Natural Language Processing Coursework

Pierre El Mqirmi
Imperial College London
pte19@ic.ac.uk

Panagiotis Fytas
Imperial College London
pf2418@ic.ac.uk

Alin Fulga
Imperial College London
af1719@ic.ac.uk

Abstract

In this project, we are tasked with creating a regression model to predict the quality of a machine-translated sentence. While trying multiple Pre-processing steps, word representations techniques and different neural network architectures we discovered that using Bert Embeddings, with only punctuation removal and as an input to an MLP performs the best. Specifically, we achieve a Pearson Correlation of 0.1404 on the test set.

1 Introduction

The task we are addressing in this report is a quality estimation of machine translation. Specifically, we are going to focus on the translation of English sentences to German sentences.

For this project we used the library `PyTorch` (Paszke et al., 2019) for the construction of our neural network models, `sklearn` (Pedregosa et al., 2011) to perform SVR algorithm, TF-IDF vectorisation, Randomized Search and Cross Validation. Finally, we used `Spacy` (Honnibal and Montani, 2017) and `nltk` (Loper and Bird, 2002) for some basic NLP functionalities.

Our **code** can be found on the following github repository: `https://github.com/PierreElm/NLP-QualityMT`

2 Pre-processing

In this section we will detail the class `PreProcessing.py` we built to apply different types of pre-processing to our corpora. This class serves as a baseline for most pre-processing we are applying to the data in all our models. It allows us to tokenize the corpus of each language with high flexibility. Specifically, we are able to change the stop words, the punctuation removal or the splitting of the punctuation just by giving different parameters to the function of this class.

Therefore, we tried different types of pre-processing for the models that we will present later. For example, we tried to use the stop words from `nltk` (Loper and Bird, 2002) but we also tried to use our own stop words. Moreover, we also tried to keep all the stop words as we thought that their presence may affect the quality of the translation. We discovered that depending on the representation of the data, removing stop-words may not be necessary.

In addition to this class, we have also tried lemmatisation from *Spacy* (Honnibal and Montani, 2017). Even though that may not make perfect sense for our task as it may lead to confusion between noun and verbs and have an impact in the meaning of the translation. However, we have not performed stemming given it can lead to transforming different sentences to similar ones.

3 Word Representation

In this section we are examining the way the representation of words affect the performance of *SVR algorithm* and *Linear Regression algorithm*.

3.1 Vocabulary

To build the vocabulary of each corpus we used the code of the class `Vocabulary` from *Tutorial 3*. However, we adapted it for more flexibility and to fit our requirements. The code can be found in the file `Vocabulary.py`.

3.1.1 Indices representation

The idea behind this representation is to represent each word by its index in vocabulary to be able to get a vector of indices for each English sentence and its German translation. During the pre-processing of the training sets, we are keeping track of the maximum size of the processed sentences.

The linear regression algorithm takes as input the concatenation of the vectors corresponding to an English sentence and its German translation. The size of the vectors is equal to the maximum size we computed before. Therefore we are padding the sentences that are shorter than this size and truncating longer sentences.

The results obtained by this method can be seen in Table 1. As we were not satisfied with those results, we decided to not focus on this representation. However, it was interesting to see that removing stop words and punctuation dramatically increased the performance of the model.

Pre-processing	Pearson	Rmse
Stop words / Punct	-0.0895	21.25
Stop words / No punct	0.0058	3176
No stop-words / punct	0.0152	0.8650
No stop-words / No punct	0.0306	0.8652

Table 1: Results on validation set using *linear regression* on Indices representation.

3.1.2 Bag of Words

In this representation, we are creating a vocabulary for each corpus. Then each sentence is represented by a binary vector. We also tried to represent them with frequency vector but binary vector was giving better performances. Therefore, we focused on the representation as a binary vector. We are applying stop words removal for this representation as it helps to reduce the size of the vocabulary. We are also removing *Hapax legomenon* as we believe it is not relevant for the quality of the translation and also helps to reduce the vocabularies' size.

The input given to the SVR model, in this case, is the concatenation of the two binary vectors corresponding to an English sentence and its German translation.

We tried to build different vocabularies by incorporating bi-gram and tri-gram in it. The idea behind this was to take context into account. However as we can see in *Table 2* the uni-gram vocabulary gives better results.

Vocabulary	Pearson	Rmse
Uni-gram	0.0562	0.8762
Bi-gram	0.0298	0.8815
Tri-gram	0.0290	0.8811

Table 2: Best results using SVR on Bag of Words

3.1.3 TF-IDF

We are using the `TfidfVectorizer` from `sklearn` to create a TF-IDF representation. Specifically, we are using the training sets to learn the Vocabularies and the $IDF_{w,D}$ of the two languages. We have also tried to consider as words not only each unigram but also bigrams and trigrams in order to add context.

Each sentence d can be represented as a vector of the $TF\text{-}IDF_{w,d,D}$ terms, where w are the uni-grams, bi-grams and tri-grams of d and D is the corpus. However, when using uni-grams, bigrams and tri-grams we need to use vectors with sizes of 355108 to represent an English sentence and its translation. Training on such datasets becomes extremely slow. In the case of the SVR model, we can solve this issue by averaging over the TF-IDF terms of one sentence. Therefore the input of our model would now be two numbers: the mean of TF-IDF terms for the English sentence and the mean of TF-IDF for the corresponding German translation.

Table 3 indicates the performance of SVR on the means of the TF-IDF terms. We can observe that TF-IDF achieves a decent Pearson Correlation score. Here, it is also important to notice that with the tri-grams we achieve the best results and that removing the stop words degrades the performance (in terms of Pearson Correlation score) of the TF-IDF considerably.

Vocabulary	Stop-words	Pearson	Rmse
Uni-gram	Not Removed	0.0670	0.8816
Bi-gram	Not Removed	0.0718	0.8816
Tri-gram	Not Removed	0.0724	0.8815
Tri-gram	Removed	0.0411	0.8816

Table 3: Best results on the validation set using SVR on TF-IDF terms

3.2 Embeddings

We have also used pre-trained embeddings as a way to represent the sentences. Specifically, we have used `spacy` to map words into pre-trained embeddings. We have tested both Glove (Pennington et al., 2014) and Bert (Devlin et al., 2018) embeddings. Specifically for Glove we have used `en_core_web_md` for English and `de_core_news_md` for German and for Bert `en_trf_xlnetbasecased_lg` for English and `de_trf_bertbasecased_lg`.

For Glove, since our sentences do not have the same number of words, we need to modify our input so it has a fixed dimensionality. We have examined two main approaches. Firstly, we can create **sentence level embeddings**. To elaborate, we average over the embedding of every word in the sentence. This creates one vector for every sentence. Then we stack those vectors to create one vector that represents an English sentence and its German translation. For instance, in the case of Glove, we stack the vectors of the English and German sentences, which has 300 features, to create one large vector with length 600.

Alternatively, we can create **word level embeddings**. This means that we create matrices that have as rows the words of the sentences and as columns the features of the embedding. The rows in the matrices are padded to the maximum length of both German and English sentences in the training set with zeros. Sentences on the test and validation set with more words than the maximum length observed in the training set are truncated appropriately.

Bert embeddings do not have the issue of the non-fixed dimensionality. The reason for this is that the pre-trained embeddings receive as input a sentence and for each sentence they output a fixed vector of length 768.

When using Glove embeddings we have observed that it performs better in our models when coupled with stop-words, punctuation removal and lower-casing. However, with Bert, the performance seems to be better when only removing punctuation. The reason for this is that we are using the case-sensitive Bert embeddings and therefore, removing stop-words and case may affect the context.

The baseline code we were given preforms SVR on the mean of the Glove embeddings of each sentence and achieves a Pearson correlation of 0.05486 on the validation dataset.

4 Architecture of the Models

In this section, we will describe the models we developed for the embedding representation. We have focused on this representation as it was performing the best on the SVR model. Even if other techniques give similar performances with SVR, we believe that the embeddings have more potential than the other representations and given the limited time, we chose to emphasise on them. For all our model we are using an Adam optimiser.

4.1 MLP

Firstly, we have built a Multi-layer Perceptron as it is a popular way to start experimenting with Neural Networks. The input of the MLP is a sentence-level embedding, as described in the previous section.

Initially, our architecture had 4 layers with ReLu activation after each layer, except the output layer. We were using Relu to introduce non-linearities to the network. The output layer must be a linear one since we have a regression problem. This architecture was overfitting the training set rapidly. Specifically, within 10 epochs the Pearson Correlation on the training set was around 0.9, while the Pearson Correlation on the validation set was extremely low and kept decreasing with more epochs.

To solve this problem, we had to add dropout layers. After experimentation, we realised that removing 2 hidden layers and reducing the number of neurons was helping the model generalise. The final architecture can be found in the file `NN.py` and is named `MLP`. Depending on the embedding we are using different networks declared in this file. For the *BERT* embedding it is called `BertMLP`.

4.2 CNN

We designed the CNN model to allow us to use Glove word-level embedding representations and apply the idea of n-grams through the convolutions. As we mentioned, we model the sentences as two-dimensional matrices. We stack those matrices to create a three-dimensional matrix. Basically, we treat the English sentence and its German translation as different channels of the same image. This acts as the input to our CNN.

We tried different architectures with this model since the kernel size of each convolutional layer acts as an N-gram representation. For example, a kernel size of two will have for effect to look for the correlation of the two-grams of each language. However, although it takes context into consideration, the convolutional layers look for N-grams that are at the same position in each sentence. Therefore, if the structure of the sentences is different, it will draw a comparison of different words.

After experiments we decided to apply 3 convolutional layers with kernel (2, embedding_size), (3,

Embeddings	Preprocessing	Model	7-fold CV		Validation Set		Codalab Competition	
			Pearson	RMSE	Pearson	RMSE	Pearson	RMSE
Glove	Stop-words & punctuation removal, lowercase	MLP	0.0830	-	0.1375	0.8568	0.0760	0.9387
Bert	Punctuation removal	MLP	0.1078	-	0.1622	0.8565	**0.1404**	0.9352
Glove	Stop-words & punctuation removal, lowercase	CNN	0.0678	-	0.1028	0.8826	0.0743	0.9750
Glove	Stop-words & punctuation removal, lowercase	ARC-I	0.1014	-	0.1061	0.8603	0.0736	0.9402

Table 4: Results summary

embedding_size) and (4, embedding_size) respectively. Each convolutional layer is followed by a ReLu activation function and a MaxPoolingLayer. We then concatenate the results of these layers and pass them through a Linear layer that will output the prediction.

4.3 ARC-I

Another architecture we have tried is ARC-I (Hu et al., 2014). This architecture is based on the word level Glove embeddings. However, instead of stacking the German and the English matrices as different channels, we treat them as separate matrices. The idea here is that for each matrix (corresponding to a sentence) we perform 1D convolutions (where the embedding dimensions are considered the channels). From those 1D convolutions, we extract 2 low-level representations, one corresponding to the English sentence and one for the translation. Afterwards, we use an MLP to compare the representations of the two sentences. Conceptually, the benefit here over the previous CNN architecture is that we do not directly compare the English and German sentences in the same positions, as they may have different structures. On Figure 1 we can see a high level overview of ARC-I

5 Hyperparamters and Training

Since we have a small training dataset, we decided to use cross-validation to select the best hyperparameters for each model in a statistically robust manner. We are performing a random search with cross-validation from `sklearn` (Pedregosa et al., 2011) to pick the hyperparameters of our models. Also, let's note here that when choosing the final number of training epochs we are performing early stopping to minimise the effect of overfitting.

An overview of the hyperparamets used for the various model can be seen in Table 5.

6 Results

In Table 4 we can see an overview of the best results we got with different combinations of neural models and embeddings. In the first place, we can observe that the MLPs (even the one with Bert) outperform the networks that use convolutions. This, in part, may be attributed to that, due to time limitations, the Randomised Search we had to perform with CNN and ARC-I was more limited than the one with the MLPs and therefore, the MLP had more optimised hyperparameters. Secondly, the MLP with the Bert embeddings significantly outperforms every other model. This is to be expected since we are using contextual embeddings that have been trained with a state-of-the-art technique on massive datasets and are able to identify the nuances of both the English and German languages.

7 Going Further

Provided that we had a short time-frame to tackle this problem, there are still different approaches that we would have liked to investigate.

Firstly, after ARC-I (Hu et al., 2014) it would have been interesting to experiment with ARC-II, which is a significantly more complex architecture, but generally performs better.

Moreover, apart from Glove (Pennington et al., 2014) and Bert (Devlin et al., 2018) trying different embeddings such as FastText (Bojanowski et al., 2016) and even combining them would be another path that could be explored. Additionally, fine-tuning Bert by training it on our dataset is something that could further improve the performance of our models.

Lastly, since TF-IDF seems to be the most promising alternative to embeddings, it could also be intriguing to modify and fine-tune our models to work with TF-IDF matrices as input.

References

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2042–2050. Curran Associates, Inc.

Edward Loper and Steven Bird. 2002. Nltk: The natural language toolkit. In *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

A Appendices

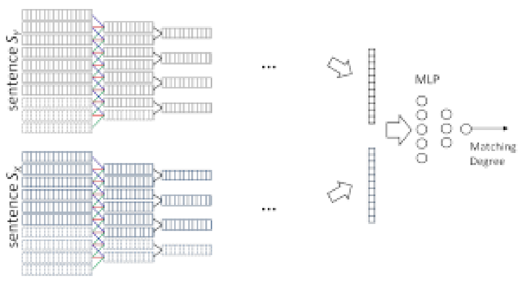

Figure 1: ARC-I. Image from (Hu et al., 2014).

B Supplementary Material

Our code can be found in the following repository `https://github.com/PierreElm/NLP-QualityMT`

On Table 5 we can see the hyperparameters used in our experiements:

Model	Learn. rate	Weight Dec.	Epochs	Batch Size
MLP	0.00646	0.0108	15	700
BertMLP	0.003491	0.001	50	256
CNN	0.001	0.001	10	256
ARC-I	0.0004	0.01	10	700

Table 5: Hyperparamets used to produce our results.