cannot discern which words in Russian and Dutch correspond to one another. Since we simply create a many-to-many mapping between all the Dutch words and all the Russian words under the same English headword, for example, the Dutch word "Strijkijzer" will be incorrectly translated into Russian as both "утюг" and "железо".

	Object to straighten clothes	Metal
English	iron	iron
Dutch	strijkijzer	ijzer
Russian	утюг	железо

Table 2: English polysemous word "iron" with separate words in Dutch/Russian

Another problem is that no word translation model (supervised and unsupervised) has been evaluated on our language pair. This makes it very difficult to compare the results of our approach to other models or even a simple baseline. For this reason we decided to compare the results of training on our own word embeddings with training on the FastText (Bojanowski et al., 2016)Wikipedia word embeddings, the same ones used by Conneau et al. (2017).

Our evaluation method uses the same CSLS and nearest neighbours (NN) metrics as Conneau et al. (2017) and as explained in Section 1, using our bootstrapped Dutch/Russian dictionary. These resulting dictionaries could provide an overly optimistic estimate of performance. That is because, for example, if the model translates the Dutch word "Strijkijzer" into the unrelated Russian word "железо," the accuracy metric would incorrectly report this as a correct translation.

Note that the `MUSE` code from Conneau et al. (2017) uses a small subset of the dictionary to test during the Procrustes refinement step for memory and speed reasons. For this portion, we also created a smaller dictionary, in which we removed polysemous words by simply choosing the first available translation.

The source code from Conneau et al. (2017) also comes with an evaluation script, which we used in evaluating the performance of our model. The reason for using this evaluation script is to use the same steps in the evaluation as in (Conneau et al., 2017) and to be able to compare our results with theirs, even though the language pairs differ. As Dutch and English are relatively similar we would expect the results on Dutch-Russian to be comparable to results on English-Russian as well as in the other translation direction.

3.2 Results

Unfortunately, we ran into some problems during model training using the `MUSE` code from Conneau et al. (2017) that seemed to have prohibited us from getting proper working results. Because Conneau et al. (2017) have reported results on an English/Russian set, we decided to choose this as a baseline model to run through their code. We used this language pair as a baseline to make sure that our system is working correctly and also because we would expect similar results to the Dutch/Russian set as the languages share relatively a lot of similarities in semantics- and morphology-wise. We used the same fastText vectors that they report using, although some of our hyperparameters had to be lowered in order to allow running on the hardware available to us.

As reported in Table 3, the performance of the Russian → English model seems very good. However, the results on English → Russian are near 0, as are the results on all other combinations of Dutch/Russian with and with fastText vectors and with our own vectors, with and without lemmatization (where applicable). Since we would expect the English → Russian direction with the fastText vectors to have a performance of better than 0, we suspect that either something in their code, in our set up or in the parameter settings of the model has gone wrong. Unfortunately, we were not able to resolve these issues. This is partly due to the fact that training the model is time-consuming. In addition, the problem of overlaying 2 vector spaces without supervision may have many local optima, and it's possible that without the right initialization and parameter settings, the model is entirely unable to converge.

		NL → RU		**RU → NL**		**EN → RU**	**RU → EN**
		Lemma	No lemma	Lemma	No lemma	**-**	**-**
fastText	NN	-	0.11	-	0.75	0.09	62.86
	CSLS	-	0.22	-	0.65	0.09	65.91
ours	NN	0.00	0.00	0.00	0.00	-	-
	CSLS	0.00	0.00	0.00	0.13	-	-

Table 3: NN and CSLS metric results (for K=1) on our own vectors created from the Wikipedia data, as compared to the fastText baselines from `MUSE`. The "lemma" column indicates that the input text was lemmatized prior to training word embeddings, whereas the "no lemma" column was not (only relevant for our vectors). The English/Russian models were also trained as a baseline to make sure the original system words as it should. Unfortunately, an error in either the code or our set up or a wrong parameter setting has prevented us from getting proper results.

4 Conclusion

For now there is, unfortunately, very little to conclude as we did not get the model to work correctly. We still think that, once we are able to resolve the issues with the model, our results would be relatively close to the English/Russian set. However because we used a much smaller data size in generating the word embeddings and our data probably still contains a lot of proper nouns, we expect our performance to be a bit lower. We also expect a positive influence on using lemmatization, even though it introduces another potential source for errors.

For future work (assuming issues with the model can be resolved) the first step could be to download significantly more data from Wikipedia that would generate larger word embeddings (with more lemmas) and train these word embeddings on the model. With more data our word embeddings would become more accurate in representing the semantics of the word and would contain a relatively lower percentage of proper nouns. A second step could be to see if we can tune the parameters better to overlap the Dutch and Russian embeddings better and obtain better translation pairs. For this we could extract and analyze the word pairs that come out of the model.

Also work on the Dutch and Russian lemmatizer could be done to improve the quality of the generated lemmas, which could facilitate performance of the model as well. As mentioned previously we could try to predict the Part-of-Speech (POS) tag for the word, given the context words. This in turn might guide us in choosing the correct lemma.

Future work could improve the gold-standard dictionary for evaluation that was created manually and contains word pairs that are not translations of each other. Future work could also look at improving methods for creating word translation dictionaries for language pairs for which there are no word translation dictionaries available. One could try to improve the method we used by using an intermediate language. As explained this method has its limits, especially for polysemous words and could be improved.

References

Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.".

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606.*

Cheng, Y., Xu, W., He, Z., He, W., Wu, H., Sun, M., and Liu, Y. (2016). Semi-supervised learning for neural machine translation. *CoRR*, abs/1606.04596.

Conneau, A., Lample, G., Ranzato, M., Denoyer, L., and Jégou, H. (2017). Word translation without parallel data. *arXiv preprint arXiv:1710.04087.*

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Jongejan, B. and Dalianis, H. (2009). Automatic training of lemmatization rules that handle morphological changes in pre-, in-and suffixes alike. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 145–153. Association for Computational Linguistics.

Lample, G., Denoyer, L., and Ranzato, M. (2017). Unsupervised machine translation using monolingual corpora only. *arXiv preprint arXiv:1711.00043.*

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781.*

Östling, R. and Tiedemann, J. (2017). Neural machine translation for low-resource languages. *CoRR*, abs/1708.05729.

Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. http://is.muni.cz/publication/884893/en.

Segalovich, I. (2003). A fast morphological algorithm with unknown word guessing induced by a dictionary for a web search engine. In *MLMTA*, pages 273–280. Citeseer.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.