Group 15: Bayes Optimal Neural Network Pruning using Relational Graph Properties

Andrei Rekesh
drei@g.ucla.edu
UID: 105343158

Sahil Bansal
sahilbansal@g.ucla.edu
UID: 905525442

Vaibhav Kumar
vaibhavk@cs.ucla.edu
UID: 305710616

Vivek Arora
vivekarora10@g.ucla.edu
UID: 505526269

ABSTRACT

Graph representations of neural networks are commonly used to assist in training-time computations like backpropagation and weight updates. However, recent works [14] have proposed a novel, relational graph representation allowing network depth to correspond to rounds of message passing. The correlation between neural network performance and two key properties of these relational graphs, average shortest path length (ASPL) and clustering coefficient, has been previously investigated by randomly generating graphs spanning a wide range of both properties and exhaustively evaluating performance. We extend the analysis to five graph properties: average degree centrality, eigenvector centrality, clustering coefficient, edge connectivity, and number of cycles. In our work we propose **B**ayesian **o**ptimized **G**raph Based **P**runing **(BoGP)** that utilized these properties to learn optimal pruning masks for deep neural networks. We find that these properties are important to finding high-performing candidate neural network pruning masks, and such masks can be identified efficiently using Bayesian Optimization.

ACM Reference Format:
Andrei Rekesh, Sahil Bansal, Vaibhav Kumar, and Vivek Arora. 2018. Group 15: Bayes Optimal Neural Network Pruning using Relational Graph Properties. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Neural networks are comprised of layers of interconnected neurons through which data passes. These neurons and connections are often represented as nodes and edges of computational graphs describing the propagation of data through the network. As the performance of neural networks has been shown to depend on their architecture, there have been recent research efforts to determine the relationship between known graph properties of these representations and model performance in an effort to develop effective graph-based pruning methods [14]. However, You et al. cite two shortcomings of computational graph representations in determining this relationship: lack of generality and dissimilarity to biological models. Instead, they develop relational graphs, representing one input and output neuron as the same node with an edge to itself. For fixed width, relational graphs are depth-invariant, meaning that neural networks with the same hidden layer width and different depths correspond to the same relational graphs. Neural network depth instead corresponds to the number of rounds of message passing between the nodes. Our work aims to evaluate the strength of relational graph properties for pruning neural networks. We accomplish this with pruning masks, two-dimensional matrices of zeroes and ones placed over a network to deactivate specific connections between neurons. Each mask can be converted to a relational graph, and as such, its graph properties can be observed. Specifically, we choose average degree centrality, eigenvector centrality clustering coefficient, edge connectivity, and number of cycles. As it is implausible to use brute force to find the optimal values for a high-dimensional feature space, one might think to try gradient-based methods as an alternative. However, backpropagating the model with respect to the values of relational graph properties of the neural network is similarly implausible. For such problems, gradient-free algorithms such as Bayesian optimization are often employed [3]. This, we design a method to implement Bayesian optimization over the values of our five properties for model performance as follows:

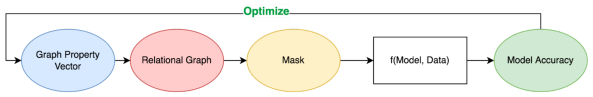


Figure 1: Proposed pipeline for optimizing over relational graph properties of pruning masks rather than over the mask directly.

(1) We generate 1000 random 0-1 masks with a given sparsity, convert them to relational graphs, and store each mask and a vector describing its properties in a database.
(2) We initialize Bayesian optimization to random values for all 5 properties.
(3) We measure performance for each iteration by determining the nearest mask (by cosine similarity of its property vector)

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

and evaluating a neural network masked by it on dataset of choice.

(4) The optimizer determines the next candidate set of property values.
(5) Repeat steps 3-4 for desired number of iterations.

Through our experiments, we show that:

- Relational graph properties are effective indicators of neural network performance. Even for a fixed sparsity, accuracy varies highly as property vectors vary.
- Bayesian optimization is capable of determining optimal masks using these properties.
- Our method can either take the required sparsity as input or adaptively learn the trade-off between pruning and model performance.

2 BACKGROUND

2.1 Relational Graphs

The most common way to represent neural networks as graphs is the computational graph. But due to its limitations, we use relational graphs to represent neural networks in our problem formulation. Relational Graphs were introduced in [14]. In the most simple example of a fixed- width fully-connected layer, we represent one input channel and one output channel together as a single node, and an edge in the relational graph represents the message exchange between the two nodes. Under this formulation, the relational graphs can represent many types of neural network layers while getting rid of the constraints of computational graphs such as directed, acyclic, bipartite, single-in-single-out. Figure 2 shows the relational graph representation for a layer of fixed-width 5 layer MLP.

In general, every layer is represented by a graph, where connections

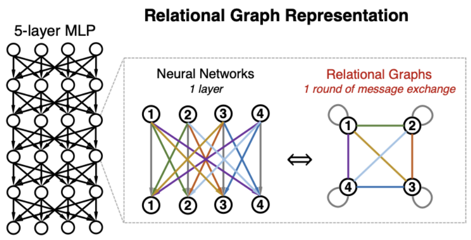
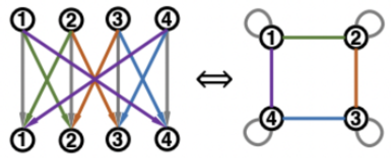


Figure 2: Relational Graph Representation for fixed-width 5 layer MLP.

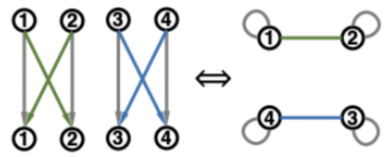
with the next layer are represented using the edges. Further, each layer computation is represented by a round of message passing in this representation. Figure 3 shows the relational graph representation for some more examples of neural network layer formulations.

2.2 Relational Graphs to Neural Networks

One neural network layer in the relational graph corresponds to one round of message exchange and to obtain the whole network, we perform this message exchange over the same graph multiple times. Some of the important parameters that help in obtaining the neural network corresponding to a relational graph are the node feature



(a) Example 1



(b) Example 2

Figure 3: Example relational graph representations for some more neural network formulations.

vectors, message functions, aggregation function for various nodes and number of rounds of message exchange. Having information of all these is sufficient to obtain the neural network corresponding to a relational graph. Figure 4 shows the neural network formulation for a 4-node relational graph with specific function values.

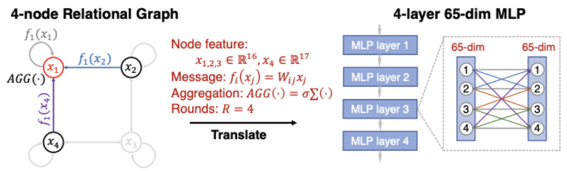


Figure 4: Neural Network formulation for a 4-node Relational Graph.

2.3 Graph Properties

Following graph properties have been referred to or utilized for the purpose of this project. For all the considered graph properties, their related function implementations in the networkx library have been referred.

(1) Clustering Coefficient : Measure of the degree to which nodes in a graph tend to cluster together. Usually calculated as proportion of edges between the nodes within a given node's neighborhood, divided by the number of edges that could possibly exist between them, averaged over all the nodes.
(2) Average Path Length: Measure of average shortest path distance between any pair of nodes in the relational graph.
(3) Average Degree Centrality: Fraction of nodes a particular node is connected to, averaged over all the nodes in the graph.
(4) Average Eigenvector Centrality: Average of eigenvector centrality computed for all the nodes in the graph. For a particular node, eigenvector centrality is based on the centrality of its neighbors. More specifically, it is computed as follows: The eigenvector centrality for node i is the i-th element of

the vector defined by the equation $Ax = \lambda x$ where A is the adjacency matrix of the graph G with eigenvalue λ.

(5) Edge Connectivity: Measure of the minimum number of edges that must be removed to disconnect the relational graph or render it trivial.

(6) Number of Cycles: Measure of length of the cycle if it exists in the graph otherwise 0.

2.4 Bayesian Optimization

Bayesian Optimization[3] provides a principled technique based on Bayes Theorem to search an input that results in the minimum or maximum cost of a given objective function. Typically, the form of the objective function is complex and intractable to analyze and is often non-convex, nonlinear, high dimension and computationally expensive to evaluate. Below is the step by step working of Bayesian Optimization:

(1) We first choose a surrogate model for modeling the true function f and define its prior.
(2) Given the set of observations (function evaluations), use Bayes rule to obtain the posterior.
(3) Use an acquisition function which is a function of the posterior to decide the next sample point.
(4) Add newly sampled data to the set of observations and go to step 2 till convergence or budget elapses.

3 MOTIVATION

We represent a neural networks as relational graphs and study their performance as a function of various graph properties. We use a 3 hidden layer neural network on MNIST dataset where we generate multiple pruning masks applied to each layer of the network. We plot the neural network performance vs the graph properties and observe that the performance is a monotonic function of multiple properties.

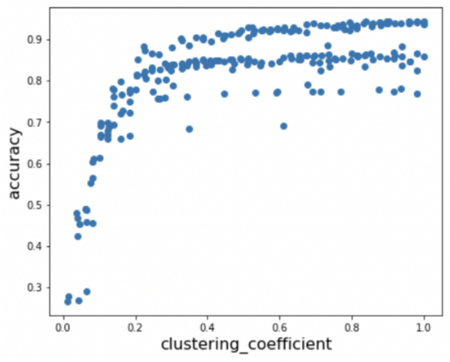


Figure 5: Neural Network Performance v/s Clustering coefficient for a 5-node Relational Graph Set for the MNIST dataset.

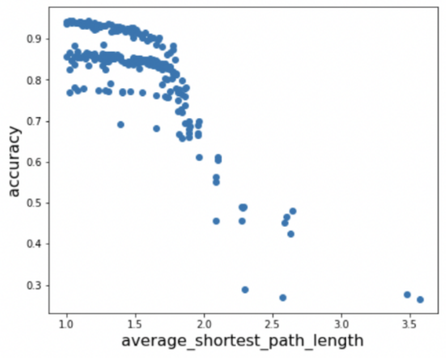


Figure 6: Neural Network Performance v/s Average Shortest Path Length for a 5-node Relational Graph Set for the MNIST dataset.

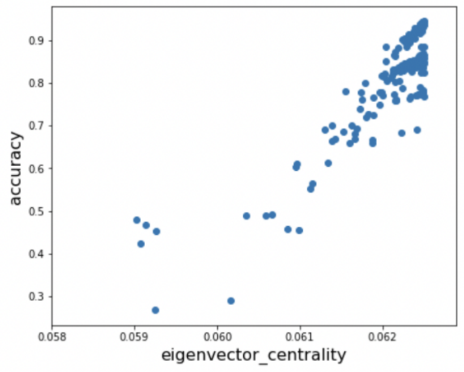


Figure 7: Neural Network Performance v/s Eigenvector Centrality for a 5-node Relational Graph Set for the MNIST dataset.

4 PROBLEM FORMULATION

Given neural network N, with l layers, we want to learn pruning masks ψ for each layer ψ_l... so on.

5 PROPOSED METHOD

5.1 Creating Graph Database

Given a graph property vector $\overrightarrow{V}$, we want to create a graph $\mathcal{G}$ that satisfies this property vector and then create the pruning mask ψ. For any arbitrary $\overrightarrow{V}$, it is not possible to create a corresponding graph. To overcome this problem, we create a database of various graphs and their corresponding feature vector. We then use this graph database to fetch graphs using their property vectors using some similarity metric (more on this in Section 5.2).

Given a set of graph feature functions $\mathcal{F}$, a sparsity level s, we can generate a graph database of size N using Algorithm 1. Each graph feature function $f \in \mathcal{F} : f(\mathcal{G}) \rightarrow \mathbb{R}$ returns a real number graph property for a graph $\mathcal{G}$. To create the set of graph feature functions

Algorithm 1 Generating Graph DB Ψ_s

```
        Inputs, $\mathcal{F}$, $N$
        Output Graph Database $\Psi$
1: procedure Make DB
2:     i ← 0
3:     $\Psi_s \leftarrow \overrightarrow{0}$
4:     for i<$N$ do
5:         Generate a random mask $\psi_s$ of sparsity s
6:         $\overrightarrow{V} \leftarrow \overrightarrow{0}$
7:         $\mathcal{G} \leftarrow Mask\_to\_graph(\psi)$
8:         for f in $\mathcal{F}$ do
9:             $\overrightarrow{V}$.append(f($\mathcal{G}$))
10:        $\Psi_s$.append($\mathcal{G}$, $\overrightarrow{V}$)
```

$\mathcal{F}$ we compute several graph properties from the networkx library and find their correlation to only keep the uncorrelated features. Figure 8 shows the correlation between 9 such graph feature functions. After removing the highly correlated features, we keep the following five features for our analysis:

(1) Average Degree Centrality
(2) Average Clustering Coefficient
(3) Edge Connectivity
(4) Eigenvector Centrality
(5) Number of Cycles

More details about these graph property functions can be found in networkx python documentation [4].

5.2 Fetching Graphs

Given a graph property vector $\overrightarrow{V}$, we want to find the corresponding mask $\mathcal{G}$ such that for the vector $\overrightarrow{U} = f(\mathcal{G})$, $\Omega(\overrightarrow{V}, \overrightarrow{U})$ is minimal where Ω is a similarity metric.

Algorithm 2 Fetching Graph from Property Vector

```
        Inputs, $\Psi_s$, $\overrightarrow{V}$
        Output Optimal Graph $\mathcal{G}^*$
1: procedure Fetch DB
2:     $\mathcal{G}^* \leftarrow NULL$
3:     $\omega^* \leftarrow 0$
4:     for $\mathcal{G}$, $\overrightarrow{U}$ ∈$\Psi_s$ do
5:         if $\Omega(\overrightarrow{U}, \overrightarrow{V}) > \omega^*$ then
6:             $\omega^* \leftarrow \Omega(\overrightarrow{U}, \overrightarrow{V}$
7:             $\mathcal{G}^* \leftarrow \mathcal{G}$
8:
9:
10:
```

For our experiments, we set the similarity metric Ω to be cosine similarity. We experimented with L1 and L2 distances but found cosine similarity to be the best (in terms of model performance over validation set).

5.3 Bayesian Objective

We devise the problem of learning the optimal pruning mask as a search problem. To solve the task, we use Bayesian Optimization. Our method can learn the pruning mask for a given sparsity level (Algorithm 3) or learn the sparsity as a trade-off to performance using a trade-off weight α (Algorithm: 4. Both Algorithm 3 and Algorithm 4 are the block-box functions we maximize using the Bayesian Optimization framework as described in Section 2.4.

Algorithm 3 Blackbox Function f

```
        Input Graph feature vector $\overrightarrow{V}$ and sparsity s
        Output Model Performance $\rtimes_{\mathcal{D}_v}$
1: procedure f
2:     Fetch $\psi$ from $\Psi_s$ using some similarity metric over $\overrightarrow{V}$
3:     Apply $\psi$ to $\mathcal{M}_\theta$ to get $\mathcal{M}_\theta^\psi$
4:     Train $\mathcal{M}_\theta^\psi$ on $\mathcal{D}_t$
5:     $\rtimes_{\mathcal{D}v} \leftarrow O(\mathcal{M}_\theta^\psi, \mathcal{D}_v)$
   =0
```

Given model $\mathcal{M}_\theta$ with the hyperparameter set θ, objective function O and data $\mathcal{D}$, we aim to learn the most optimal mask ψ^* with given sparsity s such that the performance, $\rtimes_{\mathcal{D}v}$ of the model $\mathcal{M}$ is maximized over the validation data $\mathcal{D}_v$ after training on $\mathcal{D}_t$.

Algorithm 4 Blackbox Function f$_2$

```
        Input Graph feature vector $\overrightarrow{V}$ and weight $\alpha$
        Output Weighted performance y
1: procedure f2
2:     Fetch $\psi$ from $\Psi$ using some similarity metric over $\overrightarrow{V}$
3:     Apply $\psi$ to $\mathcal{M}_\theta$ to get $\mathcal{M}_\theta^\psi$
4:     Train $\mathcal{M}_\theta^\psi$ on $\mathcal{D}_t$
5:     $\rtimes_{\mathcal{D}v} \leftarrow O(\mathcal{M}_\theta^\psi, \mathcal{D}_v)$
6:     $s \leftarrow Sparsity(\mathcal{M}_\theta^\psi)$
7:     $y \leftarrow o_{\mathcal{D}v} + \alpha * s$
```

We can also modify the blackbox function such that we optimize over a linear weighted objective of both model performance and model sparsity. We can control the trade-off using the weight linear interpolation weight α.

6 EXPERIMENTAL RESULTS

6.1 Datasets

We used the MNIST[1] datasets for all the experiments. The MNIST dataset consists of 28x28 grayscale images of handwritten digits with a training set of 60,000 images, and a validation set of 10,000 images. All performance results are reported as the accuracy on the validation set.

[1]http://yann.lecun.com/exdb/mnist/

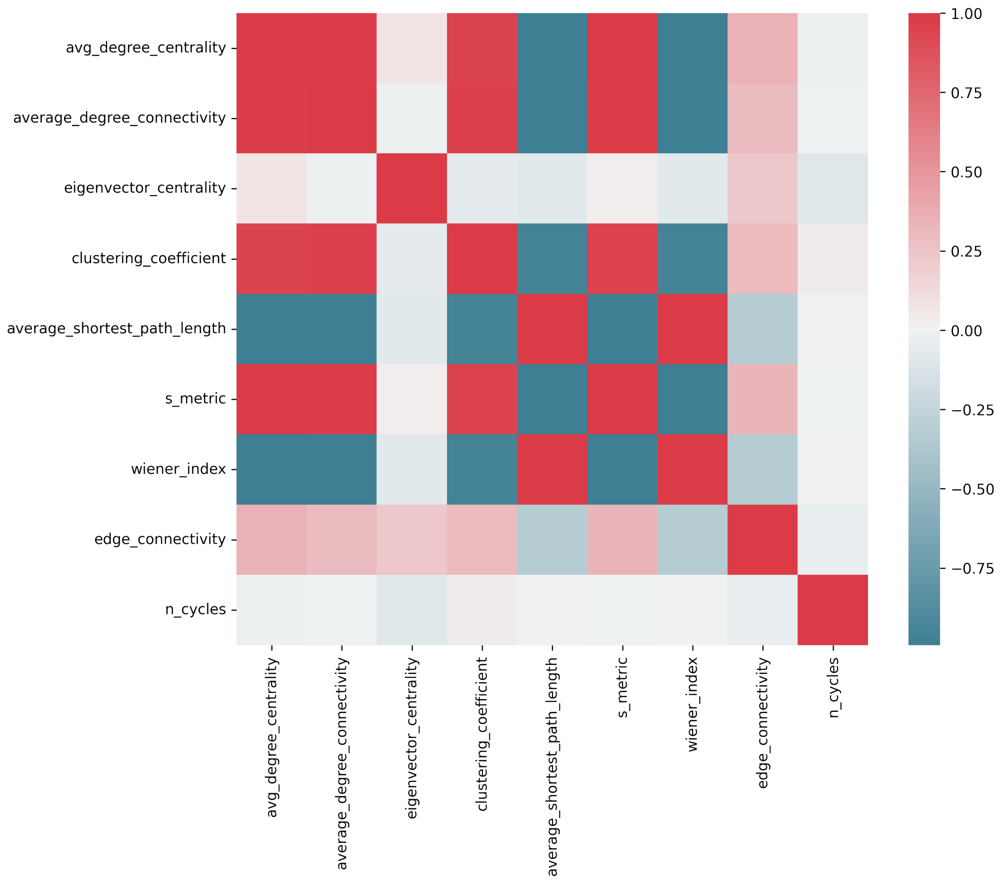


Figure 8: Correlation Between Various Graph Properties

Pruning Technique	Accuracy (s=0.10)	Accuracy (s=0.15)	Accuracy (s=0.30)	Accuracy (s=0.45)	Accuracy (s=0.60)	Accuracy (s=0.75)	Accuracy (s=0.90)
Base Model	0.970	-	-	-	-	-	-
BoGP	-	**0.972**	0.972	0.970	0.968	**0.966**	0.951
Glob$_1$	-	0.971	0.970	0.970	0.967	0.965	0.945
Glob$_5$	-	**0.972**	**0.972**	**0.971**	**0.970**	**0.966**	**0.962**
Unif$_1$	-	0.970	0.970	0.969	0.966	0.963	0.810
Unif$_5$	-	**0.972**	**0.972**	0.971	**0.970**	0.965	0.950

Table 1: Accuracy for Different Pruning Methods over the MNIST Validation Dataset. Unif$_n$ and Glob$_n$ represents the Unif and Glob pruning algorithm with n iterations respectively.

	α=0.1	α=0.15	α=0.25	α=0.5	α=0.75	α=0.9
Accuracy	0.90	0.88	0.82	0.37	0.36	0.25
Sparsity	0.73	0.76	0.79	0.93	0.94	0.98

Table 2: Bayesian Optimization over our linearly interpolated optimization function f_2 for different values of α.

We used these datasets as they are the standard datasets for exploring neural networks and considering the large number of candidate graphs that we would be training, these datasets provided us an efficient way to benchmark our approach.

6.2 Baselines

We compare Bayesian optimization-based pruning to two sparsity selection methods for magnitude-based pruning:

- Global Sparsity Constraint: A global weight magnitude threshold is applied to all layers, pooled, allowing layers to differ in sparsity [11].
- Uniform Layer Sparsity: Each layer is pruned independently to be identically sparse. [16].

We evaluate the results of each algorithm trained for 1 and 5 iterations. As both algorithms perform pruning before training, each iteration consists of a pruning step and a training step.

6.3 Implementation Details

In this section we present the implementation details of our mask database, neural network architecture, the training process and the bayesian optimization:

1. We first create a database of masks for different sparsities ranging from 0.15 to 0.9. For each sparsity value we generate 1000 random masks for our database. These masks are used for pruning the hidden layer in our neural network.
2. Our neural network architecture consists of an input layer followed by a single hidden layer and then the output layer with softmax classifier. The input layer is 784 neurons which is the size of the image, the hidden layer dimension is 128 and the output is 10 neurons which is the number of classes
3. We train our neural network using the whole dataset together instead of batched training. The number of epochs is set to 50.
4. For optimization we use the Adam Optimizer which involves a combination of two gradient descent methodologies : Momentum and Root Mean Square Propagation. We use the learning rate as $3 * 10^4$.
5. In our Bayesian Optimization we use number of iteration as 20 and kappa as 5.

6.4 Results

We compare our model Bayesian Optimized Graph based Pruning (BoGP) with Glob1, Glob5 and Unif1, Unif5. Our model performs similar to these for most of the sparsity values. We achieve the best accuracies for sparsity = 0.15 and sparsity = 0.75. For the other sparsity values our accuracies are comparable. With our experiments, we show that our method is at par with the current state of the art pruning techniques and could perform better than the state of the art by adding further improvements.

7 RELATED WORK

An existing work on pruning the neural networks can be roughly divided into one of these 3 categories.

7.1 Pruning before Training

This kind of pruning is usually applied at the initialization and the mask stays unchanged during the training stage. In [2], Frankle et al. introduced the lottery ticket hypothesis. Motivated by the same, many related works have been done. Lee et al. in [7], Verdenius et al. in [13], and Malach et al. in [9] did work either towards introducing some of the pruning heuristics that work well for neural networks or proving the lottery ticket hypothesis. Although all of these methods apply pruning during the initialization process, the way of obtaining the mask still requires training using specific datasets. Chen et al. in their work [1] introduced another pruning before training approach, but unlike the previous works that concentrate on regrading the importance of weights as the pruning criterion, utilized graph theory as guidance to prune the neural network. Further, no training was required in their process. Our work is inspired from this work and is closest as well to this work in that it utilizes graph theory as guidance. They perform edge switching to obtain different pruned versions. On the other hand, we formulate the problem as a search problem among a database of randomly generated masks with particular graph properties. In [11] by Morcos et al., a global weight magnitude threshold is applied to all layers, pooled, allowing layers to differ in sparsity. In [16] by Zhu et al., Each layer is pruned independently to be identically sparse. We utilize the last 2 works as the baselines for comparison against our proposed approach.

7.2 Pruning during Training

The methods that fall in this category are gradual pruning methods that can be seamlessly incorporated within the training process. Zhu et al. in [16], Zhang et al. in [15], He et al. in [6], and Meng et al. in [10] introduced some of the pruning during training methods that can be incorporated within the training.

7.3 Pruning after Training

In order to get a sparse network, it usually requires three-step. This includes pre-training a dense network, pruning as well as fine-tuning it. Han et al. in [5] prune the unimportant weights and then restored them to increase network capacity, finally fine-tuning the whole dense network. Lin et al. in [8] utilized the artificial bee colony algorithm in the pruning stage to efficiently find optimal pruned structure. Sehwag et al. in [12] regarded pruning as an empirical risk minimization problem with adversarial loss objectives, then used three-stage pruning to simultaneously obtain remarkable benign and robust accuracy.

8 CONCLUSION AND FUTURE WORK

We conclude that graph properties of a neural network can be used to learn optimal pruning masks. In this work, we analyze neural network's graph property vectors to optimally learn pruning masks with minimal computations.

As a part of the future work, we would like to do a theoretical analysis of the relation between neural network performance and their graph properties. We would also want to study the application of our technique for modern attention based model architectures.

REFERENCES

[1] Zhuangzhi Chen, Jingyang Xiang, Yao Lu, and Qi Xuan. 2021. RGP: Neural Network Pruning through Its Regular Graph Structure. *arXiv preprint arXiv:2110.15192* (2021).
[2] Jonathan Frankle and Michael Carbin. 2018. The Lottery Ticket Hypothesis: Training Pruned Neural Networks. *CoRR* abs/1803.03635 (2018). arXiv:1803.03635 http://arxiv.org/abs/1803.03635
[3] Peter I Frazier. 2018. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811* (2018).
[4] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proceedings of the 7th*

Python in Science Conference, Gaël Varoquaux, Travis Vaught, and Jarrod Millman (Eds.). Pasadena, CA USA, 11 – 15.

[5] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Shijian Tang, Erich Elsen, Bryan Catanzaro, John Tran, and William J Dally. 2016. DSD: regularizing deep neural networks with dense-sparse-dense training flow. (2016).

[6] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. 2018. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866* (2018).

[7] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. 2018. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340* (2018).

[8] Mingbao Lin, Rongrong Ji, Yuxin Zhang, Baochang Zhang, Yongjian Wu, and Yonghong Tian. 2020. Channel pruning via automatic structure search. *arXiv preprint arXiv:2001.08565* (2020).

[9] Eran Malach, Gilad Yehudai, Shai Shalev-Shwartz, and Ohad Shamir. 2020. Proving the Lottery Ticket Hypothesis: Pruning is All You Need. *CoRR* abs/2002.00585 (2020). arXiv:2002.00585 https://arxiv.org/abs/2002.00585

[10] Fanxu Meng, Hao Cheng, Ke Li, Huixiang Luo, Xiaowei Guo, Guangming Lu, and Xing Sun. 2020. Pruning Filter in Filter. *CoRR* abs/2009.14410 (2020). arXiv:2009.14410 https://arxiv.org/abs/2009.14410

[11] Ari S. Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. 2019. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. https://doi.org/10.48550/ARXIV.1906.02773

[12] Vikash Sehwag, Shiqi Wang, Prateek Mittal, and Suman Jana. 2020. Hydra: Pruning adversarially robust neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 19655–19666.

[13] Stijn Verdenius, Maarten Stol, and Patrick Forré. 2020. Pruning via Iterative Ranking of Sensitivity Statistics. *CoRR* abs/2006.00896 (2020). arXiv:2006.00896 https://arxiv.org/abs/2006.00896

[14] Jiaxuan You, Jure Leskovec, Kaiming He, and Saining Xie. 2020. Graph structure of neural networks. In *International Conference on Machine Learning*. PMLR, 10881–10891.

[15] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. 2018. A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers. *CoRR* abs/1804.03294 (2018). arXiv:1804.03294 http://arxiv.org/abs/1804.03294

[16] Michael Zhu and Suyog Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. https://doi.org/10.48550/ARXIV.1710.01878