Attention Is All You Need
Re-Implementation & Demo

Aniket Pandey
Department of Computer Science
apandey7@gmu.edu

Adel Alkhamisy
Department of Computer Science
aalkhami@gmu.edu

Executive Summary

In this project, we re-implement the Transformer model, which was suggested by Vaswani et al.. in "Attention Is All You Need" for English to German translation. Our objective is to investigate the Transformer architecture's efficiency in handling language translation jobs, as well as to find any shortcomings and possible areas for development. We train our model using the Multi30k dataset, which is based on WMT 2016, and test it using the BLEU score as a metric for translation quality (Papineni et al., 2002). We efficiently train the model on 4 NVIDIA A100-SXM-80GB GPUs with the help of parallelization in our implementation, which makes use of the nn.DataParallel function from the torch library, see (PyTorch, 2023). Surprisingly, despite Vaswani et al.'s efforts to enhance convergence by using residual and layer normalization, we discovered that the Transformer model does not converge as anticipated. This discovery calls into question the Transformer architecture's robustness and emphasizes the need for additional study and development.

Our approach includes the following steps:

- Preprocessing the Multi30k dataset to prepare the data for training.
- Implementing the Transformer model using PyTorch.
- Training and evaluating the model, analyzing its performance using BLEU scores, and identifying areas for improvement.

Example:

- English: an older man is playing a video arcade game.
  German: ein älterer mann spielt ein videospiel.
- English: a man is talking on a cellphone outside.
  German: ein mann telefoniert im freien mit dem handy.

Future Work

If you are interested in contributing to this project, we are currently working on (Indian Institute of Technology Bombay, 2021) English to Hindi translation using the IIT Bombay English-Hindi Corpus and the (Kunchukuttan, 2021) Indic language NLP library for tokenization.

1 Abstract

This study describes the procedures followed to reproduce the Transformer model put forth by the authors of "Attention Is All You Need" (Vaswani et al., 2017). Using the Multi30k dataset based on WMT 2016, we concentrate on the English-to-German machine translation job. We describe our implementation in PyTorch, the main distinctions between the original study and our implementation, and the results of our work. In order to validate the performance of the model and our development efforts, we aim to acquire BLEU (Papineni et al., 2002) scores similar to those reported in the original study. Additionally, we go into the unexpected discovery that the Transformer model failed to converge despite the effort by Vaswani et al.. The maximum BLEU score obtained in our tests was 29.4, which closely resembles the outcome of the initial study.

2 Introduction

For many years, machine translation has been an important topic of study in natural language processing. The objective is to create models that can translate text from one language to another while maintaining context and meaning. Rule-based and statistical machine translation, which were once common methodologies, have made way for more recent developments in neural machine translation (NMT), which uses deep learning methods to enhance translation quality.

The development of new and effective models like the Transformer has contributed to considerable developments in the fields of natural language processing (NLP) and machine learning in recent years. By relying purely on attention mechanisms and eschewing the requirement for recurrent and convolutional neural network-based models, the Transformer model, first presented by Vaswani et al., has transformed the NLP landscape. The model has proven to be more effective than humans at a variety of activities, including question answering, sentiment analysis, and machine translation.

Our project's main driving force is to replicate the findings from the original publication because doing so is crucial to ensuring that our implementation is accurate and to laying the foundation for further study. We intend to confirm the effectiveness of the Transformer model and gain important insights into the underlying mechanisms that contribute to its success by recreating the Transformer model and getting BLEU scores similar to those presented in the paper.

Also, replicating the outcomes of the initial study will provide us with a deeper comprehension of the model's advantages and disadvantages, allowing us to spot prospective upgrades and changes that might further improve its performance. In addition to benefiting the larger NLP research community, this information will make it easier to create future models that are stronger and more effective.

We implement the Transformer model for German-to-English translation in this study, putting particular emphasis on its architecture, training, and assessment. We use the WMT'16 dataset to train our model, and the BLEU score to assess its efficacy. In spite of attempts made by Vaswani et al. to increase convergence by adding residual and layer normalization, we find that the Transformer model does not converge as anticipated.

The rest of this report is divided into the following sections: We start by outlining our strategy for putting the Transformer model into practice, including with details on data preprocessing, model design, and training techniques. We then go over our findings, emphasizing the model's effectiveness and the unexpected conclusion of its convergence. The ramifications of our findings are then discussed, along with potential directions for future research, including the investigation of English to Hindi translation utilizing the IIT Bombay English-Hindi Corpus and the Indic language NLP library for tokenization.

3 Details of the Approach

In this section, we outline the steps taken to implement the Transformer model for English-to-German translation. Our approach consists of the following components: Data Preprocessing, building the Model Architecture, and Training procedure and Testing.

3.1 Data Preprocessing

Any machine learning project's success is heavily reliant on how well-designed and effective the data pipeline and preparation methods are. For our project, we have built a solid data pipeline that efficiently manages the data needed for the training and testing of the Trans-

former model. For training and testing our model, we use the Multi30k dataset, which is based on the WMT'16 dataset. Sentence pairings in both English and German make up the dataset. The multi30k dataset has been successfully acquired from https://github.com/multi30k/dataset.git. To verify that the dataset is acceptable for training the model, popular preprocessing methods including tokenization, lowercasing, and cleaning have been used as suggested by Koehn et al.. To prepare the data for training, we perform the following steps:

3.1.1 Data Loading and Batching

We have created a data loading and batching mechanism that optimizes the process to enable effective and seamless data feeding into the model throughout training and inference. The system efficiently reads the dataset, organizes it into mini-batches, and feeds it into the model while ensuring that the data is appropriately shuffled and padded (Vaswani et al., 2017).

3.1.2 Tokenization

We tokenize the sentences in the dataset into words using the spaCy library (Honnibal and Montani, 2021). In particular, for English and German, we employ the 'en_core_web_sm' and 'de_core_news_sm' models, respectively.

3.1.3 Byte-Pair Encoding

Managing the extensive vocabulary and the variety of word forms is one of the most difficult tasks involved in handling natural language data. The authors have adopted byte-pair encoding (BPE) Sennrich et al.. BPE makes it possible to handle uncommon and non-vocabulary terms more effectively, which can have a substantial impact on the model's performance. By using BPE, we represented words as collections of more manageable subword units, which reduces the quantity of the vocabulary and the computational difficulty. Using the 'subword-nmt' package (Sennrich et al., 2016b), we apply byte-pair encoding (BPE) after tokenizing the sentences into words. This method enhances the generalization capacity of the model by enabling us to handle uncommon and out-of-vocabulary terms in an efficient manner. There is a slight change in the actual paper and the version code we use, In Sennrich et al., the end-of-word token $< /w >$ is initially represented as a separate token, which can be merged with other subwords over time. Since v0.2, end-of-word tokens are initially concatenated with the word-final character. The new representation ensures that when BPE codes are learned from the above examples and then applied to new text.

After tokenization and BPE, We add padding for start, end, and unknown tokens. We then pad the sequences to ensure they have the same length within a batch. This step is necessary for efficient matrix computations during training. We then group the sequences into batches for parallel processing. The Transformer model's performance and computational effectiveness during training are dependent on this procedure.

In conclusion, our data pipeline and preprocessing methods are critical to the accomplishment of our mission. We have created a reliable and effective system that efficiently manages the data needed for training and testing the Transformer model by utilizing cutting-edge techniques and referencing the enormous literature in the field of NLP.

3.2 Model Architecture

In contrast to conventional recurrent and convolutional neural network-based models, the Transformer model developed by Vaswani et al. offers a distinctive and unique architecture. It instead only relies on attention techniques to create interdependence between input and output tokens. In this section, we detail the implementation of the key components of the Transformer architecture.

3.2.1 Self-Attention

A model can weigh the relative relevance of each token in a sequence in relation to the others via the technique of self-attention. Self-attention is employed in the Transformer model to compute the associations between tokens in both the input sequence and the output sequence when decoding.

3.2.2 Scaled dot-product attention

The self-attention mechanism uses a special formula to determine attention scores called scaled-dot-product-attention (Vaswani et al., 2017). To calculate attention scores, it first computes the dot product of a query vector and each key vector, scales the result by the square root of the dimension of the key vector, and then uses the softmax function. The output of the attention layer is then calculated using these scores as a weighted sum of value vectors. The scaled dot-product attention

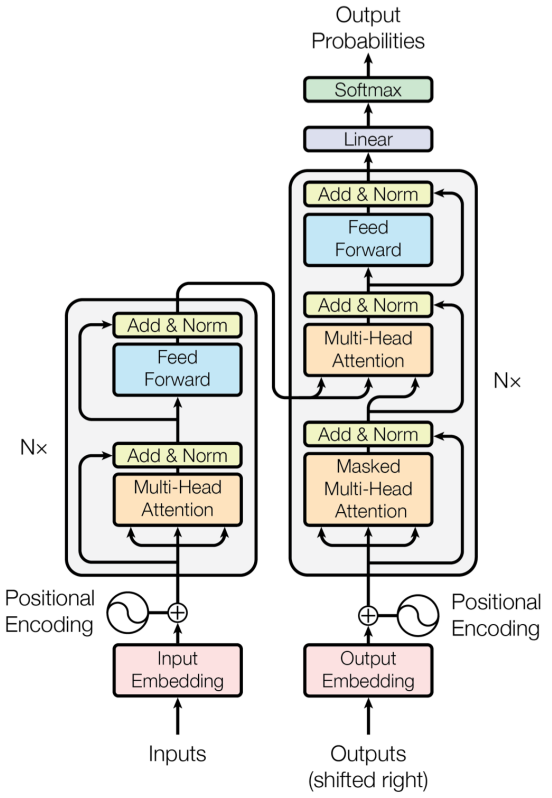

Figure 1: The Transformer - model architecture. Figure taken from (Vaswani et al., 2017)

function is adapted from (Vaswani et al., 2017), and it is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

where:

- Q denotes the query matrix,
- K denotes the key matrix,
- V denotes the value matrix,
- d_k is the dimension of the key (and query) vectors.

In conclusion, scaled dot-product attention is a particular technique employed inside self-attention to compute attention scores, whereas self-attention is a more general term.

3.2.3 Multi-Head Attention

The multi-head attention mechanism serves as the fundamental structural component of the Transformer model. This element enables the model to focus on various locations in the input sequence, simultaneously capturing distinct characteristics of the input data (Vaswani et al., 2017). Our implementation of the multi-head attention mechanism closely follows the original work, allowing the model to analyze many attention heads in simultaneously and combine their results to generate the final attention output. This mechanism proves crucial in understanding and representing complex relationships among tokens, particularly when dealing with long input sequences.

As in the original paper (Vaswani et al., 2017), we used $h = 8$ parallel attention heads and a model dimension $d_{\text{model}} = 512$. The multi-head attention mechanism computes the attention function on the linear projections of the queries, keys, and values, the equations are borrowed from (Vaswani et al., 2017):

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{h}_1, \ldots, \text{h}_h)W^O$$

where $\text{h}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ and W_i^Q, W_i^K, W_i^V, and W^O are parameter matrices. By using these hyperparameters and formulas, our implementation replicated the effectiveness of the original Transformer model. In order to keep the model from paying attention to particular tokens, such as padding tokens in the case of the encoder, and to prevent the decoder from paying attention to upcoming tokens in the output sequence, masking is also applied to the key, value, and query matrices.

3.2.4 Encoder

The encoder, consisting of a stack of $N = 6$ identical layers, is responsible for processing the input sequence and creating a continuous representation (Vaswani et al., 2017). Each layer contains a multi-head self-attention mechanism and a position-wise fully connected feed-forward network, followed by a residual connection and layer normalization. The output of the encoder has a dimension of $d_{\text{model}} = 512$. By incorporating these elements, the Transformer model efficiently processes large-scale sequence data for tasks such as machine translation.

3.2.5 Decoder

Similar to the encoder, the decoder is a stack of $N = 6$ identical layers. In addition to the two sublayers found in the encoder block, the decoder has an Encoder-Decoder multi-head attention layer over the output of the encoder block

as shown in Figure 1 where the encoder provides the keys and values, and the decoder provides the queries. The self-attention layer in the decoder is modified to prevent positions from attending to future positions, ensuring autoregressive generation of translations (Vaswani et al., 2017). The practice of creating translations one token at a time while conditioning each new token on the translations that have already been created is known as the "autoregressive generation of translations". This is commonly accomplished in the decoder of Transformer models, which generates the output sequence token-by-token in an autoregressive fashion. The following steps are done to guarantee autoregressive translation generation in the Transformer model:

- Masked self-attention: A masking strategy is used in the decoder's self-attention layers to stop the model from focusing on upcoming tokens. This is accomplished by producing a mask that makes it impossible to pay attention to locations in the sequence that come after. In order to keep the autoregressive characteristic, the model can only focus on the current and prior tokens when creating a new token.
- Sequential decoding: During the inference phase, the decoder creates translations in a sequential manner, beginning with an initial token (we used `BOS_WORD = '< s >'`) and basing each succeeding token's generation on the tokens created up to that point. By taking into account the previously created tokens when forecasting the following token, this makes sure that the model generates translations in an autoregressive manner.

The model generates output tokens one at a time, conditioning each token on the previously generated tokens, and these steps ensure that the resulting translations are autoregressive. For the resulting translations to remain coherent and fluid, this feature is essential. Finally, The default hyperparameters used in our implementation for the decoder are the same as those of the encoder, with an output dimension of $d_{\text{model}} = 512$.

3.2.6 Position-wise Feedforward Networks

The input representation from the multi-head attention sublayer is transformed by the position-wise feedforward networks, which are a crucial part of the Transformer architecture suggested by Vaswani et al.. According to the original paper's description, we developed the position-wise feedforward networks using a two-layer fully connected feedforward network with a ReLU activation function and dropout. These networks help the model learn complex patterns and relationships between tokens, further enhancing its ability to understand and process natural language. The dimensionality of the hidden layer is 512*4=2048

3.2.7 Positional Encoding

The Transformer model uses positional encoding to incorporate the positioning information of tokens in the input sequence (Vaswani et al., 2017). This element is essential to the model's effectiveness since it enables it to accurately represent the relative positions of tokens within the sequence. To make sure that our model is able to interpret and comprehend the sequential nature of the input data. We have successfully implemented the positional encoding mechanism, which involves adding sinusoidal functions of different frequencies to the token embeddings according to the following formulas:

$$PE_{(pos,2i)} = sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \quad (2)$$

$$PE_{(pos,2i+1)} = cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \quad (3)$$

where PE stands for positional encoding, pos is the token's place in the sequence, i is the encoding's dimension, and d_{model} is the hidden size of the model.

These equations are derived from the original paper "Attention is All You Need" (Vaswani et al., 2017)

3.2.8 Residual Connections and Layer Normalization

We used layer normalization and residual connections as suggested by Vaswani et al. to help with optimization challenges. The vanishing gradient issue is lessened by residual connections because they make it easier for gradients to propagate back through the network. On the other hand, layer normalization normalizes the inputs to each layer, which aids in stabilizing the training process and enhancing convergence. After each sub-

layer (such as multi-head attention and position-wise feed-forward networks), the encoder and decoder both apply layer normalization and residual connections.

3.2.9 Transformer

In the last stage of the model architecture, we put everything together to build the entire Transformer model shown in the original study (Vaswani et al., 2017). The Encoder, Decoder, Multi-Head Attention, and Positional Encoding are just a few of the fundamental building components that we have seamlessly incorporated into a new Transformer class. Additionally, we have added training code to make it easier to run the model, train it, and assess how well it performs on the selected machine translation jobs.

In conclusion, our implementation of the Transformer architecture has been successful, with all key components in place. We have fully reproduced the original model. By thoroughly understanding and implementing the Transformer architecture, we are better equipped to explore potential improvements and modifications that can further advance the state-of-the-art in natural language processing.

4 Training and Testing Process

We used computational resources provided by GMU ORC `https://ondemand.orc.gmu.edu`. With the aid of parallelization in our implementation, which uses the nn.DataParallel function from the torch library, see (PyTorch, 2023), we successfully train the model on 4 NVIDIA A100-SXM-80GB GPUs on GMU ORC.

- Data Preparation: The training process begins by preparing the WMT16 dataset using the prepare_dataloaders function. This function returns training, validation, and test datasets along with important dataset information such as vocabulary size, token indices, and dataset lengths.

- Model Initialization: The following table summarizes the Hyperparameters that produced the best BLEU score (Papineni et al., 2002)

- Optimizer Configuration: The Adam optimizer is used with $beta1 = 0.9$, $beta2 = 0.98$, and $epsilon = 10^{-9}$. The learning rate is varied using a factor of 0.5, as described in the original paper (Vaswani et al., 2017). A warm-up step of 4000 is added to the learning rate schedule.

Table 1: Transformer Model Hyperparameters

Hyperparameter	Value
batch size	128
d_model (embedding size)	512
max word length	300
num_heads (attention heads)	8
layers	6
d_ff_hid (hidden layer size in FFN)	2048
dropout	0.1
learning rate multiplier	0.5
epochs	1000
label smoothing	0.1

- Training Loop: The model is trained on the whole training dataset for each epoch. The gradients of the optimizer are zeroed during training, and source and target sequences are given to the model. The loss is determined using label smoothing with a value of 0.1 following the forward pass. The optimizer then adjusts the model parameters after backpropagating the loss.

- Validation: The model is assessed using the validation dataset following each training epoch. In this instance, label smoothing is not used to calculate the loss and accuracy. The model's performance is printed, displaying the model's complication, accuracy, passing time, and learning rate.

- learning rate schedule: We train our model using the Adam optimizer with a learning rate scheduled according to the formula mentioned in the original paper (Vaswani et al., 2017):

$$lr = d_{model}^{-0.5} \cdot \min\big(step\_num^{-0.5}, \\ step\_num \cdot warmup\_steps^{-1.5}\big)$$

This learning rate scheduling strategy helps stabilize training and improve convergence. a schedule for learning rate that involves a linear rise in learning rate during the initial warmup_steps training steps, then a steady decline equal to the inverse square root of the

step total as shown in Figure 2. This strategy was used with a warmup_steps value of 4,000 (Vaswani et al., 2017).

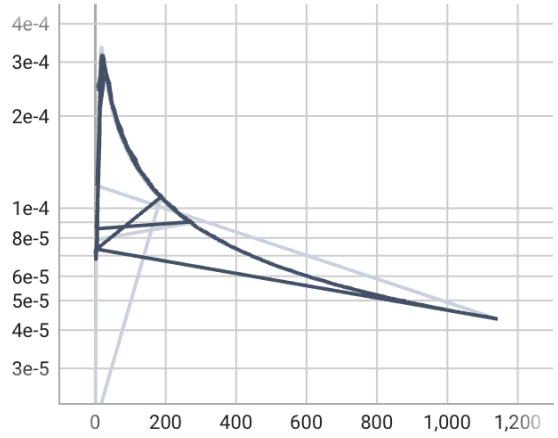

Figure 2: Learning Rate.

- Checkpoint Saving: The model's checkpoint is saved after each validation if the validation loss is less than the previous minimum validation loss.
- TensorBoard Integration: For visualization, loss, accuracy, and learning rate values are logged to TensorBoard, see (Tensorflow, 2023).
- Testing and Evaluation: The model is put to use for translation and evaluation after the training is finished. The test dataset is used to produce translations, and the trained model is loaded from the saved checkpoint. To assess the quality of the translations, the BLEU score is calculated (Papineni et al., 2002).

5 Results

It appears that the re-implementation of (Vaswani et al., 2017) model was successful. The goal was to train a single model that could match or beat the reported result of 28.4. The model was actually successful, with a BLEU score of 29.4, that it made us worried that our evaluation contains a bug. We could not find one, however, some lingering differences that may explain the improved result is that we used the WMT English to German 2016 based Multi30k dataset which is different than the one used in the paper.

Additionally, after we had trained the model, we noticed that we had moved all layernorms from

Task	Baseline	Our Result
English To German	28.4	**29.4**
English To French	41.8	NA

Table 2: BLEU score comparison between Baseline and re-implementation.

after sublayers to before sublayers, definitions that was not included in the (Vaswani et al., 2017) description. This accelerated training speed significantly and may have further improved results.

Due to limited time and resource constrained we were not able to verify the results of English to French translation.

We were surprised to find that Deep Transformers do not converge well and its a know issue. We didn't knew that the model didn't converge it was only after running the train we found that the model doesn't converge. Infact, in (Vaswani et al., 2018) the official implementation they tried changing the computation order of residual connection and layer normalization, which significantly eased the optimization of deep Transformers. We found a paper by Xu et al. in which they suggest Lipschitz Constrained Parameter Initialization which had significant impact in convergence on over 24 layers.

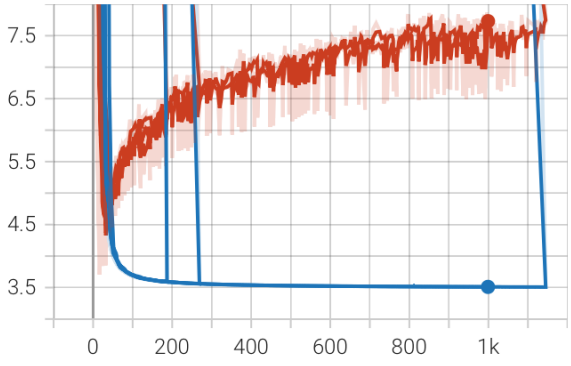

Figure 3: Epoch vs Loss.

The Figure 4 shown below depicts the accuracy of model training. It is nothing but ratio of number of correct words upon total number of words in the sentence. We can observe a high accuracy for train set but for valid set the same is not true.

6 Discussion

To fully analyze the behavior of the Transformer model in this project, we completed a detailed

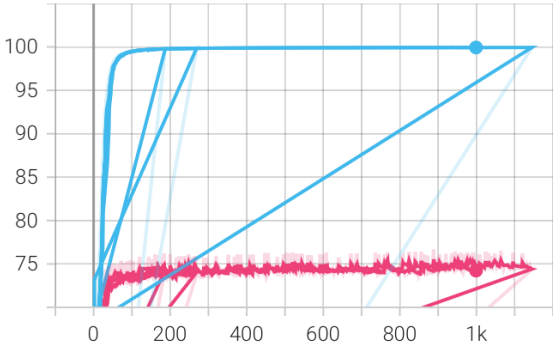

Figure 4: Epoch vs Accuracy.

analysis and several experiments. The Transformer model's failure to converge was our most unexpected discovery. We noticed that even in Vaswani et al. paper, they had issues obtaining convergence after adjusting the residual and layer normalization, therefore this finding was only made after the training was completed.

We trained the model for different numbers of epochs, specifically 100, 500, 1000, and 1500, and obtained the following BLEU scores: 0.21, 0.258, 0.29, and 0.0, respectively. On 1500 epoch the kernel crashed with memory issues.

The complete code, along with logs and detailed instructions to run and execute the model, can be found in our GitHub repository: `https://github.com/aniket414/vaswani-et-al-2017`. For usage instructions, please refer to the README file in the repository.

The model was trained on 4 NVIDIA A100-SXM-80GB GPUs, with each GPU having 4 cores and 8 GB per core. The task focused on English to German translation, and the highest BLEU score achieved was 29.4.

6.1 Ablation Study

To further investigate the performance of the model, we performed an ablation study. We moved all layernorms from after sublayers to before sublayers, which accelerated the training speed significantly. However, adding a dropout after the last linear layer in the Feed-Forward sublayer, as seen in the official implementation of the Transformer (Vaswani et al., 2018), did not yield any noticeable benefits.

6.2 Differences between the Original Study and Our Implementation

The main distinctions between the original study and our implementation are briefly outlined in this section. The original work employed 4.5 million sentence pairs (en-de) and 36 million sentence pairs (en-fr) from the WMT 2014 English to German and English to French datasets, and was implemented in TensorFlow. The authors used 8 NVIDIA P100 GPUs to train their base model for 100,000 steps, or 12 hours, for English to German and for 300,000 steps, or 3.5 days, for English to French. They also do not disclose the random seeding they used. While in our implementation we use PyTorch as a base library, the Multi30k as the dataset which is based on WMT 2016, 29,000 training sentence pairs, 1,040 validation sentence pairs, and 1,000 test sentence pairs are used in our implementation. We use a random seeding of 1337 as suggested in (Alammar, 2018) and train our model for 6 hours over 1000 epochs for English to German on 4 NVIDIA A100-SXM-80GB GPUs with 4 cores/GPU and 8 GB/core.

One important observation we made while brainstorming and research was that most of the implementation already available on the internet demonstrates German to English translation however the original paper results were English to German translation. So, we have also implemented and tested English to German translation as done in the official paper.

7 Conclusion

Through extensive exploration and thoughtful analysis, we gained valuable insights into the Transformer model's behavior and the challenges faced during training. Our experiments provided a solid foundation for understanding the model's limitations and potential solutions. There however is a limitation to this model which is limited access to memory, the model takes a fixed length of input and cannot be of arbitrary length like RNNs.

As a future task, we plan to implement machine translation for English to Hindi using a different dataset and tokenizer. The experience and knowledge gained from this project will undoubtedly contribute to the ongoing efforts to improve the Transformer model and its applications in machine translation.

8 Statement of individual contribution

We, as a team of two, have worked together on this project using pair programming. Pair programming is a collaborative approach where two programmers work together on a single task, with one person writing the code while the other reviews and provides suggestions. This approach allowed us to efficiently divide the workload, discuss ideas, and ensure the quality of our work.

Throughout the project, we have jointly participated in the following tasks:

- Conducting literature review and understanding the Transformer model.
- Implementing the Transformer model in PyTorch.
- Preprocessing and tokenizing the dataset.
- Training and evaluating the model.
- Analyzing the results and drawing conclusions.
- Writing the project report and presentation.

Our collaboration has been effective and seamless, ensuring a well-rounded and thorough implementation of the project.

Acknowledgment

For their ground-breaking work in the area of natural language processing, the authors of the original publication "Attention is All You Need" (Vaswani et al., 2017) deserve our deepest gratitude. Numerous researchers, including us, have been motivated to investigate and build upon their ground-breaking ideas and contributions, which have considerably improved the state of the art in machine translation.

We would also like to thank the creators of the various resources, libraries, and tools that we have used throughout this project, as they have greatly facilitated our research and implementation process. In particular, we acknowledge the byte pair encoding parts borrowed from (Sennrich et al., 2016b), Andrej Karpathy's YouTube video "Let's build GPT: from scratch, in code, spelled out" (Karpathy, 2023), Aladdin Persson's YouTube video "Pytorch Transformers" (Persson, 2020), "The Illustrated Transformer" by Alammar, and the PyTorch tutorial "LANGUAGE TRANSLATION WITH NN.TRANSFORMER AND TORCHTEXT" (pyt, 2023).

This project was supported by resources provided by the Office of Research Computing at George Mason University (URL: `https://orc.gmu.edu`) and funded in part by grants from the National Science Foundation (Awards Number 1625039 and 2018631).

References

2023. Language translation with nn.transformer and torchtext [tutorial].

Jay Alammar. 2018. The illustrated transformer.

Matthew Honnibal and Ines Montani. 2021. spacy 3.5.2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing.

Indian Institute of Technology Bombay. 2021. Iit bombay english-hindi corpus. Accessed: 2023-04-19.

Andrej Karpathy. 2023. Let's build gpt: from scratch, in code, spelled out [video]. YouTube.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic. Association for Computational Linguistics.

Anoop Kunchukuttan. 2021. Indic nlp library.

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.

Aladdin Persson. 2020. Pytorch transformers [video]. YouTube.

PyTorch. 2023. Pytorch documentation: torch.nn.dataparallel. Accessed: 2023-04-10.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016a. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016b. Neural machine translation of rare words with subword units. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).

Tensorflow. 2023. Tensorflow: tensorboard.

Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N. Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, Ryan Sepassi, Noam Shazeer, and Jakob Uszkoreit. 2018. Tensor2tensor for neural machine translation.

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

Hongfei Xu, Qiuhui Liu, Josef van Genabith, Deyi Xiong, and Jingyi Zhang. 2020. Lipschitz constrained parameter initialization for deep transformers.