Optimized Structured Neural Pruning with Auxiliary Training

Apoorv Walia [1] **Hemanth Kumar Jayakumar** [1] **Sumukh Bharadwaj Koride** [1]

Abstract

This paper proposes a novel, structured algorithm for neural network training and efficient pruning. This is inspired by the idea of Sparse Structured Pruning Algorithm(i-SpaSP). As per the i-SpaSP paper, we prune the network by calculating the importance and active neurons. Additionally, following the pruning step, we would apply an Iterative Thresholding on the pruned weights as we prune to obtain an optimal pruned network. We show that the error induced by pruning in our approach decomposes polynomials for two-layer network architectures with ReLU activation, where the degree of this polynomial is arbitrarily large based on the sparsity of the hidden representations of the dense network. We have implemented this over synthetic data and the MNIST dataset, Where it showed significant improvement over the i-SpaSP and better efficiency by pruning the network.

Keywords: Pruning, Greedy Selection, Iterative Thresholding, Sparse Signal Recovery

Source code: `https://github.com/strankid/514Project`

1. Introduction

Background: Neural network pruning has gained a lot of traction due to the appearance of neural networks with billions of parameters, increasing the computational and memory overhead. Several pruning methods are found to perform a sub-network search within the dense network, fishing out the high-performing network. The main ingredient to all these techniques lies in the "importance" criterion of the neural networks, defining the most optimal sparse sub-network that can be taken from the dense set.

[1]Department of Computer Science, Rice University, Houston, Texas. Correspondence to: Apoorv Walia <apoorv.walia@rice.edu>, Hemanth Kumar Jayakumar <Hemanth.Kumar.Jayakumar@rice.edu>, Koride Sumukh Bharadwaj <sk160@rice.edu>.

These Empirical successes led to approaches focusing on a strong theoretical background for pruning modeling. This work follows a sub-work within this field, greedy forward selection (GFS) (Ye et al., 2020) which is followed by the Frank-Wolfe algorithm. In an attempt to beat i-SpaSP, this work envelopes the theoretical assurances that i-SpaSP and CoSaMP while introducing newer empirical results following the conventions of GFS and bringing forth the convergence guarantees of gradient descent.

This Work: Starting with the model, we apply iterative, Sparse Structured Pruning(i-SpaSP) to prune the network by calculating the importance and the active neurons. Following this pruning, we try to apply a Gradient Descent on the pruned weights as we prune to obtain an optimal pruned network which we sparsify to obtain the final sparse set. This takes reference from the i-SpaSP paper(Wolfe & Kyrillidis, 2021) and adds additional steps to make it more finer and better. Theoretically, We demonstrate that, for two and multi-layer networks, the output residual between dense and pruned networks decays polynomially with respect to the size of the pruned network and that, in addition, the order of this polynomial rises as the hidden representations of the dense network increase and become more sparse. In experiments, We show that our work is capable of getting high-performing networks, to run across various models(two-layer networks) and MNIST datasets. Our work is easier to implement and shows improved runtime, thus promising results compared to i-SpaSP.

2. Preliminaries

Notation Lowercase characters are used to indicate vectors and scalars. Uppercase letters are used to denote matrices and some constants. Sets are represented by calligraphic uppercase letters, such as G, with set complements Gc. We represent [n] as [0, 1,..., n]. For $x \in \mathbb{R}^N$, $\parallel x \parallel_p$ is the l_p vector norm. The largest s valued component of x is x_s , where $\mid x \mid_s \geq s$. Support of x is given as supp(x). Given non-zeroes at the indices of G it is represented as $x \mid_G$, for the index set G. The Transpose of X and Frobenius of X are represented as X^T and $\parallel X \parallel_F$. For given X, $X_{i,:}, and X_{:j},$ are the i-th and j-th row of X. The row and column submatrices for index set G are represented as $X_{G,:} and X_{:G},$ respectively has rows and columns with indices of G.$\mu(X)$:

$\mathbb{R}^{m\times n} \rightarrow \mathbb{R}$ sums over the columns of the matrix (i.e., $\mu(X) = \sum_i X_{:,i}$) while vec(X): $\mathbb{R}^{m\times n} \rightarrow \mathbb{R}6mn$ stacks columns of a matrix. The row support of X is returned by rsupp(X). X $\in \mathbb{R}^{m\times n}$ is p-row compressible with magnitude R $\in \mathbb{R}^{\geq 0}$ if $\mid \mu(X) \mid_{(i)} \leq \frac{R}{i^{\frac{1}{p}}}, \forall i \in [m]$, where $\mid \cdot \mid_{(i)}$ denotes the i the sorted component. Lower p values indicate a nearly row sparse matrix and vice versa

Network Architecture

We are considering a primary two-layer network
U = $\varphi(W^{(0)}, W^{(1)}) = W^{(1)}\dot{\sigma}(W^{(0)}\dot{X})$
X $\in \mathbb{R}^{d_{in}\times B}$ denotes a matrix of input data to the neural network. The input matrix X stores all training where B is the cardinality of the dataset. $\sigma(\cdot)$ denotes the ReLU activation function. The two weight matrices of the neural network are $W^{(0)} \in \mathbb{R}^{d_{hid}\times d_{in}}$and $W^{(1)} \in \mathbb{R}^{d_{out}\times d_{hid}}$. The type of pruning we consider is that of selecting neurons; this corresponds ot sub selecting columns of $W^{(1)}$ (equivalently, we subselect the rows of $W^{(0}$ using the same index set).

3. Related Work

Greedy Selection: Combinatorial optimization issues' approximate solutions are successfully found using greedy selection (Frank & Wolfe, 1956). There are numerous frameworks and methods available for greedy selection, including Frank-Wolfe (Frank & Wolfe, 1956), CoSAMP (Needell & Tropp, 2008), and iterative hard thresholding (Khanna & Kyrillidis, 2018). A link between greedy selection and deep learning has been established thanks to the employment of Frank-Wolfe in GFS and while training these networks (Pokutta et al., 2020). In order to make our suggested methodology, we use CoSAMP (Needell & Tropp, 2008), which strengthens the relationship.

Pruning: Neural network pruning algorithms can be loosely divided into unstructured and structured variations. As examined in this study, structured pruning prunes parameter groupings rather than individual weights, enable speedups to be realized without sparse computing. According to empirical heuristics for structured pruning, parameter groups with low l1 norms should be removed ; the gradient-based sensitivity of parameter groups should be measured ; and network output should be preserved .

As seen from Lottery Ticket Hypothesis (Frankle & Carbin, 2018) and RigL(Evci et al., 2020) provided backgrounds for pruning at initialization but lacked the computational synchronicity and theoretical assurances to back up pruning before training, showing little to no empirical difference in practical scenarios over large models as seen in current trends. This led to resorting towards training after pruning to improve inference speeds and memory while maintaining the accuracy of the subnetwork chosen. Pre-training is usually the most costly phase (Lee et al., 2020). Pruning typically follows a three-part process, which also includes pre-training, pruning, and fine-tuning (Liu et al., 2018). The finetuning step assures the maximum capacity of the pruned model in comparison with the actual dense model as a practical assurance.

Gradient Descent: One of the most widely used algorithms for optimization and by far the most popular method for optimizing neural networks is gradient descent (Ruder, 2016). There are many ways of implementing the gradient descent (Lee et al., 2020). As the pruning step proceeds iteratively, gradient descent is applied on the then sparse network to obtain efficient weights responsible for better subnetworks. This would be followed by projecting onto a s-sparse plane to ensure the sparsity of the end subnetwork.

Iterative Thresholding: Iterative thresholding is an optimization method that can be used to solve certain types of problems involving the minimization of a function with L1 regularization. It works by iteratively applying a thresholding operator to the input, which has the effect of "shrinking" the input towards zero. This can help to promote sparsity in the solution, which can be useful in some applications. In each iteration, the thresholding operator is applied to the current estimate of the solution, and the resulting value is used as the input for the next iteration. This process continues until convergence is reached or a predetermined number of iterations is reached (Wright, 2010) .Typically done by minimizing a cost function that measures the difference between the original signal and the reconstructed signal, subject to a sparsity constraint that encourages the solution to have as few non-zero coefficients as possible.Iterative hard thresholding (IHT) is a variant of the iterative thresholding algorithm that uses a hard thresholding operator instead of a soft thresholding operator. It improves near optimal point and suceeds with minimum number of observations (Blumensath & Davies, 2008). Acceleration of IHT can significantly improve compared to gradient descent and frank-wolfe (Khanna & Kyrillidis, 2018).

4. Motivation

This work is inspired by the structured version of Greedy forward selection search of i-SpaSP for pruning neural networks post-training. The stability of iterative pruning provides space for adding improvements steps to the algorithm to ensure a better model from a theoretical perspective leading to a practical scenario. The theoretical assurances backed by i-SpaSP and CoSaMP along with gradient descent provide great theoretical confidence in this approach. The end-goal of this work is to provide a (relatively) efficient algorithm to prune a network to various sizes achieves an optimal subnetwork from the dense neural network while maintaining convergence guarantees as well as time complexities.

5. Methodology

The work of this paper is formulated to be a two-layer network in Algorithm 1, where the pruned network size is s and total iterations is T. S stored active neurons in the pruned model. which is refined over iterations.

It goes through a 5-step process for each iteration of our work unlike 3 step process in i-SpaSP.

5.1. Algorithms

Step I: Compute gradient with respect to $W_t \in \mathbb{R}^{d_{out} \times d_{hid}}$.

Step II: Identify s columns within the gradient wrt W_t outisde set S_t and Union with S_t.

Step III: Update step using the W_t by gradient descent on the f and focus on D_t where $\mid D_t \mid \leq 2s$

Step IV: Truncating Q_t to be s-row sparse

Step V: Improving the W_{t+1} for additional T_1 iterations by gradient descent on S_{t+1}.

We consider H = $\sigma(W^0 X)$ a fixed matrix of size $\mathbb{R}^{D_{hid} \times B}$, we sub select neurons such that we are allowed to train the weights on the select column of $W^1_{:,s}$

Compute gradient wrt W_t : $\pi_s(Y)$ keeps the s largest in the l_2 norm columns of Y. The gradient of f(W) is given by $\nabla f(W_t) = (W_t \dot{H} - U)\dot{H}^T \in \mathbb{R}^{d_{out} \times d_{hid}}$. This defines the importance of neurons within the hidden layer.

Find best s columns of $\nabla f(W_t)$ outside set S_t and unite with S_t : We abuse the gradient notation and use $\nabla_s f(W)$ to denote that we focus on the column indexed by S in $\nabla f(W)$. We denote $\nabla_{S^c} f(W)$ as the matrix construction that contains the columns living in the complement of set S.

Update $W_t$by gradient descent on f and focusing on D_t : It represents an update step where we perform gradient descent on the pruned model updating its weights. This is followed by a sparsification step as gradient descent can increase the rankness of the weights. It is an approximation step for the closed-form solution of the problem for improved computational efficiency.

f(W,S) = $\frac{1}{2} \parallel W_{:,s}\dot{H}_{s,:} - U \parallel_F^2$ which is

$W^*_{:,s} = U \cdot H \cdot H^{\dagger}_{s,:}$,

The pseudo inverse norm of H matrix indexed by S is represented as $H^{\dagger}_{s,:}$. We perform a gradient step indexed by S on W. this gets us closer to the closed form solution. This step can also be solved and represented as

$Q_t = U \cdot H \cdot H^{\dagger}_{D_{t},:}$

Algorithm 1 Variants of

```
Parameters: T := total iterations; s:=Pruned size
$S_t := \Phi; t := 0$
$W_{-1} = W^{(1)}$

# Compute hidden representation
$H = \sigma(W^{(0)}\dot{X})$
$h = (H)$

# Compute dense network output
$U = W^{(1)}\dot{H}$

while $t < T$ do
    t=t+1

    # Step I : Compute gradient wrt $W_t$
    $\nabla f(W_t) = (W_t\dot{H} - U)\dot{H}^T \in \mathbb{R}^{d_{out} \times d_{hid}}$

    # Step II : Find best s columns of $\nabla f(W_t)$ outside set $S_t$ and unite with $S_t$
    Compute $\pi_s(\nabla_{S_t^c} f(W_t)))$,
    $D_t = S_t \bigcup$ supp($\pi_s(\nabla_{S_t^c} f(W_t))$), where $\mid D_t \mid \leq 2s$

    # Step III : Update $W_t$by gradient descent on f and focusing on $D_t$
    $Q_t = W_t - \eta \nabla_{D_t} f(W_t)$

    # Step IV : Truncate $Q_t$ to be s-row sparse
    $W_{t+1} = \pi_s(Q_t)$ with $S_{t+1} = supp(W_{t+1})$

    #Step V: Improve $W_{t+1}$ for additional $T_1$ iterations by gradient descent on $S_{t+1}$
    for $i = 1$ to $m - 1$ do
        $W_{t+1} \leftarrow W_{t+1} - \eta \nabla_{S_{t+1}} f(W + t + 1)$
    end for
end while

#return pruned model that only includes neurons in S
return $W^* = \{W^0_{S,:}, W^{(1)}_{:,S}\}$
```

Truncate Q_t to be s-row sparse : From the previous step, the Q_t is made s-row sparse, updating the W_{t+1}. It helps us get an optimal solution for the given fixed index sets. It requires computing the pseudo-inverse of $H_{S,:}$. The η has to be chosen for the gradient descent step.

$$\eta = argmin_{\eta} \frac{1}{2} \parallel (W_t - \eta \nabla_{D_t} f(W_t)) \cdot H - U \parallel_F^2$$

Improve W_{t+1} for additional T_1 iterations by gradient descent on S_{t+1} : Similiar to the above step it is further improved by solving the 1-dimensional problem. Through the loop for 1 to m-1 where it keeps updating with the equation

$$W_{t+1} \leftarrow W_{t+1} - \eta \nabla_{S_{t+1}} f(W + t + 1)$$

with gradient descent of $\nabla_{S_{t+1}} f(W + t + 1)$ while column indexed over set S_{t+1}

5.2. Implementation details

Gradient Calculation for sparsification We use PyTorch modules for a faster residual gradient calculation, defining the choice of 2s sparse vectors to be included within D_t. Due to the large dimensional nature of the dataset used to calculate the gradient, we abuse the GPU factors of PyTorch for a faster calculation of the gradient.

Gradient Descent step PyTorch standard training loop with a custom loss factor is defined to train the temporary-pruned_model's weights as defined before. This allows for a faster calculation with a high boost to performance due to parallelization and optimized modular usage.

Large Datasets When leaning toward larger datasets, a mini-batch approach can be taken for both the gradient calculation as well as the descent step, separately batched to efficiently update the sparse set and the weights. This negates the need for higher computational requirements as we are performing a matrix multiplication step which is rank-heavy.

5.3. Experiments

We define the experimental settings and the corresponding results to showcase the potential of this approach with rigorous empirical analysis providing practical knowledge of the performance of the model

5.3.1. Experiment Settings

We used Pytorch version "1.12.1+cu113", Google Colab, Kaggle notebooks, Wandb, Rice computation resources (servers) for the implementation of the project. The model configuration is a 2-layer neural network, each layer configuration is as follows: Fully connected layer followed by an activation function - relu for all the experiments. The number of neurons for the hidden layer is 10000 for the

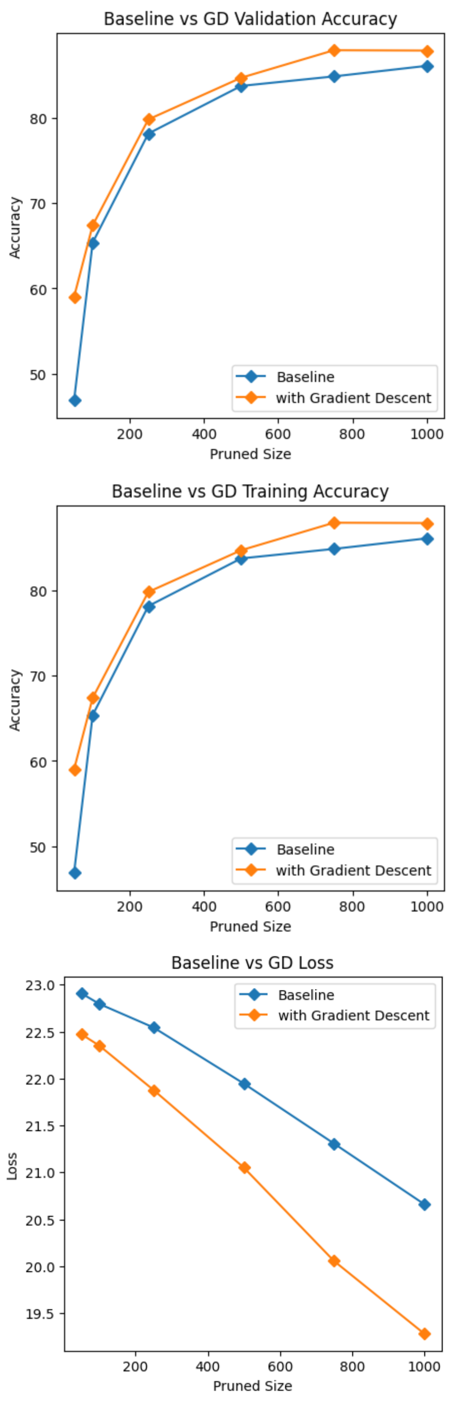

Figure 1. Model Statistics before fine-tuning

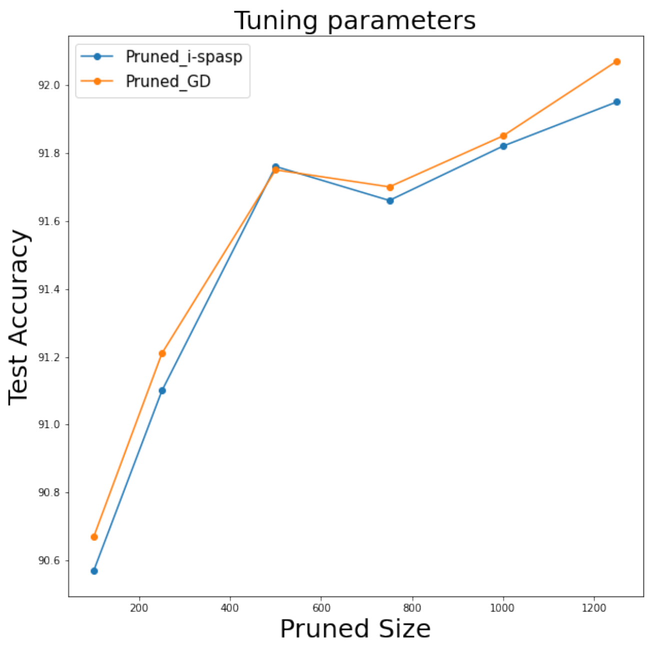

Figure 2. Test Accuracy after fine-tuning

dense model. The input and output layer dimensions depend on the dataset configuration. We used adam optimizer to minimize the losses and used cross-entropy loss as the the objective function to minimize. A momentum of 0.9 and 0.99 as well as time-proportional decay is set to ensure a better learning curve as the model slows down the learning process as it reaches the convergence point.

The datasets used for the experiments are:

1. Synthetically generated dataset
2. MNIST database (Modified National Institute of Standards and Technology database)

5.3.2. Synthetically generated dataset

We create synthetic H matrices with different row-compressibility ratios p. For each p in the set {0.3, 0.5, 0.7, 0.9}, we randomly generate three unique entries in the matrix and calculate the average performance across them. We then prune a randomly-initialized W matrix of size $d_{hid} \times d_{out}$ to different sizes s by d_{out} for all s $\in [d_{hid}]$ using T=20. Each value of s is a separate experiment, and we test two values of $d_{hid} \in \{100 \text{ and } 200\}$ and the same $W_{(1)}$ matrix is used for testing and experiments using the hidden dimensions.

We began by working with a synthetically generated data set that was designed to be compressible. Adding the learning step while working with this data set did not lead to any significant improvements. From our understanding, this does not discredit our approach. Since the synthetic data was generated to be compressible, pruning without any learning step (iSpaSP algorithm) is already quite effective and the learning step cannot really provide any further useful information to the model. However, we did notice an increase in the rate of convergence. The results of this experiment can be seen in Figure 3.

5.3.3. MNIST dataset

The basic MNIST dataset consists of handwritten images, which are flattened to be fed into a 2 layer feedforward neural network for out experiments. The images are normalized to [0, 1] and no other transformations or augmentations are performed on them.

Following this began the experiments on the MNIST dataset. We ran experiments to tune various hyperparameters such as learning rate, gradient descent epochs, and pruning iterations to observe the behavior of adding the learning step to the pruning process. In all conditions, we notice that adding the learning step is useful in the pruning process. With just 5 epochs of learning in each pruning iteration, we notice a substantial improvement in the training and testing accuracy of the models as well as lower loss. The results are presented in Figure 1. The link to the result of the full set of experiments can be found in the appendix.

All of the results discussed so far occur before fine-tuning the model. Adding the learning step is already useful before we fine-tune it. As our experiments show, the advantage gained by adding the learning step carries on even after fine-tuning. The model trained with the additional learning step is able to outperform the baseline model after fine-tuning. Our fine-tuning step is essentially training the pruned model further on the training data. As shown in Figure 2, there is a slight performance increase by the model which underwent training while pruning, showing that the subnetwork chosen by our work outperforms the subnetwork chosen from the i-SpaSP algorithm.

A thing to note here that merits discussion is the accuracy of the dense model itself. Due to computational constraints, the dense model was trained only for a few iterations and achieves an accuracy of 92% on the validation set. This elevates the result of the pruned models, as they come extremely close to the performance of the dense model with a significantly smaller set of parameters. The absence of regularization layers also contributes to this lower validation accuracy as the loss decreases upon training while the

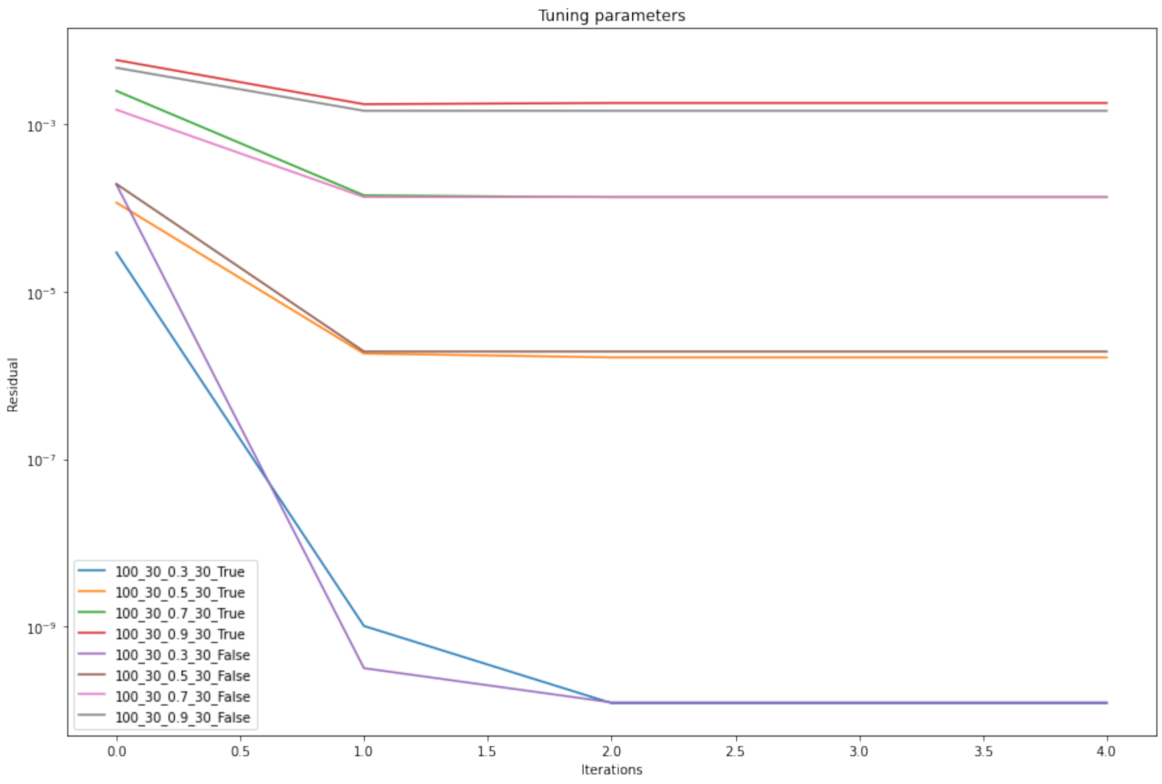

Figure 3. Loss curve on Synthetic data

accuracy oscillates near 92%.

6. Conclusions

We show that adding the learning step modification to the original iSpaSP algorithm can improve the performance of the model. There are a few caveats to consider, however. Adding these learning steps is not free. We substantially increase our per iteration cost as well as the wall-clock time of the pruning process. This increase in time complexity rewards us with a better model and weight parameters. This project serves as a proof of concept and provides validation that this idea can be further explored.

7. Future Work

1. Extending this proof of concept to larger, deeper, and more complex models such as Resnet, Transformers, etc.

2. Theoretical assurances are a necessity to overcome the results of i-SpaSP and to provide strong backgrounds towards contribution to this field.

3. Considering simultaneous updates on all the layer weights in each iteration

4. Identifying optimal adaptive learning rate via line search of the 1-dimensional optimization problem for a better gradient step.

5. Attempting abusing hardware capabilities to implement a direct pseudo-inverse calculated solution to the problem instead of approximate gradient descent.

6. Experimenting sparsification during initialization with this technique to drastically improve training time.

Acknowledgements

We would like to thank Cameron Wolfe for his support and guidance throughout the course of the project.

References

Blumensath, T. and Davies, M. E. Iterative Thresholding for Sparse Approximations. Technical Report 5, December 2008.

Evci, U., Gale, T., Menick, J., Castro, P. S., and Elsen, E. Rigging the Lottery: Making All Tickets Winners. In *International Conference on Machine Learning*, pp. 2943–2952. PMLR, November 2020. URL https://proceedings.mlr.press/v119/evci20a.html.

Frank, M. and Wolfe, P. An algorithm for quadratic programming. *Naval Research Logistics*, 3(1-2):95–110, March 1956. ISSN 0028-1441. doi: 10.1002/nav.3800030109.

Frankle, J. and Carbin, M. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. *arXiv*, March 2018. doi: 10.48550/arXiv.1803.03635.

Khanna, R. and Kyrillidis, A. IHT dies hard: Provable accelerated Iterative Hard Thresholding. In *International Conference on Artificial Intelligence and Statistics*, pp. 188–198. PMLR, March 2018. URL https://proceedings.mlr.press/v84/khanna18a.html.

Lee, C.-H., Fedorov, I., Rao, B. D., and Garudadri, H. SSGD: Sparsity-Promoting Stochastic Gradient Descent Algorithm for Unbiased Dnn Pruning. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5410–5414. IEEE, May 2020. doi: 10.1109/ICASSP40776.2020.9054436.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the Value of Network Pruning. *arXiv*, October 2018. doi: 10.48550/arXiv.1810.05270.

Needell, D. and Tropp, J. A. CoSaMP: Iterative signal recovery from incomplete and inaccurate samples. *arXiv*, March 2008. doi: 10.48550/arXiv.0803.2392.

Pokutta, S., Spiegel, C., and Zimmer, M. Deep Neural Network Training with Frank-Wolfe. *arXiv*, October 2020. doi: 10.48550/arXiv.2010.07243.

Ruder, S. An overview of gradient descent optimization algorithms. *arXiv*, September 2016. doi: 10.48550/arXiv.1609.04747.

Wolfe, C. R. and Kyrillidis, A. i-SpaSP: Structured Neural Pruning via Sparse Signal Recovery. *arXiv*, December 2021. doi: 10.48550/arXiv.2112.04905.

Wright, S. J. Iterative thresholding for sparse approximations. In *Journal of Fixed Point Theory and Applications (2009)*, 2010.

Ye, M., Gong, C., Nie, L., Zhou, D., Klivans, A., and Liu, Q. Good Subnetworks Provably Exist: Pruning via Greedy Forward Selection. In *International Conference on Machine Learning*, pp. 10820–10830. PMLR, November 2020. URL https://proceedings.mlr.press/v119/ye20b.html.

A. Link to Full Experiment Set

https://wandb.ai/opt-prune/Pruning-2-layer-experiments-Final/reports/Training-while-pruning–VmlldzozMTQwMzMw?accessToken=v3nt627rjx5r8dfnq6m2xm3dj7sjt2f9h0v6pgjlozcehyccvit1djxsw3oyyvg0

https://wandb.ai/opt-prune/Pruning-2-layer-experiments-Baseline/reports/Pruning-Baseline—VmlldzozMTQwMzY1?accessToken=ryvnh45pjhwehinq70wveafow9b1f2a80norp7jdfkejhsghvem5xzx5cjkic8c9

https://www.kaggle.com/code/hemanthkj/mlp-expts/notebook