Kronecker Products and GPT2

Ayoub @ Uni Passau

April 5, 2024

Abstract

What do we do differently: **We** have empirical evidence not to use distillation, as experiments show better results with just Vanilla supervised training. **We** don't compress the attention matrices matrices, and focus mainly on the MLPs (why?, see below). **We** try different compression ratios and see how far we can push down the size. **We** add multiple Kronecker Factors. **Finally**, we introduce a new (and efficient, as it doesn't require any extra compute) initialization trick based on pruning.

Some false narratives of the other papers

- **Parameter count**: It appears like the competitor paper do not count the output matrix, and I quote "Note that number of parameters of the models are reported excluding the output embedding layer in language modelling which is not compressed, is equal to row Parameters". If this is true, then their model would technically be a 120M. The reason why we (and GPT2 paper) don't count the output matrix is because it is identical to the embedding matrix, this known as weight tying or weight sharing. Which does not seem the case with the other papers (I have contacted the other paper for clarifications through github).
- "**We only use 3% of the data**": both papers claim that they only used 3% of the training data to train their models. We argue that in this set up, limiting your training data to only a fraction does not imply a better compression/factorization method, for the simple and following reasons:
    1. They inherit almost half the weights from the old GPT2 checkpoint that was trained for numerous epochs, and has seen the whole data.
    2. They use Van Loan method, hence, even, when they don't match the original checkpoint, VL is still a smart init, some knowledge is definitely passed through the SVD.
    3. You literally use the output matrix of a pre-trained GPT2.

Hence, we don't know "how much knowledge" has been passed through the already trained parameters. A fair comparison would be to initialize **all the parameters** of the new compressed model with random values, and not rely on any of the other pre-trained ones. Which is clearly not the case here.

Contents

1 Resutls:

Early results on wikitext103, wikitext2 [1] and Lambada [2]. **KronyPT** is our model, and the suffixes 1350 and 3950 refer to different checkpoints. Both models are already outperforming distilGPT2 ([3]) on the 3 datasets. And the 3950 checkpoint is outperforming **KnGPT2** ([4]) on Lambada, while slightly still behind **TQCompressedGPT2** [5]. We measure perplexity (duh) using the Hugging Face interface that was used by the other papers.

		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**

Table 1: Perplexity results of Krony-PT and DistilGPT

		Datasets		
# Params	Model	wikitext-103	wikitext-2	Lambada
81M	**KronyPT-81M-1350**	41.98	34.99	-
81M	**KronyPT-81M-3950**	-	-	64.92
119M	TQCompressedGPT2	40.28	32.25	64.72
119M	KnGPT-2 (Huawei)	40.97	32.81	67.62

Table 2: Perplexity of Krony-PT against other Kronecker based models.

2 Introduction

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 ([4], [6], [5]).

We study how far we can push the compression ratio, and record then impact of down-scaling on the quality of the model. We also investigate adding multiple Kronecker factors.

3 Training Set-up

Add here the basic set-up that I used to train the latest models, basically Chinchilla recommendations. (for later)

Some key differences:

- **Batch size:** We find better and more stable loss curve with higher batch sizes, a typical step would need 6 batches (gradient acc.) of 12 samples.

- **Learning rate:** We found better, and more stable results when using a constant learning rate of $6e-4$. (check this again), The other set-up is to use a cosine scheduler, peaking at $6e04$ after a warm-up of 500 steps?. Learning rate

4 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 significant reduction to these matrices would lead to a remarkable compression ratio of the whole model. We choose not to compress the other weights, namely, attention weights and embedding matrix for various reasons that we will expose later on.

4.1 The 95M model

The most basic strategy is to divide one of the dimensions of each W by 2, this would lead to a 95M model. The parameter $p_1 =$ `c_fc.weight` (resp. $p_2 =$ `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 $82M$ mark, similar to DistilGPT2, and other Factorized models (inserts other refs here).

4.2 Different decomposition schemes:

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, we have $W \in R^{(m,n)}$ where $m = 3072$ and $n = 768$. Hence, for each layer of GPT2, we aim to find the "optimal" $A \in R^{(m_1,n_1)}$, and $B \in R^{(m_2,n_2)}$, i.e.,:

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

.

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,

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

the maximum rank we can get is 768 of a $(3072, 768)$ matrix. The following table summarizes some possible combinations, alongside the reduction it would lead to per layer, and the total number of parameters in GPT2, for only those decompostions of maximal attainable rank. We are particularly interested in 3 class of models, the 67M, the 81M and the 96M. (Need to add this) Furthermore, we add multiple factors to the models labeled with **MF** (second table / a few decompositions are missing, check this out. e.g., 3072, 384).

Name	Dimension	params	Model size
67M	(64, 32)	3200	67,893,504
	(64, 48)	3840	67,908,864
	(96, 32)	3840	67,908,864
	(64, 64)	4672	67,928,832
	(128, 32)	4672	67,928,832
	(96, 48)	5120	67,939,584
	(96, 64)	6528	67,973,376
	(128, 48)	6528	67,973,376
	(128, 64)	8480	68,020,224
	(96, 96)	9472	68,044,032
	(192, 48)	9472	68,044,032
	(128, 96)	12480	68,116,224
	(192, 64)	12480	68,116,224
	(128, 128)	16528	68,213,376
MF1	(256, 64)	16528	68,213,376
	...	...	...
MF2	(1024, 256)	262153	74,108,376
	(768, 384)	294920	74,894,784
	(1024, 384)	393222	77,254,032
81M	(768, 768)	589828	81,972,576
	(1536, 384)	589828	81,972,576
	(1024, 768)	786435	86,691,144
96M	(1536, 768)	1179650	96,128,304
GPT2	(3072, 768)	2359297	124,439,832

Table 3: Different compression schemes

Name	Dimension	params	1 factor	2 factors	3 factors
MF1	(256, 64)	16528	68,2M	68,6	69M
MF2	(1024, 256)	262153	74M	80M	86M

Table 4: Adding multiple Kronecker Factors

5 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 (67M with multiple factors / 1025-256 with two factors)
- 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

6 Initialization

Since we inherit a GPT2 checkpoint that was trained for multiple epochs on the Open Web Text (OWT) (cite here), we want to initiliaze our weights in a way that leverages the old pre-training as much as possible. This is of course obvious for the parameters that are common between **GPT2** and **KronyPT** (i.e., we match). But more tricky for the weights that are decomposed into Kronecker Factors. In our work, we try two different approaches, Van Loan decomposition (cite here), and a Pruning based method exclusively for the 95M model.

		Datasets		
# Params	Model	wikitext-103	wikitext-2	Lambada
95M	**Krony-PT1**	41.80	35.50	61.34
95M	**Krony-PT1**	41.81	36.02	59.95

Table 5: Comparison of different models on various datasets.

6.1 Van Loan Method

XX (maybe not even write this section)

6.2 Pruning based Initialization

We propose a new initialization strategy by inducing sparsity in the first factor of the Kronecker Product, and prune it by half. This is equivalent to picking the second factor as $\begin{bmatrix}1\\0\end{bmatrix}$. Now, we illustrate how this procedure works with a random matrix.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \end{bmatrix} \xrightarrow[\text{pruning}]{} \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ 0 & 0 & 0 & 0 & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ 0 & 0 & 0 & 0 & 0 \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ 0 & 0 & 0 & 0 & 0 \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \end{bmatrix} \otimes \begin{bmatrix}1\\0\end{bmatrix}$$

References

[1] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[2] D. Paperno, G. Kruszewski, A. Lazaridou, Q. N. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernández, "The lambada dataset: Word prediction requiring a broad discourse context," *arXiv preprint arXiv:1606.06031*, 2016.

[3] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," in *NeurIPS* $EMC^2 Workshop$, 2019.

[4] 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.

[5] V. Abronin, A. Naumov, D. Mazur, D. Bystrov, K. Tsarova, A. Melnikov, I. Oseledets, S. Dolgov, R. Brasher, and M. Perelshtein, "Tqcompressor: improving tensor decomposition methods in neural networks via permutations," *arXiv preprint arXiv:2401.16367*, 2024.

[6] 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.