FlipOut: Uncovering Redundant Weights via Sign-flipping

Andrei Apostol, Maarten C. Stol, Patrick Forre
University of Amsterdam, BrainCreators B.V.

Introduction

Pruning has been proven as an effective method of dealing with the computational and memory demands of modern neural networks. However, four problems have been plaguing pruning methods, namely: computational intractability ([3]), requiring to train a network to convergence multiple times ([1, 2, 4]), a need for extensive hyperparameter tuning for optimal results ([7, 8, 10], and an inability to target a specific level of sparsity ([2, 6, 7, 8, 10].

The authors of [2] propose removing the lowest magnitude weights once the network is converged, a method which can be iterated in a train-prune-finetune pipeline. This, however, requires one or multiple cycles of training to convergence.

We propose a test which can determine if a value is a point of local optimum for a weight before convergence is reached, and show that this can be used for pruning by applying this test at the value of 0 for all weights simultaneously, and framing it as a saliency criterion. By design, our method is tractable, allows the user to select a specific level of sparsity and can be applied during training.

Experiments indicate that it is competitive to relevant baselines from literature, achieving state-of-the-art results in some scenarios, while also having default hyperparameter values which generate near-optimal results.

The aim test

Stochastic gradient descent computes the gradient over a mini-batch of examples and updates the weights of a neural network in the opposite direction. Given a weight θ_j^t, one could consider its possible values as being split into two regions, with a locally optimal value θ_j^* as the separation point. Specifically, the updated value θ_j^{t+1} can either land in the region opposite the local optimum (over-shooting) or get closer to it but stay in the same region (under-shooting).

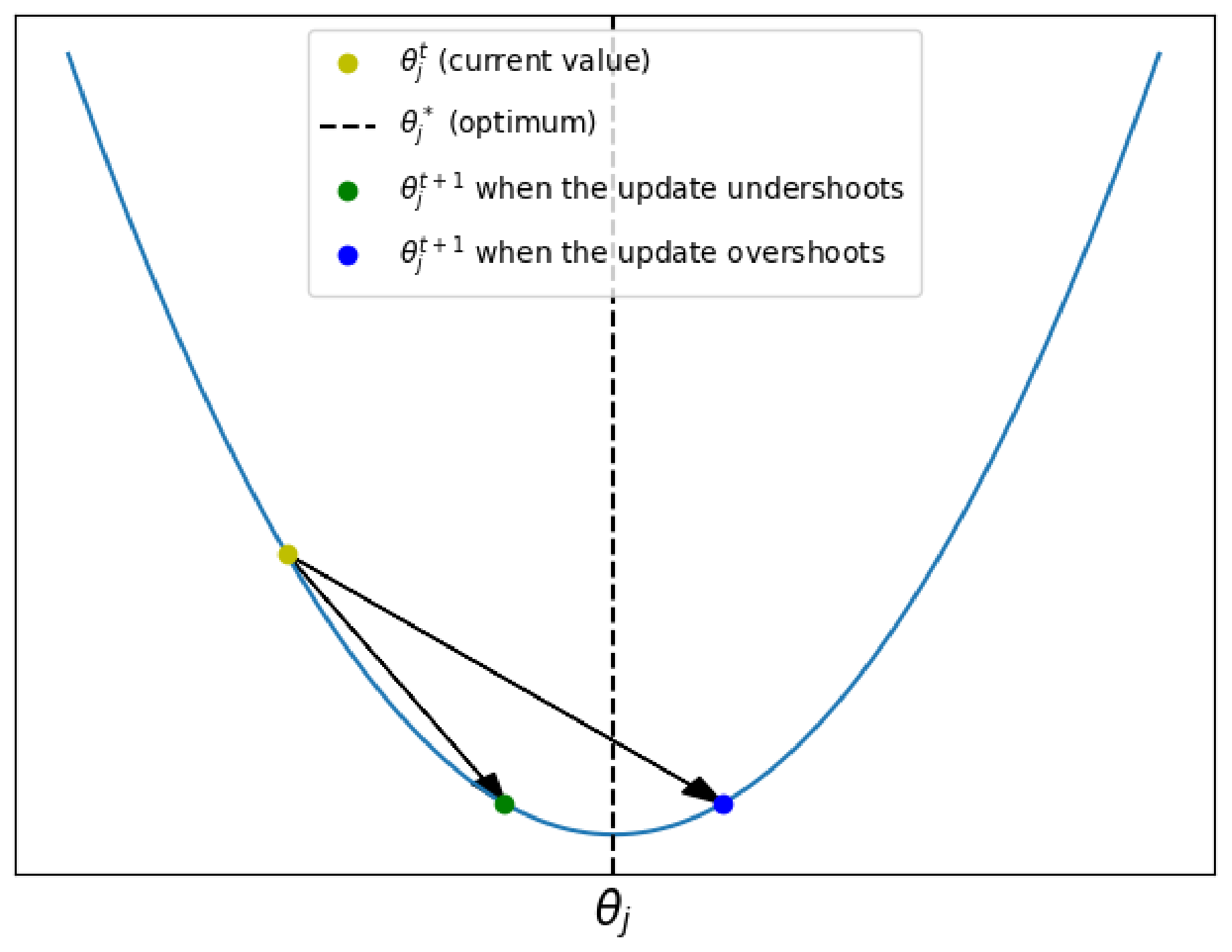
● θ_j^t (current value)
--- θ_j^* (optimum)
● θ_j^{t+1} when the update undershoots
● θ_j^{t+1} when the update overshoots
$\mathcal{L}$
θ_j

Fig. 1: Under- and over-shooting illustrated.

Using this, we can construct a test used to determine if a point is a local optimum for a weight without needing the network to reach convergence, which we coin *the aim test*: (1) pick a value ϕ_j to test as a local optimum for θ_j (2) train the model and record the number of over- or under-shooting events (3) if this number exceeds a threshold κ, conclude that ϕ_j is a local optimum for θ_j.

However, when testing at a value ϕ_j which is not a local optimum for θ_j, occurrences of under- and over-shooting can be a side effect of the weight getting updated to a true local optimum. This can lead the aim test to falsely conclude that ϕ_j is a local optimum. We refer to these events as *deceitful shots* going forward. To correct this, we make two adjustments: the number of occurrences of under- and over-shooting are weighed by the distance between the weight θ_j and the tested value ϕ_j when comparing to the threshold κ (reducing the impact of deceitful shots) and occurrences not in the vicinity of the optimum are ignored (reducing the number of deceitful shots). This second observation also allows for a simplification of the aim test; with sufficient perturbation, under-shooting can be converted into over-shooting. As such, injecting perturbation allows for under-shooting to be ignored entirely.

FlipOut

For pruning, one could apply the aim test simultaneously for all weights with $\boldsymbol{\phi} = \mathbf{0}$. We propose framing this as a saliency score; at time step t, the saliency τ_j^t of a weight θ_j^t is:

$$\tau_j^t = \frac{|\theta_j^t|^p}{\text{flips}_j^t} \quad \text{where} \quad \text{flips}_j^t = \sum_{i=0}^{t-1} [\text{sgn}(\theta_j^i) \neq \text{sgn}(\theta_j^{i+1})] \tag{1}$$

With perturbation added into the weight vector, it is enough to check for over-shooting, which is equivalent to counting the number of sign flips a weight has undergone during the training process, since $\phi_j = 0$. The denominator $|\theta_j^t|^p$ represents the proximity of the weight to the hypothesised local optimum, $|\theta_j^t - \phi_j|^p$. The hyperparameter p controls how much this quantity is weighted relative to the number of sign flips.

To allow for an exact level of sparsity to be selected, pruning is performed every m steps during training, whereby a percentage r of the remaining weights is removed each time. The final sparsity can then be computed as:

$$s = 1 - \frac{||\boldsymbol{\theta}^k||_0}{d_{\boldsymbol{\theta}}} = (1-r)^m$$

We propose adding perturbation via gradient noise. For each layer, we sample from a Gaussian distribution whose variance is proportional to the L_2 norm of that layer divided by its dimensionality:

$$\hat{\boldsymbol{g}}^{t,l} \leftarrow \boldsymbol{g}^{t,l} + \lambda \boldsymbol{\epsilon}^{t,l} \quad , \quad \boldsymbol{\epsilon}^{t,l} \sim \mathcal{N}(0, \sigma^2_{t,l}) \quad , \quad \sigma^2_{t,l} = \frac{\|\boldsymbol{\theta}^{t,l}\|_2^2}{d_l} \tag{2}$$

It is desirable to reduce the amount of added noise so that the network can successfully converge. Previous works ([9]) do this via annealing schedules. This, however, is not required under our formulation, as pruning lowers the L_2 norm of each layer and, implicitly, the variance of the distribution we sample from. The only hyperparameter we introduce here is λ, which controls the scale of the noise. Pruning periodically throughout training according to the saliency score in Eq. 1 in conjunction with adding gradient noise into the weights forms the *FlipOut* pruning method.

Comparison to baselines

We compare our method to relevant baselines from literature ([2, 5, 10] across various vision models and levels of sparsity. For the ResNet18 architecture trained on CIFAR10, we find that our method is competitive for levels of sparsity up to 98.4% and obtains state-of-the-art performance beyond this point. When testing on VGG19, a similar trend emerges. For DenseNet121 trained on ImageNette, our method again performs better than magnitude pruning in the high sparsity regimes, but HoyerSquare is Pareto-dominant for most cases. For all the plots and more details regarding experimental setup please refer to the paper version. In the next section, we detail our choice for the hyperparameters of FlipOut.

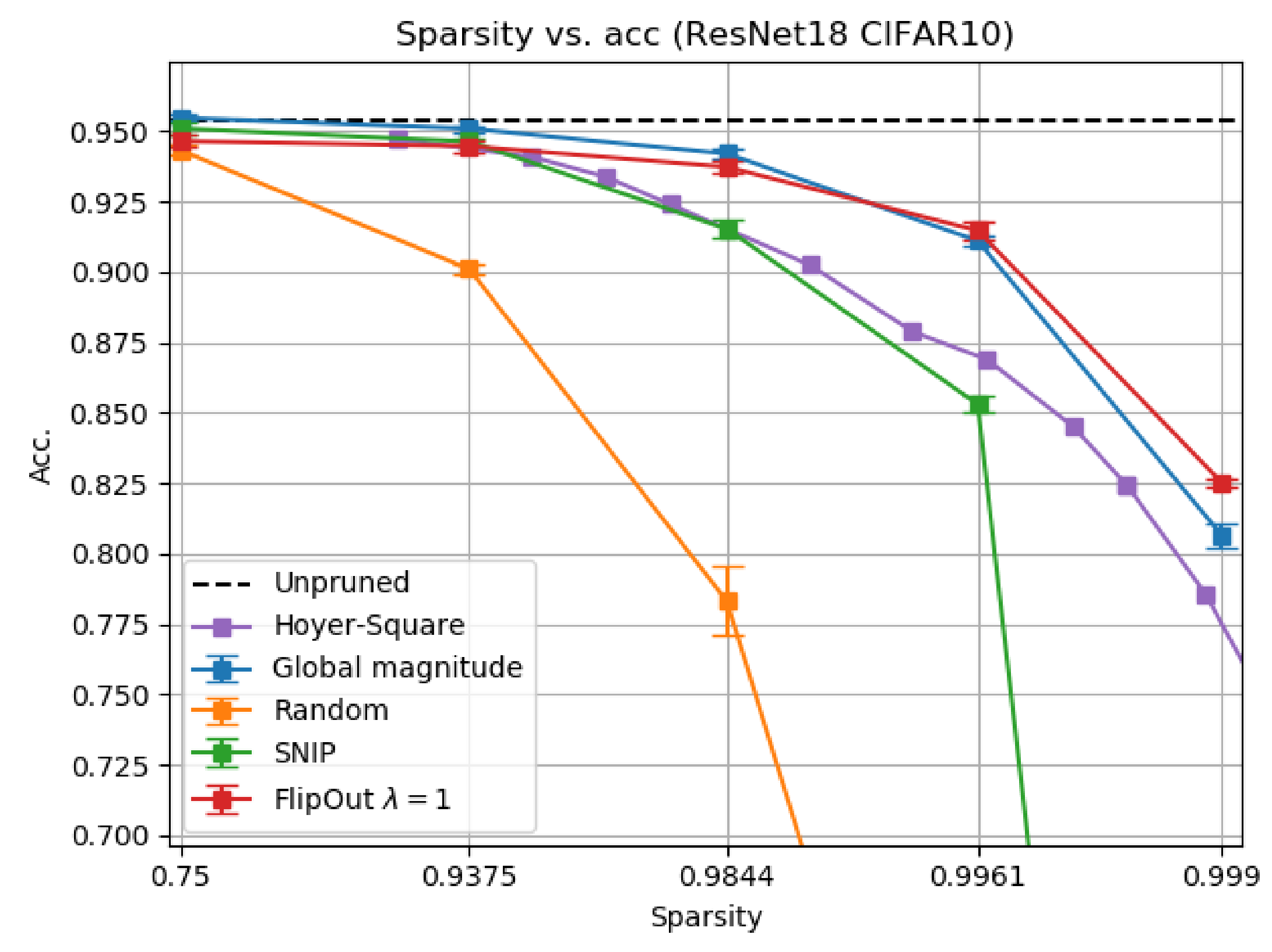

Fig. 2: Comparing FlipOut to baseline methods.

Default hyperparameter values

We have experimented with different values of the two hyperparameters and found that both p (Eq. 1) and λ (Eq. 2) have default values which offer consistent, strong results for all networks tested. Below, we compare the optimal value of λ as found by grid search to a default value $\lambda = 1$. Note that the differences are negligible at 93.75% sparsity, with the gap increasing at a higher sparsity level, but still remaining within 2 percentage points.

	Acc. at sparsity 93.75%		Acc. at sparsity 99.9%	
Model	λ^*	$\lambda = 1$	λ^*	$\lambda = 1$
ResNet18	94.58(+0.02)	94.56	83.75(+1.68)	82.07
VGG19	93.07(+0.11)	92.96	87.72(+0.48)	87.24
DenseNet121	89.75(+0.0)	89.75	73.5(+1.45)	72.05

We perform similar experiments for p on five values, $p \in \{0, \frac{1}{2}, 1, 2, 4\}$ and find that $p = 2$ generates optimal results in almost all cases, with a single exception for ResNet18 at the higher sparsity level, where $p = 4$ performs slightly better, with a 1 percentage point difference.

	Acc. at sparsity 93.75%					Acc. at sparsity 99.9%				
Model	$p=0$	$p=\frac{1}{2}$	$p=1$	$p=2$	$p=4$	$p=0$	$p=\frac{1}{2}$	$p=1$	$p=2$	$p=4$
ResNet18	93.71	88.39	94.18	**94.26**	94.11	72.69	77.08	79.83	82.07	**83.1**
VGG19	91.68	82.44	92.56	**92.96**	92.57	81.48	80.69	86.01	**87.24**	86.6
DenseNet121	10.35	77.40	88.9	**89.75**	88.86	10.35	10.35	70.85	**72.05**	60.5

Discussion

We introduce the aim test, a general method for determining if a point represents a local optimum for a weight during training, and propose using it for pruning by applying the test for all weights simultaneously and framing it as a saliency criterion. This method, coined FlipOut, has several desirable qualities: it is computationally tractable, allows for an exact level of sparsity to be selected, requires a single training run and has default hyperparameter settings which generate near optimal results, easing the burden of hyperparameter search. Experiments suggest that our method is on par with or even exceeds relevant baselines from literature.

References

[1] Jonathan Frankle and Michael Carbin. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *ICLR*. 2019.

[2] Song Han et al. "Learning both weights and connections for efficient neural network". In: *NeurIPS*. 2015, pp. 1135–1143.

[3] Babak Hassibi and David G Stork. "Second order derivatives for network pruning: Optimal brain surgeon". In: *NeurIPS*. 1993, pp. 164–171.

[4] Yann LeCun, John S. Denker, and Sara A. Solla. "Optimal Brain Damage". In: *NeurIPS*. Ed. by D. S. Touretzky. Morgan-Kaufmann, 1990, pp. 598–605.

[5] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. "SNIP: Single-shot Network Pruning based on Connection Sensitivity". In: *ICLR*. 2019.

[6] Junjie Liu et al. "Dynamic Sparse Training: Find Efficient Sparse Network From Scratch With Trainable Masked Layers". In: *ICLR*. 2020.

[7] Christos Louizos, Max Welling, and Diederik P. Kingma. "Learning Sparse Neural Networks through L_0 Regularization". In: *ICLR*. 2018.

[8] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. "Variational Dropout Sparsifies Deep Neural Networks". In: *ICML*. 2017.

[9] Arvind Neelakantan et al. "Adding Gradient Noise Improves Learning for Very Deep Networks". In: *arXiv e-prints* (Nov. 2015).

[10] Huanrui Yang, Wei Wen, and Hai Li. "DeepHoyer: Learning Sparser Neural Network with Differentiable Scale-Invariant Sparsity Measures". In: *ICLR*. 2020.