# Master thesis report 4

University of Tartu

March 6, 2023

## 1 Baseline implementation progress

I successfully was able fine-tune T5-Large on factual data(Figure 1), which took 33 hours. However, I didn't attain same EM accuracy as in [Neeman et al., 2022]. I tried to train longer but the accuracy stucks at around 0.6, used different gpu(with more memory) where I did ablation of gradient accumulation. Turns out that in the paper for evaluation they were using NQ dev set or test set according to the paper, however I was evaluating over validation set. I also take a look into comparison between validation set and their predictions. What I found was that in total I had 8K mispredictions, where for 4K the difference were significant, and for other 4k either validation entry was subset of predicted entry or vise versa(like '18 January 1981' vs 'in 18 January 1981'). I calculated that five extra words stands in a way for us to get desirable EM accuracy.

Interesting findings:

- **FP16 optimization:** gave two-fold training time decrease i.e. one epoch with FP16 on took 1:30, whereas without FP16 one epoch took 3:00.
- **FP16 pitfalls**: using 'float16' gives nan losses. To fix this I've switched to 'bfloat16'. Also nan I was getting when tried train on multiple GPU's(T5 NaN when training on multiple GPU's).
- **Gradient accumulation:** also gave training time reduction from 1:20 to 1:04 per epoch. I've used batch 64, and accumulated gradients for two steps.
- **Better optimizer:** In T5 paper the authors instead of AdamW optimizer they were using AdaFactor ([t5_ft_tips, 2022]), which consume less memory(from 21GB to 15GB in my case), and was used for pretraining.

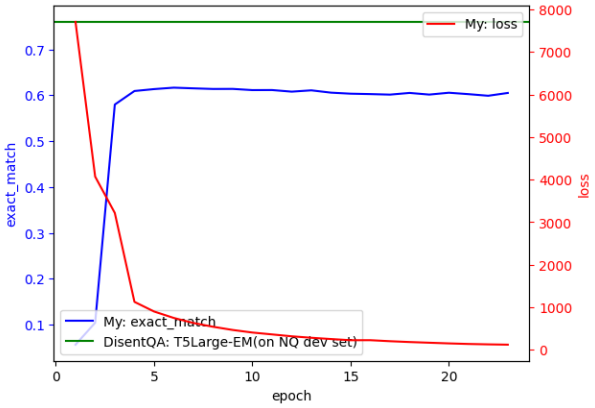


Figure 1: T5-Large finetuning on factual entries. And evaluation on validation set. However, in [Neeman et al., 2022] validation set(only factual) was used to select best checkpoints(it is not clear whether other validations sets were utilized). For evaluation [Neeman et al., 2022] used NQ development set or test set(which I don't have at the moment).

- **Settings:** my training and data wise setup is similar to T5 on TPU tutorial.

# 2 Model level generalization

I've studied implementation details of following paper:

1. **Adapter**([Houlsby et al., 2019]):
   (a) *Approach*: Inject two additional MLP(bottleneck) layers into transformers encoder layer. They were experimenting with BERT, and I need to do same injection for T5(which is luckily also encoder-decoder model). However, they were mentioning several shortcomings in terms of initialization, and that it is hard to train.
   (b) *Source code:* code
   (c) *Complications:* Initialization should be precise, they figured that initial weights initialization should be near-identity. Otherwise model fine-tuning fails.
   (d) *Limitations:* The only limitation is that I need to modify T5 code.
2. **Prompt-tuning**([Lester et al., 2021]):

(a) *Approach:* This paper require the least amount T5 code modifications. They are just concatenating prompt embedding with input embedding, where a prompt embedding obtained from small MLP. Interestingly problems only depends on input length.

(b) *Source code:* Original code is not available, however someone made an implementation.

(c) *Complications:* Details of architecture in the paper are not given. Also they were mentioning that they needed to do 'Unlearning Span Corruption', which I didn't quite understand why, but they have a lot of reasons, and better results. Which means, that I also need to do that(they released checkpoints) since in our cases models are same(T5).

(d) *Limitations:* Don't see anything

3. **Prefix-tuning**([Li and Liang, 2021]):

(a) *Approach:* With the same intuition as prompt-tuning, prefix-tuning also adds additional embeddings accross all the transformer layers(encoder and decoder including). This can be done in huggingface library quite easily by passing argument past_key_values, however since this argument can only be used by decoder, in order to make it available for encoder as well required additional modifications.

(b) *Source code:* The code is quite dirty, messy and badly written because it incorporate so many additional settings(like tuning hyperparameters, a lot of variations of prefixes etc.). But it is still possible to figure it out. On top of that, authors were using quite old version of transformers (3.5.0).

(c) *Complications:* code wise

(d) *Limitations:* A lot of T5 modifications, which might be a problematic for debugging.

(e) *Extra:*

presentation of using prefix tuning in their paper

their paper

their code(It either a mistake or my misunderstanding, since they use prompt tuning whereas in the presentation they were talkin about prefix-tuning).

# 3 Questions:

1. All of those model level modifications only give comparable performance, basically it is not better than fine tuning. And I'm suspecting that in our case it won't give better results, but only faster tuning(since only external parameters are trained).

2. Until I get [Neeman et al., 2022] data I don't know for sure whether my fine tuning attained similar results.

# 4 This week plan:

1. Implement all the lightweight tunings.

# References

[Houlsby et al., 2019] Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for nlp.

[Lester et al., 2021] Lester, B., Al-Rfou, R., and Constant, N. (2021). The power of scale for parameter-efficient prompt tuning.

[Li and Liang, 2021] Li, X. L. and Liang, P. (2021). Prefix-tuning: Optimizing continuous prompts for generation.

[Neeman et al., 2022] Neeman, E., Aharoni, R., Honovich, O., Choshen, L., Szpektor, I., and Abend, O. (2022). Disentqa: Disentangling parametric and contextual knowledge with counterfactual question answering.

[t5_ft_tips, 2022] t5_ft_tips (2022). T5 finetuning tips.