Words Embeddings and Translation

Alexis Hummel, Hamza Touzani, Guillaume Kunsch (WE HGA)

November 2022

The goal of this paper is to try various existing methods to perform word to word translation in bilingual settings. To do that we will rely on off-the-shelf embeddings and implement various mapping (linear, linear with constraint, unsupervised) between embedding spaces to translate word from one language to another.

1 Context and problem statement

1.1 Word Embedding

To perform translation, we first need to get a particular embedding. Basically, an embedding is a function that allows to transform a word in a vector (the size of the vector space is an hyperparameter).

To get an embedding, one needs to train the embedding function on a monolingual corpus of sentences. The "trick" for the training is to not care for the vector representation itself (as a matter of fact, we don't have prior vectors to feed into the algorithm), but rather to predict the likeliness of the occurrence in the neighborhood of a set of words. This idea rely on the work of Harris (1954) which states that words occurring in similar contexts tend to have similar meanings. In this way, words that are likely to appear together to complete a sentence will have a rather similar representation in the vector space. In a nutshell, words tightly linked by their meaning will be closed in the embedding space.

There are 2 ways of training an embedding: either combine the representations of surrounding words to predict the word in the middle (Continuous Bag of Words, CBOW) or starting from the word in the middle and predict representations of surrounding words (Skip-Gram).

As training an embedding is quite trickcy and computer-intensive, we won't train embeddings ourselves bur rather rely on pre-trained embeddings.

1.2 From Embedding to Translation

Once the embedding is trained (or retrieved) and available, the translation task can be expressed by the transformation from the vector space of one language to another.

1.2.1 Supervised setting

The supervised setting is the case where we have a dictionary that contains the translation of some words. The work of Mikolov et al. (2013) suggests that even a simple linear mapping is sufficient to learn translation.

By noting a given a set of word pairs and their associated vector representations $\{x_i, y_i\}_{i=1}^n$, where $x_i \in \mathbb{R}^{d_1}$ is the distributed representation of word i in the source language, and $y_i \in \mathbb{R}^{d_2}$ is the vector representation of its translation. For $W \in \mathbb{R}^{d_2 \times d_1}$ the transformation matrix between vector spaces, the optimisation problem is then:

$$\min_W \sum_i \|Wx_i - y_i\|^2 \tag{1}$$

This convex, differentiable problem can be solved using Gradient Descent method.

1.2.2 Unsupervised setting

In this setting, we don't have any dictionary encapsulating knowledge about the translation of a specific bag of words. We will to need to create it ourselves. The work of Conneau et al. (2017) showed that in this setting the use of Generative Adversarial Networks (GAN) can provide results as good as supervised settings, and even better ones in some cases. The GAN works by forming a two-player game where the discriminator try to predict if an embedding is generated or not. On the other hand , the generator aims at making source and target embedding as similar as possible. The training consists of several iterations of the discriminator step, followed by the mapping step. At each mapping step, we try to approach the generator matrix by an orthogonal matrix with this update rule :

$$W \leftarrow (1+\beta)W - \beta(WW^T)W$$

At the end of each epoch, we compute the discriminator loss and the cosine similarity (define later in (2)) and then we build the dictionary, using the best mapping.

2 Experiments

2.1 Set-up

We will use embeddings available from Fasttext, a Meta library for training embedding developed in Bojanowski et al. (2016). Besides, the data will come as well from Fasttext and, after preprocessing, is composed of 9,500 words for training and 500 words for test, among the most used for each language. Our main goal will be to translate words from english to french and vice-versa, but we will also try turkish translation to test the "proximity" of language as prominent element. We will rely on 2 kind of embeddings for each language:

- the first one, which we will call $e_{flexible}$, is flexible in the sense that it can be used to find the vector representation of any word, even ones that do not exist. Besides, we can choose the dimension of the vector representation (up to 300).
- the second one, which we will call e_{fixed} is less flexible because the vectors representations are already provided for a set of words and we can't change the size of the embedding which is 300.

We will use various technique for computing the transformation matrix W that are detailed in the following subsections. Once we have W, the translation of a particular word of representation x will be done by finding the k-nearest-neighbors of Wx with respect to the vectors representations of the set of all $\{y_i\}_{i=1}^{n}$ available. The distance used for the kNN algorithm is the cosine distance, and we will compute to accuracy metrics P@1 and P@5. P@5 (respectively P@1) is the accuracy of the method for the case of the true translation is in the the 5 (respectively 1) nearest neighbors.

2.2 Supervised setting

2.2.1 Using Euler distance

As a first approach, we implemented the optimisation problem of equation (1). We have tried it on the translation between english and french , with the 2 kinds of embedding (and for $e_{flexible}$ we have tried various dimension of vector space). The algorithms have been trained with Gradient Descent using same learning rate (0.1) and number of iterations (2,000). The results are presented in Figure 1.

Note that in this experiment the dimension d of the vector representation is the same for the starting space and the arrival space. The effect of the dimension size is clearly visible. As d increases the accuracy increases as well. It suggests that the complexity of word is better encapsulated in vector space of higher dimension. As d is limited to 300 for our $e_{flexible}$ we can't try bigger representation

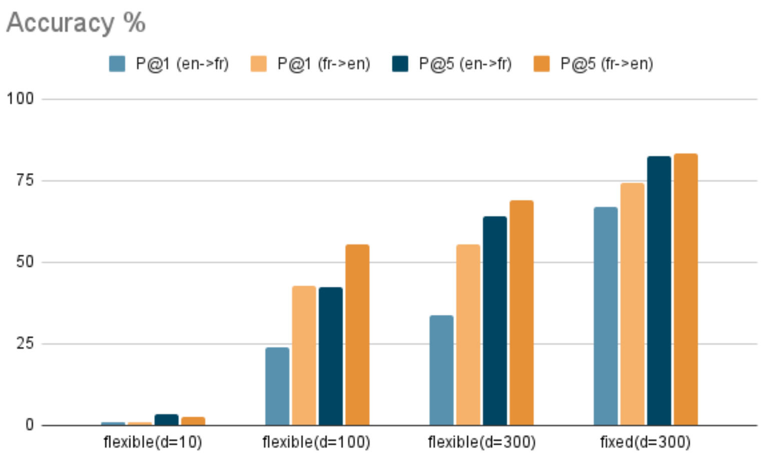

Figure 1: Accuracy for the translation between french and english for various dimension d and embeddings

to see if it allows to reach higher accuracy or on the contrary if it leads to a form of overfitting in the end.

Besides, in nearly all cases, the translation from french to english brings better accuracy than the one from english to french. It is unexpected as we used the same set of words for the 2 translations. It suggests the matrix W learned in both case is not invertible.

Finally, it is clear as well that e_{fixed} brings better accuracy than $e_{flexible}$ at the same dimension. This can be a consequence of the way they have been trained, and the trade-off between flexibility and accuracy appears blatant.

From now on, in the rest of the paper we will ony use e_{fixed} since it led to the best accuracy.

2.2.2 Using cosine distance and orthogonal constraints

The work of Xing et al. (2015) pointed to some inconsistencies of the approach developed in Mikolov et al. (2013). First, as it is the cosine distance that is used to find nearest neighbors for translation, it makes more sense to change the objective function from a Euler distance to a cosine distance. Besides, in the way the embeddings are trained, enforcing the vector representation to be normalized allow to make the inner product (which is used by Mikolov et al. (2013)) coherent with the cosine distance. Finally, as all the word vectors are located on a hypersphere (as a consequence of normalization) enforcing a W orthogonal is a way to preserve this structure.

The modified optimization problem is therefore expressed as:

$$\max_W \sum_i (Wx_i)^T y_i \text{ such that } WW^T = \text{Id} \tag{2}$$

It can be solved by Projected Gradient where at each iteration the update is performed as in classical gradient method with $\nabla W = \sum_i x_i y_i^T$, then W is projected on the space of orthogonal matrix by $\min_{\bar{W}} \|\bar{W} - W\|$ such that $\bar{W}\bar{W}^T = \text{Id}$. This last projection is known as the Procrustes method.

We implemented it using Mini Batch Gradient method, with a learning rate of 1 and only 2,000 iterations as the computing was slower. The results are displayed in Figure 2 and Table 1. Note that the results obtained here with equation (1) are less accurate than in Figure 1 because here it was perform with Mini-Batch GD and with less epoch. Our results show that, on the same number of iterations the method based on equation (2) performed way better than equation (1). However, the algorithm for training is way slower because of the compute of gradient in equation (2).

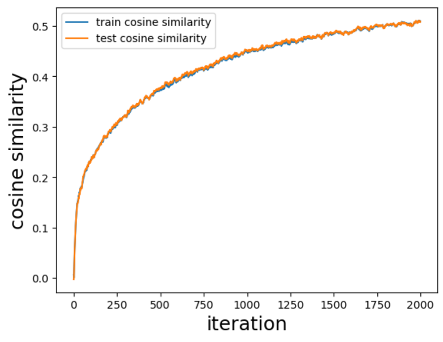

Figure 2: Cosine similarity for equation (2) for translation from french to english

Model	P@1(%)	P@5(%)
En $\rightarrow$ Fr (1)	24.4	39.6
En $\rightarrow$ Fr (2)	61.2	78.4
Fr $\rightarrow$ En (1)	19.2	30.8
Fr $\rightarrow$ En (2)	65.8	77.4

Table 1: Comparison between accuracy for equation (1) and (2)

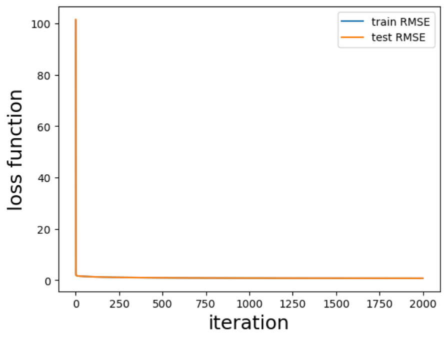

Figure 3: Loss function with Euler distance minimization with orthogonal constraints

Model	P@1(%)	P@5(%)
En $\rightarrow$ Fr (1)	67	82.6
En $\rightarrow$ Fr (3)	69.8	86
Fr $\rightarrow$ En (1)	74.2	83.2
Fr $\rightarrow$ En (3)	74.8	84.4

Table 2: Comparison between accuracy for equation (1) and (3)

2.2.3 Using Euler distance and orthogonal constraints

As we've seen, even if the method detailed in last section allows to reach more accuracy (at a fixed number of epochs), we need to deal with slower computing than can be detrimental in some cases. To cope with that, we have tried to combine the 2 methods. We will stick to the Euler distance objective function because the gradient computation is faster (so we can stick to GD on the whole train set), but at each iteration we will project the transformation matrix W on the set of orthogonal matrix.

$$\min_W \sum_i \|Wx_i - y_i\|^2 \text{ such that } WW^T = \text{Id} \tag{3}$$

As displayed on Figure 3, the training is faster thanks to this method and reach higher accuracy as displayed on Table 2.

2.2.4 Procrustes closed form solution

Finally, the work of Conneau et al. (2017) showed that we can actually compute directly a closed form solution of:

$$W^* = \underset{W \in O_d}{\text{argmin}} = UV^T \text{ with } U\Sigma V^T = \text{SVD}(YX^T) \tag{4}$$

This methods works well when the dataset is not too big, and ensure the best possible transformation matrix W. The accuracy obtained is the highest of all methods. Results are displayed in Table 3.

Model	P@1(%)	P@5(%)
En $\rightarrow$ Fr (4)	70.2	86
Fr $\rightarrow$ En (4)	75	84.6

Table 3: Results for equation (4)

2.3 Unsupervised setting

Here we use for the discriminator a MLP containing two hiddens layers of size 2048 with a leaky Relu, an input dropout of 10% and a final sigmoid activation function. The generator consist of a linear mapping initialized with identity matrix. Both generator and mapping are optimized with stochastic gradient descent of learning rate 0.1 and decay of 0.95. We also use $\beta = 0.01$ for the orthogonalization process. As said before, the discriminator loss is computed at the end of each epochs. Considering $P_{\theta_D}(source = 1|x)$ the probability that a vector x is considered to be a source embedding by the discriminator, his loss is given by the following formula :

$$\mathcal{L}_D(\theta_D|W) = -\frac{1}{n}\sum_{i=1}^{n} logP_{\theta_D}(source = 1|Wx_i) - \frac{1}{m}\sum_{i=1}^{m} logP_{\theta_D}(source = 0|y_i)$$

Those settings are adapted from the work of Conneau et al. (2017). We train this model with 50 epochs of size 50 000 with a batch size of 32.

2.3.1 Results

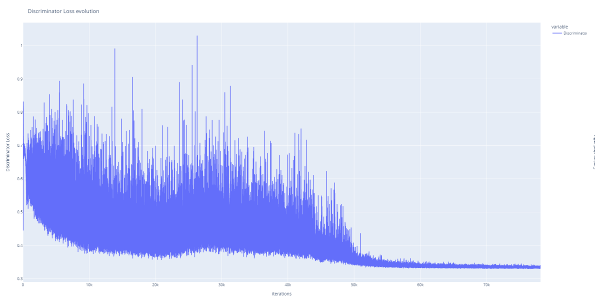

Figure 4: Discriminator Loss

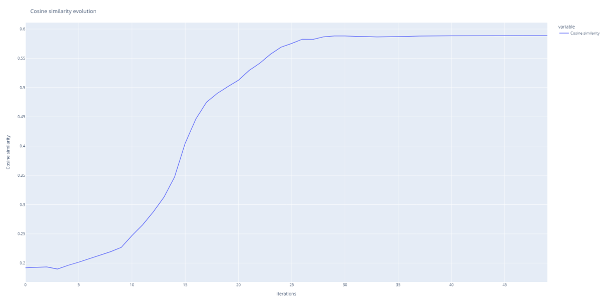

Figure 5: Cosine similarity for translation from english to french

Model	P@1(%)	P@5(%)
En $\rightarrow$ Fr (1)	70	85
Fr $\rightarrow$ En (1)	63	78

Table 4: Accuracy

As expected, the discriminator loss fluctuates a lot over the number of iterations before converging to 0,33. On the other hand it gives a good result on cosine similarity and accuracy that can already be reached with 25 epochs.

3 Conclusion

In this work we have tested various techniques for word to word translation. We have seen that embeddings are so versatile and powerful that even a simple supervised linear transform is sufficient to reach good accuracy. However, imposing orthogonal constraint is more efficient but results in a trade-off between calculation time and accuracy. Finally, even in unsupervised case, it is possible to reach accuracy as high as in supervised settings by using GANs.

References

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

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

Harris, Z. (1954). Distributional structure. *Word*, 10(2-3):146–162.

Mikolov, T., Le, Q. V., and Sutskever, I. (2013). Exploiting similarities among languages for machine translation.

Xing, C., Wang, D., Liu, C., and Lin, Y. (2015). Normalized word embedding and orthogonal transform for bilingual word translation. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1006–1011, Denver, Colorado. Association for Computational Linguistics.