Master thesis report 3

University of Tartu

February 27, 2023

1 Baseline implementation

I started implementation of the baseline model. The things that I've discovered are following the T5-XXL is around 45GB, and the dataset(natural questions) 90GB. Working in such settings required a lot of engineering to be able to do train such huge model on huge dataset. I started with testing performance of training with various number of GPU's.

Number of GPU's	Duration	Type of GPU	F1
1	21:38	A100-80GB	0.85
2	14:42	A100-80GB	0.84
3	12:46	A100-80GB	0.84
4	10:26	A100-80GB	0.82

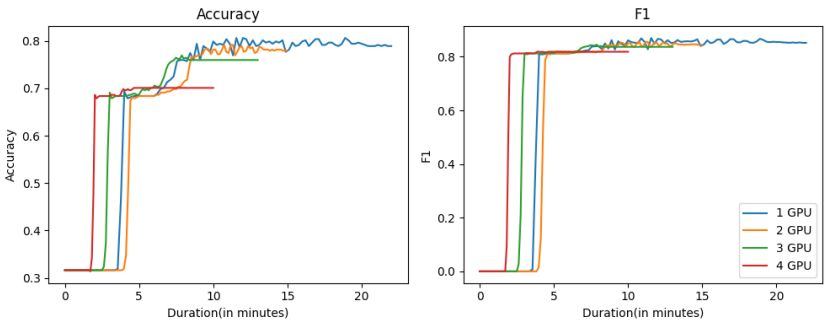

Figure 1: Accuracy and F1 over time with multiple GPU's

1.1 Technical details

- **Dataset:** In [Neeman et al., 2022] they are using simplified version of natural questions dataset([google AI, 2020]). Which is only 4GB in

contrast with full dataset of 45GB. On top of that, to train vanilla T5 they extracted only questions that has an answer(i.e. at least one out of five annotators found one of passages suitable to answer the question) which accounts for only 35%. However, they report that after filtering they've obtained 85540, however in my case it was 106926. But I followed their filtering procedure precisely i.e. they've utilized 'question_text', and 'annotations' columns, where in the latter for the context, and answer they've used 'long_answer' and 'short_answer' correspondingly. They were explicit about those details in the paper, so there are no confusion.

The number that I've obtained is actually correct. In the table below we can see the size of train and validation splits. And if we sum them $85540 + 21386$ we would get 106926, which is the exact value I've got. However, when I tried using the same filtering technique for the test set, I've got 4180, whereas in the table the obtained only 1365. Turns out that in [Neeman et al., 2022] they've used corpus substitution policy from [Longpre et al., 2021]. This type of substitution is about replacing entity a with another entity a' from the same dataset(*in-domain*). This is how they've got only 1365 instances(because not all factual examples would have entities).

	Factual	Counterfactual	Empty	Random
Train	85,540	30,653	85,540	85,540
Validation	21,386	7,698	21,386	21,386
Test	1,365	1,365	1,365	1,365

- **Model**: Training huge model such as T5-XXL is quite challenging task. The main challenge comes from the fact that the size of the model is 45GB. I have at my disposal A100-80GB, but the problem is that it is impossible to train such a model on the most powerful GPU. Because I would be able to accommodate the model, but if we need to finetune it, we also need to take into account size of activations(forward pass), gradients(which have the same size as model i.e. 45GB, because it is update for all the weights, backward pass), and also optimizer(AdamW) uses memory for momentums. We need to have size(model)+size(activations)+size(gradients)+size(optimizer)> 80GB. This situation isn't doomed, because other labs and companies still able to train LLM. The are various optimizations that I can utilize to tackle training of the model. At first I started to look into pytorch level optimization([huggingface, 2023]).

1. **Gradient checkpointing:** It is perfectly described by [Bulatov, 2018]. But the basic idea to discards some of activations, and during optimization step re-compute them. This would allow to save memory usage, whereas on the other hand it would increase training time.
2. **Gradient accumulation:** This technique emulates bigger batch sizes. The idea is not to perform backpropagation for every forward pass, but rather accumulated gradients, and perform optimization in bulk. Example how to do it here[Maheshkar, 2023]. It should be faster because optimizer is less run, the faster training should be.
3. **FP16 training:** The idea here is to use smaller precision for activations. Instead of 32-bit we would switch to 16-bit. Whereas weights and gradients would still have 32-bit precision. This optimization is also called mixed precision training. It help to reduce memory usage, and faster training.

Further I have tried model parallelism which actually gave me hope that T5-XXL would be possible to finetune. And lucky for me huggingface implemented model parallelism for T5 models. Which easily can be done with single command. Now with this approach I was able to load the model into GPU memory and start training. But after several epochs I started to receive out-of-memory (OOM) error. So I started to investigate why this is happening, because the consumption of the memory should be the same between various epochs. After 8 hours of researching I found out that pytorch caching memory, in order to reduce number of allocation in future, however, it seemed that pytorch doing it stupidly i.e. until my scripts get OOM. To counter that after each epoch I explicitly started erasing pytorch cache, this became remedy to memory leak. Afterwards, I started to obtain issues with pytorch DataLoader and huggingface Dataset, they became incompatible after I've updated pytorch to latest release 1.13.1.

Now I'm successfully were able to launch training with two A100-80GB GPU's. However, soon my enthusiasm went down, because one epoch was taking 90 minutes(for T5-Large). And I thought that I need to do 50k epochs which would take approximately 3000 days, however, lucky for me I was mistakes in the [Neeman et al., 2022] they said that they trained to 50k training steps. To my surprise epochs and training steps are different thing. Epochs refer to going through the whole dataset, whereas training steps refer to gradient updates. Thus, with batch size 32 and train set 81540, I would get $81540/32 = 2548$ updates per epoch,

hence 50000/2548 = 19 epoch in total. This was incredible relief for me, because I thought that my research is doomed. Anyway 19 epochs it is one day training for T5-Large.

Now I'm experimenting with data parallelism(only for T5-Large), where the idea is to upload full model instance into several gpus, and run their training over different portion of training data. Then during the backward pass the normalized weights updates would be shared between all instances. I was able to started training but soon after canceling my job, the nodes with A100-80GB crashes, and up until now they are still down. Such an approach would allow to reduce model training, however it is mostly relevant to the T5-XXL, and I would need to combine model and data parallelism. The only way to do that is to have two machines where each machine has two A100-80GB. Only then I would be able to accommodate a model per machine splitting it into two GPU's, and then data parallelism might work.

- **Setup:** As input to the model I'm giving question and context. To separate them I've introduce two additional tokens '⟨ **question** ⟩' and '⟨ **context** ⟩'. So, the input to the model has following format:

  "⟨ **question** ⟩: When the WWII started? ⟨ **context** ⟩: The WWII ... ⟨ /**s** ⟩".

  The ground-truth contain only short answer plus EOS symbol ⟨ /**s** ⟩".

2 Questions:

1. Training T5-XXL might be not feasible, so my question is how it is important to for the paper to replicates results with T5-XXL?

2. Can we somehow utilize smaller models to prove the point? Because it is exactly what happened between [Neeman et al., 2022] and [Li et al., 2022]. The former were using T5-Large(770M) and T5-XXL(11B), whereas the latter was using T5-XXL(11B) and PaLM(540B).

3. What is the most important part in the thesis? Which should include most of details.

4. What would be the best thesis in your view?

References

[Bulatov, 2018] Bulatov, Y. (2018). Fitting larget networks into memory.

[google AI, 2020] google AI (2020). Natural question dataset.

[huggingface, 2023] huggingface (2023). Performance and scalability: How to fit a bigger model and train it faster.

[Li et al., 2022] Li, D., Rawat, A. S., Zaheer, M., Wang, X., Lukasik, M., Veit, A., Yu, F., and Kumar, S. (2022). Large language models with controllable working memory.

[Longpre et al., 2021] Longpre, S., Perisetla, K., Chen, A., Ramesh, N., DuBois, C., and Singh, S. (2021). Entity-based knowledge conflicts in question answering.

[Maheshkar, 2023] Maheshkar, S. (2023). How to implement gradient accumulation in pytorch.

[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.