# Pruning Neural Networks with Velocity-Constrained Optimization

**Donghyun Oh** DONGHYUNOH@POSTECH.AC.KR
*POSTECH*

**Jinseok Chung** JINSEOKCHUNG@POSTECH.AC.KR
*POSTECH*

**Namhoon Lee** NAMHOONLEE@POSTECH.AC.KR
*POSTECH*

## Abstract

Pruning has gained prominence as a way to compress over-parameterized neural networks. While pruning can be understood as solving a sparsity-constrained optimization problem, pruning by directly solving this problem has been relatively underexplored. In this paper, we propose a method to prune neural networks using the MJ algorithm, which interprets constrained optimization using the framework of velocity-constrained optimization. The experimental results show that our method can prune VGG19 and ResNet32 networks by more than 90% while preserving the high accuracy of the dense network.

## 1. Introduction

Pruning a neural network [5, 6, 10] is to remove the parameters of the network while preserving its performance. In recent years, pruning has gained much attention as a way to efficiently train and deploy large-scale models by saving memory and computation costs. While pruning can be formulated as an optimization problem with sparsity constraints, directly solving it can cause troubles in large-scale problems due to difficulties in training a sparse neural network [3] and the discrete nature of the sparsity constraint.

In this work, we propose an optimization-based method for pruning neural networks. We adopt the MJ algorithm, which uses the framework of velocity-constrained optimization [13]. Specifically, it relaxes the traditional constrained optimization problem by allowing the iterates to be infeasible and considering only the local linear approximations of the possibly non-convex constraints. Our preliminary results show that we can sparsify neural networks with competitive performance. Results on CIFAR-10/100 datasets show that the MJ algorithm can prune the neural network by more than 90% while maintaining high levels of accuracy.

## 2. Velocity-Constrained Optimization and the MJ Algorithm

Consider the following inequality-constrained optimization problem:

$$\min_{x \in C} f(x), \quad C = \{x \in \mathbb{R}^n : g(x) \geq 0\}, \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function, and $g : \mathbb{R}^n \to \mathbb{R}^{n_g}$ defines the inequality constraints. The projected gradient method and the Frank-Wolfe method are the two most widely used methods

for solving (1) and involve solving a quadratic and linear program for each iteration respectively. If $C$ is non-linear and non-convex, solving linear and quadratic programs on $C$ becomes computationally intractable. Velocity-constrained optimization views the constraint in terms of *velocity* or feasible directions, which is the local approximation of the feasible set, involving only the active constraints $\mathrm{I}(x) = \{i \in \{1, \dots, n_g\} : g_i(x) \leq 0\}$. Specifically, the constraint for each iteration is defined by the set $V_\alpha$ defined as follows:

$$V_\alpha(x) = \left\{ v \in \mathbb{R}^n : \nabla g_i(x)^\top v \geq -\alpha g_i(x) \text{ for all } i \in \mathrm{I}(x) \right\} \tag{2}$$

where $\alpha$ is a hyperparameter that balances between converging to a minimum and satisfying the constraints. If $x \in C$, $V_\alpha(x)$ reduces to the tangent cone of $C$ at $x$. Even if $x$ is outside $C$, $V_\alpha(x)$ is a convex polytope consisting of a small number of active constraints. This simple structure of $V_\alpha(x)$ makes velocity-constrained optimization more computationally efficient instead of optimizing on a possibly non-linear and non-convex positional constraint $C$.

MJ algorithm, named after the authors of Muehlebach and Jordan [13], is a novel algorithm for solving constrained optimization problems under the framework of velocity-constrained optimization. It finds the direction that is closest to the steepest descent direction $-\nabla f(x)$ in the constraint set $V_\alpha(x)$. An iteration of the MJ algorithm is given as

$$\begin{cases} x_{k+1} & = x_k + \gamma_k v_k \\ v_k & = \operatorname{argmin}_{v \in V_\alpha(x_k)} \frac{1}{2} \left\| v + \nabla f\left(x_k\right) \right\|_2^2 \end{cases} \tag{3}$$

where $(\gamma_k)$ is a sequence of step sizes. $\alpha > 0$, as mentioned above, controls between converging to a minimum and satisfying feasibility. If $\alpha$ is small, the focus is on converging to a minimum, and if $\alpha$ is large, the focus is on satisfying the constraints.

Solving the problem in (3) utilizes the fact that strong duality holds for the problem. The dual problem is given as

$$\lambda_k = \operatorname{argmax}_{\lambda \in D_{x_k}} \mathcal{L}(x_k, \lambda) - \frac{1}{2\alpha} \|\nabla_{x_k} \mathcal{L}(x_k, \lambda)\|_2^2 \tag{4}$$

where $D_{x_k} = \{\lambda \in \mathbb{R}_+^{n_g} : \lambda_i = 0, \forall i \notin \mathrm{I}(x_k)\}$ and $\mathcal{L}(x_k, \lambda) = f(x_k) - \lambda^\top g(x_k)$ is the Lagrangian. By strong duality, $v_k$ and $\lambda_k$ are related by the following equation:

$$v_k = -\nabla_{x_k} \mathcal{L}(x_k, \lambda_k) = -\nabla f(x_k) + \nabla g(x_k) \lambda_k. \tag{5}$$

Therefore an iteration of the MJ algorithm can be rewritten as

$$x_{k+1} = x_k - \gamma_k \nabla f(x_k) + \gamma_k \nabla g(x_k) \lambda_k. \tag{6}$$

A striking feature of the MJ algorithm is that it allows iterates to be infeasible. Even if iterates move outside $C$, the $\gamma_k \nabla g(x_k) \lambda_k$ term in (6) acts as a reaction force that pushes the iterates back to $C$. Also note that if $\lambda_k = 0$, the MJ algorithm reduces to the standard gradient descent algorithm.

To calculate $\lambda_k$, we solve the dual problem (4) by iteratively finding the solution to the following stationarity condition:

$$0 \in \nabla g(x_k)^\top \nabla g(x_k) \lambda_k - \nabla g(x_k)^\top \nabla f\left(x_k\right) + \alpha g\left(x_k\right) + \partial \psi_{D_{x_k}}\left(\lambda_k\right) \tag{7}$$

where $\psi_{D_{w_k}}$ is the indicator function of $D_{w_k}$. Further details on finding $\lambda_k$ using (7) is discussed in Appendix A.

## 3. Problem

The problem of pruning a neural network is given as optimizing both the weight $w$ and the mask $m$, where the element of the mask $m_i$ is set to 0 if $w_i$ is pruned, and 1 if $w_i$ is not pruned.

$$\min_{w,m} \mathcal{R}_{\mathcal{D}}(w,m) := \frac{1}{N}\sum_{i=1}^{N} \mathcal{R}(h(x_i; w \odot m), y_i) \quad \text{subject to} \quad w \in \mathbb{R}^n, \|m\|_1 \leq K, m \in \{0,1\}^n \tag{8}$$

where $\mathcal{R}$ is the loss function, $h(x;\theta)$ is the neural network, $n$ is the number of trainable parameters, and $K$ is the constraint on the number of remaining parameters of the sparse neural network. Since problem (8) is combinatorial and we cannot directly apply first-order methods, we relax it into minimizing the expected loss subject to expected $L_0$ constraints [4, 11, 17].

$$\min_{w,s} f(w,s) := \mathbb{E}_{m\sim \text{Bern}(s)} \mathcal{R}_{\mathcal{D}}(w,m) \qquad \text{subject to} \qquad w \in \mathbb{R}^n, \sum_{i=1}^{n} s_i \leq K, \; s \in [0,1]^n \tag{9}$$

by allowing $m$ to follow a Bernoulli distribution with parameter $s$: $m_i \sim \text{Bern}(s_i)$. We can calculate the gradient estimator of $\nabla f$ by using the concrete distribution [8, 12] and the reparameterization trick [9, 14]. The details are explained in Appendix B.

## 4. Method

The problem (9) can be simplified as

$$\min_{w,s} f(w,s) \qquad \text{subject to} \qquad w \in \mathbb{R}^n, \quad g(s) \geq 0 \tag{10}$$

where $g : \mathbb{R}^n \to \mathbb{R}^{2n+1}$ is defined as below:

$$g(s) = \left( K - \sum_{i=1}^{n} s_i; s_1; \cdots; s_n; 1-s_1; \cdots; 1-s_n \right). \tag{11}$$

Now an iteration of the MJ algorithm is given as

$$s^{k+1} = s^k - \gamma_k \nabla_s f(w^k, s^k) + \gamma_k \nabla_s g(s^k)\lambda_k, \tag{12}$$

$$w^{k+1} = w^k - \gamma_k \nabla_w f(w^k, s^k). \tag{13}$$

Here we used the index of the iterates as $w^k$ and $s^k$ instead of $w_k$ and $s_k$ to avoid confusion from the element $s_i$ of vector $s$.

While we train the relaxed probability $s$, we have to test the model using the mask $s$. This is done by sampling from a Bernoulli distribution with probability $s$.

### 4.1. Comparison with Previous Works

There are a few works that solve the problem (8) by optimizing both the weights and the mask. Learning-Compression [1] algorithm divides the optimization into the learning step and the compression step. The learning step optimizes the augmented Lagrangian with respect to the weights, and the compression step performs projection, subjected to sparsity constraint on the mask.

Louizos et al. [11] performs unconstrained optimization by replacing the constraint with a regularization term on the loss function. The $L_0$ regularizer is relaxed using the reparameterization trick with hard concrete distribution. While it was able to prune neural networks in small datasets, its scalability to large-scale problems was questioned by Gale et al. [3]. Moreover, it required extensive hyperparameter tuning to achieve the desired sparsity level, which is at odds with the constrained optimization framework that explicitly sets the sparsity constraint.

ProbMask [17] is the closest work to ours, as it shares the same problem (reparameterized with concrete distribution, not the hard concrete distribution), and uses a constrained optimization method for pruning. Specifically, it optimizes the probability $m$ with the projected gradient method, following the observation that solving the projection onto the constraint on $m$ can be efficiently computed. While ProbMask shows state-of-the-art performance on various datasets, it also relies on several tricks such as using the gradual pruning schedule [18]. Our work shows that we can achieve competitive results without such tricks.

Gallego-Posada et al. [4] solves a similar problem, although the problem is reparameterized using the hard concrete distribution as in Louizos et al. [11]. It finds the optimal weights and mask by solving the min-max problem on the Lagrangian with the gradient descent-ascent method. In the process, it adopts an auxiliary variable, the Lagrange multiplier, that is also trained and adopts a heuristic schedule on the multiplier to achieve good performance. Our work, which can be viewed as an extension of SGD to constrained problems, uses a simpler algorithm to find sparse neural networks with competitive performance.

## 5. Experiment

### 5.1. Settings

Our experiments follow the settings of Zhou et al. [17], using ProbMask as the baseline. We apply the MJ algorithm to this setting to explore improvements brought about by our proposed method. The experiments utilized VGG19 [15] and ResNet32 [7] architectures on the CIFAR-10 and CIFAR-100 datasets. In the testing phase, we sample a binary mask to evaluate the performance of the pruned networks. For further details on hyperparameter settings, refer to Appendix C.

### 5.2. Experiments on CIFAR-10/100

The results for CIFAR-10/100 datasets are shown in Table 1 and Table 2. Across various sparsity ratios, MJ algorithm performed on par with ProbMask, and had a negligible accuracy decrease compared to the dense network, even outperforming the dense network on low sparsity levels.

| Dataset | CIFAR-10 | | | | | | CIFAR-100 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Sparsity | 30% | 50% | 70% | 80% | 90% | 95% | 30% | 50% | 70% | 80% | 90% | 95% |
| VGG19 | 93.84 | | | | | | 72.56 | | | | | |
| ProbMask | 93.85 | 93.76 | **93.51** | 93.58 | **93.52** | **93.3** | **74.03** | 73.67 | **73.29** | **73.25** | 72.96 | 70.35 |
| **Ours** | **93.93** | **93.92** | 93.5 | **93.69** | 93.35 | 93.29 | 73.66 | **73.69** | 72.74 | 73.01 | **73.0** | **72.15** |

Table 1: The test accuracy of VGG19 on ProbMask and the proposed method on the CIFAR-10 and CIFAR-100 datasets at various levels of sparsity.

| Dataset | CIFAR-10 | | | | | | CIFAR-100 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Sparsity | 30% | 50% | 70% | 80% | 90% | 95% | 30% | 50% | 70% | 80% | 90% | 95% |
| ResNet32 | 93.84 | | | | | | 72.56 | | | | | |
| ProbMask | 94.79 | 94.39 | 94.22 | **94.59** | **94.37** | 91.94 | **74.21** | 72.73 | 72.18 | 72.04 | 70.74 | 68.02 |
| **Ours** | **94.91** | **94.73** | **94.51** | 94.2 | 93.68 | **93.2** | 73.9 | **73.8** | **73.55** | **73.06** | **72.74** | **71.81** |

Table 2: The test accuracy of ResNet32 on ProbMask and the proposed method on the CIFAR-10 and CIFAR-100 datasets at various levels of sparsity.

### 5.3. Limitations

Our proposed method exhibits some limitations:

- **Behavior of Probability Parameter:** While it is desirable to have the probability parameter $s$ to have a bimodal distribution with each parameter converging to either 0 or 1, parameters training by the MJ algorithm do not exactly exhibit this behavior. Some parameters even become infeasible (smaller than 0 or larger than 1), or have values that are not close to 0 or 1 and we currently lack a method to induce a bimodal distribution for $s$. For additional details, see Appendix D.
- **Computational Cost:** The employment of an iterative method (in our experiments, we used 5 iterations) to determine the solution of Equation (7) incurs a computational cost. However, finding the optimal number of iterations that balances the tradeoff between computational efficiency and the accuracy of estimated $\lambda_k$ remains to be explored.

## 6. Conclusion and Future Work

In this work, we proposed applying the MJ algorithm towards pruning a neural network. Experiments showed that a simple first-order method expanding SGD that utilizes the gradient of the loss and the constraint can prune neural networks with high levels of sparsity while maintaining accuracy.

While the MJ algorithm showed promising results, there are several unresolved issues, which we leave for future work. First, this paper only explored global sparsity constraint, but pruning neural networks with additional conditions such as more fine-grained sparsity conditions or conditions that are orthogonal to sparsity, can also be solved with the MJ algorithm. Also, as the MJ algorithm can be seen as the generalization of the SGD algorithm with additional term for satisfying constraints, we can also consider other training heuristics such as learning rate scheduling, and check how these affect training neural networks under sparsity constraints. Finally, the MJ algorithm should be evaluated on large-scale problems such as on ImageNet classification task [2], and fine-tuning or training large language models.

## Acknowledgement

This work was supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2019-0-01906, Artificial Intelligence Graduate School Program(POSTECH)) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2023-00210466).

## References


[1] Miguel A Carreira-Perpinán and Yerlan Idelbayev. "learning-compression" algorithms for neural net pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8532–8541, 2018.

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[3] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[4] Jose Gallego-Posada, Juan Ramirez, Akram Erraqabi, Yoshua Bengio, and Simon Lacoste-Julien. Controlled sparsity via constrained optimization or: How i learned to stop tuning penalties and love constraints. *Advances in Neural Information Processing Systems*, 35:1253–1266, 2022.

[5] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

[6] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5, 1992.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017.

[9] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[10] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

[11] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018.

[12] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017.

[13] Michael Muehlebach and Michael I Jordan. On constraints in first-order optimization: A view from non-smooth dynamical systems. *Journal of Machine Learning Research*, 23(256):1–47, 2022.

[14] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014.

[15] K Simonyan and A Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations (ICLR 2015)*. Computational and Biological Learning Society, 2015.

[16] David Young. Iterative methods for solving partial difference equations of elliptic type. *Transactions of the American Mathematical Society*, 76(1):92–111, 1954.

[17] Xiao Zhou, Weizhong Zhang, Hang Xu, and Tong Zhang. Effective sparsification of neural networks with global sparsity constraint. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3599–3608, 2021.

[18] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

## Appendix A. Details on Implementing the MJ Algorithm

We will discuss how to find the solution to the following problem:

$$\lambda_k = \text{argmax}_{\lambda \in D_{x_k}} \mathcal{L}(x_k, \lambda) - \frac{1}{2\alpha}\|\nabla_{x_k}\mathcal{L}(x_k, \lambda)\|_2^2 \tag{14}$$

where $D_{x_k} = \{\lambda \in \mathbb{R}_+^{n_g} : \lambda_i = 0, \forall i \notin \mathrm{I}(x_k)\}$ and $\mathcal{L}(x_k, \lambda) = f(x_k) - \lambda^\top g(x_k)$. If we have only one constraint ($n_g = 1$) then $\lambda_k$ is expressed in the following closed form:

$$\lambda_k = \begin{cases} \frac{\nabla g(x_k)^\top \nabla f(x_k) - \alpha g(x_k)}{|\nabla g(x_k)|^2} & \text{for } g(x_k) \leq 0, \nabla g(x_k)^\top \nabla f(x_k) - \alpha g(x_k) \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

For the general case, finding $\lambda_k$ requires solving the stationarity condition:

$$0 \in \nabla g(x_k)^\top \nabla g(x_k)\lambda_k - \nabla g(x_k)^\top \nabla f\left(x_k\right) + \alpha g\left(x_k\right) + \partial \psi_{D_{x_k}}\left(\lambda_k\right) \tag{16}$$

where $\psi_{D_{w_k}}$ is the indicator function of $D_{w_k}$. It suffices to calculate non-zero elements of $\lambda_k$ corresponding to active constraints, so we can assume $\lambda_k \in \mathbb{R}_+^{|I(x_k)|}$. Now, defining $W_k = \left(\nabla g_{i_1}\left(x_k\right); \nabla g_{i_2}\left(x_k\right); \cdots; \nabla g_{|I(x_k)|}\left(x_k\right)\right)$ and $\bar{g}_k = \left(g_{i_1}\left(x_k\right); g_{i_2}\left(x_k\right); \cdots; g_{|I(x_k)|}\left(x_k\right)\right)$ where $i_j \in I(x_k)$, the stationarity condition can be written as

$$0 \in W_k^\top W_k \lambda_k - W_k^\top \nabla f(x_k) + \alpha \bar{g}_k + \partial \psi_{D_{x_k}}\left(\lambda_k\right) \tag{17}$$

Now we use the method of successive over-relaxation [16]. We decompose the matrix $W_k^\top W_k$ into

$$W_k^\top W_k = U^\top + D + U = \left(U^\top + \omega^{-1} D\right) + \left(U + \left(1 - \omega^{-1} D\right)\right) \tag{18}$$

where $U$ is upper triangular, $D$ is a diagonal matrix with diagonal elements $\|\nabla g_i(x_k)\|^2$ for $i \in I(x_k)$, and $\omega \in (0, 2)$ is a relaxation factor of the method of successive over-relaxation. Denote $\lambda^j$ as the estimate of $\lambda_k$ at $j$-th iteration of solving the stationarity condition. Then the iteration step for solving (17) becomes

$$0 \in \left(U^\top + \omega^{-1} D\right) \lambda^{j+1} + \partial \psi_{D_{x_k}}\left(\lambda^{j+1}\right) + \left(U + (1 - \omega^{-1}) D\right) \lambda^j - W^\top \nabla f(x_k) + \alpha \bar{g}_k \tag{19}$$

which can be rewritten as

$$\lambda^{j+1} = \text{prox}_{D_{x_k}}\left(\lambda^j - \omega D^{-1}\left(U^\top \lambda^{j+1} + (D + U)\lambda^j - W^\top \nabla f(x) + \alpha \bar{g}_k\right)\right) \tag{20}$$

where $\text{prox}_{D_{x_k}} : \mathbb{R}^{|I(x_k)|} \to \mathbb{R}_+^{|I(x_k)|}$ is defined as

$$\left(\text{prox}_{D_{x_k}}(x)\right)_i = \max\{x_i, 0\}, \quad i = 1, \cdots, |I(x_k)|. \tag{21}$$

In the equation (20), we can show $\lambda^j \to \lambda_k$ as $j \to \infty$. Equation (20) can be simplified as

$$\lambda^{j+1} = \text{prox}_{D_{x_k}}\left(\lambda^j - \omega D^{-1}\left(W_k^\top W_k \lambda^j - W_k^\top \nabla f(x_k) + \alpha \bar{g}_k\right)\right) \tag{22}$$

which can now be easily implemented. The MJ algorithm can be summarized as the following Algorithm 1.

**Algorithm 1:** Implementation of the MJ Algorithm

**Input :** objective function $f$, constraint function $g$, learning rate $\eta$, relaxation factor $\omega$, initialization $x_0 \in \mathbb{R}^n$.

**Output:** approximate optimum to the inequality constrained problem.

**while** $k < \textit{MaxIter}$ **do**

  Compute the active set $I(x_k) = \{i : g_i(x_k) \leq 0\}$

  Define $W_k = \left(\nabla g_{i_1}(x_k); \nabla g_{i_2}(x_k); \cdots; \nabla g_{|I_k|}(x_k)\right)$ where $i_j \in I(x_k)$

  Define $\bar{g}_k = \left(g_{i_1}(x_k); g_{i_2}(x_k); \cdots; g_{|I(x_k)|}(x_k)\right)$ where $i_j \in I(x_k)$

  **while** $j < \textit{MaxProxIter}$ **do**

    $\lambda^{j+1} = \text{prox}_{\mathbb{R}_+^{|I_k|}}\left(\lambda^j - \omega D^{-1}\left(W^\top W \lambda^j - W^\top \nabla f(x) + \alpha \bar{g}_k\right)\right)$

    $j \leftarrow j + 1$

  **end**

  $\lambda_k = \lambda^j$

  Perform the update $x_{k+1} = x_k - \eta \nabla f(x_k) + \eta W_k \lambda_k$

  $k \leftarrow k + 1$

**end**

## Appendix B. Reparameterization Trick with Concrete Distribution

Consider a model with a stochastic node $X$ that involves sampling from a distribution parameterized by $\phi : X \sim p_\phi(x)$. Suppose we want to optimize expected objective $L(\theta, \phi) = \mathbb{E}_{X \sim p_\phi(x)} f_\theta(X)$ for deterministic objective function $f_\theta(x)$. Optimizing such a model using gradient-based methods necessitates estimating the gradient of $L(\theta, \phi)$ with respect to $\phi$. However, the gradient

$$\nabla_\phi L(\theta, \phi) = \nabla_\phi \int p_\phi(x) f_\theta(x) dx = \int f_\theta(x) \nabla_\phi p_\phi(x) dx \tag{23}$$

is not an expectation with respect to $x$ and does not give a straightforward Monte Carlo gradient estimator.

The reparameterization trick enables us to estimate the gradient by reparameterizing the expectation with respect to fixed, $\phi$-free distribution. Specifically, if sampling from $p_\phi(x)$ is equivalent to first sampling from a fixed, $\phi$-free distribution $q(z)$ and transforming that sample with function $g_\phi(z)$, we can reformulate the expected loss function as

$$L(\theta, \phi) = \mathbb{E}_{X \sim p_\phi(x)} f_\theta(x) = \mathbb{E}_{Z \sim q(z)} f_\theta(g_\phi(Z)) \tag{24}$$

Now its gradient $\nabla_\phi L(\theta, \phi) = \mathbb{E}_{Z \sim q(z)} f'(g_\phi(Z)) \nabla_\phi g_\phi(Z)$ can be estimated by the Monte Carlo method.

Concrete distribution, also known as Gumbel-Softmax distribution, was introduced to backpropagate stochastic nodes that involve sampling from a discrete categorical distribution. The idea is to "smooth" the categorical distribution into a continuous concrete distribution and use the concrete distribution for training. The concrete random variable $X$ on the simplex $\Delta^{n-1} = \{x \in \mathbb{R}^n : x_k \in [0, 1], \sum_{k=1}^n x_k = 1\}$ with parameters $\alpha \in \Delta^{n-1}$ and temperature $\beta \in (0, \infty)$ is defined as

$$X_k = \frac{\exp((\log \alpha_k + G_k)/\beta}{\sum_{i=1}^n \exp((\log \alpha_i + G_i)/\beta}, k = 1, \cdots, n. \tag{25}$$

where $G_k$'s are i.i.d and follows $\mathrm{Gumbel}(0, 1)$.

For binary case ($k = 2$), the concrete distribution smooths the Bernoulli distribution and becomes

$$X_2 = \sigma\left(\frac{1}{\beta}\log\frac{\alpha_2}{1-\alpha_2} + G_2 - G_1\right) \tag{26}$$

$$= \sigma\left(\frac{1}{\beta}\log\frac{\alpha_2}{1-\alpha_2} + \log\frac{U}{1-U}\right) \tag{27}$$

$$= \sigma\left(\frac{1}{\beta}\log\frac{\phi U}{1-U}\right) \tag{28}$$

and $X_1 = 1 - X_2$ where $\sigma$ is the sigmoid function, $U \sim \mathrm{Uniform}(0, 1)$ and parameter $\alpha = (\alpha_1, \alpha_2)$ is replaced by $\phi = \frac{\alpha_2}{1-\alpha_2} \in (0, \infty)$. We can apply the reparameterization trick to transform expectation with respect to concrete distribution with parameter $\phi$ to expectation with respect to uniform distribution between 0 and 1.

Although the binary concrete distribution is a good proxy of the Bernoulli distribution, Louizos et al. [11] propose a variant of it, called the hard concrete distribution, which is obtained by stretching and clamping the concrete distribution. Formally, given a concrete random variable $X$, the hard concrete random variable $Y$, with additional parameter $\zeta > 1$ and $\gamma < 0$, is given as

$$Y = \min(1, \max(0, (\zeta - \gamma)X + \gamma)) \tag{29}$$

Compared to concrete random variable, hard concrete random variable has larger support $[0, 1]$ with non-zero probability at 0 and 1. This feature is claimed to better approximate the discrete nature of the Bernoulli distribution. However, it was pointed out by Zhou et al. [17], that the stretching and clamping cause the gradient vanishing problem, hindering training on large-scale problems. For our method, we use the concrete distribution instead of the hard concrete distribution to calculate the gradients.

## Appendix C. Experiment Details

The table below summarizes the specific settings used for the experiments conducted on the CIFAR10 and CIFAR100 datasets.

| Dataset | CIFAR10/100 |
|---|---|
| Batch Size | 256 |
| Epochs | 300 |
| Finetune Epochs | 30 |
| Optimizer | Adam |
| LR | 0.001 |
| $\alpha$ | 0.01 |
| Temperature Annealing | ✓ |

The calculation of $\log s_i$ and $\log(1 - s_i)$ terms while sampling from the concrete distribution can cause errors if $s$ does not satisfy feasibility, which is allowed in the MJ algorithm. To enable feasible sampling, we clamp $s$ to have values between 0 and 1. At the end of the training, we sample the final binary masks to create a pruned network, which is fine-tuned for 20 epochs, following Zhou et al. [17].

## Appendix D. Details on the Distribution of the Probability Parameter

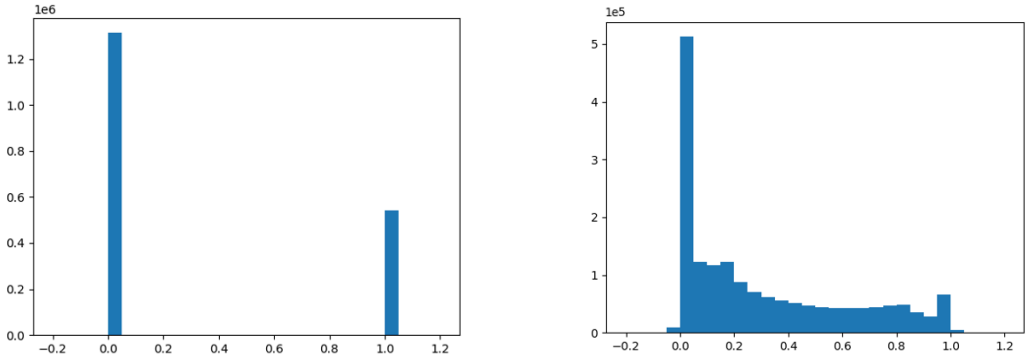


Figure 1: Histograms for ProbMask(Left), ours(Right)

Figure 1 shows the distribution of the probability parameter for the proposed method and ProbMask. Unlike ProbMask, it is evident that the probability parameters in the proposed method are not almost separated into 0 and 1, indicating the non-deterministic behavior of the proposed method. As mentioned in [17], there is a potential for improvement through the adjustment of temperature parameter $\beta$. This technique could help in mitigating the non-deterministic behavior observed in the proposed method, leading to enhanced performance and reliability. The exploration of temperature annealing as a means to enhance the proposed method is considered as a part of future work.