FNet as Efficient Attention Substitute

Pablo Ortega Sanchez
Computational Linguistics Student / Heidelberg University
ort.san.pablo@gmail.com

Abstract

This document explains the thought process, implementation and experiments of my project about FNet as efficient, drop-in transformer architecture (Lee-Thorp et al., 2021). In comparison to the attention mechanism proposed in Vaswani et al. (2017), FNet employs fourier transforms to mix tokens in an efficient way to obtain meaning representations. FNet is not reliant on as much compute as standard attention but, unfortunately, so far has only been implemented in the encoder layer of an transformer. However, easy implementation and outlook towards a more efficient transformer, that runs on "worse" hardware is promising, which was my motivation for this project.

1 Implementation

As base implementation I cloned the in-official implementation of Vaswani et al. (2017) by Ko (2019) from github. I then duplicated the contents to have a clear division between the two methods. Afterwards, I took the in-official implementation of Lee-Thorp et al. (2021) by Rishikesh20 (2021) and modified one of the duplicate directories, so that the encoder layer of the transformer uses the FNetBlock instead of standard attention.

1.1 Environment

I built my environment using conda. It is recommended to install the environment via the environment.yml file in the repository:

```
conda env create -f environment.yml
```

It will eventually fail at catching the spacy languages so the following command steps are necessary:

```
1. conda activate opitz-transformer
2. python -m spacy download
de_core_news_sm
3. python -m spacy download
en_core_web_sm
4. pip install -U matplotlib
```

2 Experiments

Several experiments were conducted on two different GPUs. The first GPU is a RTX 3060 TI with 8 GB VRAM available, so a powerful consumer GPU. The second GPU is a GTX 970 with 3.5 + 0.5 GB VRAM available. In total, 4 GB of VRAM are available, however, it is split up and in my experience causes a lot of headache. It was one of the more popular GPUs a few years ago. It's important to note that the GPUs are installed in two different machines and the specifications are different, however the idea is, that in theory, there are configurations of the attention transformer architecture that run out of video-memory but the same configuration works with the FNet transformer architecture. Unfortunately, I did not manage to find such a configuration for the RTX 3060 TI, but I did for the GTX 970. The models are trained on the Multi30k dataset. In order to have comparable results with respect to efficiency, I used tracemalloc and cuda_max commands in the respective train.py files to track memory and VRAM utilization. Unfortunately, those numbers are not perfect, e.g. cuda_max does not distinguish between functions, or even programs, it just tracks the general GPU utilization. Note that only the last evaluated BLEU score is reported, which usually represents the models' performance quite well. The full results and corresponding graphs can be found in the respective models' directory in the 'results'-directories.

2.1 Experiments: RTX 3060 TI

For the experiments on the RTX 3060 TI the following parameters were set in the configuration file:

```
batch_size = 128
max_len = 512
d_model = 512
n_layers = 6
n_heads = 8
ffn_hidden = 2048
drop_prob = 0.1
```

The results can be seen in Table 1 in Section A.1 of the Appendix. The suffixes '100' and '1000' indicate the amount of epochs. The FNet model is consistently faster, but in the case of convergence, achieves a lower BLEU score. Also, both models, when trained for 1000 epochs seem to overfit the data, as the training loss is noticeably lower than the test loss. If trained for only 100 epochs, FNet achieves a significantly higher BLEU score. This result is unexpected, but is probably due to inconsistencies in training, and I suspect that repeating the experiments several times would result in the attention model achieving slightly higher scores.

2.2 Experiments: GTX 970

For the second GPU, the low resource scenario, three experiment runs were conducted. The results can be seen in Table 2 in Section A.2 of the Appendix. Unfortunately the VRAM utilization seems to slightly increase over time, which often lead to 'out of memory' errors after a few hundred epochs. This resulted in a lot of time wasted on training models, thus I decided to limit the training in the low resource scenario to 100 Epochs.

lowres1: The first experiment in the low resource setting had the following parameters:

```
batch_size = 64
max_len = 64
d_model = 512
n_layers = 6
n_heads = 8
ffn_hidden = 2048
drop_prob = 0.1
```

With exception of batch_size and max_len, the configuration is the same as for the experiments in Section 2.1. Unsurprinsingly, both models ran out of memory. However, the attention model did not start training while the FNet model managed to go through 20 epochs and achieve a BLUE score of eight before running out of memory.

lowres2: The second experiment severely reduced the amount of trainable parameters; the configuration was as follows:

```
batch_size = 128
max_len = 512
d_model = 256
n_layers = 4
n_heads = 8
ffn_hidden = 1024
drop_prob = 0.1
```

The amount of layers is lowered from six to four, the feed-forward hidden layer size is halved down to 1024 and the embedding dimenstion 'd_model' is reduced to 256 compared to the previous experiment. Nevertheless, the batch_size and max_len are increased to 128 and 512 respectively, which is the same as the experiments in Section 2.1 used. In this scenario, FNet trains not only faster and utilizes less VRAM, the BLEU score is better, too. The model can compete with the much bigger version FNet100 from the experiments in Section 2.1.

lowres3: The last round of experiments used the following parameters:

```
batch_size = 128
max_len = 512
d_model = 256
n_layers = 4
n_heads = 8
ffn_hidden = 2048
drop_prob = 0.1
```

In comparison to the lowres2 experiment, the feed-forward hidden layer size has been increased again to its original value, 2048. First, the attention model quickly ran out of memory. The FNet model however trained for the full 100 epochs and achieved a surprisingly high BLEU score of 20. This is significantly better than the FNet100 model from Section 2.1, and only slightly worse than the FNet1000 model. Additionally, the train time was a order of magnitude faster, and the VRAM utilization was almost 50% lower. This was a very unsuspected result and definitely represents the best time to BLEU ratio that I have ever experienced myself.

3 Conclusion

In summary I was surprised how quick I managed to get the FNet method running. Also, in general FNet models seem to learn quicker, as in they need fewer epochs to achieve the same BLEU score as compared to a attention model. In the end I believe that the methods of measuring memory utilization are one of the biggest challenges in this kind of

work, as it is very hard to determine what is actually measured. Nevertheless, I managed to train a 20-BLEU score model in 100 minutes on my old machine, which is incredibly efficient and I would have never believed to be possible.

References

Kevin Ko. 2019. Implementation of "attention is all you need" using pytorch. https://github.com/hyunwoongko/transformer.

James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontañón. 2021. Fnet: Mixing tokens with fourier transforms. *CoRR*, abs/2105.03824.

Rishikesh20. 2021. Unofficial implementation of google's fnet: Mixing tokens with fourier transforms. https://github.com/rishikksh20/FNet-pytorch.

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.

A Appendix

A.1 Experiments: RTX 3060 TI

RTX 3060 TI 8 GB VRAM	parameters	training time	tracemalloc peak	cuda_max peak	last BLEU
ATT1000	55.21 M	811m 59s	17.7140 MB	5504.9693 MB	**23.7854**
FNet1000	48.90 M	**727m 4s**	**17.6138 MB**	**5057.7490 MB**	20.2948
ATT100	55.21 M	83m 1s	17.6351 MB	5422.9780 MB	13.2904
FNet100	48.90 M	**76m 0s**	**17.5283 MB**	**5009.3614 MB**	**17.2470**

Table 1: Experiments run on RTX 3060 TI with 8 GB VRAM.

A.2 Experiments: GTX 970

GTX 970 3.5 + 0.5 GB VRAM	parameters	training time	tracemalloc peak	cuda_max peak	last BLEU
lowres2ATT100	12.91 M	90m 28s	9.3867 MB	2319.6534 MB	13.8115*
lowres2FNet100	11.86 M	**82m 10s**	**9.2905 MB**	**2156.0591 MB**	**17.1187**
lowres3ATT100	-	-	-	OOM	-
lowres3FNet100	16.06 M	**99m 48s**	**9.2900 MB**	**2648.0097 MB**	**20.0250**

Table 2: Experiments run on GTX 970 with 3.5 + 0.5 GB VRAM. *The BLEU actually went up to about 15 throughout the experiment but dropped again.