UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Hain Zuppur

Aligning contextual vector spaces between independent neural translation systems

Master's Thesis (30 ECTS)

Supervisor(s): Mark Fišel, PhD

Tartu 2023

Aligning contextual vector spaces between independent neural translation systems

Abstract:
Numerous pre-trained machine translation models are available for translating between different languages. However, these models are limited to a fixed set of languages they were trained for. When there is no translation model available for a specific language pair, we need to translate to one or more intermediate languages, which can result in reduced translation quality. We investigate the possibility of combining two translation models by aligning the vector spaces between them using a simple regressor. We explore the effectiveness of various regression methods for achieving this alignment and evaluate their performance. We show that combining two different translation models is possible, although doing so leads to a decrease in translation quality.

Keywords:
natural language processing, neural machine translation, vector space transformations

CERCS: P176 Artificial intelligence

Kontekstuaalsete vektorruumide joondamine erinevate neuromasintõlke süsteemide vahel

Lühikokkuvõte:
Erinevate keelte vahel tõlkimiseks on saadaval arvukalt eeltreenitud masintõlke mudeleid. Need mudelid on aga piiratud kindla keelte komplektiga, mille jaoks nad on treeniti. Kui konkreetse keelepaari jaoks pole tõlkemudelit saadaval, peame tõlkima ühte või mitmesse vahepealsesse keelde, mis võib põhjustada tõlkekvaliteedi halvenemist. Uurime kahe tõlkemudeli kombineerimise võimalust, joondades nendevahelised vektorruumid lihtsa regressori abil. Uurime erinevate regressiooni meetodite tõhusust selle joondamise saavutamiseks ja hindame nende effektiivsust. Näitame, et kaht erinevat tõlkemudelit on tõepoolest võimalik kombineerida, kuigi see viib tõlkekvaliteedi languseni.

Võtmesõnad:
loomuliku keele töötlus, tehisnärvivõrkudel põhinev masintõlge, vektorruumi teisendused

CERCS: P176 Tehisintellekt

Contents

1 Introduction

Machine translation (MT) is used to translate text from one language to another. Although human translators can offer superior translations in terms of quality, MT systems are faster and cheaper. Modern MT solutions are based on transformers [1], which consist of an encoder and a decoder and have shown much better results than previous approaches. These systems consist of a tokenizer and a transformer-based model. Tokenizer splits the text to be translated into tokens, which are given to the transformer-based model. The model's encoder produces embeddings for the tokens, which are used to generate the translated text. There are many pre-trained neural machine translation (NMT) models freely available; for example, at the time of writing, the Hugging Face [2] open-source library and community has over 2000 transformer-based NMT models.

When no translation model is available for a specific language pair, we need to translate to one or more intermediate languages, which can result in reduced translation quality. We investigate the possibility of combining two translation models by aligning the vector spaces between them using a simple regressor.

The aim of this thesis is to see if we can combine the encoder from one transformer-based model with the decoder of another model by training a simple regressor to align the token embeddings. We take the token embeddings from one model produced by one model encoder, transform them with our regressor and then generate the translated text using the decoder from a different model.

Training these MT models is time-consuming and requires powerful hardware and a lot of training data. If it is possible to combine transformer models encoders and decoders in this way, then it allows us to easily create new MT models for specific cases by combining different pre-trained models. This would allow us to save on training time because we do not need to train a new translation model.

Our aim is to create a system for generating the matching token embeddings between two transformer-based NMT systems. Then, we will use these matching token embeddings to train different regression methods. We will conduct experiments to find the best regression model, and measure the performance in different situations. Finally, we will use the best regression method to transform token embeddings from one model encoder and then generate the translation with these transformed embeddings using the decoder from a different model.

We are searching for answers to the following questions.

- How well do different regression models (linear regression, orthogonal regression and multi-layer perceptron) work for mapping the token embeddings?
- How well does the aligning work on know languages and unknown languages and between models with different architecture?
- How much translation quality do we lose with this mapping?

In the next chapter, we give a brief overview of the concepts and technologies we are going to use. Chapter 3 goes over the regressor training data generation process, the different regression methods we are going to use and finally, how we will generate the translated text with the aligned token embeddings. In chapter 4, we will go over the data and models we used and aim to answer the questions we proposed here with experiments. Finally, in chapter 5 we will discuss our results and ideas for future work.

2 Background and Related Work

This chapter gives a short overview of the concepts and technologies used in this thesis. We will explain what transformers are and why they are used for MT. We will go over different tokenization methods, Byte-Pair Encoding (BPE), Unigram and SentencePiece, and explain why tokenization is used in natural language processing (NLP). In addition to this, we explain how the similarity between vectors is measured and how is the quality of translations measured. And finally, we will go over adapters, which could also be used to combine the encoder from one model with the decoder from a different model.

2.1 Transformers

The current state-of-the-art MT models are based on the transformer architecture introduced by Ashish Vaswani *et al.* [1]. In the Seventh Conference on Machine Translation [3], all of the winning MT models were based on some variation of transformers [4, 5, 6, 7, 8, 9]. The previous approaches for MT used recurrent neural networks (RNNs) and long short-term memory (LSTM), but these systems process the input sequentially, which becomes a problem with longer sequence lengths [1]. The architecture of the transformers is shown in Figure 1.

The transformers consist of these key components: encoder and decoder, multi-head attention, positional encoding and transformer layers.

The transformers consist of an encoder and a decoder. The encoder takes the tokenized input sentence and outputs a list of vector embeddings [10]. Then the decoder uses these vector embeddings to generate the output one token at a time until the end-of-sentence symbol (EOS) is produced [10].

Ashish Vaswani *et al.* explain that transformers use multi-head attention. Using multi-head attention, the model can attend to information from multiple representation subspaces at different positions simultaneously. The multi-head attention is used in transformers in three different ways. Firstly it is used in the encoder-decoder attention layers, where the queries come from previous decoder layers while the memory keys and values are obtained from the encoder's output. This setup enables each position within the decoder to attend to all positions present in the input sequence. Secondly, multi-head attention is used in the self-attention layers of the encoder. In the encoder,

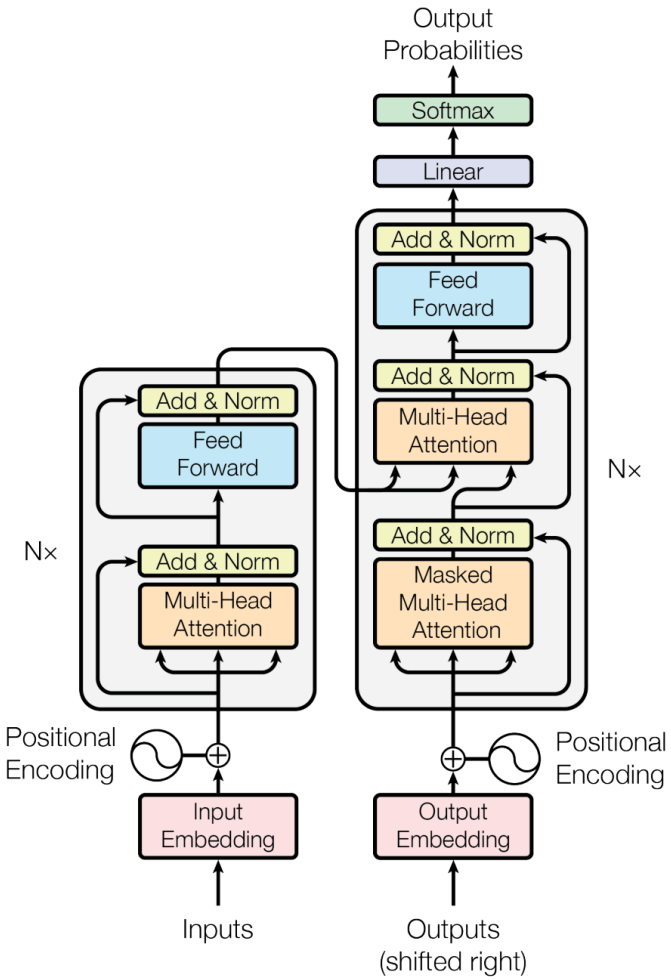

Figure 1. The Transformer - model architecture. [1]

the self-attention mechanism allows the encoder in each position to attend to all of the positions in the previous layer. Thirdly, multi-head attention is used in the self-attention layers of the decoder. In the decoder, self-attention enables each position in the decoder to attend to all positions up to and including its own position. The authors explain that this is done to maintain the auto-regressive property; it is necessary to restrict the flow of information leftward in the decoder.

According to its authors [1], the positional encoding is added to the inputs at the start of the encoder and decoder stacks. The authors explain that because transformers can process the sequence in parallel, it is important to add positional encoding, so the model

can make use of the order of tokens in the sequence.

Transformer's authors [1] explains that the encoder and decoder both consist of a stack of transformer layers. In the encoder, each transformer layer consists of two sub-layers, the first is the multi-head self-attention mechanism, and the second is a fully connected feed-forward network. After each of these sub-layers, there is a residual connection and layer normalization. The transformer layers in the decoder consist of three sub-layers: the multi-head self-attention mechanism, fully connected feed-forward network and multi-head attention over the output of the encoder. The authors explain that, similarly to the encoder, there is residual connection and layer normalization after each sub-layer in the decoder layers.

Transformers have revolutionized the field of MT by enabling faster and more efficient processing of text data. Along with the use for MT, transformers are also widely used in other NLP tasks, computer vision (CV) [11] and audio processing [12].

2.2 Tokenizers

An essential pre-processing step for NLP is tokenization, which splits text into small units called tokens [13]. There are multiple ways to tokenize text. Earlier approaches focused on word-level tokenisation, splitting text into tokens, where each token is a word [14].

Sabrina J. Mielke *et al.* [14] explained that early word-level tokenizers saw words as "space-separated sub-strings", but this causes problems for words with punctuation, for example, the word *"don't"*. If we split on punctuation, we get three tokens *"don"*, *"'"*, *"t"*, but a alternative way to tokenize it would be tokens *"don"* and *"'t"*. Authors explained that in recent years, this type of tokenization has fallen out of favour because many neural language models such as BERT require a fixed size vocabulary and word level tokenization with fixed size vocabulary will replace unknown words with new token UNK (unknown) and the features of the unknown word will be lost.

Sub-word-based tokenization aims to solve the fixed vocabulary problem by splitting rare and unknown words as sequences of sub-word units [15]. One approach is BPE, introduced by Rico Sennrich *et al.* [15]. The authors [15] explain that BPE works by first initialising the symbol vocabulary with the character vocabulary. This way, each word is represented as a sequence of characters and a special end-of-word symbol to be

able to restore the original sentence. Now we iterate over the training data and count all of the symbol pairs and replace each occurrence of the most frequent pair ("A", "B") with a new symbol "AB". This process is repeated until we reach the predefined symbol vocabulary size. Authors describe that with this approach, common words will be one token and unknown words will be represented by sub-words and features of the unknown word will not be lost.

According to Taku Kudo [16], the main flaw with BPE is that the same sentence can be encoded in different ways, and this can cause issues for NMT. The word "Hello" could be encoded in different ways, for example, it could be represented by the sub-words: *"Hell/o/"*, *"H/ello"* and *"He/llo/"*. To solve this problem, author [16] proposed a new tokenization algorithm called Unigram. Taku Kudo [16] explains that this algorithm works by initializing a large base vocabulary containing all of the characters and the most frequent sub-strings in the training corpus. Then, for each sub-word in the vocabulary, it is computed how much the overall loss would increase if the sub-word was removed. After this, the sub-words in the vocabulary are sorted by their loss value, and the top n % (usually 80 %) are kept, and the rest are deleted. The one-character sub-words are never removed to avoid not being able to tokenize some unknown words. This process is repeated until we reach the desired vocabulary size. The author [16] says that unigram tokenizers perform better than BPE, especially for low-resource and open-domain settings.

All of the above-mentioned tokenization methods have been focused on languages where words are separated by whitespaces, but words are not separated by whitespaces in all languages, for example, Japanese, Korean and Chinese [17]. This can be solved by pre-processing the text with language-specific tokenizers to split the text into words and then using the pre-possessed text as input for the tokenization method [18]. But this kind of approach creates additional complexity, as it requires pre-processing and post-processing the text and managing multiple tokenizers for different languages [17].

SentencePiece tokenizer proposed by Taku Kudo *et al.* [17] combines the BPE and Unigram to create an end-to-end system that does not need any language-specific pre or post-processing. The authors of SentencePiece [17] describe that this tokenizer consists of four parts: normalizer, trainer, encoder, and decoder. The normalizer normalizes the Unicode characters into canonical forms in the corpus. Then the normalized corpus

is used by the trainer to train the sub-word segmentation model. The encoder uses the normalizer and the sub-word segmentation model to tokenize the input, and the decoder is used to detokenize. The authors [17] explain that SentencePiece is a self-contained tokenizer that can be used for any combination of languages and provides perfect repeatability of the sub-word segmentation.

All of the MT models used in this thesis use the SentencePiece tokenizer.

2.3 Measuring Vector Similarity

One of the most popular ways to measure the similarity of two vectors is cosine similarity. It is used in different natural language processing tasks such as information retrieval [19], text classification [20] and text clustering [21]. Cosine similarity is calculated by measuring the cosine angle between two vectors [22]. Given two vectors A and B, the cosine similarity is measured by using the following formula:

$$Cosine(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{1}$$

Cosine similarity has several desirable properties that make it useful for comparing vectors in high-dimensional spaces. Firstly, cosine similarity only depends on the direction of the vectors, not their length [21]. This property is particularly useful in natural language processing tasks, where the length of the vectors may vary.

Secondly, cosine similarity is bounded between -1 and 1; if the similarity between vectors is -1, they are opposite of each other; if the similarity is 0, they are orthogonal, and if the similarity is 1, they have the same direction [22]. This makes interpreting the cosine similarity score easy and a good measure of similarity between vectors.

Finally, cosine similarity is computationally efficient and easy to compute, which makes it a popular choice for many natural language processing tasks [20].

Another way to measure vector similarity is Euclidean distance. Euclidean distance is used for many different applications, for example, image recognition[23], language feature extraction[24], and face recognition[25].

The Euclidean distance is not affected by how far the vectors are from zero; for example, the distance between a vector $[1, 1, 1]$ and vector $[2, 2, 2]$ will be 1.732, but the

distance between vectors $[100, 100, 100]$ and $[200, 200, 200]$ is 173.205, which is much larger. If we want the Euclidean distance to be relative to how far the vectors are from the zero point, we can divide the distances between the two vectors by the distance from one vector to the zero vector. By doing this, the relative distance between the pair of vectors $[1, 1, 1]$, $[2, 2, 2]$ and $[100, 100, 100]$, $[200, 200, 200]$ will be the same. The relative Euclidean distance is calculated by the following formula:

$$RelativeEuclideanDistance(A, B) = \frac{\|A - B\|}{\|A - \vec{0}\|} \tag{2}$$

If two vectors are the same, then the Euclidean distance between them would be zero; if the vectors are not the same, then the smaller the distance between the vectors, the more similar they are. While the cosine similarity is bounded between -1 and 1, the Euclidean distance has no bound; its values can be from 0 to infinity. Because of this reason, in the experiments, we used cosine similarity as the main similarity metric.

2.4 Translation Quality Evaluation

An important step in developing MT systems is evaluating the translation quality. Human evaluation can provide the necessary feedback, but it can be time-consuming, expensive and subjective [26]. Because of these drawbacks, human evaluation is only feasible on a small scale. To provide more consistent, quicker and cheaper feedback on the translation quality, many automatic evaluation metrics have been developed [26].

One of the most popular automatic evaluation metrics is BLEU [27]. According to its authors [27], BLEU uses modified n-gram precision to evaluate translation quality against one or multiple reference translations. The precision is calculated by counting the n-gram in a reference translation and then clipping the total count of the n-gram in the candidate translation to the maximum reference count and then divided by the candidate translation's total n-gram count. This is done to penalize the repeating of n-grams in the candidate translation. The n-gram precision already penalizes candidate translations that are longer than reference translation, but it does not penalize shorter translations. The authors explain that to mitigate this, there is a brevity penalty factor to ensure that a high-scoring candidate translation matches the candidate translation in length.

Deborah Coughlin [26] found that the BLEU score correlates with human evaluations,

especially on larger datasets. With BLEU, we can reliably and easily validate the translation quality for the MT system.

2.5 Related Work

One approach to change the embedding of a trained transformer model is adapters proposed by Neil Houlsby *et al.* [28]. Adapters are new transformer layers added between layers of a pre-trained network. When the adapters are tuned, the other layers are frozen, meaning they won't be changed. This means we have to train fewer parameters, so the models are smaller and need less training data. Its authors explain that adapters are useful when we need to train a model to be suitable for multiple different downstream tasks because we can just have one copy of the model and then adapters for each task.

Mikel Artetxe *et al.* [29] showed that it is possible to use adapters to transform a transformer-based model to a different language by learning an embedding matrix. Adapters were not used in this thesis to see if it is possible to transform embeddings with a simple regression model.

3 Method

This chapter gives an overview of how the token embeddings were obtained for training the regressor to transform the embeddings, which regressors were used and how the translated text was generated using the transformed embeddings. This process consists of three main steps, obtaining the training data, training the regressor and translating.

Firstly, we find the tokens that are split the same way between the tokenizers and then extract the token embeddings from the encoders for these matching tokens. Then using these token embeddings, we train a regressor to transform embeddings from model ***A*** to be more similar to model ***B*** embeddings. And finally, to translate a sentence with our trained regressor, we tokenize the sentence with model ***A*** tokenizer and use these tokens to obtain the token embeddings model ***A*** encoder. Then these token embeddings are transformed using our trained regressor, and then using the transformed embeddings, we generate the translated text using the model ***B*** decoder. An overview of the translation process using the transformed embeddings is shown in Figure 2.

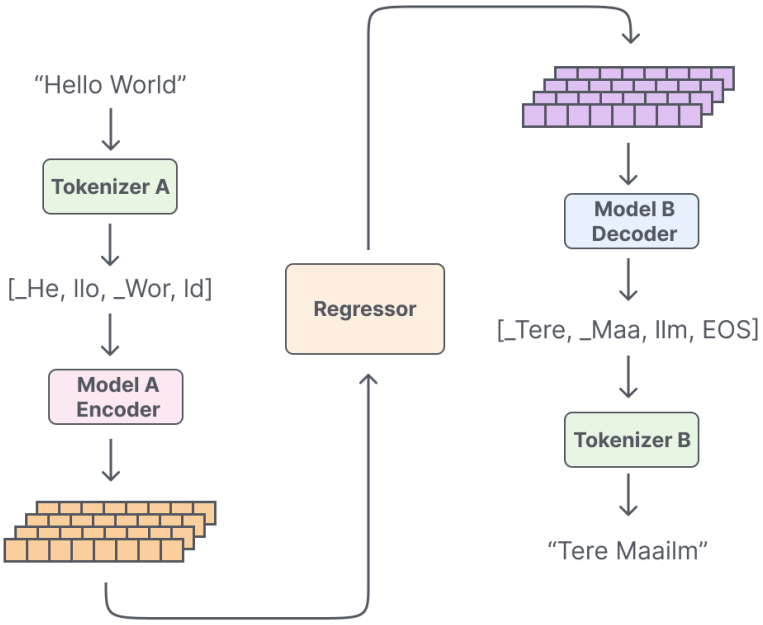

Figure 2. Translating text with the Model A embedding transformed to Model B embeddings with the regressor.

3.1 Getting the Training Data

To be able to train and evaluate regression models to transform one model embeddings into a different model embedding, we need to find tokens that are the same for both models. The process used to obtain the training data is shown in the Figure 3.

Tokenizers split the input string into tokens and some words might consist of multiple tokens. Because of this, independently trained tokenizers might split the same word in a different way; for example, the Swedish word "ordet" is split into two tokens "or", "det" by opus-mt-NORTH_EU-NORTH_EU[1] model tokenizer, but opus-mt-SCANDINAVIA-SCANDINAVIA[2] model tokenizer encodes the word into one token "ordet"

To be able to train models to convert embeddings obtained from model ***A*** to be potentially compatible with model ***B***, we need to find the tokens that are split the same way by both models ***A*** and model ***B*** tokenizers.

We tokenize a sentence and then look at how each word is split into tokens. Then we iterate over the tokens to find the id-s of the matching tokens; these id-s are later used to extract the correct embeddings. If tokens do not match, we discard that and the rest of the tokens in that word.

After we have obtained the id-s of the tokens that are the same for both tokenizers, we give the full tokenized sentence to both models and the list of id-s to extract the correct embeddings. We now perform a forward pass for both models, and the sentence embedding is extracted from the last hidden layer of the encoder. From the sentence embedding, we can extract the matching token embeddings from both models using the id-s we got from the earlier step.

Now we have the embeddings from model ***A*** and model ***B*** from one sentence that are from the same tokens. This process is repeated to get the desired amount of embeddings for training the regressor.

[1] opus-mt-NORTH_EU-NORTH_EU webpage: https://huggingface.co/Helsinki-NLP/opus-mt-NORTH_EU-NORTH_EU

[2] opus-mt-SCANDINAVIA-SCANDINAVIA webpage: https://huggingface.co/Helsinki-NLP/opus-mt-SCANDINAVIA-SCANDINAVIA

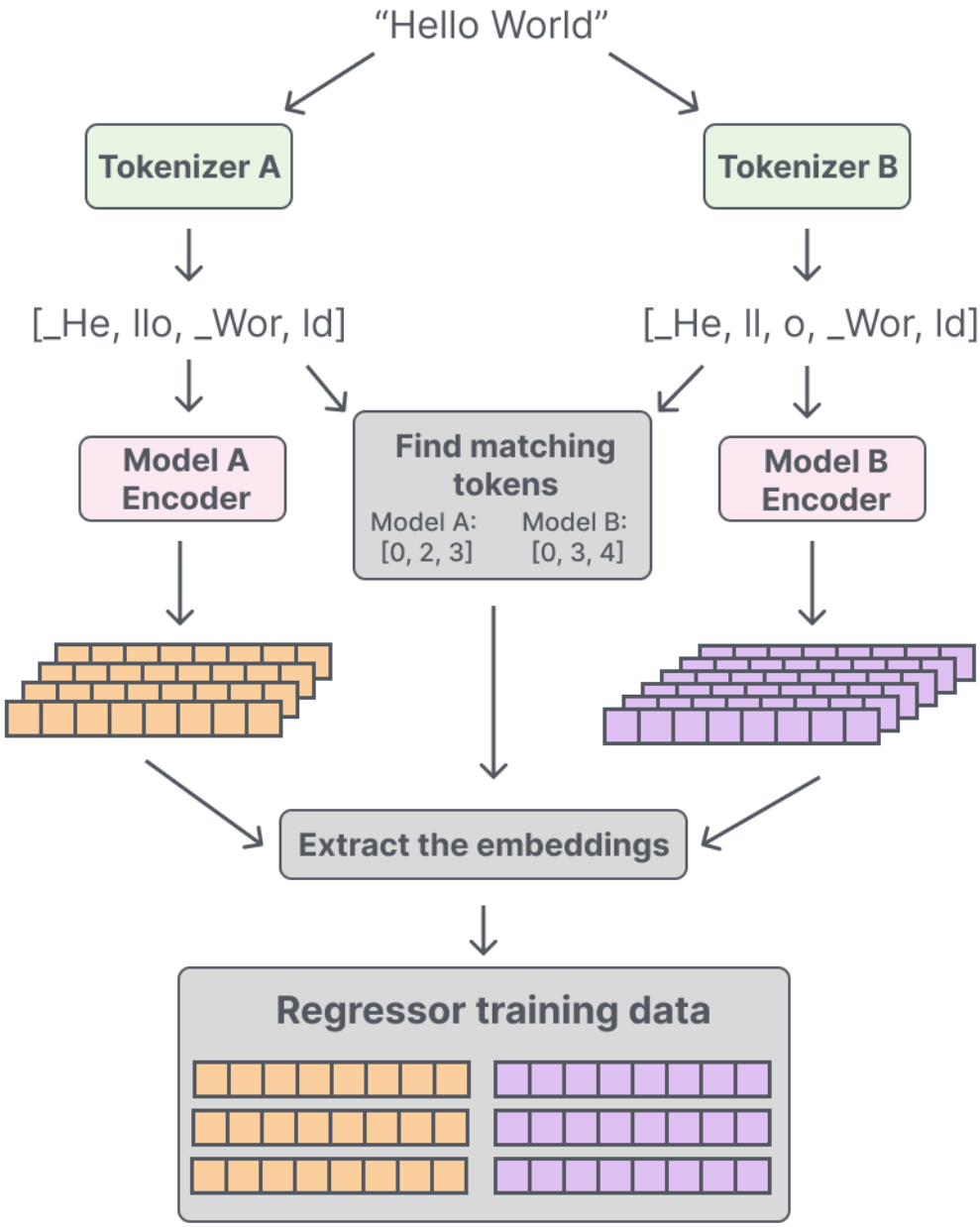

Figure 3. Overview of the regressor training data obtaining process.

3.2 Transforming the Embeddings

To transform the embeddings, three different regression algorithms were chosen: linear regression, orthogonal regression and multi-layer perceptron.

Linear regression was chosen to see if there is a linear relationship between token embedding from different models. Linear regression is a common method for transforming vectors into a different vector space. Linear regression is a simple and efficient method that can capture linear relationships between variables. The scikit-learn[3] implementation of linear regression was used in this thesis.

Orthogonal regression is a linear regression with the additional constraint that the weight matrix must be an orthogonal matrix/projection. Chao Xing *et al.* [30] found that using orthogonal transformations on word vectors performed better than using linear transformations. Orthogonal regression was chosen to see if the word embeddings transformations also improve when using orthogonal regression when compared to linear regression. The scikit-matter[4] implementation of orthogonal regression was used in this thesis.

Multi-layer perceptron was chosen because linear transformations may not be suitable for word embedding transformation because it assumes a linear relationship between the input and output variables. In contrast, a multi-layer perceptron (MLP) is a neural network model that can capture non-linear relationships between input and output variables. The scikit-learn[5] implementation of MLP was used in this thesis.

3.3 Generating Translated Text with Decoder

After we have trained a regressor to transform model $\boldsymbol{A}$ token embeddings to model $\boldsymbol{B}$ token embeddings, we can start translating.

Firstly we need to tokenize the sentence with model $\boldsymbol{A}$ tokenizer and pass the tokenized sentence to the encoder and perform the forward pass. After this, we can extract

[3] scikit-learn linear regression webpage: `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html`

[4] scikit-matte orthogonal regression webpage: `https://scikit-cosmo.readthedocs.io/en/latest/linear_models.html#orthogonal-regression`

[5] scikit-learn MLP webpage: `https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html`

the embeddings from the encoder's last hidden layer.

Then these embeddings obtained from model ***A*** encoder are transformed to model ***B*** compatible embedding using the trained regressor. The model-specific special tokens, like the target language token and EOS, are not transformed with the regressor. These special tokens are obtained by passing the tokenized sentence to model ***B*** and performing the forward pass to obtain these embeddings from the last hidden layer of the model ***B*** encoder. Then the sentence embeddings obtained from the regressor are combined with the special token embeddings to form the token embeddings used as the input for the decoder.

Now we start generating the translated text using the greedy generation algorithm. We give the token embeddings to the decoder, set the decoder input id-s to the beginning-of-sequence (BOS) token, and perform a forward pass. After this, we extract the last hidden state of the decoder and give it to the model head to obtain the scores for each token. From the model head output, we take the token with the highest score and add it to the list of decoder input id-s.

Now we repeat the process with the new decoder input id-s until we generate EOS token or reach the arbitrary iteration limit set by the model configuration.

In natural language processing, beam search is a popular method for generating text in a more diverse and accurate manner than a greedy generation. However, for this thesis, greedy generation was chosen over beam search due to ease of implementation.

One of the main advantages of greedy generation over beam search is that it is a simpler and more straightforward method; it simply selects the word with the highest probability at each step of generation without considering any other possible options. [31] This makes it easy to implement and requires fewer computational resources, which can be particularly beneficial when working with limited computing power.

3.4 Tools used for writing this thesis

In this thesis, Grammarly[6] was used to fix grammatical mistakes and improve sentence structure. It reviews spelling, punctuation, grammar, clarity and delivery mistakes in the text.

[6]Grammarly is a cloud-based typing assistant. Grammarly's webpage: `https://grammarly.com`

4 Experiments

In this section, we aim to answer the questions proposed in the introduction:

- How well do different regression models (linear regression, orthogonal regression and multi-layer perceptron) work for mapping the token embeddings?
- How well does the aligning work on know languages and unknown languages and between models with different architecture?
- How much translation quality do we lose with this mapping?

4.1 Choosing the Best Regression Method

To be able to evaluate different regressors' performance, we need to make sure we have sufficiently trained the different regressors. We can be sure that the regressor is sufficiently trained when increasing the training data amount does not improve the performance of the model.

For these experiments, the OPUS-MT [32] machine translation models opus-mt-NORTH_EU-NORTH_EU[7] and opus-mt-SCANDINAVIA-SCANDINAVIA[8] were used with the OpenSubtitles open-source parallel corpora from movie and TV subtitles[33] using the Swedish sentences from the Danish-Swedish corpus. Both of these models are multi-lingual and share the Swedish language.

From the OpenSubtitles Danish-Swedish corpus [33], 500 thousand sentences were used to generate the training data. For both of the machine translation models, the embeddings of tokens that were split the same way between both of the models were extracted. After this process, 2.14 million token embeddings were obtained.

For each amount of training data, we randomly took n amount of token embeddings for the training set and 10 000 token embeddings as a testing set, making sure the test and train set do not overlap. Then we trained the regressor with the training set and evaluated its performance on the test set using cosine similarity. This process was repeated ten

[7] opus-mt-NORTH_EU-NORTH_EU webpage: `https://huggingface.co/Helsinki-NLP/opus-mt-NORTH_EU-NORTH_EU`

[8] opus-mt-SCANDINAVIA-SCANDINAVIA webpage: `https://huggingface.co/Helsinki-NLP/opus-mt-SCANDINAVIA-SCANDINAVIA`

times for each train data amount, and the average from all of the runs was taken. This was done to avoid random variance on different train and test sets. These experiments were run on the University of Tartu high-performance computing centre [34].

4.1.1 Linear and Orthogonal Regression

Linear and Orthogonal regressors stopped improving their score meaningfully after being trained on 6 400, as shown in Figure 4. These results mean that these regressors can be fully trained with only token embeddings obtained from 1 500 sentences. From this experiment, we can also see that the orthogonal regressor performs better when we have very little amount of training data, but the linear regressor starts to outperform it from 800 tokens.

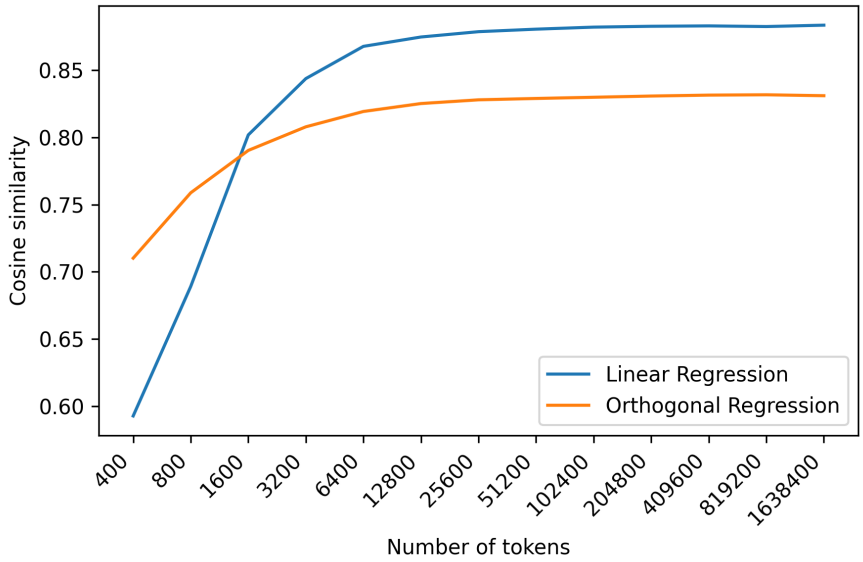

Figure 4. Training data amount effect on Linear Regression and Orthogonal Regression.

Linear regression performs better than orthogonal regression when we have more than 800 tokens to train the regressor. When both of these regressors are fully trained, the Linear regression outperforms the orthogonal regression by 6% when using cosine similarity as the performance measure, from orthogonal regression cosine similarity score of about 0.83 to linear regressions cosine similarity score of about 0.88.

4.1.2 Multi-Layer Perceptron

For Multi-Layer perceptron based regressors, we need to find out how many hidden layers we need and how many neurons in each of them. We train the different regressors with varying amounts of training data as described above.

Firstly we find out how many neurons we need for MLP with one hidden layer. We test six different MLP regressors with one hidden layer consisting of 256, 512, 1024, 2048, 4096 and 8192 neurons. Both input and output vector lengths are 512, so for the first regressor, the hidden layer size is half of the input and output size, the second regressor's hidden layer size is the same as the input and output, and the third model's hidden layer size is twice the input and output and so on.

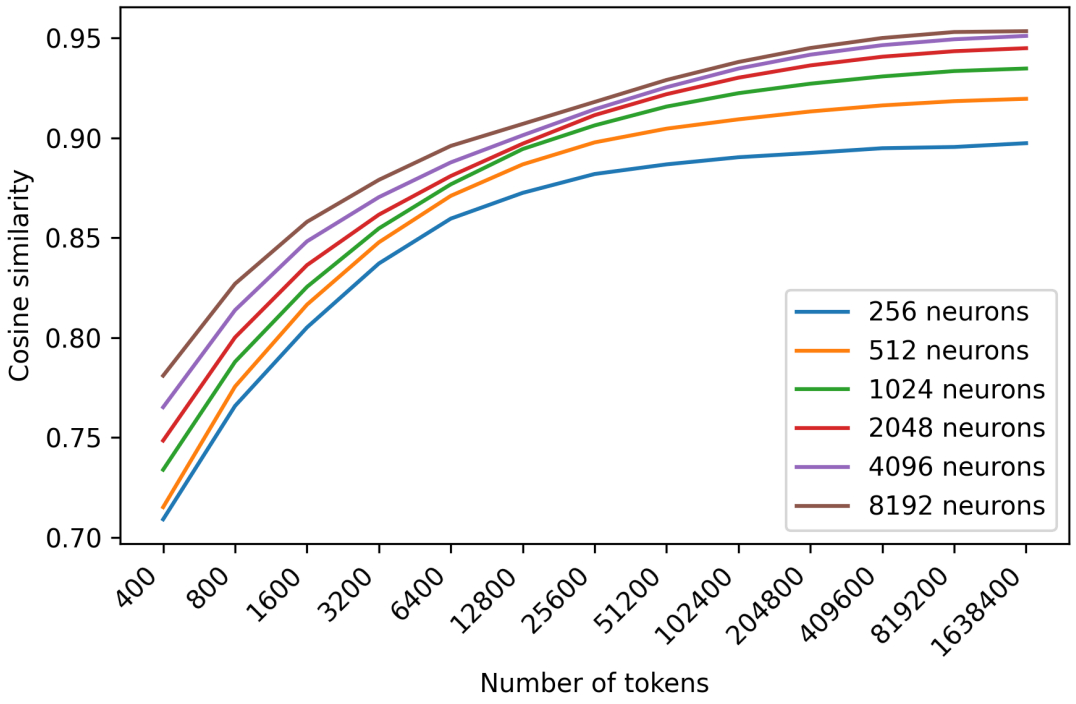

Figure 5. Effect of training data amount on MLP regressors with one hidden layer of varying size.

In the Figure 5, we can see the result for MLP regressors with one hidden layer. From this, we can see that larger hidden layer size results in better performance, even when

Table 1. One layer MLP regressors with different hidden layer sizes performance when training with 1 638 400 tokens.

Hidden layer size	**Cosine similarity**
256	0.897
512	0.920
1024	0.935
2048	0.945
4096	0.951
8192	0.953

the amount of training data is low. From the Table 1, we can see that the performance difference between the smaller hidden layers is quite noticeable, but the differences in performance between the two largest are minuscule. Because of this, it was decided not to test out larger hidden layer sizes. We can also see that the models stop improving when the amount of tokens used for training exceeds one million, so they were not trained further with more training data.

Table 2. MLP regressors with different amounts of hidden layers performance when training with 1 638 400 tokens.

Hidden layer size	**Cosine similarity**
One hidden layer (8192)	0.953
Two hidden layers (2420, 2420)	0.949
Three hidden layers (1800, 1800, 1800)	0.946

Secondly, we found out how will having multiple hidden layers affect the performance. The MLP regressor with one hidden layer with 8192 neurons has about 8.3 million parameters. The hidden layer size for the regressor with two hidden layers was chosen so that the parameter count would also be 8.3 million, and the same approach was used for the regressor with three hidden layers.

The results of this experiment are in the Figure 6. We can see that the one-layer MLP performed better than one and two-layer MLP on all training sizes. From the Table 2, we can see that the MLP regressor with one hidden layer achieved an increase of about 0.1

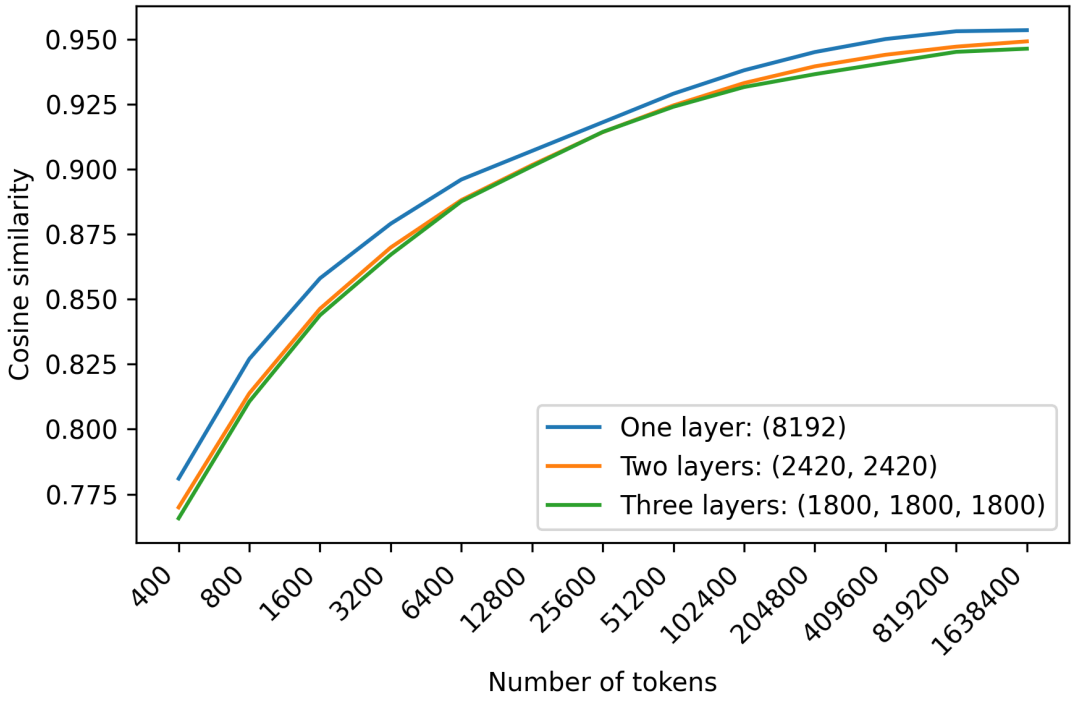

Figure 6. Effect of training data amount on MLP regressors with different amounts of hidden layers.

when compared to the regressors with two or three hidden layers.

4.1.3 Best Regression Method

From these experiments, we can see that the best regression method for transforming the embeddings is MLP. From the Figure 7 we can see that MLP outperformed linear and orthogonal regression on all different training data amounts. The Table 3 shows that the difference between the best results of MLP and linear and orthogonal regression is large, from average cosine similarity on testing data of 0.884 for linear regression and 0.831 for orthogonal regression to 0.953 for MLP.

We can also see that the average relative Euclidean distance measures correlate with the cosine similarity scores, the orthogonal regression showed the largest average distance, the linear regression distance scores were slightly lower, and the MLP average

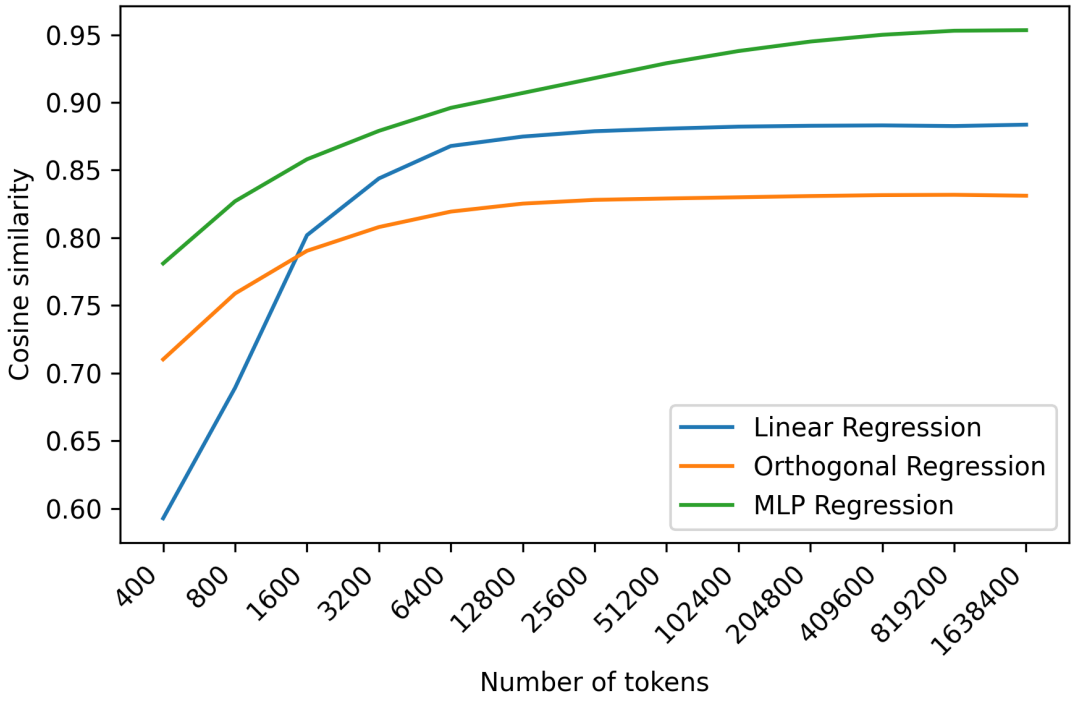

Figure 7. Different regression methods performance compared when training with different amounts of data.

Table 3. The best cosine similarity and relative Euclidean distance for all regression models.

Regressor type	**Cosine similarity**	**Relative Euclidean distance**
Linear regression	0.884	0.505
Orthogonal regression	0.831	0.537
MLP regression	0.953	0.292

relative Euclidean distance was the smallest. Based on these results, we used MLP regressor with one hidden layer with 8192 neurons trained with 1.6 million tokens for the rest of the experiments.

4.2 Aligning Embeddings

After finding out the best regression method for aligning the token embeddings, we can test the transformation quality on the language we used for training the regressor and on languages that are unknown for the regressor and test if the transformation quality will be affected when transforming between models with different architectures.

In these experiments, MLP regressor with one hidden layer with 8192 neurons was used to transform the model $\boldsymbol{A}$ token embedding and then calculate the cosine similarity between the transformed embedding and the embedding from model $\boldsymbol{B}$. For accessing the token embedding transforming quality, we used the FLORES-200 [35] dataset. For each language, we found the tokens that are the same between both of the models and used them to find the average cosine similarity for that language.

4.2.1 Known and Unknown Languages

To find out how the embedding transformation works on known and unknown languages, we used the same models from OPUS-MT [32] as in the previous chapter, the opus-mt-NORTH_EU-NORTH_EU[9] and opus-mt-SCANDINAVIA-SCANDINAVIA[10]. The regressor used in this experiment was trained using 1,6 million tokens obtained from Swedish sentences from the Danish-Swedish corpus of the OpenSubtitles open-source parallel corpora [33].

From the Table 4, we can see that the regressor performed the best on the Swedish language, which is expected, as the regressor was trained on tokens obtained from Swedish sentences. The regressor achieved an average cosine similarity of 0.896 on Swedish, which is 5.9 % higher than the second-best result of 0.846, which was achieved on the Norwegian Bokmål language. We can see from the results that the languages can be split into two groups based on the regressor performance. One group consists of Swedish, Danish, Norwegian Bokmål and Norwegian Nynorsk, and the second group is Faroese and Icelandic.

Anders Holmberg *et al.* [36] says that Scandinavian languages fall into two groups by

[9] opus-mt-NORTH_EU-NORTH_EU webpage: https://huggingface.co/Helsinki-NLP/opus-mt-NORTH_EU-NORTH_EU

[10] opus-mt-SCANDINAVIA-SCANDINAVIA webpage: https://huggingface.co/Helsinki-NLP/opus-mt-SCANDINAVIA-SCANDINAVIA

Table 4. Embedding transformation performance when using MLP regressor trained on Swedish language and tested on FLORES-200 dataset.

Language	**Cosine similarity with Standard deviation**	**Euclidean distance**
Swedish	0.896 ± 0.075	0.493 ± 0.217
Danish	0.843 ± 0.109	0.641 ± 0.318
Norwegian Bokmål	0.846 ± 0.108	0.645 ± 0.335
Norwegian Nynorsk	0.829 ± 0.117	0.686 ± 0.343
Faroese	0.726 ± 0.128	0.963 ± 0.360
Icelandic	0.717 ± 0.132	0.996 ± 0.360
Average	0.782 ± 0.137	0.813 ± 0.387

their syntax one is Danish, Norwegian and Swedish, and the second group is Icelandic, Old Scandinavian and Faroese. There are many syntactic differences between these languages, for example, Danish, Norwegian and Swedish have a system of subject-verb agreement morphology, while the Icelandic, Old Scandinavian and Faroese have no such system. The authors describe that the languages in the first group also have a rich system of case morphology, while the languages in the second group have an impoverished system of case morphology.

These syntax differences can explain the grouping of the languages in the Table 4. The regressor was trained using the Swedish language, so it makes sense that the languages Danish, Norwegian Bokmål and Norwegian Nynorsk that are syntactically more close to Swedish and transform better with our regressor.

From this experiment, we can also see that the type of text will also affect the performance. The regressor was trained on OpenSubtitles open-source parallel corpora from movie and TV subtitles [33] using the Swedish sentences from the Danish-Swedish corpus and from the Table 3 we can see, that the MLP regressor achieved average cosine similarity of 0.953. From the table 4, we can see that on the FLORES-200 [35] Swedish dataset, which contains sentences from web articles, the MLP regressor achieved an average cosine similarity of 0.896.

We can conclude that the aligning embeddings work better on known languages and on similar types of text. For unknown languages, the performance of the aligning will be

affected by how similar the language is to the language used for training.

4.2.2 Aligning Between models with different architecture

To find out how using models with different architecture affect the embedding transformation performance, we used the M2M100 418M [37] model and the OPUS-MT [32] opus-mt-NORTH_EU-NORTH_EU model. These models have a different amount of transformer layers, and the M2M100 418M model token embedding length is 1024, while the opus-mt-NORTH_EU-NORTH_EU token embedding length is 512.

Same as in the previous chapter, we trained the regressor using 1,6 million tokens obtained from Swedish sentences from the Danish-Swedish corpus of the OpenSubtitles open-source parallel corpora [33].

Table 5. Embedding transformation performance when transforming from m2m100 418M to opus-mt-NORTH_EU-NORTH_EU using MLP regressor trained on Swedish language and tested on FLORES-200 dataset. The performance difference is between the opus-mt-NORTH_EU-NORTH_EU to opus-mt-SCANDINAVIA-SCANDINAVIA transformation and m2m100 418M to opus-mt-NORTH_EU-NORTH_EU transformation

Language	**Cosine similarity with Standard deviation**	**Performance difference**
Swedish	0.821 ± 0.123	-8.370%
Danish	0.722 ± 0.207	-14.353%
Norwegian Bokmål	0.750 ± 0.196	-11.348%
Norwegian Nynorsk	0.708 ± 0.208	-14.596%
Faroese	0.653 ± 0.203	-10.055%
Icelandic	0.627 ± 0.238	-12.552%
Average	0.712 ± 0.208	-8.951%

From the table 5, we can see that when aligning between the m2m100 418M to opus-mt-NORTH_EU-NORTH_EU models, we lose on average about 8.951 % in cosine similarity when compared to aligning between opus-mt-NORTH_EU-NORTH_EU to opus-mt-SCANDINAVIA-SCANDINAVIA. There is also a bigger performance loss on the unknown languages (Danish, Norwegian Bokmål, Norwegian Nynorsk, Faroese,

Icelandic) compared to the known language (Swedish). We can also see a similar grouping of languages based on the cosine similarity scores as in the previous experiment.

There is a clear performance decrease when aligning embedding between models with different architecture compared to models with the same architecture. One potential reason for the performance decrease might be that because of the different architecture, the embeddings are more different. Another potential reason might be that the m2m100 418M model token embedding length is 1024, compared to the opus-mt-NORTH_EU-NORTH_EU model token embedding length of 512. This means the MLP regressor has 12.6 million trainable parameters instead of 8.3 million as in the previous experiment.

4.3 Translation

So far, we have just compared how similar the embeddings are, but in this section, we will use the transformed embeddings to generate translations and validate the translation quality. For evaluating the translation quality, we use the most popular implementation of the BLEU score, SacreBLEU [38]. For the text generation, we use our implementation of greedy generation described in Chapter 3.3. We used the OpenSubtitles Swedish - Danish corpus [33], and the FLORES-200 [35] dataset for measuring the translation quality.

We used the OPUS-MT [32] models opus-mt-NORTH_EU-NORTH_EU and opus-mt-SCANDINAVIA-SCANDINAVIA and the same MLP regressor as in the Chapter 4.2. We translated from Swedish to Danish to test embedding aligning on a known language for the regressor and from Danish to Swedish to test embedding aligning on an unknown language for the regressor. For each translation direction, we tested with sentences from the OpenSubtitles corpus [33] and the FLORES-200 [35] dataset. We used our generation implementation to translate using the opus-mt-NORTH_EU-NORTH_EU model, opus-mt-SCANDINAVIA-SCANDINAVIA model and our Embedding aligning model, which used the encoder from opus-mt-NORTH_EU-NORTH_EU model, decoder from opus-mt-SCANDINAVIA-SCANDINAVIA model and our MLP regressor.

From the Table 6, we can see that on every test set, the models opus-mt-NORTH_EU-NORTH_EU and opus-mt-SCANDINAVIA-SCANDINAVIA performed very similarly. The difference in BLEU score between these two models on all test sets was on average just 1.2 %. Using our embedding aligning model when translating on the language and

the dataset the regressor was trained, the BLEU score is 23.84, which is about 38.1 % lower than the other models. From the Table 7, we can see that the translations can have mistakes, but the end result is legible.

Table 6. Using greedy generation to translate from Swedish to Danish and vice versa on two different test sets. The Embedding aligning model uses opus-mt-NORTH_EU-NORTH_EU encoder to obtain the embeddings, then uses the MLP regressor with one hidden layer with 8192 neurons trained on OpenSubtitles Swedish dataset and uses the decoder of opus-mt-SCANDINAVIA-SCANDINAVIA.

Direction	**Test Set**	**Model**	**BLEU**
sv → da	OpenSubtitles n = 1012	opus-mt-NORTH_EU-NORTH_EU	38.22
		opus-mt-SCANDINAVIA-SCANDINAVIA	38.79
		Embedding aligning model	23.84
	FLORES-200 n = 1012	opus-mt-NORTH_EU-NORTH_EU	33.08
		opus-mt-SCANDINAVIA-SCANDINAVIA	33.29
		Embedding aligning model	7.00
da → sv	OpenSubtitles n = 1012	opus-mt-NORTH_EU-NORTH_EU	36.96
		opus-mt-SCANDINAVIA-SCANDINAVIA	36.91
		Embedding aligning model	10.85
	FLORES-200 n = 1012	opus-mt-NORTH_EU-NORTH_EU	32.00
		opus-mt-SCANDINAVIA-SCANDINAVIA	32.50
		Embedding aligning model	2.55

Table 7. Translating from Swedish to Danish with sentences from OpenSubtitles and FLORES-200 when using our embedding aligning model.

OpenSubtitles	Source:	Nej, jag bryr mig inte.
	Reference:	Det bryder jeg mig ikke om.
	Predicted:	Jeg er ligeglad.
	Source:	Vad hände med din gamla laptop?
	Reference:	Hvad er der med din gamle laptop?
	Predicted:	Hvad skete der med din gamle nar?
	Source:	Det är inte vår värld därute.
	Reference:	Det er ikke vores verden.
	Predicted:	Det er ikke vores verden.
FLORES-200	Source:	Det har gett oss tåget, bilen och många andra transportmedel.
	Reference:	Det har givet os toget, bilen og mange andre transportmidler.
	Predicted:	Det gav mig to, to, to, to, to, to, to, to, to, to, to, to, to, to, to, to, to, to, to - repeats till generation limit
	Source:	I början var kläder starkt influerade av den bysantinska kulturen i öst.
	Reference:	I starten var tøjet i høj grad inspireret af den byzantinske kultur i østen.
	Predicted:	begyndelsen var der en stor indflydelse på den bymæssige verden.
	Source:	Satelliten i rymden får signalen och reflekterar tillbaka den ned nästan omedelbart.
	Reference:	Satellitten i rummet modtager opkaldet og reflekterer det ned igen næsten med det samme.
	Predicted:	Sadel i rummet får vi fat i det og vender tilbage til det.

Translating from Swedish to Danish on the FLORES-200 [35] dataset with our embedding aligning model, the results are much worse than the other models. While

the opus-mt-NORTH_EU-NORTH_EU and opus-mt-SCANDINAVIA-SCANDINAVIA model achieved BLEU scores of 33.08 and 33.29 respectively, our embedding aligning model achieved a BLEU score of 7. Translation samples are shown in Table 7, from there we can see that when using the FLORES-200 dataset, there are much more mistakes, and in some cases, the decoder does not generate the EOS token, which leads to many repeating tokens being generated until the generation limit is reached.

When we translate from Danish to Swedish, then the source language is unknown for our regressor, so it makes sense that the BLEU scores are lower for our embedding aligning model in this translation direction. Compared to the other models, our model's performance is about 70.6 % lower on the OpenSubtitles corpus [33] and 92.1 % lower on the FLORES-200 [35] dataset.

From this experiment, we can see that when we transform embeddings on a language and type of text we use to train the regressor, we can perform a translation with these transformed embedding with some loss of quality. In cases the language is unknown to the regressor or the language domain is too different, the resulting translation quality is too poor to be useful. We have shown that translating with transformed embeddings is possible, but an MLP regressor might not be enough to achieve good translation quality.

5 Conclusion

In this thesis, we explored if it is possible to transform token embeddings from one machine translation model encoder so they would be similar to the embeddings of another machine translation model. For this, we developed a system to extract matching token embeddings from two translation models and a system to use these transformed embeddings with a decoder of another machine translation model.

We stated three main research questions in the introduction. Firstly, we wanted to find out the best regression method for transforming the token embeddings. We tested three different regression methods: linear regression, orthogonal regression and multi-layer perceptron. From our test, we concluded that the multi-layer perceptron achieved the best results, so we performed experiments to find out the optimal size and amount of hidden layers. We concluded that multi-layer perception, one hidden layer of 8192 neurons trained on 1.6 million tokens, achieved the best cosine similarity scores in our tests, so we used that regressor for transforming the embeddings.

Secondly, we wanted to see how well the aligning works on known languages and unknown languages and between models with different architecture. The aligning worked the best on language known to the regressor, for unknown languages, the performance was affected by how closely the languages were related to the language that was used for training the regressor. The languages that were more closely related to the regressor training language achieved better similarity scores. The aligning of embedding between models with different architectures resulted in a lower performance when compared to aligning between models with the same architecture.

Finally, we wanted to find out how much translation quality we lose with this mapping. When we translated on known language for the regressor and on the same dataset that was used to train the regressor, the translation BLEU score was 38.1 % lower than when using the original model. When we translated on unknown language or on different datasets, the BLEU score was much lower, 70 - 90 % decrease in BLEU score when compared to the original models.

In conclusion, we showed that embedding transforming with a simple regressor is possible, but the performance loss is quite large. To be practical, this approach needs to be improved, for example, use some more complicated neural network instead of the multi-layer perceptron with one hidden layer.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[2] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[3] Tom Kocmi, Rachel Bawden, Ondřej Bojar, Anton Dvorkovich, Christian Federmann, Mark Fishel, Thamme Gowda, Yvette Graham, Roman Grundkiewicz, Barry Haddow, et al. Findings of the 2022 conference on machine translation (wmt22). In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 1–45, 2022.

[4] Changtong Zan, Keqin Peng, Liang Ding, Baopu Qiu, Boan Liu, Shwai He, Qingyu Lu, Zheng Zhang, Chuang Liu, Weifeng Liu, et al. Vega-mt: The jd explore academy translation system for wmt22. *arXiv preprint arXiv:2209.09444*, 2022.

[5] Bing Han, Yangjian Wu, Gang Hu, and Qiulin Chen. Lan-bridge mt's participation in the wmt 2022 general translation shared task. In *Proceedings of the Seventh Conference on Machine Translation, Online. Association for Computational Linguistics*, 2022.

[6] Maali Tars, Taido Purason, and Andre Tättar. Teaching unseen low-resource languages to large translation models. In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 375–380, 2022.

[7] Makoto Morishita, Keito Kudo, Yui Oka, Katsuki Chousa, Shun Kiyono, Sho Takase, and Jun Suzuki. Nt5 at wmt 2022 general translation task. In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 318–325, 2022.

[8] Zhiwei He, Xing Wang, Zhaopeng Tu, Shuming Shi, and Rui Wang. Tencent ai lab-shanghai jiao tong university low-resource translation system for the wmt22 translation task. *arXiv preprint arXiv:2210.08742*, 2022.

[9] Artur Nowakowski, Gabriela Pałka, Kamil Guttmann, and Mikołaj Pokrywka. Adam mickiewicz university at wmt 2022: Ner-assisted and quality-aware neural machine translation. *arXiv preprint arXiv:2209.02962*, 2022.

[10] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[11] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International conference on machine learning*, pages 4055–4064. PMLR, 2018.

[12] Linhao Dong, Shuang Xu, and Bo Xu. Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5884–5888. IEEE, 2018.

[13] Jonathan J Webster and Chunyu Kit. Tokenization as the initial phase in nlp. In *COLING 1992 volume 4: The 14th international conference on computational linguistics*, 1992.

[14] Sabrina J Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y Lee, Benoît Sagot, et al. Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp. *arXiv preprint arXiv:2112.10508*, 2021.

[15] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[16] Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. *arXiv preprint arXiv:1804.10959*, 2018.

[17] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

[18] Guillaume Lample and Alexis Conneau. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*, 2019.

[19] Faisal Rahutomo, Teruaki Kitasuka, and Masayoshi Aritsugi. Semantic cosine similarity. In *The 7th international student conference on advanced science and technology ICAST*, volume 4, page 1, 2012.

[20] Baoli Li and Liping Han. Distance weighted cosine similarity measure for text classification. In *Intelligent Data Engineering and Automated Learning–IDEAL 2013: 14th International Conference, IDEAL 2013, Hefei, China, October 20-23, 2013. Proceedings 14*, pages 611–618. Springer, 2013.

[21] Anna Huang et al. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, volume 4, pages 9–56, 2008.

[22] Ritika Singh and Satwinder Singh. Text similarity measures in news articles by vector space model using nlp. *Journal of The Institution of Engineers (India): Series B*, 102:329–338, 2021.

[23] Liwei Wang, Yan Zhang, and Jufu Feng. On the euclidean distance of images. *IEEE transactions on pattern analysis and machine intelligence*, 27(8):1334–1339, 2005.

[24] Dongyang Wang, Junli Su, and Hongbin Yu. Feature extraction and analysis of natural language processing for deep learning english language. *IEEE Access*, 8:46335–46345, 2020.

[25] MD Malkauthekar. Analysis of euclidean distance and manhattan distance measure in face recognition. In *Third International Conference on Computational Intelligence and Information Technology (CIIT 2013)*, pages 503–507. IET, 2013.

[26] Deborah Coughlin. Correlating automated and human assessments of machine translation quality. In *Proceedings of Machine Translation Summit IX: Papers*, 2003.

[27] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[28] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[29] Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. On the cross-lingual transferability of monolingual representations. *arXiv preprint arXiv:1910.11856*, 2019.

[30] Chao Xing, Dong Wang, Chao Liu, and Yiye Lin. 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, 2015.

[31] Yun Chen, Victor OK Li, Kyunghyun Cho, and Samuel R Bowman. A stable and effective learning strategy for trainable greedy decoding. *arXiv preprint arXiv:1804.07915*, 2018.

[32] Jörg Tiedemann and Santhosh Thottingal. Opus-mt–building open translation services for the world. In *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation*. European Association for Machine Translation, 2020.

[33] Pierre Lison and Jörg Tiedemann. Opensubtitles2016: Extracting large parallel corpora from movie and tv subtitles. In *Proceedings of the 10th International Conference on Language Resources and Evaluation*, pages 923–929. European Language Resources Association, 2016.

[34] University of Tartu. Ut rocket, 2018.

[35] Marta R Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, et al. No language left behind: Scaling human-centered machine translation. *arXiv preprint arXiv:2207.04672*, 2022.

[36] Anders Holmberg and Christer Platzack. *The Role of Inflection in Scandinavian Syntax*. Oxford University Press, 1995.

[37] Angela Fan, Shruti Bhosale, Holger Schwenk, Zhiyi Ma, Ahmed El-Kishky, Siddharth Goyal, Mandeep Baines, Onur Celebi, Guillaume Wenzek, Vishrav Chaudhary, Naman Goyal, Tom Birch, Vitaliy Liptchinsky, Sergey Edunov, Edouard Grave, Michael Auli, and Armand Joulin. Beyond english-centric multilingual machine translation, 2020.

[38] Matt Post. A call for clarity in reporting bleu scores. *arXiv preprint arXiv:1804.08771*, 2018.

Appendix

I. Source Code Repository

Scripts developed for this thesis are openly accessible at the following code repository:
`https://github.com/hzuppur/Aligning-contextual-vector-spaces/`

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Hain Zuppur**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Aligning contextual vector spaces between independent neural translation systems,

   supervised by Mark Fišel.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Hain Zuppur
09/05/2023