Machine Translation for Different Language Pairs

Jordan Lau & Chelsea Brown

300240600 & 300290019

Department of Linguistics, University of Ottawa

LIN 2330: Computation in Linguistics

Professor Kevin McMullin

April 17, 2025

Introduction

Machine learning is all the rage right now, and one of the most studied tasks is automatic translation. Of course, computers treat language very differently than humans do, but studying how they break down language translation into a modular process is a fascinating topic in computational linguistics.

It seems intuitive that "similar" languages might be easier to translate between each other. But what defines languages as similar or different? Vocabulary? Word order? Genetic origin? Writing system?

In this experiment, we observe how artificial neural networks — the most popular machine learning algorithm to date — translate textual sequences from English to languages with varying degrees of similarity. We evaluate patterns in the mistakes they make, and the difference in performance between language pairs. Our goals are to:

- Gain insight on the nature of translation as a task, for humans or computers.
- Learn what similarities and differences in languages impact translation quality between them.
- Gain hands-on experience with machine learning, which adds a layer of practicality to build on our theoretical knowledge.

Our python code is released at https://github.com/jordanxlau/comparison_of_language_pairs.

Experimental Procedure

1. Choosing the language pairs

One paper (Georgi et al., 2010) uses typological features from the WALS database to compute language similarity. They determine how much various languages have in common with English by creating a list of features for each language, and computing the cosine distance between those lists. They suggest that languages with similar features (for instance, English and Finnish) are closer than those with genetic relationships (such as English and Persian, two Indo-European languages).

Inspired by this analysis, we choose to compare translations to Finnish and Persian (also known as Farsi). We also add Dutch, as it is widely known to be extremely similar to English. The languages are:

- English to Dutch, a minimally different language pair.
- English to Finnish, a moderately different language pair.
- English to Persian, a very different language pair.

	English	Dutch	Finnish	Persian
Order of Subject, Object and Verb	SVO	None Dominant	SVO	SOV
Order of Adposition	Prepositions	Prepositions	Postpositions	Prepositions

Position of Tense-Aspect Affixes	Suffixes	Suffixes	Suffixes	Both
Number of Cases	2	0	>10	2
Writing System	Alphabetical	-	Alphabetical	Consonantal

Table 1. Comparison of our language pairs using selected WALS (Dryer & Haspelmath, 2013) features.

2. Choosing a Dataset

To train our neural network, we use an open-source dataset called OPUS-100 (Helsinki-NLP, n.d.). The dataset has up to 1 million pairs of sentences and their respective translations for many different language pairs. It was curated by automatically processing other available corpuses that had been translated by humans, including movie subtitles and The Bible (Zhang et al., 2020). We have discovered that some sentence pairs in this dataset are not perfectly accurate: the supposed translation does not match the reference sentence. This is likely due to flaws in the algorithm they used to gather the data, and will certainly have affected our experiments.

In `data.py`, the dataset is loaded, then split randomly into separate *train* (`train`), *validation* (`val`) and *test* (`test`) subsets. It is important to test on a separate test set in order to verify that the model is generalizable to new phrases, and is not simply memorizing translations it has seen before.

```
# Downloading the data
text =
pd.read_parquet("hf://datasets/Helsinki-NLP/opus-100/"+lang1+"-"+lang2+"/train-00
000-of-00001.parquet")
```

Snippet 1. Downloading the dataset.

```
random.shuffle(text_pairs)
num_val_samples = int(0.15 * len(text_pairs))
num_train_samples = len(text_pairs) - 2 * num_val_samples
train_pairs = text_pairs[:num_train_samples]
val_pairs = text_pairs[num_train_samples : num_train_samples + num_val_samples]
test_pairs = text_pairs[num_train_samples + num_val_samples :]
```

Snippet 2. Splitting the data into training, validation and test subsets.

Both the input and output sentences in the dataset are *vectorized* (*tokenized*) into a list of tokens. Each token is similar to a morpheme; one represents a single unit of meaning. This allows the model to understand how the parts of a sentence work together, and results in better generalization to sentences it has never seen translated before.

```
# Formatting Datasets
def format_dataset(l1, l2):
    l1 = lang1_vectorization(l1)
    l2 = lang2_vectorization(l2)
    return (
        {
            "encoder_inputs": l1,
            "decoder_inputs": l2[:, :-1],
        },
        l2[:, 1:],
    )
```

Snippet 3. Tokenization of the datasets, before model training.

3. Configuring the model

The model architecture is defined in `training.py` as a Sequence-to-Sequence Transformer, based on a template from Keras (Chollet, 2021). The model definition can be seen in Snippet 4 below. Transformers are a very complex, sophisticated type of neural network commonly used in translation tasks. Fully explaining a transformer is completely beyond the scope of this project. One part of the model worth noting is the `PositionalEmbedding` layer; here, tokens are *embedded* as extremely long vectors (lists of numbers). The position of the word in the sentence, as well as its contextual semantic meaning, are supposedly encapsulated in this vector.

```
encoder_inputs = keras.Input(shape=(None,), dtype="int64", name="encoder_inputs")
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(encoder_inputs)
encoder_outputs = TransformerEncoder(embed_dim, latent_dim, num_heads)(x)
encoder = keras.Model(encoder_inputs, encoder_outputs)

decoder_inputs = keras.Input(shape=(None,), dtype="int64", name="decoder_inputs")
encoded_seq_inputs = keras.Input(shape=(None, embed_dim),
name="decoder_state_inputs")
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(decoder_inputs)
x = TransformerDecoder(embed_dim, latent_dim, num_heads)([x, encoder_outputs])
x = layers.Dropout(0.5)(x)
decoder_outputs = layers.Dense(vocab_size, activation="softmax")(x)
decoder = keras.Model([decoder_inputs, encoded_seq_inputs], decoder_outputs)

transformer = keras.Model(
    {"encoder_inputs": encoder_inputs, "decoder_inputs": decoder_inputs},
```

```
    decoder_outputs,
    name="transformer",
)
```

Snippet 4. Definition of the sequence to sequence transformer architecture.

When we ask the model to translate an English sentence, (Snippet 5, in `data.py`) we will provide it with that sequence (`input_sentence`). We must also start it off with the first token(s) of the output sequence, in the target language (`decoder_inputs`). We can simply set that first token to `[start]`, indicating that the output sentence has just begun. This is because a translated sentence in the requested language is outputted one token at a time, left-to-right[1]. The model's learned parameters will tell it what the most probable next word is based not only on the English sentence, but also on the translated tokens it has already added to the output sentence. At some point, the model will add the token `[end]`, indicating to itself to stop adding words to the output sequence.

```
tokenized_input_sentence = lang1_vectorization([input_sentence])
decoded_sentence = "[start]"
for i in range(max_decoded_sentence_length):
    tokenized_target_sentence = lang2_vectorization([decoded_sentence])[:, :-1]
    predictions = transformer(
        {
            "encoder_inputs": tokenized_input_sentence,
            "decoder_inputs": tokenized_target_sentence,
        }
    )
```

Snippet 5. Decoding sentences.

4. Training the models

Neural networks are considered a form of machine learning because each layer of the model has several million parameters (weights) that need to be learned. Each of our models has a total of 19,960,216 parameters that will help it decide on the most likely translations. No wonder the process is so long!

During the training phase (executed by `transformer.fit()` in `training.py`), the model learns by seeing the example pairs from `train_ds` and `val_ds` many times, and adjusting its parameters according to how accurate its output translation was. The key is that this learning is *supervised*, meaning the model knows the correct answer/correct translation for all of the training data, and can adjust itself accordingly.

```
# Training our model
transformer.fit(train_ds, epochs=epochs, validation_data=val_ds)
```

Snippet 6. Training the `transformer` neural network model.

[1]At least, this is the case for languages like English and Dutch that read left-to-right.

The variable `epoch` is set to 10, meaning the model will see all of the training data ten times. We also chose to show the model 350,000 training examples. These are set earlier, in `data.py`. The higher, the better — up to a certain point of course — but training is very time and resource intensive.

5. Decoding and testing the models

Running `decode.py` tests the model on a separate reserved test set of 75,000 pairs of sequences (input, reference) from the dataset. This script also prints out each example. To calculate the quality of each test sequence, we use the BLEU (BiLingual Evaluation Understudy) metric. We define three sets:

1. `test_inputs`, containing the input sentences, in English. For example, "Oh, my God!"
2. `translated_references`, containing the corresponding reference translations given in the dataset. For example, "O, mijn god!"
3. `predictions`, containing the translations outputted by our model. For example, "mijn god"

```
test_inputs = [pair[0] for pair in test_pairs]
test_references = [pair[1] for pair in test_pairs]
print(len(test_inputs))

for i in range(len(test_inputs)):
    input_sentence = test_inputs[i]
    reference_sentence = test_references[i]
    translated_sentence = decode_sequence(input_sentence)
    predictions.append(translated_sentence.split())  # Tokenize prediction
    ground_truths.append([reference_sentence.split()])  # Tokenized reference
```

Snippet 7. Decoding the models.

```
# Compute BLEU score
bleu_score = bleu_metric._calculate_bleu_score(ground_truths, predictions)
```

Snippet 8. Calculation of BLEU score across all reference sequences and our model's prediction.

For each row of the test data, the prediction is calculated by the model, and passed to the `calculate_bleu_score()` function along with the set of references. The actual translations are compared against the references by checking the proportion of matching n-grams. The average of these results is used to score the effectiveness of the translation model as a whole.

The BLEU metric aims to evaluate the alignment of a translated sequence with an accepted human translation. The metric is computed by calculating the proportion of n-grams in the output that appear in the reference translation (Chiusano, 2022). This proportion is calculated for unigrams, bigrams, trigrams, and 4-grams in the output, and then the geometric average is multiplied by a brevity penalty (used to penalize outputs that are too short compared to the expected reference—the greater the difference in length, the greater the penalty will be). Usually,

4-grams are the longest substrings used for comparison, as the greater the n-grams, the less likely it is to find overlap, and the proportion of matching n-grams will be significantly lower. This avoids unnecessarily penalizing the translation further. The final BLEU score will be a value between 0 and 1, where 1 is a perfect match.

For example, the BLEU score for the Dutch translation in Figure 1 would be ~0.229. Assume the input sentence “Where did you say Dr. Choi was again?” has the given Reference and Candidate translation[2].

Figure 1. Example BLEU scoring.

This is a medium-low score because they have 5 unigrams, 2 bigrams and no trigrams or 4-grams in common. In ideal datasets, each input sentence is accompanied by multiple valid reference translations, as the same idea *can* be communicated in different terms. Having multiple references to compare to increases the chance of overlapping n-grams. Thus, BLEU scores tend to improve the more robust a dataset is.

Results

Our final models were trained with 10 epochs and 350,000 training examples. We display the accuracy throughout training, as well as the computed BLEU score, which was calculated over a holdout set of 75,000 examples, that the models had not seen during training.

	Dutch (NL)	Finnish (FI)	Persian (FA)
Training Accuracy	~0.20	~0.14	~0.14
BLEU Score on Test Set	0.0442	0.0128	0.0328

Table 2. Results for final models.

The Dutch model is notably more accurate than the Finnish or Persian. This is in alignment with our hypothesis that sequences in similar languages will be better suited to translate from one to the other. However, interestingly, Finnish performed worse than Persian, contrary to our hypothesis. None of the models are strong enough to compete with the best commercially available translation software.

[2] Adjusted slightly for clarity of the example.

Below, in Tables 3 to 5, we display examples of our model's predictions on various types of sentences, for each of the three languages.

	Input Sequence	**Reference Translation + Gloss**	**Predicted Translation + Gloss**
NL	Where did you say Dr. Choi was again?	Waar zei u dat Dr Choi ook alweer was? where say.PST.2SG you.FEM that Dr Choi again was?	waar zei je dr [UNK][3] was where say.PST.2SG you.INF Dr [UNK] was?
FI	Where did you say you got this?	Mistä sait tämän? from.where get.PST.2SG this?	Mihin to.where
FA	Why can't they return to the jungle?	چرا برنمی‌گردند به جنگل‌های اطراف؟ why NEG-IPFV-return-3PL to jungles-of surroundings	چرا [UNK] به جنگل [UNK] why [UNK] to jungle [UNK]

Table 3. Wh-phrases.

	Input Sequence	**Reference Translation + Gloss**	**Predicted Translation + Gloss**
NL	Turn it off, Gabe.	Zet het uit, Gabe. set/put it out/off Gabe	leg het af [UNK] lay/put it off/down [UNK]
FI	Get down now!	Tule alas! come.IMP.2SG down	Alas down
FA	Go home, Addy.	برو خونه ادي go.IMP.2SG home Addy	برو خونه go.IMP.2SG home

Table 4. Imperative Clauses.

Declarative Clauses

[3] [UNK] represents a token the model does now know.

	Input Sequence	Reference Translation	Predicted Translation
NL	I've never seen her before.	Ik had haar nog nooit gezien. I had her yet/still never seen	ik heb haar nog nooit meer gezien I have her yet/still never more seen
FI	Her name is Mandy.	Hän on Mandy. he/she is Mandy	hänen nimensä on her.GEN name.POSS.3SG is
FA	My sister will decide what he looks like.	خواهر من تصمیم میگیره اون چیه Sister my decision take.3SG that/he what-is	[UNK] من [UNK] [UNK] [UNK] my [UNK] [UNK]

Table 5. Declarative Clauses.

	Input Sequence	Reference Translation	Predicted Translation
NL	Alright.	Waarom ook niet. Why also not	oké Okay
FI	No. No.	Ei! No	ei ei No no
FA	Oh, hey.	اوه هي ما بوديم.. Oh hey we were	اوه ، هي Oh hey

Table 6. Exclamations.

In all language pairs, proper nouns, most frequently names, come up as UNK or are simply dropped, likely because they are not repeated enough in the training to be learned by the model in any meaningful way. However certain trends are more common depending on the language.

In the Dutch results, grammatical information is dropped or altered in the predicted translation. (the feminine marker in Table 3 becomes infinitive). The original English phrase does not explicitly show that the subject is feminine, so this conclusion is not surprising. Synonyms are frequently used, sometimes at the loss of the original meaning. The example in Table 4 should translate to “Turn it off, Gabe,” however the predicted translation produces “Put it down, Gabe.” These are small changes in vocabulary that, on a word-by-word basis, don't seem to make a difference, but actually impact the meaning of the sentence significantly.

In Finnish, the endings of sentences are omitted completely, not simply marked as an unknown word. In Table 3's example, only the wh- word, “where” remains, but is still the incorrect form (communicating “where is the destination” rather than “where is the source”). In the declarative clause (Table 4), the predicted translation would be a better literal translation than the reference, if the name “Mandy” was present (“her name is [Mandy]” vs. “She is Mandy”). However, there are likely contextual reasons why the latter is more preferred in everyday speech.

Finally, it appears that the Persian model had more difficulty in learning words and morphemes, which explains the abundance of unknown tokens. This may be due to the structure of Persian orthography and variance in the way the same morpheme can be symbolized depending on its adjacent morphemes (Sagot, 2012). Characters can either be combined with, next to, or separated (by whitespace) from following or preceding characters, and this affects the way words appear in text. The meaning of certain words or morphemes may be lost during tokenization, as a result.

Discussion

Interestingly, the Finnish model did not have an advantage over the Persian one in terms of its overall accuracy numbers. Although this goes against our initial hypothesis, the small sample size (just three language pairs) must be kept in mind. This caused us to rethink our method of calculating language similarity. Perhaps similarity of vocabulary is a major factor, which typological features do not account for. Bella et al. (2021) and others have researched lexical similarity. They find, understandably, that Dutch and English have very close vocabularies. Unfortunately, they do not show Finnish and Persian in their paper. The availability of data for these languages is a major factor affecting our research, as is the quality of the machine learning models.

Lexical similarity, and other metrics of language similarity are excellent areas for future research. We would like to eventually learn which metrics correlate most closely with the accuracy of neural translation models for those languages. To this end, it would also be hugely beneficial to compare more languages, to gather more data.

From a practical standpoint, we have learned that machine translation is very complex algorithmically, taking lots of time, processing power and storage. The amount of data necessary to perform translation, as a task, is significantly greater than that for a classification task like the MNIST dataset (LeCun, Y., & Cortes, n.d.). That dataset has just 60,000 data points, yet many models implemented with basic *convolutional* neural networks have great results, perhaps because these models have a fixed number of inputs and outputs. Our translation models were trained on datasets of hundreds of thousands of examples, and have much poorer results.

The saved model files themselves are over 100 MB, so even sharing these is an obstacle. Training on a computer with 32GB of RAM and an 8 core processor with 350 thousand training examples and 10 epochs took 10 hours in total.

Input Sequence	**Predicted Translation**
Selects the colour of the non-metal elements	die die die die die die die die die

Table 7. Translation from an early model of English to Afrikaans with 1 epoch and 100 training examples.

An interesting observation is that only at 6 to 10 epochs do the model predictions become coherent at all. An early model we had trained on 100 examples[4] with just 1 epoch made extremely poor predictions. The prediction in Table 7 above is likely because *die*, the Afrikaans definite article, is a common word. So, the model produces the most accurate results by repeating it. In the future, we would like to train more models with more epochs, to achieve more accurate translations overall.

Another useful progression of our experiment would be to evaluate the translation based on metrics other than BLEU. BLEU is widely used because of its simplicity and low cost to implement, but it doesn't take into account semantic meaning, whether through synonyms, alternative spellings, formality of speech, etc. Other metrics exist that follow a more nuanced approach. One such metric, METEOR (Metric for Evaluation of Translation with Explicit ORdering) considers synonymy and stemming in the calculation, which allows the metric to assign high scores to outputs that may have low precision in terms of n-grams, but that preserve semantic meaning. Implementing the METEOR metric would be a beneficial next step in our experiment, as our results would match human references more accurately.

We believe this work is important in the study of language typology and in machine translation of under-resourced languages.

[4] Yes, 100 — not 100 thousand.

References

Bella, G., Batsuren, K., & Giunchiglia, F. (2021). A database and visualization of the similarity of contemporary lexicons. *Lecture Notes in Computer Science*, 95–104. https://doi.org/10.1007/978-3-030-83527-9_8.

Chiusano, F. (2022, January 11). *Two minutes NLP — Learn the BLEU metric by examples*. Medium. Retrieved April 17, 2025, from https://medium.com/nlplanet/two-minutes-nlp-learn-the-bleu-metric-by-examples-df015ca73a86

Chollet, F. (2021, May). *English-to-Spanish translation with a sequence-to-sequence transformer*. Keras Documentation. Retrieved from https://keras.io/examples/nlp/neural_machine_translation_with_transformer/.

Dryer, M, & Haspelmath, Martin. (2013). World Atlas of Language Structures. https://wals.info

Georgi, R., Xia, F., & Lewis, W. (2010). Comparing Language Similarity across Genetic and Typologically-Based Groupings. *Proceedings of the 23rd International Conference on Computational Linguistics*. Retrieved from https://aclanthology.org/C10-1044.pdf.

Helsinki-NLP. (n.d.). *Opus-100 Dataset*. Hugging Face. Retrieved from https://huggingface.co/datasets/Helsinki-NLP/opus-100/viewer/am-en?views%5B%5D=am_en_train.

LeCun, Y., & Cortes, C. (n.d.). *MNIST Digits Classification Dataset*. Keras Documentation. Retrieved from https://keras.io/api/datasets/mnist/.

Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2001). BLEU: a method for automatic evaluation of machine translation. *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*.

Sagot, B., & Walther, G. (2012, November 14). A morphological lexicon for the Persian language. *HAL Open Science*. https://shs.hal.science/halshs-00751629v1.

Zhang, B., Williams, P., Titov, I., & Sennrich, R. (2020). Improving massively multilingual neural machine translation and zero-shot translation. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. https://doi.org/10.18653/v1/2020.acl-main.148.