Guideline for GPT-2: Investigating Model Variants For Different Computational Budgets

Stanford CS224N Custom Project

Ryan Kang
Department of Computer Science
Stanford University
txryank@stanford.edu

Abstract

The GPT model has been demonstrated to generalize well to a variety of downstream tasks when trained on a very large corpus. GPT-2 demonstrated tasks in the zero-shot and one-shot learning whereas GPT-3 increases the number of parameters and shows good downstream performance for more complex finetuning tasks. As such, generally increasing the model size tends to lead to better performance but what if you are constrained by a computational budget? The focus of this project is to provide a comprehensive guideline for which GPT variants work best on what tasks given certain computational constraints (e.g. want to run on a smaller device). The primary method to keep larger models under the computational limit will be to apply factorization and dimension-reduction techniques to the MLP layers immediately after the attention block. We will call these techniques "accelerators" and eventually make claims such as "*use GPT variant A with accelerator X instead of a smaller GPT variant B.*"

1 Key Information to include

Collaborating with Azalia Mirhoseini, former Staff Research Scientist and manager at Google Brain and currently a researcher at Anthropic. Also in collaboration with Mohit Tiwari, PhD candidate in Computer Science at Stanford.

2 Introduction

ChatGPT has taken the world by storm but the NLP craze started before this. In 2018 OpenAI announced the first Generative Pre-Training (GPT) model which was a transformer-based language model (i.e. the objective is to predict the next word in a passage) that was pre-trained on a large text corpus. Before this, most state-of-the-art NLP models primarily used supervised learning which necessitated large amounts of manually labeled data. Naturally, this process was expensive which limited the scalability of such models. The GPT obviated a large portion of this by employing unsupervised pre-training on a large corpus. In the following years (2019 and 2020), GPT-2 and GPT-3 were released which showed that the GPT models perform well in zero-shot, few-shot, and finetuning settings. In essence, these models shared the same architecture but the complexity of parameters was increased by a factor for each subsequent model. Specifically, the number of parameters went from 117 million, to 1.5 billion, to 175 billion. This is clearly an intractable complexity for most local machines and would still take hundreds of hours to train on industrial computing machines. Even if we're only considering inference time, the massive parameter complexity is extremely costly. Therefore, part of the narrative has shifted towards accelerating the GPT models within a given budget. This paper explores this narrative within the GPT-2 model.

3 Related Work

Common methods to reduce the complexity of large language models can be categorized into two approaches: optimizing the attention scheme and optimizing the linear layers. to motivate the project more clearly, this section will introduce two papers – Big Bird and Monarch – that highlights the contributions and limitations of the two approaches.

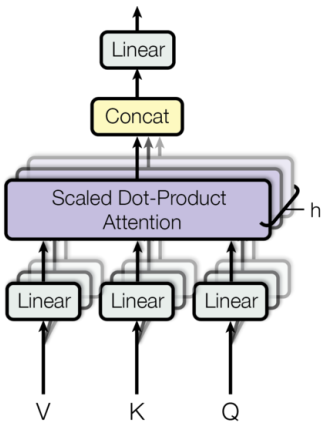
For the first approach, Big Bird picks at the fact that the fully-connected attention scheme of transformers adds a constraint to the size of the input sequence length since the model scales quadratically (i.e. $O(n^2)$) with input size n. There have been previous models such as the LongFormer that have designed a sparse-attention scheme to reduce a single attention layer's complexity to $O(n)$ but empirical performance is worse than the fully-connected transformers. Big Bird addresses these issues by building on LongFormer and adding an additional r random tokens for better downstream-task performances. More specifically, they model the attention scheme as a graph-connection problem, showing that they're able to satisfy all the known theoretical properties of full transformer. They also empirically show that the extended context modelled by Bigbird benefits variety of NLP tasks, achieving SOTA results for question answering and document summarization on a number of different datasets such as HotpotQA, NaturalQ, TriviaQA, and WikiHop. Finally, they introduce a novel application of attention based models where long contexts are beneficial: extracting contextual representations of genomics sequences like DNA.

Unfortunately, Big Bird has it's limitations, namely their claim on the "linearity" of attention scheme doesn't account for the extra attention layers that the algorithm will require to get their claimed expressivity. On top of that, BigBird doesn't give theoretical bounds on the number of attention layers needed (in the extreme, we'll need all n layers which is quadratic in input size). A second limitation is that BigBird actually requires the same memory as the original BERT implementation. To be more precise, there are a total of 512 tokens which is identical to BERT. However, the sequence length is longer (x8 according to empirical results) which does NOT mean that we can run transformers on small memory machines. Instead, we can run on the same memory machines but on longer documents and have some shared global information. But, even if we're willing to be lenient to these limitations, perhaps the biggest issue is that BigBird doesn't scale well with the model size. An intuition for this is that, as the model grows larger, it now has more capacity to use the "less important" attention connections (e.g. the attention connecting the first and last word) which is why large models benefit from the full-connected attention networks. Then, is focusing on the attention scheme the correct approach at all?

This is a natural segue to the second approach – optimizing the linear layers of the attention block. Namely, Monarch is a method of factorizing a dense-structured matrix into a product of two block-diagonal matrices where each matrix has m blocks of size m x m (i.e. $M = PLP^TR$ where P is a permutation matrix). Such a decomposition into two 3D-tensor block-diagonal matrices allows users to leverage optimized batch-matrix-multiplication (bmm) routines on GPUs. More precisely, the permutation P is fixed which means Monarch is able to avoid the problem of irregular sparse patterns, allowing this factorization method to exploit highly optimized parallelism and vectorization. This is what differentiates Monarch with previous approaches of learning these factorizations since, in that case, we were actively learning the permutation and can't rely on a fixed structure. One could imagine factorizing the linear layers of the attention block using such factorization techniques.

4 Approach

As hinted in the section above, the approach taken in this project is the second one – optimizing the fully-connected linear layer of each attention block using factorization techniques. The `linear layer` that's getting factorized is the one at the very top of the figure (not the linear layers for the `V`, `K`, and `Q`) where the figure represents a single attention block. These factorization techniques are dubbed `accelerators` and this project uses the SVD decomposition, butterfly matrices [1], and monarch factorizations [2].



The performance of accelerators are given by the **compute multiplier** (ratio of `scaling/loss`) for different GPT variants under a restricted computational setting. Comparisons will be made such that variants with larger multipliers have a sharper negative slope, similar to what's shown in this paper on scaling laws [3]. All model variants are based on the GPT-2 models [4] provided by huggingface's open-source repo. Namely, this project compares the GPT-small (117M), GPT-Medium (345M), GPT-large (762M), GPT-2 (1542M) variants with the number of parameters specified above. These models are nice for comparison since they're basically the same architecture, just with different parameter scaling. Consequently, the baselines will be the compute multipliers of these models without any accelerators applied.

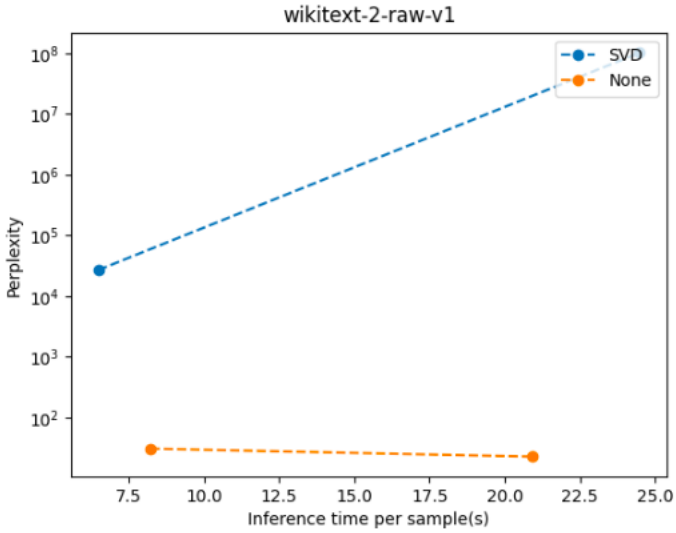
5 Experiments

- **Data**: There is no additional data that is needed for training since the purpose of these experiments is to understand the performance of GPT variants during *inference time*.

- **Evaluation method**: Since we're working with GPT-2 variants, we used the same evaluation methods as specified in the original paper. These methods can be categorized into two settings: zero-shot and one-shot. For zero-shot, we use a mixture of perplexity and accuracy when predicting the next sequence on datasets such as LAMBADA, Children's Book Test (CBT), or One Billion Word Benchmark. For one-shot, we evaluate F1, ROGUE, BLEU, and accuracy scores on the Conversation Question Answering dataset, CNN and Daily Mail datset, WNT014 English-French test set, and Natural Questions dataset respectively.

- **Experimental details**: Since our baselines are GPT-2 variants, we use the default parameters provided in huggingface when running all inference experiments. Additionally, in order to identify which layer would benefit the most from applying the accelerators, we profiled the GPT models to find the computational bottleneck. Finally, we applied these accelerators and found the compute multipliers for each transformed GPT variant.

- **Results**: Contrary to common belief, our profiling results found that the bottleneck of the GPT model wasn't in the quadratic attention scheme but the MLP layers immediately following the attention blocks (40% : 60% computation split). This supported our decision to apply the accelerators to the linear layer immediately following the attention scheme.



Applying the SVD to the linear layers for zero-shot learning resulted in a perplexity that was far worse than the baseline. Driven by these results, Monarch and Butterfly were applied with finetuning resulting in comparable scores on the GLUE tasks with 1.5x speedup. Unfortunately, none of GPT variants performed above the original baseline compute multiplier for the the zero-shot and few-shot case.

6 Analysis

Intuitively, factorizing the layers with no finetuning fails because the factorized weights are no longer the optimum for the original language model objective. Another supporting evidence for this is that the factorized linear layers did indeed perform on-par with the baselines on finetuning tasks where we are allowed to train the reduced parameters. Similarly, if this hypothesis is accurate, pretraining the GPT variants from scratch would lead to final weights that generalize equally well to the baseline GPT models but are able to train faster. Such experiments have not been conducted but would be an interesting extension.

Another observation is that there seems to be a shift in the bottleneck computation as the GPT model scales. More precisely, with the smaller variants, the compute multiplier is less forgiving of the attention scheme with profiling results closer to 50% computation split (sometimes going over). This result is pretty obvious because smaller GPT variants have smaller linear layers which means the attention scheme complexity is larger in comparison. This shift is even more noticeable as we apply the factorization in the linear layers, sometimes resulting in a 20% to 80% computation split.

7 Conclusion

This project investigated how the GPT model scales within certain computational constraints and what accelerators can be added to improve performance. The accelerators that were tested include SVD, butterfly, and monarch factorizations which were all applied to the last linear layer of each attention block. As shown in the results, off-the-shelf implementations of these factorizations often lead to worse performances whereas adding the leniency of finetuning results in on-par performance. This seems to suggest that the generalizabilty of the pretrained weights depends on the scale of the model too and not just the size of the dataset. In other words, the GPT model learns a different set of pretrained weights depending on the original model complexity. Intuitively, this means that the weights learned from the original pre-training is at a local minimum. Future work in this area can be to investigate how to make GPT find the global minima. Perhaps we will need to insert factorization techniques in layers outside of the attention block too.

References

[1] Eichhorn Rudra Re Dao, Gu. Learning fast algorithms for linear transforms using butterfly factorizations. In *International Conference for Machine Learning*, 2019.

[2] Dao et. al. Monarch: Expressive structured matrices for efficient and accurate training.

[3] Kaplan et. al. Scaling laws for neural language models.

[4] Child Luan Amodei Sutskever. Radford, Wu. Language models are unsupervised multitask learners.