AP: Selective Activation for De-sparsifying Pruned Networks

Anonymous authors
Paper under double-blind review

Abstract

The rectified linear unit (ReLU) is a highly successful activation function in neural networks as it allows networks to easily obtain sparse representations, which reduces overfitting in overparameterized networks. However, in the context of network pruning, we find that the sparsity introduced by ReLU, which we quantify by a term called dynamic dead neuron rate (DNR), is not beneficial for the pruned network. Interestingly, the more the network is pruned, the smaller the dynamic DNR becomes during and after optimization. This motivates us to propose a method to explicitly reduce the dynamic DNR for the pruned network, i.e., de-sparsify the network. We refer to our method as Activate-while-Pruning (AP). We note that AP does not function as a stand-alone method, as it does not evaluate the importance of weights. Instead, it works in tandem with existing pruning methods and aims to improve their performance by selective activation of nodes to reduce the dynamic DNR. We conduct extensive experiments using various popular networks (e.g., ResNet, VGG, DenseNet, MobileNet) via two classical and three state-of-the-art pruning methods. The experimental results on public datasets (e.g., CIFAR-10, CIFAR-100) suggest that AP works well with existing pruning methods and improves the performance by 3% - 4%. For larger scale datasets (e.g., ImageNet) and state-of-the-art networks (e.g., vision transformer), we observe an improvement of 2% - 3% with AP as opposed to without. Lastly, we conduct an ablation study to examine the effectiveness of the components comprising AP.

1 Introduction

The rectified linear unit (ReLU) Glorot et al. (2011), $\sigma(x) = \max\{x, 0\}$, is the most widely used activation function in neural networks (e.g., ResNet He et al. (2016), Transformer Vaswani et al. (2017)). The success of ReLU is mainly due to fact that existing networks tend to be overparameterized and ReLU can easily regularize overparameterized networks by introducing sparsity (i.e., post-activation output is zero) Glorot et al. (2011), leading to promising results in many computer vision tasks (e.g., image classification Simonyan & Zisserman (2014); He et al. (2016), object detection Dai et al. (2021); Joseph et al. (2021)).

In this paper, we study the ReLU's sparsity constraint in the context of network pruning (i.e., a method of compression that removes weights from the network). Specifically, we question the utility of ReLU's sparsity constraint, when the network is no longer overparameterized during iterative pruning. In the following, we summarize the workflow of our study together with our contributions.

1. **Motivation and Theoretical Study.** In Section 3.1, we introduce a term called dynamic Dead Neuron Rate (DNR), which quantifies the sparsity introduced by ReLU neurons that are not completely pruned during iterative pruning. Through rigorous experiments on existing networks (e.g., ResNet He et al. (2016)), we find that the more the network is pruned, the smaller the dynamic DNR becomes during and after optimization. This suggests that the sparsity introduced by ReLU is not beneficial for pruned networks. Further theoretical investigations also reveal the importance of reducing dynamic DNR for pruned networks from an information bottleneck (IB) Tishby & Zaslavsky (2015) perspective (see Section 3.2).

2. **A Method for De-sparsifying Pruned Networks.** In Section 3.3, we propose a method called Activate-while-Pruning (AP) which aims to explicitly reduce dynamic DNR. We note that AP does not function as a stand-alone method, as it does not evaluate the importance of weights. Instead, it works in

tandem with existing pruning methods and aims to improve their performance by reducing dynamic DNR. AP has two variants: (i) AP-Lite which slightly improves the performance of existing methods, but without increasing the algorithm complexity, and (ii) AP-Pro which introduces an addition retraining step to the existing methods in every pruning cycle, but significantly improves the performance of existing methods.

3. **Experiments.** In Section 4, we conduct experiments on CIFAR-10, CIFAR-100 Krizhevsky et al. (2009) with various popular networks (e.g., ResNet, VGG, MobileNet Sandler et al. (2018), DenseNet Huang et al. (2017)) using two classical and three state-of-the-art (SOTA) pruning methods. The results demonstrate that AP works well with existing pruning methods and improve their performance by 3% - 4%. For the larger scale dataset (e.g., ImageNet Deng et al. (2009)) and SOTA networks (e.g., vision transformer Dosovitskiy et al. (2020)), we observe an improvement of 2% - 3% with AP as opposed to without.

4. **Ablation Study.** In Section 4.3, we carry out an ablation study to further investigate and demonstrate the effectiveness of several key components that make up the proposed AP.

2 Background

Network pruning is a method used to reduce the size of the neural network, with its first work LeCun et al. (1998) dating back to 1990. In terms of the pruning style, all existing methods can be divided into two classes: (i) **One-Shot Pruning** and (ii) **Iterative Pruning**. Assuming that we plan to prune $Q\%$ of the parameters of a trained network, a typical **pruning cycle** consists of three basic steps:

1. Prune $\eta\%$ of existing parameters based on given metrics.

2. Freeze pruned weights as zero.

3. Retrain the pruned network to recover the performance.

In One-Shot Pruning, η is set to Q and the parameters are pruned in one pruning cycle. While for Iterative Pruning, a much smaller portion of parameters (i.e., $\eta << Q$) are pruned per pruning cycle. The pruning process is repeated multiple times until $Q\%$ of parameters are pruned. As for performance, Iterative Pruning often results in better performance compared to One-Shot Pruning Han et al. (2015); Frankle & Carbin (2019); Li et al. (2017); Vysogorets & Kempe (2023); Zhang & Freris (2023). So far, existing works aim to improve the pruning performance by exploring either new pruning metrics or new retraining methods.

Pruning Metrics. Weight magnitude is the most popular approximation metric used to determine less useful connections; the intuition being that smaller magnitude weights have a smaller effect in the output, and hence are less likely to have an impact on the model outcome if pruned He et al. (2020); Li et al. (2020a;b). Many works have investigated the use of weight magnitude as the pruning metric, i.e. Han et al. (2015); Frankle & Carbin (2019). More recently, Lee et al. (2020) introduced layer-adaptive magnitude-based pruning (LAMP) and attempts to prune weights based on a scaled version of the magnitude. Park et al. (2020) proposed a method called Lookahead Pruning (LAP), which evaluates the importance of weights based on the impact of pruning on neighbor layers. Another popular metric used for pruning is via the gradient; the intuition being that weights with smaller gradients are less impactful in optimizing the loss function. Examples are LeCun et al. (1998); Theis et al. (2018), where LeCun et al. (1998) proposed using the second derivative of the loss function with respect to the parameters (i.e., the Hessian Matrix) as a pruning metric and Theis et al. (2018) used Fisher information to approximate the Hessian Matrix. A recent work Blalock et al. (2020) reviewed numerous pruning methods and suggested two classical pruning methods for performance evaluation:

1. **Global Magnitude**: Pruning weights with the lowest absolute value globally (anywhere in the network).

2. **Global Gradient**: Pruning weights with the lowest absolute value of (weight$\times$gradient) globally.

Retraining Methods. Another factor that significantly affects the pruning performance is the retraining method. According to Han et al. (2015), Han et al. trained the unpruned network with a learning rate schedule and retrained the pruned network using a constant learning rate (i.e., often the final learning rate of the learning rate schedule). A recent work Renda et al. (2019) proposed learning rate rewinding which used the same learning rate schedule to retrain the pruned network, leading to a better pruning performance.

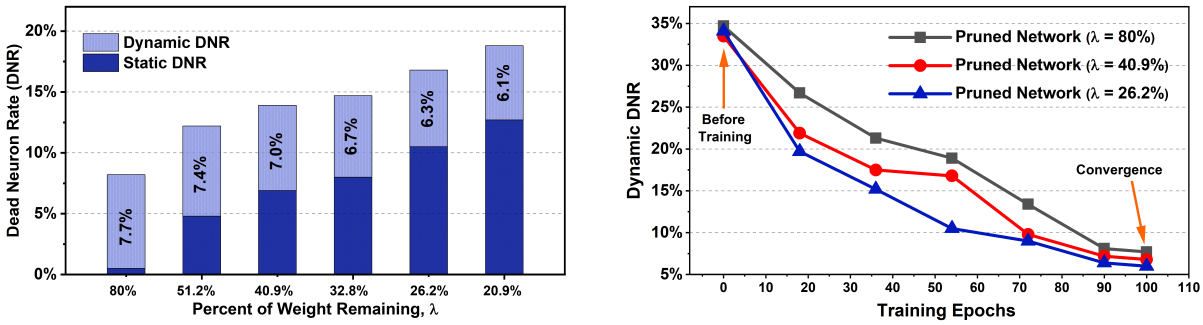

Figure 1: **Left**: Dynamic and static Dead Neuron Rate (DNR) when iteratively pruning ResNet-20 using Global Magnitude; **Right**: The corresponding dynamic DNR during optimization.

More recently, Liu et al. (2021a) attempted to optimize the choice of learning rate (LR) during retraining and proposed a LR schedule called S-Cyc. They showed that S-Cyc could work well with various pruning methods, further improving the existing performance. Most notably, Frankle & Carbin (2019) found that resetting the unpruned weights to their original values (known as **weight rewinding**) after each pruning cycle could lead to even higher performance than the original model. Some follow-on works Zhou et al. (2019); Renda et al. (2019); Malach et al. (2020) investigated this phenomenon more precisely and applied this method in other fields (e.g., transfer learning Mehta (2019), reinforcement learning and natural language processing Yu et al. (2020)). One interesting work to mention is Chen et al. (2022), which further examined the lottery ticket hypothesis from other perspectives, such as interpretability and geometry of loss landscapes.

Other Works. In addition to works mentioned above, several other works also share some deeper insights on network pruning Liu et al. (2019b); Zhu & Gupta (2018); Liu et al. (2019a); Wang et al. (2020); Li & et al (2022); Wang & et al (2022). For example, Liu et al. (2019a) demonstrated that training from scratch on the right sparse architecture yields better results than pruning from pre-trained models. Similarly, Wang et al. (2020) suggested that the fully-trained network could reduce the search space for the pruned structure. More recently, Luo & Wu (2020) addressed the issue of pruning residual connections with limited data and Ye et al. (2020) theoretically proved the existence of small subnetworks with lower loss than the unpruned network. You & et al (2022) motivated the use of the affine spline formulation of networks to analyze recent pruning techniques. Liu et al. (2022) applied the network pruning technique in graph networks and approximated the subgraph edit distance.

3 Activate-while-Pruning

In this section, we first conduct experiments to evaluate the DNR during iterative pruning in Section 3.1. Next, in Section 3.2, we link the experimental results to theoretical studies and motivate Activate-while-Pruning (AP). In Section 3.3, we introduce the idea of AP and present its algorithm. Lastly, in Section 3.4, we illustrate how AP can improve on the performance of existing pruning methods.

3.1 Experiments on DNR

We study the state of the ReLU function during iterative pruning and introduce a term called Dead Neuron Rate (DNR), which is the percentage of dead ReLU neurons (i.e., a neuron with a post-ReLU output of zero) in the network averaged over all training samples when the network converges. Mathematically, the DNR can be written as

$$\text{DNR} = \frac{1}{n}\sum_{i=1}^{n}\frac{\#\text{ of dead ReLU neurons}}{\text{all neurons in the unpruned network}} \tag{1}$$

where n is the number of training samples. We classify a dead neuron as either dynamically dead or statically dead. The **dynamically dead neuron** is a dead neuron in which not all of the weights have been pruned.

Hence, it is not likely to be permanently dead and its state depends on its input. As an example, a neuron can be dead for a sample X, but it could be active (i.e., post-ReLU output > 0) for a sample Y. The DNR contributed by dynamically dead neurons is referred to as **dynamic DNR**. The **statically dead neuron** is a dead neuron in which all associated weights have been pruned. The DNR contributed by statically dead neurons is referred to as **static DNR**.

DNR is a term that we introduce to quantify the sparsity introduced by ReLU. Many similar sparsity metrics have been proposed in the literature Hurley & Rickard (2009). As an example, the Gini Index Goswami et al. (2016) computed from Lorenz curve (i.e., plot the cumulative percentages of total quantities) can be used to evaluate the sparsity of network graphs. Another popular metric will be Hoyer measure Hoyer (2004) which is the ratio between L1 and L2 norms, can also be used to evaluate the sparsity of networks. The closest metric to DNR is parameter sparsity Goodfellow et al. (2016) which computes the percentage of zero-magnitude parameters among all parameters. Both parameter sparsity and DNR will contribute to sparse representations, and in this paper, we use DNR to quantify the sparsity introduced by ReLU.

Experiment Setup and Observations. Given the definition of DNR, static and dynamic DNR, we conduct pruning experiments using ResNet-20 on the CIFAR-10 dataset with the aim of examining the benefit (or lack thereof) of ReLU's sparsity for pruned networks. We iteratively prune ResNet-20 with a pruning rate of 20 (i.e., 20% of existing weights are pruned) using the Global Magnitude (i.e., prune weights with the smallest magnitude anywhere in the network). We refer to the standard implementation reported in Renda et al. (2019); Frankle & Carbin (2019) (i.e., SGD optimizer Ruder (2016), 100 training epochs and batch size of 128, learning rate warmup to 0.03 and drop by a factor of 10 at 55 and 70 epochs) and compute the static DNR and dynamic DNR while the network is iteratively pruned. The experimental results are shown in Fig. 1, where we make two observations.

1. As shown in Fig. 1 (left), the value of DNR (i.e., sum of static and dynamic DNR) increases as the network is iteratively pruned. As expected, static DNR grows as more weights are pruned.

2. Surprisingly, dynamic DNR tends to decrease as the network is iteratively pruned (see Fig. 1 (left)), suggesting that pruned networks do not favor the sparsity of ReLU. In Fig. 1 (right), for pruned networks with different λ (i.e., percent of remaining weights), they have similar dynamic DNR at beginning, but the pruned network with smaller λ tends to have a smaller dynamic DNR during and after optimization.

Result Analysis. One possible reason for the decrease in dynamic DNR could be due to the fact that once the neuron is dead, its gradient becomes zero, meaning that it stops learning and degrades the learning ability of the network Lu et al. (2019); Arnekvist et al. (2020). This could be beneficial as existing networks tend to be overparameterized and dynamic DNR may help to reduce the occurrence of overfitting. However, for pruned networks whose learning ability are heavily degraded, the dynamic DNR could be harmful as a dead ReLU always outputs the same value (zero as it happens) for any given non-positive input, meaning that it takes no role in discriminating between inputs. Therefore, during retraining, the pruned network attempts to restore its performance by reducing its dynamic DNR so that the extracted information can be passed to the subsequent layers. Similar performance trends can be observed using VGG-19 with Global Gradient (see Fig. 3 in the **Appendix**). Next, we present a **theoretical study** of DNR and show its relevance to the network's ability to discriminate.

3.2 Theoretical Insights: Relevance to Information Bottleneck and Complexity

Here, we present some theoretical results and subsequent insights that highlight the relevance of the dynamic DNR of a certain layer of the pruned network to the Information Bottleneck (IB) method proposed in Tishby & Zaslavsky (2015). In the IB setting, the computational flow is denoted as $X \rightarrow T \rightarrow Y$, where X represents the input, T represents the extracted representation, and Y represents the network's output. In Tishby & Zaslavsky (2015), the authors observed that the training of neural networks is essentially a process of minimizing the mutual information Cover & Thomas (2006) between X and T (denoted as $I(X;T)$) while keeping $I(Y;T)$ large (precisely what IB suggests). A consequence of this is that over-compressed features (very low $I(X;T)$) will not retain enough information to predict the labels, whereas under-compressed

features (high $I(X;T)$) imply that more label-irrelevant information is retained in T which can adversely affect generalization performance. Next, we provide a few definitions.

Definition 1. Layer-Specific dynamic DNR ($D_{DNR}(T)$): We are given a dataset $S = \{X_1, ..., X_m\}$, where $X_i \sim P \ \forall i$ (i.i.d) and P is the data generating distribution. We denote the dynamic DNR of the neurons at a certain layer within the network represented by the vector T, by $D_{DNR}(T)$. $D_{DNR}(T)$ is computed over the entire distribution of input in P.

Definition 2. Layer-Specific static DNR ($S_{DNR}(T)$): In the same manner as $D_{DNR}(T)$, we define the layer-specific static DNR of a network layer T.

With this, we now outline our first theoretical result which highlights the relevance of $D_{DNR}(T)$ and $S_{DNR}(T)$ to $I(X;T)$, as follows.

Theorem 1. We are given the computational flow $X \rightarrow T \rightarrow Y$, where T represents the features at some arbitrary layer within a network, which are represented with finite precision (e.g., float32 or float64). We consider the subset of network configurations for which (a) the activations in T are less than a threshold τ and (b) the zero-activation probability of each neuron in T is upper bounded by some $p_S < 1$. Let $dim(T)$ represent the dimensionality of T, i.e., the number of neurons at that depth. We then have,

$$I(X;T) \leq C \times dim(T) \times \Big(1 - S_{DNR}(T) - D_{DNR}(T)\Big(1 - \frac{1}{C}\log\frac{1 - S_{DNR}(T)}{D_{DNR}(T)}\Big)\Big), \tag{2}$$

for a finite, independent constant C that only depends on the network architecture, τ and p_S.

The following corollary addresses the dependencies of Theorem 1. The proof of Theorem 1 and Corollary 1 are provided in the **Appendix**.

Corollary 1. In the setting of Theorem 1, the right hand side of equation 2 decreases as $D_{DNR}(T)$ or $S_{DNR}(T)$ increases.

Remark 1. (Relevance to Complexity) We see that Shamir et al. (2010) notes how the metric $I(X;T)$ represents the *effective complexity* of the network. As Theorem 3 in Shamir et al. (2010) shows, $I(X;T)$ captures the dependencies between X and T and directly correlates with the network's ability to fit the data. Coupled with the observations from Theorem 1 and Corollary 1, for a fixed pruned network configuration (i.e., fixed $S_{DNR}(T)$), greater $D_{DNR}(T)$ will likely reduce the *effective complexity* of the network, undermining the function-fitting ability of the neural network.

Remark 2. (Motivation for AP) Theorem 1 also shows that a pruned network, which possesses large $S_{DNR}(T)$, leads to a higher risk of *over-compression* of information (low $I(X;T)$). To address this issue, we can reduce the dynamic DNR (from Corollary 1) so that the upper bound of $I(X;T)$ can be increased, mitigating the issue of *over-compression* for a pruned network. This agrees with our initial motivation that the sparsity introduced by ReLU is not beneficial for the pruned network and reducing dynamic DNR helps in avoiding over-compressed features while simultaneously increasing the effective complexity of the network.

3.3 Algorithm of Activate-while-Pruning

The experimental and theoretical results above suggest that, in order to better preserve the learning ability of pruned networks, a smaller dynamic DNR is preferred. This motivates us to propose Activate-while-Pruning (AP) which aims to explicitly reduce dynamic DNR.

We note that the proposed AP does not work alone, as it does not evaluate the importance of weights. Instead, it serves as a booster to existing pruning methods and help to improve their pruning performance by reducing dynamic DNR (see Fig. 2). Assume that the pruning method X removes $p\%$ of weights in every pruning cycle (see the upper part in Fig. 2). After using AP, the overall pruning rate remains unchanged as $p\%$, but $(p-q)\%$ of weights are pruned according to the pruning method X with the aim of pruning less important weights, while $q\%$ of weights are pruned according to AP (see the lower part in Fig. 2) with the aim of reducing dynamic DNR. Consider a network $f(\theta)$ with ReLU activation function. Two key steps to reducing dynamic DNR are summarized as follows.

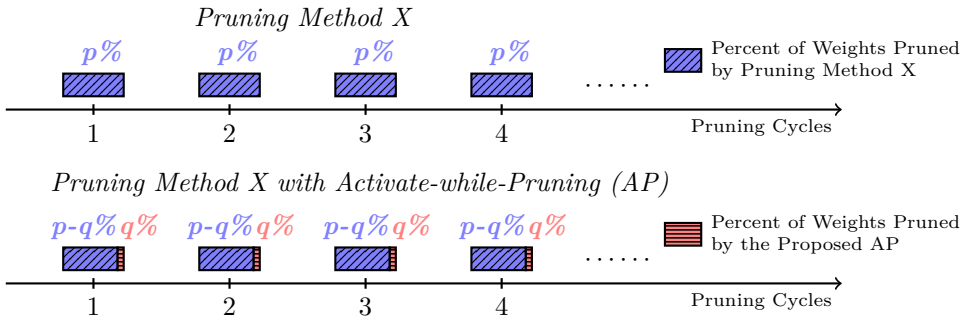

Figure 2: Illustration of how AP works in tandem with existing pruning methods (e.g., method X).

(1) Locate Dead ReLU Neurons. Consider a neuron in the hidden layer with ReLU activation function, taking n inputs $\{X_1W_1, ..., X_nW_n | X_i \in \mathbb{R}$ is the input and $W_i \in \mathbb{R}$ is the associated weight$\}$. Let j be the pre-activated output of the neuron (i.e., $j = \sum_{i=1}^{n} X_iW_i$) and $\mathcal{J}$ be the post-activated output of the neuron ($\mathcal{J} = ReLU(j)$). Let $\mathcal{L}$ be the loss function and assume the neuron is dead ($\mathcal{J} = 0$), then the gradient of its associated weights (e.g., W_1) with respected to the loss function will be $\frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial \mathcal{J}} \cdot \frac{\partial \mathcal{J}}{\partial j} \cdot \frac{\partial j}{\partial W_1} = 0$ as $\frac{\partial \mathcal{J}}{\partial j} = 0$. If a neuron is often dead during training, the weight movement of its associated weights is likely to be smaller than other neurons. Therefore, we compute the difference between weights at initialization (θ_0) and the weights when the network convergences (θ_*), i.e., $|\theta_* - \theta_0|$ and use it as a heuristic to locate dead ReLU neurons.

(2) Activate Dead ReLU Neurons. Assume we have located a dead neuron in the hidden layer with n inputs $\{X_1W_1, ..., X_nW_n$ $| X_i \in \mathbb{R}$ is the input and $W_i \in \mathbb{R}$ is the associated weight$\}$. We note that X_i is non-negative as X_i is usually the post-activated output from the previous layer (i.e., the output of ReLU is non-negative). Therefore, a straightforward way to activate the dead neuron is to prune the weights with the negative value. By pruning such negative weights, we can increase the value of the pre-activation output, which may turn the the pre-activation output into positive so as to reduce dynamic DNR.

3.4 How AP Improves Existing Methods

We now summarize how AP can improve existing pruning methods in Algorithm 2, where the upper part is the algorithm of a standard iterative pruning method called pruning method X and the lower part is the algorithm of method X with AP. The proposed AP has two variants: **AP-Pro** and **AP-Lite**. We note that both AP-Pro and AP-Lite contain the same three steps, summarized as follows.

- **Step 1: Pruning.** Given a network at convergence with a set of dynamically dead ReLU neurons, $\mathcal{N}_1 = \{n_1, n_2, n_3, ...\}$. The pruning step of AP aims to activate these dynamically dead ReLU neurons (i.e., reduce dynamic DNR) so as to preserve the learning ability of the pruned network (see the pruning metric of AP in algorithm 1).

- **Step 2: Weight Rewinding.** Resetting unpruned weights to their values at the initialization. We note that different weight initializations could lead to different sets of $\mathcal{N}$. In step 1, AP aims to reduce dynamic DNR for the target $\mathcal{N}_1$ and weight rewinding attempts to prevent the target $\mathcal{N}_1$ from changing too much. Since the weights of ReLU neurons in $\mathcal{N}_1$ have been pruned by AP, these neurons could become active during retraining. The effect of weight rewinding is evaluated via an ablation study.

- **Step 3: Retraining.** Retrain the pruned network to recover performance.

The **key difference** between AP-Lite and AP-Pro is that AP-Lite applies these three steps only once at the end of pruning (i.e., when all pruning cycles ends). It aims to slightly improve the performance, but does not substantially increase the algorithm complexity. For AP-Pro, it applies the three steps above in every pruning cycle, which increases the algorithm complexity (mainly due to the retraining step), but aims to significantly improve the performance, which could be preferred in performance oriented tasks.

Algorithm 1 The Pruning Metric of the Proposed AP

Require: (i) Network f with unpruned weights θ_0 at initialization, $f(\theta_0)$; (ii) Network f with unpruned weights θ_* at convergence, $f(\theta_*)$; (iii) Pruning Rate of AP, q;
Locate Dead Neurons: Sort $|\theta_*$ - $\theta_0|$ in an ascending order.
Activate Dead Neurons: In the ascending order of $|\theta_*$ - $\theta_0|$, prune first q% negative weights.

Algorithm 2 The Pruning Method X with and without AP

Require: (i) Network, $f(\theta)$; (ii) Pruning Rate of Method X, p; (iii) Pruning Rate of AP, q; (iv) Pruning Cycles, n; (v) Pro_Flag = {0: AP-Lite, 1: AP-Pro};

——— **The Conventional Pruning Method X** ———

for $i = 1$ to n **do**
  Randomly initialize unpruned weights, $\theta \leftarrow \theta_0$.
  Train the network to convergence, arriving at parameters θ_*.
  Prune p % of θ_* according to the pruning method X.
end for
Retrain: Retrain the network to recover its performance.

——— **The Conventional Pruning Method X with Proposed AP** ———

for $i = 1$ to n **do**
  Randomly initialize unpruned weights, $\theta \leftarrow \theta_0$.
  Train the network to convergence, arriving at parameters θ_*.
  Prune (p - q) % of θ_* according to the pruning method X.
  if Pro_Flag **then** # Execution of AP-Pro
    (i) *Pruning: Prune q % of parameter θ_* according to the metric of AP (see details in Algo. 1).*
    (ii) *Weight Rewinding: Reset the remaining parameters to their values in θ_0.*
    (iii) *Retrain: Retrain the pruned network to recover its performance.*
  end if
end for
if NOT Pro_Flag **then** # Execution of AP-Lite
  (i) *Pruning: Prune q % of the parameters θ_* according to the metric of AP (see details in Algo. 1).*
  (ii) *Weight Rewinding: Reset the remaining parameters to their values in θ_0.*
  (iii) *Retrain: Retrain the pruned network to recover its performance.*
end if

4 Performance Evaluation

In this section, we first summarize the experiment setup in Section 4.1. Next, in Section 4.2, we compare and analyze the results obtained. Lastly, in Section 4.3, we conduct an ablation study to evaluate the effectiveness of several components in AP.

4.1 Experiment Setup

(1) Experiment Details. To demonstrate that AP can work well with different pruning methods, we shortlist two classical and SOTA pruning methods. The details are summarized as follows.

1. Pruning ResNet-20 on the CIFAR-10 dataset using Global Magnitude with and without AP.

2. Pruning VGG-19 on the CIFAR-10 dataset using Global Gradient with and without AP.

3. Pruning DenseNet-40 Huang et al. (2017) on CIFAR-100 using Layer-Adaptive Magnitude-based Pruning (LAMP) Lee et al. (2020) with and without AP.

4. Pruning MobileNetV2 Sandler et al. (2018) on the CIFAR-100 dataset using Lookahead Pruning (LAP) Park et al. (2020) with and without AP.

5. Pruning ResNet-50 He et al. (2016) on the ImageNet (i.e., ImageNet-1000) using Iterative Magnitude Pruning (IMP) Frankle & Carbin (2019) with and without AP.

6. Pruning Vision Transformer (ViT-B-16) on CIFAR-10 using IMP with and without AP.

We train the network using SGD with momentum = 0.9 and a weight decay of 1e-4 (same as Renda et al. (2019); Frankle & Carbin (2019)). For the benchmark pruning method, we prune the network with a pruning rate $p = 20$ (i.e., 20% of existing weights are pruned) in 1 pruning cycle. After using AP, the overall pruning rate remains unchanged as 20%, but 2% of existing weights are pruned based on AP, while the other 18% of existing weights are pruned based on the benchmark pruning method to be compared with (see Algorithm 2). We repeat 25 pruning cycles in 1 run and use the early-stop top-1 test accuracy (i.e., the corresponding test accuracy when early stopping criteria for validation error is met) to evaluate the performance. The experimental results averaged over 5 runs and the corresponding standard deviation are summarized in Tables 1 - 6, where λ is the percentage of weights remaining.

(2) Hyper-parameter Selection and Tuning. To ensure fair comparison against prior results, we utilize standard implementations (i.e., network hyper-parameters and learning rate schedules) reported in the literature. Specifically, the implementations for Tables 1 - 6 are from Frankle & Carbin (2019), Zhao et al. (2019), Chin et al. (2020), Renda et al. (2019) and Dosovitskiy et al. (2020). The implementation details can be found in Section B.2 of the **Appendix**. In addition, we also tune hyper-parameters using the validation dataset via grid search. Some hyper-parameters are tuned as follows. (i) The training batch size is tuned from {64, 128, ..., 1024}. (ii) The learning rate is tuned from 1e-3 to 1e-1 via a stepsize of 2e-3. (iii) The number training epochs is tuned from 80 to 500 with a stepsize of 20. The validation performance using our tuned parameters are close to that of using standard implementations. Therefore, we use standard implementations reported in the literature to reproduce benchmark results.

(3) Reproducing Benchmark Results. By using the implementations reported in the literature, we have correctly reproduced the benchmark results. For example, the benchmark results in our Tables 1 - 6 are comparable to Fig.11 and Fig.9 of Blalock et al. (2020), Table.4 in Liu et al. (2019b), Fig.3 in Chin et al. (2020), Fig. 10 in Frankle et al. (2020), Table 5 in Dosovitskiy et al. (2020), respectively.

(4) Source Code & Devices: We use Tesla V100 devices for our experiments and the source code (including random seeds) will be released at the camera-ready stage.

4.2 Performance Comparison

(1) Performance using Classical Pruning Methods. In Tables 1 & 2, we show the performance of AP using classical pruning methods (e.g., Global Magnitude, Global Gradient) via ResNet-20 and VGG-19 on CIFAR-10. We observe that as the percent of weights remaining, λ decreases, the improvement of AP becomes larger. For example, in Table 1, the performance of AP-Lite at $\lambda = 26.2\%$ is 1.3% higher than the benchmark result. The improvement increases to 2.6% at $\lambda = 5.7\%$. Note that AP-Lite does not increase the algorithm complexity of existing methods. As expected, in Table 1, AP-Pro leads to a more significant improvement of 2.0% and 4.1% at $\lambda = 26.2\%$ and $\lambda = 5.7\%$, respectively. Similar performance trends can be observed in Table 2 as well. **The results for more values of λ can be found in the Appendix**.

(2) Performance using SOTA and Classical Pruning Methods. In Tables 3 and 4, we show that AP can work well with SOTA pruning methods (e.g., LAMP, LAP). In Table 3, we show the performance of AP using LAMP via DenseNet-40 on CIFAR-100. We observe that AP-Lite improves the performance of LAMP by 1.2% at $\lambda = 13.4\%$ and the improvement increases to 1.6% at $\lambda = 5.7\%$. Note that AP-Lite does not increase the algorithm complexity of existing methods. For AP-Pro, it causes a larger improvement of 4.6% and 3.8% at $\lambda = 13.4\%$ and $\lambda = 5.7\%$, respectively. Similar performance trends can be observed in Table 4, where we show the performance of AP using LAP via MobileNetV2 on CIFAR-100.

(3) Performance on ImageNet. In Table 5, we show the performance of AP using Iterative Magnitude Pruning (IMP, i.e., the lottery ticket hypothesis pruning method) via ResNet-50 on ImageNet (i.e., the ILSVRC version) which contains over 1.2 million images from 1000 different classes. We observe that AP-

Original Top-1 Test Accuracy: 91.7% ($\lambda = 100\%$)				
λ	32.8%	26.2%	13.4%	5.72%
Global Magnitude	90.3 ± 0.4	89.8 ± 0.6	88.2 ± 0.7	81.2 ± 1.1
Global Magnitude with AP-Lite	90.4 ± 0.7	90.2 ± 0.8	88.7 ± 0.7	82.4 ± 1.4
Global Magnitude with AP-Pro	**90.7 ± 0.6**	**90.4 ± 0.4**	**89.3 ± 0.8**	**84.1 ± 1.1**

Table 1: Performance (top-1 test accuracy ± standard deviation) of pruning ResNet-20 on CIFAR-10 using Global Magnitude with and without the proposed AP.

Original Top-1 Test Accuracy: 92.2% ($\lambda = 100\%$)				
λ	32.8%	26.2%	13.4%	5.72%
Global Gradient	90.2 ± 0.5	89.8 ± 0.8	89.2 ± 0.8	76.9 ± 1.1
Global Gradient with AP-Lite	90.5 ± 0.8	90.3 ± 0.7	89.7 ± 0.9	78.4 ± 1.4
Global Gradient with AP-Pro	**90.8 ± 0.6**	**90.7 ± 0.9**	**90.4 ± 0.8**	**79.2 ± 1.3**

Table 2: Performance (top-1 test accuracy ± standard deviation) of pruning VGG-19 on CIFAR-10 using Global Gradient with and without the proposed AP.

Original Top-1 Test Accuracy: 74.6% ($\lambda = 100\%$)				
λ	32.8%	26.2%	13.4%	5.72%
LAMP	71.5 ± 0.7	69.6 ± 0.8	65.8 ± 0.9	61.2 ± 1.4
LAMP with AP-Lite	71.9 ± 0.8	70.3 ± 0.7	66.6 ± 0.7	62.2 ± 1.2
LAMP with AP-Pro	**72.2 ± 0.7**	**71.1 ± 0.7**	**68.8 ± 0.9**	**63.5 ± 1.5**

Table 3: Performance (top-1 test accuracy ± standard deviation) of pruning DenseNet-40 on CIFAR-100 using Layer-Adaptive Magnitude Pruning (LAMP) with and without the proposed AP.

Original Top-1 Test Accuracy: 73.7% ($\lambda = 100\%$)				
λ	32.8%	26.2%	13.4%	5.72%
LAP	72.1 ± 0.8	70.5 ± 0.9	67.3 ± 0.8	64.8 ± 1.5
LAP with AP-Lite	72.5 ± 0.9	70.9 ± 0.8	68.2 ± 1.2	66.2 ± 1.5
LAP with AP-Pro	**72.8 ± 0.7**	**71.4 ± 0.8**	**69.1 ± 0.8**	**67.4 ± 1.1**

Table 4: Performance (top-1 test accuracy ± standard deviation) of pruning MobileNetV2 on CIFAR-100 using Lookahead Pruning (LAP) with and without the proposed AP.

Lite improves the performance of IMP by 1.5% at $\lambda = 5.7\%$. For AP-Pro, it improves the performance of IMP by 2.8% at $\lambda = 5.7\%$.

(3) Performance on SOTA networks (Vision Transformer). Several recent works Liu et al. (2021b); Yuan et al. (2021); Chen et al. (2021) demonstrated that transformer based networks tend to provide excellent performance in computer vision tasks (e.g., classification). We now examine the performance of AP using Vision Transformer (i.e., ViT-B16 with a resolution of 384). We note that the ViT-B16 uses Gaussian Error Linear Units (GELU, GELU(x) = x$\Phi(x)$, where $\Phi(x)$ is the standard Gaussian cumulative distribution function) as the activation function. Similar to ReLU which blocks the negative pre-activation output, GELU heavily regularizes the negative pre-activation output by multiplying an extremely small value of $\Phi(x)$, suggesting that AP could be helpful with pruning GELU based models as well.

We repeat the same experiment setup as above and evaluate the performance of AP using ViT-B16 in Table 6. We observe that AP-Lite helps to improve the performance of IMP by 1.8% at $\lambda = 5.7\%$. For AP-Pro, it improves the performance of IMP by 3.3% at $\lambda = 5.7\%$.

Original Top-1 Test Accuracy: 77.0% ($\lambda = 100\%$)				
λ	32.8%	26.2%	13.4%	5.72%
IMP	76.8 ± 0.2	76.4 ± 0.3	75.2 ± 0.4	71.5 ± 0.4
IMP with AP-Lite	77.2 ± 0.3	76.9 ± 0.4	76.1 ± 0.3	72.6 ± 0.5
IMP with AP-Pro	**77.5 ± 0.4**	**77.2 ± 0.3**	**76.8 ± 0.6**	**73.5 ± 0.4**

Table 5: Performance (top-1 validation accuracy ± standard deviation) of pruning ResNet-50 on ImageNet using Iterative Magnitude Pruning (IMP) with and without AP.

Original Top-1 Test Accuracy: 98.0% ($\lambda = 100\%$)				
λ	32.8%	26.2%	13.4%	5.72%
IMP	97.3 ± 0.6	96.8 ± 0.7	88.1 ± 0.9	82.1 ± 0.9
IMP with AP-Lite	98.0 ± 0.4	97.3 ± 0.7	89.9 ± 0.6	83.6 ± 0.8
IMP with AP-Pro	**98.2 ± 0.6**	**97.6 ± 0.5**	**91.1 ± 0.8**	**84.8 ± 1.0**

Table 6: Performance (top-1 test accuracy ± standard deviation) of pruning Vision Transformer (ViT-B-16) on CIFAR-10 using IMP with and without AP.

4.3 Ablation Study

We now conduct an ablation study to evaluate the effectiveness of components in AP. Specifically, we remove one component at a time in AP and observe the impact on the pruning performance.

1. **AP-Lite-NO-WR**: Using AP-Lite without the weight rewinding step (i.e., remove step (ii) from AP-Lite in Algo. 2). This aims to evaluate the effect of weight rewinding on the pruning performance.

2. **AP-Lite-SOLO**: Using only AP-Lite without the benchmark pruning method (i.e., in every pruning cycle, pruning weights only based on AP). This aims to evaluate if the pruning metric of AP can be used to evaluate the importance of weights.

In Table 7, we conduct experiments of Pruning ResNet-20 on the CIFAR-10 dataset using Global Magnitude. Based on this configuration, we compare the performance of AP-Lite-NO-WR, AP-Lite-SOLO to AP-Lite so as to demonstrate the effectiveness of components in AP. We note that, same as above, we utilize the implementation reported in the literature. Specifically, the hyper-parameters and the learning rate schedule are from Frankle & Carbin (2019).

Effect of Weight Rewinding. In Table 7, we compare the performance of AP-Lite-NO-WR to AP-Lite while the key difference is that AP-Lite utilizes weight rewinding (see Algorithm 2) and AP-Lite-NO-WR does not. We find that the performance of AP-Lite at $\lambda = 51.2\%$ is 2.4% higher than AP-Lite-NO-WR. It suggests the crucial role of weight rewinding in improving the performance. S

When AP Works Solely. The pruning metric of AP (see Algorithm 1) aims to reduce dynamic DNR by pruning. We compare the performance of AP-Lite-SOLO to AP-Lite to evaluate if the pruning metric of AP can be used solely, without working with other pruning methods. In Table 7, we observe that AP-Lite-SOLO performs much worse than AP-Lite. For example, at $\lambda = 51.2\%$, the performance of AP-Lite-SOLO is 87.1, which is 2.8% lower than AP-Lite. It suggests that the pruning metric of AP is not suitable to evaluate the importance of weights. The effect of AP's metric on reducing dynamic DNR and its pruning rate q on pruning performance are discussed in Section 5.

More Results using AP-Pro, VGG-19 and larger datasets. To further validate the results, we also conduct the ablation study using AP-Pro and VGG-19. We summarize the results in Table 8, where the results largely mirror those in Table 7. We note that, we only show the results of ablation study using CIFAR-10 due to limited computational resources. We intend to show results on larger datasets (e.g., ImageNet-1000) in the camera ready version.

λ	64%	51.2%	40.9%	32.8%
AP-Lite	**90.0 ± 0.2**	**89.6 ± 0.5**	**88.9 ± 0.6**	**88.5 ± 0.8**
AP-Lite-SOLO	87.8 ± 0.4	87.1 ± 0.7	86.3 ± 0.9	85.2 ± 1.1
AP-Lite-NO-WR	88.3 ± 0.6	87.5 ± 0.5	86.8 ± 0.8	85.9 ± 0.9

Table 7: Ablation Study: Performance Comparison (top-1 test accuracy ± standard deviation) between AP-Lite and AP-SOLO, AP-Lite-NO-WR on pruning ResNet-20 on CIFAR-10 via Global Magnitude.

λ	64%	51.2%	40.9%	32.8%
AP-Pro	**91.8 ± 0.3**	**91.5 ± 0.5**	**91.2 ± 0.9**	**90.8 ± 0.6**
AP-Pro-SOLO	90.2 ± 0.5	89.1 ± 0.9	87.2 ± 1.4	85.8 ± 1.5
AP-Pro-NO-WR	90.6 ± 0.5	90.2 ± 0.4	88.9 ± 1.0	88.1 ± 1.2

Table 8: Ablation Study: Performance Comparison (top-1 test accuracy ± standard deviation) between AP-Pro and AP-Pro-SOLO, AP-Pro-NO-WR on pruning VGG-19 using CIFAR-10 via Global Gradient.

5 Reflections

In this section, we present some points and experimental results related to the proposed AP.

(1) Pruning Rate of AP, q. Active Pruning removes q% of remaining parameters in every pruning cycle, so as to reduce dynamic DNR. The value of q is usually much smaller than the pruning rate of the pruning method it works with. As an example, in Section 4, the overall pruning rate is fixed as 20% and 2% of weights are pruned based on Active Pruning, which is much smaller than the pruning rate of the benchmark method compared with (i.e., 18%). Adjusting the value of q is a trade-off between pruning less important weights and reducing dynamic DNR. A large q value indicates preferential reduction of dynamic DNR, while a small q value means preferential removal of less important weights.

We repeat the experiments of pruning ResNet-20 on CIFAR-10 using Global Magnitude and AP-Lite. We note that the overall pruning rate is fixed as 20% and the pruning rate of AP increases from 1% to 5%. Correspondingly, the pruning rate of Global Magnitude decreases from 19% to 15%. The experimental results are summarized in Table 9. We observe that as we increase the pruning rate of AP from 2%, the performance tends to decrease. Similar performance trends can be observed using VGG-19 on CIFAR-10 as well (see Table 10). The theoretical determination of the optimal value of q is clearly worth deeper thought. Alternatively, q can be thought of as a hyper-parameter and tuned via the validation dataset and let $q = 2$ could be a good choice as it provides promising results in various experiments.

(2) Dynamic DNR with Active Pruning. We now examine the effect of AP in reducing dynamic DNR. We repeat the same experiments in Section 3.2 and compare the dynamic DNR with and without using AP-Lite in Tables 11 & 12. In Table 11, we observe that the dynamic DNR is reduced from 9.8% to 9.1% at $\lambda = 80\%$ after applying AP-Lite with a pruning rate of 2%. As λ decreases, AP-Lite also works well and reduces the dynamic DNR from 5.1% to 4.4% at $\lambda = 20.9\%$. Similar performance trends can be observed in Table 12, where we prune ResNet-20 using global magnitude. This suggests that AP works as expected and explains why AP is able to improve the pruning performance of existing pruning methods.

(3) Reducing Static DNR. The Theorem 1 shows that, in addition to dynamic DNR, reducing static DNR also can improve the upper bound of $I(X;T)$. In fact, reducing static DNR has been incorporated directly or indirectly into the existing pruning methods. As an example, LAMP (i.e., one SOTA pruning method used in performance evaluation, see Table 3) takes the number of unpruned weights of neurons/layers into account and avoids pruning weights from neurons/filters with less number of unpruned weights. This prevents neurons from being statically dead. Differ from existing methods, AP is the first method targeting the dynamic DNR. Hence, as a method that works in tandem with existing pruning methods, AP improves existing methods by filling in the gap in reducing dynamic DNR, leading to much better pruning performance.

AP Pruning Rate, q	1%	2%	3%	5%
AP-Lite ($\lambda = 64.0\%$)	89.8 ± 0.1	**90.0 ± 0.2**	89.5 ± 0.4	89.2 ± 0.6
AP-Lite ($\lambda = 40.9\%$)	88.2 ± 0.4	**88.9 ± 0.6**	88.5 ± 0.7	87.3 ± 0.5
AP-Lite ($\lambda = 26.2\%$)	87.1 ± 0.5	**87.9 ± 0.8**	86.7 ± 0.8	86.3 ± 0.9

Table 9: Performance (top-1 test accuracy ± standard deviation) of AP-Lite when iterative pruning ResNet-20 on CIFAR-10 with different pruning rate.

AP Pruning Rate, q	1%	2%	3%	5%
AP-Lite ($\lambda = 64.0\%$)	91.1 ± 0.2	**91.7 ± 0.3**	90.2 ± 0.5	89.8 ± 0.6
AP-Lite ($\lambda = 40.9\%$)	88.7 ± 0.5	**89.8 ± 0.9**	89.3 ± 1.1	88.0 ± 0.9
AP-Lite ($\lambda = 26.2\%$)	87.9 ± 0.8	**88.5 ± 0.7**	88.1 ± 1.3	87.1 ± 1.3

Table 10: Performance (top-1 test accuracy ± standard deviation) of AP-Lite when iterative pruning VGG-19 on CIFAR-10 with different pruning rate.

λ	80%	51.2%	40.9%	32.8%	26.2%	20.9%
Global Gradient	9.8%	9.2%	7.6%	7.0%	5.8%	5.1%
AP-Lite (2%)	**9.1%**	**8.6%**	**7.0%**	**6.1%**	**4.9%**	**4.4%**

Table 11: The dynamic DNR when iteratively pruning VGG-19 on CIFAR-10 using Global Gradient and AP-Lite (with a pruning rate of 2%).

λ	80%	51.2%	40.9%	32.8%	26.2%	20.9%
Global Magnitude	7.7%	7.4%	7.0%	6.7%	6.3%	6.1%
AP-Lite (2%)	**7.3%**	**7.1%**	**6.8%**	**6.3%**	**5.9%**	**5.5%**

Table 12: The dynamic DNR when iteratively pruning ResNet-20 on CIFAR-10 using Global Magnitude and AP-Lite (with a pruning rate of 2%).

(4) Working with Non-ReLU based Networks. We would like to highlight that AP also works well with non-ReLU based networks. For example, in Table 5, we show the performance of AP using Vision Transformer which uses GELU as the activation function. In this setup, AP also leads to an improvement of 2% - 3%. We posit that this could be due to the fact that, similar to ReLU which blocks the negative pre-activation output, GELU heavily regularizes the negative pre-activation output by multiplying an extremely small value of $\Phi(x)$, suggesting that AP could be helpful with pruning GELU based models as well.

6 Conclusion

In this paper, we propose a new pruning method called Activate-while-Pruning (AP). Unlike existing pruning methods which remove less important parameters, the proposed AP works in tandem with existing pruning methods and aims to improve their pruning performance by de-sparsifying pruned networks.

Theoretically, we show the benefits of de-sparsifying pruned networks from the perspective of information bottleneck. Empirically, we use six different sets of experiments to demonstrate that AP can work well with a diverse range of networks (e.g., ResNet, VGG, DenseNet, MobileNet) and pruning methods (e.g., IMP, LAP, LAMP, etc) on both CIFAR-10/100 and ImageNet. It should be noted that AP is a generic approach, and by using the proposed AP, the pruning performance of existing pruning methods can be improved by 3% - 8%. Lastly, we conduct an ablation study to further investigate and demonstrate the effectiveness of several key components that make up the proposed AP.

References

Isac Arnekvist, J. Frederico Carvalho, Danica Kragic, and Johannes A. Stork. The effect of target normalization and momentum on dying relu. *CoRR*, abs/2005.06195, 2020.

Davis Blalock et al. What is the state of neural network pruning? In *Proceedings of the Machine Learning and Systems (MLSys)*, 2020.

Chun-Fu Richard Chen, Quanfu Fan, and Rameswar Panda. Crossvit: Cross-attention multi-scale vision transformer for image classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 357–366, 2021.

Tianlong Chen, Zhenyu Zhang, Jun Wu, Randy Huang, Sijia Liu, Shiyu Chang, and Zhangyang Wang. Can you win everything with a lottery ticket? *Transactions of Machine Learning Research*, 2022. URL `https://openreview.net/forum?id=JL6MU9XFzW`.

Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Towards efficient model compression via learned global ranking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1518–1528, 2020.

Thomas M Cover and Joy A Thomas. *Elements of Information Theory, 2nd edition*. John Wiley & Sons, 2006.

Xiyang Dai, Yinpeng Chen, Bin Xiao, Dongdong Chen, Mengchen Liu, Lu Yuan, and Lei Zhang. Dynamic head: Unifying object detection heads with attentions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7373–7382, 2021.

Jia Deng et al. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. IEEE, 2009.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611*, 2019.

Jonathan Frankle et al. Linear mode connectivity and the lottery ticket hypothesis. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 3259–3269, 2020.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323. JMLR Workshop and Conference Proceedings, 2011.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Swati Goswami, C. A. Murthy, and Asit K. Das. Sparsity measure of a network graph: Gini index, 2016.

Song Han et al. Learning both weights and connections for efficient neural network. In *Proceedings of the Advances in Neural Information Processing Systems (Neurips)*, pp. 1135–1143, 2015.

Kaiming He et al. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Yang He, Yuhang Ding, Ping Liu, Linchao Zhu, Hanwang Zhang, and Yi Yang. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2009–2018, 2020.

Patrik O. Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, 5(9), 2004.

Gao Huang et al. Densely connected convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4700–4708, 2017.

Niall P. Hurley and Scott T. Rickard. Comparing measures of sparsity, 2009.

KJ Joseph, Salman Khan, Fahad Shahbaz Khan, and Vineeth N Balasubramanian. Towards open world object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5830–5840, 2021.

Alex Krizhevsky et al. Learning multiple layers of features from tiny images. 2009.

Yann LeCun et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86 (11):2278–2324, 1998.

Jaeho Lee, Sejun Park, Sangwoo Mo, Sungsoo Ahn, and Jinwoo Shin. Layer-adaptive sparsity for the magnitude-based pruning. In *International Conference on Learning Representations (ICLR)*, 2020.

Bailin Li, Bowen Wu, Jiang Su, and Guangrun Wang. Eagleeye: Fast sub-net evaluation for efficient neural network pruning. In *European conference on computer vision*, pp. 639–654. Springer, 2020a.

Hao Li et al. Pruning filters for efficient convnets. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Yawei Li and et al. Revisiting random channel pruning for neural network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 191–201, 2022.

Yun Li, Weiqun Wu, Zechun Liu, Chi Zhang, Xiangyu Zhang, Haotian Yao, and Baoqun Yin. Weight-dependent gates for differentiable neural network pruning. In *European Conference on Computer Vision*, pp. 23–37. Springer, 2020b.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019a.

Linfeng Liu, XU HAN, Dawei Zhou, and Liping Liu. Towards accurate subgraph similarity computation via neural graph pruning. *Transactions on Machine Learning Research*, 2022. URL `https://openreview.net/forum?id=CfzIsWWBlo`.

Shiyu Liu, Chong Min John Tan, and Mehul Motani. S-cyc: A learning rate schedule for iterative pruning of relu-based networks. *CoRR*, abs/2110.08764, 2021a.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10012–10022, 2021b.

Zhuang Liu et al. Rethinking the value of network pruning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019b.

Lu Lu, Yeonjong Shin, Yanhui Su, and George Em Karniadakis. Dying relu and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*, 2019.

Jian-Hao Luo and Jianxin Wu. Neural network pruning with residual-connections and limited-data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1458–1467, 2020.

Eran Malach et al. Proving the lottery ticket hypothesis: Pruning is all you need. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 6682–6691, 2020.

Rahul Mehta. Sparse transfer learning via winning lottery tickets. In *Proceedings of the Advances in Neural Information Processing Systems Workshop on Learning Transferable Skills*, 2019.

Sejun Park, Jaeho Lee, Sangwoo Mo, and Jinwoo Shin. Lookahead: A far-sighted alternative of magnitude-based pruning. 2020.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *International Conference on Learning Representations (ICLR)*, 2019.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

Ohad Shamir, Sivan Sabato, and Naftali Tishby. Learning and generalization with the information bottleneck. *Theoretical Computer Science*, 411(29):2696–2711, 2010.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Lucas Theis et al. Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787*, 2018.

Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Artem Vysogorets and Julia Kempe. Connectivity matters: Neural network pruning through the lens of effective sparsity. *Journal of Machine Learning Research*, 24(99):1–23, 2023.

Huan Wang and et al. Recent advances on neural network pruning at initialization. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI, Vienna, Austria*, pp. 23–29, 2022.

Yulong Wang et al. Pruning from scratch. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 12273–12280, 2020.

Mao Ye et al. Good subnetworks provably exist: Pruning via greedy forward selection. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 10820–10830. PMLR, 2020.

Haoran You and et al. Max-affine spline insights into deep network pruning. *Transactions on Machine Learning Research (TMLR)*, 2022. ISSN 2835-8856.

Hao nan Yu et al. Playing the lottery with rewards and multiple languages: lottery tickets in RL and NLP. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

Li Yuan, Qibin Hou, Zihang Jiang, Jiashi Feng, and Shuicheng Yan. Volo: Vision outlooker for visual recognition. *arXiv preprint arXiv:2106.13112*, 2021.

Yuyao Zhang and Nikolaos M Freris. Adaptive filter pruning via sensitivity feedback. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

Chenglong Zhao et al. Variational convolutional neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2780–2789, 2019.

Hattie Zhou et al. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Proceedings of the Advances in Neural Information Processing Systems (Neurips)*, 2019.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.