Kronecker Products and GPT2

Ayoub @ Uni Passau

March 4, 2024

Abstract

GPT2 checkpoints from HuggingFace reach a approximate loss of 3.11 on OpenWebText. In this work, we factorize the main parameters of GPT2, as Kronecker Products leading to a compression factor of $x\%$, and ask the following question: What is the fastest way to reach back to the 3.11 loss using the Kronecker factors. And how to optimally manipulate training and distillation of the small compressed network. We particularly investigate freezing (for both training and distillation), and smooth transitioning (explained later on).

- **freezing** the pre-trained weights, helps boost accuracy.
- One by One distillation: drop the weights slowly and distill through the network.

Methodology:

1. Train GPT2 to a certain level. – 3.11.
2. Decompose the MLP weights.
3. Try a bunch of methods and see which one guarantees getting back to the original loss as fast as possible.

Contents

1 Resutls

		Datasets		
# Params	Model	wikitext-103	wikitext-2	Lambada
124M	GPT2	29.16	24.67	45.28
82M	DistilGPT2	44.53	36.48	76.00
81M	**KronyPT-81M-1350**	41.98	34.99	-
81M	**KronyPT-81M-3950**	-	-	64.92
81M	TQCompressedGPT2	40.28	32.25	64.72
81M	KnGPT-2 (Huawei)	40.97	32.81	67.62

Table 1: Comparison of different models on various datasets.

2 Introduction and Setup

A lot of work has been done on the compression of LLMs using factorization methods, including Kronecker Products. Most notably for encoder based based, and rarely applied to decoder based methods ([1], [2]). To the best of our knowledge, not many have focused on Freezing the initial pre-trained weights, i.e., most methods drop newly factorized matrices into the already pre-trained parameters, without carefully blending, nor assessing/studying how much the model is relying on already trained weights.

In this work:

- We investigate the impact of freezing on the quality of learning, and also demonstrate the capacity for the network to just not rely on the new weights, and just use the already pre-trained one.
- We also introduce a new distillation / training mechanism to ease the entry of these factorized weights into the old architecture.
- We also show that freezing the other weights, helps the new introduced get more activation.

When adding multiple Kronecker factors, we also investigate the impact of freezing the each factor, and train on only some parts of data..

In this work, we demonstrate that without freezing, the network relies more on the pre-trained weights (to a varying degree, depending on how the new proposed weights are initialized). And with freezing, newly introduced weights learn faster.

Usually the newly introduced weights are brute forced into training, and in many cases not, (knowing how many LLMs are under-trained + many residual connections), the pre-trained carry the weights. We also compare the gradient activations in both cases (when we free and when don't).

In this work we demonstrate the effectiveness of freezing, and of slow integration of weights, either through training or distillation.

Using Van Loan has no remarkable benefit improvement over 50% pruning.

2.1 Learning rate, batch size, and gradient accumulation

In this section, we discuss the impact of the **learning rate**, **batch size** and **gradient accumulation** on the training trajectory, namely on the first 10000 iterations of training.

Based on these results, we follow the

- Learning rate: cosine schedule, with a warm-up of 500 tiers.
- Gradient accumulation: 3 across 4 nodes (a good balance between 2 and 5)
- Batch size: 12, better stability.

This configuration allows a balanced trade-off between speed of updates and stochasticity of the loss with each iterations. And it also allows multiple experiments to be run simultaneously on a node of 4 A100s. Reframe this: you lose a bit of performance, but you gain a lot of memory to try out other approach at the same time, sounds silly, but this is important when you're GPU poor.

So, with this config, and with only 100k steps, avereging approx 0.5

What can we do better? distillation?

2.2 Distillation

In this section, we inherit the checkpoint from the section before, and use GPT2 as a teacher network for a classic Inter Layer distillation. We try two methods.

2.3 a subsection Freezing maybe?

We believe that not freezing the original pre-trained weight, is eventually detrimental to overall training, and the largest the model is, the more the network relies on already pre-trained parameters. This is tracked using gradient checking. When we freeze the weights. To validate our hypothesis, we track gradient activation (also known as the attention matrix, not to be confused with attention matrix of transformer models, and notice that, when we freeze then fine-tune, we notice better gradient activation.)

grad.sum for : 1. resuming training for all params for 3k steps. 2. freezing for 2k, train all for 1k 3. 1 by 1 training with freezing (at each step, direct flow the flow to origin gpt2) for 2k, the tran all for 1k

same logic applies in

We test some interesting properties of distillation. We first position ourselves in a small setup, and then study how these observations scale to larger models.

3 Kronecker Decomposition

In this section, we study different Kronecker decomposition setups, and the percentage of compression it would lead to. So far we only decompose the weights `c_fc.weight` and `c_proj.weight` (each has $2.3M$ in the original GPT2-small architecture.). Each transformer layer (there are 12 in total) has two of these weights. They count to $56.6M$ in total (45% of GPT2 $124M$). Hence any significance reduction to these matrices would lead to a big (xx) compression of the whole model. We choose not to compress the weights of the attention for various reasons, first, it is showed that they're pretty dense (change this, and add more refs.), and second for implementation purposes (Flash Attention is too good!).

The parameter $p_1 =$ `c_fc.weight` (and $p_2 =$ `c_proj.weight`) has a shape of $(3072, 768)$ (resp. $(768, 3072)$). We first try the following decomposition: $p_1 = \underbrace{W_{11}}_{(3072,384)} \otimes \underbrace{W_{12}}_{(1,2)}$ and $p_2 = \underbrace{W_{21}}_{(384,3072)} \otimes \underbrace{W_{22}}_{(2,1)}$

This decomposition would lead to to reduction of $28M$ (50%). The new network would have approx $95M$. Our goal is to eventually reach the $85M$ mark, similar to DistilGPT2.

3.1 Different decomposition schemes:

In this section, we try different decompositions.

It is reasonable to aim for a decomposition that guarantees the maximum rank we can get. Since the Rank of Kronecker products is multiplicative, i.e., $\text{rank}(A \otimes B) = \text{rank}(A) \cdot \text{rank}(B)$ [1], we can easily compute the rank of each possible decomposition.

In our case, let $W \in R^{(m,n)}$ where $m = 3072$ and $n = 768$. So, for each layer of GPT2, we aim to find the "optimal" $A \in R^{(m_1,n_1)}$, and $B \in R^{(m_2,n_2)}$, s.t.:

$$W \approx A \otimes B, m = m_1 m_2, n = n_1 n_2$$

.

W.l.o.g, for each decomposition (A, B) the maximum rank we can reach is $\min(m_1, n_1) \cdot \min(m_2, n_2)$. And each of the reduced decompositions would have exactly $m_1 n_1 + m_2 n_2$ parameters. Hence, theoritically, the maximum rank we can get is 384 of a $(3072, 768)$ matrix. The following table summarizes all possible combinations, alongside the reduction it would lead to per layer, and the total number of parameters in GPT2, for only those decompostions of rank 384.

[1] Link to proof: https://math.stackexchange.com/questions/707091/elementary-proof-for-textrank-lefta-otimes-b-right-textranka-cdot

4 Impact of down-scaling: How small can we go

In this section, we study the impact of down-scaling on the compressed models, we train 4 different architectures, with variant dimensions:

- 67M: diverges // try with higher batch size (the least we can get)
- 81M: Super.
- 81M: with a Variant scheme
- 95M: duh duh duh. (the highest we can get)

Keep in mind:

- You can probably stabilize training using various tricks. It's just an endless loop. We are not going to play the What-IF game. One single config, that's it.
- One could try to have a mixed strategy:
  - higher levels of compression in the early layers.
  - higher levels of compression in the late layers.
  - Every odd layer
  - In the middle
-
-

4.1 Initialization

Here we try different initialization tricks, and see how the loss evolves with a few iterations of training. We particularly try Random initialization, Van Loan Initialization, and a special initialization that introduce and call, Sign Maximization, where we try to optimize the signs of the Kronecker Product then randomize. Results are summarized below:

Add a bar for 3.11, to show the target.

4.2 Teacher/Student architecture

- Teacher network: a 10M GPT2 like model. 6 layers of 6 heads.
- Student network: a 4M compressed model of the teacher where each weight W of the FFN is decomposed to $W \approx W_1 \otimes W_2$.

4.3 Methodology

1. We pre-train the teacher model for 1k steps.
2. Decompose the MLP weights into Kronecker Products.
3. Resume the training with 2 variations:
    - How we initialize the KronP factors (either Random or Van Loan)
    - Either or freeze the other pre-trained weights.

4.4 Questions

- When we plug new KP matrices to the pre-trained model, is it useful to freeze the other weights that are already pre-trained?
- When don't don't freeze the weights. Does the network rely on other pre-trained methods? * **The idea that I'm challenging here**: A lot of work on NNs compression using factorized methods, claim that the network only need to be (post-)trained on small * But, what the fail to mention or elaborate on at least, is that not the weights matrices of the network are decomposed. * And with all the residual connection that are present in most attention networks, one could suspect, that maybe the weights that were not decomposed are taking over... * A good remedy for this is to freeze the original weights during the post-training, and only allow the new dropped in (factorized matrices) to be updated with backprop.
- This strategy could also be implemented in distillation. We refer to this method as **forced distillation.**
- When we freeze, we investigate how useful is it to distill matrices one by one, rather than drop in the matrices all at once. And also investigate if the order of dropping has any significance, bottom up or top bottom...

5 Toy experiments: 14M GPT2 model.

I'll refer with the normal setup **NS** to the original model (with no factorization), and with the Kronecker Products Setup **KPS** to the new KP factorized model.

We first run the following 5 training runs:

-
- Run the NS for 4K steps.
- 2 runs with different learning rate strategies.
- Run the NS for 2K, plug-in the KP weights (randomized) then train for 2k more steps.

- Same as **2.** with a Freezing variation:
  - Run the NS for 2K, plug-in the KP weights (VL init)
  - Freeze the other weight, and only train the KP matirces for 1K
  - Unfreeze the original weights, i.e., train all parameters again.
- Run the NS for 2K, plug-in the KP weights (VL init) then train for 2k more steps.
- Same as **4.** with a Freezing variation:
  - Run the NS for 2K, plug-in the KP weights (VL init)
  - Freeze the other weight, and only train the KP matrices for 1K
  - Unfreeze the original weights, i.e., train all parameters again.

Another thing I should try:

- compare 1 on 1 training with whole training at once. Just for training. One the same number of iterations. With freezing of course.
- play with different lr strategies of pre-trained / post-trained parameters

Notes:

- It is worth mentioning that changing the learning rate can significantly change the outcome, since the original weights are smoother with a weight-decay. One should be vigilant.
- The model overfit quickly.

somestuffstyle

Training the base model for 5k steps with different learning rates curves as depicted below, leads to the following training / validation loss curves:

As a rule, I'll set the learning$_rateofallalready-pretrainedtoaconstant$(1e3)$, whiledecayingthenewfactoriz$

6 Results

References

[1] M. Tahaei, E. Charlaix, V. Nia, A. Ghodsi, and M. Rezagholizadeh, "Kroneckerbert: Significant compression of pre-trained language models through kronecker decomposition and knowledge distillation," in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2116–2127, 2022.

[2] A. Edalati, M. Tahaei, A. Rashid, V. P. Nia, J. J. Clark, and M. Rezagholizadeh, "Kronecker decomposition for gpt compression," *arXiv preprint arXiv:2110.08152*, 2021.