Final Report

Zhuang Yulun 11811126

November 21, 2023

1 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.

2 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.

3 Experiments

This section describes the training policy for our models.

3.1 Hyperparameters

Overall there are 9 hyperparameters (Tab.1) affacting 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

3.2 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.

3.3 Training Setup

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

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*.

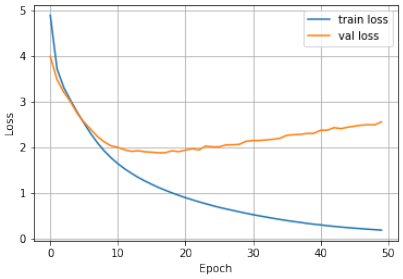
4 Results

The highest BLEU score for our model is 37.14.

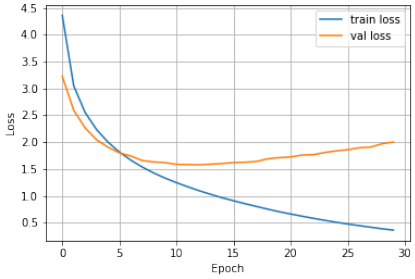
4.1 Model Evaluation and BLEU Scores

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*[**?**] will perform best. As required for compotetion 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) Overfitting at $val\_loss \approx 2$



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

Figure 1: Overfitting training and validation loss curves

4.2 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 data were trained on our local laptop and those named with partial data 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 min_fre increasing from 1 to 3, the highest BLEU scores raising from 34.36 to 37.14.

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	37.14		3	3	3	256	512	0.1	128

4.3 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.

German-English translations

src	In einem Interview sagte Bloom jedoch , dass er und Kerr sich noch immer lieben .
ref	However , in an interview , Bloom has said that he and *Kerr* still love each other .
best	In an interview , however , Bloom said that he and *Kerr* still love .
base	However , in an interview , Bloom said that he and **Tina** were still <unk> .
src	Wegen der von Berlin und der Europ
ref	The imposed on national economies through adherence to the common currency , has led many people to think Project Europe has gone too far .
best	Because of the strict respective national economy is forced to adhere to the common currency , many people believe that the European project has gone too far .
base	Because of the pressure ct has gone too far .

Table 4: **Sample translations** – for each example, we show the source (*src*), the human translation (*ref*), the translation from our best model (*best*), and the translation of a non-attentional model (*base*). We italicize some *correct* translation segments and highlight a few **wrong** ones in bold.