Sparse Full-Rank MLPs for Increased Efficiency of Language Modeling

Stanford CS224N Custom Project

Aaryan Singhal
Department of Computer Science
Stanford University
aaryan04@stanford.edu

Quinn McIntyre
Department of Computer Science
Stanford University
qam@stanford.edu

Abstract

Memory bandwidth has increasingly become a limiting factor in efficient inference and training of Large Language Models Vaidya (2024). To overcome this, considerable work has been done on minimizing the number of parameters per-layer, or the size of these parameters. These methods include LoRA Hu et al. (2021), Quantization, Monarch Matrices Dao et al. (2022), and many more approaches. Even as memory bandwidth increases on consumer hardware. We sweep over monarch matrix sparsity configurations at the small language modeling scale on TinyStories Eldan and Li (2023), and confirm a direct exponential relationship in the decrease in model parameters as Monarch blocking becomes more sparse with a difference of 5.786 perplexity points from full matrix to diagonal monarch sparsity. We confirm that the decrease in model quality as blocking increases scales to GPT2-m sized models Radford et al. (2019). We demonstrate that there is an immediate cost that comes with speedup in this way, and show that there "is no free lunch."

1 Key Information to include

- Mentor: Sonia Chu
- External Collaborators: N/A
- Sharing project: N/A

2 Introduction

The NVIDIA RTX 4090 has 1 Tb/s memory speeds NVIDIA Corporation (2022) and 24GB of RAM. Considered to be frontier hardware, it is capable of running models up to 24GB on a single GPU. The compute die on the 4090 has over 80 Tflops NVIDIA Corporation (2022), so when running transformer architectures Vaswani et al. (2023), the bottleneck is entirely in the loading of parameters. The local cache on the 4090 Vaidya (2024) is not large enough to hold the entire model, so the parameters must be loaded one layer at a time. Given that there are $nm(2k-1)$ flops in an $n \times k$, $k \times m$ matrix multiply, and the MLPs in SOTA consumer models (llama 3 7b) is approximately 2/3 of the parameters AI@Meta (2024).

Training large machine learning models is expensive due to the incredible compute costs from operations on large matrices. An obvious approach to reduce the training cost is to enforce some structure on the parameters of the model so as to optimize the computations on the GPUs and have more efficient training.

A problem with current approaches is that the quality of the final model suffers greatly for any enforced sparsity on the matrix. The gains from sparsity occur not just in the training, but in the inference step of the model.

3 Related Work

3.1 LoRA

One method of layer-wise parameter efficiency is to use Low-Rank adapters to lower the parameter counts of layers Hu et al. (2021). This is commonly done in fine-tuning settings, but a model could be pretrained with low-rank matrices which would

3.2 Monarch

In Monarch Dao et al. (2022), Dao et al. present Monarch Matrices. Monarch matrices are sparse matrices that are highly expressive and can represent a wide range of operations. Let $n = m^2$. An $n \times n$ Monarch matrix has the form:

$$\mathbf{M} = \mathbf{P}\mathbf{L}^\top\mathbf{R},$$

where $\mathbf{L}$ and $\mathbf{R}$ are block-diagonal matrices, each with m blocks of size $m \times m$, and $\mathbf{P}$ is the permutation that maps $[x_1, \ldots, x_n]$ to $\left[x_1, x_{1+m}, \ldots, x_{1+(m-1)m}, x_2, x_{2+m}, \ldots, x_{2+(m-1)m}, \ldots, x_m, x_{2m}, \ldots, x_n\right]$.

These matrices have many convenient properties to suggest them as a promising candidate for sparse model training systems, namely that monarch matrices and the product of monarch matrices can express many operations, like the Fast Fourier Transform, Toeplitz matrices, and many other matrix transformations.

These matrices can be dropped in to a model in place of linear layers and ran out-of-box. When training a GPT-2 architecture with monarch matrices, they achieve similar perplexity but with 2x the speed.

Additionally, the paper presents three training methods that can be employed using monarch matrices. First, end-to-end sparse training of models, which is training a model with monarch matrices. Second, sparse to dense training, where a model is trained mostly with monarch matrices but then the other weights are turned on to allow for full expressivity near the end of training, and third, dense to sparse training, where monarch matrices are distilled from a dense network to allow for fast inference on the network while maintaining similar quality.

4 Approach

We build on the work of Monarch Matrices Dao et al. (2022) Fu et al. (2023) and explore the effect of variable blocking in different language modeling regimes. Blocking reduces parameters per layer, but this leads to a reduction in of the language model's quality.

4.1 Parameter Calculation Formulas

4.1.1 Standard MLP

The formula for the parameter count of each layer in a standard MLP is:

$$\text{Parameters}_{\text{standard}} = \text{in\_features} \times \text{out\_features} + \text{out\_features}$$

This calculation includes the weights matrix and the bias vector for each layer.

4.1.2 Monarch Linear Model

The calculation for the Monarch Linear model with different block sizes is given by:

$$\text{Parameters}_{\text{blkdiag1}} = \begin{cases} \text{nblocks} \times \text{in\_blksz} \times \text{in\_blksz} & \text{if in\_features\_extended} < \text{out\_features\_extended} \\ \text{nblocks} \times \text{out\_blksz} \times \text{in\_blksz} & \text{otherwise} \end{cases}$$

$$\text{Parameters}_{\text{blkdiag2}} = \begin{cases} \text{nblocks} \times \text{out\_blksz} \times \text{in\_blksz} & \text{if in\_features\_extended} < \text{out\_features\_extended} \\ \text{nblocks} \times \text{out\_blksz} \times \text{out\_blksz} & \text{otherwise} \end{cases}$$

$$\text{Parameters}_{\text{bias}} = \text{out\_features}$$

$$\text{Total Parameters}_{\text{Monarch Linear}} = \text{Parameters}_{\text{blkdiag1}} + \text{Parameters}_{\text{blkdiag2}} + \text{Parameters}_{\text{bias}}$$

These formulas account for the number of blocks (nblocks), the calculated block sizes (in_blksz and out_blksz), and whether the extended feature dimensions (in_features_extended and out_features_extended) affect the shape of the block diagonal matrices.

4.2 Consequential Bandwidth Gains

The effects of this sparsity are immediate, as seen by the speedup in loading each matrix and the full MLP. Directly increasing the sparsity in this way gives a directly proportional lift in speed of loading on the 4090. Looking at table 4.2, the direct effects of lower parameters in the MLP are realized immediately. Because a model like llama-3 7b Vaidya (2024) has to be loaded one layer at a time due to the size of the cache and the MLP make up approximately 2/3 of the parameters in the model, with increased sparsity in an architecture like llama, this will lead to a much larger speedup than any optimization of attention AI@Meta (2024).

Configuration	**Standard MLP**	**8 Blocks**	**16 Blocks**	**32 Blocks**
First Layer (1024 to 4096)	4,198,400	659,456	331,776	167,936
Second Layer (4096 to 1024)	4,195,328	656,384	328,704	164,864
Total Parameters	8,393,728	1,315,840	660,480	332,800
Loading Time (s)	1.68×10^{-5}	2.63×10^{-6}	1.32×10^{-6}	6.66×10^{-7}

Table 1: Loading times for various blocking configurations of a neural network model in bf16 format being loaded from RAM into the cache on an NVIDIA RTX 4090.

5 Experiments

We ran experiments in the small and large language modeling regime and demonstrated that the trade-off in performance via sparsity and perplexity scales from the small to large language modeling scale.

5.1 Data

To explore the effects of various degrees of blocking, we started with a small language modeling task with a smaller token space than general web text to use a lightweight model to see if there are any effects.

We scaled up to larger language modeling and trained on a larger slice of human language data to verify that our results hold in the large-language modeling regime.

5.1.1 TinyStories

For the small language modeling task, we trained on the TinyStories dataset Eldan and Li (2023) which consists of simple synthetically generated stories and contains 2,141,709 samples. We used the Neox tokenizer Andonian et al. (2023) to tokenize the data.

5.1.2 SlimPajama

For the large language modeling task, we trained on the SlimPajama Soboleva et al. (2023) dataset and use a 6B token slice.

5.2 Evaluation method

We evaluated the performance of our language modeling approaches by the Cross-Entropy loss and perplexity that the various sparsity methods achieved. This loss objective was applied to the next token prediction task with an 80/10/10 train/val/test split.

5.3 Experimental details

The configuration we used for TinyStories is as follows in 2:

Table 2: Training Configuration for TinyStories GPT Model

Hyperparameter	Value
Dataset	tinystories
Tokenizer	GPT2
Gradient Accumulation Steps	1
Batch Size	32
Block Size	64
Number of Layers	8
Number of Heads	16
Embedding Dimension	64
Feedforward Network Type	monarch
Number of Blocks	[1, 2, 4, 8, 16, 32, 64]
Learning Rate	3e-4
Learning Rate Schedule	Linear Warmup with Cosine Annealing
Minimum Learning Rate	4e-5
Optimizer	Adam
Weight Decay	1e-1
Beta1	0.9
Beta2	0.95
Gradient Clipping	1
Trainer Maximum Steps	10,000

The configuration used for SlimPajama is as follows in 3

Table 3: Training Configuration for SlimPajama

Hyperparameter	Value
Dataset	SlimPajama
Tokenizer	GPT2
Gradient Accumulation Steps	16
Batch Size	16
Block Size	2048
Hidden Dimension	1024
Number of Heads	16
Number of Layers	24
Learning Rate	8e-4
Learning Rate Schedule	Linear Warmup with Cosine Annealing
Optimizer	AdamW
Feedforward Network Type	monarch
Number of Blocks	[1, 4, 8, 16, 32]
Minimum Learning Rate	8e-5
Weight Decay	1e-1
Beta1	0.9
Beta2	0.95
Gradient Clipping	1
Trainer Maximum Steps	20,000

5.4 Results

5.4.1 Small Language Modeling

In figure 1 and figure 2 we demonstrate the quality drop off proportional to the parameters (via blocking) because perplexity is exponential of loss, we demonstrate a direct correlation between parameters in a blocked matrix MLP and increase in perplexity.

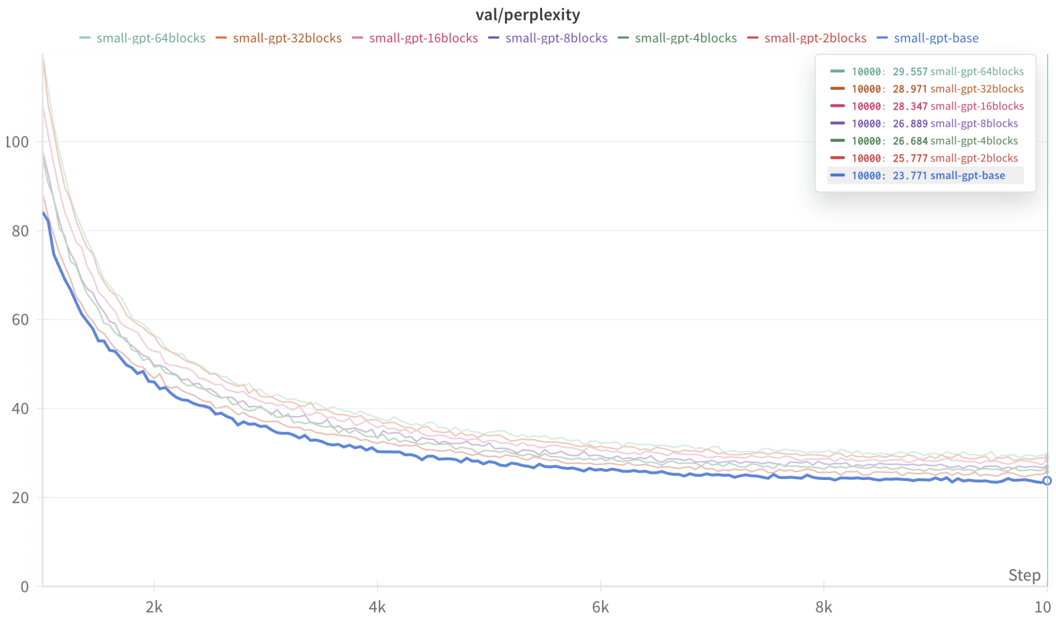

Figure 1: Perplexity of Monarch Matrices Trained on TinyStories

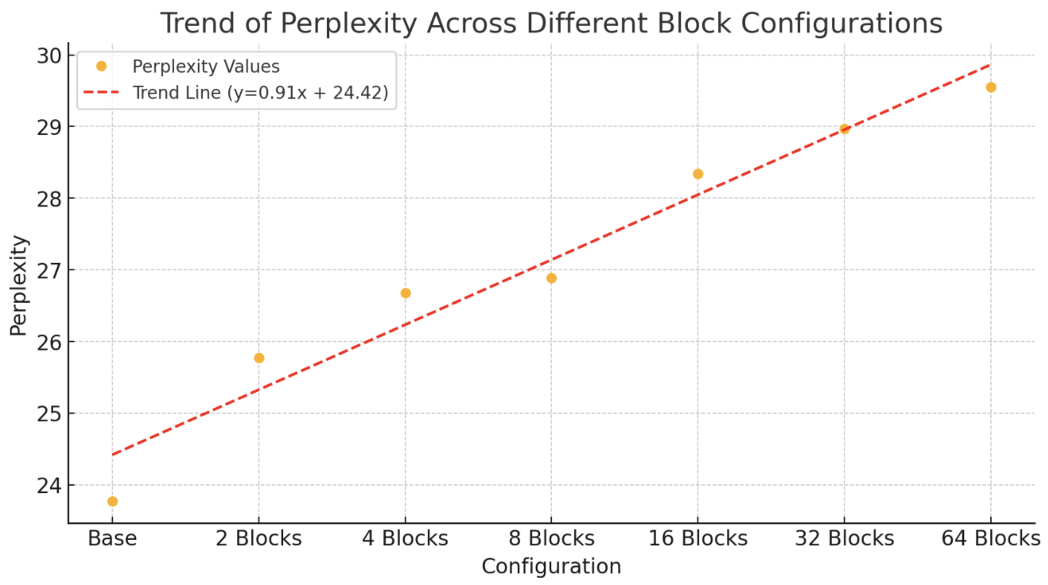

Figure 2: Perplexity of Monarch Matrices Trained on TinyStories Linear Fit

5.4.2 Large Language Modeling

We observe a similar trend on large language models (although we trained on fewer blocking configurations due to the expense of training models at this scale, as seen in table 4.

Base	4 Blocks	8 Blocks	16 Blocks	32 Blocks
14.685	16.431	17.276	18.085	18.572

Table 4: Perplexity values across various blocking degrees.

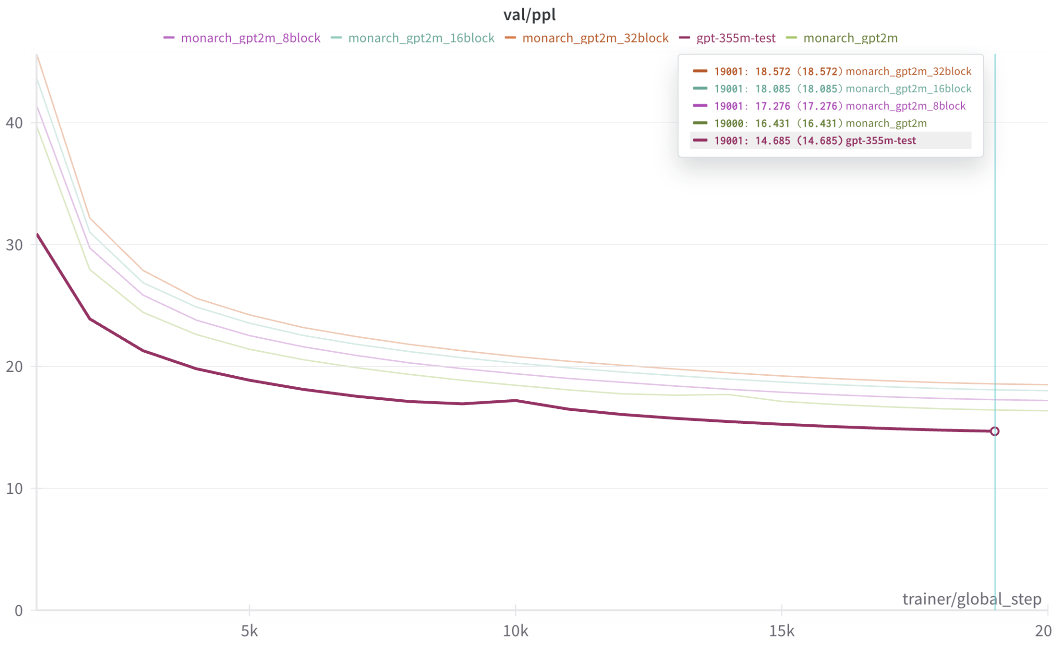

Figure 3: Perplexity of Monarch Matrices Trained on SlimPajama

6 Analysis

We quantitatively demonstrated a drop-off in model quality proportional to increasing sparsity of the monarch layers in the MLPs. We see similar drop off in the quality of writing. Upon inspection of the effects of increased sparsity within TinyStories:

Base model example generation:

```
<|startoftext|>and the family took a big hug. They hugged her and said
they was sad, so they was happy. That was a little girl who loved to play
with her friends. She went to make a big hug and took her special day.

She<|endoftext|>
```

16 Block Monarch example generation:

```
<|startoftext|> to the old toy. Dad took out a bow and went!<|endoftext|>
Once there was a boy named Mia. He liked to go and his friends to play.
One day, he had an old man and find one. He would have many things when
```

Once the blocking has been increased to 64 blocks in the Monarch matrix, the quality of the generations has notably decreased.

```
<|startoftext|> every afternoon. It was a yellow thing to explore the sky.
One day, the end of the old man said, "What’s go back?"
The man became happy and thanked the man in the table. The little girl was
```

One notable difference is that while general grammer is correct regardless of the sparsity, increasing the sparsity of the Monarch MLPs leads to words being used in non-sensical places. The sentence flows, the meanings of many things do not fit.

This suggests that there is some

7 Conclusion

We find that in the consumer inference regime, where users of large language models are purely bandwidth bound, training end-to-end sparse model via Monarch Matrices of the same configuration leads to a loss in perplexity proportional with the log number of parameters lost. This means that the loss in model quality is directly proportional to speedup in inference of the model, pointing to a "no free lunch" property of these models.

8 Ethics Statement

With any method that seeks to improve the quality and speed of models deployed on edge compute will allow for individuals to run larger and higher quality models feasibly on their own machine. Additionally, with work like chain of thought Wei et al. (2023), it has been shown that model quality can improve with recursive prompting, so faster inference allows for a higher-performance model in the hands of individuals.

When deployed through an API, models like GPT-4V and Anthropic those companies can manage the requests being made to the language models as well as the things that the language models are allowed to do. This serves as providing an institutional barrier which mitigates the harms.

To mitigate this risk, the best approach is to release high quality models that are aggressively aligned, with the hope that it is difficult to jailbreak them. Additionally, aggressive monitoring of digital content to detect AI generated materials could identify when LLMs are being used to generate nefarious text in public forum which could mitigate harm.

References

AI@Meta. 2024. Llama 3 model card.

Alex Andonian, Quentin Anthony, Stella Biderman, Sid Black, Preetham Gali, Leo Gao, Eric Hallahan, Josh Levy-Kramer, Connor Leahy, Lucas Nestler, Kip Parker, Michael Pieler, Jason Phang, Shivanshu Purohit, Hailey Schoelkopf, Dashiell Stander, Tri Songz, Curt Tigges, Benjamin Thérien, Phil Wang, and Samuel Weinbach. 2023. GPT-NeoX: Large Scale Autoregressive Language Modeling in PyTorch.

Tri Dao, Beidi Chen, Nimit Sohoni, Arjun Desai, Michael Poli, Jessica Grogan, Alexander Liu, Aniruddh Rao, Atri Rudra, and Christopher Ré. 2022. Monarch: Expressive structured matrices for efficient and accurate training.

Ronen Eldan and Yuanzhi Li. 2023. Tinystories: How small can language models be and still speak coherent english?

Daniel Y. Fu, Simran Arora, Jessica Grogan, Isys Johnson, Sabri Eyuboglu, Armin W. Thomas, Benjamin Spector, Michael Poli, Atri Rudra, and Christopher Ré. 2023. Monarch mixer: A simple sub-quadratic gemm-based architecture.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models.

NVIDIA Corporation. 2022. NVIDIA Ada GPU Architecture Whitepaper. Technical report, NVIDIA Corporation. Accessed: 2024-06-05.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. 2023. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. `https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama`.

Neal Vaidya. 2024. Mastering llm techniques: Inference optimization. NVIDIA Developer Blog.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.