Decoupled Weight Decay for Any p Norm

Nadav Joseph Outmezguine [1 2] **Noam Levi** [3 4]

Abstract

With the success of deep neural networks (NNs) in a variety of domains, the computational and storage requirements for training and deploying large NNs have become a bottleneck for further improvements. Sparsification has consequently emerged as a leading approach to tackle these issues. In this work, we consider a simple yet effective approach to sparsification, based on the Bridge, or L_p regularization during training. We introduce a novel weight decay scheme, which generalizes the standard L_2 weight decay to any p norm. We show that this scheme is compatible with adaptive optimizers, and avoids the gradient divergence associated with $0 < p < 1$ norms. We empirically demonstrate that it leads to highly sparse networks, while maintaining generalization performance comparable to standard L_2 regularization.

1. Introduction

Deep neural networks (NNs) have garnered unparalleled success across a variety of domains ranging from vision (He et al., 2016) to language (Vaswani et al., 2017; van den Oord et al., 2016; Kalchbrenner et al., 2018). Modern network performance has been shown to scale with both model complexity and dataset size, now operating in the jointly large parameter and large data size regime (Hestness et al., 2017). The resources required to train and deploy large NNs can, consequently, impose a bottleneck on further improvements (Kaplan et al., 2020). For instance, Inception-V4 (Szegedy et al., 2016), requires 16 billion arithmetic operations and 43 million parameters to be evaluated, while GPT-4 (OpenAI et al., 2023) requires over 1.75 trillion parameters (2 TiB assuming 16 bits per parameter). Furthermore, training such models becomes increasingly expensive. Large language models (LLMs) already require supercomputers for training, with costs potentially reaching tens of millions of dollars per run, as cited in GPT-3 (Brown et al., 2020). Moreover, these models induce tremendous energy costs, as highlighted in the study on energy costs (de Vries, 2023). It is therefore critical to study *sparsification* during the training process as an avenue to manage resources during training and deployment (Hastie et al., 2015).

We define the sparsity of an NN as the fraction of its parameters that have a value of exactly zero. Higher sparsity therefore corresponds to fewer informative parameters, and thus, potentially, lower computational and storage requirements. With zero valued weights, any multiplications (which dominate neural network computation) can be skipped, and sufficiently sparse models can be stored and transmitted compactly using sparse matrix formats. Sparse models are required to store more information per parameter relative to their denser counterparts. They may, therefore, be less prone to overfitting, and exhibit better generalization performance (e.g. LeCun et al., 1989; Hassibi & Stork, 1992; Reed, 1993; Hoefler et al., 2021). It has been empirically shown that deep NNs can perform effectively even with high levels of sparsity (Han et al., 2015; Narang et al., 2017; Ullrich et al., 2017; Gromov et al., 2024). This property is leveraged to reduce costs and enable the deployment of state-of-the-art models in resource-constrained environments (Theis et al., 2018; Kalchbrenner et al., 2018; Valin & Skoglund, 2018). In particular, modern GPU architectures like NVIDIA's Ampere, equipped with Sparse Tensor Cores, can leverage unstructured sparsity at levels as low as 50% to achieve significant inference speedups (Mishra et al., 2021). Additionally, recent research has demonstrated that applying sparsity in the fine-tuning of large language models can lead to substantial inference acceleration on both CPUs and GPUs, without compromising accuracy (Kurtic et al., 2023).

In recent years, various techniques for inducing sparsity in NNs have been proposed, including post-training pruning and dynamical regularization-based approaches (Kwon et al., 2022; Lasby et al., 2024; Yin et al., 2023). Our work falls in the latter category, focusing in particular on weight regularization. Weight regularization methods methods introduce a penalty term (regularizer) into the loss function

[1]Berkeley Center for Theoretical Physics, University of California, Berkeley, CA 94720, USA [2]Theory Group, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA [3]École Polytechnique Fédérale de Lausanne (EPFL), Switzerland. [4] Raymond and Beverly Sackler School of Physics and Astronomy, Tel-Aviv University, Tel-Aviv 69978, Israel. Correspondence to: Nadav J. Outmezguine <NJO@Berkeley.edu>, Noam Levi <noam.levi@epfl.ch, noam@mail.tau.ac.il>.

to constrain the magnitude of each of the model parameters. This constraint implicitly biases the network towards sparser, rather than denser model representations and gradually reduces the magnitudes of the network weights during the training process. Generally, regularization methods can be written as $\mathcal{L}'(\boldsymbol{x}, \boldsymbol{w}) = \mathcal{L}(\boldsymbol{x}, \boldsymbol{w}) + R(\boldsymbol{w})$. Here, $\mathcal{L}$ is the original loss function defined on the weights $\boldsymbol{w}$ and the data samples $\boldsymbol{x}$, while R is the regularizer term which acts only on the weights.

The most common weight regularization method is L_2. While L_2 regularization achieves smaller weights and better generalization error at the end of the training process (e.g. Plaut et al., 1986; Nowlan & Hinton, 1992; Krogh & Hertz, 1991; Moody, 1991; Wei et al., 2019), it does not result in a sparse network representation. This is since the penalty term is 'rotationally invariant', meaning that it does not favor any particular direction in the weight space. A ubiquitous regularization method which does result in sparse networks is L_1, or Lasso regularization (Tibshirani, 1996). Elastic-net regularization, which combines both L_1 and L_2 norms of the weights, was suggested as a method that exhibits both the sparsity of L_1 and the generalization performance of L_2 (Zou & Hastie, 2005).

Towards a more general form of regularization, Frank & Friedman (1993) proposed Bridge regularization, or L_p regularization, $R_p \sim \|\boldsymbol{w}\|_p^p$, in which p is chosen based on the problem at hand. *This is the underlying regularization method which we base our work upon*. Bridge regression enjoys some desirable statistical properties, such as sparsity and near-unbiased estimates for L_p norms in the range $p \in (0, 1)$ (Fan & Li, 2001; Sleem et al., 2024). Importantly, when $p < 1$, the so-called p-norm does not adhere to the triangle inequality. It does, however, satisfy the weaker condition $\|x + y\| \leq C \times (\|x\| + \|y\|)$ for some $C > 1$, which qualifies it as a quasi-norm. Additionally, in this range of $p < 1$, the quasi-norm is non-convex, making it more challenging to optimize (Zhang, 2010). Previous works suggested variations on Bayesian sampling approaches to bypass these issues for $p \in (0, 1)$ (Polson et al., 2014; Loría & Bhadra, 2023). Despite these complexities, for ease of notation, we refer to it as the p-norm throughout this paper.

In this work, we introduce a straightforward implementation of L_p regularization. This method maintains the key benefits of L_p regularization, such as sparsification and generalization, while *avoiding numerical instabilities caused by exploding gradients at the small weights limit*. Our approach integrates a single, simple step that complements any modern optimizer with minimal computational overhead. Furthermore, it can easily be adapted to more flexible regularization schemes, including variants of the Elastic Net.

Our main contributions are as follows:

- In Sec. 3, we illustrate how an $L_{p<2}$ regularized problem with N parameters is equivalent to another optimization problem with N additional auxiliary parameters. We show that the optimal solutions of both problems coincide, and for $p < 2$ these solutions are expected to be sparse.
- In Sec. 4, we introduce our main contribution, the 'p-norm Weight Decay' (pWD), a novel weight decay scheme for any p-norm regularization. We use a toy example to demonstrate that, across all $0 \leq p$ values, pWD avoids gradient instabilities and stabilizes training dynamics. We then present the pWD algorithm which implements this new weight decay method.
- In Sec. 5, we empirically assess the performance of p-norm Weight Decay (pWD) across various tasks and architectures, including comparisons with other sparsification methods. Our results show that pWD achieves high levels of sparsity while maintaining excellent network generalization.
- In Sec. 6, we discuss some limitations of pWD, suggest possible extensions, and propose future research directions.

2. Related Work

Regularization and sparsification: Besides linear regression, Bridge regularization has been applied to support vector machines (Liu et al., 2007), giving impressive results. As a special case of Bridge regularization, $L_{1/2}$ has been shown to exhibit useful statistical properties including sparseness and unbiasedness (Xu et al., 2010). Different training algorithms have been proposed for training neural networks with $L_{1/2}$ weight penalty (Fan et al., 2014; Yang & Liu, 2018). In terms of Bayesian estimation, Ridge and Lasso penalties imply a Gaussian and Laplacian prior on model weights, respectively. On the other hand, an L_p penalty corresponds to the Generalized Gaussian prior on the model weights (Frank & Friedman, 1993).

Proximal operators: The proximal operator for the (lasso) regularization, known as the soft thresholding operator, is widely used in the literature (Daubechies et al., 2003). The proximal operator for various other specific values of norms has also been studied in the literature, for example in (Xu et al., 2012; Chen et al., 2016), but result in cumbersome schemes. Partially for that reason, approximated operators were devised, for example in (O'Brien & Plumbley, 2018).

Bridge regression: First suggested by (Frank & Friedman, 1993), Bridge regression, or L_p regularization, has been studied extensively. It has been shown that Bridge regularization performs better than Ridge, Lasso and elastic-net in certain regression problems (Park & Yoon, 2011). In recent

years, works such as (Polson et al., 2012; Khan et al., 2018) consider stochastic variations, while McCulloch et al. (2023) integrate the concept of L_p regularization for subset selection with constitutive NNs to obtain sparse networks, and (Zijun Guo & Song, 2023) consider an adaptive re-weighting method. Of special importance is (Toh et al., 2023), which proposes an analytic solution for Bridge regression based on solving a penalized error formulation using a proximal operator approach, closely in line with this work.

3. Equivalent Formulation of L_p Regularization

Our starting point is the optimization problem of minimizing the empirical risk, or loss function $\mathcal{L}(\boldsymbol{w})$, with respect to the weights $\boldsymbol{w} \in \mathbb{R}^{N_w}$, where N_w is the total number of weights (including biases), subject to an L_p regularization term, $R_p(\boldsymbol{w}) = (\lambda_p/p)\,\|\boldsymbol{w}\|_p^p$, where $p > 0, \lambda_p \in \mathbb{R}^+$ and $\|\cdot\|_p$ is the p-norm. In this section we introduce a higher dimensional dual optimization problem, where the loss is regularized instead by

$$R_p(\boldsymbol{w}, \boldsymbol{s}) = \frac{\lambda_p}{2} \sum_i \left[s_i w_i^2 + K(s_i) \right], \tag{1}$$

where $w_i, s_i \in \mathbb{R}, i = 1, \ldots, N_w$. Here $K(s_i)$ is a function of s_i only, chosen such that the two regularization terms satisfy the equality $R_p(\boldsymbol{w}) = \min_{\boldsymbol{s}} R_p(\boldsymbol{w}, \boldsymbol{s})$. Specifically, for $p \neq 2$, one suitable choice for $K(s_i)$ is given by

$$K(s_i) = \frac{2-p}{p} s_i^{p/(p-2)}, \tag{2}$$

under the restriction that $s_i > 0$. In App. A, we prove formally that the extended optimization problem is equivalent to the original one, in the sense that they share the same global and local minima. By design, the minimum of the original optimization problem coincides with that of the extended one, namely,

$$\min_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) + \lambda_p R_p(\boldsymbol{w}) = \min_{\boldsymbol{w}, \boldsymbol{s}} \mathcal{L}(\boldsymbol{w}) + \lambda_p R_p(\boldsymbol{w}, \boldsymbol{s}). \tag{3}$$

Before we move on, it is important to note that $R_p(\boldsymbol{w}, \boldsymbol{s})$ is non-convex[1]. Therefore, even for $p \geq 1$ the extended optimization problem is non-convex.

The regularizer $R_p(w, s)$ is what is known as a *biconvex function* (Gorski et al., 2007). In simple terms, it is a convex function of w for any fixed $s > 0$ and vice versa. Biconvex functions can exhibit multiple local minima. Nevertheless, we refer the reader again to App. A for a formal proof that both local and global minima of the original and extended optimization problems coincide. Specifically, for $p < 1$, the generalized loss $\mathcal{L}(\boldsymbol{w}) + \lambda_p R_p(\boldsymbol{w}, \boldsymbol{s})$ will exhibit local minima at $w_i = 0$, where the global minimum of $R_p(w_i, s_i)$ is located. This is to be expected since it is also the case in the original formulation of the $p < 1$ norm.

[1]This can very easily be seen in the $w \to \infty$ limit, where R_p is given by $\lambda_p s w^2$, independent of K, and the hessian has eigenvalues $\pm w$.

4. p-norm Weight Decay

Having established that the empirical risk minimization problem with L_p regularization can be expressed as a biconvex optimization problem, we now turn to an important implication of this formulation, which is based on the Alternate Convex Search (ACS) algorithm. Alternate Convex Search is a common strategy for optimizing biconvex functions, in which we alternate between optimizing with respect to one variable while keeping the other fixed. In each step, standard convex optimization techniques can be used, and the problem is guaranteed to converge to a (possibly local) minimum (Gorski et al., 2007). Building upon this approach, in this section we derive a weight decay step, analogous to traditional L_2 weight decay, but extended to any p norm. We refer to this method as p-norm Weight Decay (pWD).

4.1. Proximal Operator Representation

As mentioned above, we will follow the ACS approach, where we optimize over $\boldsymbol{w}$ and $\boldsymbol{s}$ sequentially. In each $\boldsymbol{s}$ update step we set it to its optimal value,

$$\boldsymbol{s}_n = |\boldsymbol{w}_n|^{p-2}, \tag{4}$$

where the absolute value is taken element-wise, and the subscript n denotes the n-th iteration. Since we are dealing with sparsity inducing regularization, we expect some weights to vanish. For $p < 2$, we see that the s_i's corresponding to vanishing weights will diverge.

Moving now to optimize over $\boldsymbol{w}$, following the ACS approach we hold $\boldsymbol{s}$ fixed at last value $\boldsymbol{s}_n$. Note that $K(\boldsymbol{s}_n)$ is also fixed and does not impact the optimization with respect to $\boldsymbol{w}$. Therefore, we can effectively optimize $\mathcal{L} + (\lambda_p/2) \sum_i s_i w_i^2$. This second term is seemingly a standard L_2 regularization term, however, the possible divergence of some of the s_i's calls for a subtle treatment

From this point on, we shall focus on gradient based optimization methods, as they are most commonly used when training NNs. Taking any gradient based approach, we will have to include the gradient of R_p with respect to $\boldsymbol{w}$, which is given by

$$\boldsymbol{\nabla}_{\boldsymbol{w}} R_p(\boldsymbol{w}, \boldsymbol{s}_n) = \lambda_p \boldsymbol{s}_n \circ \boldsymbol{w} = \lambda_p |\boldsymbol{w}_n|^{p-2} \circ \boldsymbol{w}, \tag{5}$$

where "$\circ$" denotes element-wise multiplication. For $p < 1$, this gradient will become very large for small weights, rendering any finite learning rate approach unstable. Taking

instead a decoupled weight decay approach (Loshchilov & Hutter, 2019), we are tempted to write the $\boldsymbol{w}$ update step as

$$\boldsymbol{w} \leftarrow (\boldsymbol{w} - \alpha\delta\boldsymbol{w}) \circ (\mathbf{1} - \alpha\lambda_p \boldsymbol{s}_n), \tag{6}$$

where $\delta\boldsymbol{w}$ is either the gradient of $\mathcal{L}$, or any other adaptive step based on the unregularized loss $\mathcal{L}$, and $\alpha \in \mathbb{R}^+$ is the learning rate. This update rule, however, still does not ensure stability; weights that the regularization term drives to zero will be multiplied by a divergent negative weight decay factor, giving rise to an oscillatory behavior around 0.

To overcome this instability, we propose to use the proximal operator of $R_p(\boldsymbol{w}, \boldsymbol{s}_n)$ with respect to $\boldsymbol{w}$ at fixed $\boldsymbol{s}_n$. We review the basics of proximal operators in App. B, where we also derive the proximal gradient step for R_p,

$$\boxed{\boldsymbol{w} \leftarrow \frac{\boldsymbol{w} - \alpha\delta\boldsymbol{w}}{\mathbf{1} + \alpha\lambda_p \boldsymbol{s}_n} = \frac{\boldsymbol{w} - \alpha\delta\boldsymbol{w}}{\mathbf{1} + \alpha\lambda_p |\boldsymbol{w}_n|^{p-2}}\,.} \tag{7}$$

In the equation above, division is carried out element-wise. For $\alpha\lambda_p s_n \ll 1$, this is equivalent to the decoupled weight decay step in Eq. (6). However, here, the numerator is always larger than 1, driving the weights to 0 for $\alpha\lambda_p s_n \gg 1$, as desired. As seen in Eq. (7), $w_{n,i} = 0$ is a fixed point of the proximal gradient step for all $p < 2$. The stability of this fixed point is discussed in App. D, where we find it to be stable only for $p < 1$, similar to the original problem.

Eq. (7) is the main result of this work, and we refer to this method as *p-norm Weight Decay (pWD)*.

4.2. Toy Example

To demonstrate the challenges of L_p regularization, and the benefits of our approach, we start with a simple toy example. Consider the single variable regularized loss function

$$\mathcal{L}(w) = \frac{1}{2}(w-1)^2 + \frac{\lambda_p}{p}|w|^p, \qquad w \in \mathbb{R}. \tag{8}$$

For any $\lambda_p > 0$ and $p > 0$, the minimum of $\mathcal{L}(w)$ lies on the segment $w \in [0, 1]$. For $\lambda_p \geq 1$ and $p \leq 1$, the minimum is found at $w = 0$. For simplicity, we consider simple gradient descent as the update rule for w, leading to the equation

$$w_{t+1} = w_t - \alpha\left[w_t\left(1 + \lambda_p |w_t|^{p-2}\right) - 1\right]. \tag{9}$$

For $p < 1$, and $\lambda_p > 1$, the regularized loss gradient will drive the weight to $w = 0$, leading to a divergent gradient. We demonstrate this in Figure 1, where the evolution of the weight under gradient descent for $p = 0.6$, $\lambda_p = 1$, and a learning rate $\alpha = 0.1$ is shown as a cyan line. The weight starts flowing towards 0, until reaching a point where the p-norm gradient becomes too large and the weight changes sign, leading to an oscillatory behavior around 0. On the contrary, the orange lines represent the evolution of the weight under the update rule in Eq. (7). For the solid line we update $s_n = |w|^{-1.4}$ before each w_n step, which results in a smooth convergence to 0. For the dashed line we initialize $s_0 = 0.1$ and update $s_n = |w|^{-1.4}$ once every 20 w steps. We see that the weight converges smoothly to 0 in a step-wise pattern, without any oscillations. In the remainder of this work, we adopt the smoothest approach and update s at every w step.

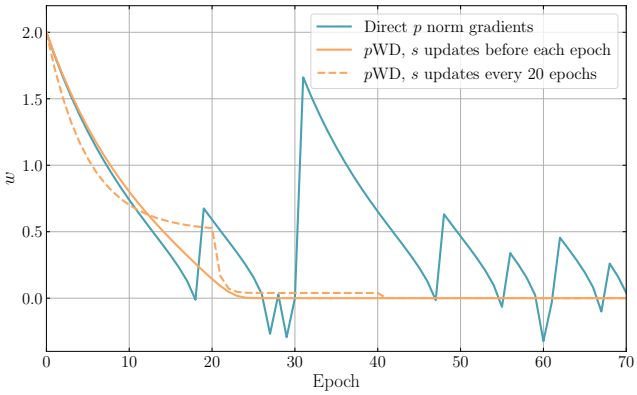

Figure 1: Toy example of weight evolution under gradient descent for the loss $\mathcal{L} = (w-1)^2/2 + \|w\|_p^p/p$. **Dotted line:** represents simple gradient descent where the norm is added directly to the gradient. The weight fails to converge to 0 due to the exploding gradient of the p-norm near 0. The **dashed line** represents the evolution of the weight under the update rule in Eq. (7), where we update s every 20 w steps. The **solid line** represents the evolution of the weight under the update rule in Eq. (7), where we update s at every w step. The latter is an implementation of *p-norm Weight Decay (pWD)*. We see that in both implementations of our method, the weight converges smoothly to 0.

4.3. The pWD Algorithm

Based on the discussion above, we can now present our proposed L_p weight decay algorithm, Algorithm 1. To limit the scope of this work, we will focus only on the case where $\boldsymbol{s}$ is updated at every $\boldsymbol{w}$ step. We note, however, that the algorithm can be easily modified to update $\boldsymbol{s}$ every n $\boldsymbol{w}$ steps, where n is a hyper-parameter. This modification, along with few other variants, are discussed in Sec. 6. In App. E, we provide an example `PyTorch` implementation of pWD based on the Adam optimizer.

Lines $1-8$ of Algorithm 1 are the usual steps for any gradient based optimizer, such as SGD (Robbins & Monro, 1951), Adam (Kingma & Ba, 2014), RMSprop (Tieleman et al., 2012) etc., encapsulated by the function OptimizerWeightUpdate(t, g_t), which may include momentum or higher moments. We have also explicitly included

Algorithm 1 Gradient Based pWD

1: **given** initial learning rate $\alpha \in \mathbb{R}^+$, weight decay regularization factor $\lambda_p \in \mathbb{R}^+$ weight norm number $p \in \mathbb{R}^+$, gradient-based optimization algorithm and its hyper-parameters
2: **initialize** time step $t \leftarrow 0$, parameter vector $\boldsymbol{w}_{t=0} \in \mathbb{R}^n$, schedule multiplier $\eta \in \mathbb{R}^+$
3: **repeat**
4: $t \leftarrow t + 1$
5: $g_t \leftarrow \boldsymbol{\nabla}\mathcal{L}(\boldsymbol{w}_{t-1})$
6: $\delta\boldsymbol{w}_t \leftarrow$OptimizerWeightUpdate$(g_t, t)$
7: $\eta_t \leftarrow$ SetScheduleMultiplier(t)
8: $\tilde{\boldsymbol{w}}_t \leftarrow \boldsymbol{w}_{t-1} - \eta_t \alpha \delta \boldsymbol{w}_t$
9: The pWD Step:

$$\boldsymbol{w} \leftarrow \frac{|\boldsymbol{w}_{t-1}|^{2-p}}{|\boldsymbol{w}_{t-1}|^{2-p} + \eta_t \alpha \lambda_p} \tilde{\boldsymbol{w}}_t$$

10: **until** stopping criterion is met
11: **return** optimized parameters $\boldsymbol{w}_t$

learning rate scheduling. The novelty appears at Line 9, where we impose the pWD weight decay step. This weight decay step is the same as the one in Eq. (7), assuming the auxiliary parameters $\boldsymbol{s}$ are set before every $\boldsymbol{w}$ step.

In App. C, we prove that gradient descent with the pWD step guarantees that the original loss function is non-increasing. We note that Lines 8 and 9 are split into two steps for clarity of presentation, but in practice can be carried out simultaneously to avoid the memory overhead of storing $\tilde{\theta}_t$. As discussed in Sec. 4.1 and App. D, the pWD step has a fixed point at $w_i = 0$ for all $p < 2$. **Regardless of the stability of this fixed point, it is important to stress that a parameter initialized at $w_i = 0$ will remain fixed to this value during training.** In the experiments presented in Sec. 5, we therefore avoid decaying parameter tensors that are initialized at 0, such as biases and batch normalization parameters.

5. Sparsity with pWD in Realistic Settings

In this section, we empirically test the performance of our pWD scheme. We use Adam (Kingma & Ba, 2014) as our base optimizer, and supplement it with the proposed pWD step. We refer to this optimizer as pAdam. A simple `PyTorch` implementation is described in App. E. Throughout this work, we keep the Adam hyper-parameters fixed at their default values. For all our experiments, we adopted a learning rate schedule that combines a linear warm-up phase with a subsequent cosine annealing. The precise experimental details, architectures, and hyper-parameters are given in App. F.

The experiments were conducted on two standard datasets: CIFAR-10 (Krizhevsky, 2012) and Tiny Shakespeare (Karpathy, 2015). We employed two deep learning models, for vision and text, namely ResNet18 (He et al., 2015) and NanoGPT (Karpathy, 2023); a character-level language based on GPT2 (Radford et al., 2019). We trained both architectures with a cross-entropy loss, and used sparsity and accuracy as our main validation matrices[2]. We trained our models using the pAdam optimizer for a range of p values, and using AdamW (Loshchilov & Hutter, 2019) as a point of reference for performance. For each optimizer and each p value choice, we scanned over both the learning rate and the weight decay λ_p. Importantly, we did not decay one dimensional parameter tensors, such as biases and batch normalization parameters. Such parameters are commonly initialized at 0, which, as mentioned above, will remain there during training under pWD. At the same time, they constitute a small fraction of the total number of parameters, and thus do not significantly affect the overall sparsity level.

5.1. Sparsity and Performance for pWD

The main results of this paper are shown in Figures 2 and 3, where we show the validation accuracy against sparsity, with sparsity defined as the fraction of weights smaller than 10^{-13}. It is evident that pAdam finds sparse network representations, with accuracy level gradually decreasing as sparsity increases. For ResNet18 trained on CIFAR10, we observe that models with sparsity as high as 99.5% have achieved accuracy higher than 90%, significantly higher than random guessing, which gives 10% accuracy. The highest sparsity level we find for accuracy drop of less than 2% relative to the best AdamW accuracy was 94.4% for ResNet18, and 89.9% for NanoGPT.

Different colors in Figures 2 and 3 represent different p norms. We find that the sparsest networks are obtained at values of $p < 1$ while the best generalizing networks are found for $1 < p < 2$. See discussion in Sec. 6 for possible explanations and improvement strategies. These results imply that for a given dataset and architecture, an optimal p may be found, under a choice of accuracy/sparsity trade-off. In the right panel of Figure 2, we show a concrete realization of this trade-off, whereby a loss of 1% in accuracy is equal to a 100% increase in sparsity, defined by

$$\text{Trade-off Metric} = \text{Val. Acc.}[\%] + \text{Sparsity}. \quad (10)$$

For ResNet18, the validation+Sparsity scatter exhibit a clear peak at around a sparsity of 80%, indicating on the optimal $p \simeq 1.2$ under such trade-off. The nanoGPT results are less conclusive, and in general less well behaved, but still the sparsity+accuracy scatter seem to be roughly constant up to sparsity of $\sim 90\%$, achieved by $p \geq 0.8$

[2]While accuracy might not be the first choice for token level language model practitioners, we find it suitable and intuitive for our character level language model experiments.

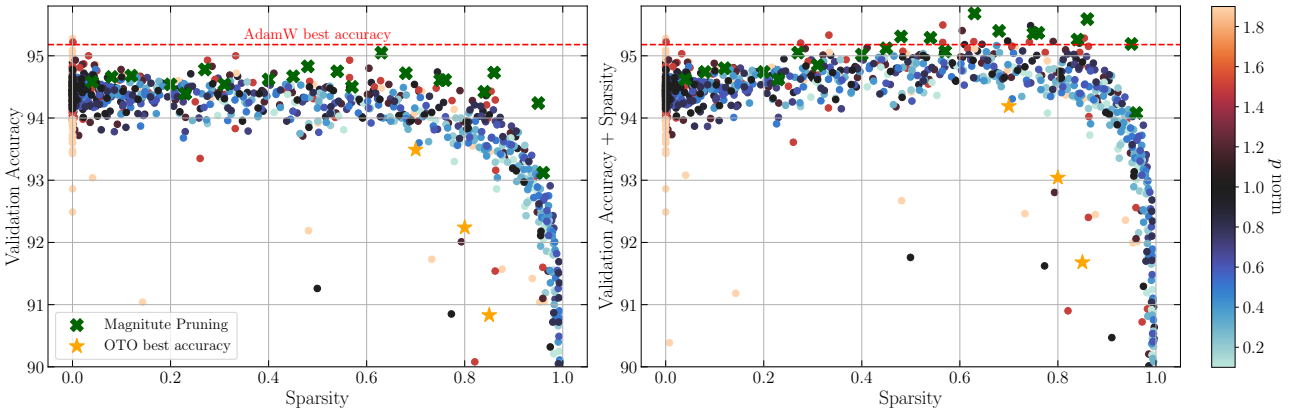

Figure 2: Validation accuracy vs. sparsity for ResNet18 trained on CIFAR-10. Each point represents a different instance of the network trained for 100 epochs, with a different choice of p, λ_p, and learning rate α. Points of different colors indicate different choices of p, optimizing over λ_p, α. The **dashed-red line** indicate the best accuracy achieved using AdamW. The **orange stars** indicate the best accuracy runs obtained using *Only Train Once* (*OTO*, Chen et al., 2021). The **green crosses** indicate the best accuracies obtained using *iterative magnitude pruning*. **Left:** Validation accuracy vs. sparsity. **Right:** Example of the accuracy/sparsity trade-off given in Eq. (10).

Table 1: Accuracies above a given sparsity level for ResNet-18 on CIFAR-10. Comparison of pWD to different sparsification methods.

SPARSITY	0%	70%	80%	90%
MP	95.18%	94.73%	94.73%	94.24%
OTO	95.18%	93.49%	92.24%	87.82%
pWD	95.28%	94.74%	94.43%	93.79%

5.2. Comparison with Other Methods

Next, we compare the performance of pWD to other sparsification methods. We focus on two methods for sparsificaiton during training. The first is the *Only Train Once* (*OTO*, Chen et al., 2021) method, which we apply only to the ResNet18 experiments. The second, which we apply to both ResNet18 and nanoGPT, is a simple iterative magnitude pruning (MP) method. There, we simply set to 0 all weights smaller than a certain threshold, once every fixed number of iterations. For both MP and OTO, we use AdamW as the base optimizer, and scan over the learning rate, weight decay, and one pruning hyper-parameter (pruning threshold for MP and pruning fraction for OTO). For both methods we start pruning after 10% of the training epochs, and increase the pruning threshold/fraction for the next 10% of the epochs. The results are shown in Figure 2 and in Tables 1 and 2. For ResNet18, we find that pWD outperforms OTO, but is inferior to MP until a sparsity of about 90%, where pWD retains higher accuracy. For nanoGPT, we find, much like for ResNet18, that MP outperforms pWD until a sparsity of about 95%, where pWD performs slightly better.

Overall, our findings indicate that pWD achieves results comparable to MP and surpasses the performance of OTO. It's important to note, however, that the MP approach involved an additional advantage: the use of AdamW combined with iterative setting of small weights to zero. This approach is akin to the Elastic Net (Zou & Hastie, 2005), where sparsity is induced by the L_1 term and optimization stabilization and generalization are aided by the L_2 term. In contrast, pWD employs a single regularization term, with the parameter p effectively balancing sparsity and generalization performance[3]. In the subsequent section, we will explore how pWD can be elegantly extended to simultaneously enhance both these aspects. As a simple demonstration of the potential of an extended pWD, we ran an additional nanoGPT experiment with $p = 0.8$, this time using AdamW instead of Adam. We fixed the L_2 weight decay of 2×10^{-3} and scanned over the learning rate and the pWD weight decay $\lambda_{0.8}$. For sparsity of 90% we obtained an accuracy of 57.15%. Improving over the pAdam result and surpassing the MP result.

6. Limitations of pWD and Possible Variations

This paper, with the goal of establishing pWD as a viable L_p regularization method, was focused on a specific imple-

[3]Since our goal was to highlight the utility of pWD alone, we did not combine it with other sparsification methods, but this is trivially done. In particular, combining iterative magnitude pruning and pWD is as simple as performing MP with regular WD.

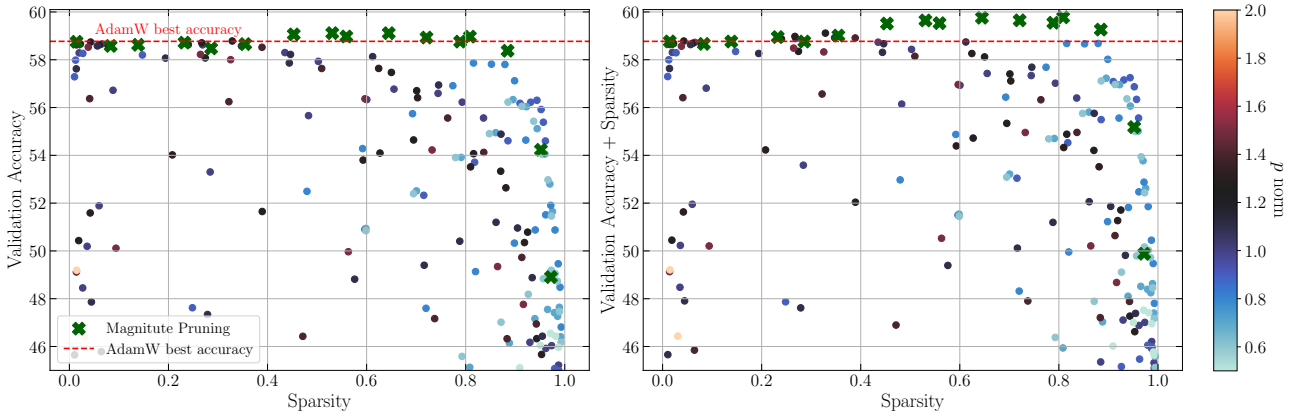

Figure 3: Validation accuracy vs. sparsity for nanoGPT trained on Tiny Shakespeare. Each point represents a different instance of the network trained for 5000 iterations, with a different choice of p, λ_p, and learning rate α. Points of different colors indicate different choices of p, optimizing over λ_p, α. The dashed-red line indicates the best accuracy achieved using AdamW. The **green crosses** indicate the best accuracies obtained using *iterative magnitude pruning*. **Left:** Validation accuracy vs. sparsity. **Right:** Example of the accuracy/sparsity trade-off given in Eq. (10).

Table 2: Accuracies above a given sparsity level for nanoGPT on Tiny Shakespeare. Comparison of pWD to different sparsification methods.

SPARSITY	0%	80%	90%	95%
MP	59.11%	58.97%	57.07%	54.23%
pWD	58.79%	57.93%	56.49%	55.92%

mentation of pWD. In this implementation, $\boldsymbol{s}$ is updated to its optimal value, given in Eq. (4), at every $\boldsymbol{w}$ update step. We identify two aspects of pWD that call for further investigation: The first is the existence of a fixed point at $w_i = 0$ for all $p < 2$, which is also a local minimum for $p < 1$. The second aspect is the generalization performance of the resulting networks. In this section we discuss these two aspects, and propose possible variations of pWD that might improve the performance of the resulting networks. The common theme of these variations is that they all involve richer dynamics, which comes with a price of increased complexity, and an increased number of hyper-parameters.

6.1. Avoiding the $w_i = 0$ Fixed Point

The $w_i = 0$ fixed point arises due to the large denominator in Eq. (7) whenever $w_i \to 0$. The disadvantage of this fixed point is especially important for parameters initialized at $w_i = 0$, which will remain frozen during training. At the same time, the existence of this fixed point is crucial for the algorithm to converge to the sparse solutions it was designed to find. A successful algorithm will therefore avoid getting stuck at the $w_i = 0$ local minimum, while still converging to $w_i = 0$ when appropriate. Below we discuss a few possible approaches to achieve this goal.

$\boldsymbol{s}$ dynamics: In the proposed pWD Algorithm 1, $\boldsymbol{s}$ in Eq. (7) is updated to its optimal value before every $\boldsymbol{w}$ step. Here, we suggest promoting $\boldsymbol{s}$ to a learnable parameter. From that perspective, currently at each epoch $\boldsymbol{s}$ is updated according to Eq. (4). Therefore, a weight initialized at $w_i = 0$ will force $s_i \to \infty$, which will in turn force w_i to remain at 0. From the perspective of a dynamical system, we can say that $\boldsymbol{s}$ is a 'fast' variable, which reaches optimal value before the 'slow' variable, $\boldsymbol{w}$, has had enough time to update its value. By 'slowing down' $\boldsymbol{s}$, we can avoid the $w = 0$ fixed point. We can initialize $\boldsymbol{s} = \mathbf{1}$, which means that the network starts evolving under a standard weight decay. Then, during training we can let $\boldsymbol{s}$ evolve in a desired pace towards its optimal value. This can be done either by updating $\boldsymbol{s}$ every n steps of $\boldsymbol{w}$ updates, or by applying an SGD-like update rule to $\boldsymbol{s}$. In this case, if a weight passes through $w_i = 0$, it will be able to continue evolving as long as s_i is not too large. We note that while this approach does come with a small memory overhead. The computational overhead is negligible as the gradients of $\boldsymbol{s}$ are trivial to compute and implement.

p scheduling: It is clear from Eq. (7) that for $p = 2$ there is no fixed point at $w_i = 0$ and we revert to the standard weight decay scheme. Having the network start training with $p = 2$ for a few epochs, and then gradually decreasing p towards a desired $p < 2$ value, would allow the network to avoid the $w_i = 0$ fixed point at initialization. At the same time, the network will still be able to converge to $w_i = 0$ at later stages as p decreases. Further, restarting p to $p = 2$ and

decreasing to a smaller value repeatedly, would allow the network to explore the parameter region around the $w_i = 0$ fixed point, and possibly escape it.

6.2. Generalization Performance:

One important observation from Figure 2 is that the best generalizing networks are found for $1 < p < 2$, while the most sparse networks are found for $p < 1$. One possible explanation is that $p > 1$ norms, such as AdamW, incur larger penalties on larger weights than smaller ones, in contrast to $p < 1$. The importance of regularizing large weights is well known (for example Loshchilov & Hutter, 2019), and perhaps $1 < p < 2$ provide a better balance between the two. This hypothesis is also supported by the performance of MP with AdamW, which is essentially a combination of $p = 2$ and $p = 0$ penalties.

Both p scheduling and $\boldsymbol{s}$ dynamics, discussed above, can potentially achieve as similar effect even when $p < 1$. For example, consider the case of p scheduling with restarts. Upon each restart, larger weights will be penalized more harshly, while the smaller ones will instead be regularized towards the end of a cycle when p is small. Alternatively, if we adopt slow $\boldsymbol{s}$ dynamics, large weights will be penalized until s reaches its optimal value (which is small for large weights).

Elastic Weight Decay: Another possible approach is to use a variation of elastic net proposed in (Zou & Hastie, 2005). In the original elastic net, the loss is regularized by a combination of L_1 and L_2 norms. Supposedly, this combination allows the network to benefit from the sparsity inducing properties of L_1 regularization, while still benefiting from the stability of L_2 regularization. Repeating the steps leading to Eq. (7), we can in principle add both an L_1 and an L_2 norm and achieve the following elastic net weight decay step

$$\boldsymbol{w} \leftarrow \frac{\boldsymbol{w} - \alpha\delta\boldsymbol{w}}{\mathbf{1} + \alpha\left(\lambda_1|\boldsymbol{w}|^{-1} + \lambda_2\right)} \,. \tag{11}$$

Moreover, the flexibility of our proposed pWD allows us to generalize the elastic net approach to any combination of $L_{p<2}$ norms, $\sum_p \lambda_p \|\boldsymbol{w}\|_p^p$. In which case, the weight decay step becomes

$$\boldsymbol{w} \leftarrow \frac{\boldsymbol{w} - \alpha\delta\boldsymbol{w}}{\mathbf{1} + \alpha\sum_p \lambda_p|\boldsymbol{w}|^{p-2}} \,. \tag{12}$$

While our derivation of Eq. (7) relied on the specific construction as presented in Eqs. (1) and (2), which is valid only for $p < 2$, we see no reason why Eq. (12) should not be valid for any $p > 0$[4]. In principle, optimizing with a combination of one $p < 1$ norm and one $p > 1$ norm, might provide a better balance between the two, and improve the generalization performance of the resulting network. As mentioned in the previous section, a verification of this prediction was tested on nanoGPT which essentially combined $p = 2$ and $p = 0.8$ weight decay. The results were superior to both MP and a single p implementation of pWD.

7. Conclusions

In this work, building upon the works of Frank & Friedman (1993) and Loshchilov & Hutter (2019), we developed a novel regularization-based sparsification scheme, which we dubbed *p-norm Weight Decay*. Our method, a proximal approximation of L_p regularization, dynamically drives weights to zero during training within a stable optimization framework. pWD is as simple to implement as any standard optimizer. It operates as a supplemental weight decay step and is, therefore, compatible with any modern optimizer. Additionally, it incurs negligible memory and computational overhead. Our ultimate goal is to incorporate pWD into popular deep learning frameworks such `PyTorch` and `Tensorflow`, therby to rendering sparse training as straightforward as using any modern optimizer.

Through empirical evaluation, we demonstrate that our approach enables performance gains and high levels of sparsity. Specifically, we are able to prune ResNet and NanoGPT models to extremely sparse configurations while retaining high accuracy. Our results clearly demonstrate that pWD provides an effective approach for network sparsification, competing with state-of-the-art methods in terms of maintaining accuracy while achieving highly sparse networks.

This work is, however, only a first step towards unveiling the full potential of pWD. Iterative magnitude pruning, for example, outperformed some aspects of pWD in the experiments presented in Sec. 5. As discussed in Sec. 6, we believe that the performance of pWD can be further improved by incorporating richer dynamics. We leave the thorough exploration of these dynamics to future work.

Going beyond the scope of NNs, pWD is essentially a noval gradient based approximated optimization approach for $p \leq 2$ norms. As such pWD can be implemented on many problems other than machine learning. One example may lie in Variational Quantum Circuits (Cerezo et al., 2021), where decreasing the number of parameters is desirable.

In conclusion, the straightforward implementation, flexibility, and potentially adaptive nature of pWD, have promise to stimulate new areas of investigation into optimizing neural networks and automated architecture design.

[4] In the case of simple SGD, the $w \neq 0$ fixed point of pWD is given by the $\boldsymbol{w}$ solving $0 = \nabla\mathcal{L}(\boldsymbol{w}) + \lambda_p|\boldsymbol{w}|^{p-2} \circ \boldsymbol{w}$. This is precisely the equation for the minimum of $\mathcal{L}(\boldsymbol{w}) + (\lambda_p/p)\|\boldsymbol{w}\|_p^p$, assuming $\mathcal{L}$ is convex, regardless of the value of p.

Impact Statement

The method suggested in this paper simplifies sparsification in neural networks training. Thereby, potentially making machine learning more efficient and accessible in environments with limited resources. By reducing energy and computational demands, our approach could have a wider impact, facilitating sustainable AI technology use across various sectors.

Acknowledgements

We would like to thank Andrey Gromov for useful discussions and Ioannis Mavrothalassitis for assisting us to derive some of the convergence proofs. NJO acknowledges support from the National Science Foundation under the grant No. PHY-1915314. NJO further thanks B. Nachman for computing resources. NL would like to thank G-Research for the award of a research grant, as well as the CERN-TH department for their hospitality during various stages of this work.

References

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S. C., Endo, S., Fujii, K., McClean, J. R., Mitarai, K., Yuan, X., Cincio, L., and Coles, P. J. Variational quantum algorithms. *Nature Reviews Physics*, 3(9): 625–644, August 2021. ISSN 2522-5820. doi: 10.1038/s42254-021-00348-9. URL http://dx.doi.org/10.1038/s42254-021-00348-9.

Chen, F., Shen, L., and Suter, B. Computing the proximity operator of the lp norm with 0 ¡ p ¡ 1. *IET Signal Processing*, 10(5):557–565, July 2016. ISSN 1751-9675. doi: 10.1049/iet-spr.2015.0244. Publisher Copyright: © The Institution of Engineering and Technology 2016.

Chen, T., Ji, B., Ding, T., Fang, B., Wang, G., Zhu, Z., Liang, L., Shi, Y., Yi, S., and Tu, X. Only train once: A one-shot neural network training and pruning framework. *CoRR*, abs/2107.07467, 2021. URL https://arxiv.org/abs/2107.07467.

Daubechies, I., Defrise, M., and Mol, C. D. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint, 2003.

de Vries, A. The growing energy footprint of artificial intelligence. *Joule*, 7(10):2191–2194, 2023. ISSN 2542-4351. doi: https://doi.org/10.1016/j.joule.2023.09.004. URL https://www.sciencedirect.com/science/article/pii/S2542435123003653.

Fan, J. and Li, R. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American statistical Association*, 96(456):1348–1360, 2001.

Fan, Q., Zurada, J. M., and Wu, W. Convergence of online gradient method for feedforward neural networks with smoothing l1/2 regularization penalty. *Neurocomputing*, 131:208–216, 2014.

Frank, L. E. and Friedman, J. H. A statistical view of some chemometrics regression tools. *Technometrics*, 35(2): 109–135, 1993.

Garrigos, G. and Gower, R. M. Handbook of convergence theorems for (stochastic) gradient methods, 2024.

Gorski, J., Pfeuffer, F., and Klamroth, K. Biconvex sets and optimization with biconvex functions: a survey and extensions. *Mathematical Methods of Operations Research*, 66(3):373–407, 2007. doi: 10.1007/s00186-007-0161-1. URL https://doi.org/10.1007/s00186-007-0161-1.

Gromov, A., Tirumala, K., Shapourian, H., Glorioso, P., and Roberts, D. A. The unreasonable ineffectiveness of the deeper layers, 2024.

Han, S., Pool, J., Tran, J., and Dally, W. J. Learning both weights and connections for efficient neural network. In *NIPS*, pp. 1135–1143, 2015.

Hassibi, B. and Stork, D. Second order derivatives for network pruning: Optimal brain surgeon. In Hanson, S., Cowan, J., and Giles, C. (eds.), *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992. URL https://proceedings.neurips.cc/paper_files/paper/1992/file/303ed4c69846ab36c2904d3ba8573050-Paper.pdf.

Hastie, T., Tibshirani, R., and Wainwright, M. *Statistical Learning with Sparsity: The Lasso and Generalizations*. Chapman & Hall/CRC, 2015. ISBN 1498712169.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*, pp. 770–778, Las Vegas, NV, USA, June 27-30 2016.

Hestness, J., Narang, S., Ardalani, N., Diamos, G. F., Jun, H., Kianinejad, H., Patwary, M. M. A., Yang, Y., and Zhou, Y. Deep learning scaling is predictable, empirically. *CoRR*, abs/1712.00409, 2017.

Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., and Peste, A. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks, 2021.

Kalchbrenner, N., Elsen, E., Simonyan, K., Noury, S., Casagrande, N., Lockhart, E., Stimberg, F., van den Oord, A., Dieleman, S., and Kavukcuoglu, K. Efficient neural audio synthesis. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pp. 2415–2424, 2018.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models, 2020.

Karpathy, A. char-rnn. https://github.com/karpathy/char-rnn, 2015.

Karpathy, A. nanogpt, 2023. URL https://github.com/karpathy/nanoGPT.

Khan, N., Shah, J., and Stavness, I. Bridgeout: stochastic bridge regularization for deep neural networks, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2014.

Krizhevsky, A. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Krogh, A. and Hertz, J. A simple weight decay can improve generalization. In Moody, J., Hanson, S., and Lippmann, R. (eds.), *Advances in Neural Information Processing Systems*, volume 4. Morgan-Kaufmann, 1991. URL https://proceedings.neurips.cc/paper_files/paper/1991/file/8eefcfdf5990e441f0fb6f3fad709e21-Paper.pdf.

Kurtic, E., Kuznedelev, D., Frantar, E., Goin, M., and Alistarh, D. Sparse fine-tuning for inference acceleration of large language models, 2023.

Kwon, W., Kim, S., Mahoney, M. W., Hassoun, J., Keutzer, K., and Gholami, A. A fast post-training pruning framework for transformers. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=0GRBKLBjJE.

Lasby, M., Golubeva, A., Evci, U., Nica, M., and Ioannou, Y. Dynamic sparse training with structured sparsity, 2024.

LeCun, Y., Denker, J., and Solla, S. Optimal brain damage. In Touretzky, D. (ed.), *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989. URL https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.

Liu, Y., Zhang, H. H., Park, C., and Ahn, J. Support vector machines with adaptive lq penalty. *Computational Statistics & Data Analysis*, 51(12):6380–6394, 2007.

Loría, J. and Bhadra, A. Sure-tuned bridge regression, 2023.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *Proceedings of the Seventh International Conference on Learning Representations*, 2019.

McCulloch, J. A., Pierre, S. R. S., Linka, K., and Kuhl, E. On sparse regression, lp-regularization, and automated model discovery, 2023.

Mishra, A., Latorre, J. A., Pool, J., Stosic, D., Stosic, D., Venkatesh, G., Yu, C., and Micikevicius, P. Accelerating sparse deep neural networks, 2021.

Moody, J. The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In Moody, J., Hanson, S., and Lippmann, R. (eds.), *Advances in Neural Information Processing Systems*, volume 4. Morgan-Kaufmann, 1991. URL https://proceedings.neurips.cc/paper_files/paper/1991/file/d64a340bcb633f536d56e51874281454-Paper.pdf.

Narang, S., Diamos, G. F., Sengupta, S., and Elsen, E. Exploring sparsity in recurrent neural networks. *CoRR*, abs/1704.05119, 2017.

Nowlan, S. J. and Hinton, G. E. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4:473–493, 1992. URL https://api.semanticscholar.org/CorpusID:5597033.

O'Brien, C. and Plumbley, M. D. Inexact proximal operators for ℓ_p-quasinorm minimization. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4724–4728, 2018. doi: 10.1109/ICASSP.2018.8462524.

OpenAI, :, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M.,

Belgum, J., Bello, I., Berdine, J., Bernadett-Shapiro, G., Berner, C., Bogdonoff, L., Boiko, O., Boyd, M., Brakman, A.-L., Brockman, G., Brooks, T., Brundage, M., Button, K., Cai, T., Campbell, R., Cann, A., Carey, B., Carlson, C., Carmichael, R., Chan, B., Chang, C., Chantzis, F., Chen, D., Chen, S., Chen, R., Chen, J., Chen, M., Chess, B., Cho, C., Chu, C., Chung, H. W., Cummings, D., Currier, J., Dai, Y., Decareaux, C., Degry, T., Deutsch, N., Deville, D., Dhar, A., Dohan, D., Dowling, S., Dunning, S., Ecoffet, A., Eleti, A., Eloundou, T., Farhi, D., Fedus, L., Felix, N., Fishman, S. P., Forte, J., Fulford, I., Gao, L., Georges, E., Gibson, C., Goel, V., Gogineni, T., Goh, G., Gontijo-Lopes, R., Gordon, J., Grafstein, M., Gray, S., Greene, R., Gross, J., Gu, S. S., Guo, Y., Hallacy, C., Han, J., Harris, J., He, Y., Heaton, M., Heidecke, J., Hesse, C., Hickey, A., Hickey, W., Hoeschele, P., Houghton, B., Hsu, K., Hu, S., Hu, X., Huizinga, J., Jain, S., Jain, S., Jang, J., Jiang, A., Jiang, R., Jin, H., Jin, D., Jomoto, S., Jonn, B., Jun, H., Kaftan, T., Łukasz Kaiser, Kamali, A., Kanitscheider, I., Keskar, N. S., Khan, T., Kilpatrick, L., Kim, J. W., Kim, C., Kim, Y., Kirchner, H., Kiros, J., Knight, M., Kokotajlo, D., Łukasz Kondraciuk, Kondrich, A., Konstantinidis, A., Kosic, K., Krueger, G., Kuo, V., Lampe, M., Lan, I., Lee, T., Leike, J., Leung, J., Levy, D., Li, C. M., Lim, R., Lin, M., Lin, S., Litwin, M., Lopez, T., Lowe, R., Lue, P., Makanju, A., Malfacini, K., Manning, S., Markov, T., Markovski, Y., Martin, B., Mayer, K., Mayne, A., McGrew, B., McKinney, S. M., McLeavey, C., McMillan, P., McNeil, J., Medina, D., Mehta, A., Menick, J., Metz, L., Mishchenko, A., Mishkin, P., Monaco, V., Morikawa, E., Mossing, D., Mu, T., Murati, M., Murk, O., Mély, D., Nair, A., Nakano, R., Nayak, R., Neelakantan, A., Ngo, R., Noh, H., Ouyang, L., O'Keefe, C., Pachocki, J., Paino, A., Palermo, J., Pantuliano, A., Parascandolo, G., Parish, J., Parparita, E., Passos, A., Pavlov, M., Peng, A., Perelman, A., de Avila Belbute Peres, F., Petrov, M., de Oliveira Pinto, H. P., Michael, Pokorny, Pokrass, M., Pong, V., Powell, T., Power, A., Power, B., Proehl, E., Puri, R., Radford, A., Rae, J., Ramesh, A., Raymond, C., Real, F., Rimbach, K., Ross, C., Rotsted, B., Roussez, H., Ryder, N., Saltarelli, M., Sanders, T., Santurkar, S., Sastry, G., Schmidt, H., Schnurr, D., Schulman, J., Selsam, D., Sheppard, K., Sherbakov, T., Shieh, J., Shoker, S., Shyam, P., Sidor, S., Sigler, E., Simens, M., Sitkin, J., Slama, K., Sohl, I., Sokolowsky, B., Song, Y., Staudacher, N., Such, F. P., Summers, N., Sutskever, I., Tang, J., Tezak, N., Thompson, M., Tillet, P., Tootoonchian, A., Tseng, E., Tuggle, P., Turley, N., Tworek, J., Uribe, J. F. C., Vallone, A., Vijayvergiya, A., Voss, C., Wainwright, C., Wang, J. J., Wang, A., Wang, B., Ward, J., Wei, J., Weinmann, C., Welihinda, A., Welinder, P., Weng, J., Weng, L., Wiethoff, M., Willner, D., Winter, C., Wolrich, S., Wong, H., Workman, L., Wu, S., Wu, J., Wu, M., Xiao, K., Xu, T., Yoo, S., Yu, K., Yuan, Q., Zaremba, W., Zellers, R., Zhang, C., Zhang, M., Zhao, S., Zheng, T., Zhuang, J., Zhuk, W., and Zoph, B. Gpt-4 technical report, 2023.

Park, C. and Yoon, Y. J. Bridge regression: adaptivity and group selection. *Journal of Statistical Planning and Inference*, 141(11):3506–3519, 2011.

Plaut, D. C., Nowlan, S. J., and Hinton, G. E. Experiments on learning back propagation. Technical Report CMU–CS–86–126, Carnegie–Mellon University, Pittsburgh, PA, 1986.

Polson, N. G., Scott, J. G., and Windle, J. The bayesian bridge, 2012.

Polson, N. G., Scott, J. G., and Windle, J. The bayesian bridge. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 76(4):713–733, 2014.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. 2019.

Reed, R. Pruning algorithms-a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993. doi: 10.1109/72.248452.

Robbins, H. and Monro, S. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3): 400–407, 1951. ISSN 00034851. URL http://www.jstor.org/stable/2236626.

Sleem, O. M., Ashour, M. E., Aybat, N. S., and Lagoa, C. M. Lp quasi-norm minimization: algorithm and applications. *EURASIP Journal on Advances in Signal Processing*, 2024(1):22, 2024. doi: 10.1186/s13634-024-01114-6. URL https://doi.org/10.1186/s13634-024-01114-6.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. Inception-v4, inception-resnet and the impact of residual connections on learning, 2016.

Theis, L., Korshunova, I., Tejani, A., and Huszár, F. Faster gaze prediction with dense networks and fisher pruning. *CoRR*, abs/1801.05787, 2018. URL http://arxiv.org/abs/1801.05787.

Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996.

Tieleman, T., Hinton, G., et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4 (2):26–31, 2012.

Toh, K.-A., Molteni, G., and Lin, Z. Deterministic bridge regression for compressive classification. *Information Sciences*, 648:119505, 2023. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2023.119505. URL https://www.sciencedirect.com/science/article/pii/S0020025523010903.

Ullrich, K., Meeds, E., and Welling, M. Soft weight-sharing for neural network compression. *CoRR*, abs/1702.04008, 2017.

Valin, J. and Skoglund, J. Lpcnet: Improving neural speech synthesis through linear prediction. *CoRR*, abs/1810.11846, 2018. URL http://arxiv.org/abs/1810.11846.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., and Kavukcuoglu, K. Wavenet: A generative model for raw audio. *The 9th ISCA Speech Synthesis Workshop*, pp. 125, 2016.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pp. 6000–6010, 2017.

Wei, C., Lee, J. D., Liu, Q., and Ma, T. Regularization matters: Generalization and optimization of neural nets v.s. their induced kernel. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/8744cf92c88433f8cb04a02e6db69a0d-Paper.pdf.

Xu, Z., Zhang, H., Wang, Y., Chang, X., and Liang, Y. L 1/2 regularization. *Science China Information Sciences*, 53:1159–1169, 2010.

Xu, Z., Chang, X., Xu, F., and Zhang, H. $l_{1/2}$ regularization: A thresholding representation theory and a fast solver. *IEEE Transactions on Neural Networks and Learning Systems*, 23(7):1013–1027, 2012. doi: 10.1109/TNNLS.2012.2197412.

Yang, D. and Liu, Y. L1/2 regularization learning for smoothing interval neural networks: Algorithms and convergence analysis. *Neurocomputing*, 272:122–129, 2018.

Yin, L., Li, G., Fang, M., Shen, L., Huang, T., Wang, Z., Menkovski, V., Ma, X., Pechenizkiy, M., and Liu, S. Dynamic sparsity is channel-level sparsity learner, 2023.

Zhang, C.-H. Nearly unbiased variable selection under minimax concave penalty. *The Annals of Statistics*, 38 (2):894–942, 2010. ISSN 00905364, 21688966. URL http://www.jstor.org/stable/25662264.

Zijun Guo, Mengxing Chen, Y. F. and Song, Y. A general adaptive ridge regression method for generalized linear models: an iterative re-weighting approach. *Communications in Statistics - Theory and Methods*, 52(18):6420–6443, 2023. doi: 10.1080/03610926.2022.2028841. URL https://doi.org/10.1080/03610926.2022.2028841.

Zou, H. and Hastie, T. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 67(2):301–320, 2005.

A. Proof of the Extended Problem Equivalence

In this appendix, we prove the equivalence between the original and the extended optimization problems. First, we show that $R_p(\boldsymbol{w}, \boldsymbol{s})$ in Eq. (1) with K as in Eq. (2) satisfies $R_p(\boldsymbol{w}) = \min_{\boldsymbol{s}} R_p(\boldsymbol{w}, \boldsymbol{s})$, provided $s_i > 0$ and $0 < p < 2$. We want to show that

$$R_p(\boldsymbol{w}) = \frac{\lambda_p}{p} \sum_i |w_i|^p = \min_{\boldsymbol{s}>0} \frac{\lambda_p}{2} \sum_i \left[w_i^2 s_i + \frac{2-p}{p} s^{p/(p-2)} \right] = \min_{\boldsymbol{s}>0} R_p(\boldsymbol{w}, \boldsymbol{s}) \,. \tag{13}$$

It is enough to show for a single component since the problem is separable. By taking second partial derivatives, the bi-convexity of $R_p(w, s > 0)$ is established. The minimum of $R_p(w, s > 0)$ at fixed w is therefore unique and can be found by setting $\partial R_p(w, s > 0)/\partial s = 0$. This gives

$$\frac{\partial R_p(w, s_*)}{\partial s_*} = \frac{\lambda_p}{2} \left[w^2 - s_*^{2/(p-2)} \right] = 0 \quad \Rightarrow \quad s_* = |w|^{p-2} \quad \Rightarrow \quad R_p(w, s_*) = \frac{\lambda_p}{p} |w|^p \,. \tag{14}$$

This shows that the minimum of $R_p(w, s > 0)$ is indeed $R_p(w)$, and therefore the equivalence in Eq. (13) holds. For future reference, we note that the minimum of $R_p(w, s > 0)$ at fixed w is unique and a continuous function of w.

The equivalence of the optimization problem is stated in the following theorem.

Theorem A.1. *Let $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$, let $R : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ be a smooth bi-convex function. Define $F(\boldsymbol{w}, \boldsymbol{s}) = \mathcal{L}(\boldsymbol{w}) + R(\boldsymbol{w}, \boldsymbol{s})$, and $\hat{s}(\cdot) = \operatorname{argmin}_{\boldsymbol{s}} R(\cdot, \boldsymbol{s})$. Assuming $\hat{s}$ is continuous, the following holds:*

(i) Let $(\boldsymbol{w}^*, \boldsymbol{s}^*)$ be a local minimum of $F(\boldsymbol{w}, \boldsymbol{s})$. Then, $\boldsymbol{w}^*$ is a local minimum of $F(\boldsymbol{w}, \hat{s}(\boldsymbol{w}))$.

(ii) Let $\boldsymbol{w}^*$ be a local minimum of $F(\boldsymbol{w}, \hat{s}(\boldsymbol{w}))$. Then, $(\boldsymbol{w}^*, \boldsymbol{s}^*)$ is a local minimum of $F(\boldsymbol{w}, \boldsymbol{s})$ with $\boldsymbol{s}^* = \hat{s}(\boldsymbol{w}^*)$.

(iii) In particular, the global minimum of $F(\boldsymbol{w}, \hat{s}(\boldsymbol{w}))$ is the same as the global minimum of $F(\boldsymbol{w}, \boldsymbol{s})$ and occurs at the same value of $\boldsymbol{w}$.

Proof.

(i) Assume that $(\boldsymbol{w}^*, \boldsymbol{s}^*)$ is a local minimum of $F(\boldsymbol{w}, \boldsymbol{s})$. By the biconvexity of R, we have that $F(\boldsymbol{w}^*, \boldsymbol{s})$ is convex in $\boldsymbol{s}$ for any fixed $\boldsymbol{w}^*$. Therefore, $\boldsymbol{s}^* = \hat{s}(\boldsymbol{w}^*)$ is the unique minimizer of $F(\boldsymbol{w}^*, \boldsymbol{s})$ for any fixed $\boldsymbol{w}^*$. We therefore know that the there is a local minimum for $F(\boldsymbol{w}, \hat{s}(\boldsymbol{w}))$ at $(\boldsymbol{w}^*, \boldsymbol{s}^* = \hat{s}(\boldsymbol{w}^*))$. Since this is a local minimum, $\exists \epsilon_w > 0, \epsilon_s > 0$ such that $\forall \boldsymbol{w}, \boldsymbol{s}$ with $\|\boldsymbol{w} - \boldsymbol{w}^*\| < \epsilon_w$ and $\|\boldsymbol{s} - \boldsymbol{s}^*\| < \epsilon_s$ we have $F(\boldsymbol{w}, \boldsymbol{s}) \geq F(\boldsymbol{w}^*, \boldsymbol{s}^*)$. To prove that $\boldsymbol{w}^*$ is a local minimum of $F(\boldsymbol{w}, \hat{s}(\boldsymbol{w}))$, we need to show that $\exists \epsilon > 0$ such that $\forall \boldsymbol{w}$ with $\|\boldsymbol{w} - \boldsymbol{w}^*\| < \epsilon_w$ we have $F(\boldsymbol{w}, \hat{s}(\boldsymbol{w})) \geq F(\boldsymbol{w}^*, \boldsymbol{s}^*)$. By the continuity if $\hat{s}$, $\lim_{\boldsymbol{w} \to \boldsymbol{w}^*} \hat{s}(\boldsymbol{w}) = \boldsymbol{s}^*$. Therefore, for small enough $\|\boldsymbol{w} - \boldsymbol{w}^*\|$, we have $\|\hat{s}(\boldsymbol{w}) - \boldsymbol{s}^*\| < \epsilon_s$. Thus, there exists a neighborhood of $\boldsymbol{w}^*$, $0 < \delta \leq \min(\epsilon, \epsilon_w)$ such that $\forall \boldsymbol{w}$ with $\|\boldsymbol{w} - \boldsymbol{w}^*\| < \delta$ we have $\|\hat{s}(\boldsymbol{w}) - \boldsymbol{s}^*\| < \epsilon_s$ and therefore $F(\boldsymbol{w}, \hat{s}(\boldsymbol{w})) \geq F(\boldsymbol{w}^*, \boldsymbol{s}^*)$.

(ii) Assume next that $\boldsymbol{w}^*$ is a local minimum of $F(\boldsymbol{w}, \hat{s}(\boldsymbol{w}))$. Meaning that $\exists \epsilon_W > 0$ such that $\forall \boldsymbol{w}$ with $\|\boldsymbol{w} - \boldsymbol{w}^*\| < \epsilon_w$ we have $F(\boldsymbol{w}, \hat{s}(\boldsymbol{w})) \geq F(\boldsymbol{w}^*, \hat{s}(\boldsymbol{w}^*))$. From the definition of $\hat{s}(\boldsymbol{w})$, we know that $F(\boldsymbol{w}, \boldsymbol{s}) \geq F(\boldsymbol{w}, \hat{s}(\boldsymbol{w}))$. Therefore, $\forall \boldsymbol{w}$ with $\|\boldsymbol{w} - \boldsymbol{w}^*\| < \epsilon_w$ we have $F(\boldsymbol{w}, \boldsymbol{s}) \geq F(\boldsymbol{w}^*, \hat{s}(\boldsymbol{w}^*))$, making $(\boldsymbol{w}^*, \hat{s}(\boldsymbol{w}^*))$ a local minimum of $F(\boldsymbol{w}, \boldsymbol{s})$.

(iii) To show the value of the minima coincide, we use the definition of $\hat{s}(\boldsymbol{w})$ to write

$$\min_{\boldsymbol{w}} F(\boldsymbol{w}, \hat{s}(\boldsymbol{w})) = \min_{\boldsymbol{w}} \left[\min_{\boldsymbol{s}} F(\boldsymbol{w}, \boldsymbol{s}) \right] = \min_{\boldsymbol{w}, \boldsymbol{s}} F(\boldsymbol{w}, \boldsymbol{s}).$$

Next, assume that $\boldsymbol{w}^*$ is the global minimum of $F(\boldsymbol{w}, \hat{s}(\boldsymbol{w}))$. Then we $F(\boldsymbol{w}, \boldsymbol{s}) \geq F(\boldsymbol{w}, \hat{s}(\boldsymbol{w})) \geq F(\boldsymbol{w}^*, \hat{s}(\boldsymbol{w}^*))$, meaning, $(\boldsymbol{w}^*, \hat{s}(\boldsymbol{w}^*))$ is a global minimum of $F(\boldsymbol{w}, \boldsymbol{s})$. The other direction is due to the following. Since $F(\boldsymbol{w}, \boldsymbol{s}) \geq F(\boldsymbol{w}, \hat{s}(\boldsymbol{w}))$ the minimum of $F(\boldsymbol{w}, \boldsymbol{s})$ always satisfies $s = \hat{s}(\boldsymbol{w})$. Therefore, the global minimum of $F(\boldsymbol{w}, \boldsymbol{s})$ is the same as the global minimum of $F(\boldsymbol{w}, \hat{s}(\boldsymbol{w}))$.

□

B. Proximal Operators

The Proximal Operator of a function $f(w)$ is the functional

$$\text{prox}_f(w) = \underset{u}{\text{argmin}} \left[f(u) + \frac{1}{2}(u-w)^2 \right]. \tag{15}$$

Meaning, for any function f, it returns the u that minimizes $f(u) + (u-w)^2/2$. Generalization from 1D to any space are trivial.

In the case of regularized loss, proximal operators become handy once we observe the following. Say that the loss is decomposed into an unregularized loss $\mathcal{L}_0$ and a regularizer R, namely $\mathcal{L} = \mathcal{L}_0 + R$. For any $\alpha > 0$, w^* is an extremum of ℓ if and only if

$$w^* = \text{prox}_{\alpha R}(w^* - \alpha \nabla \mathcal{L}_0(w^*)) = \underset{u}{\text{argmin}} \left[\alpha R(u) + \frac{1}{2}(u + \alpha \nabla \mathcal{L}_0(w^*) - w^*)^2 \right]. \tag{16}$$

Based on the definition of the proximal operator, it is a manner of simple algebra to show the above expression is equivalent to $\nabla \mathcal{L}(w^*) = 0$. In case both $\mathcal{L}_0$ and R are convex, w^* is therefore the global minimum of $\mathcal{L}$.

In the context of learning, one can iteratively obtain w^* through the sequence

$$w^{(t+1)} = \text{prox}_{\alpha R} \left[w^{(t)} - \alpha \nabla \mathcal{L}_0 \left(w^{(t)} \right) \right]. \tag{17}$$

In this context, α is identified with the learning rate. More generality, if we use some optimization algorithm to update our weights (e.g. Adam), such that $w \leftarrow w - \alpha \cdot \delta w$. To incorporate a proximal operator of R as regularization, we will simply update w as $w \leftarrow \text{prox}_{\alpha R}(w - \alpha \cdot \delta w)$.

For the case of L_2 regularization, the proximal operator is given in closed form by

$$\text{prox}_{\alpha \lambda_2 |\cdot|^2/2}(w) = \frac{w}{1 + \alpha \lambda_2}, \tag{18}$$

this is the result we have used in deriving Eq. (7). For completeness, the proximal operator for the L_1 norm is known as the soft-thresholding operator, and is given by

$$\text{prox}_{\alpha \lambda_1 |\cdot|}(w) = \text{sign}(w) \max \{ |w| - \alpha \lambda_1, 0 \}. \tag{19}$$

C. Non-increasing Loss under pWD

In this appendix, we show that the original loss function is non-increasing under the pWD step.

Lemma C.1. *Let $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$ be a continuous differentiable function,, assume the gradient of $\mathcal{L}$ is Lipschitz continuous with constant L, namely $\|\nabla \mathcal{L}(\boldsymbol{w}) - \nabla \mathcal{L}(\boldsymbol{v})\| \leq L \|\boldsymbol{w} - \boldsymbol{v}\|$. Let $R : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Define $F(\boldsymbol{w}) = \mathcal{L}(\boldsymbol{w}) + R(\boldsymbol{w})$.*

The sequence $\boldsymbol{w}^{(t+1)} = \text{prox}_{\alpha R}(\boldsymbol{w}^{(t)} - \alpha \nabla \mathcal{L}(\boldsymbol{w}^{(t)}))$ satisfies

(i)

$$R(\boldsymbol{w}^{(t+1)}) \leq R(\boldsymbol{w}^{(t)}) - \frac{1}{2\alpha} \|\boldsymbol{w}^{(t+1)} - \boldsymbol{w}^{(t)}\|^2 - \left\langle \nabla \mathcal{L}(\boldsymbol{w}^{(t)}), \boldsymbol{w}^{(t+1)} - \boldsymbol{w}^{(t)} \right\rangle. \tag{20}$$

(ii)

$$F(\boldsymbol{w}^{(t+1)}) \leq F(\boldsymbol{w}^{(t)}) - \frac{1 - \alpha L}{2\alpha} \|\boldsymbol{w}^{(t+1)} - \boldsymbol{w}^{(t)}\|^2. \tag{21}$$

Proof. See (Garrigos & Gower, 2024, Thm. 11.3) □

We prove the following theorem.

Theorem C.2. *Let $\mathcal{L}: \mathbb{R}^n \to \mathbb{R}$ be a continuous differentiable function, assume the gradient of $\mathcal{L}$ is Lipschitz continuous with constant L, namely $\|\nabla\mathcal{L}(\boldsymbol{w}) - \nabla\mathcal{L}(\boldsymbol{v})\| \leq L\|\boldsymbol{w} - \boldsymbol{v}\|$. Define $F(\boldsymbol{w}) = \mathcal{L}(\boldsymbol{w}) + (\lambda_p/p)\,\|\boldsymbol{w}\|_p^p$. Then, the sequence*

$$\boldsymbol{w}^{(t+1)} = \frac{\boldsymbol{w}^{(t)} - \alpha\nabla\mathcal{L}(\boldsymbol{w}^{(t)})}{1 + \alpha\lambda_p|\boldsymbol{w}^{(t)}|^{p-2}}$$

is such that

$$F(\boldsymbol{w}^{(t+1)}) \leq F(\boldsymbol{w}^{(t)}) - \frac{1 - \alpha_t L}{2\alpha_t}\|\boldsymbol{w}^{(t+1)} - \boldsymbol{w}^{(t)}\|^2.$$

Clearly, it is non-increasing, provided $\alpha_t \leq 1/L$.

Proof. We will use the generalized loss function $F(\boldsymbol{w}, \boldsymbol{s}) = \mathcal{L}(\boldsymbol{w}) + R_p(\boldsymbol{w}, \boldsymbol{s})$, where $R_p(\boldsymbol{w}, \boldsymbol{s})$ is defined in Eq. (1). Importantly, $F(\boldsymbol{w}) = \min_{\boldsymbol{s}} F(\boldsymbol{w}, \boldsymbol{s})$. We assume the following update rule:

$$\boldsymbol{w}^{(t+1)} = \text{prox}_{\alpha_t R_p(\cdot, \boldsymbol{s}_t)}\left(\boldsymbol{w}^{(t)} - \alpha_t\nabla\mathcal{L}(\boldsymbol{w}^{(t)})\right) = \frac{\boldsymbol{w}^{(t)} - \alpha_t\nabla\mathcal{L}(\boldsymbol{w}^{(t)})}{1 + \lambda_p\alpha_t\boldsymbol{s}^{(t)}} \quad , \quad \boldsymbol{s}^{(t+1)} = |\boldsymbol{w}^{(t+1)}|^{p-2},$$

where all the operations are done element-wise. We note that given the update rule of $\boldsymbol{s}$ above, the update rule of $\boldsymbol{w}$ is the same as the one in the theorem. At fixed $\boldsymbol{s} = \boldsymbol{s}^{(t)}$, we can apply Lemma C.1 to $F(\boldsymbol{w}, \boldsymbol{s}^{(t)})$, and obtain

$$F(\boldsymbol{w}^{(t+1)}, \boldsymbol{s}^{(t)}) \leq F(\boldsymbol{w}^{(t)}, \boldsymbol{s}^{(t)}) - \frac{1 - \alpha_t L}{2\alpha_t}\|\boldsymbol{w}^{(t+1)} - \boldsymbol{w}^{(t)}\|^2.$$

From App. A and Eq. (14), we know that the update rule for $\boldsymbol{s}$ is such that $F(\boldsymbol{w}^{(t)}) = F(\boldsymbol{w}^{(t)}, \boldsymbol{s}^{(t)}) \leq F(\boldsymbol{w}^{(t)}, \boldsymbol{s})$ for any $\boldsymbol{s}$. Therefore, we have

$$F(\boldsymbol{w}^{(t+1)}) \leq F(\boldsymbol{w}^{(t)}) - \frac{1 - \alpha_t L}{2\alpha_t}\|\boldsymbol{w}^{(t+1)} - \boldsymbol{w}^{(t)}\|^2.$$

□

We note that while very similar steps are the base for the proof of general convergence of proximal gradient methods (Garrigos & Gower, 2024, E.g. Thm. 11.3), our case is more involved due to the $\boldsymbol{s}$ dependence of the proximal operator.

D. Stability of the $w_i = 0$ Fixed Point

In this appendix, we discuss the stability of the $w_i = 0$ fixed point of the proximal gradient step. We will focus on a single weight case, the generalization to a higher dimension follows trivially. Starting from the proximal gradient step in Eq. (7), we want to ask whether a point arbitrarily close to $w = 0$ will be driven to $w = 0$ by the proximal gradient step. We will show that this is indeed the case for $p < 1$, while for $p > 1$ the fixed point is unstable. We assume that $w = 0$ is not the global minimum of the unregularized loss, and assume that the weight prior to the proximal gradient step is small compared to the weight update by the unregularized loss, namely,

$$w = \epsilon \quad : \quad |\epsilon| \ll \alpha|\delta w|. \tag{22}$$

This means that the current proximal gradient step is given by

$$w \leftarrow \frac{\epsilon - \alpha\delta w}{1 + \alpha\lambda_p\epsilon^{p-2}} \simeq -\frac{\alpha\epsilon^{2-p}\delta w}{\epsilon^{2-p} + \alpha\lambda_p} \simeq -\frac{\delta w\epsilon^{2-p}}{\lambda_p}. \tag{23}$$

In the last step we have assumed that $\alpha\lambda_p\epsilon^{p-2} \gg 1$, which is a necessary condition for the proximal gradient step to drive the weight to zero. The ratio of the updated weight to the original weight is thus given by

$$\left|\frac{w}{\epsilon}\right| \simeq \frac{|\delta w|}{\lambda_p}|\epsilon|^{1-p}. \tag{24}$$

For sufficiently small $|\epsilon|$, the above ratio is smaller than 1 for $p < 1$, and larger than 1 for $p > 1$. This means that $w = 0$ is a stable fixed point for $p < 1$, and an unstable for $p > 1$. For $p = 1$, the fixed point is stable for $|\delta w| < \lambda_1$, as it should for L_1 regularization.

E. pAdam Code

Here, we provide an example for implementing pWD on the standard Adam algorithm.

Listing 1: `PyTorch` pAdam optimizer implementation

```
class pAdam(torch.optim.AdamW):
def __init__(self, params, lr=1e-3, betas=(0.9, 0.999), eps=1e-8, lambda_p=1e-2,
    p_norm=1, *args, **kwargs):
    super(pAdam, self).__init__(params, lr=lr, betas=betas, eps=eps, weight_decay=0, *
        args, **kwargs)
    self.p_norm = p_norm
    self.lambda_p = lambda_p


@torch.no_grad()
def step(self, closure=None):
    # Store the old params
    old_params = []
    for group in self.param_groups:
        old_params.append({param: param.data.clone() for param in group['params'] if
            param.grad is not None})

    # Perform the standard AdamW step
    loss = super(pAdam, self).step(closure)

    # Perform the pWD step
    for group, old_group in zip(self.param_groups, old_params):
        lambda_p_group = group.get('lambda_p', self.lambda_p)  # support prams groups
        if lambda_p_group > 0:  # Apply regularization only for lambda_p > 0
            for param in group['params']:
                if param.grad is None:
                    continue

                # Use old parameters in the decay factor
                param_old = old_group[param]
                X = param_old.abs()**(2 - self.p_norm)
                update_term = X / (X + self.p_norm * group['lr'] * lambda_p_group)

                # pWD step
                param.data.mul_(update_term)

    return loss
```

F. Experimental Details

In this section, we provide additional details on the experimental setup and hyperparameters used in our experiments. We include supplementary figures that were omitted from the main text.

In all experiments we used Adam as our base optimizer. We held the Adam hyperparameters constant for all experiments:

- $\beta_1 = 0.9$.
- $\beta_2 = 0.999$.
- $\epsilon = 10^{-8}$.

We used a learning rate schedule comprised of a linear warm-up, up to `max_lr`, followed by a cosine annealing reaching a minimum learning rate of `min_lr=max_lr`/100.

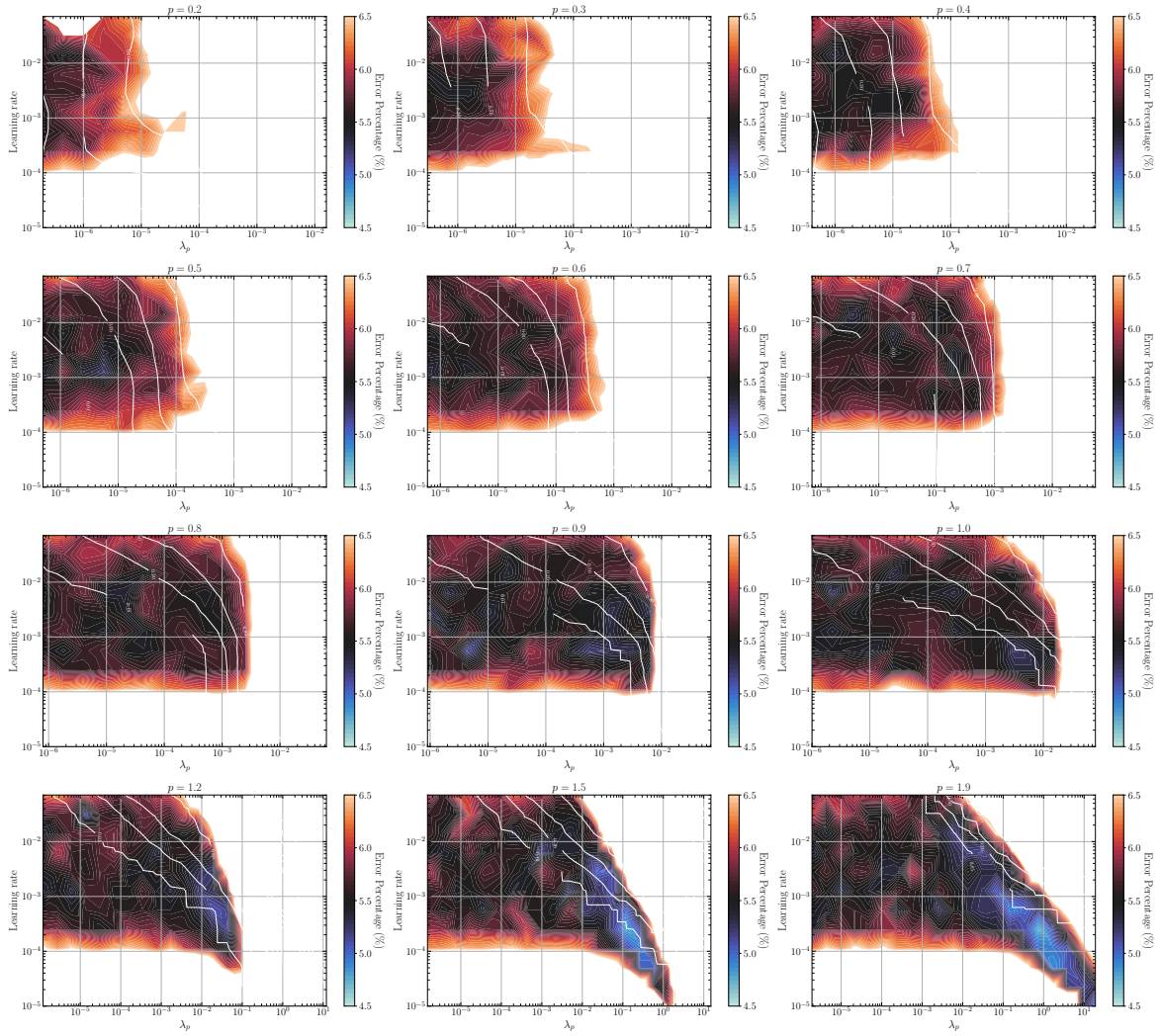

Figure 4: Contours of validation accuracy after 100 training epochs on the λ_p vs. learning rate plane, for ResNet18 on CIFAR-10. White contours represent the [0.01, 0.2, 0.4, 0.8] sparisty levlel.

F.1. ResNet18 on CIFAR-10

We used the standard ResNet18 architecture for our experiments. We trained the network for 100 epochs with a batch size of 64, and 4 workers for data loading. The linear warm-up was set to 3 epochs. We scanned `max_lr` and λ_p for a range of p values. The accuracy contours are shown below in Figure 4.

F.2. nanoGPT on Tiny Shakespeare

We used the nanoGPT architecture for our experiments. We trained the network for 5000 iterations. We used a batch size of 64, block size of 256, 6 attention heads, 6 layers, embedding dimension of size 384, and gradient clipping of 1.0. We scanned `max_lr` and λ_p for a range of p values. The linear warm-up was set to 100 iterations. The accuracy contours are shown below in Figure 5.

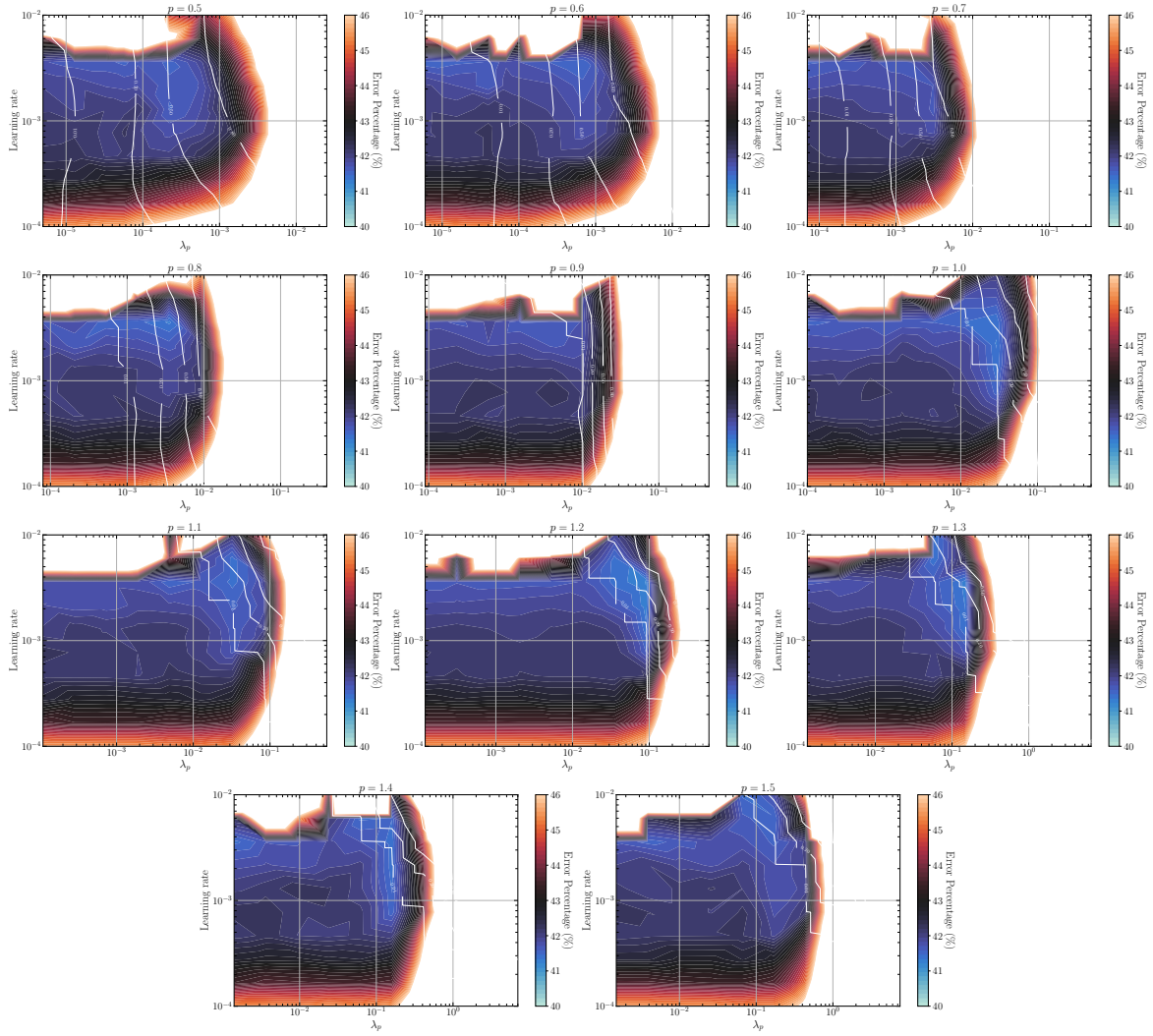

Figure 5: Contours of validation accuracy after 5000 training iterations on the λ_p vs. learning rate plane, for nanoGPT on Tiny Shakespeare. White contours represent the [0.01, 0.2, 0.4, 0.8] sparisty levlel.