Optimal Brain Projection: Neural Network Compression using Mixtures of Subspaces

Daniel Garcia
sorvisto@stanford.edu

Roberto Garcia (not in class)
robgarct@stanford.edu

Andri Vidarsson (not in class)
andriv@stanford.edu

Abstract

Neural Networks, particularly Large Language Models (LLMs), have shown impressive performance in machine learning tasks. However, they often require an immense amount of parameters, which presents a practical setback when using them for inference. Hence, recent techniques have emerged in LLM compression. Compression has also found popularity in other areas recently, such as image or video compression, particularly through Subspace Clustering. In this work, we frame the LLM compression task as finding the union of multiple low-dimensional linear subpsaces that span the rows from the weight matrices in the LLM. To solve this task, we introduce a new neural network and LLM compression framework called Optimal Brain Projection. Additionally, we study and coin the term Projection Loss, which is the second order Taylor expansion of the loss of the model incurred when projecting its weights to a union of low-dimensional linear subspaces. Notably, Optimal Brain Projection leverages Subspace Clustering and Projection Loss to perform model compression. We demonstrate the efficacy of the proposed approach by compressing the LLM Phi-1.5, and evaluating it on multiple benchmarks. Optimal Brain Projection obtained outstanding results, outperforming a modified version of LLMPruner, the state-of-the-art at their time of publication, by a 3% increased average accuracy. We also demonstrate the fine-grained accuracy of Projection Loss at predicting changes in validation loss, which we think was a key component to the efficacy of the proposed approach.

- External mentor: Professor Azalia Mirhoseini
- CS224N mentor: Aditya Agrawal
- Project is shared by team member Daniel Sorvisto in class APPPHYS229.

1 Introduction and Related Work

State-of-the-art Large Language Models (LLMs) have parameters in the order of billions or even trillions. There's a huge computation cost involved when training or doing inference using these models. To mitigate this, many researchers have resorted to training from scratch the same model using multiple sizes and releasing multiple versions of them. This is very inefficient. Hence, there's been a recent focus in the field of Large Language Model compression through quantization [1], pruning [2] or distillation [3].

The groundbreaking works of Optimal Brain Damage [4] and Optimal Brain Surgeon [5] were the first to propose a method to prune weights of conventional neural networks by their "importance" as a method to compress models. They define importance as the change in loss incurred by taking a weight and perturbing it, which is estimated through a second order Taylor expansion. In a later work, [6] compressed the matrices of a Language Model using a low-rank approximation of them obtained by minimizing the second order approximation of the loss incurred by using the low-rank matrix instead of the full-rank matrix.

More recent work has focused on pruning LLMs. Two lines of research have emerged in this field. Unstructured pruning seeks to remove individual entries from the weight matrices of the models, which reduces the memory required to store them. However, it rarely speeds up inference since sparse matrix multiplication is slow on conventional hardware [7]. On the other hand, structured pruning removes entire blocks of weights from the weight matrices. This approach achieves practical speedups, however, it usually results in compressed models that perform worse than those obtained through unstructured pruning. Our work takes the structured pruning route. Recent progress has been made in this end. Namely, LLM-Pruner[2] was able to preserve up to 95% of model performance when removing 20% of the model parameters. LoRAPrune[8], another recent method, proposed an efficient method of estimating importances using a LoRA adapter [9].

Orthogonal to neural network compression, in the past few decades, a line of research has focused on tackling Subspace Clustering [10]. Subspace Clustering is the task of splitting any data into different subspaces, commonly linear in nature. Algorithms to address this task have been particularly popular in the area of image and video representation. As outlined in [11], multiple algorithmic frameworks have been proposed to address this problem. Namely, iterative methods like K-subspaces [12], algebraic methods like Generalized PCA [13] or spectral clustering methods like Sparse Subspace Clustering[11] have been proposed. Spectral Clustering methods have found recent popularity, due to their lack of assumptions and empirical success. Notably, in our approach, we leverage Sparse Subspace Clustering to project weight matrices into multiple low-dimensional subspaces.

In this work, we develop a method for LLM compression, capable of reducing the memory and floating-point-operations (FLOPs) required by the model. Concretely, we improve upon the approaches proposed in [6] and [2]. In their work, [6] make some assumptions which we believe deteriorate model compression quality. Namely, their method is limited to assume that every weight on a row of a weight matrix is equally important. In addition, their method approximates weight matrices using a single low-rank approximation. On the other hand, the method from [2] is limited to zero-out entire rows or columns of the weight matrices in the model. In this work, we allow for different elements in the weight matrices to have varying importances. Additionally, we use multiple low-rank approximations to compress the weight matrices. Finally, we do not force our approach to completely zero-out any rows or columns from the weights.

2 Methods

2.1 Mixture Of Subspaces Approximation

First, we describe and formalize the problem of approximating the weights in a neural network using a mixture of subspaces, which is the core of our proposal. To begin, consider a neural network with multiple weight matrices W^i. Now, given a dataset D consisting of several training samples and labels, the original model was trained to minimize the loss on the dataset at hand $L(D, \{W^1, W^2, ..., W^k\})$ parameterized by all the weight matrices. Now, when compressing a model, we try to come up with a compressed set of weights $\hat{W}_i$ such that the loss $L(D, \{\hat{W}^1, \hat{W}^2, ..., \hat{W}^k\})$ is minimized. Hence we are trying to find the optimal $\hat{W}$ to solve:

$$\text{argmin}_{\hat{W}} L(D, \{\hat{W}^1, \hat{W}^2, ..., \hat{W}^m\}) \tag{1}$$

For our compression, we seek to find matrices $\hat{W}_i$ that live in a low-dimensional space. Concretely, we want to find matrices $\hat{W}_i$ whose rows are approximated by a union of low-dimensional linear subspaces, which we refer to as mixture of subspaces. This problem could be better understood as trying to find disjoint groups of rows $G = \{g_1, g_2, ..., g_k\}$ of a given matrix W such that we can approximate each of the groups individually by the low-rank matrices $\Omega = \{W'_1, W'_2, ..., W'_k\}$. For simplicity, in the case our neural network is parameterized by a single matrix W, then our problem becomes to find:

$$\text{argmin}_{G,\Omega} L(D, \hat{W}(G, \Omega)) \tag{2}$$

Observe that in eqn. 2, our compressed $\hat{W}(G, \Omega)$ is parametrized by the row groups G and low-rank matrices Ω. Here, we construct $\hat{W}$ by looking at each row of W, finding its row group g_i in G and then getting the approximation of that row by extracting it from the corresponding low-rank matrix W'_i from Ω. As you can see, W is effectively compressed to a set of low-rank matrices Ω and the row groups G.

2.1.1 Projection Loss

Optimizing eqn. 2 is non-trivial. One important setback is that evaluating $L(D, \hat{W})$ is expensive. For that reason we start by considering the second order Taylor expansion to approximate the loss of the model for a given dataset D incurred by taking an arbitrary set of weights w (expressed as a vector in eqn. 3 for simplicity) and perturbing them to $\hat{w} = w + \delta$:

$$\Delta L(D) = L(D, w) - L(D, \hat{w}) \approx \sum_i \frac{\partial L(D)}{\partial w_i}\delta_i + \frac{1}{2}\delta^T H \delta \approx \frac{1}{2}\sum_i H_{i,i}\delta_i^2 \tag{3}$$

As can be observed in eqn. 3, we make two simplifying assumptions to come up with the second order approximation of the change in loss $\Delta L(D)$. Normally, the model has been trained already in the given dataset D up to convergence, hence we expect the first order term $\frac{\partial L(D)}{\partial w_i}$ to be roughly 0. In addition, since computing the whole hessian is intractable with models having billions of parameters, we assume the hessian to be diagonal.

Now, with a reasonable approximation of change in loss at hand, we define the Projection Loss incurred by approximating the matrix W using $\hat{W}$ in eqn. 4. This is the loss obtained by projecting the groups of rows G of the matrix W into their corresponding low-rank subspaces Ω. Note that minimizing it is equivalent to minimizing eqn. 3.

$$\text{ProjLoss}(W, \hat{W}) = \sum_{i,j} H_{i,j}(W_{i,j} - \hat{W}_{i,j})^2 \quad \text{where: } H_{i,j} = \frac{\partial^2 L(D)}{\partial W_{i,j}^2} \tag{4}$$

More importantly, observe that optimizing for Projection Loss 4 is approximately equivalent to optimizing for the canonical loss from eqn. 2. This is a key observation in our approach, since evaluating ProjLoss$(W, \hat{W})$ is significantly cheaper than evaluating $L(D, \hat{W}(G, \Omega))$. Concretely, our approximation is:

$$\text{argmin}_{G,\Omega} L(D, \hat{W}(G,\Omega)) \approx \text{argmin}_{G,\Omega} \text{ProjLoss}(W, \hat{W}(G,\Omega)) \tag{5}$$

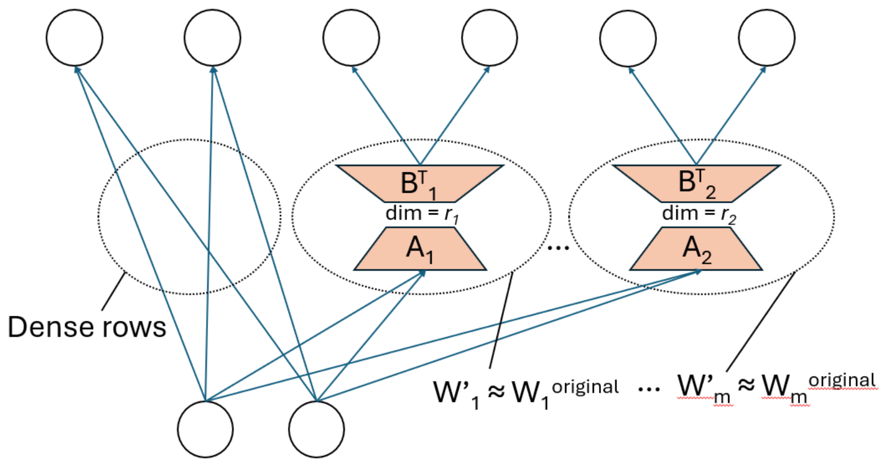
To minimize the Projection Loss from eqn. 4, recall that we are trying to find 2 optimal parameters. Concretely, we want disjoint groups of rows $G = \{g_1, g_2, ..., g_k\}$ such that we can approximate each of the groups individually by the low-rank matrices $\Omega = \{W'_1, W'_2, ..., W'_k\}$. Hence our original optimization problem from eqn. 2 can be simplified to optimizing:

$$\begin{gathered} \text{argmin}_{G,\Omega} \text{ProjLoss}(W, \hat{W}(G,\Omega)) \\ \text{subject to: PARAMS}(\Omega) \leq \gamma \end{gathered} \tag{6}$$

Observe that we have introduced a constraint in eqn. 6. Namely, we require the number of parameters (PARAMS) of all the matrices in Ω to remain below a threshold γ. Note that the number of parameters is directly controlled by the rank of each low-rank matrix W'_i since they can be represented by the product of 2 matrices $W'_i = A_i B_i$ that, in conjunction, can have less parameters than the original full-rank matrix. This constraint is the one controlling the model compression and speedup, since the FLOPs our model requires are proportional to the number of parameters.

2.2 Optimal Brain Projection

To find the optimal low-dimensional approximation $\hat{W}$ to the weight matrix W, we follow a 3-step procedure. Concretely, the task at hand is to find the optimal low-rank matrices $\Omega = \{W'_1, W'_2, ..., W'_k\}$, which we refer to as Subspace Experts, that constitute our compressed approximation $\hat{W}$, as well as the group to which each row in the original matrix W belongs to, described by $G = \{g_1, g_2, ..., g_k\}$. To achieve this, we first perform Subspace Clustering 2.2.1, further we do Rank Allocation 2.2.2, and finally we do Dense Matrix Splitting 2.2.3. The final compression resulting of our proposed approach, applied to a single MLP layer, can be observed in fig. 1.



B^T_1
dim = r_1
A_1
B^T_2
dim = r_2
A_2
...
Dense rows
$W'_1 \approx W_1^{\text{original}}$ ... $W'_m \approx W_m^{\text{original}}$

Figure 1: Optimal Brain Projection applied to a single MLP layer.

2.2.1 Subspace Clustering

The problem of taking an arbitrary matrix M and finding disjoint groups of rows that belong to different low-dimensional linear subspaces has been well studied in the past [10]. This task is commonly referred to as Subspace Clustering. However, Subspace Clustering, unfortunately, is just a proxy solution to our original optimization problem described in eqn. 6 since algorithms for this problem, to some extent, seek to minimize the unweighted Projection Loss for a matrix W:

$$\text{argmin}_{\Omega,G} \sum_{i,j} (W_{i,j} - \hat{W}(\Omega, G)_{i,j})^2 \tag{7}$$

Observe that eqn. 7 is unweighted Projection Loss, as it is missing the diagonal hessian $H_{i,i}$ weighting from our original Projection Loss from eqn. 4.

Nevertheless, we empirically found that using the solution to the unweighted-projeciton loss works well when used in combination with an appropriate rank allocation (described in section 2.2.2) and appropriate dense-matrix splitting (described in section 2.2.3). Hence, we use the Sparse Subspace Clustering approach. That approach essentially finds the row groups $G = \{g_1, g_2, ..., g_k\}$ that belong to different linear subspaces by approximating each row in the original matrix W by a linear combination of all the other rows. In fact, they frame this task as a convex optimization problem and enforce each linear combination to be sparse through $l1$ regularization. Further, after approximating each row by a linear combination of the others, they build a similarity matrix and do Spectral Clustering[14] on it. Their approach then yields the row-groups $G = \{g_1, g_2, ..., g_k\}$ that approximately solve the unweighted Projection Loss from eqn. 7. We refer readers to [11] for a detailed description of their algorithm.

2.2.2 Rank Allocation

Rank allocation is the process of optimally allocating FLOPs (equivalently parameters PARAMS) to the low-rank approximations W_i' from Ω given a set of row groups G (obtained from Subspace Clustering) and given a FLOP (or parameter) budget, usually 50% of the original matrix. Observe that we can allocate FLOPs to each low-rank matrix W_i' by varying its rank.

Now, to allocate a rank to every low-rank matrix W_i', we start by having them all be full rank. Note that, at that point, we are using more FLOPs and parameters than our desired budget, hence we have to iteratively reduce the rank of every low-rank matrix W_i' from Ω, starting from the full-rank matrices. To do this, we measure the Projection Loss incurred by removing one rank from a given matrix W_i'. We measure that amount by computing the difference in Projection Loss when using $W_i'^r$ v.s. $W_i'^{r-1}$, where the former is a rank r approximation and the latter a rank $r-1$ approximation. Notably, we really care about the Projection Loss incurred normalized by the FLOPs we will gain if we remove a rank from that matrix W_i'. The Projection Loss per FLOPs gained can be easily computed for every matrix W_i' by $(W_i'^r$ - $W_i'^{r-1})/(2 * (m+n))$ given $W_i' \in R^{m,n}$. With this in mind, we iteratively look at the Projection Loss per FLOPs gained by reducing the rank of every matrix W_i' by 1 and then remove a rank from the one with the highest value. Repeating this procedure enough times results in low-rank matrices W_i' that have a reduced FLOP count (or parameter count) and that optimize the Projection Loss from eqn. 4 given a fixed FLOP budget. This procedure is explained in detail by the following algorithm:

1. Start from a group of matrices Ω with an overbudget rank allocation $\{r_1, ..., r_n\}$ with f FLOPs. Compute the low-rank matrices Ω and their Projection Losses $\{p_1, ..., p_n\}$.
2. Compute the (uncomputed) candidate low-rank matrices Ω' for the next step with ranks $\{r_1 - 1, ..., r_n - 1\}$ and their Projection Losses $\{p_1', ..., p_n'\}$.
3. Decrease the rank of the matrix $W_i' \in \mathbb{R}^{m \times n}$ whose Projection Loss increase per FLOP decrease $\frac{p_i' - p_i}{2(m+n)}$ is the smallest. Replace r_i by $r_i - 1$, p_i by p_i' and decrease f by $2(m+n)$.
4. Repeat steps 1-3 until f is under the FLOP budget. For step 2, the only uncomputed matrix is the rank-decreased W_i'.

2.2.3 Dense Matrix Splitting

It is expected, as well as empirically observed, that some row vectors have a significantly higher importance than others. A weighted low-rank approximation will optimize its subspace so that it aligns closer to these vectors, worsening the projection of the other vectors. A naive idea would be to place the high-importance rows in their own low-rank matrix such that the optimal rank allocation algorithm will assign a high rank to that matrix. However, when the rank is large enough, the low-rank matrix actually consumes more FLOPs and parameters than the original dense matrix it is trying to approximate. We can get a better Projection Loss for a given FLOP budget if we do not project these high-importance rows at all.

The dense matrix splitting algorithm starts from the solution given by the optimal rank allocation, and ranks each row $\hat{W}(\Omega, G)_{i,:}$ by its Projection Loss per the increase in FLOPs if that row were to be fully included in Ω. In each iteration, a certain number of ranks are removed from $\hat{W}$, similar to the rank allocation algorithm, which frees up FLOPs to allocate for the dense rows. The rows with the highest Projection Losses per FLOPs are then moved to the dense matrix until the FLOP limit is reached. Since the ranks and row groups have changed, the low-rank matrices need to be recomputed at every iteration. This process is expensive, but can be made faster by removing a large number of ranks at every iteration, allowing for lots of rows to be moved into the dense matrix. The algorithm stops when the next iteration no longer improves Projection Loss. Empirically, the size of the optimal dense matrix is around 5-20% of the original matrix W. This procedure is explained in detail by the following algorithm:

1. Start from the low-rank matrices Ω and ranks $\{r_1, ..., r_n\}$ given by the rank allocation algorithm and their Projection Losses $\{p_1, ..., p_n\}$.
2. Compute the candidate low-rank matrices Ω' with ranks $r_i' = \text{round}(0.95 r_i)$ and their Projection Losses $\{p_1', ..., p_n'\}$.
3. Decrease the rank of the matrix $W_i' \in R^{m \times n}$ whose Projection Loss increase per FLOP decrease $\frac{p_i' - p_i}{2(r_1 - r_i')(m+n)}$ is the smallest. Replace W_i' with its lower-rank version. The current FLOPs f are decreased by $2(r_i - r_i')(m+n)$. Replace r_i with r_i'.

4. For all the rows in Ω, compute their Projection Loss $\sum_j H_{i,j}(W_{i,j} - \hat{W}(\Omega, G)_{i,j})^2$ and the FLOP increase from moving each row into the dense matrix $2(n - r_i)$. Move the rows with the highest Projection Loss per FLOPs ratio to the dense matrix until f is just under the FLOP budget.
5. Since the row groups G have changed, Ω has to be recomputed. Calculate the updated Ω'' and replace Ω with it if the total Projection Loss is smaller. If not, the algorithm stops.

3 Experiments, Results and Discussion

To evaluate our method, we applied it to the Multi-Layer Perceptron (MLP) layers of the Phi-1.5 [15] Large Language Model. This model has 24 MLP layers, each with two weight matrices sized 8192 by 2048, which we compressed using our technique. Following this, we performed a brief post-training phase using LoRA adapters to adjust the compressed weights one last time.

For our experiments, we used 50,000 samples from the TinyTextbooks [16] dataset for post-training, unless stated otherwise. We conducted our post-training using the HuggingFace library. We opted for a batch size of 60 samples, each up to 1024 tokens, and trained with the AdamW optimizer at a learning rate of 10^{-4} and standard settings. We started with the default hyper-parameters we found in the LLM-Pruner code and modified them a bit to fit our computational resources and goals. These settings helped us achieve training loss convergence in no more than 2 epochs, using either a single V100 Nvidia GPU or sometimes an A5000 Nvidia GPU. We report the cross-entropy (CE) loss on the TinyTextbooks validation set of 2,000 samples, and the accuracy on 1,000 random samples from several NLP benchmarks that test language understanding and common sense reasoning, namely: HellaSwag [17], PIQA [18], BoolQ [19], and WinoGrande [20]. Our main results are listed in Table 1.

3.1 Projection Loss

As shown in eqn. 5, Projection Loss is just a second-order Taylor approximation to the actual loss we are trying to minimize. However, in order for us to obtain Projection Loss, we had to compute the diagonal of the Hessian of the weights. Unfortunately PyTorch [21] is not well equipped to compute second order derivatives efficiently. Hence, we practically resorted to use the Fisher-Information matrix, which is commonly used to approximate the diagonal terms of the Hessian matrix [22].

Further, we conducted some experiments to evaluate the accuracy of such Projection Loss. We essentially look at the correlation between the Projection Loss measured in a set of Phi-1.5 models with perturbed weights and their post-training loss. Concretely, we take multiple instances of Phi-1.5 models with their original weights and perturb their weight matrices by two different noise levels. Each of these noise levels, namely 0.5 and 1, represents the magnitude of the noise added to each matrix. We perturb each matrix following the update $W := W + l * N * mean(abs(W))$ where l is the noise level, N is nose obtained from a standard normal distribution and the $mean(abs(W))$ is the mean absolute value of all the entries in the matrix W at hand. Finally, after obtaining multiple perturbations of the model, we measure their Projection Loss, post-train them for 200 steps, and obtain their validation loss. Notably, in addition to measuring the Projection Loss as described in eqn. 4, we also measure the unweighted Projection Loss from eqn. 7, to serve as a baseline.

Results can be observed in figs. 2a and 2b. As we can see, Projection Loss is very correlated to the post-training validation loss. This to us represents a very positive signal that minimizing Projection Loss when compressing the weights of a model should result in a model that also minimizes our validation loss. That is, this experiment is a good indicator that our approximation from eqn. 5 is very accurate.

3.2 Optimal Brain Projection

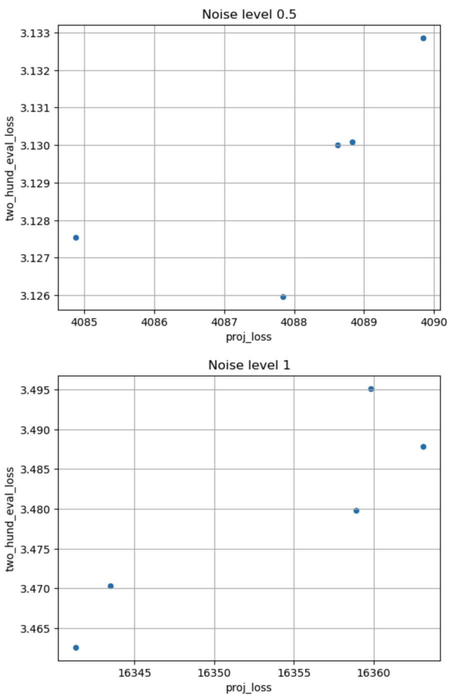
To assess the capability of our compression approach, we compress Phi-1.5's weights from the MLP layers by 50%, effectively reducing the FLOPs and number of parameters of the MLP layers by 50%. Concretely, we collected the cross-entropy loss and accuracy on benchmarks for 4 models, 3 which serve as baselines to our proposed approach. Their practical setup is described by the following:

Phi-1.5 (not pruned): This is the plain Phi-1.5 model out of the box from the HuggingFace library. Accuracy and loss results on this model serve as an upperbound to the results of compressed models.

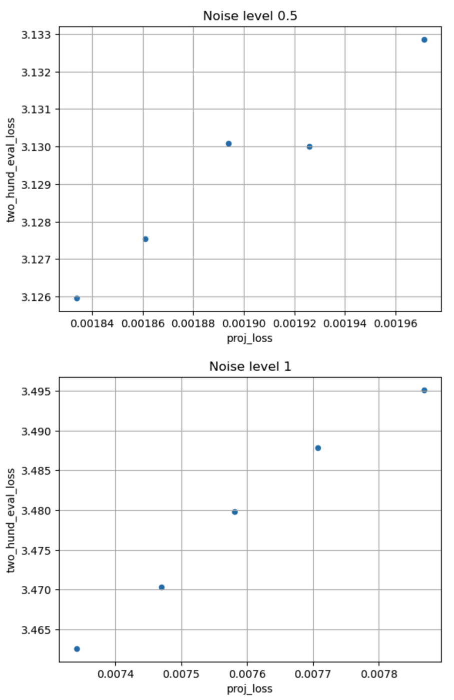
Magnitude-Pruned: For this model, we perform structured pruning of the hidden neurons of each MLP layer in Phi-1.5, similar to that suggested in [4]. Concretely, we drop the hidden neurons of the model that have the lowest magnitude on average. With the magnitude-pruned model at hand, we further performed post-training.

LLM-Pruner: Here, we apply LLM-Pruner[2] to each MLP layer in Phi-1.5. In their approach, to determine which neurons to drop, they look at a metric they call importance. In their case, the importance of each neuron is the second order approximation of the loss incurred by dropping the given neurons from the model. For a fair comparison, we only apply LLM-Pruner to the MLP layers, as opposed to the whole model. We refer to their work for details on their approach.

Optimal Brain Projection: This is our proposed approach. For this model, we just apply Subspace Clustering 2.2.1, Rank Allocation 2.2.2 and Dense Matrix Splitting 2.2.3, followed by post-training.



(a) Post-training validation loss v.s. unweighted Projection Loss



(b) Post-training validation loss v.s Projection Loss

Figure 2: Correlation of validation loss to unweighted and weighted Projection Loss.

Model	HellaSwag	PIQA	BoolQ	WinoGrande	Average Acc.	TinyTextbooks CE
Phi-1.5 Not-Pruned	48.0	77.1	73.6	72.3	67.8	3.17
Magnitude-Pruned	35.8	69.7	56.8	59.5	55.5	3.67
LLM-Pruner	36.4	72.6	64.6	57.5	57.7	3.38
Optimal Brain Projection (ours)	**40.8**	**73.1**	**64.9**	**64.1**	**60.7**	**3.13**

Table 1: Performance of pruning Phi-1.5 to 50% of the weights in the Multi Layer Perceptron layers using multiple methods. Bolded metrics are the best obtained in the respective benchmark, not considering the Phi-1.5 Not Pruned Model.

Results of all compression schemes can be observed in Table 1. Notably Optimal Brain Projection performed the best across all benchmarks, gaining a 3% average accuracy increase over our main baseline, LLM-Pruner, which was the state-of-the-art method as it's time of publication. In terms of cross-entropy loss, the proposed approach improved by 0.25 over LLM-Pruner, but more interestingly it also improved by 0.04 over the Not-Pruned Phi-1.5. The latter result seems counter intuitive, but we have a hypothesis for it. Since the dataset used to train Phi-1.5 is not public, we had to resort to use the closest open-source dataset we could find, namely TinyTextbooks. The use of a different dataset indicates us that Phi-1.5 might not have weights optimal for the TinyTextbooks dataset, hence why our post-trained pruned model was able to achieve a slightly better cross-entropy loss than it.

4 Conclusion

In this work we demonstrate that leveraging multiple low-rank approximations to compress weight matrices in a neural network is effective, as observed in Table 1, where Optimal Brain Projection outperforms LLM-Pruner, the state-of-the-art at its time of publication, on MLP compression. Additionally, we empirically demonstrate the fine-grained accuracy that Projection Loss has at approximating the actual loss of the model whenever its weights are perturbed. Importantly, we attribute a big portion of the efficacy of our approach to the high correlation of Projection Loss and validation loss, which is observed in fig. 2b. Moreover, we believe our approach should be practically useful due to its structured pruning nature, where literature has shown speed-ups can be achieved in conventional hardware.

Future work lies on more extensive hyperparameter tuning, studying the Projection Loss deeper and how the structure of the model is influenced by it, applying the approach to the full model (instead of only the MLP layers) and implementing hardware-efficient Subspace Expert layers.

5 Ethical Considerations

The development and deployment of model compression techniques present several ethical challenges and potential societal risks. The increased efficiency and accessibility of compressed LLMs could lead to widespread use in applications without adequate consideration of the ethical implications, such as surveillance, misinformation spread, or biased decision-making. This risk is particularly acute given the potential for these models to be deployed in a broader range of environments due to their reduced computational requirements.

In addition, the process of model compression itself may introduce or exacerbate biases in the model's outputs. By selectively pruning and approximating parts of the model, there is a risk that the resulting compressed model may not retain the original model's ability to fairly represent all demographic groups. This can lead to unfair or discriminatory outcomes, especially if the compressed models are used in sensitive applications like hiring, law enforcement, or loan approval processes.

To mitigate these risks, the following strategies can be implemented:

Ethical Oversight: Incorporate continuous ethical review processes, which engage diverse stakeholders to evaluate the implications of deploying compressed models in real-world scenarios.

User Education: Educate users about the strengths and limitations of compressed models, ensuring that they are aware of the appropriate and ethical use cases for these technologies.

These measures can help ensure that the benefits of LLM compression are realized while minimizing potential harms to society.

References

[1] H. Bai, W. Zhang, L. Hou, L. Shang, J. Jin, X. Jiang, Q. Liu, M. Lyu, and I. King, "Binarybert: Pushing the limit of bert quantization," 2021.

[2] X. Ma, G. Fang, and X. Wang, "Llm-pruner: On the structural pruning of large language models," 2023.

[3] H. Pan, C. Wang, M. Qiu, Y. Zhang, Y. Li, and J. Huang, "Meta-kd: A meta knowledge distillation framework for language model compression across domains," 2022.

[4] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2. Morgan-Kaufmann, 1989. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf

[5] B. Hassibi, D. Stork, and G. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE International Conference on Neural Networks*, 1993, pp. 293–299 vol.1.

[6] Y.-C. Hsu, T. Hua, S. Chang, Q. Lou, Y. Shen, and H. Jin, "Language model compression with weighted low-rank factorization," 2022.

[7] M. Sun, Z. Liu, A. Bair, and J. Z. Kolter, "A simple and effective pruning approach for large language models," 2024.

[8] M. Zhang, H. Chen, C. Shen, Z. Yang, L. Ou, X. Yu, and B. Zhuang, "Loraprune: Pruning meets low-rank parameter-efficient fine-tuning," 2023.

[9] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," 2021.

[10] R. Vidal, "Subspace clustering," *IEEE Signal Processing Magazine*, vol. 28, no. 2, pp. 52–68, 2011.

[11] E. Elhamifar and R. Vidal, "Sparse subspace clustering: Algorithm, theory, and applications," 2013.

[12] D. Wang, C. Ding, and T. Li, "K-subspace clustering," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2009, pp. 506–521.

[13] R. Vidal, Y. Ma, and S. Sastry, "Generalized principal component analysis (gpca)," 2012.

[14] A. Ng, M. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," *Advances in neural information processing systems*, vol. 14, 2001.

[15] Y. Li, S. Bubeck, R. Eldan, A. D. Giorno, S. Gunasekar, and Y. T. Lee, "Textbooks are all you need ii: phi-1.5 technical report," 2023.

[16] Nam Pham, "tiny-textbooks (revision 14de7ba)," 2023. [Online]. Available: https://huggingface.co/datasets/nampdn-ai/tiny-textbooks

[17] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, “Hellaswag: Can a machine really finish your sentence?” *CoRR*, vol. abs/1905.07830, 2019. [Online]. Available: http://arxiv.org/abs/1905.07830

[18] Y. Bisk, R. Zellers, R. L. Bras, J. Gao, and Y. Choi, “PIQA: reasoning about physical commonsense in natural language,” *CoRR*, vol. abs/1911.11641, 2019. [Online]. Available: http://arxiv.org/abs/1911.11641

[19] C. Clark, K. Lee, M. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova, “Boolq: Exploring the surprising difficulty of natural yes/no questions,” *CoRR*, vol. abs/1905.10044, 2019. [Online]. Available: http://arxiv.org/abs/1905.10044

[20] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, “WINOGRANDE: an adversarial winograd schema challenge at scale,” *CoRR*, vol. abs/1907.10641, 2019. [Online]. Available: http://arxiv.org/abs/1907.10641

[21] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” 2019.

[22] N. Srebro and T. Jaakkola, “Weighted low-rank approximations,” in *Proceedings of the 20th international conference on machine learning (ICML-03)*, 2003, pp. 720–727.