A DEMON AT WORK: LEVERAGING NEURON DEATH FOR EFFICIENT NEURAL NETWORK PRUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

When training deep neural networks, the phenomenon of 'dying neurons' —units that become inactive and output zero throughout training—has traditionally been viewed as undesirable, linked with optimization challenges, and contributing to plasticity loss in continual learning scenarios. In this paper, we reassess this phenomenon through the lens of network sparsity and pruning. By systematically exploring the influence of various hyperparameter configurations on the occurrence of dying neurons, we unveil their potential to facilitate simple yet effective structured pruning algorithms. We introduce 'Demon's Pruning' (DemP), a method that controls the proliferation of dead neurons, dynamically sparsifying neural networks as training progresses. Remarkably, our approach, characterized by its simplicity and broad applicability, outperforms existing structured pruning techniques, while achieving results comparable to prevalent unstructured pruning methods. These findings pave the way for leveraging dying neurons as a valuable resource for efficient model compression and optimization.

1 INTRODUCTION

Dying neurons, a phenomenon frequently observed during the learning process of neural networks, are traditionally viewed as detrimental, often leading to suboptimal performance (Maas et al., 2013; Xu et al., 2015) or loss of plasticity, especially in non-stationary settings (Lyle et al., 2023; Abbas et al., 2023). In response, alternative activation functions without a hard-saturated state, such as Leaky ReLU (Maas et al., 2013), Swish (Ramachandran et al., 2018), GELU (Hendrycks & Gimpel, 2016), have been proposed.

In this work, we reexamine the phenomenon of dying neurons through the lens of network sparsity and pruning. Building upon both intuitive and theoretical insights into neuron death within networks trained using stochastic optimization methods, we demonstrate how varying hyperparameters such as learning rate, batch size, and L2 regularization parameter influence the occurrence of dead neurons during training. We present and validate a method for actively managing the emergence of dead units and for dynamically pruning them throughout the training process.

Notably, we observe that a higher level of noise or stronger regularization leads to sparser solutions, characterized by a higher number of dead neurons. Capitalizing on the simplicity of our pruning criterion –removing the inactive neurons– we introduce at no additional cost a structured pruning method, *Demon Pruning* (DemP), both performant and easy to implement. DemP can be seamlessly integrated into any training algorithm and readily combined with existing pruning techniques.

DemP marks a significant departure from traditional methodologies in pruning. Previous methods relied on heuristics-based interventions: training is paused to ablate weights (or neurons) based on a specific criterion. Training then resumes and tries to recover from the intervention. In contrast, DemP is the first instantiation of a potential family of algorithms that leverage insights into how the interplay between stochasticity and sparsity affects learning dynamics. With DemP, the optimization process directly leads to sparse networks, removing the need for direct interventions during training. Moreover, because the neurons removed by DemP in ReLU networks were inactive, the learning dynamics are not impacted by the pruning procedure, removing the need for recovery.

Structured pruning methods, even in the absence of specialized sparse computation primitives (Elsen et al., 2020; Gale et al., 2020), can more effectively exploit the computational advantages of GPU

hardware (Wen et al., 2016) compared to unstructured methods. This becomes particularly crucial as deep learning models continue to grow; as considerations for environmental impacts become increasingly significant (Strubell et al., 2019; Lacoste et al., 2019; Henderson et al., 2020), developing methods with reduced energy footprint that can be adopted widely is becoming fundamental.

Our main contributions are:

1. **Analysis of Neuron Mortality**: We provide insights into the mechanisms underlying neuron death, highlighting the pivotal role of stochasticity, as well as the influence of varying hyperparameters such as learning rate, batch size, and regularization parameters (Section 3).
2. **A Structured Pruning Method.** Leveraging our insights, we introduce DemP, a novel pruning approach that both promotes the proliferation of dead neurons in a controlled way, and removes dead neurons in real time as they arise during training, offering substantial training speedups (Section 4).
3. **Empirical Evaluation.** Through extensive experiments on various benchmarks, we demonstrate that DemP, despite its simplicity and broad applicability, surpasses existing structured pruning methods in terms of accuracy-compression tradeoffs, while achieving comparable results to prevalent unstructured pruning methods (Section 5).

2 Related Works

Dead Neurons and Capacity Loss. It is widely recognized that neurons, especially in ReLU networks, can die during training (Agarap, 2018; Trottier et al., 2017; Lu et al., 2019). In particular, Evci (2018) noted the connection of the dying rate with the learning rate and derived a pruning technique from it.

More recently, dead neurons were studied in continual and reinforcement learning through the lens of *plasticity loss* (Berariu et al., 2021; Lyle et al., 2022), that progressively makes a model less capable of adapting to new tasks Kirkpatrick et al. (2016). The inability to adapt has also been observed in supervised learning (Ash & Adams, 2020).

In some scenarios, a cause of plasticity loss has been attributed to the accumulation of dead units (Sokar et al., 2023; Lyle et al., 2023; Abbas et al., 2023; Dohare et al., 2021). These works have shown that under rapid shifts in training distribution, neural network activations can collapse to a region where the gradient is 0. Although ReLU activation seems to amplify the phenomenon, it has also been observed for various activation functions that have a saturated regime (Dohare et al., 2021). Simple solutions such as resetting the dead units (Sokar et al., 2023) or concatenating ReLU activations (Abbas et al., 2023) have proven effective to mitigate the issue.

Sparsity Induced by Stochasticity. Work by Pesme et al. (2021); Vivien et al. (2022) studied the impact of Stochastic Gradient Descent's (SGD) noise on training, following empirical observations that SGD can be beneficial to generalization over Gradient Descent (GD) (Keskar et al., 2017; Masters & Luschi, 2018). The noise structure of SGD (Wojtowytsch, 2023; Pillaud-Vivien, 2022) plays a key role in their observations.

Pruning. Pruning is used to reduce the size and complexity of neural networks by removing redundant or less important elements, be they neurons or weights while maintaining their performance (LeCun et al., 1989). Recent advances such as those based on the Lottery Ticket Hypothesis (Frankle & Carbin, 2019) have demonstrated the existence of subnetworks trainable to comparable performance as their dense counterpart but with fewer parameters. Pruning techniques can broadly be categorized into two groups: structured pruning and unstructured pruning.

Structured pruning aims to remove entire structures within a network, such as channels, filters, or layers. It results in smaller and faster models that maintain compatibility with existing hardware accelerators and software libraries (Wen et al., 2016; Li et al., 2017). We highlight and benchmark against recent works that use criteria based on gradient flow to evaluate which nodes to prune (Verdenius et al., 2020; Wang et al., 2020; Rachwan et al., 2022). Other works employed either L0 or L1 regularization on gate parameters (or batch normalization scaling parameters) to enforce sparsity (Liu et al., 2017; Louizos et al., 2018; You et al., 2019), but we do not benchmark them as they are outperformed by Rachwan et al. (2022).

Unstructured pruning, on the other hand, focuses on removing individual weight from the network (LeCun et al., 1989; Han et al., 2016b). This approach often leads to higher compression rates but requires specialized hardware or software implementations for efficient execution due to the irregularity of the resulting sparse models (Han et al., 2016a). One notable method in unstructured pruning is magnitude-based pruning (Han et al., 2015), where weights with magnitudes below a certain threshold are removed. More recent approaches include dynamic sparse training methods such as RigL (Evci et al., 2020; Lasby et al., 2023) and SNFS (Dettmers & Zettlemoyer, 2019), which iteratively prune and regrow connections during training based on their importance.

Regularization-based pruning has been popular for both structured and unstructured pruning, with canonical papers employing L0 or L1 regularization to induce sparsity directly (Louizos et al., 2018; Liu et al., 2017; Ye et al., 2018) while L2 can help identify the connections to prune with the smallest weight criterion (Han et al., 2015). Because uniform regularization can quickly degrade performance (Wen et al., 2016; Lebedev & Lempitsky, 2016), Ding et al. (2018) and Wang et al. (2019) proposed to adapt the regularization for different parameter groups. Recently, Wang et al. (2021) showed that growing the L2 regularization can leverage Hessian information to identify the filters to prune in pre-trained networks.

3 Neural Death: An Analysis

In this section, we study the phenomenon of dead neurons accumulation during training in deep neural networks. Our aim is to provide theoretical insights into this phenomenon and investigate how various training heuristics and hyperparameters affect neuron mortality.

Given a deep neural network and a set of n training data samples, we denote by $a_j^\ell \in \mathbb{R}^n$ the vector of activations of the jth neuron in layer ℓ for each training input. We adopt the following definition of a "dead neuron" throughout the paper:

Definition: The j-th neuron in layer ℓ is *inactive* if it consistently outputs zero on the entire training set, i.e $a_j^\ell = 0$.[1] A neuron that becomes and remains inactive during training is considered as *dead*. [2]

Many modern architectures use activations functions with a saturation region that includes 0 at its boundary. In this case, when a neuron becomes inactive during training, its incoming weights also receive zero or very small gradients, which makes it difficult for the neuron to recover. In this paper, we mostly work with the Rectified Linear Unit (ReLU) activation function, $\sigma(x) = \max(0, x)$. In this case, the activity of a neuron depends on the sign of the corresponding pre-activation feature.

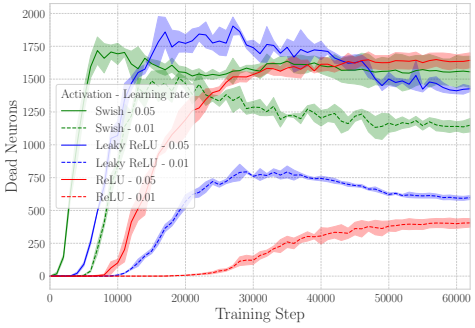


Figure 1: Dead neurons accumulation for a ResNet-18 trained on CIFAR-10.

The network with parameter $\boldsymbol{w}$ is trained to minimize the training loss $L(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell_i(\boldsymbol{w})$, where $\ell_i(\boldsymbol{w})$ is the loss function on sample i, using stochastic gradient descent (SGD) based methods. At each iteration, this requires an estimate of the loss gradient $g(\boldsymbol{w}) := \nabla L(\boldsymbol{w})$, obtained by computing the mean gradient on a random minibatch $b \subset \{1 \cdots n\}$. For simple SGD with learning rate η, the update rule takes the form

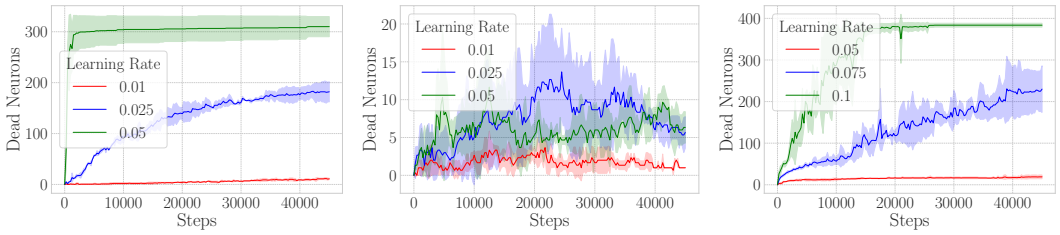
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \hat{g}(\boldsymbol{w}_t, b_t), \quad \hat{g}(\boldsymbol{w}, b) := \frac{1}{|b|}\sum_{i \in b} \nabla \ell_i(\boldsymbol{w}). \tag{1}$$

3.1 Neurons Die During Training

We begin with some empirical observations. Using the above definition with a fixed thresholding parameter ($\epsilon = 0.01$), we monitor the accumulation of dead neurons during training of a Resnet-18

[1] In practice, especially for non-ReLU activation functions, we will be using the notion of ϵ-inactivity, defined by the condition $|a_i^\ell| < \epsilon$, for some threshold parameter ϵ.

[2] For convolutional layers, we treat the filters as neurons, see Appendix A



(a) SGD noise (b) Gaussian noise (c) SGD

Figure 2: A 3-layer MLP trained over a subset on MNIST. (a) The noisy part of the minibatch gradient is isolated and used exclusively to update the NN. It shows that noisy updates are *sufficient* to kill a subset of neurons following standard initialization. Because SGD gradient is 0 for dead neurons, there is an asymmetry: only live neurons are subject to noisy updates. (b) In contrast, Gaussian noise does not share the same assymmetry as SGD noise and is much less prone to dead neuron accumulation (Gaussian noise can revive neurons, contrary to SGD noise). (c) Standard SGD. Dead neurons accumulate quickly in noisy settings, but they plateau when the NN converges (leading to zero gradient). Results are averaged over 3 seeds.

(He et al., 2016) on CIFAR-10 (Krizhevsky et al., 2009) with the Adam optimizer (Kingma & Ba, 2015), with various learning rates and different choices of activation functions. We use a negative slope of $\alpha = 0.05$ for Leaky ReLU, and a $\beta = 1$ for Swish.

Results are shown in Fig.1. We observe a sudden sharp increase in the number of inactive neurons at the beginning of training; few of these recover later in training (see Appendix C). Overall, this leads a significant portion of the 3904 neurons/filters in the convolutional layers of the ResNet-18 to die during training, especially with a high learning rate. Note that this phenomenon is not specific to ReLU activations.

Intuition. Similar to Maxwell's demon thought experiment (Maxwell, 1872), one can picture a playful being, ReLUcifer, overseeing a boundary in the weight space that demarcates active and inactive neuron regions. Neurons can move freely within the active zone, but entering the inactive region – where all movement is impeded – is a one-way process governed by ReLUcifer. If the neuron's movements include random components, a risk of inadvertent crossover appears. This risk would be influenced by various factors: noise from the data, being too close to the border, and taking imprudent gradient steps that are too large. Once in the inactive zone, neurons can only be reactivated if the boundary itself shifts. This asymmetry makes it more likely for neurons to die than to revive.

This analogy can be formalized as a biased random walk, an exercise that we touch upon in Appendix B. It also motivates further exploration into how the stochastic nature of various optimizers – related in particular to learning rate and batch size (He et al., 2019; Smith et al., 2018) — contributes to the accumulation of dead neurons in neural networks.

Role of noise. Although not all saturated units are due to noise, an important question is how much can noise contribute to neuron death. We argue that noisy training can significantly contribute to dead neuron accumulation. To verify that noise in itself is enough to kill neurons, we trained a 3 layers-deep MLP (of size 100-300-10) on a subset of 10 000 images of MNIST dataset. To isolate the noise from a minibatch (of size 1) gradient ($\hat{g}(\boldsymbol{w}_j^t)$) we deduced from it the full gradient ($g(\boldsymbol{w}_j^t)$).

Figure 2 shows that noise can indeed contribute to dead neuron accumulation and that we should not expect that every neuron dying during training did so because of their individual gradient pointing toward the dead region. We also compare with different noisy regimes to illustrate that the noise structure of SGD plays an important role in the final amount of dead neurons.

3.2 Training Hyperparameters Impact on Dying Ratios

We close this section by testing empirically some of the implications of the above discussions. The main goal is to quantify the impact of hyperparameters on the ratio of dead neurons. The setup is the same as in Section 3.1. Additional training details can be found J.1.

Learning rate and batch size. Our simple model expose a link between learning rate, batch size, and dying probability: by influencing the noise variance of the optimizer updates, they both should impact the ratios of dying neurons. This prediction proves accurate, as depicted in Fig. 3.

Regularization. Regularization is a popular strategy to control the volume of the solution space that ML optimizers can reach. It restrains the model capacity by favoring solutions with smaller norms, i.e. solutions that are closer to the origin. We remark that for a NN having ReLU activations, $\boldsymbol{w}_j^t = \mathbf{0}$ is a point that belongs to the dead region, likewise for the points where all parameters of a neuron are negative, ensuring $\mathrm{ReLU}(\boldsymbol{w}_j \boldsymbol{x}_l^T) = 0$ (since $x_l^i >= 0$, that is the layer inputs are always positive in ReLU networks).

Even if we do not know where the actual death border lies in parameter space, getting closer to the origin is expected to bring a neuron closer to it. According to our model, the neuron should become more likely to die by doing so. As such, regularization can also be an important factor influencing dead neuron accumulation, as empirically demonstrated in Fig. 10.

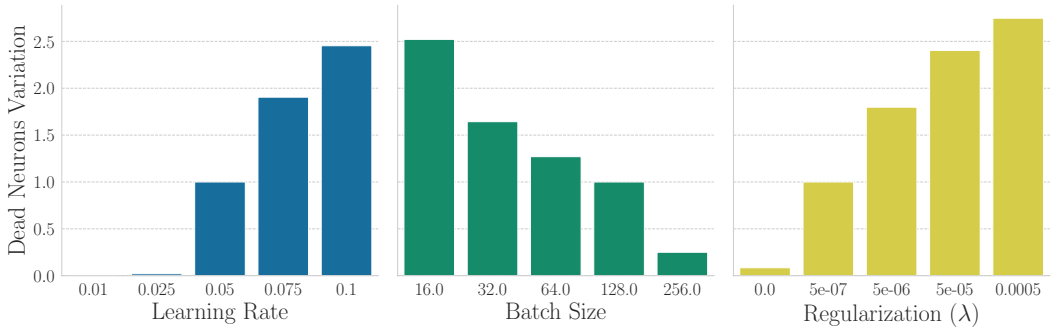


Figure 3: Varying the hyperparameters of a ResNet-18 (CIFAR-10) impacts the number of dead neurons. For the learning rate and batch size histogram, we varied around the combination learning rate 0.05, batch size 128 and λ=0, which on average led to 388 neurons at the end of training. The bar heights indicate the multiplicative ratio of dead neurons with respect to this base configuration. For regularization, we started with learning rate 0.005, batch size 128 and $\lambda = 5 \times 10^{-7}$ (1257 final dead neurons). For the batch size variation, we kept the number of training steps **constant** for a fair comparison. Quantities are averaged over 3 random seeds.

Optimizer. The choice of optimizer inevitably influences the final count of dead neurons post-training, by altering the effective learning rate per parameter. We observed a notable discrepancy when using the ADAM optimizer (Kingma & Ba, 2015) as opposed to SGD with momentum (refer to figure 10). As also highlighted by Lyle et al. (2023), we hypothesize that this discrepancy is primarily attributed to the specific selection of hyperparameters for the ADAM optimizer ($\beta_1, \beta_2, \epsilon$), which significantly impacts neuron death. We further discuss this in Appendix E).

4 Pruning Method

The observations collected have a direct application for structured pruning: removing the dead neurons arising during the training process. This simple pruning criterion comes with the main advantage of requiring no additional overhead for its implementation. It only requires monitoring the activation outputs, already computed by the forward pass during training.

From the previous section, we know that neuron sparsity can be influenced by the learning rate, the batch size, the optimizer, and regularization strength. However, the optimizer choice is usually a design choice, while varying the learning rate and the batch size can cause instability during optimization (Cohen et al., 2021). Moreover, performing a grid search over all those hyperparameters would be costly, defeating the purpose of pruning the network during training for acceleration purposes. The possibility to do so however remains if the intent is to maximize sparsity at inference. In the rest, for its simplicity and convenience, we decide to resort to controlling the regularization strength as a mechanism to control sparsity. This choice is backed by the works of Wang et al. (2019) and Wang et al. (2021) that demonstrated the potential of L2 regularization for structured pruning. While similar in spirit, there are notable differences between ours and their approaches:

1. Their methods are for doing structured pruning on a pre-trained NN, while the method we propose is for structured pruning during the initial training phase, recovering a sparse NN directly after. As such, the analysis to justify their methods relies on the solution properties at convergence. The justification we provide for our method relies on its observed impact on the training dynamics.

2. Wang et al. (2021) uses L2 regularization to exploit the underlying Hessian information, and they use L1-norm as a pruning criterion. We use regularization to promote neuron death during training and the criterion for pruning is neural activity.

Our approach to pruning intersects with existing criteria, such as saliency (Molchanov et al., 2016) – dead neurons have a null gradient and would be picked up by this criterion. However, there is a significant shift in pruning methodology: our method influences the learning dynamics to learn sparser solutions. The need to score individual neurons for ablation is removed, observing neuron activations during the forward pass is sufficient to recover the generated sparse subnetwork. We named our method Demon's Pruning (DemP), drawing from the analogy that inspired our work and alluding to the method's darker aspect, namely, the anticipation of neural death. DemP is derived from the interplay of a single hyperparameter – regularization – and dead neurons as measured in section 3. The interplay with the other hyperparameters is not leveraged by DemP, leaving space for a broader exploration of new methods that act on the learning dynamics to retrieve sparser solutions. We now describe the specifics of our pruning method.

Dynamic Pruning. To realize computational gain during training, we prune dynamically the NN at every k steps (with a default of $k = 1000$). Dead neurons end up removed almost as they appear, not giving them any chance to be revived. This strategy allows to speed up the training with no significant change in performance (see Appendix H.1). Additionally, it removes the need for choosing correctly *when* to prune Wang et al., 2021; Rachwan et al., 2022: neurons die by themselves during training and *can be removed safely without degrading the current and future performance* (Fig. 13). The need for iterative pruning (Verdenius et al., 2020) also becomes unnecessary since pruning is not done in a single shot anymore, but instead happens gradually during the early training phase. We note that this smooth gradual pruning process is compatible with our approach in part because there is no added cost for computing the pruning criterion.

Dead Criterion Relaxation. The definition we choose for a dead neuron asks for it to be inactive to the entire dataset. In practice, we found that this criterion could be relaxed and defaulted to using 1024 examples from the training dataset to measure the death state (making the measurement across multiple minibatches when necessary). Fig. 14 shows that using this proxy for tracking dead neurons is sufficient.

Regularization Schedule. Because we noticed that neurons tend to die in the early phase of training, we gradually decay the regularization parameter over the course of training, possibly allowing the remaining neurons to recover from the earlier high regularization. Empirically, we found that using a one-cycle scheduler for the regularization parameter (λ) is a good strategy (Appendix H.3).

Weight Decay. Our method defaults back to traditional regularization, with a term added directly to the loss, as opposed to the weight decay scheme proposed by Loshchilov & Hutter (2019). By doing so, the adaptive term in optimizers takes into account regularization, and neurons move more quickly toward their death border. From a pruning perspective, it allows to achieve higher sparsity than weight decay for the same regularization strength. This is desirable because regularization affects noticeably the performance at high values.

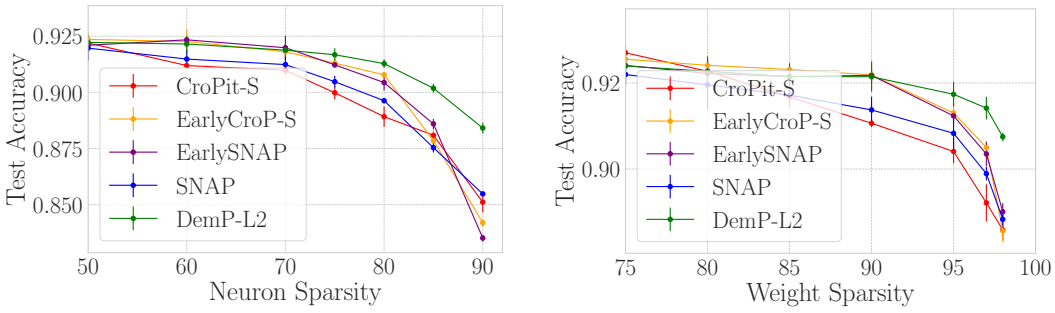


Figure 4: For ResNet-18 networks on CIFAR-10 trained with ADAM, DemP can find sparser solutions maintaining better performance than other structured approaches. **Left:** Neural sparsity, structured methods. **Right:** Weight sparsity, structured methods.

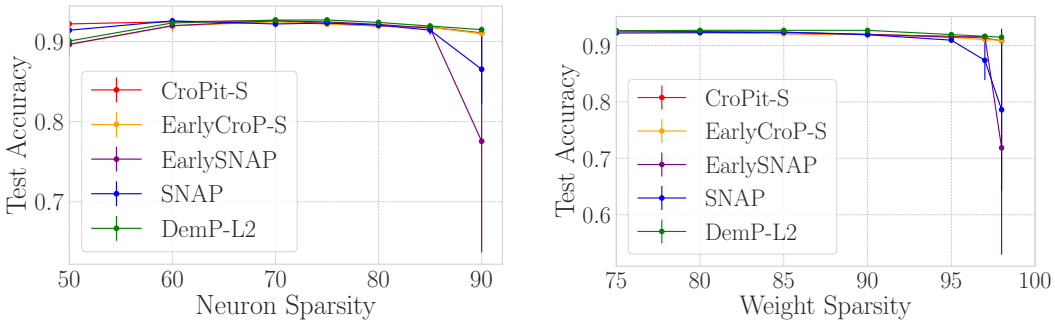


Figure 5: The results on VGG-16 networks trained with ADAM on CIFAR-10. DemP better maintains performance at higher sparsities than other structured approaches. **Left:** Neural sparsity, structured methods. **Right:** Weight sparsity, structured methods.

5 Empirical Evaluation

We focus our experiments on computer vision tasks, which is standard in pruning literature (Gale et al., 2019). We train ResNet-18 and VGG-16 netowrks on CIFAR-10, and ResNet-50 networks on ImageNet (He et al., 2016; Simonyan & Zisserman, 2015; Krizhevsky et al., 2009; Deng et al., 2009). We follow the training regimes from Evci et al. (2020) for ResNet architectures and use a setting similar to Rachwan et al. (2022) for the VGG to broaden the scope of our experiments. More details are provided in Appendix J.

Our method is a structured one, removing entire neurons at a time. The pruning happens during training, going from a dense network to a sparse one. The methods we compare with also fall into this paradigm, excluding methods like Lasby et al. (2023) which achieves impressive performance, but remains essentially unstructured pruning followed by structured reorganization. We employ the following structured pruning baselines: Crop-it/EarlyCrop (Rachwan et al., 2022), SNAP (Verdenius et al., 2020) and a modified version using the early pruning strategy from Rachwan et al. (2022) (identified as EarlySNAP). The baselines were trained using the recommended configuration of the original authors, and are not subjected to the regularization schedule employed by our method. In all scenarios, our method matches or outperforms those other structured pruning methods (Fig. 4, 5, 6, 7, and Table 1).

We included results from Lee et al. (2023) and Evci et al. (2020) in Table 1 to better illustrate the trade-off between structured and unstructured pruning methods. While unstructured methods currently offer more potential to maintain performance at higher parameter sparsity, structured methods offer direct speedup advantages.

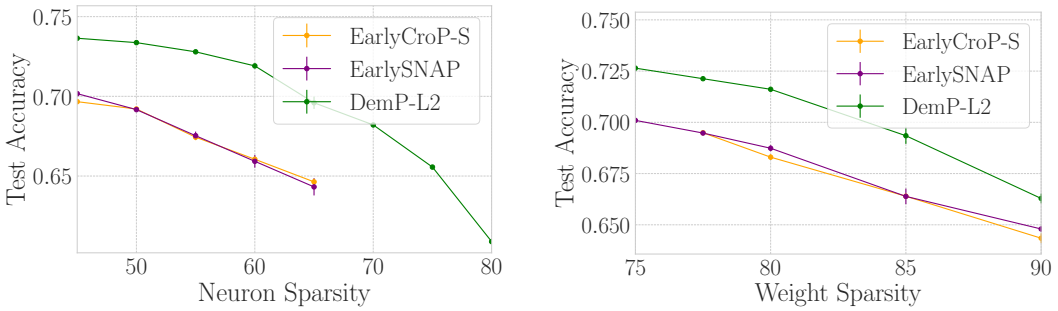


Figure 6: DemP also outperforms other structured approaches for ResNet-50 networks trained with ADAM on ImageNet, identifying more neurons that can be removed without degrading performance. SNAP and CroPit-S are excluded since they underperform considerably in this setting (see Table 1). **Left:** Neural sparsity, structured methods. **Right:** Weight sparsity, structured methods.

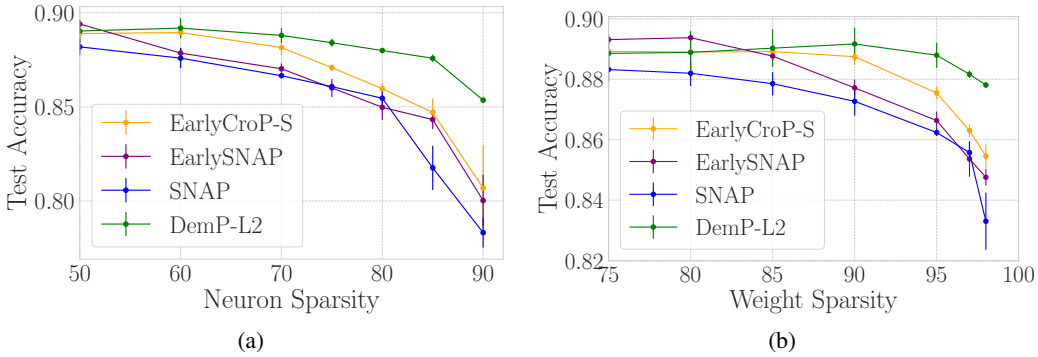


Figure 7: ResNet-18 networks with *Leaky ReLU* trained on CIFAR 10. DemP again outperforms the baseline structured pruning methods. **Left:** Neural sparsity, structured methods. **Right:** Weight sparsity, structured methods.

Leaky ReLU. Dead neurons, and thus the pruning mechanism behind our method, are naturally defined with ReLU activation functions, in which neurons can completely deactivate. However, multiple activation functions, such as Leaky ReLU (Maas et al., 2013), also exhibit a "soft" saturated region. We postulate that neurons firing solely from the saturated region do not contribute much to the predictions and can be considered *almost dead*. We test this hypothesis by employing our method in a network with Leaky ReLU activations (Fig. 7), removing neurons with only negative activation across a large minibatch. Again, our method is able to outperform other structured methods.

6 Conclusion

In this work, we have revealed how stochasticity can lead to sparsity in neural networks optimized with SGD-like methods. We have empirically demonstrated—and elucidated intuitively—how factors such as learning rate, batch size, and regularization, along with architectural and optimizer choices, collectively impact the sparsity of trained neural networks by influencing the number of neurons that die throughout the learning process. We highlighted that such effects, indicative of a loss in plasticity, can paradoxically be advantageous in a supervised learning setting, contrasting sharply with continual and reinforcement learning settings where they are deemed detrimental. Exemplifying this, we showed how the relationship between regularization and dead neurons can be leveraged to devise a simple yet effective pruning method.

This simplicity makes us confident to be able to adapt the method to a variety of situations. To make it compatible with settings specifying the desired level of sparsity in advance, we could continuously

Table 1: Comparison between different criteria when pruning a ResNet-50 trained on ImageNet around 80% (first line) and 90% (second line) weight sparsity. Because structured pruning methods do not have precise control of weight sparsity, we reported the numbers closest to these target values that we have obtained. ± indicates the standard deviation, computed from 3 seeds for the structured methods. The sparsity numbers indicate the removed ratio.

	Method	Test accuracy	Neuron sparsity	Weight sparsity	Training speedup	Training FLOPs	Inference FLOPs
	Dense	74.98% ±0.08	-	-	1.0x	1.0x (3.15e18)	1.0x (8.2e9)
Structured	SNAP	28.28% ±0.08	36.9%	81.4%	0.51x	0.32x	0.32x
		27.17% ±0.07	56.0%	90.1%	0.48x	0.25x	0.25x
	CroPit-S	28.34% ±0.52	36.9%	81.4%	0.52x	0.32x	0.32x
		27.36% ±0.16	53.2%	89.9%	0.47x	0.27x	0.27x
	EarlySNAP	68.67% ±0.15	51.70%	80.37%	0.95x	0.63x	0.63x
		63.80% ±0.58	66.6%	90.06%	0.75x	0.46x	0.45x
	EarlyCroP-S	68.26% ±0.31	51.60%	79.97%	0.94x	0.66x	0.66x
		64.20% ±0.27	66.6%	90.37%	0.82x	0.51x	0.50x
	DemP-L2	**71.52%** ±0.09	61.83%	80.13%	0.81x	0.57x	0.49x
		66.34% ±0.16	74.1%	89.93%	0.61x	0.42x	0.34x
	Dense†	76.67%	-	-	-	-	-
	Dense∗	76.8 ±0.09 %	-	-	-	1.0x (3.2e18)	1.0x (8.2e9)
Unstructured	Mag†	**75.53%**	-	80%	-	-	-
	Sal†	74.93%	-	80%	-	-	-
	SET∗	72.9%±0.39	-	80%	-	0.23x	0.23x
		69.6%±0.23	-	90%	-	0.10x	0.10x
	RigL (ERK)∗	75.10%±0.05	-	80%	-	0.42x	0.42x
		73.00%±0.04	-	90%	-	0.25x	0.24x

†values obtained from Lee et al. (2023)
∗values obtained from Evci et al. (2020)

increase regularization before cutting it off at the target ratio. It is also easy to extend existing pruning methods with it. Multiple pruning criteria will better identify the parameters to prune if they belong to a dead neuron. Unstructured methods could leverage the added structure sparsity of high regularization to achieve better computational gains.

Moreover, our experiments with Leaky ReLU exemplify that the methodology is compatible with activation functions that feature a softer saturation region than ReLU. This opens up the possibility to sparsify transformer architectures (Vaswani et al., 2017) during training since they commonly rely on activation functions such as GELU and Swish. Due to the model sizes involved in their typical training regimes, the computational gains and environmental benefits of applying our methodology there could be considerable.

REFERENCES

Zaheer Abbas, Rosie Zhao, Joseph Modayil, Adam White, and Marlos C. Machado. Loss of plasticity in continual deep reinforcement learning. *CoRR*, abs/2303.07507, 2023. doi: 10.48550/arXiv.2303.07507.

Abien Fred Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018.

Naman Agarwal, Rohan Anil, Elad Hazan, Tomer Koren, and Cyril Zhang. Disentangling adaptive gradient methods from learning rates. *CoRR*, abs/2002.11803, 2020.

Russell W. Anderson. Chapter 7 - biased random-walk learning: A neurobiological correlate to trial-and-error. In *Neural Networks and Pattern Recognition*, pp. 221–244. Academic Press, San Diego, 1998. ISBN 978-0-12-526420-4. doi: https://doi.org/10.1016/B978-012526420-4/50008-2.

Jordan T. Ash and Ryan P. Adams. On warm-starting neural network training. In *Advances in Neural Information Processing Systems, NeurIPS 2020*, 2020.

Tudor Berariu, Wojciech Czarnecki, Soham De, Jorg Bornschein, Samuel Smith, Razvan Pascanu, and Claudia Clopath. A study on the plasticity of neural networks. *arXiv preprint arXiv:2106.00042*, 2021.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

Xiang Cheng, Dong Yin, Peter Bartlett, and Michael Jordan. Stochastic gradient and Langevin processes. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1810–1819. PMLR, 13–18 Jul 2020.

Jeremy Cohen, Simran Kaur, Yuanzhi Li, J. Zico Kolter, and Ameet Talwalkar. Gradient descent on neural networks typically occurs at the edge of stability. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

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 Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pp. 248–255. IEEE Computer Society, 2009. doi: 10.1109/CVPR.2009.5206848.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *CoRR*, abs/1907.04840, 2019.

Xiaohan Ding, Guiguang Ding, Jungong Han, and Sheng Tang. Auto-balanced filter pruning for efficient convolutional neural networks. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 6797–6804. AAAI Press, 2018. doi: 10.1609/aaai.v32i1.12262.

Shibhansh Dohare, A. Rupam Mahmood, and Richard S. Sutton. Continual backprop: Stochastic gradient descent with persistent randomness. *CoRR*, abs/2108.06325, 2021.

Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14629–14638, 2020.

Utku Evci. Detecting dead weights and units in neural networks. *CoRR*, abs/1806.06068, 2018.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 2943–2952. PMLR, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *CoRR*, abs/1902.09574, 2019.

Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. Sparse gpu kernels for deep learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14. IEEE, 2020.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 1135–1143, 2015.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. EIE: efficient inference engine on compressed deep neural network. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*, pp. 243–254. IEEE Computer Society, 2016a. doi: 10.1109/ISCA.2016.30.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016b.

Fengxiang He, Tongliang Liu, and Dacheng Tao. Control batch size and learning rate to generalize well: Theoretical and empirical evidence. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 1141–1150, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90.

Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. Towards the systematic reporting of the energy and carbon footprints of machine learning. *The Journal of Machine Learning Research*, 21(1):10039–10081, 2020.

Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 3215–3222. AAAI Press, 2018.

Ioannis Karatzas and Steven E Shreve. Brownian motion and stochastic calculus. Springer New York, NY, 2014.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796, 2016.

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

Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *CoRR*, abs/1910.09700, 2019.

Mike Lasby, Anna Golubeva, Utku Evci, Mihai Nica, and Yani A. Ioannou. Dynamic sparse training with structured sparsity. *CoRR*, abs/2305.02299, 2023. doi: 10.48550/arXiv.2305.02299.

Vadim Lebedev and Victor S. Lempitsky. Fast convnets using group-wise brain damage. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 2554–2564. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.280.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, pp. 598–605. Morgan Kaufmann, 1989.

Joo Hyung Lee, Wonpyo Park, Nicole Mitchell, Jonathan Pilault, Johan S. Obando-Ceron, Han-Byul Kim, Namhoon Lee, Elias Frantar, Yun Long, Amir Yazdanbakhsh, Shivani Agrawal, Suvinay Subramanian, Xin Wang, Sheng-Chun Kao, Xingyao Zhang, Trevor Gale, Aart J. C. Bik, Woohyun Han, Milen Ferev, Zhonglin Han, Hong-Seok Kim, Yann Dauphin, Karolina Dziugaite, Pablo Samuel Castro, and Utku Evci. Jaxpruner: A concise library for sparsity research. 2023.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pp. 2755–2763. IEEE Computer Society, 2017. doi: 10.1109/ICCV.2017.298.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization. 2018.

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

Clare Lyle, Mark Rowland, and Will Dabney. Understanding and preventing capacity loss in reinforcement learning. In *International Conference on Learning Representations*, 2022.

Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Ávila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 23190–23211. PMLR, 2023.

Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *CoRR*, abs/1804.07612, 2018.

J.C. Maxwell. *Theory of Heat*. Textbooks of science. Longmans, Green, and Company, 1872. ISBN 9780598862662.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *CoRR*, abs/1611.06440, 2016. URL http://arxiv.org/abs/1611.06440.

Bernt Karsten Oksendal. Stochastic differential equations: an introduction with applications. 6th edition. Springer Berlin, Heidelberg, 2010.

Scott Pesme, Loucas Pillaud-Vivien, and Nicolas Flammarion. Implicit bias of sgd for diagonal linear networks: a provable benefit of stochasticity. In *Advances in Neural Information Processing Systems*, volume 34, pp. 29218–29230. Curran Associates, Inc., 2021.

Loucas Pillaud-Vivien. Rethinking sgd's noise. https://francisbach.com/implicit-bias-sgd/, 2022. Accessed: 2023-08-26.

John Rachwan, Daniel Zügner, Bertrand Charpentier, Simon Geisler, Morgane Ayle, and Stephan Günnemann. Winning the lottery ahead of time: Efficient early network pruning. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 18293–18309. PMLR, 2022.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates, 2018.

Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. Don't decay the learning rate, increase the batch size. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 32145–32168. PMLR, 2023.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pp. 3645–3650. Association for Computational Linguistics, 2019. doi: 10.18653/v1/p19-1355.

Ludovic Trottier, Philippe Giguere, and Brahim Chaib-Draa. Parametric exponential linear unit for deep convolutional neural networks. In *2017 16th IEEE international conference on machine learning and applications (ICMLA)*, pp. 207–214. IEEE, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017.

Stijn Verdenius, Maarten Stol, and Patrick Forré. Pruning via iterative ranking of sensitivity statistics. *CoRR*, abs/2006.00896, 2020.

Loucas Pillaud Vivien, Julien Reygner, and Nicolas Flammarion. Label noise (stochastic) gradient descent implicitly solves the lasso for quadratic parametrisation. In *Proceedings of Thirty Fifth Conference on Learning Theory*, volume 178 of *Proceedings of Machine Learning Research*, pp. 2127–2159. PMLR, 02–05 Jul 2022.

Chaoqi Wang, Guodong Zhang, and Roger B. Grosse. Picking winning tickets before training by preserving gradient flow. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Huan Wang, Qiming Zhang, Yuehai Wang, Lu Yu, and Haoji Hu. Structured pruning for efficient convnets via incremental regularization. In *International Joint Conference on Neural Networks, IJCNN 2019 Budapest, Hungary, July 14-19, 2019*, pp. 1–8. IEEE, 2019. doi: 10.1109/IJCNN.2019.8852463.

Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural pruning via growing regularization. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 2074–2082, 2016.

Stephan Wojtowytsch. Stochastic gradient descent with noise of machine learning type part I: discrete time analysis. *J. Nonlinear Sci.*, 33(3):45, 2023. doi: 10.1007/s00332-023-09903-3.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics.

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853, 2015.

Jianbo Ye, Xin Lu, Zhe Lin, and James Z. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 2130–2141, 2019.

A DEAD NEURONS IN CONVOLUTIONAL LAYERS

In convolutional layers, ReLU is applied element-wise to the pre-activation feature map. We consider an individual neuron (filter) dead if all elements of the feature map post-activation are 0. Formally,

Definition: The j-th neuron/filter in the convolutional layer ℓ is *inactive* if it consistently outputs a feature map (post-activation) with elements summing to zero on the entire training set, i.e $\sum_{k,l} F^{\ell}_{jkl} = 0$. A neuron/filter that becomes and remains inactive during training is considered as *dead*.

B BIASED RANDOM WALK MODEL

To formalize the intuition from section 3.1, we follow a standard line of work (Cheng et al., 2020) taking the view of SGD in Eq. 1 as a biased random walk (Anderson, 1998), described by the Langevin process,

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta g(\boldsymbol{w}_t) + \sqrt{\eta}\,\hat{\boldsymbol{\xi}}(\boldsymbol{w}_t, b_t) \tag{2}$$

where the zero mean variable $\hat{\boldsymbol{\xi}}(\boldsymbol{w}, b) := \sqrt{\eta}\,(g(\boldsymbol{w}) - \hat{g}(\boldsymbol{w}, b))$ represents the gradient noise. In the limit of small learning rate, Eq. 2 is well approximated (Cheng et al., 2020, Theorem 2) by the following stochastic differential equation (SDE),

$$d\boldsymbol{w}_t = -g(\boldsymbol{w}_t) + M(\boldsymbol{w}_t) d\boldsymbol{B}_t, \tag{3}$$

where $\boldsymbol{B}_t$ denotes a standard Brownian motion and $M(\boldsymbol{w}) := \sqrt{\mathbb{E}_b[\hat{\boldsymbol{\xi}}(\boldsymbol{w}, b)\hat{\boldsymbol{\xi}}(\boldsymbol{w}, b)^\top]}$.

A key aspect of the gradient noise in SGD is that it is *multiplicative*, i.e., it depends on the parameter. Now, a well-known property of systems with multiplicative noise is that regions of lower noise magnitude can act as attractors (Oksendal, 2010). Intuitively, this is because the noise pushes the system away from regions where it has a higher impact, leading to a higher probability of staying in regions where it has a lower impact. Mathematically, this is characterized by a tendency for the invariant distribution to have higher probability density in regions of lower noise magnitude.

We illustrate this on a simplified version of Eq. 3, which partially captures the effect we want to highlight.

Absorbing Brownian motion. We consider a one-dimensional absorbing Brownian motion with a boundary at zero, which can be described by the SDE:

$$dw_t = \begin{cases} \sqrt{\eta} dB_t & \text{as long as } w_t > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

It models a system subject to noise in an 'active' region $w > 0$, which gets stopped at 0 – and remains there, hence 'dies' – once it hits 0. This illustrative example can be thought of as a simplified description of a regime where the dynamics (4) is dominated by noise, such as e.g., a neuron encoding features with very low correlation to the task.

In this case, the probability that the system is still active at time t, i.e $w_t > 0$, is related to the distribution of the first hitting time at 0 of a standard Brownian motion: it is given by $P(T_0 > t)$, where $T_0 = \inf\{t \geq 0 : B_t = 0\}$. A well-known property of Brownian motion (Karatzas & Shreve, 2014) is $\lim_{t\to\infty} P(T_0 > t) = 0$, which shows that the system Eq. (4) eventually dies with probability 1. More generally, the following result specifies the dependence on initialization:

Proposition 1. *Consider the system 4 initialized at $w_0 > 0$. Then the probability that the system is still active at a given time $t > 0$ is given by*

$$P(w_t > 0 \,|\, w_0) = \sqrt{\frac{2}{\pi}} \int_0^{w_0/\sqrt{t}} e^{\frac{-u^2}{2\eta}} du \tag{5}$$

Prop. 1 implies that (i) the system eventually dies almost surely, (ii) for any given finite horizon time t, the smaller the initialization, the more likely the system is dead at t, $\lim_{w_0\to 0^+} P(w_t = 0 \,|\, w_0) = 1$. Finally, the dependence on the scaling η, which represents the learning rate, illustrates how a noisier environment can accelerate this dying process.

C FEW DEAD NEURONS REVIVE

While empirical observations have shown a gradual accumulation of dead neurons (Fig. 1), we also observed that neurons can revive (Appendix D.1). To better assess the potential impact of reviving neurons on performance, we measured the overlap ratio ($|X \cap Y| / \min(|X|, |Y|)$) between the historical set of dead neurons at previous iterations and the set of dead neurons at the current iteration. This methodology directly follows Sokar et al. (2023). The results in figure 8 show that most neurons (over 90%) inactive at any point during training end up dead at the final iteration. This – coupled with our results showing that dying neurons can be dynamically pruned during training without impacting performance (Appendix H.1) – strongly suggests that neurons becoming inactive at any point during training in ReLU networks do not contribute significantly to the final performance of the trained model.

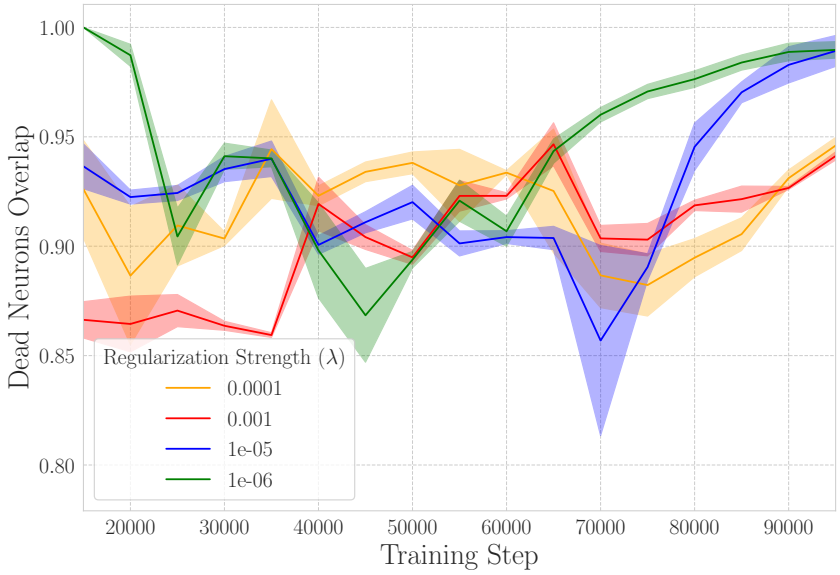


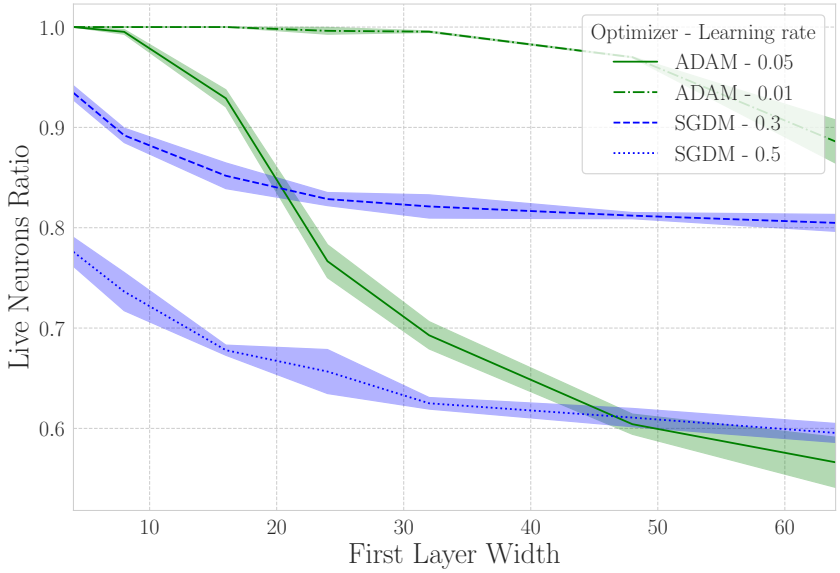
Figure 8: Overlap ratio of dead neurons during training, as measured across all layers of a ResNet-18 trained with ADAM on CIFAR-10 at various regularization strengths. Results are shown for training steps bigger than 15k because no dead neurons were observed previously for the lowest regularization strength. We observe that the vast majority (over 85%) of neurons dying early never revive. More importantly, even if they may have been revived earlier, $\approx 95\%$ of neurons that became inactive at any point during training are dead when training finishes.

D HYPERPARAMETERS IMPACT, ADDITIONAL RESULTS

D.1 TRAINING TIME

The relation with training time, asserting that the probability of a neuron dying increases as training progresses (Prop. 1) doesn't entirely align with practical applications. Modern overparameterized architectures often have the capacity to memorize the entire training dataset, achieving zero loss in the process. Given that the gradient signal is proportional to the loss, it would concurrently diminish to zero for all neurons, preventing any further death.

We observe a pattern consistent with this idea (Fig. 1), where the total count of dead neurons spikes sharply in early training to then fluctuate slightly before stabilizing. The fluctuations demonstrate that neurons *can indeed revive*. However, additional experiments with ReLU networks revealed that most reviving neurons die again later (Fig. 8) and that their dynamic elimination has negligible to no impact on performance (Fig. 13).



(a)

Figure 9: An increased width leads to a higher ratio of neurons dying, independently of the optimizer. We use the number of channels in the initial layer of the ResNet-18 to indicate the width, with 64 being the typical number of channels in the first convolution layer.

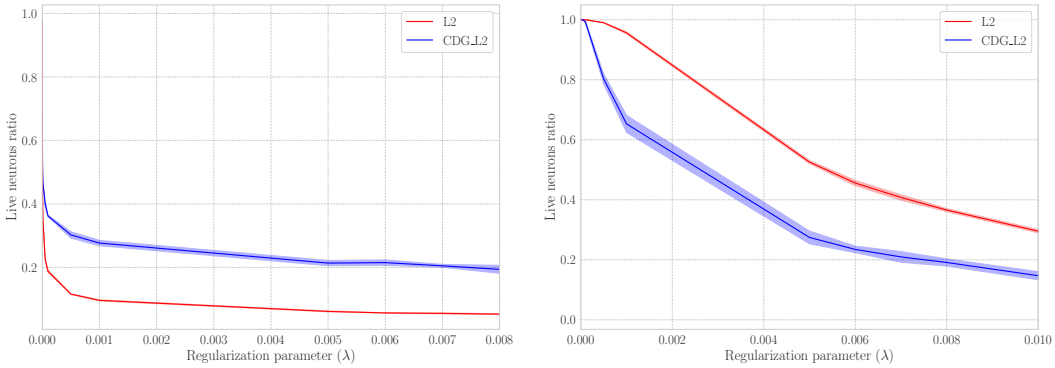
D.2 Network Width

The widths of a neural network's layers also influence the ratio of live neurons (live neurons to total neurons in the network) post-training (see figure 9). Typically, this ratio increases with the width; however, the total number of live neurons continues to rise with increased width. This phenomenon is somewhat anticipated as incorporating more neurons with random initialization in any given layer can only amplify the training noise, especially in the initial phase. Moreover, since initialization functions usually adjust their standard deviation proportionally to the number of channels ($\sigma \propto \sqrt{\frac{1}{\text{fan\_in}+\text{fan\_out}}}$), widening the network places neurons closer to their death border right from the initialization. The connection between width and dead neurons maintains its significance as neural network sizes are inclined to increase over time with the availability of more computational resources. If this trend persists, the accumulation of dead neurons could potentially become increasingly pervasive.

D.3 Regularization

We also experimented with a slightly modified version of L2, that solely regularizes the positive weights. The intuition behind it was that by regularizing only the positive weights, the average pre-activation output of a ReLU network would gradually shift toward negative outputs. We dubbed it the *coup de grâce* L2 normalization (CDG_L2), and describe it further in Appendix F. It proved more aggressive than classical L2 regularization, except when using batch normalization (BN) layers with ADAM optimizer (fig.10 and Appendix G).

The particular impact of normalization layers is interesting. Without regularization and with everything else left equal, BN leads to less dead neuron accumulation (Appendix G). But with regularization, two distinct modalities appear in neuron pre-activations distribution: one centered around 0 and another toward the positive values. Normalization then shifts the mean of this distribution to 0, bringing the modality that was previously close to the origin on the negative (dead) side. Normalization paired with regularization can therefore be a very effective strategy for promoting neuron death.



(a) (b)

Figure 10: Increasing the regularization parameter (λ) quickly increases the number of dead units that accumulate during training by bringing the neurons closer to their death border. BN was used in the experiments that generated both figures. **Left**: When using ADAM optimizer, dead neurons accumulate more quickly with classical L2 regularization. ADAM is particularly effective for killing neurons, indicated by the much steeper decrease in live neurons when regularization is increased. **Right:** The situation is reversed when using momentum, with CDG_L2 generating more dead neurons.

E ADAM IS A NEURON KILLER

The greater impact of ADAM over the dying ratio compared to momentum must be due to the second-moment term, which is the only significant difference with momentum. Recall that ADAM update is (Kingma & Ba, 2015):

$$
\begin{aligned}
m_t &= \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \\
v_t &= \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \\
\hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\
\hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \\
\theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t
\end{aligned}
$$

Earlier, we hypothesized that the neurons ending up dead were the ones experiencing very small gradients, such that the noise dominated their update trajectories. If this is the case, g_t^2 (the squared gradient) would be very small for those neurons' parameters, eventually leading to a very small second-moment estimation $\hat{v}_t$. In such a scenario, ϵ would end up dominating $\sqrt{\hat{v}_t}$, effectively multiplying the learning rate by ϵ which is typically set to 1×10^{-8}. Moreover, as the decay ($\beta_2 = 0.99$) of $\hat{v}_t$ is usually slower than the one of $\hat{m}_t$ ($\beta_1 = 0.9$), a few sudden noisier updates would be sufficient to make huge random steps.

It is worth noting that RL practitioners typically set epsilon to a higher value (Hessel et al., 2018), as it has empirically been found to perform better. Higher ϵ values should reduce the number of dead neurons induced by ADAM optimizer, which could be the cause for the improved performance/stability observed in RL. Also, because of constant distribution shifts, rapid accumulation of dead neurons often occurs in RL tasks.

Also notable, HuggingFace Transformers library (Wolf et al., 2020) default ϵ ADAM parameter to 1×10^{-6}, following RoBERTa example (Liu et al., 2019). Manipulating the ϵ parameter of AdaGrad was also observed to impact significantly a transformer performance model (Agarwal et al., 2020). Verifying if those heuristic choices are due to their impact on dead neuron accumulation would be quite interesting.

F CDG_L2

Although we don't know where the actual death border lies in a neural network using ReLU, a unit is certain to be dead (except in the initial layer) if all of its weights are smaller than zero, i.e. if $w_j^i <= 0 \ \forall i$. This is because the output of neuron j will then always be negative $\boldsymbol{w}_j \boldsymbol{x}_l^T$ since $x_l^i >= 0$, that is the layer inputs are always positive in ReLU networks.

Therefore, by pushing a neuron's parameters toward the negative side, we should make it more probable to die. It is to that end that we introduce the *coup de grâce* L2 normalization (CDG_L2). Formally, we define it as:

$$CDG\_L2 = \lambda \sum_{w_i > 0} w_i^2 \tag{6}$$

It is in essence L2 regularization applied only to the positive weights of the network. The intuition behind it is that by regularizing only the positive weights, the average pre-activation output of a ReLU network will gradually shift toward negative outputs. This simple change proved enough to push over the dead border neurons that otherwise activated sporadically.

G Normalization Layers

To study the impact of normalization layers on dead neurons, we monitored the number of inputs for which individual neurons were activated after training a NN. The experiences were performed with a ResNet-18 trained with ADAM on CIFAR-10, making the range for activation count between 0 and 50,000. This allows us to monitor the amount of dead neurons, the ones for which the activation count is zero. We plotted those results as density histograms. Figure 11 shows the results when using L2 and CDG_L2 with BN layers and in figure 12 we display the results when there are no BN layers in the network. Dead neurons are the ones belonging to the leftmost bucket.

A first observation is that when BN layers are present, there are more neurons activating for all or almost all of the entire dataset. This supports our claim that BN layers can help reduce the amount of dead neurons when everything else is kept equal. But we can also observe that BN layers make the first modality much more light-tailed, meaning that fewer neurons are activated for a small portion of the dataset. We believe this is due to the normalization process that puts the distribution mean at the origin. This brings the entire heavy-tailed, pre-normalized first modality in the negative regions before ReLU. Thus, when BN layers are used, neurons either activate almost all the time or never activate. This in turn makes the dead neuron criterion that we use, i.e. neurons that do not activate for the entire training dataset, particularly appropriate when BN layers are used.

Finally, we also observe that CDG_L2 "drains" the rightmost modality more quickly than L2, supporting our claim that the CDG variant is generally more aggressive for neuron killing.

H Pruning Method Details

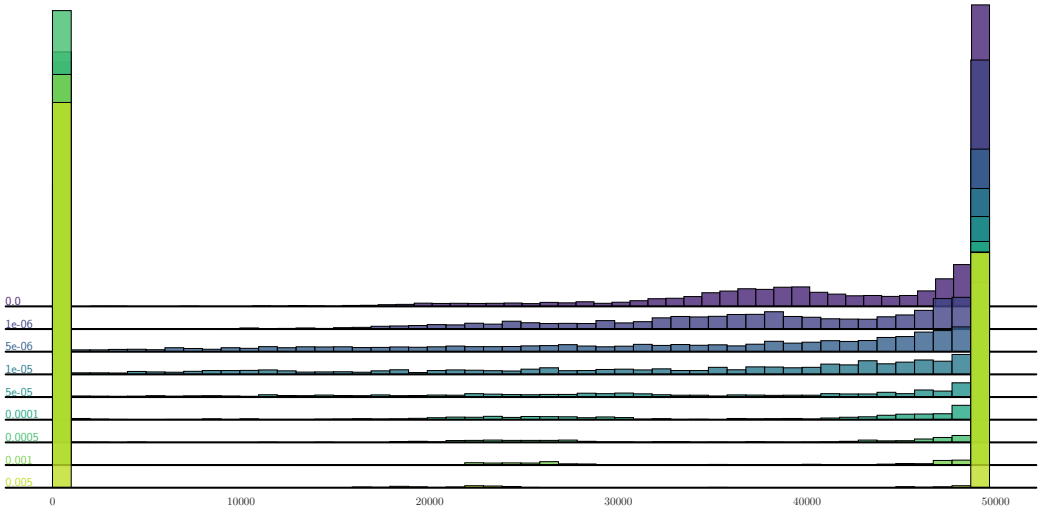
We validate and justify the heuristic choices made for our pruning method via empirical observation exposed in this section. We used the same setup as before for a ResNet-18 trained on CIFAR-10.

H.1 Dynamic Pruning

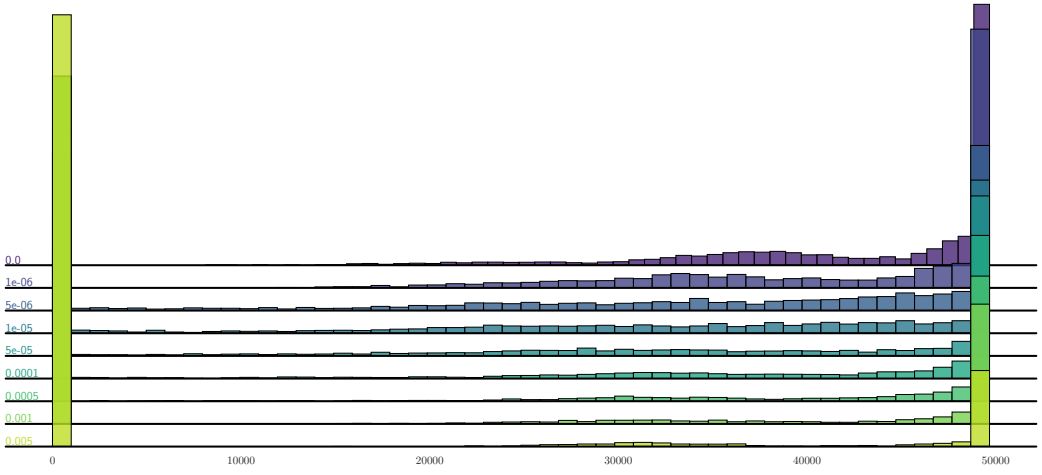
To verify the impact of dynamic pruning, we measured if there were any performance discrepancies when it was enabled or not. Across runs, we varied the regularization strength while measuring accuracy and sparsity. The results, in figure 13, show that enabling dynamic pruning does not affect the final performance. The very slight variations between runs fall well between the expected variance across different runs. This experiment really reinforces the hypothesis that neurons that die and later revive during training do not contribute significantly to the learning process.

H.2 Dead Criterion Relaxation

To measure if a minibatch could be used to measure the death state instead of the entire dataset, we tracked the number of dead neurons during training with both metrics. We used a minibatch

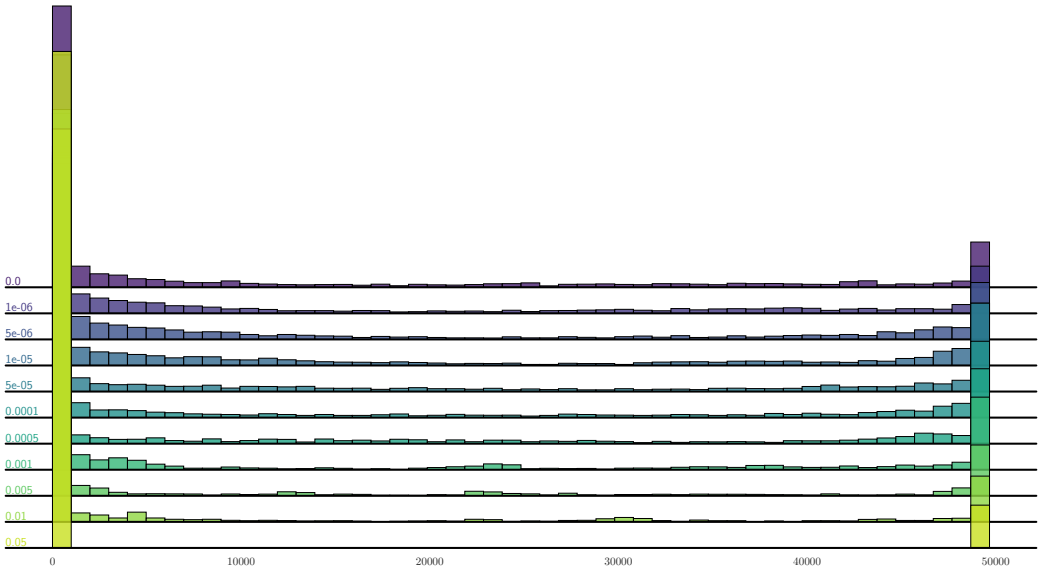


(a)

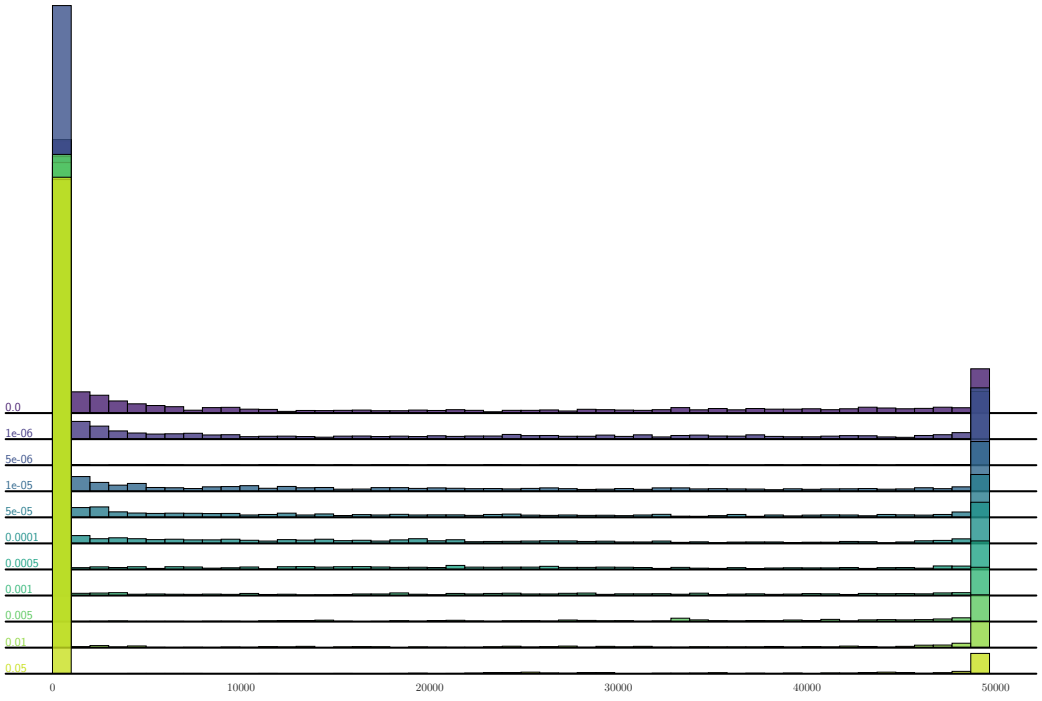


(b)

Figure 11: Histogram of individual neuron's activation count across the training dataset (CIFAR-10) for a ResNet-18 **with** BN layers. Distributions are shown for different values of the regularization parameter (λ). Regularization increases toward the front, with the specific value indicated on the far left of the figure. The leftmost modalities are single peaks at 0 when BN layers are present, indicating that neurons either never activate, or activate almost all the time with normalization. **Top:** L2 regularization. **Bottom:** CDG_L2 regularization.



(a)



(b)

Figure 12: Same experiments as figure 11, but for a ResNet-18 **without** BN layers. When BN is removed, the leftmost modalities become more heavy-tailed, indicating that a portion of the neurons activate very sparsely across the training dataset. **Top:** L2 regularization. **Bottom:** CDG_L2 regularization.

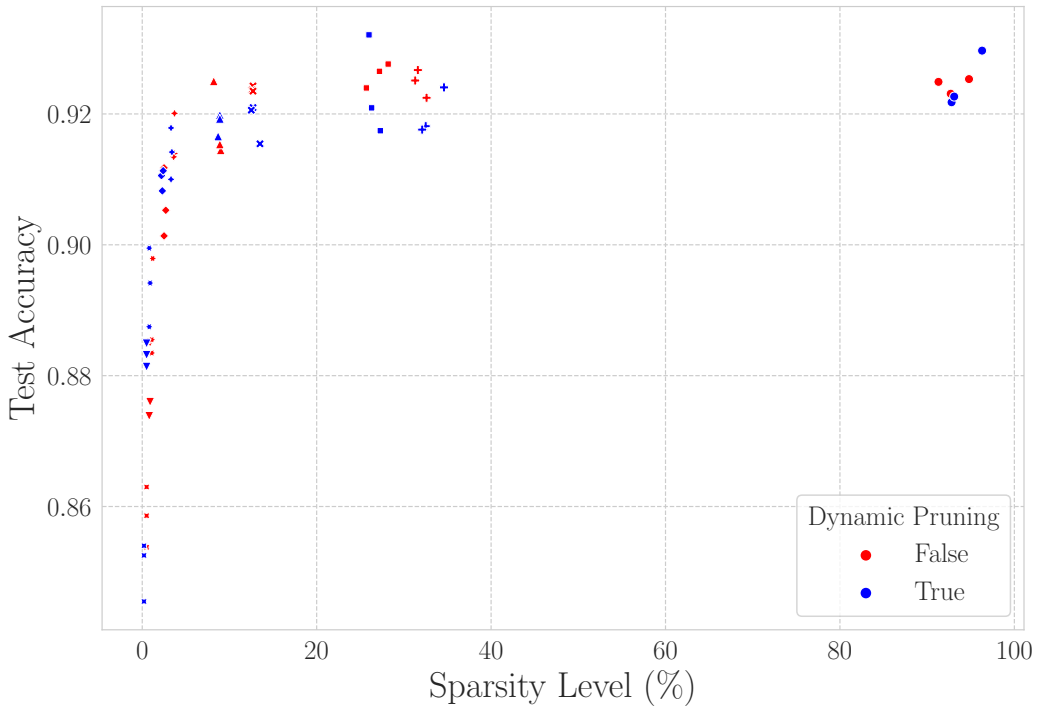


Figure 13: Measuring final accuracy vs. sparsity when dynamic pruning is enabled or not. There are barely any variations between the two strategies, allowing us to conclude that using dynamic pruning does not affect performance. Different symbols were associated to different regularization strengths. Experiment performed with ResNet-18 on CIFAR-10 for 3 seeds.

containing 512 inputs from the training dataset for the proxy measurement. We can see that both curves closely track each other. More importantly, they match at the end of the training, indicating that overall the same amount of neurons would be removed when performing the death check over the minibatch. Dynamic pruning was disabled for this experiment.

H.3 Decaying the Regularization Parameter

Finally, we also empirically tested different schedules over the regularization parameter, trying to mitigate the impact of high regularization by decaying the parameter over the course of the training. We settled on using a one-cycle scheduler for the regularization strength because of slightly better performance in the higher sparsity level. However, we remark that even a constant schedule over the regularization parameter is sound with our method

I Additional Comparison with Unstructured Methods

We employ the `JaxPruner` package (Lee et al., 2023) to illustrate further trade-off of our method against some unstructured methods. Our method is capable of achieving *similar* performance to unstructured ones for the ResNet-18 and VGG-16 experiments (Fig. 16). The comparisons with the unstructured methods use their default configuration from JaxPruner, which was tuned for a ResNet-50. We expect their performance on ResNet-18 and VGG-16 to be improved by tuning the pruning distribution, the pruning schedule, and the pruning iterations scheme (Lee et al., 2023). However, for those not interested in expensive tuning, our method becomes an interesting default choice.

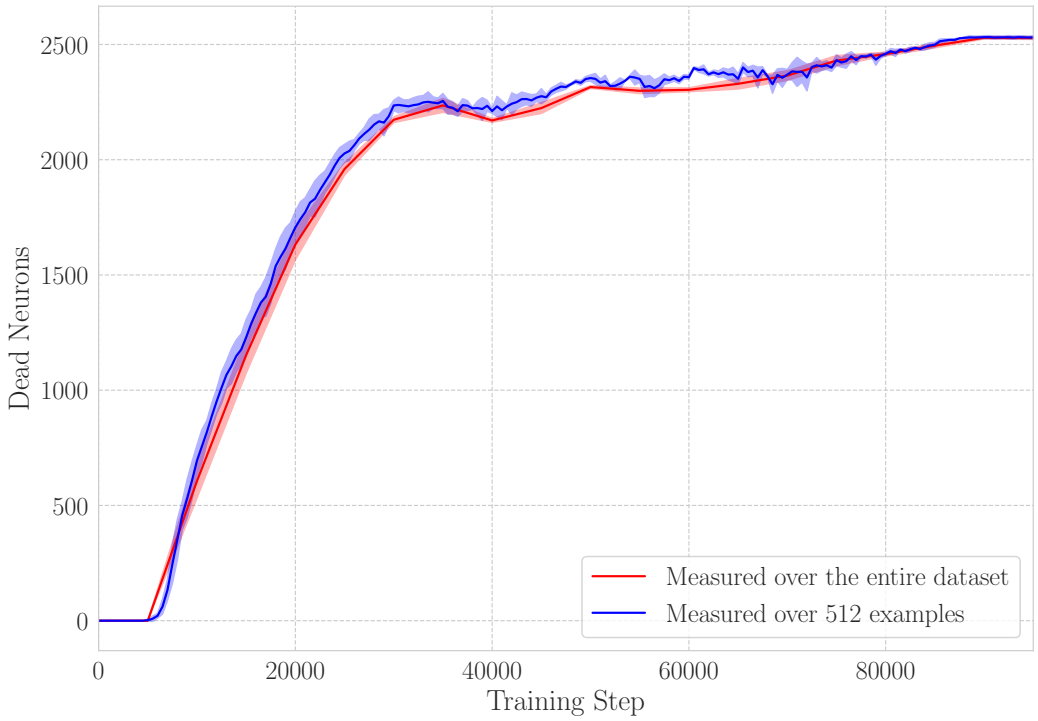


Figure 14: Instead of validating the death state of neurons against the entire training dataset, it proves sufficient to use a smaller dataset. The curves meet at the end of the training, indicating that the same final number of neurons would be removed for both strategies. Experiment performed with ResNet-18 on CIFAR-10 for 3 seeds.

J Implementation details

J.1 ResNet-18/ResNet-50

We mostly followed the training procedure of Evci et al. (2020) for the ResNet architectures.

ResNet-18. We train all networks for 250 epochs using a batch size of 128. The learning rate is initially set to 0.005 and is thereafter divided by 5 every 77 epochs. We use ADAM optimizer because it induces higher sparsity. While varying regularization (L2 or CDG_L2) is used with our method, we default to a constant weight decay (0.0005) for all other methods than ours. Random crop and random horizontal flips are used for data augmentation.

ResNet-50. We trained the ResNet-50 for 100 epochs, with a batch size of 256 instead of 4096. The initial learning rate is set to 0.005, before being decayed by a factor of 10 at epochs 30, 70, and 90. Label smoothing (0.1) and data augmentation (random resize to either 256 $\times$ 256 or 480 $\times$ 480, before randomly cropping to 224 $\times$ 224. Followed by random horizontal flip and input normalization) are also used. We again use ADAM, vary the regularization with our method but use a constant weight decay (0.0001) for other methods.

J.2 VGG-16

We followed a training procedure similar to Rachwan et al. (2022). We used ADAM with a learning rate of 0.005 and a batch size of 256, with the One Cycle Learning Rate scheduler (Smith & Topin, 2018). The networks are trained for 80 epochs. CIFAR-10 images are normalized and resized to 64 $\times$ 64 before applying random crop and random horizontal flip for data augmentation.

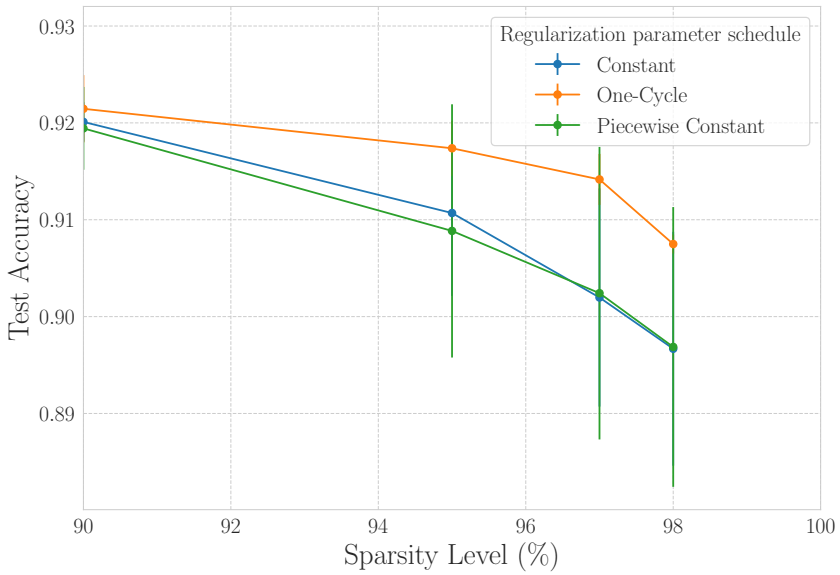
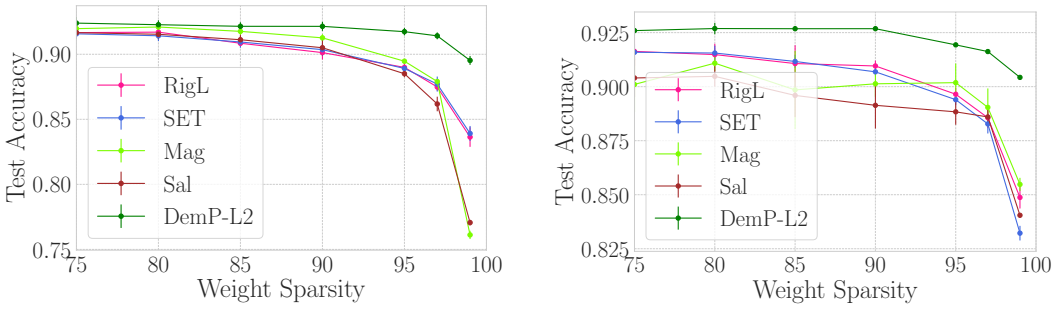


Figure 15: We studied the impact of different schedules over the regularization parameter for our method, settling down on a one-cycle scheduler as default. Experiment performed with ResNet-18 on CIFAR-10 across 3 seeds.



(a) (b)

Figure 16: At default configuration DemP competes with common unstructured methods for networks trained on CIFAR-10, especially at high sparsity. **Left:** Weight sparsity in a ResNet-18. **Right:** Weight sparsity in a VGG-16.

J.3 Structured Methods

We closely reimplement in JAX (Bradbury et al., 2018) the structured methods from Rachwan et al. (2022), keeping all the hyperparameters specific to every method as is. The training hyperparameters are the same as specified in J.1 and J.2.

J.4 Unstructured Methods

For the unstructured methods, we rely on Lee et al. (2023) implementations, using their method's configuration for pruning a ResNet-50 for all our experiments. The training hyperparameters are the same as specified in J.1 and J.2.