Neural Machine Translation from German to English with Transformer on Multi30K Dataset

Yulun Zhuang*, Mandan Chao*, Baiyue Wang*

June 6, 2021

* Department of Mechanical and Energy Engineering, SUSTech

1 Introduction

Before Transformer [1], machine translation were generally treated by recurrent neural networks with encoder-decoder architectures [2, 3, 4, 5]. Those recurrent neural networks typically suffer from their sequential nature of computation, precluding parallel computation during training, and therefore become limited in efficiency when the inputs are long sequences. Transformer is designed to solve these problems, who abandoned a recurrent architecture and instead was built upon an entirely attention-based mechanism.

Models that use convolutional neural networks require a large number of operations to relate two signal positions, and the operations grow with respect to the distance between two signals [6, 7, 8]. Transformer makes this operation number constant. It is also the first machine translation model that dose not rely on sequence-aligned RNNs or convolution, but relies only on self-attention, an attention mechanism relating different positions of a sequence to compute a representation.

Our Neural Machine Translation (NMT) model is based on Transformer, whose architecture will be introduced in details in the following sections. In general, the model contains an encoder-decoder structure, with *positional encoding* to deal with positional information of a sequence, *attention* functions to calculate query, keys and values, and *feed-forward networks*.

2 Problem Definition

In this project, we are going to design a neural machine translator which translate German to English in order to gain a higher BLEU score. This project must be completed without plagiarism. We will clearly explain our algorithm and evaluate the performance of our method on the provided test dataset with BLEU evaluation metric. We will use Multi30K dataset for this task. It contains around 30,000 parallel English, German and French sentences, each with about 12 words per sentence.

To make sure that the results can be reproduced, we will explicitly set the random seeds for deterministic results. During the training, a validation set will be used to monitor the overfitting status. We will use the loss curve of our training set and validation set to evaluate the training result.

3 Data Preparation

Data preparation is of vital important for any kinds of Machine Learning (ML) tasks. The Multi30K dataset has $29k$ training sentences, $1k$ validation sentences and $1k$ test sentences. In training sentences, it has 18669 and 9795 unique words for German and English vocabulary respectively.

In our approach, we use spaCy trained tokenizer *en_core_web_sm* for English and *de_core_news_sm* for German. Our data preparation contains 5 steps.

- Tokenization: break a piece of text down into a list of tokens.
- Building vocabulary: use a dictionary (hash table) to count word frequencies.
- One-Hot Encoding: use the dictionary to map words to a list of indices, so-called sequence.
- Batchtify: transform a dataset into mini-batches that can be processed efficiently.
- Sentences Aligning: pad sentences with zeros to the maximum length in one batch.

These five steps should be done for training, validation and test data. Functions *build_vocab*, *sen2tensor*, *batchtify* in *utils.py* are implemented for this section.

4 Model Architecture

The model is built upon an *encoder* and a *decoder*, where the *encoder* encode the source sentence into context vector and the *decoder* decode the context vector to the target sentence. Within the *encoder* and the *decoder*, there are the following layers that exploit information from the sentences.

4.1 Embedding & Positional Encoding

Before the input tokens are fed into multi-head layers, they need to first be put into a standard *embedding layer*, as well as a *positional encoding layer*. The *positional encoding layer* provide the tokens with their position information. In the previous implementation of *positional encoding* in [1], they used a fixed static embedding. However, in our approach, a learned *positional embedding* is applied instead.

The token embeddings are multiplied by a scaling factor $\sqrt{d_{model}}$ to reduce variance, and a dropout is applied. Then the two embeddings are summed together to form a vector with information of both the token and their position in the sequence. The combined embeddings are then fed into the encoder layers.

4.2 Encoder

4.2.1 Attention

Attention is a mechanism that focuses a sentence on its emphasis. It can be thought of as queries Q, keys K and values V. The **Transformer** uses *scaled dot-product attention* between Q and K, and applies the softmax operation and scales by the *head dimension* d_k and then multiply the data by V.

$$\text{Attention}(Q, K, V) = \text{Softmax}(\frac{QK^T}{\sqrt{d_k}})V. \tag{1}$$

In fact, instead of using a single attention, the three matrix Q, K and V are split into h heads, which are then calculated in a scaled product attention in parallel. Therefore, the Multi-head Attention pays attention to h different concepts.

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V), \tag{2}$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O, \tag{3}$$

where matrices QW^Q, KW^K and VW^V are the outputs from linear layers, fed by Q, K and V. W^O is the linear layer applied at the end of the multi-head layer.

4.2.2 Feedforward Networks

In this layer inside the encoder layer, the vectors from multi-head attention is of hidden dimension. This *position-wise feedforward layer* transform the vectors from hidden dimension to a larger feedforward dimension, and are applied with the ReLU activation and dropout, and then transform into hidden dimension again. The reason behind this layer is unexplained in [1].

4.3 Decoder

The outputs, or the target sentences, go through the same *embedding* and *positional encoding* layers before decoder. These two results are also combined and fed into the decoder layers.

The decoder layers function as a translator that translate the vectors from encoder to the target sentence. The decoder contains two multi-head attention layers: a *masked multi-head attention layer* over the target sequence, and a *multi-head attention layer* that uses the encoder outputs as K and V and the decoder representation as Q.

The first *multi-head attention layer* can be seen as a self-attention, as it is in the encoder. The output vectors from *standard embedding* and *positional encoding* represent K, Q and V themselves. This layer also uses the target sequence mask, which is said to prevent the "leftward information flow".

The second layer takes vectors from both encoder and decoder, as we mentioned above. A dropout is applied to the output of this layer.

Finally, the output is passed to a *feedforward layer* in Section 4.2.2 and another dropout, residual connection and layer normalization [9].

4.4 Sequences to Sequences Model

The *Seq2Seq* module is therefore the combination of the encoder and the decoder. Also, masks are created inside the module.

The *source mask* is used to check whether the source sequence is not equal to a $< pad >$ token. This mask makes sure the input is correctly broadcast.

The *target mask* has the same mask for the $< pad >$ tokens, and is followed by a *subsequent mask*. This mask creates a diagonal triangle down-left matrix, where the elements above the diagonal is set to zero and below is set to the input value. For example, the mask looks like

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix} \tag{4}$$

Then the two masks sum together and is used with the encoder and decoder along with the source and target sentences.

5 Experiments

This section describes the training policy for our models.

5.1 Hyperparameters

Overall there are 10 hyperparameters (Table 1) affecting the model performance significantly.

Table 1: Hyperparameter Notations

B	=	Batch size
N_{enc}	=	Encoder layers numbers
N_{dec}	=	Decoder layers numbers
d_{model}	=	Embedding dimension
d_{ff}	=	Feedforward network dimension
h	=	Head numbers of multihead attention layer
P_{drop}	=	Dropout rate
n	=	Training epoch numbers
LR	=	Learning rate
Fre	=	Min frequencies of words for building vocabulary

5.2 Hardware

We trained our models on one laptop with NVIDIA GTX1060 GPU and one GPU server with NVIDIA GTX2080 GPU respectively. For our average models using $10M - 50M$ parameters, each training epoch took about 60 seconds on laptop and 30 seconds on GPU server. We trained the base models for a total of 40 epochs.

5.3 Training Setup

We design our loss function, optimizer and regularization to be as close as those described by *Ashish et al*[1]. However, we design our learning rate to be a constant, which was proven to be more efficient than a noam decay function[10].

Table 2: Training Setup

Loss Function	*nn.CrossEntropyLoss*
Optimizer	*optim.Adam* with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$
Regularization	*nn.Dropout* and *nn.LayerNorm*.

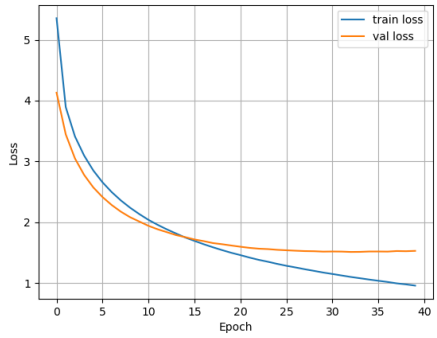
6 Results

The highest BLEU score for our model is 37.39.

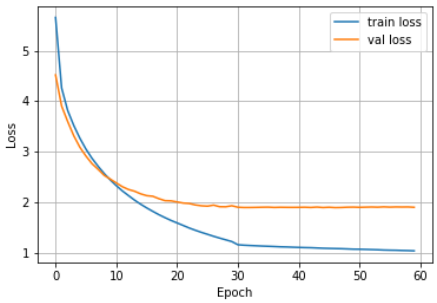
6.1 Model Evaluation and BLEU Scores

We use training loss and validation loss to evaluate our models directly. We observe that Transformer with the half size parameters as described by *Ashish et al*[1] will perform best. As required for competition purpose, we also use Bilingual Evaluation Understudy (BLEU) as our evaluation metric. BLEU has been a de-facto standard for evaluating translation outputs since it was first proposed in 2002. We compute the BLEU score of our de-en model with the $multi - bleu.perl$ script.

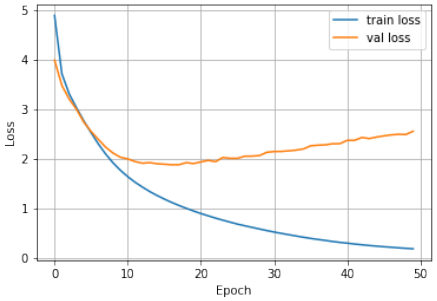
Since our models are training on a quite small dataset , overfitting becomes the most troublesome challenge shown in Fig.1.



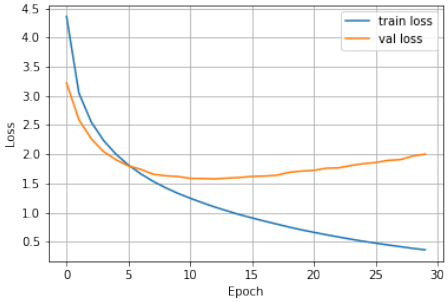
(a) Good training curve with $P_{drop} = 0.2$ and $Fre = 2$



(b) Oscillating with large $P_{drop} = 0.4$



(c) Overfitting at $val\_loss \approx 2$



(d) Overfitting at $val\_loss \approx 1.5$

Figure 1: Overfitting training and validation loss curves

6.2 Hyper-Parameters Tuning

In this project, we have tuned a large set of hyperparameters to gain a higher BLEU score from Transformer. As shown in Table 3, models named with full date were trained on our local laptop and those named with partial date were trained on a GPU server.

To evaluate the importance of different components of the Transformer, we varied our model in different ways. We found that BLEU drops off with too many heads but BLEU stay unchanged from 4 to 8. In Table 3 rows 1 to 4, we vary the embedding size from 512 to 1024 and encountered great overfitting due to the large number of embedding layer parameters. We further observe in rows 11 and 11 that, not as expected, bigger models are not better, and dropout is very helpful in avoiding over-fitting.

We have also encountered two strange evidences. Due to the difference of PyTorch version, we get different vocabulary size from laptop to server with same code. That is to say, the model we trained on server can not be evaluated on local laptop. Another strange point is that BLEU score keep raising as the min frequencies of words for building vocabulary increasing. As Fre increasing from 1 to 3, the highest BLEU scores raising from 34.36 to 37.39.

Table 3: Hyperparameter Tuning

$Model$	$BLEU$	$Loss$	Fre	N_{enc}	N_{dec}	d_{model}	d_{ff}	P_{drop}	B
5-18	33.13		1	3	3	512	512	0.1	128
5-19-0.5	34.36		1	3	3	512	512	0.5	128
5-19	28.77		1	4	4	512	512	0.1	128
5-19-base	29.16		1	6	6	512	1024	0.1	64
5-19-base2	35.7		2	3	3	512	1024	0.1	64
5-20-1	30.64		2	3	3	256	512	0.3	256
5-20-2	30.29		1	3	3	256	512	0.2	256
5-20-3	34.61		1	3	3	512	2048	0.3	64
5-20-4			1	3	3	256	512	0.1	128
21-7	32.86	1.843	1	3	3	512	1024	0.2	128
21-6	36.06	1.617	2	3	3	256	256	0.1	256
21-8	34.57	1.589	2	3	3	256	512	0.1	128
21-9	33.69	1.849	1	3	3	512	512	0.1	128
my-21-11		fail	1	3	3	512	512	0.1	128
my-21-11		fail	1	3	3	512	512	0.3	128
21-11(layernorm)	31.78	1.79	1	2	2	512	1024	0.1	128
21-11(layernorm)	33.28	1.848	1	2	2	512	2048	0.2	256
6-5-1-best	37.39		3	3	3	256	512	0.1	128

6.3 Translation Results

In this section, we implemented two translation algorithms, greedy search and beam search. Greedy Search algorithm selects one best candidate as an input sequence for each time step. Choosing just one best candidate might be suitable for the current time step, but when we construct the full sentence, it may be a sub-optimal choice. The beam search algorithm selects k (beam width) alternatives for an input sequence at each timestep based on conditional probability. At each time step, the beam search selects k number of best alternatives with

the highest probability as the most likely possible choices for the time step. Superisingly the changing range of BLEU is within 0.2. The sample translation sentences are shown in Table 4.

Table 4: Sample Translations

src	ein mann mit freiem oberkörper und shorts steht auf ein paar steinen und angelt.
ref	a shirtless man in shorts is fishing while standing on some rocks .
best	a shirtless man wearing shorts and shorts is fishing on some rocks and fishing .
base	a shirtless man wearing no shorts stands on a few rocks and fishing .
src	zwei männer in badehosen springen auf einem mäßig belebten strand in die luft.
ref	two men wearing swim trunks jump in the air at a moderately populated beach .
best	two men in swim trunks jumping on a busy beach in the air .
base	two men in swim trunks jump into the air on a busy beach in midair .

7 Conclusion

In this work, we implemented a Transformer architecture to realize a full attention neural network that learns to translate German to English. The source language and the target language were processed to tokens that represent the corresponding words, and then they were encoded to an Encoder-decoder structure that use multi-head attention layers and feed forward layers to gather information from the sentence. A learned network then is capable of translation.

Our experiments show that after adequate tuning of the hyperparameters, the model generates a *BLEU* score up to 37.39, when the minimum frequency of words is selected to be 3. It then occurred to us that the minimum frequency of words has the greatest effect on the performance of our model. However, the higher the minimum frequency is set, the more rare words will be lost during the translation. Therefore, the consequence of lost words indicates that in order to train a more robust network, a larger training set that contains necessarily large number of word frequency is needed.

8 Contributions

Yulun Zhuang: Model implementation and parameter tuning.
Mandan Chao: Report writing and model comparison;partial work on code realization and parameter tuning.
Baiyue Wang: Model selection and comparison; partial work on code realization and parameter tuning.

A Appendix Code

Our source code can be acquired from GitHub.

References

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

[2] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation, 2015.

[3] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling, 2016.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[5] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[7] Łukasz Kaiser and Samy Bengio. Can active memory replace attention?, 2017.

[8] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time, 2017.

[9] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.