FlipOut: Uncovering Redundant Weights via Sign Flipping*

Andrei C. Apostol[1,2,3], Maarten C. Stol[2], and Patrick Forré[1]

[1] Informatics Institute, University of Amsterdam, The Netherlands
[2] BrainCreators B.V., Amsterdam, The Netherlands
[3] apostol.andrei@braincreators.com

Abstract. We propose a novel pruning method which uses the oscillations around 0 (i.e. sign flips) that a weight has undergone during training in order to determine its saliency. Our method can perform pruning before the network has converged, requires little tuning effort due to having good default values for its hyperparameters, and can directly target the level of sparsity desired by the user. Our experiments, performed on a variety of object classification architectures, show that it is competitive with existing methods and achieves state-of-the-art performance for levels of sparsity of 99.6% and above for 2 out of 3 of the architectures tested. For reproducibility, we release our code publicly at https://github.com/AndreiXYZ/flipout.

Keywords: deep learning · network pruning · computer vision.

1 Introduction

The success of deep learning is motivated by competitive results on a wide range of tasks ([3,9,24]). However, well-performing neural networks often come with the drawback of a large number of parameters, which increases the computational and memory requirements for training and inference. This poses a challenge for deployment on embedded devices, which are often resource-constrained, as well as for use in time sensitive applications, such as autonomous driving or crowd monitoring. Moreover, costs and carbon dioxide emissions associated with training these large networks have reached alarming rates ([21]). To this end, pruning has been proven as an effective way of making neural networks run more efficiently ([5,6,13,15,18]).

Early works ([6,13]) have focused on using the second-order derivative to detect which weights to remove with minimal impact on performance. However, these methods either require strong assumptions about the properties of the Hessian, which are typically violated in practice, or are intractable to run on modern neural networks due to the computations involved.

One could instead prune the weights whose optimum lies at or close to 0 anyway. Building on this idea, the authors of [5] propose training a network until

* Supported by BrainCreators B.V.

convergence, pruning the weights whose magnitudes are below a set threshold, and allowing the network to re-train, a process which can be repeated iteratively. This method is improved on in [4], whereby the authors additionally reset the remaining weights to their values at initialization after a pruning step. Yet, these methods require re-training the network until convergence multiple times, which can be a time consuming process.

Recent alternatives either rely on methods typically used for regularization ([17,18,26]) or introduce a learnable threshold, below which all weights are pruned ([16]). All these methods, however, require extensive hyperparameter tuning in order to obtain a favorable accuracy-sparsity trade-off. Moreover, the final sparsity of the resulting network cannot be predicted given a particular choice of these hyperparameters. These two issues often translate into the fact that the practitioner has to run these methods multiple times when applying them to novel tasks.

To summarize, we have seen that the pruning methods presented so far suffer from one or more of the following problems: (1) computational intractability, (2) having to train the network to convergence multiple times, (3) requiring extensive hyperparameter tuning for optimal performance and (4) inability to target a specific final sparsity.

We note that by using a heuristic in order to determine during training whether a weight has a locally optimal value of low magnitude, pruning can be performed before the network reaches convergence, unlike the method proposed by the authors of [5]. We propose one such heuristic, coined *the aim test*, which determines whether a value represents a local optimum for a weight by monitoring the number of times that weight oscillates around it during training, while also taking into account the distance between the two. We then show that this can be applied to network 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.

Our experiments, conducted on a variety of object classification architectures, indicate that it is competitive with respect to relevant pruning methods from literature, and can outperform them for sparsity levels of 99.6% and above. Moreover, we empirically show that our method has default hyperparameter settings which consistently generate near optimal results, easing the burden of tuning.

2 Method

2.1 Motivation

Mini-batch stochastic gradient descent ([2]) is the most commonly used optimization method in machine learning. Given a mini-batch of B randomly sampled training examples consisting of pairs of features and labels $\{(x_b; y_b)\}_{b=1}^{B}$, a neural network parameterised by a weight vector $\boldsymbol{\theta}$, a loss objective $L(\boldsymbol{\theta}; \boldsymbol{x}; \boldsymbol{y})$ and a

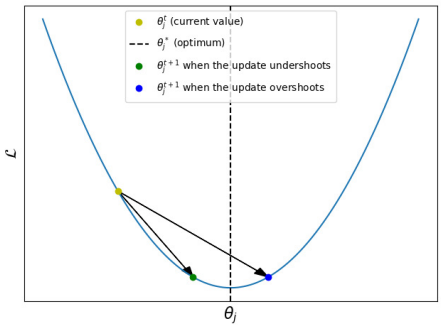


Fig. 1: Over- and under-shooting illustrated. The vertical line splits the x-axis into two regions relative to the (locally-)optimal value $_j$. Overshooting corresponds to when a weight gets updated such that its new value lies in the opposite region (blue dot), while undershooting occurs when the updated value is closer to the optimal value, but stays in the same region (green dot).

learning rate , the update rule of stochastic gradient descent is as follows:

$$\boldsymbol{g}^t = \frac{1}{B}\sum_{b=1}^{B} r \ _t L(\boldsymbol{\theta}^t; x_b; y_b)$$

$$\boldsymbol{\theta}^{t+1} \quad \boldsymbol{\theta}^t \quad \boldsymbol{g}^t$$

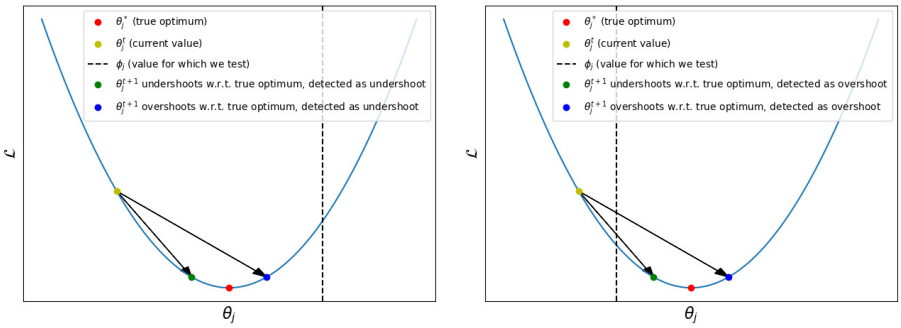
Given a weight $_j^t$, one could consider its possible values as being split into two regions, with a locally optimal value $_j$ as the separation point. Depending on the value of the gradient and the learning rate, the updated weight $_j^{t+1}$ will lie in one of the two regions. That is, it will either get closer to its optimal value while remaining in the same region as before or it will be updated past it and land in the opposite region. We term these two phenomena under- and over-shooting, and provide an illustration in Fig. 1. Mathematically, they correspond to $|g_j^t| < |\ _j^t \quad _j|$ and $|g_j^t| > |\ _j^t \quad _j|$, respectively.

With the behavior of under- and over-shooting, one could construct a heuristic-based test in order to evaluate whether a weight has a local optimum at a specific point without needing the network to have reached convergence:

1. For a weight $_j$, a value of $_j$ is chosen for which the test is conducted
2. Train the model regularly and record the occurrence of under- and over-shooting around $_j$ after each step of SGD
3. If the number of such occurrences exceeds a threshold , conclude that $_j$ has a local optimum at $_j$ (i.e. $_j = \ _j$)

We coin this method *the aim test*.

Previous works have demonstrated that neural networks can tolerate high levels of sparsity with negligible deterioration in performance ([4,5,16,18]). It is then reasonable to assume that for a large number of weights, there exist local



(a) Deceitful observations of under-shooting.(b) Deceitful observations of over-shooting.

Fig. 2: In the plots above, the dotted vertical line represents the value at which the aim test is conducted (i.e. a value we would like to determine as a local optimum or not), while the red dot represents the value of a true local optimum. When testing for a value which is not a locally optimal value $_j$ 6= $_j$, over- or under-shooting around $_j$ can be merely a side-effect of that weight getting updated towards its true optimum $_j$. These observations would then contribute towards the aim test returning a false positive outcome, i.e. $_j =$ $_j$. Whether we observe an over-shoot or an under-shoot in this case depends on the relationship between $_j$ and $_j$. In (a), we have $_j >$ $_j$, where if the hypothesised and true optimum are sufficiently far apart, we observe an under-shoot. Conversely, in (b), we have $_j <$ $_j$ and observe over-shooting.

optima at exactly 0, i.e. $_j = 0$. One could then use the aim test to detect these weights and prune them. Importantly, when using the aim test for $_j = 0$, the two regions around the tested value are the set of negative and positive real numbers, respectively. Checking for over-shooting then becomes equivalent to testing whether the sign of $_j$ has changed after a step of SGD, while under-shooting can be detected when a weight has been updated to a smaller absolute value and retained its sign, i.e. $(j\ _j^{t+1} j < j\ _j^t j) \wedge (\text{sgn}(\ _j^t) = \text{sgn}(\ _j^{t+1}))$.

However, under-shooting can be problematic; for instance, a weight could be updated to a lower magnitude, while at the same time being far from 0. This can happen when a weight is approaching a non-zero local optimum, an occurrence which should not contribute towards a positive outcome of the aim test. By positive outcome, we refer to determining that $_j = 0$ is indeed a local optimum of $_j$. A similar problem can occur for over-shooting, where a weight receives a large update that causes it to change its sign but not lie in the vicinity of 0. These scenarios, which we will refer to as *deceitful shots* going forward, are illustrated in the general case, where $_j$ can take any value, in Fig. 2a and Fig. 2b. Following, we make two observations which help circumvent this problem.

Firstly, one could reduce the impact of deceitful shots by also taking into account the distance of the weight to the hypothesised local optimum, i.e. $j\ _j \quad _j j$, when conducting the aim test. In other words, the number of occurrences of under-

and over-shooting should be weighed inversely proportional to this quantity, even if they would otherwise exceed .

Our second observation is that by ignoring updates which are not in the vicinity of $_j$, the number of deceitful shots are reduced. In doing so, one could also simplify the aim test; with a sufficiently large perturbation to $_j$, an update that might otherwise cause under-shooting can be made to cause over-shooting. Adding a perturbation of is, in effect, inducing a boundary around the tested value, [$_j$; $_j$ +]; all weights that get updated such that they fall into that boundary will be said to over-shoot around $_j$. With this framework, checking for over-shooting is sufficient; updates that under-shoot and are within of the tested value are made to over-shoot (Fig. 3a) and updates which under-shoot but are not in the vicinity of $_j$, i.e. a deceitful shot, are now not recorded at all (Fig. 3b). This can also be seen as restricting the aim test to only operate within a vicinity around $_j$.

2.2 FlipOut: applying the aim test for pruning

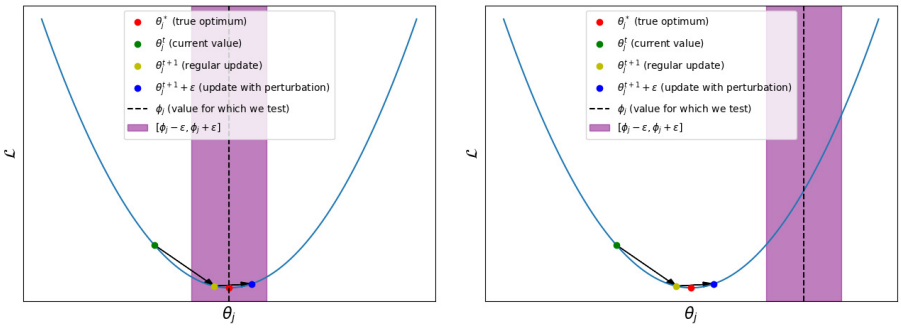
Determining which weights to prune Pruning weights that have local optima at or around 0 can obtain a high level of sparsity with minimal degradation in accuracy. The authors of [5] use the magnitude of the weights once the network is converged as a criterion; that is, the weights with the lowest absolute value (i.e. closest to 0) get pruned. The aim test can be used to detect whether a point represents a local optimum for a weight and can be applied before the network reaches convergence, during training. For pruning, one could then apply the aim test simultaneously for all weights with $\phi = \mathbf{0}$. We propose framing this as a saliency score; at time step t, the saliency ^t_j of a weight ^t_j is:

$$^t_j = \frac{j \; ^t_j j^p}{\text{flips}^t_j} \tag{1a}$$

$$\text{flips}^t_j = \sum_{i=0}^{t} \frac{1}{}[\text{sgn}(\; ^i_j) \; \cancel{6}= \text{sgn}(\; ^{i+1}_j)] \tag{1b}$$

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 when $_j = 0$ (Eq. 1b); a scheme for adding such perturbation is described in Section 2.2. In Equation 1a, the denominator $j \; ^t_j j^p$ represents the proximity of the weight to the hypothesised local optimum, $j \; ^t_j \quad _j j^p$ (which is equivalent to the weight's magnitude since we have $_j = 0$ for all weights). The hyperparameter p controls how much this quantity is weighted relative to the number of sign flips.

When determining the amount of parameters to be pruned, we adopt the strategy from [4], i.e. pruning a percentage of the remaining weights each time, which allows us to target an exact level of sparsity. Given m, the number of times pruning is performed, r the percentage of remaining weights which are removed at each pruning step, k the total number of training steps, d the dimensionality



(a) Under-shooting can become over-shooting by adding perturbation.

(b) Ignoring deceitful shots.

Fig. 3: (a) All weights that under-shoot but are within of $_j$ will be made to over-shoot. (b) When testing at a value which is not a local optimum for $_j$, i.e. $_j$ 6= $_j$ and adding a perturbation to $_j$, not taking under-shooting into account means that if the weight gets updated such that it does not lie in the boundary around $_j$ induced by the perturbation, an event that would otherwise contribute to a false positive outcome for the aim test will not be recorded, so the likelihood of rejecting $_j$ as an optimum increases.

of the weights and jj jj_0 the L_0-norm, the resulting sparsity s of the weight tensor after training the network is simply:

$$s = 1 \quad \frac{jj\theta^k jj_0}{d} = (1 \quad r)^m \tag{2}$$

This final sparsity can then be determined by setting m and r appropriately.

Perturbation through gradient noise Adding gradient noise has been shown to be effective for optimization ([19,25]) in that it can help lower the training loss and reduce overfitting by encouraging an exploration in the parameter space, thus effectively acting as a regularizer. While the benefits of this method are helpful, our motivation for its usage stems from allowing the aim test to be performed in a simpler manner; weights that get updated closer to 0 will occasionally pass over the axis due to the injected noise, thus making checking for over-shooting sufficient. We scale the variance of the noise distribution by the L_2 norm of the parameters θ, normalize it by the number of weights and introduce a hyperparameter which scales the amount of noise added into the gradients. For a layer l and d_l its dimensionality, the gradient for the weights in

that layer used by SGD for updates will be:

$$\hat{\boldsymbol{g}}^{t,l} \quad \boldsymbol{g}^{t,l} + \quad \boldsymbol{\epsilon}^{t,l} \tag{3a}$$

$$\boldsymbol{\epsilon}^{t,l} \quad \mathcal{N}(0;\ {}^{2}_{t,l}) \tag{3b}$$

$${}^{2}_{t,l} = \frac{k\boldsymbol{\theta}^{t,l}k_2^2}{d_l} \tag{3c}$$

As training is performed, it is desirable to reduce the amount of added noise so that the network can successfully converge. Previous works use annealing schedules by decaying the variance of the Gaussian distribution proportional to the current time step. Under our proposed formulation, however, explicitly using an annealing schema is not necessary. By pruning weights, the term in the numerator in Eq. 3c decreases, while the denominator remains constant. This ensures that annealing will be induced automatically through the pruning process, and there is no need for manually constructing a schedule.

Pruning periodically throughout training according to the saliency score in Eq. 1a in conjunction with adding gradient noise into the weights forms the *FlipOut* pruning method.

3 Related work

3.1 Deep-R

In Deep-R ([1]), the authors split the weights of the neural network into two matrices, the connection parameter $_k$ and a constant sign s_k with s_k 2 f 1; +1g; the final weights of the network are then defined as $\boldsymbol{\theta}$ $\boldsymbol{s}$. The connections whose $_k$ is negative are inactive; whenever a connection changes its sign, it is turned dormant and another randomly sampled connection is re-activated, ensuring the same sparsity level is maintained throughout training. Gaussian noise is also injected into the gradients during training.

Two similarities with our method can be observed here, namely the fact that the authors also use sign flipping as a signal for pruning a weight, and the addition of Gaussian noise. However, our methods differ in that we do not impose a set level of sparsity throughout training; instead, we use the number of sign flips of a weight in order to determine its saliency, while in Deep-R a single sign flip is required for a weight to be removed. Our method of injecting noise into the gradients also differs in that it does not explicitly encode an annealing scheme, allowing for the pruning process itself to reduce the noise throughout training. Finally, in Deep-R, the network is initialized with a specific level of sparsity which is maintained throughout training, while our method prunes gradually.

3.2 Magnitude and uncertainty pruning

The M&U pruning criterion is proposed in [11]. Given a weight $_j$, its uncertainty estimate $\tilde{}_{\theta_j}$ and a parameter controlling the trade-off between magnitude and

uncertainty, the M&U criterion will evaluate the saliency of the weight as:

$$_j = \frac{j \ _j j}{+ \tilde{}_{\theta_j}}$$

Uncertainty is estimated as the standard deviation across the previous n values of that weight, via a process called pseudo-bootstraping. This criterion is a generalization of the Wald test, and is equivalent to it when $= 0$.

Our method is similar in that our saliency score also normalizes the weight's magnitude by a function of its past values. However, this method assumes asymptotic normality. While this is the case when using negative log-likelihood or an equivalent as the loss function, this property does not necessarily hold when using modified variants of the SGD estimator, such as Adam ([10]) or RMSprop ([22]). In contrast, FlipOut is not derived from the Wald test and does not make any assumptions about the weight distribution at convergence.

4 Experiments

4.1 General Setup

Baselines As baselines, we consider a slightly modified version of magnitude pruning ([5]) (Global magnitude), due to the similarity between its saliency criterion and that of our own method, SNIP ([14]) due to it being an easily applicable method which does not suffer from any of the issues that are commonly found in pruning methods (Section 1) and Hoyer-Square, as introduced in [26], for the state-of-the-art results that it has demonstrated. We also include random pruning (Random) as a control. For FlipOut, Global magnitude and Random, pruning is performed periodically throughout training. We compare these methods at five different compression ratios, chosen at regular log-intervals (Table 1); for Hoyer-square, the performance at those points is estimated by a sparsity-accuracy trade-off curve. Magnitude pruning, in its original formulation, performs pruning only once the network has reached convergence. However, employing this strategy can create a confounding variable: training time. Since we would like to compare all methods at equal training budgets, we have opted to simply perform pruning after a fixed number of epochs for these methods. Note that the training budget that we allocate allows all of the networks that we consider to reach convergence when trained without performing any pruning. We make an exception to this equal budget rule for Hoyer-Square, since it prunes after training and would otherwise not benefit from any SGD updates after sparsification. As such, we have performed an additional 150 epochs of fine-tuning without the regularizer, as per the original method, although we have observed negligible benefits to this. All baselines were modified to rank the weights globally when a pruning decision is made, as per the strategy from [4], in order to avoid creating bottleneck layers. The models that we test on are ResNet18 ([7]) and VGG19 ([20]) trained on the CIFAR-10 dataset ([12]), and DenseNet121 ([9]) trained on Imagenette ([8]).

Table 1: Compression ratios, resulting sparsity levels and prune frequencies used in the experiments, assuming 350 epochs of training and that 50% of the remaining weights are removed at each step.

Compression ratio ($\frac{d_{\boldsymbol{\theta}}}{jj\boldsymbol{\theta}jj_0}$)	Resulting sparsity ($1 - \frac{jj\boldsymbol{\theta}jj_0}{d_{\boldsymbol{\theta}}}$)	Epochs before pruning
2^2	75%	117
2^4	93.75%	70
2^6	98.43%	50
2^8	99.61%	39
2^{10}	99.9%	32

Hyperparameters The training parameters for all experiments are taken from [23]; specifically, we use a learning rate of 0:1, batch size of 128, 350 epochs of training and a weight decay penalty of 5e 4. The learning rate is decayed by a factor of 10 at epochs 150 and 250. The networks are trained with the SGD optimizer with a momentum value of 0:9 ([2]). For the methods that perform iterative pruning (Global magnitude, Random, FlipOut), we remove 50% of the remaining weights at each pruning step, with the pruning frequencies chosen such that the compression ratios from Table 1 are achieved; we use the same pruning rates and frequencies across all three methods. SNIP accepts a single hyperparameter, namely the desired final sparsity, which we have chosen such that it matches the aforementioned compression ratios. For Hoyer-Square, which does not allow for a specific level of sparsity to be chosen and, instead, relies on parameter tuning, we generate a sparsity-accuracy trade-off curve by using 15 different values for the regularization term, ranging from 1e 7 to 6e 3 with 3 values at each decimal point (e.g. 1e 7, 3e 7, 6e 7, 1e 6 etc.) and a fixed pruning threshold of 1e 4. Finally, for FlipOut, we use the values of $p = 2$ (Eq. 1) and = 1 (Eq. 3) for all experiments, a choice we motivate in Section 4.2.

4.2 Choosing the hyperparameters for FlipOut

We have experimented with different values of the two hyperparameters and found that $p = 2$ (Eq. 1a) and = 1 (Eq. 3a) offer consistent, strong results for all networks tested. In the following paragraphs, we detail the procedure used in determining these values.

Choosing λ For , we have run all networks at 15 different values, ranging from 0:75 to 1:5 in increments of 0:05. The value of $p = 2$ was used. The networks are evaluated on a validation set, created by removing a random subset of samples from the training set. The size of the validation set was 10000 for CIFAR10 and 2000 for Imagenette. For our subsequent experiments, (Sections 4.3 and 4.4), the networks have been trained on the full training set. As a metric, we have used the accuracy of the networks at the end of training for the sparsity levels of 93:75% and 99:9%. We provide in Table 2 the accuracies generated by the optimal

Table 2: Accuracies when using the best value of discovered by grid search and the value of = 1 at two levels of sparsity. The parantheses indicate the gain offered by the optimal parameter.

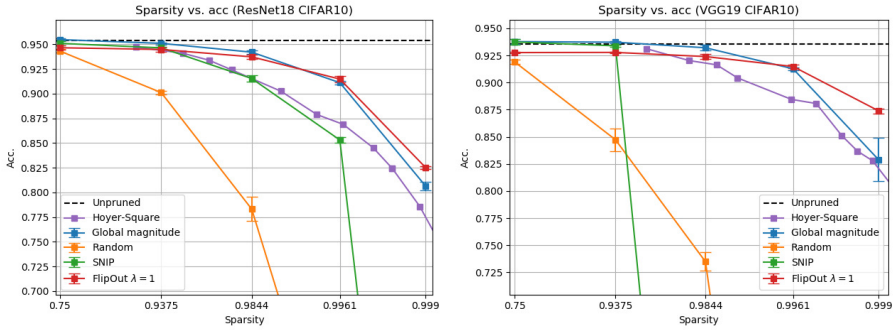
	Acc. at sparsity 93.75%		Acc. at sparsity 99.9%	
Model	λ	$\lambda = 1$	λ	$\lambda = 1$
ResNet18	94.58(+0.02)	94.56	83.75(+1.68)	82.07
VGG19	93.07(+0.11)	92.96	87.72(+0.48)	87.24
DenseNet121	89.75(+0.0)	89.75	73.5(+1.45)	72.05

Table 3: Table of results for different values of p at two levels of sparsity.

	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	**83.15**
VGG19	91.68	82.44	92.56	**92.96**	92.57	81.48	80.69	86.01	**87.24**	86.64
DenseNet121	10.35	77.40	88.9	**89.75**	88.86	10.35	10.35	70.85	**72.05**	60.55

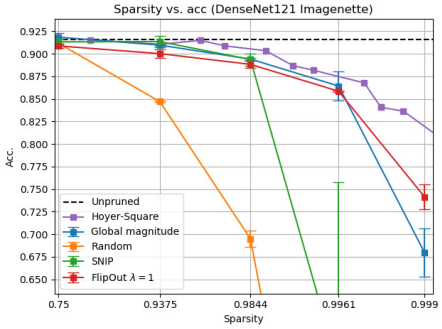
value of , as discovered through this process, and the ones generated at = 1. Notice that the differences are almost negligible at 93.75% sparsity. For the larger sparsity level the disparity increases, although the default value still remains within 2 percentage points of the optimum value for all networks considered. The largest gap can be seen for ResNet18 and DenseNet121, at approximately 1.7 and 1.5 percentage points, respectively. Since there are only two out of six cases in which optimizing has helped beyond a negligible amount, we have used the value of 1 for this hyperparameter throughout our experiments.

Choosing p We perform similar experiments for p on five values, $p \in \{0, \frac{1}{2}, 1, 2, 4\}$. Note that the value of $p = 0$ corresponds to the case when the magnitudes of the weights are not taken into account; that is, the pruning decisions will be made solely based on the number of sign flips. As can be seen in Table 3, the value of $p = 2$ consistently outperforms all other tested values, with the exception of ResNet18 at 99.9% sparsity, for which the value of $p = 4$ achieves better results by approximately 1 percentage point. Another interesting observation is that the values of 1, 2 and 4 tend to perform better than 0 and $\frac{1}{2}$; we conjecture that this is due to the fact that deceitful shots (Section 2.1) occur when not taking into account the distance between the weight and its hypothesised local optimum, which have a negative impact on the pruning decision. This can be especially observed at the higher sparsity level and in the case of DenseNet121, where pruning with $p = 0$ causes the network to not perform better than random guessing. Given that the value of $p = 2$ is favored in 5 out of 6 cases, we have decided to use it as a default value in our subsequent experiments.



(a) ResNet18 on CIFAR10

(b) VGG19 on CIFAR10



(c) DenseNet121 on Imagenette

Fig. 4: Results of pruning ResNet18 and VGG19 on the CIFAR10 dataset. Each point is averaged over 3 runs; error bars indicate standard deviation.

4.3 Comparison to baselines

The results for the three models tested are found in Figure 4. FlipOut obtains state-of-the-art performance on ResNet18 and VGG19 for sparsity levels of 99.61% and beyond. For the highest tested sparsity level, it outperforms the second-best method by 1.9 and 4.5 percentage points, respectively (Fig. 4a, 4b). Notably, when using FlipOut on VGG19 for this sparsity, the drop in accuracy compared to the unpruned model is only 6.2 percentage points. At the same time, it remains competitive with other baselines for lower degrees of sparsity, staying within a 1 percentage point difference compared to the best method and with a minimal drop relative to the unpruned model. For DenseNet121, however, Hoyer-Square dominates all other methods tested in most cases (Fig. 4c), with FlipOut as second best for the highest sparsity level.

Interestingly, the simple criterion of magnitude pruning, when modified to rank the weights globally instead of a layer-by-layer basis, is competitive with other, more recent, baselines, and even obtains state-of-the-art results for moderate levels of sparsity. However, at high levels of sparsity, which correspond to more

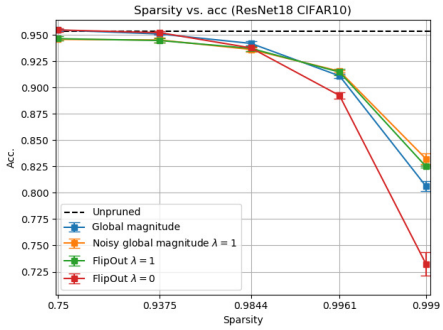
frequent and implicitly earlier pruning steps (Table 1) there is a performance degradation. This suggests that the magnitude of a weight by itself is not a good measure of saliency when the network is far from reaching convergence. It is also worth noting that SNIP collapses at high levels of sparsity, causing the network to perform no better than random guessing. Upon inspecting these cases (not shown for visibility) we noticed that at least one layer has been entirely pruned, effectively blocking any signal from passing. Interestingly, this does not happen for any of the other baselines (except for Random). We conjecture that this collapse as well as the cases where SNIP performs worse than random pruning (Fig. 4b) are a result of pruning at initialization; pruning too too early can cause the saliency criterion to be inaccurate, but also impedes training in and of itself.

During our experiments, we empirically observed that Hoyer-Square requires extensive hyperparameter tuning for optimal performance. Our method, however, has strong default values and can also target the final sparsity directly, while also not requiring additional epochs of fine-tuning. Finally, SNIP, the only other baseline which does not suffer from any of the issues commonly found among pruning methods (Section 1) compromises on performance for high levels of sparsity, whereas FlipOut does not.

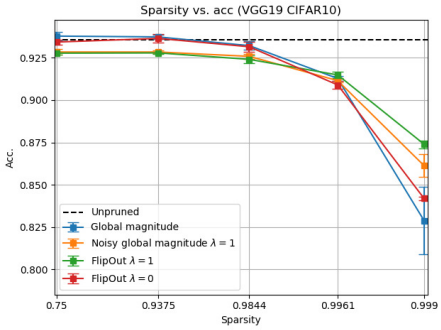
4.4 Is it just the noise?

The performance of FlipOut could simply be a result of the noise addition, which is known to aid optimization ([19,25]). To investigate this, we perform experiments with global magnitude as the pruning criterion in which we add noise into the gradients using the recipe from Equation 3c and compare it to our own method. Notably, the saliency criterion of these two methods differ only in that FlipOut normalizes the magnitude by the number of sign flips (denominator in Eq. 1a). The hyperparameters were kept at their default values of $p = 2$ for FlipOut and $= 1$ for both methods. We also include runs of FlipOut where no noise was added (i.e. $= 0$). These serve as a control, decoupling the two novel components of our method: noise addition and scaling magnitudes by the number of sign flips. The same pruning rates and frequency of pruning steps have been used as before (Table 1). The results are illustrated in Fig. 5.

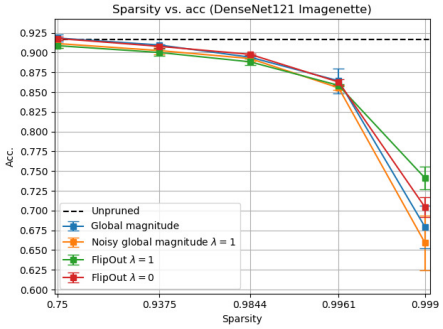
For sparsity levels up to 98.44%, adding gradient noise causes a slight deterioration on performance, as can be seen by the fact that both global magnitude and FlipOut with $= 0$ outperform their noisy counterparts. It can also be seen that FlipOut with $= 1$ performs comparably to noisy global magnitude, indicating that measuring saliency by sign flips does not benefit accuracy in these regimes compared to using only the magnitude, and the performance gap between the noisy and non-noisy methods is likely a result of noise addition. For sparsity levels of 99.61% and above, however, the opposite is true. It seems that gradient noise disproportionately benefits networks with a small number of remaining parameters; we conjecture that this is due to the fact that the exploration in parameter space induced by noise is more effective when that space is heavily constrained. Focusing on the highest level of sparsity, FlipOut outperforms noisy global magnitude on VGG19 (Fig. 5b) and DenseNet121 (Fig. 5c) by 1.2 and



(a) ResNet18 on CIFAR10



(b) VGG19 on CIFAR10



(c) DenseNet121 on Imagenette

Fig. 5: Results of the ablation study on the noise. Each point is averaged over 3 runs. Global magnitude without adding noise is also shown for comparison.

8.2 percentage points, respectively, while being outperformed by 0.8 percentage points on ResNet18 (Fig. 5a). The standard deviation of FlipOut at this point is lower than for noisy global magnitude for all networks tested, making it more robust to initial conditions and the noise sampling process. At this level, the addition of gradient noise to FlipOut also shows performance boosts compared to its non-noisy counterpart, namely 9.3 percentage points for ResNet18, 3.2 for VGG19 and 3.7 for DenseNet121. The benefits caused by adding noise to global magnitude as compared to adding it to FlipOut are similar for VGG19; however, it is relatively small for ResNet18 at 2.6 percentage points and even causes a 2 percentage point drop in performance for DenseNet121.

Since FlipOut with $\quad = 1$ outperforms noisy global magnitude in 2 out of 3 cases for the highest level of sparsity while maintaining similar performance in all other cases as well as being less sensitive to the choice of seed, we conclude that its results cannot be explained only by the addition of noise and is also caused by the sign flips being taken into account when computing saliency.

Additionally, we conjecture that occurrences of under-shooting are indeed converted into over-shooting when adding gradient noise, allowing FlipOut to more accurately compute saliencies. This is evidenced by the fact that gradient noise addition benefits FlipOut more so than it does global magnitude, and implies that our method of dealing with deceitful shots is sound.

5 Discussion

In this work, we introduce the aim test, a general method for determining whether 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, demonstrates 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.

We compare the performance of FlipOut to relevant baselines from literature on a variety of object classification architectures. We show that it achieves state-of-the-art performance at the highest levels of sparsity tested for 2 out of 3 networks, and maintains competitive performance in less sparse regimes. Finally, we conduct an ablation study on the two components of our algorithm, gradient noise addition and the saliency criterion, and find that both play an important role in yielding this performance performance.

References

1. Bellec, G., Kappel, D., Maass, W., Legenstein, R.: Deep rewiring: Training very sparse deep networks. In: International Conference on Learning Representations (2018), https://openreview.net/forum?id=BJ_wN01C-
2. Bottou, L.: Online algorithms and stochastic approximations. In: Saad, D. (ed.) Online Learning and Neural Networks. Cambridge University Press, Cambridge, UK (1998), http://leon.bottou.org/papers/bottou-98x, revised, oct 2012
3. Brock, A., Donahue, J., Simonyan, K.: Large scale GAN training for high fidelity natural image synthesis. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=B1xsqj09Fm
4. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=rJl-b3RcF7
5. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in neural information processing systems. pp. 1135–1143 (2015)
6. Hassibi, B., Stork, D.G.: Second order derivatives for network pruning: Optimal brain surgeon. In: Advances in neural information processing systems. pp. 164–171 (1993)
7. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)

8. Howard, J.: Imagenette (2019), accessed April 6th, 2020 at https://github.com/fastai/imagenette/tree/6395a747bef7a9760b95cd582ece09d90f8a4769
9. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4700–4708 (2017)
10. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. International Conference on Learning Representations (12 2015)
11. Ko, V., Oehmcke, S., Gieseke, F.: Magnitude and uncertainty pruning criterion for neural networks. In: 2019 IEEE International Conference on Big Data (Big Data). pp. 2317–2326 (2019)
12. Krizhevsky, A.: Learning multiple layers of features from tiny images (2009)
13. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Touretzky, D.S. (ed.) Advances in Neural Information Processing Systems 2, pp. 598–605. Morgan-Kaufmann (1990), http://papers.nips.cc/paper/250-optimal-brain-damage.pdf
14. Lee, N., Ajanthan, T., Torr, P.: Snip: Single-shot network pruning based on connection sensitivity. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=B1VZqjAcYX
15. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. In: International Conference on Learning Representations (2017)
16. Liu, J., Xu, Z., Shi, R., Cheung, R.C.C., So, H.K.: Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. In: International Conference on Learning Representations (2020), https://openreview.net/forum?id=SJlbGJrtDB
17. Louizos, C., Welling, M., Kingma, D.P.: Learning sparse neural networks through l_0 regularization. In: International Conference on Learning Representations (2018), https://openreview.net/forum?id=H1Y8hhg0b
18. Molchanov, D., Ashukha, A., Vetrov, D.: Variational dropout sparsifies deep neural networks. In: Proceedings of the 34th International Conference on Machine Learning (2017)
19. Neelakantan, A., Vilnis, L., Le, Q.V., Sutskever, I., Kaiser, L., Kurach, K., Martens, J.: Adding Gradient Noise Improves Learning for Very Deep Networks. arXiv e-prints arXiv:1511.06807 (Nov 2015)
20. Simonyan, K., Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. In: International Conference on Learning Representations (2015)
21. Strubell, E., Ganesh, A., McCallum, A.: Energy and policy considerations for deep learning in nlp. In: ACL (2019)
22. Tieleman, T., Hinton, G.: "Lecture 6.5—rmsprop: Divide the gradient by a running average of its recent magnitude". COURSERA: Neural Networks for Machine Learning (2012)
23. Train CIFAR10 with PyTorch (GitHub Repository), Unknown Author: Pytorch cifar-10 github repository (2017), accessed April 6th, 2020 at https://github.com/kuangliu/pytorch-cifar/tree/ab908327d44bf9b1d22cd333a4466e85083d3f21
24. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017)
25. Welling, M., Teh, Y.W.: Bayesian learning via stochastic gradient langevin dynamics. In: Proceedings of the 28th international conference on machine learning (ICML-11). pp. 681–688 (2011)
26. Yang, H., Wen, W., Li, H.: Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. In: International Conference on Learning Representations (2020), https://openreview.net/forum?id=rylBK34FDS