Machine Translation: Project Exam
Sagittarius-A

Monique Noss, M.-Nr.: 794007

August 15, 2021

This report documents the results for the final research project for the class "Machine Translation".

This project implements a MT system using the OpenNMT library and was tested and evaluated using BLEU. The task was to train a model that performs as well as possible. This involves:
• careful preprocessing with sensible options (e.g., number of BPE units),
• choosing a good model architecture with good parameters,
• training the model for long enough,
• choosing useful training data.

This report describes the experiments, detailing the modelling choices and discusses failed experiments.

The name of the final model "Sagittarius-A" is derived from the black hole in the Milky way with the same name, being a play on the amount of resources (energy, time) it takes to train such a big, good-working model. I came up with this, naming planned models first after stars and moons, but then decided to name the biggest and best model after the black hole that could basically swallow all of these.

The final code used in the course of the project, can be found in this Git repository:
https://github.com/mmonimon/Sagittarius-A
It contains all of the different bash and python scripts used to automate the training process.

1 WMT

For this project, I selected the language pair English → German for my experiments, because German is my native language which enables me to make judgements about a correct translation in terms of grammar, vocabulary and phrasing best. I was also curious about which score one can reach as an "amateur" who has not had a lot of experience with MT models, as it is known that English-German translations can reach very high BLEU scores.

2 Hardware and Software

During the execution of this project the provided server with a NVIDIA GeForce GTX 1080Ti broke down. Therefore, a majority of the training was performed on a desktop PC with the following specifications:

- Intel Core i9-9900K
- 32 GB main memory
- NVIDIA GeForce RTX 3080 with 10 GB GPU memory

Throughout the project, the training was performed using the following software versions:

- Windows 10
- NVIDIA driver 471.41
- NVIDIA CUDA 11.4
- PyTorch 1.9.0 with Cuda 11 support
- torchtext 0.4.0
- subword-nmt 0.3.7
- OpenNMT-py 1.2.0
- moses preprocessing tools from provided server

3 Data

I decided to download several of the corpora provided by WMT 2019 and 2020 to see how they look. After having a look at each of the corpora, I decided for the following three corpora from statmt.org, targeting a subset of the main sources for the WMT 2020 (http://www.statmt.org/wmt20/) for DE-EN translations. Below are the download links and names of the corpora I used:
- **Europarl corpus** (http://www.statmt.org/europarl/v10/training/europarl-v10.de-en.tsv.gz),
- **The news-commentary corpus** (http://data.statmt.org/news-commentary/v15),
- **The ParaCrawl corpus** (https://s3.amazonaws.com/web-language-models/paracrawl/release5.1/en-de.txt.gz).

On a first glance, these resources seem to be formatted already in a way that is easy to preprocess in terms of data splitting (German/English corpus, train/test/val data sets).

Furthermore, I used the datasets provided on Moodle for development which correspond to plain text versions of the news dev sets for writing a bash script that helped me automating the model building process, the pre-processing and testing procedures. The final version of the script can be found in the Git repository.

To try out different parameters for building and testing the translation model, I however used a bigger data set, in particular I started using the ParaCrawl corpus, because it was already formatted as desired (one sentence per line).

4 Modelling options

4.1 Preprocessing

4.1.1 Data split

As a first step, I split up the data into English and German corpora for all of the three downloaded data sets (the newstest corpus was already split up in two pieces). As these are all structured the same way (German-English translation in one line, separated by a tab), I used a simple "cut -f1/2" command to separate the German from the English data and saved them into a new file using a matching language suffix (e.g. europarl-corpus.de). For Europarl, I tried using sentence_split.perl script from our lecture in order to put each sentence into one line. However, I noticed that the amount of lines will differ after execution, so I reverted the change. For News-Commentary, I additionally needed to remove empty lines from the corpus (a 'sed -i '/^$/d' command did not do the job here as this also resulted into a different line count), so I wrote a small script in Python that converts the big corpus into two smaller ones:

```
with open('data/news-commentary-v15.de-en.tsv', mode='r', encoding='utf8') as data:
    with open('data/news-commentary-corpus.de', mode='w', encoding='utf8') as de:
        with open('data/news-commentary-corpus.en', mode='w', encoding='utf8') as en:
            for line in data:
                splitted_line = line.strip().split('\t')
                if len(splitted_line) != 2: continue
                if len(splitted_line[0]) == 0 or len(splitted_line[1]) == 0: continue
                de.write(splitted_line[0]+'\n')
                en.write(splitted_line[1]+'\n')
```

The split of the data into train, test and validation set is part of the bash script developed for automating the preprocessing the data sets. A Python script is called by the bash script which again separated the data into three different sets for each language. The full Python split.py script can be found in the Git repository.

At first, I did not sample the data, but I quickly noticed during development that the Paracrawl corpus contains a lot of repeating sentences and sentence structures, so I decided to randomize the choice of lines in the Python script. For the randomization a fixed seed value was used to be able to reproduce the same data set several times.

4.1.2 Normalization

The normalization step simplifies the punctuation of the text by using re-writing rules (e.g., remove extra spaces, normalize different types of quotation marks, etc.). For this step, I used one of the preprocessing tools provided in the lecture, more specifically, the normalize-punctuation.perl perl script from moses.

4.1.3 Tokenization

The tokenization step splits the sentences in individual tokens, deciding the boundaries of each word (for example detaching a comma from a word). It separates words from punctuation with a white space, for example as in "hello!" $\rightarrow$ "hello !". For this step, I also used one of the moses preprocessing tools provided in the lecture, more specifically, the tokenizer.perl perl script.

4.1.4 Truecasing

The truecasing step defines the proper capitalization of words. To train a truecasing model, I used the moses train-truecaser.perl script and validated the test sets using truecase.perl as provided in the lecture.

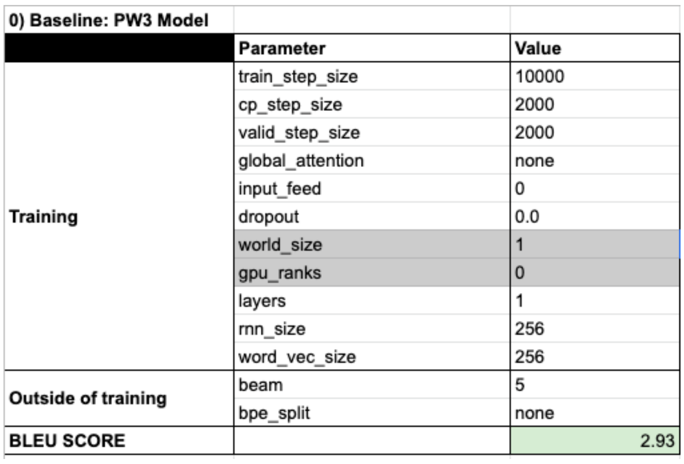
4.1.5 BPE Split

According to huggingface.co "BPE creates a base vocabulary consisting of all symbols that occur in the set of unique words and learns merge rules to form a new symbol from two symbols of the base vocabulary". For building and applying a BPE model, I used the the subword-nmt tool as suggested in PW4.

I started using BPE Splitting early on for some of the models in order to see if it will improve the BLEU score gradually. For the first model, I did not use BPE conversion, and introduced it for the second one, using 3,000 BPE merge operations which did not show a significant improvement. I changed the value to 32,000 which almost doubled the BLEU score (0.67 => 1.38) and then, later on increased it to 64,000 which again showed an improvement of the BLEU Score (4.68 => 5.95). However, in the course of the project, I noticed that the default vocabulary size (src_vocab_size and tgt_vocab_size) of onmt_translate was only 50000. I therefore decided to train the best 64,000 BPE model (titan) again with identical parameters but with 40,000 BPE merge operations, hoping that this would lead to an improvement in the BLEU score. The name of the adapted model is callisto. The assumption was confirmed and callisto achieved a BLEU score 1.98 points higher than titan.

4.2 Model Architecture Parameters

I first started with a simple model as a baseline using the suggested settings in the assignment PW3. I then gradually modified the existing parameter for the onmt_train command, to check whether they improve BLEU scores on the development set, such as -global_attention general, dotprod, mlp).



0) Baseline: PW3 Model		
	Parameter	Value
Training	train_step_size	10000
	cp_step_size	2000
	valid_step_size	2000
	global_attention	none
	input_feed	0
	dropout	0.0
	world_size	1
	gpu_ranks	0
	layers	1
	rnn_size	256
	word_vec_size	256
Outside of training	beam	5
	bpe_split	none
BLEU SCORE		2.93

All together, I trained 20 models for the Paracrawl corpus and 5 models for the News-commentary corpus. For an overview, please see the "MT-training-parameters.xlsx" excel sheet in the material folder of the Git repository.

4.2.1 ParaCrawl

For the Paracrawl corpus, BLEU Scores were calculated based on 1) translated test set splits from the original data set and 2) the translated newstest2019 development test set which resulted into two BLEU Scores for each model. Below table is summarizing the gradual change per model during training for 4,0000,000 random sentences of the Paracrawl corpus. I did not train with the complete corpus, as pre-processing took several hours. A subset of 4,000,000 sentences seemed sufficient to compare the models. For the final model sagittarius_a I used 98% of the corpus for training which corresponds to 33,683,879 sentences. The table shows which parameter has been modified with each new model and the changed value. Impactful changes are marked in green, worsening changes are red. The next model always carries over the previous configurations (e.g. fenrir uses the bpe_split from dione2, io uses general global_attention as adjusted in europa etc.). The table does not always reflect the order in which the models were trained.

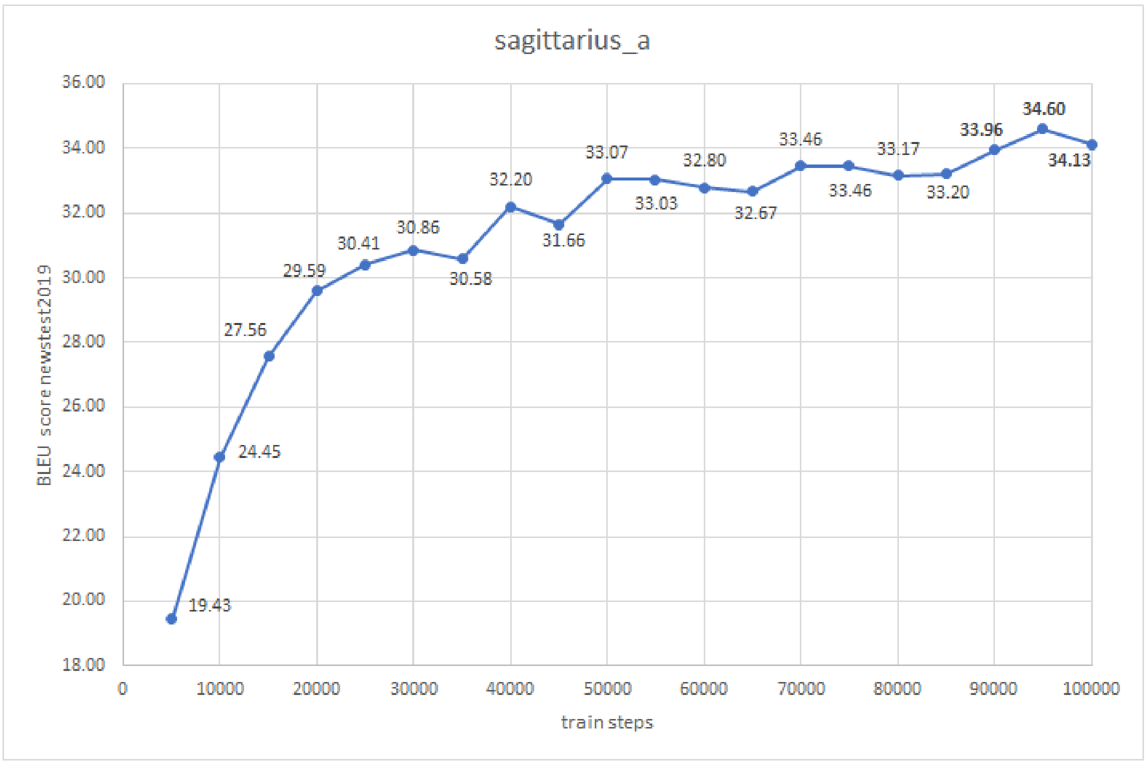
Model	Changed parameter	Changed value	BLEU test	BLEU newstest2019
0 - baseline	N/A	N/A	2.93	0.84
1 - ariel	bpe_split	3000	2.86	0.67
2 - bebhionn	bpe_split	32000	5.34	1.38
3 - charon	train/cp/valid_step_size	50000/10000/10000	4.84	3.82
4 - dione	train/cp/valid_step_size	100000/20000/20000	5.79	4.68
5 - dione2	bpe_split	64000	8.35	5.95
6 - fenrir	global_attention	dot	17.29	21.24
7 - ganymed	global_attention	mlp	17.59	21.38
8 - europa	global_attention	general	17.92	21.74
9 - io	layers	2	17.98	22.24
10 - kallisto	layers	4	17.15	19.42
11 - mond	layers	6	0.64	0.28
12 - oberon	rnn_size	512	18.70	22.74
13 - phoebe	rnn_size	1024	out-of-memory	out-of-memory
14 - rhea	word_vec_size	512	18.91	24.27
15 - thethys	transformer	true	35.78	28.64
16 - titan	layers/input_feed/dropout	6/1/0.1	37.71	31.36
17 - callisto	bpe_split	40000	38.35	33.34
18 - sagittarius_a	cp_step_size	5000	**39.53**	**34.13**

Changing only one parameter at a time may not give the best result, as some parameters are highly interdependent and only work well together. I therefore tried to modify several steps together (e.g. step_sizes) in order to find out if they interact.

The phoebe model config did not finish training because the GPU ran in an out-of-memory event during the first validation step. This could be reproduced several times, therefore a rnn_size of 1024 seems to be too high for the given hardware and would require additional resources or a lower batch_size.

Average model

During the training of the models I observed that the 80,000 step model was sometimes better than the 100,000 step model. Therefore, I decided to use a smaller cp_step_size of 5,000 for the final model (sagittarius_a) in order to average the last models using onmt_average_models and thus further increase the BLEU score. In the following graph it can be seen that the model from the train_step 95,000 achieves a BLEU score of 34.60, which is 0.47 higher than the model from the train_step 100,000.



Therefore, several average models of the last models were created as shown in the following table.

Model	averaged train step models	BLEU newstest2019
18.1 - sagittarius_a - average1	85,000; 90,000; 95,000; 100,000	34.93
18.2 - sagittarius_a - average2	90,000; 95,000; 100,000	35.12
18.3 - sagittarius_a - average3	95,000; 100,000	**35.13**

The average model of train_step 95,000 and 100,000 performs best with a BLEU score of 35.13. This score is 1.00 higher than the 100,000 train_step BLEU score.

Translation beam_size optimization

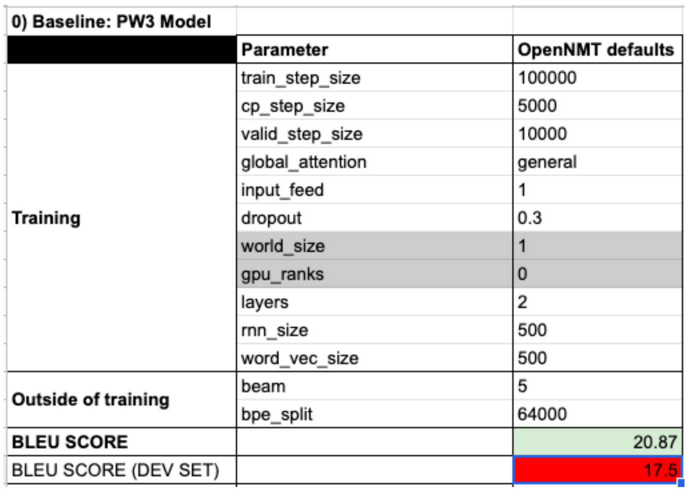
After finding the best model with "18.3 - sagittarius_a - average3", the last step was to investigate whether the BLEU score could be further improved by using a different beam_size during translation. For this purpose, the newstest2019 test set was translated with beam_sizes from 1 to 10. The results are shown in the following table.

translate beam_size	BLEU newstest2019
1	34.77
2	35.18
3	35.21
4	**35.27**
5	35.13
6	35.16
7	35.14
8	35.14
9	35.04
10	35.04

In this case, a beam_size of 4 gives the best BLEU score of 35.27. This is a BLEU score improvement over the default beam_size of 5, of 0.14.

4.2.2 news-commentary

I trained the news-commentary corpus parallel to Paracrawl on the Uni server before it stopped working. For this one, BLEU Scores were calculated based on the translated newstest development test set only, because translation for bigger test sets (10% of original data set) took very long in general. Below table is summarizing the gradual change per model during training for the News-Commentary corpus. Again, it shows the modified parameter and scores that have been reached for the model. For News-Commentary, I started off with a better baseline model, the default model suggested by OpenNMT, because it has already performed well on the ParaCrawl corpus. The baseline model scored a BLEU Score of 20.87 for the test set and 17.5 on the newstest development test set. Below, please find the detailed parameter information for the news-commentary baseline model.



0) Baseline: PW3 Model		
	Parameter	OpenNMT defaults
Training	train_step_size	100000
	cp_step_size	5000
	valid_step_size	10000
	global_attention	general
	input_feed	1
	dropout	0.3
	world_size	1
	gpu_ranks	0
	layers	2
	rnn_size	500
	word_vec_size	500
Outside of training	beam	5
	bpe_split	64000
BLEU SCORE		20.87
BLEU SCORE (DEV SET)		17.5

Model	Changed parameter	Changed value	BLEU Score (BLEU test set)
0 - baseline	N/A	N/A	17.5 (20.87)
1 - betelgeuse	input_feed/dropout/transformer	0/0.0/true	11.4
1 - alphacentauri	transformer	true	n/a
2 - pollux	layers	4	13.68
3 - proximacentauri	transformer	true	n/a
4 - polaris	layers	6	7.93
5 - rigel	transformer	true	n/a
6 - sirius	dropout/layers/rnn_/word_vec_size	0.1/3/1024/1024	14.42 (6.97)

Even though the baseline model looked very promising, changing parameters did not improve the score, but rather worsened it. Many layers without using a transformer decreased the score, but also this time using the transformer did not in improve it. Additionally, I tested changing the rnn_size and word_vec_size combined with lowering dropout and adding one layer which did not return a promising score either. From this, I concluded that the dataset probably contained to less data with 361445 lines per corpus (de/en) only, which is probably too less data to train an appropriate model. I therefore stopped training models on newscommentary, and marked BLEU scores as n/a in above table.

5 Observations

Early during the project it was noticed that it is very important to randomize the training part of the corpus, because it will otherwise be trained on very similar (almost duplicate) data. Head/tail commands should therefore be avoided.

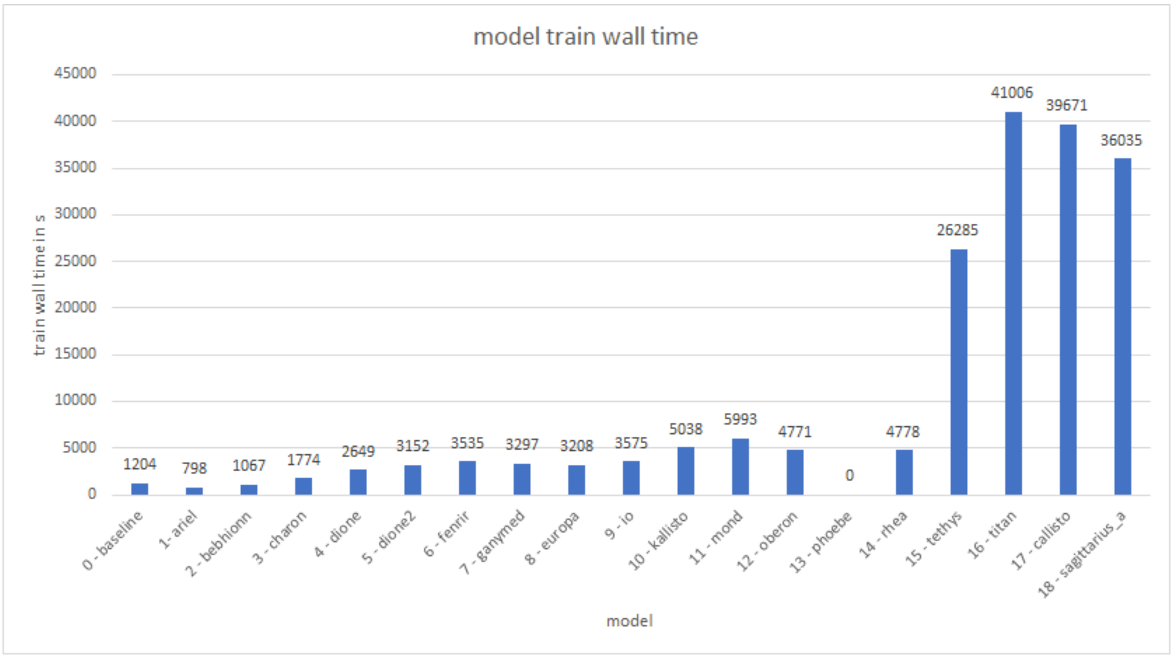
One of the major observation when training ParaCrawl was that the default settings of OpenNMT are much better than the chosen baseline model. If the default settings had been used as a baseline, better BLEU scores would have been obtained more quickly.

One thing that I spent a lot of time on in the early stages of the project, was the translation of the 10% test data of the original data set, with onmt_translate sometimes taking more time than the actual training. Generally, translating a test data set of 500,000 sentences took more than an hour (model: europa). A smaller subset for testing (e.g. 1-5% of the data) could be used instead where I would anticipate a similar BLEU score as for the 10%.

Later on, it was also found that only about 2 GB of GPU memory was used during the translation with only 2500 tokens per second being translated (–report_time parameter). By using a higher –batch_size of 200 (default is 30), the translation speed could be doubled to about 5000 tokens per second. Higher –batch_size values did not improve the translation performance further. The GPU memory usage went up to about 5 GB. Depending on the GPU memory size, the –batch_size parameter may need to be adjusted.

5.1 Train wall time

In the course of the project, a small script was developed which extracts the training time from the log files: train_wall_time.sh. With the help of this script, one gains an insight into the complexity of the individual models.



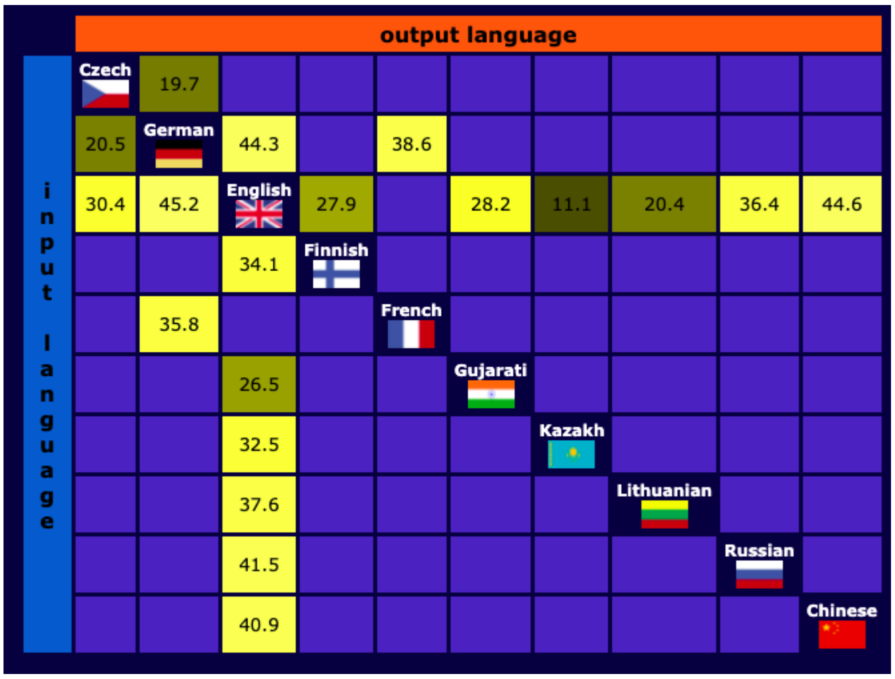
The following parameters have a strong influence on the training time:

- train_step_size: If the train_step_size is doubled, the training time is approximately doubled (compare: bebhionn vs. charon vs. dione)
- valid_step_size: If the validation set is very large, a low valid_step_size can greatly increase the training time
- layers: The more layers are used, the longer the training takes (compare: europe vs. io vs. callisto vs. moon)
- rnn_size: Doubling the rnn_size resulted in 33% longer training time in my tests (compare: io vs. oberon)
- transformer: The use of Transformer was by far the most computationally intensive change. The training time was more than quintupled (compare: rhea vs. tethys)

The remaining parameters only marginally influenced the training time.

6 Benchmarks

According to http://matrix.statmt.org/matrix the best BLEU score for the newstest2019 test set for English to German is 45.2. Compared to these results the score of the final model of this project was less (35.27) and not reached by the given data sets and configurations for training. Based on the result, the trained model would still receive an entry in the Scored Systems leaderboard: http://matrix.statmt.org/matrix/systems_list/1909. Of course, it has to be considered that research groups have more experiences using and tuning appropriate parameters as well as being able to train more data on high-performing clusters with multiple GPUs.

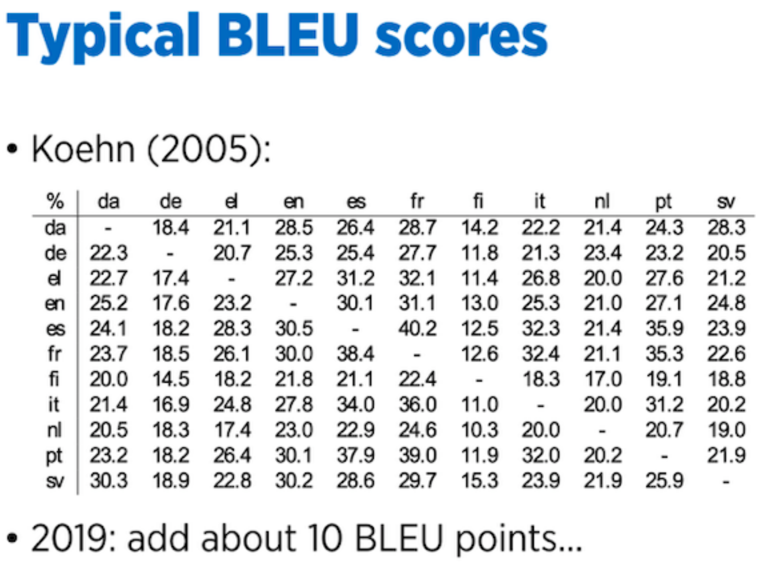


output language

input language

Czech	19.7								
20.5	German	44.3		38.6					
30.4	45.2	English	27.9		28.2	11.1	20.4	36.4	44.6
		34.1	Finnish						
	35.8			French					
		26.5			Gujarati				
		32.5				Kazakh			
		37.6					Lithuanian		
		41.5						Russian	
		40.9							Chinese

However, when looking at typical BLEU scores from 2005, the model has exceeded the score of 17.6 about 100%. This is also true when looking at typical BLEU scores of 2019, assuming that it would be +10 BLEU points reaching a score of 27.6. With 35.27 the trained model is 7.6 BLEU points above the typical BLEU score from 2019.



Typical BLEU scores

- Koehn (2005):

%	da	de	el	en	es	fr	fi	it	nl	pt	sv
da	-	18.4	21.1	28.5	26.4	28.7	14.2	22.2	21.4	24.3	28.3
de	22.3	-	20.7	25.3	25.4	27.7	11.8	21.3	23.4	23.2	20.5
el	22.7	17.4	-	27.2	31.2	32.1	11.4	26.8	20.0	27.6	21.2
en	25.2	17.6	23.2	-	30.1	31.1	13.0	25.3	21.0	27.1	24.8
es	24.1	18.2	28.3	30.5	-	40.2	12.5	32.3	21.4	35.9	23.9
fr	23.7	18.5	26.1	30.0	38.4	-	12.6	32.4	21.1	35.3	22.6
fi	20.0	14.5	18.2	21.8	21.1	22.4	-	18.3	17.0	19.1	18.8
it	21.4	16.9	24.8	27.8	34.0	36.0	11.0	-	20.0	31.2	20.2
nl	20.5	18.3	17.4	23.0	22.9	24.6	10.3	20.0	-	20.7	19.0
pt	23.2	18.2	26.4	30.1	37.9	39.0	11.9	32.0	20.2	-	21.9
sv	30.3	18.9	22.8	30.2	28.6	29.7	15.3	23.9	21.9	25.9	-

- 2019: add about 10 BLEU points...

6.1 Future outlook

I came up with some ideas on how the BLEU score could be improved even further. Unfortunately, this was not possible due to lack of time and hardware resources. Research groups often use 8 or more GPUs for training and can therefore train much larger models faster. Other parameters that should be investigated are for example:

- Training with more than 100,000 steps (–train_step_size).

- The parameters for the transformer can also be adjusted and investigated individually
- increase −batch_size and −accum_count. Unfortunately, an increase of batch_size was not possible due to GPU memory restrictions and a doubling of accum_count led to double training time.
- It might be worth considering to combine several data sets with each other, instead of evaluating each data set independently. Also, the quality of the data not the best for some of the corpora, e.g. ParaCrawl which contained a lot of noisy data (English vocabulary, number sequences that did not belong to a sentence, unmatching punctuation etc.). Improving the the preprocessing using other tools than the ones learned in our lectures, e.g. I read about OpenNMT v.2 being able to do preprocessing on its own, might be an option for a follow-up project.
- In preprocessing, the parameter −share_vocab should be tested. In addition, −share_embeddings and −share_decoder_embeddings should then be examined in training. Some papers suggest that these options can lead to better scores.
- All models were trained with single precision (fp32). However, newer NVIDIA GPUs have tensor cores that allow faster training with half precision (fp16). A Mixed precision training can be activated in OpenNMT with the option −model_dtype fp16. Initial tests showed that this option accelerates the training with the RTX 3080 by 30%, but this was not investigated further, as inexplicable out-of-memory situations occurred from time to time when using this option.

7 Resources

- Klein, Guillaume et. al (2017): OpenNMT: Open-Source Toolkit for Neural Machine Translation
- Vaswani, Ashish et. al (2017): Attention Is All You Need
- Klein, Guillaume et. al (2020): The OpenNMT Neural Machine Translation - Toolkit: 2020 Edition
- Klein, Guillaume et. al (2020): Efficient and High-Quality Neural Machine Translation with OpenNMT
- OpenNMT-py: FAQ - How do I use the Transformer model? Link: https://opennmt.net/OpenNMT-py/legacy/FAQ.htmlhow-do-i-use-the-transformer-model (05.08.2021)