Sparse maximal update parameterization: A holistic approach to sparse training dynamics

Nolan Dey **Shane Bergsma** **Joel Hestness**
Cerebras Systems
{nolan,joel}@cerebras.net

Abstract

Several challenges make it difficult for sparse neural networks to compete with dense models. First, setting a large fraction of weights to zero impairs forward and gradient signal propagation. Second, sparse studies often need to test multiple sparsity levels, while also introducing new hyperparameters (HPs), leading to prohibitive tuning costs. Indeed, the standard practice is to re-use the learning HPs originally crafted for dense models. Unfortunately, we show sparse and dense networks do not share the same optimal HPs. Without stable dynamics and effective training recipes, it is costly to test sparsity at scale, which is key to surpassing dense networks and making the business case for sparsity acceleration in hardware.

A holistic approach is needed to tackle these challenges and we propose sparse maximal update parameterization (SμPar) as one such approach. SμPar ensures activations, gradients, and weight updates all scale independently of sparsity level. Further, by reparameterizing the HPs, SμPar enables the same HP values to be optimal as we vary both sparsity level and model width. HPs can be tuned on small dense networks and transferred to large sparse models, greatly reducing tuning costs. On large-scale language modeling, SμPar training improves loss by up to 8.2% over the common approach of using the dense model standard parameterization.

1 Intro

Sparsity has emerged as a key technique to mitigate the increasing computational costs of training and inference in deep neural networks. *Activation sparsity* can cut down feed-forward network computation via techniques like mixture-of-experts [12] and nonlinearities with zero-output regions [38], while further techniques target attention mechanisms to reduce their quadratic complexity [6, 30, 60].

Complementing activation sparsity, this work focuses on *weight sparsity*, whereby a significant fraction of model weights are kept at zero. It has long been known that dense neural networks can be heavily pruned *after* training [32]. With the goal of reducing costs *during* training, recent work has explored static weight sparsity from initialization. In particular, using a random sparsity pattern has re-emerged as a surprisingly effective strategy [35, 62], and we adopt this strategy in this paper.

Unfortunately, several challenges have hindered progress in weight-sparse neural networks. First, sparsity impairs signal propagation during training [33, 11, 1]. Second, with today's techniques, sparse training is costly. Sparse techniques typically introduce extra hyperparameters (HPs), e.g., number of pruning iterations at initialization [64, 7, 59], and it is common to train models across different sparsity levels. Since tuning should be performed at each level and the search space grows exponentially with the number of HPs, the tuning costs essentially "defeat the purpose" of sparsity, i.e., to *reduce* computation [64]. Finally, today there is only a nascent ecosystem of hardware acceleration for unstructured sparsity, so most researchers get little sparsity benefit when tuning.

Preprint. Under review.

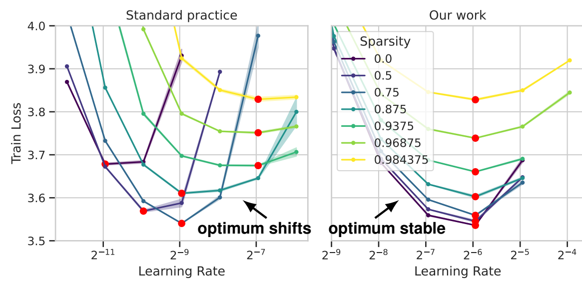

Figure 1: Our work allows stable optimum HPs for any sparsity level, unlike standard practice.

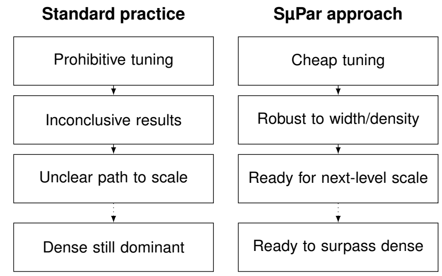

Figure 2: SμPar enables sparse training at scale, helping to surpass dense and motivate sparsity in hardware.

These costs have led to the standard practice of *simply re-using HPs that were previously optimized for the baseline dense models* (Section 2). One might hope that sparse models thrive with the same learning rates and other HPs as their dense counterparts. Unfortunately, they do not: different sparsity levels (including 0% sparsity) have different optimal HPs (Figure 1, left). With impaired training dynamics, prohibitive tuning cost, and lacking the established training recipes enjoyed by dense models, it is unclear how to effectively train sparse networks at scale (Figure 2).

To remedy this situation, we propose sparse maximal update parameterization (SμPar, pronounced "soo-pahr"), a novel, holistic approach to stabilize sparse training dynamics. SμPar fulfills the maximal update desiderata (Section 3) by parameterizing weight initialization and learning rates with respect to change in width *and* sparsity level. Analogous to maximal update parameterization (μP) [68, 67], SμPar enjoys well-controlled activation, gradient, and weight update scales in expectation, avoiding exploding or vanishing signal when changing both sparsity and model width.

By reparameterizing HPs in this way, SμPar enables the same HP values to be optimal as sparsity varies (Figure 1, right). We therefore enjoy μTransfer: we can tune small proxy models and transfer optimal HPs directly to models at scale. In fact, we discovered our μP HPs, tuned for dense models in prior work (and equivalent to SμPar with sparsity=0%), correspond to the optimal learning rate and initial weight variance for *all* sparse models tuned in this paper! As sparsity increases, our formulation shows the standard parameterization (SP) and μP suffer from vanishing signal, further clarifying prior observations of gradient flow issues in sparse networks. The improvements enabled by SμPar set the Pareto-frontier best loss across sparsity levels. Figure 3 previews this improvement for large language models trained from compute-optimal configurations [24]. Here, SμPar benefits grow with increasing sparsity, to 8.2% better than SP and 2.1% better than μP at 99.2% sparsity. These loss improvements correspond to 4.1× and 1.5× compute efficiency gains along the Chinchilla scaling law, respectively.

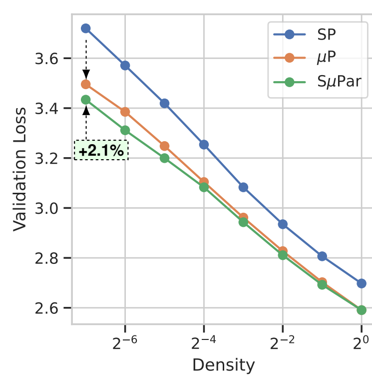

Figure 3: For LLMs, SμPar forms the Pareto frontier loss across sparsity levels, with no HP tuning required.

2 Related work

Sparse training landscape While pruning-after-training has the goal of more-efficient inference [21, 27], sparse training aims to reduce training costs, ultimately unlocking sparse models that are bigger and better than the largest possible dense models [10, 23]. Sparse training can be divided into static sparsity, where the connectivity is fixed (our focus) and dynamic sparsity, where the sparsity mask can evolve [23]. We use *unstructured* sparsity, though our approach generalizes to structured approaches where a particular sparsity pattern increases efficiency on specific hardware [71, 28, 41, 14, 31, 1]. Unstructured connectivity may be based on both random pruning [43, 18, 61, 35, 62] and various pruning-at-initialization criteria [34, 64, 65, 59, 7]. Liu et al. [35] found that as models scale, the relative performance of randomly pruned networks grow. Furthermore, Frantar et al. [15] found the optimal level of sparsity increases with the amount of training data [15]. Together, these findings suggest that as neural networks continue to get wider and deeper, and trained on more and more data, very sparse randomly-pruned networks may emerge as an attractive option.

Improving sparse training dynamics Many prior works identify various training dynamics issues when training sparse models. In particular, prior works note sparsity impacts weight initialization [37, 33, 52, 11], activation variance [31], gradient flow [65, 40, 61, 11, 1], and step sizes during weight updates [15]. These prior works fix these issues in targeted ways, often showing benefits to sparse model training loss. We advocate for a holistic approach, and discuss the relationship between these prior works and our approach in Section 5 after describing and evaluating SμPar.

Sparse sensitivity to HPs Due to the costs of training with fixed weight sparsity, re-using dense HPs is standard practice.[1] However, some prior work has suggested such training is sensitive to HPs, e.g., learning rates [37, 61], or learning rate schedules [16], although systematic tuning was not performed. For dynamic sparse training (DST), it is also conventional to re-use dense HPs, whether in dense-to-sparse [40, 15] or sparse-to-sparse (evolving mask) training [2, 8, 36, 11, 63]. As with fixed sparsity, work here has also suggested sensitivity to HPs, e.g., to dropout and label smoothing [16]. DST may also benefit from extra training steps [10] or smaller batch sizes [36], although in DST this may mainly be due to a greater number of opportunities for connectivity exploration [36].

3 Sparse maximal update parameterization (SμPar)

We now provide background, motivation, and derivation for SμPar, first introducing notation (Section 3.1) and then defining μDesiderata (Section 3.2) with a brief overview of μP (Section 3.3). Finally we show the problem SμPar solves (Section 3.4), and provide an overview of SμPar (Section 3.5).

3.1 Notation

The operations for a single sparse training step are illustrated in Figure 4. The definition and dimensions are: layer index $l \in [0, L]$, batch size B, learning rate η, loss function $\mathcal{L}$, forward pass function $\mathcal{F}$, input dimension d^{l-1}, input activations $\mathbf{X}^l \in \mathbb{R}^{B \times d^{l-1}}$, input activation gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{X}^l} = \nabla_{\mathbf{X}^l} \mathcal{L} = \nabla \mathbf{X}^l \in \mathbb{R}^{B \times d^{l-1}}$, output dimension d^l, output activations $\mathbf{X}^{l+1} \in \mathbb{R}^{B \times d^l}$, output activation gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{X}^{l+1}} = \nabla_{\mathbf{X}^{l+1}} \mathcal{L} = \nabla \mathbf{X}^{l+1} \in \mathbb{R}^{B \times d^l}$, weights $\mathbf{W}^l \in \mathbb{R}^{d^{l-1} \times d^l}$, initialization variance σ_{W^l} for weights $\mathbf{W}^l$, weight update $\Delta \mathbf{W}^l \in \mathbb{R}^{d^{l-1} \times d^l}$, and $\Delta \mathbf{X}^{l+1} \in \mathbb{R}^{B \times d^l}$ is the effect of the weight update on output activations: $\Delta \mathbf{X}^{l+1} = \mathbf{X}^l (\Delta \mathbf{W}^l \odot \mathbf{M}^l)$. Unless otherwise specified, $\mathbf{M}^l \in \{0, 1\}^{d^{l-1} \times d^l}$ is an unstructured random static mask with sparsity s and density $\rho = 1 - s$. When changing model scale or sparsity, we refer to a width multiplier $m_{d^l} = \frac{d^l}{d^l_{\text{base}}}$ and density multiplier $m_\rho = \frac{\rho}{\rho_{\text{base}}}$.

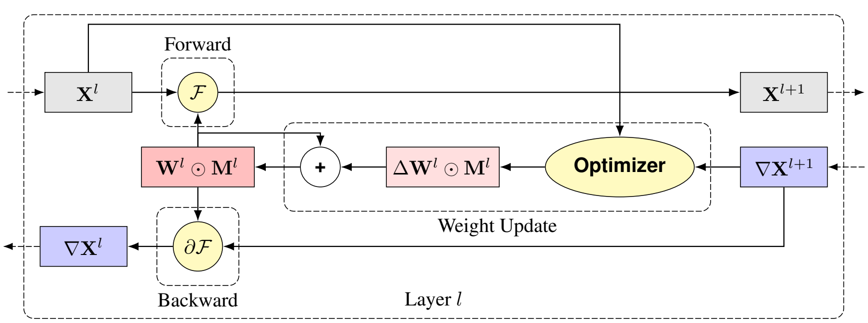

Figure 4: By controlling the scale of the forward pass, backward pass, and weight update operations, across all sparse layers, and all training steps, we achieve stable training dynamics.

[1] Such re-use is typically indicated in paper appendices or supplemental materials, e.g., [43, 34, 37, 33, 16, 64, 65, 59, 13, 7, 18, 61, 35, 62]. Also, dynamic sparsity approaches often compare to fixed sparsity; these baselines are likewise reported to re-use the dense HPs [2, 44, 10, 36, 11, 63].

If we apply sparsity to a linear layer (i.e., $\mathcal{F}$ is a fully-connected layer), our aim is to control:

1. **Forward pass:** $\mathbf{X}^{l+1} = \mathcal{F}(\mathbf{X}^l, \mathbf{W}^l \odot \mathbf{M}^l) = \mathbf{X}^l(\mathbf{W}^l \odot \mathbf{M}^l)$
2. **Backward pass:** $\nabla \mathbf{X}^l = \nabla \mathbf{X}^{l+1}(\mathbf{W}^l \odot \mathbf{M}^l)^\top$.
3. **Weight update:** $\mathbf{X}^{l+1} + \Delta \mathbf{X}^{l+1} = \mathbf{X}^l(\mathbf{W}^l \odot \mathbf{M}^l) + \mathbf{X}^l(\Delta \mathbf{W}^l \odot \mathbf{M}^l)$.

3.2 µDesiderata: Defining the goal of µP and SµPar

Prior works [67, 69] introduce desiderata which define the goal of µP. We define a more general set of desiderata which we refer to as "Generalized-µDesiderata".

Generalized-µDesiderata: $\|\mathbf{X}^l\|_F, \|\nabla \mathbf{X}^l\|_F, \|\Delta \mathbf{X}^l\|_F$ are each invariant to *some variable(s)* we would like to scale, $\forall l$.

Variables to scale include width [68, 67, 69], depth [70], and sparsity (this work). Satisfying µDesiderata represents a more holistic approach to stabilizing training dynamics compared to controlling only a subset of operations in a training step (e.g., only $X^l, \forall l$).

3.3 Maximal update parameterization (µP)

Here we provide a brief overview of maximal update parameterization (µP) [68, 67, 69]. Yang and Hu [68] first show that as model width increases, the scale of activations throughout training also increases. This motivated defining the µP-µDesiderata.

µP-µDesiderata: $\|\mathbf{X}^l\|_F, \|\nabla \mathbf{X}^l\|_F, \|\Delta \mathbf{X}^l\|_F$ are each invariant to change in width $m_{d^l}, \forall l$.

µP was introduced as the unique parameterization that satisfies the µDesiderata with respect to width. They show µP enables µTransfer: the optimum learning rate, initialization weight variance, scalar multipliers, and learning rate schedule all remain consistent as width is increased for µP models. They leverage µTransfer to take a *tune small, train large* approach where hyperparameters are extensively tuned for a small model then transferred, enabling improvements over standard practice. Yang et al. [69] show that the µP-µDesiderata can also be satisfied by controlling the spectral norm of weights.

3.4 Sparsifying models causes vanishing activations and gradients

As Yang et al. [67] show, activation magnitudes explode with increasing model width. In Figure 5 we show sparsity has the opposite effect: increasing sparsity causes shrinking activation magnitudes.

Finding 1*: Sparsity causes vanishing activations and gradients with both SP and µP.*

This finding motivates us to define the SµPar-µDesiderata and develop SµPar to satisfy it.

SµPar-µDesiderata: $\|\mathbf{X}^l\|_F, \|\nabla \mathbf{X}^l\|_F, \|\Delta \mathbf{X}^l\|_F$ are each invariant to change in width m_{d^l} **and** change in density m_ρ, $\forall l$.

3.5 SµPar Overview

SµPar is the unique parameterization which satisfies the SµPar-µDesiderata. In this section, we walk through the changes required to control each of the three operations in a sparse training step, providing an overview of the SµPar derivation. We focus on the AdamW [39] optimizer used in our experiments. For a more detailed derivation, including both SGD and Adam, see Appendix B.

Forward pass at initialization To ensure $\|\mathbf{X}^{l+1}\|_F$ is invariant to changes in width $m_{d^{l-1}}$ and density m_ρ, we can control the mean and variance of $\mathbf{X}_{ij}^{l+1}$. Since at initialization $\mathbb{E}[\mathbf{W}^l] = 0$, $\mathbb{E}[\mathbf{X}^{l+1}] = 0$ and the mean is controlled. The variance of $\mathbf{X}_{ij}^{l+1}$ can be written as:

$$\mathrm{Var}(\mathbf{X}_{ij}^{l+1}) = m_{d^{l-1}} d_{\text{base}}^{l-1} m_\rho \rho_{\text{base}} \sigma_{W^l}^2 (\mathrm{Var}(\mathbf{X}^l) + \mathbb{E}[\mathbf{X}^l]^2) \tag{1}$$

To ensure $\mathrm{Var}(\mathbf{X}_{ij}^{l+1})$ scales independent of $m_{d^{l-1}}$ and m_ρ, we choose $\sigma_{\mathbf{W}^l}^2 = \frac{\sigma_{\mathbf{W}^l, base}^2}{m_{d^{l-1}} m_\rho}$.

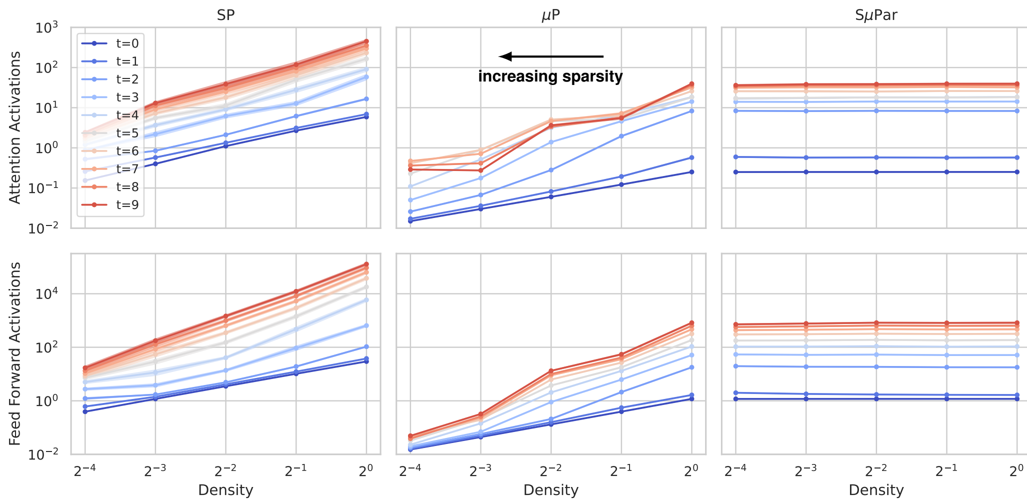

Figure 5: Mean absolute value of output activations for attention and feed forward blocks after training step t. In SP and µP models, decreasing density causes activations to vanish (note axes on log-scale). In SµPar models, density has little effect on activation scales and there is no vanishing.

Backward gradient pass at initialization To ensure $\|\nabla \mathbf{X}^l\|_F$ is invariant to changes in width $m_{d^{l-1}}$ and density m_ρ we can control the mean and variance of $\nabla \mathbf{X}^l$. Since at initialization $\mathbb{E}[\mathbf{W}^l] = 0$, $\mathbb{E}[\nabla \mathbf{X}^l] = 0$ and the mean is controlled. The variance of $\nabla \mathbf{X}^l_{ij}$ can be written as:

$$\mathrm{Var}(\nabla \mathbf{X}^l_{ij}) = m_{d^l} d^l_{\text{base}} m_\rho \rho_{\text{base}} \sigma^2_{\mathbf{W}^l} \mathrm{Var}(\nabla \mathbf{X}^{l+1}) \quad (2)$$

To ensure $\mathrm{Var}(\nabla \mathbf{X}^l_{ij})$ scales independent of m_{d^l} and m_ρ, we choose $\sigma^2_{\mathbf{W}^l} = \frac{\sigma^2_{\mathbf{W}^l,base}}{m_{d^l} m_\rho}$. Typically $m_{d^l} = m_{d^{l-1}}$ across hidden layers, allowing the same $\sigma^2_{\mathbf{W}^l}$ to fix both forward and backward scales.

Effect of Adam weight update We desire $\|\Delta \mathbf{X}^{l+1}\|_F$ to be is invariant to changes in width $m_{d^{l-1}}$ and density m_ρ. By the law of large numbers, the expected size of each element can be written as:

$$\Delta \mathbf{X}^{l+1}_{ij} \to \eta^l m_{d^{l-1}} d^{l-1}_{\text{base}} m_\rho \rho_{\text{base}} \mathbb{E}\left[\mathbf{X}^l_{ik}\left(\frac{\sum_t^T \gamma_t \sum_h^b \mathbf{X}^{l,t}_{hk} \nabla \mathbf{X}^{l+1,t}_{hj}}{\sqrt{\sum_t^T \omega_t \sum_h^b (\mathbf{X}^{l,t}_{hk} \nabla \mathbf{X}^{l+1,t}_{hj})^2}}\right)\right], \text{ as } (d^{l-1}\rho) \to \infty \quad (3)$$

To ensure $\Delta \mathbf{X}^{l+1}_{ij}$ and $\|\Delta \mathbf{X}^{l+1}\|_F$ scale invariant to $m_{d^{l-1}}, m_\rho$, we choose $\eta^l = \frac{\eta^l_{\text{base}}}{m_{d^{l-1}} m_\rho}$.

Implementation Summary Table 1 summarizes the differences between SP, µP, and SµPar. Since we only sparsify hidden weights, SµPar matches µP for input, output, bias, layer-norm, and attention logits. Also note width and density multipliers are usually the same for all layers, allowing simplified notation m_d, m_ρ for width and density multipliers respectively. This correction is equivalent to µP [67] when $\rho = 1$ and $m_\rho = 1$. The correction to hidden weight initialization we derive is similar to the sparsity-aware initialization in prior work [37, 52, 11]. SµPar should also easily extend to 2:4 sparsity pattern because, in expectation, the rows and columns of M^l should have equal density.

4 SµPar Training Results

Here, we present empirical results showing the effectiveness of SµPar over SP and µP when training sparse models. When using SP or µP, optimal HPs drift as we change the sparsity level, possibly leading to inconclusive or even reversed findings. SµPar has stable optimal HPs across both model width and sparsity level, and we show it improves over SP and µP across different scaling approaches.

Table 1: Summary of SP, μP, and SμPar

Parameterization	SP	μP	SμPar
Embedding Var.	σ^2_{base}	σ^2_{base}	σ^2_{base}
Embedding LR	η_{base}	η_{base}	η_{base}
Embedding Fwd.	$\mathbf{X}^0\mathbf{W}_{\text{emb}}$	$\alpha_{\text{input}} \cdot \mathbf{X}^0\mathbf{W}_{\text{emb}}$	$\alpha_{\text{input}} \cdot \mathbf{X}^0\mathbf{W}_{\text{emb}}$
Hidden Var.	σ^2_{base}	$\sigma^2_{\text{base}}/m_d$	$\sigma^2_{\text{base}}/(m_d m_\rho)$
Hidden LR (Adam)	η_{base}	η_{base}/m_d	$\eta_{\text{base}}/(m_d m_\rho)$
Unembedding Fwd.	$\mathbf{X}^L\mathbf{W}^\top_{\text{emb}}$	$\alpha_{\text{output}}\mathbf{X}^L\mathbf{W}^\top_{\text{emb}}/m_d$	$\alpha_{\text{output}}\mathbf{X}^L\mathbf{W}^\top_{\text{emb}}/m_d$
Attention logits	$\mathbf{Q}^\top\mathbf{K}/\sqrt{d_{\text{head}}}$	$\mathbf{Q}^\top\mathbf{K}/d_{\text{head}}$	$\mathbf{Q}^\top\mathbf{K}/d_{\text{head}}$

Taken together, we see that SμPar sets the Pareto frontier best loss across all sparsities and widths, including when we scale to a large dense model with width equal to GPT-3 XL [4]. Optimal *dense* μP HPs—when adjusted using SμPar—are also optimal HPs for all sparse models that we test here.

All tests in this section use GPT-like transformer language models [51, 9], trained on the SlimPajama dataset [57]. We refer the reader to Appendix C for full methodology of all experiments.

4.1 Sparse hyperparameter transfer

We first show sparsifying a dense model using either SP or μP leads to non-smooth drift in optimal HPs as the sparsity level changes. Figure 6 shows validation loss for SP, μP, and SμPar models when trained with varying sparsity levels and sweeping across different peak learning rates. For the SP configuration, as sparsity increases, the optimal learning rate increases in a somewhat unpredictable way. μP experiences similar shift in optimal learning rate, though shifts are even more abrupt. For SμPar, the optimal learning rate is consistently near 2^{-6} across all sparsity levels.

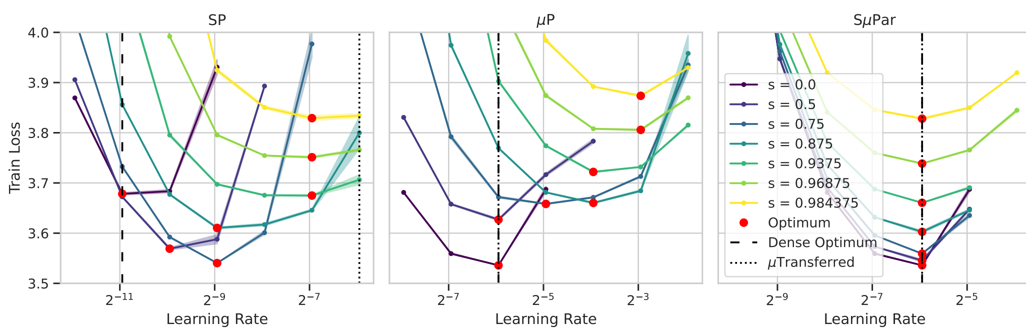

Figure 6: SμPar ensures stable optimal learning rate for any sparsity level, unlike SP and μP.

We also sweep base weight initialization values and find even more chaotic behaviors for SP and μP with different sparsity levels (Figure 7, left and center, respectively)[2]. μP even shows discontinuities in optimal initialization values at different sparsity levels. We attribute this discontinuity to widely varying expected activation scales between embedding and transformer decoder layers, where embedding activation scales will tend to dominate for high sparsity levels. SμPar shows consistent optimal initialization (right plot). Figures 6 and 7 demonstrate our second finding.

***Finding 2**: With SP and μP, dense and sparse networks do not share the same optimal HPs.*

Figure 8 summarizes our HP transfer tests, showing loss for each parameterization across all sparsities. Even when selecting the best learning rate at each sparsity level for SP and μP, SμPar (largely) forms the Pareto frontier with an average gap of 0.8% better than SP and 2.1% better than μP.

[2]These results are taken from a point early in training as models with widely varying initialization tend to become unstable later in training.

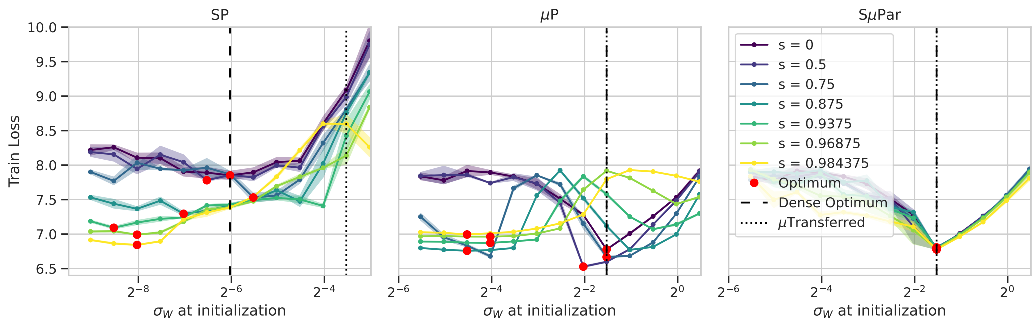

Figure 7: Across sparsity levels, SP and μP show unstable optimal initialization. SμPar is stable.

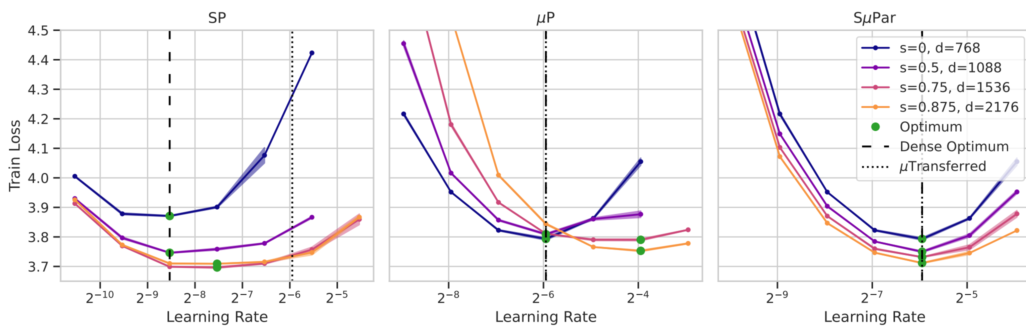

Figure 9: SμPar ensures stable optimal learning rate in Iso-Parameter sparse + wide scaling.

***Finding 3**: SμPar corrects HPs to achieve Pareto frontier loss across sparsity levels.*

4.2 Studying SμPar Indicates How Some Sparse Scaling Techniques Appear to Work

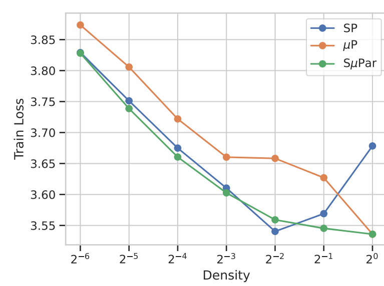

Figure 8: Summarizing loss results from Figure 6 with best tuned HPs for each parameterization and sparsity.

So far, we see SμPar can transfer optimal HPs across sparsity levels, but we have also designed it to transfer HPs across different model widths (hidden sizes), similar to μP. Here, we further demonstrate that SμPar transfers optimal HPs across width. More generally, sparse scaling that keeps a fixed number of non-zero weights per neuron allows SP and μP to also transfer HPs.

Figure 9 shows learning rate transfer tests when changing both the model's hidden size, d_{model}, and sparsity level in a common scaling approach called *Iso-Parameter scaling*. Iso-Parameter scaling keeps the model's number of non-zero parameters approximately the same, as width and sparsity are varied[3]. Here, we see the common result that SP models starting from dense HPs *do* tend to significantly improve as we increase width and sparsity. Note, though, the optimal learning rate for each sparsity level still shifts. When we correct dense HPs using μP or SμPar, the dense baseline significantly improves, but only SμPar shows consistent loss improvement and stable HPs in Iso-Parameter scaling.

Although SP and μP have better stabilized HPs when Iso-Parameter scaling, SμPar still dominates in full training runs. Figure 10 shows losses at the end of training for small models scaled up using Iso-Parameter scaling. Here, all runs use dense optimal HPs, but the SP and μP models experience detuning as sparsity increases.

[3] Not perfectly Iso-Parameter due to unsparsified layers (embedding, bias, layer-norm, etc.)

In the Iso-Parameter setting, SP, μP, and SμPar show similar losses early in training with high sparsity levels and optimal HPs. This consistency is expected based on the SμPar formulation: When the number of non-zero weights per neuron (WPN) in the network is the same, μP and SμPar become synonymous, because initialization and learning rate adjustment factors will be constant (i.e., $d_{\text{model}} \cdot \rho = \text{WPN} = O(1)$). Optimized SP HPs will also tend to work well.

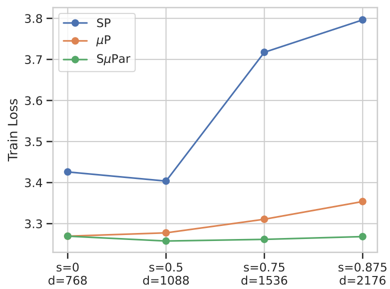

Figure 10: Losses at the end of training when Iso-Parameter scaling, keeping the number of non-zero parameters fixed.

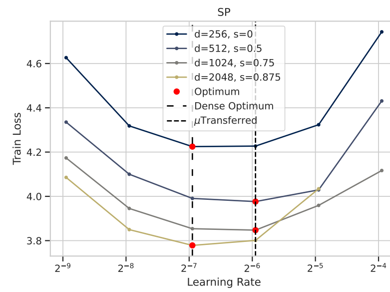

Figure 11: The SP optimized LR is stable when scaling width and sparsity to maintain same number of non-zero weights per neuron (Iso-WPN).

We define this new scaling setting, which we call Iso-WPN, to verify this hypothesis. In Figure 11, we test SP HPs with Iso-WPN scaling and see the optimal learning rate stays consistently between 2^{-7} and 2^{-6} with roughly aligned curves (we omit similar μP and SμPar plots for space, because their corrections are the same). The conclusion is that when scaling SP models in an Iso-WPN sparse setting, HPs should maintain similar training dynamics. More generally, as WPN decreases (e.g., by increasing sparsity), the optimal learning rate will tend to increase proportionally, and vice versa[4].

Reviewing results in Figures 6, 7, 9, and 11, SμPar is the only parameterization that ensures optimal HP transfer across model widths and sparsity levels, satisfying our SμPar μDesiderata.

***Finding 4**: SμPar enables optimal HP transfer for any combination of width and sparsity.*

4.3 SμPar Scaling to Large Language Model Pretraining

We conclude this section reflecting on the demonstration of SμPar improvements in a large-scale language model. We train 610M parameter models starting from a Chinchilla [24] compute-optimal training configuration with 20 tokens per parameter from the SlimPajama dataset. This larger model—with hidden size 2048, 10 layers, and attention head size 64—permits sweeping over a larger range of sparsity levels, so we test up to 99.2% sparsity (density 2^{-7}).

Figure 3 shows validation loss for each parameterization as we sweep sparsity levels. As sparsity increases, SP and μP losses fall farther behind SμPar. Since these models are trained with a large number of tokens, we attribute the widening loss gap mostly to increasingly under-tuned learning rates for SP and μP as sparsity increases–the weight updates lose gradient information throughout training. Retuning SP and μP could recover some of the gap to SμPar, but that could be costly: These runs take 3-6 hours each on a Cerebras CS-3 system (or > 9 days on an NVIDIA A100 GPU).

***Finding 5**: Large networks trained with SμPar improve over SP and μP due to improved tuning.*

5 Discussion and Limitations

SμPar can be a holistic solution As mentioned, prior works make targeted corrections to improve sparse training. These corrections arise from observations that sparsity can cause degraded activation, gradient, and/or weight update signal propagation. We review these observations and corrections in light of the SμPar μDesiderata to advocate for holistic control of sparse training dynamics.

[4]Our results generalize the Yang et al. finding that optimal LR decreases as width increases [67, Figure 1].

Sparsifying Can Cause Vanishing Activations Evci et al. [11] note that by initializing weights using dense methods (e.g., [17, 22]), the "vast majority" of sparse networks have vanishing activations. Lasby et al. [31, App. A] analyze activation variance as a guide for selecting structured sparsity. The µDesiderata suggest activation norms be measured and controlled with respect to sparsity, so activation variance can be considered a proxy to whether sparsity might negatively impact training dynamics. Evci et al. [11] ultimately initialize variances via neuron-specific sparse connectivity, while Liu et al. [37] and Ramanujan et al. [52] propose scaling weight variances proportional to layer sparsity. These corrections, however, only target controlling activations but not weight updates.

Gradient Flow Partially Measures the Weight Update µDesideratum Sparsity also impairs *gradient flow*—the magnitude of the gradient to the weights—during training [11, 1]. Since gradient flow is measured using the norm of the weight gradients, it measures a piece of the weight update. Unfortunately, gradient flow does not directly measure the effect of the weight update step, which can also involve adjustments for things like optimizer state (e.g., momentum and velocity), the learning rate, and weight decay. Prior works propose techniques to improve gradient flow during sparse training and pruning by adjusting individual hyperparameters or adding normalization [65, 40, 11, 1]. However, these techniques might overlook the effects of the optimizer and learning rates in weight updates. Notably, Tessera et al. [61] *do* consider some of these effects, but their proposed techniques maintain gradient flow only in the Iso-Parameter scaling setting rather than arbitrary sparsification.

Frantar et al. [15, App. A.1] also endeavor to control weight updates, where they observe diminished step sizes when optimizing sparse networks with Adafactor [55]. They correct this by computing Adafactor's root-mean-square scaling adjustments over *unpruned* weights and updates. However, such normalization does not prevent activations from scaling with model width [67, 69]. In this sense, sparsity-aware fixes to Adafactor can improve dynamics, but will not address instability holistically.

Weight Initialization Only Controls Dynamics at Initialization We noted works above that adjust sparse weight initializations [11, 37, 52]. Additionally, Lee et al. [33] explore orthogonal weight initialization [49], both before pruning (to ensure SNIP [34] pruning scores are on a similar scale across layers) and after (to improve trainability of the sparse network). While adjusting weights can improve sparse training dynamics at initialization, such adjustments are insufficient to stabilize signals *after multiple steps of training*, in the same way that standard weight initializations fail to stabilize training of dense networks.

Limitations While we have focused on pre-training with static sparsity, it is also common to prune a pre-trained dense model, then fine-tune to recover accuracy. SµPar requires further extension to handle this case, as well as dynamic sparse training. One challenge is that by making pruning (and re-growing) of weights dependent on weight values, the pruned weight distribution significantly differs from the unpruned distribution. Handling such cases is a subject of our ongoing research.

For weight sparsity more generally, the most pressing limitation is the lack of hardware acceleration [41]. While new software [53, 31, 46] continues to better leverage existing hardware, the growth of software and hardware co-design is also encouraging [63, 5], as effective sparsity techniques can be specifically optimized in deep learning hardware. But to effectively plan hardware, we need to train and test sparse prototypes at next-level sizes, at scales where the optimum sparsity level may be higher than in current networks [15]. Performing such *scaling law*-style studies requires incredible resources even for dense models with well-established training recipes [29, 24]. As SµPar reduces training and tuning costs, it can help unlock these studies and guide future hardware design.

For a discussion of the broader impacts of SµPar, see Appendix A.

6 Conclusion

Nobody said training with sparsity was easy. We showed that with the standard parameterization and µP, increasing sparsity level directly correlates with vanishing activations. Impaired training dynamics prevent sparse models from sharing the same optimal hyperparameters, suggesting prior results based on re-use of dense HPs merit re-examination. In contrast, by holistically controlling the training process, SµPar prevents vanishing activations and enables HP transfer (across both width and sparsity). LLMs trained with SµPar improve over µP and the standard parameterization. As such, we hope SµPar makes things a little easier for sparsity research going forward.

Acknowledgements

We would like to thank Gavia Gray, who provided helpful feedback on the manuscript, and Gurpreet Gosal, who tuned the μTransferred hyperparameters seen throughout the document.

References

[1] Abhimanyu Rajeshkumar Bambhaniya, Amir Yazdanbakhsh, Suvinay Subramanian, Sheng-Chun Kao, Shivani Agrawal, Utku Evci, and Tushar Krishna. 2024. Progressive Gradient Flow for Robust N:M Sparsity Training in Transformers. *arXiv preprint arXiv:2402.04744* (2024).

[2] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. 2017. Deep rewiring: Training very sparse deep networks. *arXiv preprint arXiv:1711.05136* (2017).

[3] Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big?. In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*. 610–623.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems* 33 (2020), 1877–1901.

[5] Cerebras Systems. 2024. Train a Model with Weight Sparsity. Cerebras Systems Documentation. https://docs.cerebras.net/en/2.1.1/wsc/how_to_guides/sparsity.html Version 2.1.1.

[6] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509* (2019).

[7] Pau de Jorge, Amartya Sanyal, Harkirat S Behl, Philip HS Torr, Gregory Rogez, and Puneet K Dokania. 2020. Progressive skeletonization: Trimming more fat from a network at initialization. *arXiv preprint arXiv:2006.09081* (2020).

[8] Tim Dettmers and Luke Zettlemoyer. 2019. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840* (2019).

[9] Nolan Dey, Daria Soboleva, Faisal Al-Khateeb, Bowen Yang, Ribhu Pathria, Hemant Khachane, Shaheer Muhammad, Zhiming, Chen, Robert Myers, Jacob Robert Steeves, Natalia Vassilieva, Marvin Tom, and Joel Hestness. 2023. BTLM-3B-8K: 7B Parameter Performance in a 3B Parameter Model. arXiv:2309.11568 [cs.AI]

[10] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. 2020. Rigging the lottery: Making all tickets winners. In *International conference on machine learning*. PMLR, 2943–2952.

[11] Utku Evci, Yani Ioannou, Cem Keskin, and Yann Dauphin. 2022. Gradient flow in sparse neural networks and how lottery tickets win. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 36. 6577–6586.

[12] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research* 23, 120 (2022), 1–39.

[13] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. 2020. Pruning neural networks at initialization: Why are we missing the mark? *arXiv preprint arXiv:2009.08576* (2020).

[14] Elias Frantar and Dan Alistarh. 2023. SparseGPT: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*. 10323–10337.

[15] Elias Frantar, Carlos Riquelme, Neil Houlsby, Dan Alistarh, and Utku Evci. 2023. Scaling laws for sparsely-connected foundation models. *arXiv preprint arXiv:2309.08520* (2023).

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

[17] Xavier Glorot and Yoshua Bengio. 2010. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (PMLR)*.

[18] Anna Golubeva, Behnam Neyshabur, and Guy Gur-Ari. 2020. Are wider nets better given the same number of parameters? *arXiv preprint arXiv:2010.14495* (2020).

[19] Hila Gonen and Yoav Goldberg. 2019. Lipstick on a pig: Debiasing methods cover up systematic gender biases in word embeddings but do not remove them. *arXiv preprint arXiv:1903.03862* (2019).

[20] Yiwen Guo, Chao Zhang, Changshui Zhang, and Yurong Chen. 2018. Sparse dnns with improved adversarial robustness. *Advances in neural information processing systems* 31 (2018).

[21] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE international conference on computer vision*. 1026–1034.

[23] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research* 22, 241 (2021), 1–124.

[24] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack William Rae, and Laurent Sifre. 2022. An Empirical Analysis of Compute-optimal Large Language Model Training. In *The Conference on Neural Information Processing Systems (NeurIPS)*.

[25] Sara Hooker, Aaron Courville, Gregory Clark, Yann Dauphin, and Andrea Frome. 2019. What do compressed deep neural networks forget? *arXiv preprint arXiv:1911.05248* (2019).

[26] Sara Hooker, Nyalleng Moorosi, Gregory Clark, Samy Bengio, and Emily Denton. 2020. Characterising bias in compressed models. *arXiv preprint arXiv:2010.03058* (2020).

[27] Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aäron van den Oord, Sander Dieleman, and Koray Kavukcuoglu. 2018. Efficient neural audio synthesis. In *International Conference on Machine Learning*. PMLR, 2410–2419.

[28] Hyeong-Ju Kang. 2019. Accelerator-aware pruning for convolutional neural networks. *IEEE Transactions on Circuits and Systems for Video Technology* 30, 7 (2019), 2093–2103.

[29] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling Laws for Neural Language Models. arXiv:2001.08361 [cs.LG] https://arxiv.org/abs/2001.08361

[30] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451* (2020).

[31] Mike Lasby, Anna Golubeva, Utku Evci, Mihai Nica, and Yani Ioannou. 2023. Dynamic Sparse Training with Structured Sparsity. *arXiv preprint arXiv:2305.02299* (2023).

[32] Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. *Advances in Neural Information Processing Systems* 2 (1989).

[33] Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip HS Torr. 2019. A signal propagation perspective for pruning neural networks at initialization. *arXiv preprint arXiv:1906.06307* (2019).

[34] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. 2018. SNIP: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340* (2018).

[35] Shiwei Liu, Tianlong Chen, Xiaohan Chen, Li Shen, Decebal Constantin Mocanu, and Mykola Wang, Zhangyang an d Pechenizkiy. 2022. The unreasonable effectiveness of random pruning: Return of the most naive baseline for sparse training. *arXiv preprint arXiv:2202.02643* (2022).

[36] Shiwei Liu, Lu Yin, Decebal Constantin Mocanu, and Mykola Pechenizkiy. 2021. Do we actually need dense over-parameterization? In-time over-parameterization in sparse training. In *International Conference on Machine Learning*. PMLR, 6989–7000.

[37] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270* (2018).

[38] Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, and other s. 2023. Deja Vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*. PMLR, 22137–22176.

[39] Ilya Loshchilov and Frank Hutter. 2017. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*.

[40] Ekdeep Singh Lubana and Robert P Dick. 2020. A gradient flow framework for analyzing network pruning. *arXiv preprint arXiv:2009.11839* (2020).

[41] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. 2021. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378* (2021).

[42] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2019. Model cards for model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*. 220–229.

[43] Decebal Constantin Mocanu, Elena Mocanu, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. 2016. A topological insight into restricted Boltzmann machines. *Machine Learning* 104 (2016), 243–270.

[44] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. 2018. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications* 9, 1 (2018), 2383.

[45] Moin Nadeem, Anna Bethke, and Siva Reddy. 2020. StereoSet: Measuring stereotypical bias in pretrained language models. *arXiv preprint arXiv:2004.09456* (2020).

[46] Neural Magic. 2024. DeepSparse. GitHub repository. https://github.com/neuralmagic/deepsparse

[47] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training Language Models to Follow Instructions With Human Feedback. arXiv:2203.02155 [cs.CL] https://arxiv.org/abs/2203.02155

[48] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. 2021. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350* (2021).

[49] Jeffrey Pennington, Samuel Schoenholz, and Surya Ganguli. 2017. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. *Advances in neural information processing systems* 30 (2017).

[50] Ofir Press, Noah Smith, and Mike Lewis. 2022. Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation. In *International Conference on Learning Representations*.

[51] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf

[52] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. 2020. What's hidden in a randomly weighted neural network?. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 11893–11902.

[53] Erik Schultheis and Rohit Babbar. 2023. Towards Memory-Efficient Training for Extremely Large Output Spaces – Learning with 670k Labels on a Single Commodity GPU. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 689–704.

[54] Noam Shazeer. 2020. GLU Variants Improve Transformer. arXiv:2002.05202 [cs.LG] https://arxiv.org/abs/2002.05202

[55] Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*. PMLR, 4596–4604.

[56] Emily Sheng, Kai-Wei Chang, Premkumar Natarajan, and Nanyun Peng. 2019. The woman worked as a babysitter: On biases in language generation. *arXiv preprint arXiv:1909.01326* (2019).

[57] Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. 2023. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama. https://huggingface.co/datasets/cerebras/SlimPajama-627B

[58] Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. *arXiv preprint arXiv:1906.02243* (2019).

[59] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. 2020. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in neural information processing systems* 33 (2020), 6377–6389.

[60] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2022. Efficient transformers: A survey. *Comput. Surveys* 55, 6 (2022), 1–28.

[61] Kale-ab Tessera, Sara Hooker, and Benjamin Rosman. 2021. Keep the gradients flowing: Using gradient flow to study sparse network optimization. *arXiv preprint arXiv:2102.01670* (2021).

[62] Vithursan Thangarasa, Abhay Gupta, William Marshall, Tianda Li, Kevin Leong, Dennis DeCoste, Sean Lie, and Shreyas Saxena. 2023. SPDF: Sparse pre-training and dense fine-tuning for large language models. In *Uncertainty in Artificial Intelligence*. 2134–2146.

[63] Vithursan Thangarasa, Shreyas Saxena, Abhay Gupta, and Sean Lie. 2023. Sparse-IFT: Sparse Iso-FLOP transformations for maximizing training efficiency. *arXiv preprint arXiv:2303.11525* (2023).

[64] Stijn Verdenius, Maarten Stol, and Patrick Forré. 2020. Pruning via iterative ranking of sensitivity statistics. *arXiv preprint arXiv:2006.00896* (2020).

[65] Chaoqi Wang, Guodong Zhang, and Roger Grosse. 2020. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376* (2020).

[66] Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. 2023. Jailbroken: How does LLM safety training fail? *Advances in Neural Information Processing Systems* 36 (2023).

[67] Greg Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. 2021. Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer. In *Advances in Neural Information Processing Systems*.

[68] Greg Yang and Edward J Hu. 2020. Feature learning in infinite-width neural networks. *arXiv preprint arXiv:2011.14522* (2020).

[69] Greg Yang, James B Simon, and Jeremy Bernstein. 2023. A spectral condition for feature learning. *arXiv preprint arXiv:2310.17813* (2023).

[70] Greg Yang, Dingli Yu, Chen Zhu, and Soufiane Hayou. 2023. Feature Learning in Infinite Depth Neural Networks. In *International Conference on Learning Representations*.

[71] Zhuliang Yao, Shijie Cao, Wencong Xiao, Chen Zhang, and Lanshun Nie. 2019. Balanced sparsity for efficient DNN inference on GPU. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 5676–5683.

A Broader impacts

Sparsity is recognized to reduce carbon emissions [48] and offset well-known environmental and financial costs of large model training [3]. For example, unstructured sparsity can be accelerated by the Cerebras Wafer-Scale Engine[5] and 2:4 block sparsity can be accelerated by NVIDIA Ampere GPUs[6]. There is growing recognition that HP tuning is a key contributor to these costs. HP tuning is costly, possibly undermining equity in AI research due to financial resources [58]. During model *re*training, *sensitivity* to HPs also leads to downstream costs [58]. SμPar can reduce these costs and sensitivities and thus improve equity.

Sparsity also has potential drawbacks. Hooker et al. [25] showed that even when top-line performance metrics are comparable, pruned networks may perform worse on specific subsets of the data (including on underrepresented groups [26]), may amplify sensitivity to adversarial examples, and may be more sensitive to distribution shift. These sensitivities may depend on the degree of sparsity [20]. It remains an open question whether such drawbacks occur only with pruning or when training with sparsity from scratch (as in SμPar) [23], and how such sensitivity may impact susceptibility to misuse [66]. We require sparsity-specific methods to detect [56, 45] and mitigate [19, 47] harm. Moreover, since many large models are later pruned for deployment, we recommend testing and documenting in the model card [42] any adverse affects of sparsification at the time of model release.

B SμPar detailed derivation

B.1 Forward pass at initialization

The first stage where we would like to control training dynamics is in the layer's forward function. For a random unstructured sparsity mask $\mathbf{M}^l$, since each *column* of $\mathbf{M}^l$ has $d^{l-1}\rho$ non-zero elements in expectation, we can rewrite the forward pass as:

$$\mathbf{X}^{l+1}_{ij} = \left[\mathbf{X}^l(\mathbf{W}^l \odot \mathbf{M}^l)\right]_{ij} = \sum_{q=1}^{d^{l-1}} \mathbf{X}^l_{iq}(\mathbf{W}^l_{qj} \cdot \mathbf{M}^l_{qj}) = \sum_{k:\mathbf{M}^l_{kj}=1}^{d^{l-1}\rho} \mathbf{X}^l_{ik}\mathbf{W}^l_{kj} \tag{4}$$

Our goal is to ensure $\|\mathbf{X}^{l+1}\|_F$ is invariant to changes in width $m_{d^{l-1}}$ and density m_ρ. To achieve this we can ensure the mean and variance of $\mathbf{X}^{l+1}_{ij}$ are invariant to $m_{d^{l-1}}$ and m_ρ.

Mean: As expectation is linear and $\mathbf{X}^l$ and $\mathbf{W}^l$ are independent at initialization:

$$\mathbb{E}[\mathbf{X}^{l+1}_{ij}] = \mathbb{E}\left[\sum_{k:\mathbf{M}^l_{kj}=1}^{d^{l-1}\rho} \mathbf{X}^{l+1}_{ik}\mathbf{W}^l_{kj}\right] = \sum_{k:\mathbf{M}^l_{kj}=1}^{d^{l-1}\rho} \mathbb{E}[\mathbf{X}^l_{ik}\mathbf{W}^l_{kj}] = \sum_{k:\mathbf{M}^l_{kj}=1}^{d^{l-1}\rho} \mathbb{E}[\mathbf{X}^l_{ik}]\mathbb{E}[\mathbf{W}^l_{kj}] \tag{5}$$

Therefore, since at initialization $\mathbb{E}[\mathbf{W}^l_{ij}] = 0$, $\mathbb{E}[\mathbf{X}^{l+1}_{ij}] = 0$ and the mean is controlled.

Variance: As expectation is linear and each weight element is IID:

$$\text{Var}(\mathbf{X}^{l+1}_{ij}) = \text{Var}\left(\sum_{k:\mathbf{M}^l_{kj}=1}^{d^{l-1}\rho} \mathbf{X}^l_{ik}\mathbf{W}^l_{kj}\right) = \sum_{k:\mathbf{M}^l_{kj}=1}^{d^{l-1}\rho} \text{Var}(\mathbf{X}^l_{ik}\mathbf{W}^l_{kj}) \tag{6}$$

Then, since $\mathbf{X}^l$ and $\mathbf{W}^l$ are independent at initialization:

$$\text{Var}(\mathbf{X}^{l+1}_{ij}) = \sum_{k:\mathbf{M}^l_{kj}=1}^{d^{l-1}\rho} (\text{Var}(\mathbf{X}^l_{ik}) + \mathbb{E}[\mathbf{X}^l_{ik}]^2)(\text{Var}(\mathbf{W}^l_{kj}) + \mathbb{E}[\mathbf{W}^l_{kj}]^2) - (\mathbb{E}[\mathbf{X}^l_{ik}]\mathbb{E}[\mathbf{W}^l_{kj}])^2 \tag{7}$$

Finally, since at initialization $\mathbb{E}[\mathbf{W}^l_{kj}] = 0$ and redefining $\text{Var}(\mathbf{W}^l_{kj}) = \sigma^2_{\mathbf{W}^l}$:

$$\text{Var}(\mathbf{X}^{l+1}_{ij}) = \sum_{k:\mathbf{M}^l_{kj}=1}^{d^{l-1}\rho} (\text{Var}(\mathbf{X}^l_{ik}) + \mathbb{E}[\mathbf{X}^l_{ik}]^2)\text{Var}(\mathbf{W}^l_{kj}) = d^{l-1}\rho\sigma^2_{\mathbf{W}^l}(\text{Var}(\mathbf{X}^l) + \mathbb{E}[\mathbf{X}^l]^2) \tag{8}$$

[5] https://www.cerebras.net/blog/harnessing-the-power-of-sparsity-for-large-gpt-ai-models
[6] https://www.nvidia.com/en-us/data-center/ampere-architecture/

Rewriting in terms of multipliers for the width $m_{d^{l-1}} = \frac{d^{l-1}}{d^{l-1}_{\text{base}}}$ and the change in density $m_\rho = \frac{\rho}{\rho_{\text{base}}}$:

$$\text{Var}(\mathbf{X}^{l+1}_{ij}) = m_{d^{l-1}} d^{l-1}_{\text{base}} m_\rho \rho_{\text{base}} \sigma^2_{\mathbf{W}^l} (\text{Var}(\mathbf{X}^l) + \mathbb{E}[\mathbf{X}^l]^2) \tag{9}$$

Solution: To ensure $\text{Var}(\mathbf{X}^{l+1}_{ij})$ scales independently of $m_{d^{l-1}}$ and m_ρ, we choose to set $\sigma^2_{\mathbf{W}^l} = \frac{\sigma^2_{\mathbf{W}^l,base}}{m_{d^{l-1}} m_\rho}$. This ensures that $\|\mathbf{X}^{l+1}\|_F$ is invariant to changes in width $m_{d^{l-1}}$ and density m_ρ.

Note that this correction is equivalent to µP [67] when $m_\rho = 1$. Further, the sparsity factor in the denominator matches the correction for sparsity-aware initialization from Evci et al. [11].

B.2 Backward gradient pass at initialization

The next stage we would like to control training dynamics is in the layer's backward pass. For a random unstructured sparsity mask $\mathbf{M}^l$, since each *row* of $\mathbf{M}^l$ has $d^l\rho$ non-zero elements in expectation, we can rewrite the backward pass as:

$$\nabla\mathbf{X}^l_{ij} = \left[\nabla\mathbf{X}^{l+1}(\mathbf{W}^l \odot \mathbf{M}^l)^\top\right]_{ij} = \sum_q^{d^l} \nabla\mathbf{X}^{l+1}_{iq}(\mathbf{W}^l_{jq} \cdot \mathbf{M}^l_{jq}) = \sum_{k:\mathbf{M}^l_{jk}=1}^{d^l\rho} \nabla\mathbf{X}^{l+1}_{ik}\mathbf{W}^l_{jk} \tag{10}$$

Our goal is to ensure $\|\nabla\mathbf{X}^l\|_F$ is invariant to changes in width $m_{d^{l-1}}$ and density m_ρ. To achieve this, we can ensure the mean and variance of $\nabla\mathbf{X}^l$ are invariant to $m_{d^{l-1}}$ and m_ρ.

Mean: As expectation is linear and $\mathbf{X}^l$ and $\mathbf{W}^l$ are (roughly) independent at initialization:

$$\mathbb{E}[\nabla\mathbf{X}^l_{ij}] = \mathbb{E}\left[\sum_{k:\mathbf{M}^l_{jk}=1}^{d^l\rho} \nabla\mathbf{X}^{l+1}_{ik}\mathbf{W}^l_{jk}\right] = \sum_{k:\mathbf{M}^l_{jk}=1}^{d^l\rho} \mathbb{E}[\nabla\mathbf{X}^{l+1}_{ik}\mathbf{W}^l_{jk}] = \sum_{k:\mathbf{M}^l_{jk}=1}^{d^l\rho} \mathbb{E}[\nabla\mathbf{X}^{l+1}_{ik}]\mathbb{E}[\mathbf{W}^l_{jk}] \tag{11}$$

Therefore, since at initialization $\mathbb{E}[\mathbf{W}^l_{jk}] = 0$, $\mathbb{E}[\nabla\mathbf{X}^l_{ij}] = 0$, the mean is controlled.

Variance: As expectation is linear and each weight element is IID:

$$\text{Var}(\nabla\mathbf{X}^l_{ij}) = \text{Var}\left(\sum_{k:\mathbf{M}^l_{jk}=1}^{d^l\rho} \nabla\mathbf{X}^{l+1}_{ik}\mathbf{W}^l_{jk}\right) = \sum_{k:\mathbf{M}^l_{jk}=1}^{d^l\rho} \text{Var}(\nabla\mathbf{X}^{l+1}_{ik}\mathbf{W}^l_{jk}) \tag{12}$$

From the backward pass mean derivation, we know $\mathbb{E}[\nabla\mathbf{X}^{l+1}_{ij}] = 0$. Then, similar to the forward pass variance derivation, we can simplify using the facts that at initialization, $\nabla\mathbf{X}^{l+1}$ and $\mathbf{W}^l$ are (roughly) independent and $\mathbb{E}[\mathbf{W}^l] = 0$. Similarly we can also define $\text{Var}(\mathbf{W}^l_{kj}) = \sigma^2_{\mathbf{W}^l}$ and rewrite in terms of width multiplier $m_{d^l} = \frac{d^l}{d^l_{\text{base}}}$ and changes in density $m_\rho = \frac{\rho}{\rho_{\text{base}}}$:

$$\text{Var}(\nabla\mathbf{X}^l_{ij}) = m_{d^l} d^l_{\text{base}} m_\rho \rho_{\text{base}} \sigma^2_{\mathbf{W}^l} \text{Var}(\nabla\mathbf{X}^{l+1}) \tag{13}$$

Solution: To ensure $\text{Var}(\nabla\mathbf{X}^l_{ij})$ scales independently of m_{d^l} and m_ρ, we choose to set $\sigma^2_{\mathbf{W}^l} = \frac{\sigma^2_{\mathbf{W}^l,base}}{m_{d^l} m_\rho}$. This ensures that $\|\nabla\mathbf{X}^l_{ij}\|_F$ is invariant to changes in width m_{d^l} and density m_ρ. Typically, we scale model width such that $d^l = d^{l-1}$, or these dimensions are scaled proportionally. This proportional scaling allows the same initialization variance to correct both forward activation and backward gradient scales, making them independent of width. Further, since we assume random sparsity across layer's weights, the sparsity initialization correction factor, m_ρ, is the same for both the forward activations and backward gradients.

B.3 Effect of Adam weight update

We desire that the Frobenius norm of the effect of the Adam weight update, $\|\Delta\mathbf{X}^{l+1}\|_F$, is invariant to changes in width $m_{d^{l-1}}$ and density m_ρ. To achieve this we examine the expected size of each element.Here, we use η to be the learning rate for layer l. For a random unstructured sparsity mask $\mathbf{M}^l$, since each *column* of $\mathbf{M}^l$ has $d^{l-1}\rho$ non-zero elements in expectation:

$$\Delta\mathbf{X}_{ij}^{l+1} = \left[\eta\mathbf{X}^l(\Delta\mathbf{W}^l \odot \mathbf{M}^l)\right]_{ij} = \eta\sum_{q=1}^{d^{l-1}}\mathbf{X}_{iq}^l(\Delta\mathbf{W}_{qj}^l \cdot \mathbf{M}_{qj}^l) = \eta\sum_{k:\mathbf{M}_{kj}^l=1}^{d^{l-1}\rho}\mathbf{X}_{ik}^l\Delta\mathbf{W}_{kj}^l \tag{14}$$

Following the formulation in Yang et al. [67], Adam weight updates take the form:

$$\Delta\mathbf{W}_{kj}^l = \frac{\sum_t^T \gamma_t \sum_b^B \mathbf{X}_{bk}^{l,t}\nabla\mathbf{X}_{bj}^{l+1,t}}{\sqrt{\sum_t^T \omega_t \sum_b^B (\mathbf{X}_{bk}^{l,t}\nabla\mathbf{X}_{bj}^{l+1,t})^2}} \tag{15}$$

where T is the current training step and γ_t, ω_t are the moving average weights at each training step. Here, we can just consider the weight update associated with an unpruned weight, since a pruned weight will have value and update 0 (i.e., pruned weights trivially satisfy that their effect on forward activations cannot depend on width or sparsity). We can expand $\Delta\mathbf{X}_{ij}^{l+1}$ as:

$$\Delta\mathbf{X}_{ij}^{l+1} = \eta\sum_{k:\mathbf{M}_{kj}^l=1}^{d^{l-1}\rho}\mathbf{X}_{ik}^l\left(\frac{\sum_t^T \gamma_t \sum_b^B \mathbf{X}_{bk}^{l,t}\nabla\mathbf{X}_{bj}^{l+1,t}}{\sqrt{\sum_t^T \omega_t \sum_b^B (\mathbf{X}_{bk}^{l,t}\nabla\mathbf{X}_{bj}^{l+1,t})^2}}\right) \tag{16}$$

By the Law of Large Numbers:

$$\Delta\mathbf{X}_{ij}^{l+1} \to \eta d^{l-1}\rho\mathbb{E}\left[\mathbf{X}_{ik}^l\left(\frac{\sum_t^T \gamma_t \sum_h^b \mathbf{X}_{hk}^{l,t}\nabla\mathbf{X}_{hj}^{l+1,t}}{\sqrt{\sum_t^T \omega_t \sum_h^b (\mathbf{X}_{hk}^{l,t}\nabla\mathbf{X}_{hj}^{l+1,t})^2}}\right)\right],\ \text{as } (d^{l-1}\rho) \to \infty \tag{17}$$

Rewriting in terms of width multiplier $m_{d^{l-1}} = \frac{d^{l-1}}{d_{\text{base}}^{l-1}}$ and changes in density $m_\rho = \frac{\rho}{\rho_{\text{base}}}$.

$$\Delta\mathbf{X}_{ij}^{l+1} \to \eta m_{d^{l-1}} d_{\text{base}}^{l-1} m_\rho \rho_{\text{base}}\mathbb{E}\left[\mathbf{X}_{ik}^l\left(\frac{\sum_t^T \gamma_t \sum_h^b \mathbf{X}_{hk}^{l,t}\nabla\mathbf{X}_{hj}^{l+1,t}}{\sqrt{\sum_t^T \omega_t \sum_h^b (\mathbf{X}_{hk}^{l,t}\nabla\mathbf{X}_{hj}^{l+1,t})^2}}\right)\right],\ \text{as } (d^{l-1}\rho) \to \infty \tag{18}$$

Solution: To ensure $\Delta\mathbf{X}_{ij}^{l+1}$ and $\|\Delta\mathbf{X}^{l+1}\|_F$ scale invariant to $m_{d^{l-1}}, m_\rho$, we choose $\eta = \frac{\eta_{\text{base}}}{m_{d^{l-1}} m_\rho}$. Note that this correction is equivalent to μP [67] when $\rho = 1, m_\rho = 1$.

B.4 SGD weight update

Similar to the Adam weight update analysis above, we also analyze a weight update with stochastic gradient descent (SGD). We desire that the Frobenius norm of the effect of the SGD weight update, $\|\Delta\mathbf{X}^{l+1}\|_F$, is invariant to changes in width $m_{d^{l-1}}$ and density m_ρ. To achieve this we examine the expected size of each element. For a random unstructured sparsity mask $\mathbf{M}^l$, since each column of $\mathbf{M}^l$ has $d^{l-1}\rho$ non-zero elements in expectation:

$$\Delta\mathbf{X}_{ij}^{l+1} = \left[\eta\mathbf{X}^l(\Delta\mathbf{W}^l \odot \mathbf{M}^l)\right]_{ij} = \eta\sum_{k=1}^{d^{l-1}}\mathbf{X}_{ik}^l(\Delta\mathbf{W}_{kj}^l \cdot \mathbf{M}_{kj}^l) = \eta\sum_{k:\mathbf{M}_{kj}^l=1}^{d^{l-1}\rho}\mathbf{X}_{ik}^l\Delta\mathbf{W}_{kj}^l \tag{19}$$

Following the formulation in Yang et al. [67], SGD weight updates take the form:

$$\Delta\mathbf{W}_{kj}^l = \left[\frac{(\mathbf{X}^l)^\top\nabla\mathbf{X}^{l+1}}{d^{l-1}}\right]_{kj} = \frac{1}{d^{l-1}}\sum_{b=1}^{B}\mathbf{X}_{bk}^l\nabla\mathbf{X}_{bj}^{l+1} \tag{20}$$

We can expand $\Delta\mathbf{X}_{ij}^{l+1}$ as:

$$\Delta\mathbf{X}_{ij}^{l+1} = \frac{\eta}{d^{l-1}}\sum_{k:\mathbf{M}_{kj}^l=1}^{d^{l-1}\rho}\mathbf{X}_{ik}^l(\sum_{b=1}^{B}\mathbf{X}_{bk}^l\nabla\mathbf{X}_{bj}^{l+1}) \tag{21}$$

By the Law of Large Numbers:

$$\Delta \mathbf{X}_{ij}^{l+1} \to \frac{\eta d^{l-1} \rho}{d^{l-1}} \mathbb{E}[\mathbf{X}_{ik}^{l} (\sum_{b}^{B} \mathbf{X}_{bk}^{l} \nabla \mathbf{X}_{bj}^{l+1})], \text{ as } (d^{l-1}\rho) \to \infty \tag{22}$$

Rewriting in terms of change in density $m_\rho = \frac{\rho}{\rho_{\text{base}}}$.

$$\Delta \mathbf{X}_{ij}^{l+1} \to \eta m_\rho \rho_{\text{base}} \mathbb{E}[\mathbf{X}_{ik}^{l} (\sum_{b}^{B} \mathbf{X}_{bk}^{l} \nabla \mathbf{X}_{bj}^{l+1})], \text{ as } (d^{l-1}\rho) \to \infty \tag{23}$$

Solution: To ensure $\Delta \mathbf{X}_{ij}^{l+1}$ and $\|\Delta \mathbf{X}^{l+1}\|_F$ scale independently of $m_{d^{l-1}}$ and m_ρ, we choose $\eta = \frac{\eta_{\text{base}}}{m_\rho}$. Note that this correction is equivalent to µP [67] when $\rho = 1, m_\rho = 1$.

B.5 Additional notes about derivation

We make a few supplementary notes about the above derivation:

- Throughout our derivation, we use ρ to refer to the density level. Note that since this derivation is local to a single layer in the model, the density (or sparsity) level can also be parameterized independently for each layer. If a sparsity technique will use layer-wise independent sparsity levels, appropriate corrections should be made for each layer.
- Similar to the ρ notation, we use η to denote the learning rate, but this learning rate can be layer-specific depending on sparsity level. Appropriate corrections must be made if using layer-wise independent sparsities.
- The use of the Law of Large Numbers in portions of the above derivation indicate that SµPar is expected to provide stable training dynamics as the number of non-zero weights per neuron (WPN) tends to infinity. However, in sparse settings, the WPN can tend to be small. If WPN is small, training dynamics may be affected, and this might be a direction for future work.
- In this work, we only consider sparsifying linear projection layers. As a result, SµPar matches µP for other layers like input, output, bias, layer-norm, and attention logits. Depending on the sparsification technique, these other layers might need to be reviewed for their effects on training dynamics.

C Experimental details

In Table 2, we provide extensive details on hyperparameters, model size, and training schedule for all experiments in this paper. All models in this paper were trained on the SlimPajama dataset [57], a cleaned and deduplicated version of the RedPajama dataset.

SµPar Base Hyperparameter Tuning To find the optimized set of hyperparameters for SµPar, we actually tune µP HPs on a dense proxy model. By formulation of SµPar, these HPs transfer optimally to all the sparse models trained for this work. This dense proxy model is a GPT-2 model, but with small changes: ALiBi position embeddings [50] and SwiGLU nonlinearity [54]. We configure it with width: $d_{\text{model}} = d_{\text{model,base}} = 256$, number of layers: $n_{\text{layers}} = 24$, and head size: $d_{\text{head}} = 64$, resulting in 39M parameters. We trained this proxy model on 800M tokens with a batch size of 256 sequences and sequence length 2048 tokens. We randomly sampled 350 configurations of base learning rates, base initialization standard deviation, and embedding and output logits scaling factors. From this sweep we obtained the tuned hyperparameters listed in Table 3.

Table 2: Experimental details for all figures in this paper.

Figure	d_{model}	L	d_{head}	B	LR	Init. Stdev.	α_{input}	α_{output}	LR decay	LR warm-up steps	Steps	Tokens
Fig. 1 6, 8	4096	2	64	128	Variable	SP: 2.166E-2 µP, SµPar: 0.087	9.1705	1.095	10x linear	116	1169	306M
Fig. 3	2048	10	64	504	SP: 2e-4 µP, SµPar: 1.62E-2	SP: 0.02 µP, SµPar: 0.087	9.1705	1.095	10x linear	1175	11752	12.13B
Fig. 5	2048	2	32	4	1.68E-02	0.101	11.22	1	Constant	0	10	82K
Fig. 7	4096	2	64	8	SP: 1.011E-3 µP, SµPar: 1.62E-2	Variable	9.1705	1.095	Constant	10	100	1.6M
Fig. 9 Fig. 9	Variable	2	64	128	Variable	SP: 0.02 µP, SµPar: 0.087	9.1705	1.095	10x linear	116	1169	306M
Fig. 10	Variable	10	64	256	SP, $d_{model} \leq 1088$: 6e-4 SP, $d_{model} > 1088$: 2e-4 µP, SµPar: 1.62E-2	SP: 0.02 µP, SµPar: 0.087	9.1705	1.095	10x linear	190	1907	1B
Fig. 11	Variable	2	64	128	Variable	0.087 for SP.	N/A	N/A	10x linear	116	1169	306M

Table 3: Tuned hyperparameters for our dense proxy model.

Hyperparameter	Value
$\sigma^2_{W,\text{base}}$	0.08665602
η_{base}	1.62E-2
α_{input}	9.1705
α_{output}	1.0951835