Language Translation using Transformers

Abstract

Language translation is one of the fundamental applications of NLP systems. Providing accurate translations of handwritten languages while maintaining the linguistic and cultural properties of the languages under consideration is a challenging task. Translation involves inadvertent risks in terms of paying attention to the grammar and syntax of the source and target language respectively. In this work I have tried to make efficient use of transformers to effectively translate languages by comparing performances across mainly two kinds of transformers; T5 (Text-to-Text Transfer Transformer) and marianMT (marian Machine Translation). During evaluation, BLEU and Meteor score improvements were found in languages that have better resources .

1 Introduction

The fundamental part of being human is the ability to communicate. Presently, there are around 7,100 languages spoken worldwide. In order to maintain economic and cultural connections between people belonging to different cities and countries, language translation is a crucial tool. More common use cases of language translation are: government, education, media, commerce, business. With the advent of technology, the previously observed disadvantages of machine translation like grammatically incorrect sentences and errors in syntax are declining. Nevertheless, the fact that machine translation is faster and cheaper cannot be ignored, in fact the very same advantage is the driving force in developing more user friendly and quality machine translation systems.

Machine translation mainly involves processing entire sentences of a particular language into the intended target language. Transformer is a novel architecture that processes such sentences in a sequence-to-sequence format rather than processing word to word translation of languages. Transformer as a NLP model was first introduced in Vaswani et al. (2017), according to which, The Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution. The attention layer provides a way of modelling dependencies of sequences without having to worry about their length in both the input and output sequences. This has laid the foundation for using attention layer as an integral part of modelling sequences and transduction (input-output sequence conversion).

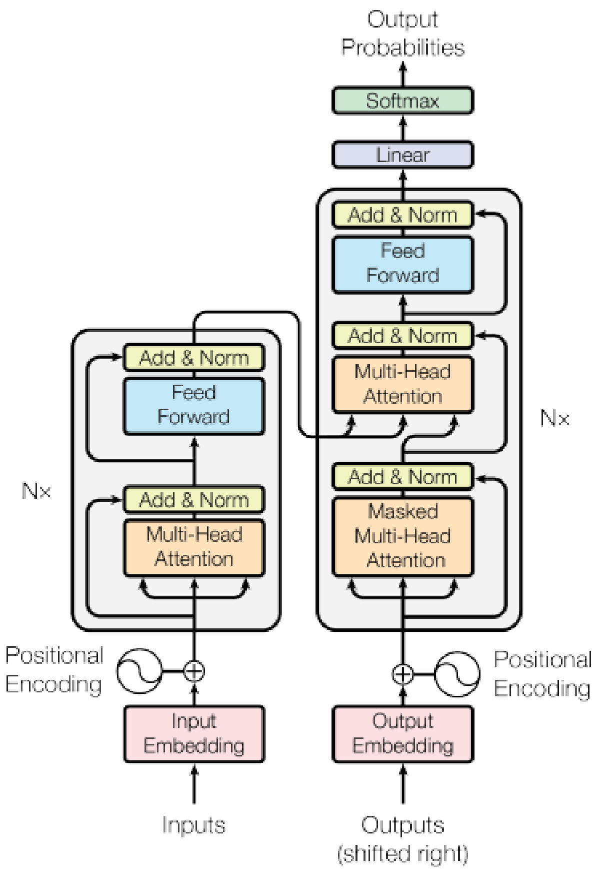


Figure 1: Transformer Architecture adapted from Vaswani et al. (2017)

Figure 1 shows the architecture of the transformer model. Through the figure it can be observed that attention layer is one of the crucial sections of the architecture. The attention layer acts as a medium in guiding the model into selecting and processing relevant parts of an input sequence. Representations of sequence is generated by using relative analysis of varied positions of the sequence. The layer is labelled as Multi-head attention because of the multiple iterations used to compute self-attention of a sequence. Multiple encoder and decoder layers are stacked over one another to form encoder-decoder blocks. The number of layers to be considered in the mechanism of the transformer is a hyper parameter and can be specified based on the requirements of the application and performance statistics. The input to the encoder stack is the word embeddings of an input sequence, which is further passed on to the decoder layers stack.

Inspired by the work of Raffel et al. (2019) and Junczys-Dowmunt et al. (2018), I propose to use T5 and marianMT variants of transformers to obtain better machine translation results in NLP systems.
T5: Other applications of transformers (T5 in particular) includes document summarization, sentiment classification and question answering to name a few. The documentation of the transformers on the 'Hugging Face' website[1] can be used to learn more about applications of various versions of transformers like T5 as they are available in different sizes based on need of application. Presently five sizes of the T5 transformer are available – t5-small, t5-base, t5-large, t5-3b, t5-11b. For the purpose of this project t5-small is being used.
marianMT: On the other hand, there are around 1000 models of the marianMT transformer available for use based on source and target language. The models were originally trained using Marian C++ library, well known for fast training and translation properties. The smaller attributes of the marianMT transformer model when compared to other translation models available, accounts for its usage in experiments involving fine-tuning and integration test. This work mainly focuses on using different widely available translation datasets of languages to build a translation pipeline using T5 and marianMT transformers. I have empirically tried to evaluate the performance of the machine translation system when using both kinds of transformer models. A comparison of evaluation has also been shown for future scope of the implementation. I have briefly discussed the features and downsides of various existing architectures applicable to machine translation in specific.

The paper is structured as follows: In Section 2, I present a brief of previous related work in machine translation, Section 3 provides the resources for the datasets used. Section 4 explains the proposed method along with instructions on importing the dataset into the project. Section 5 displays the comparative evaluation performed on the proposed models. Section 6 can be viewed as a guide for future research directions and lastly, I conclude with Section 7.

2 Related Work

The need for machine translation is colossal in the present day scenario and this importance is clearly depicted in Saini and Sahula (2015). A type of machine translation known as the Statistical Machine Translation uses the monolingual and bilingual corpora to define and train models to perform better on tasks like word alignment and language modelling, an overview of which is mentioned in Babhulgaonkar and Bharad (2017). Over time a huge number of technologies have been developed to improve the efficiency and accuracy of machine translation and one among them is the neural machine translation model, which uses neural network nodes for language translation. He et al. (2016) shows a comparison of performance between the Statistical Machine Translation and Neural Machine Translation. As an improvisation to the SMT, Recurrent Neural Networks were used for machine translation and Premjith et al. (2019) provides the evaluation metric improvements for the same. RNN models use the property of sequence to sequence learning and a network of encoders and decoders. Rishita et al. (2019) provides comparison of machine translation performance when different versions of the RNN models are used. Further upgrade in accuracy were achieved in using RNN with LSTM(Long short-term memory). LSTM helps in avoiding the long term dependencies that occur in RNN due to the presence of a recurrence, where the hidden state of previous word is required before encoding present word in a sentence. The recurrence is what contributes to the problem of long term dependencies since before getting to the present cell long

[1] https://huggingface.co/

term information in sentences has to sequentially pass through all cells of the network, hence risking loss of information in the process. Attention mechanism mainly focuses on raw data to increase the accuracy of machine learning models (Tang et al., 2016). Present day, RNN models also use attention mechanism (Luong et al., 2015) but this cannot compensate for the long term dependencies caused by the recurrence.
Major research works in the field of machine translation mostly include datasets that include English – French translation with English mostly being the source language and French as the target language. This calls for better translation models trained using datasets of low resource languages (ex: English-Dutch translation).
In this work, I have provided a comparison of performances when different datasets of English-French, English-German and English-Dutch are used to train machine translation pipelines using both T5 and marianMT transformer types. Though initially the analysis was performed by reporting the BLEU score of the trained model (Papineni et al., 2002). Callison-Burch et al. (2006) depicts the need for using multiple evaluation metrics in deciding the performance accuracy of machine translation models. As a result, meteor scores (Denkowski and Lavie, 2014) have also been used as an evaluation metric in deciding the performance of the individual models.

3 Datasets

The following datasets have been used for the purpose of this work:

KDE4 (Tiedemann, 2012) : Kernel Density Estimation dataset with 92 languages.[2]

WMT16 (Bojar et al., 2016) : Introduced in the Findings of the 2016 Conference on Machine Translation. [3]

Opus Books : The open parallel corpus. The opus book dataset is a constantly growing [4] dataset of curated language translation corpus.[5]

Source Language : English

Target Languages : French, German, Dutch

4 Methodology

Before discussing more about the steps involved in implementing the proposed work let us look at some of the key points of the T5 and marianMT transformers. Note that both kinds of transformer models use the generic Encoder-Decoder Transformer architecture previously explained in Introduction.

T5:

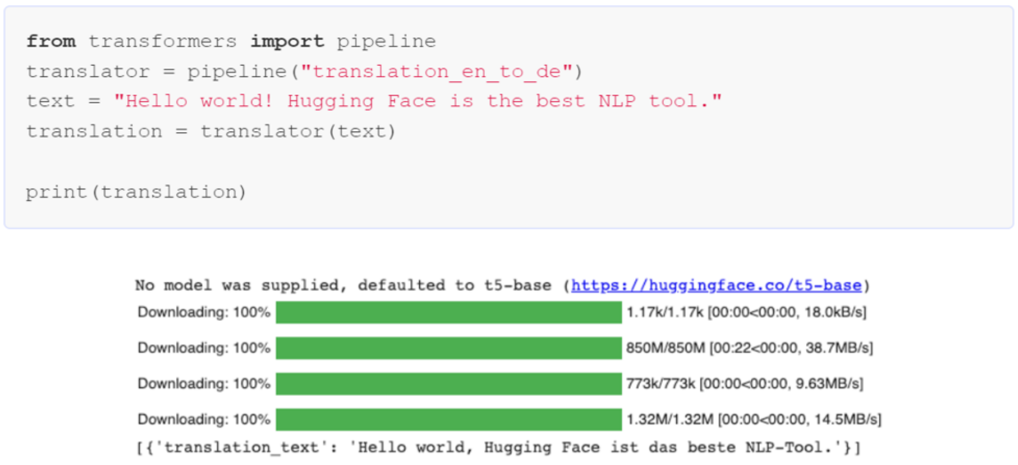
- Unified framework to convert all text language problems into text-to-text format.
- Trained on Colossal Clean Crawled Corpus(C4).
- Requires prepending of prefix, defining language task to be performed on the data. Example: for translation – 'translate English to German'; for summarization – 'summarize'.

marianMT:

- Built on 'Marian' – A Neural Machine Translation framework with an integrated automatic differentiation engine [based on dynamic computation graphs].
- Faster training and translation.
- Automatically edits machine translated output.
- Trained on Open Parallel Corpus (OPUS).

The first step in building the machine translation model is the collection of data. The Hugging face platform provides options for filtering through datasets to choose the right kind of data set required by the user based on the application. The platform also provides for using pre trained models in building our own machine translation pipeline. Presently, translation pipelines are available under the Transformers library in which case the input text is provided as an object to the pipeline variable and the resulting output is the translation of the source language. Figure 2 shows the implementation of the same. One major drawback of using these kinds of pipelines is that the default

[2]https://huggingface.co/datasets/kde4
[3]https://huggingface.co/datasets/wmt16
[4]https://opus.nlpl.eu
[5]https://huggingface.co/datasets/opus_books



```
from transformers import pipeline
translator = pipeline("translation_en_to_de")
text = "Hello world! Hugging Face is the best NLP tool."
translation = translator(text)

print(translation)
```

```
No model was supplied, defaulted to t5-base (https://huggingface.co/t5-base)
Downloading: 100%  1.17k/1.17k [00:00<00:00, 18.0kB/s]
Downloading: 100%  850M/850M [00:22<00:00, 38.7MB/s]
Downloading: 100%  773k/773k [00:00<00:00, 9.63MB/s]
Downloading: 100%  1.32M/1.32M [00:00<00:00, 14.5MB/s]
[{'translation_text': 'Hello world, Hugging Face ist das beste NLP-Tool.'}]
```

Figure 2: In-Built transformer pipeline implementation

transformer used is T5- base and the translation is restricted to English being the source language and French, German, Romanian being the target languages. In order to extend the machine translation to using Transformers of choice and also in translation of other available languages, it is necessary to build our own custom translation pipeline. Figure 3 shows the steps involved in building our own Machine translation transformer model. Also shown are the brief descriptions of individual steps involved in each stage of the implementation. A large number of pre-trained models are available for use in training our custom machine translation transformer model.
Model name syntax for T5 transformer –> “t5-small”; marianMT –> “Helsinki-NLP/opus-mt-src-target”

Loading Dataset:
The dataset can be loaded from the Hugging Face interface using the load_dataset of the datasets library. The input object to the load_dataset instance will be the name of the dataset being downloaded and the source(src)-target language code. Language codes are also available under the datasets filter provided by the interface. In this case, en-fr(English-French) ; en-de(English-German); en-nl(English-Dutch) language codes are being used. The downloaded data will have to undergo some further preprocessing steps to make it suitable for being passed as an input to the pre-trained transformer model.

Tokenization:
Tokenization is the process of converting raw data to numbers to enable the model in making sense of the data. First the text is split into words and sub-words and later each token is mapped to an integer. Output of the tokenizer contains a dictionary with two keys- attention mask and input ids. Attention mask performs the task of batching the input sequence together and input ids act as unique identifiers for tokens in a sentence. Tokens that have an attention mask of 0 will be ignored and the ones with 1 are considered important and are further processed. Each pre trained model is provided with its own tokenizer which is specific to the transformer model parameters. It is imperative to use the respective tokenizers while training the transformer model. ‘AutoTokenizer.from_pretrained(model_name)’ is the syntax used for initializing the tokenizer object. The tokenized inputs are passed to the model and the model generates an output of vectors.

Pre processing:
A closer look at the different datasets will show that the train test split for each of them is different. For this reason, a preprocessing function is defined to perform operations like splitting the data, creating subsets, and applying the tokenizer object on the dataset. The preprocessing function receives all instances of the data as input and divides them into input and target sequences. The tokenizer object is applied to both the input and target sequences and the maximum length of the sequences is set to 128 in our case. The preprocessing function is applied to the entire data set and the instances are mapped accordingly. A train-test split of 90-10 is usually preferred. Subsets of the train and test dataset splits are made for efficiency and faster training of the machine translation model.

Transformer model:

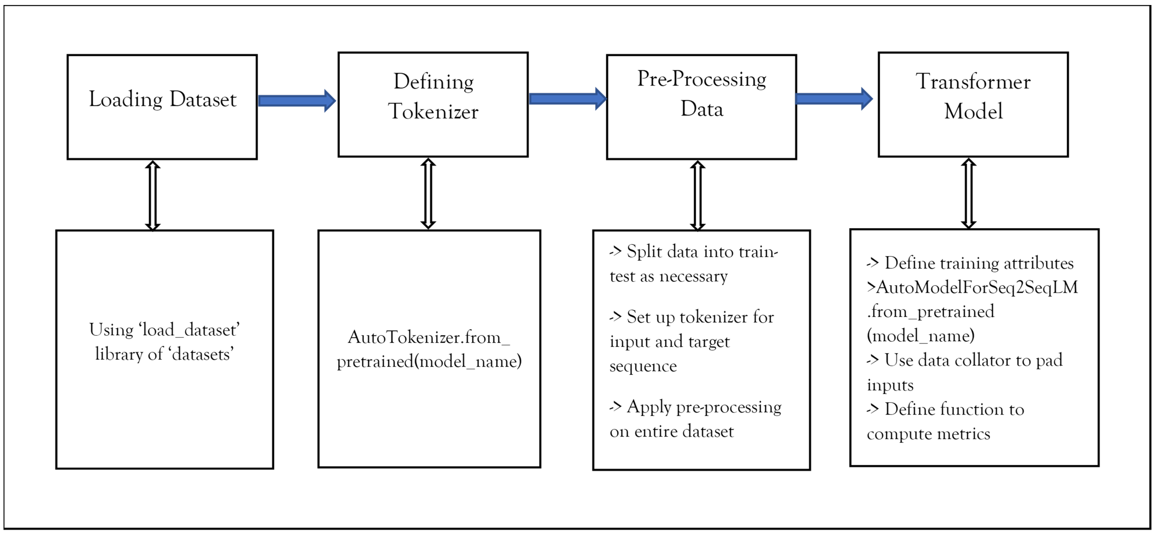

Loading Dataset
Defining Tokenizer
Pre-Processing Data
Transformer Model
Using 'load_dataset' library of 'datasets'
AutoTokenizer.from_pretrained(model_name)
-> Split data into train-test as necessary
-> Set up tokenizer for input and target sequence
-> Apply pre-processing on entire dataset
-> Define training attributes
>AutoModelForSeq2SeqLM.from_pretrained (model_name)
-> Use data collator to pad inputs
-> Define function to compute metrics

Figure 3: Methodology

The trainer API instance of the model requires parameters like the model name, training arguments passed, the training data set, the evaluation data set, data collator, tokenizer and the metrics computation function. The first step is to initialize the model object. 'AutoModelForSeq2SeqLM.from_pretrained(model_name) is the syntax used for the same. The next step is to define the training attribute, the batch size chosen for this purpose is 16. After the training attributes have been defined it is necessary to define a data collator to pad the inputs and label them. A custom metrics computing function has also been defined to generate the BLEU and meteor scores and display them. Before computing the metrics, it is necessary to decode both the prediction sequences and the label sequences. It is also required to check the format of the labels as there can be garbage values present in it. In cases where the metrics computing function is not defined, the training loss and the validation loss of training the machine translation model are reported by default. Lastly, the train function is used to initialize the trainer object and train the model. The model fine tuned in this way can be saved in the local directory and will now be considered as a pre-trained model. Hugging face also provides for an option to push the model and share it in the hub.

5 Evaluation Results

BLEU score is a measure of correspondence between a human's translation of an input sequence and the machine's output translation of the input sequence. On the other hand, the meteor score is based on the concept of unigram matching between the reference translation of the human produced translation and the machine produced translation. Unigram matching means that one word sequences of both the label sequence and the predicted sequence are matched for similarities. Greater the BLEU and meteor score, better is the performance of the machine translation model. BLEU score has a range of 0-100 and Meteor score has a range of 0-1. Both the metrics can be computed using the 'compute' feature available in the 'load_metric' property of the 'datasets' library. Usage of all the libraries mentioned in this work will require installation of the same in the Jupyter notebook environment or any other run environment being used.

English-Dutch(opus$_{b}ooks$)		
	BLEU	Meteor
T5	0.36	0.06
marianMT	14.72	0.325

Table 1: Evaluation metrics of English-Dutch Translation

English-French(kde4)		
	BLEU	Meteor
T5	10.71	0.15
marianMT	42.72	0.29

Table 2: Evaluation metrics of English-French Translation

Tables 1 , 2 and 3 show the evaluation metrics scores for English-Dutch, English-French and

English-German (opus$_books$)			English-German (wmt16)		
	BLEU	Meteor		BLEU	Meteor
T5	3.42	0.16	T5	13.01	0.28
marianMT	15.77	0.33	marianMT	36.38	0.50

Table 3: Evaluation metrics of English-German Translation

English-German translations respectively. From table 1 it is evident that the model's performance on the English to Dutch translation is low because Dutch is a low resource language and the datasets available for its translation are scarce. It can be absorbed from table 2 that marianMT transformer performs better than the T5 transformer for English to French translation. Another important point to note is that table 3 shows how there is a difference in performance when two different kinds of datasets for the same source-target language pair performed differently.
Junczys-Dowmunt et al. (2018) can also be referred to observe improvements in performance when marianMT is used when compared to using other machine translation models like RNN and LSTM.

6 Future work

Further work in this area can be performed in comparing the performance of the proposed transformers with respect to a wide variety and languages and datasets. The models previously discussed can be optimized more by using pre-defined optimizer functions, such models can be finetuned to work better on low resource languages in generating pre-trained models for them. Low resource language datasets can be curated and used for training machine translation models to close the gap between existing pre-trained models underperformance on such languages.

7 Conclusion

In conclusion, through comparative analysis I have established that transformers perform better machine translation when compared to existing RNN and LSTM models. MarianMT transformer in particular showed superior performance when compared to the T5 transformer. The best performance of these models can be achieved by using optimizers to produce fine tuned models for language specific and task specific operations. Further experiments and improvements are required to out perform state of the art machine translation models.

References

A. R. Babhulgaonkar and S. V. Bharad. 2017. Statistical machine translation. In *2017 1st International Conference on Intelligent Systems and Information Management (ICISIM)*, pages 62–67.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurélie Névéol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 131–198, Berlin, Germany. Association for Computational Linguistics.

Chris Callison-Burch, Miles Osborne, and Philipp Koehn. 2006. Re-evaluating the role of Bleu in machine translation research. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 249–256, Trento, Italy. Association for Computational Linguistics.

Michael Denkowski and Alon Lavie. 2014. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 376–380, Baltimore, Maryland, USA. Association for Computational Linguistics.

Wei He, Zhongjun He, Hua Wu, and Haifeng Wang. 2016. Improved neural machine translation with smt features. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1).

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. 2018. Marian: Fast neural machine translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia. Association for Computational Linguistics.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. 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, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

B. Premjith, M. Anand Kumar, and K.P. Soman. 2019. Neural machine translation system for english to indian language translation using mtil parallel corpus. *Journal of Intelligent Systems*, 28(3):387–398.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683.

Middi Venkata Sai Rishita, Middi Appala Raju, and Tanvir Ahmed Harris. 2019. Machine translation using natural language processing. In *MATEC Web of Conferences*, volume 277, page 02004. EDP Sciences.

Sandeep Saini and Vineet Sahula. 2015. A survey of machine translation techniques and systems for indian languages. *2015 IEEE International Conference on Computational Intelligence & Communication Technology*, pages 676–681.

Yujin Tang, Jianfeng Xu, Kazunori Matsumoto, and Chihiro Ono. 2016. Sequence-to-sequence model with attention for time series classification. *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 503–510.

Jörg Tiedemann. 2012. Parallel data, tools and interfaces in opus. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA).

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.