Sparse Training from Random Initialization: Aligning Lottery Ticket Masks using Weight Symmetry

Mohammed Adnan[*1,3], Rohan Jain[*1], Ekansh Sharma[2,3], Yani Ioannou[1]
[1]University of Calgary [2]University of Toronto [3]Vector Institute for AI
{rohan.jain1,adnan.ahmad,yani.ioannou}@ucalgary.ca, ekansh@cs.toronto.edu

The Lottery Ticket Hypothesis (LTH) suggests that there exists a sparse *winning ticket* mask and weights that achieves the same generalization performance as the dense model while using significantly fewer parameters. LTH achieves this by iteratively sparsifying and re-training within the pruned solution basin. However, this procedure is computationally expensive, and a winning ticket's sparsity mask does not generalize to other weight initializations. Recent work has suggested that Deep Neural Networks (DNNs) trained from random initialization find solutions within the same basin modulo weight symmetry, and proposed a method to align trained models within the same basins. We propose permuting the winning ticket mask to align with the new optimization basin when performing sparse training from a different random initialization. Using this permuted mask, we show it is possible to significantly increase the generalization performance of sparse training from random initialization with the permuted mask as compared to sparse training naively using the non-permuted mask. We empirically demonstrate that our proposed method improves the generalization of LTH with the new random initialization on multiple datasets (CIFAR-10/100 and ImageNet) using VGG11 and ResNet-20/50 models of varying widths.

1. Introduction

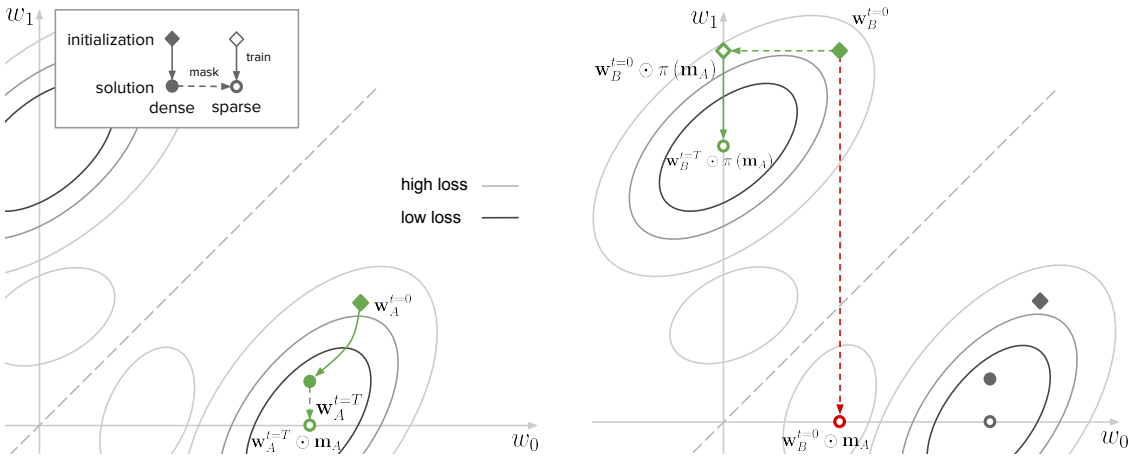
In recent years, foundation models have achieved state-of-the-art results for different tasks. However, the exponential increase in the size of state-of-the-art models requires a similarly exponential increase in the memory and computational costs required to train, store and use these models — decreasing the accessibility of these models for researchers and practitioners alike. To overcome this issue, different model compression methods, such as pruning, quantization and knowledge distillation, have been proposed to reduce the model size at different phases of training or inference. Post-training model pruning [1] has been shown to be effective in compressing the model size, and seminal works have demonstrated that large models can be pruned after training with minimal loss in accuracy [2, 3]. While model pruning makes inference more efficient, it does not reduce the computational cost of training the model.

Motivated by training a sparse model from a random initialization, Frankle and Carbin [4] demonstrated that training with a highly sparse mask is possible and proposed the Lottery Ticket Hypothesis (LTH) to identify sparse networks that, when trained, can match the performance of a dense model. The key caveat is that a dense model must first be trained to find the sparse mask, which can *only* be used with the same random initialization that was used to train the dense model. Despite LTH seeing significant traction in the research community, lottery tickets cannot be applied to a new random initialization, which renders LTH impractical for training sparse models. This posits an important question — *how can we use winning tickets from LTH with a different random initialization?* If the winning ticket can be used again with a different random initialization, then it might make LTH a more practical algorithm for training multiple sparse models (ensembles), offsetting the cost of training a dense model once, and would provide a deeper understanding of how sparse training and LTH work.

In this work, we try to understand why LTH does not work for different random initializations from a weight-space symmetry perspective and show that once permutation symmetries are accounted for, it

[*]Equal contribution.

Submitted to Second Conference on Parsimony and Learning Spotlight Track (CPAL 2025)



(a) Dense training and pruning model A. (b) Sparse training model B with A mask.

Figure 1: **Weight Symmetry and the Sparse Training Problem**. A model with a single layer and only two parameters, $\mathbf{w} = (w_0, w_1)$, operating on a single input scale x_0 has weight symmetry in the 2D loss landscape as illustrated above. In (a) the original dense model, $\mathbf{w}_A$, is trained from a random dense initialization, $\mathbf{w}_A^{t=0}$ to a dense solution, $\mathbf{w}_A^{t=T}$, which is then pruned using weight magnitude resulting in the mask $\mathbf{m}_A = (1,0)$. In (b), naively using the same mask to train a model, B, from a different random initialization will likely result in the initialization being far from a good solution. Permuting the mask to match the (symmetric) basin in which the new initialization is in will enable sparse training. See Fig. 12 in Appendix C for the full figure including LTH.

is possible to reuse the winning ticket mask with different random initializations. Our hypothesis is that to reuse the LTH winning ticket mask with a different random initialization, the winning ticket mask obtained needs to be permuted such that it aligns with the optimization basin associated with the new random initialization. We illustrate our hypothesis in Fig. 1. To empirically validate our hypothesis, we obtain a sparse mask using Iterative Magnitude Pruning (IMP) [3,5] on model A (from Fig. 1) and show that given a permutation that aligns the optimization basin of model A and a new random initialization, the mask can be reused. The sparse model (with the permuted mask) can be trained to closer match the generalization performance of the LTH solution, and the permuted mask improves the generalization of the trained sparse model compared to the non-permuted mask. Our contributions are as follows:

1. Due to the permutation symmetry in weight space, there are many functionally equivalent loss basins and the new random initialization lands in a different loss basin equivalent up to symmetry as shown in Fig. 1b [6]. We show that LTH fails to generalize well with a new random initialization due to a mismatch between the optimization basin of the winning ticket mask and the new random initialization's solution basin.
2. We propose a method based on permutation matching between two dense models, that permutes the winning ticket's sparse mask to align with the optimization basin of the new random initialization.
3. We empirically demonstrate on CIFAR-10/100 and ImageNet datasets using VGG11 and ResNet models of varying widths that permuting the LTH sparse mask to align with the new random initialization improves the performance of the trained model (permuted), compared to the model trained without permuting the sparse mask (naive).

2. Background & Related Work

Linear Mode Connectivity. A pair of trained neural networks are said to be linearly connected if the loss along the linear path between the networks remains small. The phenomenon of linear (mode) connectivity was first observed in the context of Stochastic Gradient Descent (SGD) by Nagarajan and Kolter [7], where they showed that two neural networks trained from the same initialization but differ-

ent data orders exhibit linear connectivity. The term *linear mode connectivity* was introduced by Frankle et al. [8], where they showed that independently trained neural networks can be linearly connected.

Lottery Ticket Hypothesis. The LTH proposes to solve the sparse training problem by re-using the same initialization as used to train the pruned models. For very small models, training from such an initialization maintains the generalization performance of the pruned model and demonstrates that training with a highly sparse mask is possible [4]. However, subsequent work has shown that obtaining winning tickets for modestly-sized models requires using *weight rewinding* [8] — requiring significantly more compute than dense training alone, especially considering that LTH also requires IMP, i.e. training of iteratively sparsified models. We include a detailed description of IMP in Appendix A.3. Paul et al. [9] analyzed the IMP algorithm and showed that sparse network obtained after K^{th} IMP iteration is linearly connected to the sparse model obtained after $K+1^{\text{th}}$ IMP iteration. Furthermore, recent work has shown that the LTH effectively re-learns the original pruned solution it is derived from [10]. To make any practical use of sparse training, finding methods of sparse training from random initialization is necessary to realize any efficiency gains in training.

Weight Symmetry. The process of training Deep Neural Networks (DNNs) requires optimizing over a non-convex loss landscape consisting of numerous local minima, narrow ravines, plateaus, saddle points and loss basins [11–16]. Despite non-convex optimization problems being NP-hard [17], the nature of first-order stochastic optimizers such as SGD [18] have been proven to be highly effective in optimizing DNNs in practice [19, 20]. Empirical evidence suggests that when training independent DNN using SGD, with different batch orders and initializations, the resulting training trajectories often exhibit remarkable similarities [6, 21]. This phenomenon has been attributed to overparameterization, which creates numerous minima in the loss landscape, leading to multiple distinct functions that fit the data similarly [22, 23]. However, as early as the 1990s, Hecht-Nielsen [24] demonstrated that neural networks are *permutation invariant* possessing a weight-symmetrical property, where swapping any two neurons within a hidden layer does not alter the underlying function being learned. In other words, the permuted network remains functionally equivalent to its original configuration. Hence, the existence of permutation symmetries in the loss landscape contributes to its non-convexity, as it creates copies of global minima at different points in weight space due to weight symmetry [25, 26].

Linear Mode Connectivity *modulo* Permutation. Entezari et al. [25] further observed that while a model and its randomly permuted counterpart are functionally equivalent, they are rarely *linearly connected* in the weight space. This misalignment suggests the presence of *loss barriers*—regions along a linear path between models where the loss is significantly higher than at the endpoints. They conjectured that independently obtained SGD solutions exhibit no loss barrier when accounting for permutation symmetries, suggesting that all SGD-trained networks converge to a single basin modulo permutations. Building on this conjecture, several algorithms have been developed to address permutation invariance by aligning trained networks to the same optimization basin [6, 27–29]. Ainsworth et al. [6] demonstrated that two models trained from different random initializations find solutions within the same basin modulo permutation symmetry. They proposed a permutation matching algorithm to permute the units of one model to align it with a reference model, enabling Linear-Mode Connectivity (LMC) [8]. The use of activation matching for model alignment was originally introduced by Li et al. [30] to ensure models learn similar representations when performing the same task. Jordan et al. [27] investigated the poor performance of interpolated networks, attributing it to a phenomenon they termed "variance collapse". To address this, they proposed a method that rescales the hidden units, leading to significant improvements in the performance of interpolated networks. A rigorous study from Sharma et al. [31] introduced a notion of *simultaneous weak linear connectivity* where a permutation, π, aligning two networks also simultaneously aligns two larger fully trained networks throughout the entire SGD trajectory and the same π also aligns successive iterations of independently sparsified networks found via weight rewinding. Sharma et al. [31] also showed that for certain neural networks, sparse mask obtained via weight rewinding can be reused modulo permutations without hurting the test performance.

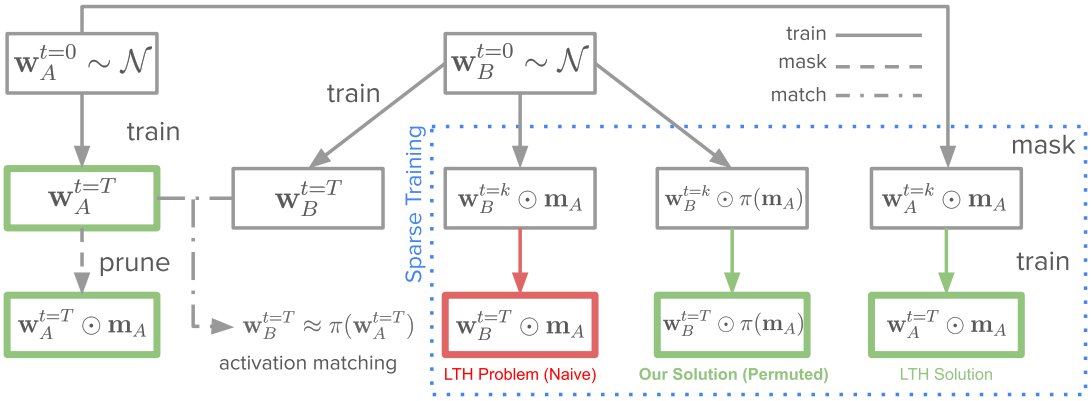


Figure 2: The overall framework of the training procedure, beginning with two distinct dense random weight initializations, $\mathbf{w}_A^{t=0}$, $\mathbf{w}_B^{t=0}$ sampled from a normal distribution, $\mathcal{N}$. The sparse training problem attempts to train the random initialization, $\mathbf{w}_B^{t=0}$ using the naive mask $\mathbf{m}_A$, found by pruning a dense trained model, $\mathbf{w}_A^{t=T}$. However, this results in poor generalization performance [8]. We propose to instead train $\mathbf{w}_B^{t=k}$ at some rewound epoch k, equipped with a *permuted* mask $\pi(\mathbf{m}_A)$. We show that this achieves more comparable generalization to the pruned model/trained LTH solution, $\mathbf{w}_A^{t=T} \odot \mathbf{m}_A$.

3. Method

Motivation. In this work, we try to understand *why LTH masks fail to transfer to a new random initialization*. Our hypothesis is that the loss basin corresponding to the LTH mask is not aligned with the new random initialization, as shown in Fig. 1. Since the sparse mask is not in alignment with the basin of the new random initialization, sparse training does not work well; therefore, aligning the LTH mask with new random initialization may improve sparse training and enable the transfer of LTH masks to new random initializations.

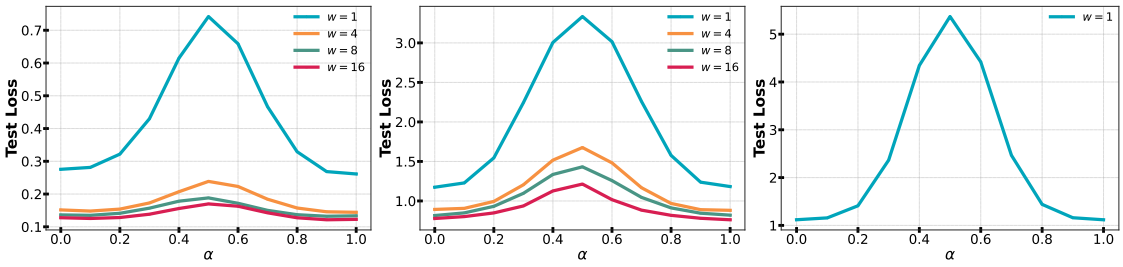
Permutation Matching. Ainsworth et al. [6] showed the permutation symmetries in the weight space can be leveraged to align the basin of two models trained from different random initializations. The permutation mapping can be obtained by either matching activations or weights. In this work, we use activation matching to obtain the permutation mapping as it has been shown to be more stable in recent works [31]. Activation matching tries to find a permutation mapping, $\pi \in S_d$ (where S_d is the permutation group of order $d!$) such that by permuting the parameters of the second model, the correlation between the activations of the two models is maximized. For a model consisting of L layers, each layer is sequentially matched and permuted starting from the input layer. Let $Z_l^A, Z_l^B \in \mathbb{R}^{d \times n}$ be the activations of layer l of model A and B respectively obtained using the training data, where d represents the dimensionality of the activations at layer l and n is the number of training data points. Then a permutation mapping for layer l, π_l, is obtained by solving:

$$\pi_l = \underset{\pi}{\operatorname{argmin}} ||Z_l^B - \pi Z_l^A|| = \underset{\pi}{\operatorname{argmax}} \langle \pi, Z^B (Z^A)^\top \rangle_F, \tag{1}$$

where $\langle .,. \rangle_F$ denotes the Frobenius inner product. Eq. (1) can be formulated as a linear assignment problem (LAP) [32, 33] solved via the Hungarian algorithm [34]; however, the permutation found is not global optima but a greedy/approximate solution as permutation matching is a NP-hard problem. Once the permutation mapping is obtained for all the layers, the model A can be permuted to match model B. To ensure that the permuted model does not change functionally when permuting the output dimension of layer l, the input dimension of the next layer is also permuted accordingly. Let W_l and b_l be the weights and bias of layer l respectively, then the permuted weight matrix W_l^p and permuted bias b_l^p for each layer can be mathematically represented as,

$$W_l^p = \pi_l W_l (\pi_{l-1})^\top, \qquad b_l^p = \pi_l b_l. \tag{2}$$

Evaluating Permutation Matching. Since LAP uses a greedy search to find an approximate solution, to ensure that the permuted model A and model B lie in the same basin, we evaluate the LMC (loss



(a) ResNet20/CIFAR-10 (b) ResNet20/CIFAR-100 (c) ResNet50/ImageNet

Figure 3: **Larger width exhibits better LMC.** Plots showing linear interpolation between $\pi(\mathbf{w}_A^{t=T})$ and $\mathbf{w}_B^{t=T}$ where π was obtained through activation matching between two dense models for varying widths. As the width of the model increases, the permutation matching algorithm gets more accurate, thereby reducing the loss barrier (i.e., better LMC), which is evaluated on the test set. This shows that the permutation matching can find a better mapping, π, for wider models, explaining why the permuted mask works better in case of wider models.

barrier) between the two models. More formally, let θ_1,θ_2 be the parameters of two networks, then the loss barrier $\mathcal{B}$ is defined as:

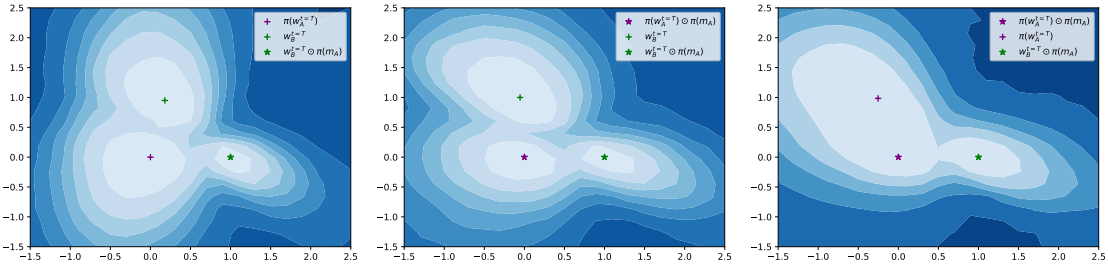
$$\mathcal{B}(\theta_1,\theta_2) := \sup_{\alpha\in[0,1]} \left[\mathcal{L}\big((1-\alpha)\theta_1+\alpha\theta_2\big) - \big((1-\alpha)\mathcal{L}(\theta_1)+\alpha\mathcal{L}(\theta_2)\big)\right] \geq 0\,, \tag{3}$$

where $\mathcal{L}$ is the loss function evaluated on the training dataset. If $\mathcal{B}(\theta_1,\theta_2) \approx 0$, it is said that θ_1 and θ_2 are linearly mode connected.

To ensure that the permutation mapping, π, can closely match model A and model B, we evaluate the loss barrier between the permuted model A and model B. However, aligning neurons alone is not sufficient to establish a low loss barrier due to variance collapse [27]. To overcome the variance collapse issue, we used REPAIR [27] to correct the variance of the activations in the interpolated/merged model. As shown in Fig. 3, the loss barrier after permutation matching and correcting the variance (REPAIR) is lower than the loss at random initialization, showing permutation mapping can match the models to bring them closer/in the loss basin.

Aligning Masks via Weight Symmetry. In contrast to previous works [6], we are interested in permuting the mask obtained by LTH such that the optimization basin of the permuted sparse mask and the new random initialization is aligned. To validate our hypothesis, we train two dense models, $\mathbf{w}_A^{t=0}$ and $\mathbf{w}_B^{t=0}$, where t denotes the epoch, to convergence (trained for T epochs) and then use activation matching (implemented by Jordan et al. [27]) to find the permutation mapping π, such that the activations of $\pi(\mathbf{w}_A^{t=T})$ and $\mathbf{w}_B^{t=T}$ are aligned. Mask $\mathbf{m}_A$, obtained using IMP, is also permuted with the same permutation map π. The intuition is that the permuted mask aligns with the loss basin of model $\mathbf{w}_B^{t=T}$, which is necessary for sparse training and, therefore, the sparse model can be more easily optimized (see Fig. 2). We denote training with the permuted mask, $\pi(\mathbf{m}_A)$ as *permuted* and with the non-permuted mask, $\mathbf{m}_A$ as *naive*. As illustrated Fig. 4a, the permuted model A, $\pi(\mathbf{w}_A^{t=T})$, is linearly mode connected with the converged model B, $\mathbf{w}_B^{t=T}$. Fig. 4b shows that the permuted LTH solution is linearly mode connected with the model B trained with the permuted mask. Fig. 4c shows that permuted LTH solution, permuted model A and permuted sparse solution all lie in the same basin.

Sparse Training. For evaluating the transferability of the permuted LTH mask, we use a new random initialization $\mathbf{w}_B^{t=0}$ and sparse masks $\mathbf{m}_A$ and $\pi(\mathbf{m}_A)$ for sparse training the naive and permuted solution respectively. We also evaluate the LTH baseline, i.e., training model $\mathbf{w}_A^{t=0}$ with mask $\mathbf{m}_A$. Since LTH requires weight rewinding to an earlier point in training, we also use a rewind checkpoint from epoch $t = k \ll T$ for both the baselines and permuted solution. In sparse training, the model is trained with a mask m, masking some of the weights, during both forward and backward passes.



(a) (b) (c)

Figure 4: We visualize the 0–1 loss landscape of ResNet20$\times\{4\}$/CIFAR-10. The figures are generated by evaluating the 0–1 loss spanned by three models in each corresponding figure. The sparse model in each of the figures is obtained by weight rewinding to achieve $\approx 90\%$ sparsity. We show that, modulo permutations, reusing the permuted mask leads to convergence in the same mode as the original model, i.e. the LTH solution. Hence, there is a small loss barrier between the permuted and LTH solutions, demonstrating they are within the same linearly connected mode. In the visualizations, lighter and darker regions correspond to lower and higher loss, respectively.

4. Results

To validate our hypothesis, we trained ResNet20 [35] and VGG11 [36] models on the CIFAR-10/100 datasets [37] (details in Appendix A.1) across different levels of sparsity ($S = 0.80, 0.90, 0.95, 0.97$). We used ResNet20 with varying widths ($w = 1, 4, 8, 16$) to study the effect of increasing width on the permutation matching and, thereby, the performance of the permuted sparse model. We also demonstrate our hypothesis on the large-scale ImageNet dataset [38] using ResNet50, showing the efficacy of our method across different models and datasets of varying sizes.

4.1. Experimental Results.

ResNet20/CIFAR-10 & CIFAR-100. We trained ResNet20 on the CIFAR-10/100 datasets. As shown in Figs. 5 and 6, the permuted solution outperforms the naive baseline across all model widths and rewind points. Since it is more difficult to train models with higher sparsity, the gap between naive and permuted solutions increases as sparsity increases, as shown in Figs. 5d, 5h and 5l. It can also be observed that at higher sparsity increasing the rewind point improves both the LTH and permuted solution but not the naive solution. The improved performance of the permuted solution over naive supports our hypothesis and shows that misalignment of LTH masks and loss basin corresponding to the new random initialization could explain why LTH masks do not transfer to different initializations. We also show accuracy vs. sparsity plots for $k = \{10, 25, 50, 100\}$ (details in Appendix A.5); as sparsity increases, the gap between permuted and naive solution increases for all rewind points. As illustrated in figure Fig. 5, neither the LTH nor the permuted solution performs effectively with random initialization ($k = 0$) but improves on increasing the rewind point up to a certain point, beyond which it plateaus. Detailed results are presented in Tables 4 to 7 in Appendix A.4.

We also validated our hypothesis on CIFAR-100 using ResNet20 with varying widths. As shown in Fig. 6, the permuted solution consistently outperforms the naive solution, showing that our hypothesis holds true across different models and datasets. Similar to the CIFAR-10 dataset, as we increase the width, the gap between the permuted and naive solution increases showing the efficacy of our method. Detailed results are presented in Tables 10 to 13 in Appendix A.4.

VGG11/CIFAR-10. We utilize the modified VGG11 architecture implemented by Jordan et al. [27] trained on CIFAR-10 (details in Appendix A.1). We observe that for a moderate sparsity (80%) in Fig. 8a, the gap between the permuted and the naive baseline is not large, however for a higher sparsity level (90%), the permuted solution significantly outperforms the naive solution as shown in Fig. 8b. For the VGG11 model, on increasing the rewind point, the permuted solution closely matches the accuracy of LTH, while the naive solution significantly plateaus and does not improve on increasing the rewind point. For higher sparsities, the naive baseline was unstable in training as

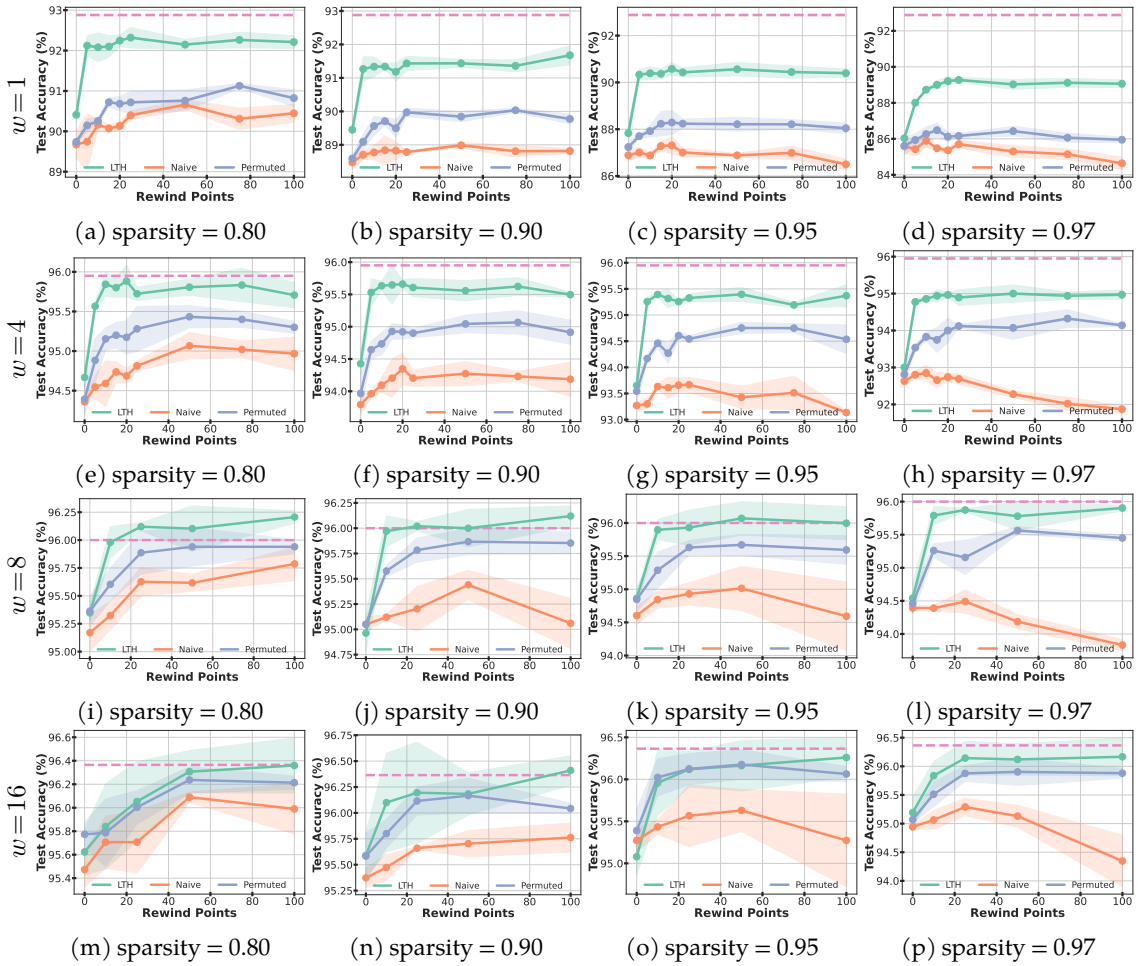
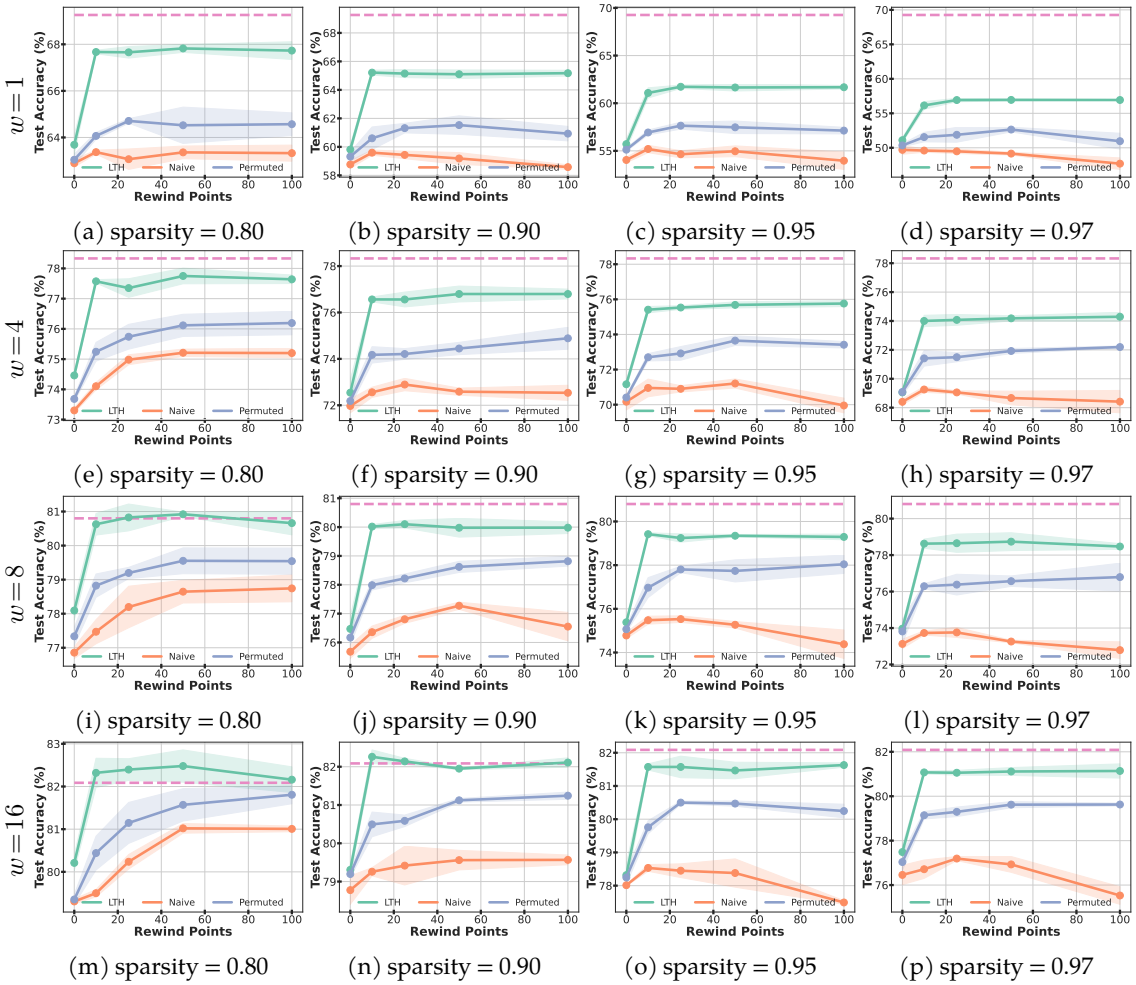


Figure 5: **ResNet20**$\times\{w\}$**/CIFAR-10**.Test accuracy of sparse network solutions vs. increasing rewind points for different sparsity levels and widths, w. The dashed (**- -**) line shows the dense model accuracy. The effect of the rewind point on the test accuracy for different sparsities is shown. As the width increases, the gap between training from a random initialization with the permuted mask and the LTH/dense baseline (dashed line) decreases, unlike training with the non-permuted mask (naive), showing a model trained with the permuted mask generalizes better than naive.

the modified VGG11 architecture does not have BatchNorm layers [39]; we omit those results in the discussion for a fair comparison. Detailed results are presented in Table 8 in Appendix A.4.

ResNet50/ImageNet. We also validated our hypothesis on the ILSVRC 2012 (ImageNet) dataset, which consists of 1.28 million images across 1,000 classes [38]. We used the ResNet50 model to evaluate the performance of the permuted mask at different sparsity levels. As observed in Fig. 7, the permuted solution outperforms the naive solution across all sparsity levels, showing that our hypothesis holds true on large-scale datasets as well. While the permuted solution performs better than the naive solution, there is still a significant gap between LTH and the permuted solution in the case of the ImageNet dataset as compared to the CIFAR-10/100 dataset. This could be due to permutation matching not being accurate enough, as only a small subset of the training dataset was used for activation matching. This can also be visualized in terms of the loss barrier in Fig. 3c between the permuted model A and model B; the loss barrier after permutation is more prominent compared to the CIFAR dataset (Figs. 3a and 3b). Thus, the permutation mapping π cannot match the models perfectly in the case of ImageNet since the permutation matching algorithm uses a greedy search algorithm to find the permutation mapping. However, given a perfect mapping, it could be possible to further improve the performance of the permuted solution as discussed in Section 4.2. Detailed



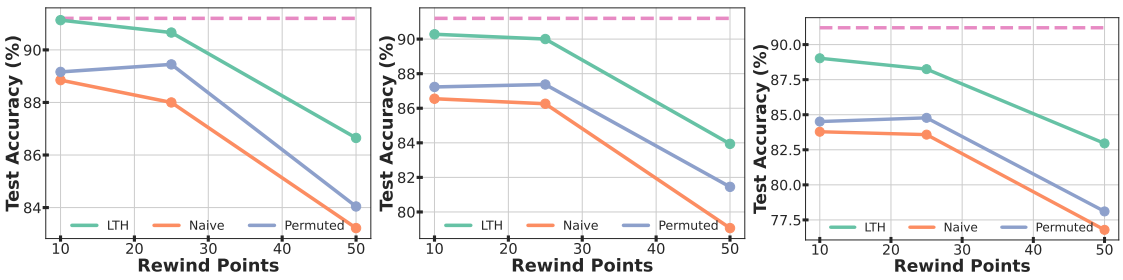
(a) sparsity = 0.80 (b) sparsity = 0.90 (c) sparsity = 0.95 (d) sparsity = 0.97

(e) sparsity = 0.80 (f) sparsity = 0.90 (g) sparsity = 0.95 (h) sparsity = 0.97

(i) sparsity = 0.80 (j) sparsity = 0.90 (k) sparsity = 0.95 (l) sparsity = 0.97

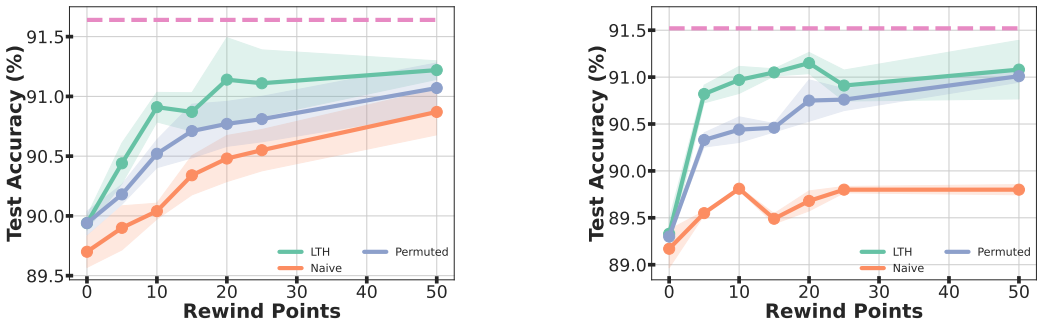
(m) sparsity = 0.80 (n) sparsity = 0.90 (o) sparsity = 0.95 (p) sparsity = 0.97

Figure 6: **ResNet20**$\times\{w\}$**/CIFAR-100**. Test accuracy of sparse network solutions vs. increasing rewind points for different sparsity levels and widths, w. The dashed (**- -**) line shows the dense model accuracy. The effect of the rewind points on the test accuracy for different sparsities is shown. As the width increases, the gap between training from a random initialization with the permuted mask and the LTH/dense baseline (dashed line) decreases, unlike training with the non-permuted mask (naive), showing model trained with the permuted model generalizes better than naive.



(a) sparsity = 0.80 (b) sparsity = 0.90 (c) sparsity = 0.95

Figure 7: **ResNet50**$\times\{1\}$**/ImageNet**. **Top-5** test accuracy vs. rewinds points of sparse network solutions at various sparsity levels. We observe the permuted solution consistently performing better than the naive solution for all sparsities. The dashed (**- -**) line shows the dense model accuracy.



(a) sparisty = 0.80 (b) sparsity = 0.90

Figure 8: **VGG11**$\times\{1\}$**/CIFAR-10.** Test accuracy of sparse network solutions at increasing rewind points for different sparsity levels. The dashed (**- -**) line shows the dense model accuracy. In Fig. 8b, the permuted solution closely matches the LTH solution. However, beyond a certain rewind point, particularly for $k \geq 20$ the performance of the naive solution plateaus. Resulting in a more noticeable gap between the permuted and naive solutions compared to Fig. 8a.

results are presented in Table 9 in Appendix A.4. As demonstrated in Table 9, the permuted solution outperforms the naive approach by nearly 2% at higher sparsity levels.

4.2. Effect of Model Width Multiplier.

Permutation matching is an NP-hard problem; the activation matching algorithm proposed by Ainsworth et al. [6] does not find the global optimum; rather, it uses a greedy search to explore a restricted solution space. Therefore, the permutation matching may not perfectly align/match two models. However, it has been observed that for wider models, the algorithm works better in practice and can closely align two models [6, 31]. To understand how the performance of the permuted model is affected by the approximation error of the matching algorithm, we evaluated the LMC and the accuracy of the permuted solution on ResNet20 models with varying layer widths. As shown in Fig. 3, on increasing the layer width, the loss barrier of the interpolated network reduces, showing that permutation mapping becomes more accurate and aligns two models better. Also, it can be observed in Figs. 5 and 6 that the permuted solution becomes close to the LTH solution on increasing the model width, showing that as the permutation matching becomes more accurate, the gap between the LTH and the permuted solution reduces. Given the hypothesis of Ainsworth et al. [6], i.e., *neural network loss landscapes nearly contain a single solution basin modulo permutations*, it may be that with an ideal/perfect permutation mapping, we would be able to train a sparse model with the permuted mask using a new random initialization that would match the LTH solution. However, our experiments still corroborate our hypothesis, and our work provides novel insights into why LTH does not transfer well to a new initialization.

5. Conclusion

Sparse training and the Lottery Ticket Hypothesis (LTH) have gained significant traction in recent years. In this work, we seek new insights into sparse training from random initialization and the LTH by leveraging weight symmetry in Deep Neural Networks (DNNs). Our empirical findings across various models and datasets support the hypothesis that misalignment between the mask and loss basin prevents effective use of LTH masks with new initialization. Although finding a permutation to align dense models is computationally expensive, the goal of our work is to develop insights into the working of LTH and how the sparse mask can be reused, not to improve the efficiency of LTH. We hope that our work will spur future work in this direction and will be useful to the research community working in the realm of sparse training.

References

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

[2] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *CoRR*, abs/1902.09574, 2019. URL http://arxiv.org/abs/1902.09574.

[3] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural network. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1135–1143, 2015. URL https://proceedings.neurips.cc/paper/2015/hash/ae0eb3eed39d2bcef4622b2499a05fe6-Abstract.html.

[4] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

[5] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=S1gSj0NKvB.

[6] Samuel K. Ainsworth, Jonathan Hayase, and Siddhartha S. Srinivasa. Git re-basin: Merging models modulo permutation symmetries. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=CQsmMYmlP5T.

[7] Vaishnavh Nagarajan and J. Zico Kolter. Uniform convergence may be unable to explain generalization in deep learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/05e97c207235d63ceb1db43c60db7bbb-Paper.pdf.

[8] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[9] Mansheej Paul, Feng Chen, Brett W. Larsen, Jonathan Frankle, Surya Ganguli, and Gintare Karolina Dziugaite. Unmasking the lottery ticket hypothesis: What's encoded in a winning ticket's mask? In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=xSsW2Am-ukZ.

[10] Utku Evci, Yani Ioannou, Cem Keskin, and Yann N. Dauphin. Gradient flow in sparse neural networks and how lottery tickets win. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 6577–6586. AAAI Press, 2022. doi: 10.1609/AAAI.V36I6.20611. URL https://doi.org/10.1609/aaai.v36i6.20611.

[11] Anna Choromanska, MIkael Henaff, Michael Mathieu, Gerard Ben Arous, and Yann LeCun. The Loss Surfaces of Multilayer Networks. In Guy Lebanon and S. V. N. Vishwanathan, editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pages 192–204, San Diego, California, USA, 09–12 May 2015. PMLR. URL https://proceedings.mlr.press/v38/choromanska15.html.

[12] Yann N. Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NeurIPS'14, page 2933–2941, Cambridge, MA, USA, 2014. MIT Press.

[13] Ian J. Goodfellow and Oriol Vinyals. Qualitatively characterizing neural network optimization problems. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6544.

[14] Chaoyue Liu, Libin Zhu, and Mikhail Belkin. Loss landscapes and optimization in over-parameterized non-linear systems and neural networks, 2021.

[15] Quynh Nguyen and Matthias Hein. The loss surface of deep and wide neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2603–2612. PMLR, 2017. URL http://proceedings.mlr.press/v70/nguyen17a.html.

[16] Berfin Simsek, François Ged, Arthur Jacot, Francesco Spadaro, Clément Hongler, Wulfram Gerstner, and Johanni Brea. Geometry of the loss landscape in overparameterized neural networks: Symmetries and invariances. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 9722–9732. PMLR, 2021. URL http://proceedings.mlr.press/v139/simsek21a.html.

[17] Alexander Shapiro and Arkadi Nemirovski. On complexity of stochastic programming problems. *Continuous optimization: Current trends and modern applications*, pages 111–146, 2005.

[18] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[19] Moritz Hardt, Ben Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1225–1234. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/hardt16.html.

[20] Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M. Kakade, and Michael I. Jordan. How to escape saddle points efficiently. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1724–1732. PMLR, 2017. URL http://proceedings.mlr.press/v70/jin17a.html.

[21] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Sy8gdB9xx.

[22] Kenji Kawaguchi. Deep learning without poor local minima. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 586–594, 2016. URL https://proceedings.neurips.cc/paper/2016/hash/f2fc990265c712c49d51a18a32b39f0c-Abstract.html.

[23] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural*

Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 5947–5956, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/10ce03a1ed01077e3e289f3e53c72813-Abstract.html.

[24] Robert Hecht-Nielsen. On the algebraic structure of feedforward network weight spaces, 1990.

[25] Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=dNigytemkL.

[26] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. ISBN 978-0-262-03561-3. URL http://www.deeplearningbook.org/.

[27] Keller Jordan, Hanie Sedghi, Olga Saukh, Rahim Entezari, and Behnam Neyshabur. REPAIR: renormalizing permuted activations for interpolation repair. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=gU5sJ6ZggcX.

[28] Sidak Pal Singh and Martin Jaggi. Model fusion via optimal transport. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/fb2697869f56484404c8ceee2985b01d-Abstract.html.

[29] N. Joseph Tatro, Pin-Yu Chen, Payel Das, Igor Melnyk, Prasanna Sattigeri, and Rongjie Lai. Optimizing mode connectivity via neuron alignment. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/aecad42329922dfc97eee948606e1f8e-Abstract.html.

[30] Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John Hopcroft. Convergent learning: Do different neural networks learn the same representations? In Dmitry Storcheus, Afshin Rostamizadeh, and Sanjiv Kumar, editors, *Proceedings of the 1st International Workshop on Feature Extraction: Modern Questions and Challenges at NIPS 2015*, volume 44 of *Proceedings of Machine Learning Research*, pages 196–212, Montreal, Canada, 11 Dec 2015. PMLR. URL https://proceedings.mlr.press/v44/li15convergent.html.

[31] Ekansh Sharma, Devin Kwok, Tom Denton, Daniel M. Roy, David Rolnick, and Gintare Karolina Dziugaite. Simultaneous linear connectivity of neural networks modulo permutation. In *Machine Learning and Knowledge Discovery in Databases. Research Track*, pages 262–279, Cham, 2024. Springer Nature Switzerland. URL https://doi.org/10.1007/978-3-031-70368-3_16.

[32] Dimitri Bertsekas. *Network optimization: continuous and discrete models*, volume 8. Athena Scientific, 1998.

[33] Akira Ito, Masanori Yamada, and Atsutoshi Kumagai. Analysis of linear mode connectivity via permutation-based weight matching. *CoRR*, abs/2402.04051, 2024. doi: 10.48550/ARXIV.2402.04051. URL https://doi.org/10.48550/arXiv.2402.04051.

[34] Harold W. Kuhn. The hungarian method for the assignment problem. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 29–47. Springer, 2010. doi: 10.1007/978-3-540-68279-0_2. URL https://doi.org/10.1007/978-3-540-68279-0_2.

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

[36] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.1556.

[37] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[38] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255. IEEE Computer Society, 2009. doi: 10.1109/CVPR.2009.5206848. URL https://doi.org/10.1109/CVPR.2009.5206848.

[39] Sergey Ioffe and Christian Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 448–456. JMLR.org, 2015.

[40] PyTorch Contributors. Pytorch examples, 2024. URL https://github.com/pytorch/examples/tree/main/imagenet. Accessed: 2024-11-26.

[41] Michela Paganini and Jessica Zosa Forde. Streamlining tensor and network pruning in pytorch. *CoRR*, abs/2004.13770, 2020. URL https://arxiv.org/abs/2004.13770.

A. Appendix

A.1. Implementation Details for ResNet20 & VGG11 on CIFAR-10/100

Architectures For residual neural networks, we train the standard ResNet20 on CIFAR-10 and CIFAR-100 with varying width. We implemented a scalar, w, that adjusts the number of channels in each convolutional and fully connected layer:

- **First Convolution Layer**: The number of output channels is scaled from 16 to $w \times 16$.
- **Layer 1,2,3**: The number of output channels for the convolutional blocks in these layers are scaled from 16, 32, and 64 to $w \times 16$, $w \times 32$, and $w \times 64$, respectively.
- **Fully Connected Layer**: The input dimension to the final linear layer is scaled to $w \times 64$.

For convolutional neural networks, we train a modified version of the standard VGG11 implemented by [27] on CIFAR-10. Primary differences are:

- A single fully connected layer at the end which directly maps the flattened feature map output from the convolutional layers to the 10 classes for CIFAR-10 classification.
- The classifier is set up for CIFAR-10 with 10 output classes as originally VGG11 was designed for ImageNet with 1000 output classes [38].

Each of our results for a given rewound point, k, is averaged over 3 runs.

Datasets For our set of experiments we used the CIFAR-10 and CIFAR-100 datasets [37]. We apply the following standard data augmentation techniques to the training set:

- `RandomHorizontalFlip`: Randomly flips the image horizontally with a given probability (by default, 50%).
- `RandomCrop`: Randomly crops the image to a size of 32×32 pixels, with a padding of 4 pixels around the image.

Optimizers We use the following hyperparameters for ResNet20 and VGG11 trained on CIFAR-10/100, as outlined in Table 1.

Hyperparameter	Value
Optimizer	SGD
Momentum	0.9
Dense Learning Rate	0.08
Sparse Learning Rate	0.02
Weight Decay	5×10^{-4}
Batch Size	128
Epochs (T)	200

Table 1: Hyperparameters for dense and sparse training of both ResNet20 and VGG11.

A.2. Implementation Details for ResNet50 on ImageNet

Architecture We utilize the standard ResNet50 implementation provided by torchvision and customize PyTorch's distributed data parallel codebase for training models on ImageNet [40].

Dataset For our set of experiments we used the ImageNet datatset [38]. We apply the following standard data augmentation techniques to the training set:

- `RandomHorizontalFlip`: Randomly flips the image horizontally with a given probability (by default, 50%).

- `RandomResizedCrop`: Randomly crops a region from the image and resizes it to 224×224 pixels.

Optimizers We use the following hyperparameters for ResNet50 trained on ImageNet, as outlined in Table 2.

Hyperparameter	Value
Optimizer	SGD
Momentum	0.9
Dense Learning Rate	0.4
Sparse Learning Rate	0.4
Weight Decay	1×10^{-4}
Batch Size	1024
Epochs (T)	80

Table 2: Hyperparameters for dense and sparse training of ResNet50.

A.3. Pruning

We apply standard Iterative Magnitude Pruning - Fine Tuning (IMP-FT) [3–5] to obtain our final mask, $\mathbf{m}_A$, producing a sparse subnetwork $\mathbf{w}_A^{t=T} \odot \mathbf{m}_A$. For pruning, we utilize PyTorch's `torch.nn.utils.prune` library [41].

1. In an unstructured, global manner, we identify and mask (set to zero) the smallest 20% of unpruned weights based on their magnitude.
2. This process is repeated for s rounds to achieve the target sparsity S, with each subsequent round pruning 20% of the remaining weights.
3. During each round, the model is trained for `train_epochs_per_prune` epochs.

Hyperparameter	ResNet20/VGG11	ResNet50
`train_epochs_per_prune`	50	20
Learning Rate	0.01	0.04

Table 3: Hyperparameters used for pruning ResNet20/VGG11 on CIFAR-10/100 and ResNet50 on ImageNet.

A.4. Results

Detailed results for ResNet20$\times\{w\}$/CIFAR-10 are provided in Tables 4 to 7, for VGG11$\times\{1\}$/CIFAR-10 in Table 8, for ResNet50$\times\{1\}$/ImageNet in Table 9, and for ResNet20$\times\{w\}$/CIFAR-100 in Tables 10 to 13.

A.5. Additional Plots

Refer to Figs. 9 and 10 for additional accuracy-vs-sparsity plots for ResNet20 on CIFAR-10 and CIFAR-100. Refer to Fig. 11 for Top-1 accuracy vs. rewind points for ResNet50 on ImageNet.

B. Computational Overhead of the Permuted Solution

The primary difference in computational complexity between the LTH, naive, and permuted solutions lies in the process of neuronal alignment, where weight/activation matching is used to locate permutations in order to bring the hidden units of two networks into alignment. To obtain the permuted solution, two distinct models must be trained independently to convergence, after which their weights or activations are aligned through a permutation-matching process. This alignment, though relatively efficient, adds a small computational overhead compared to LTH and naive solutions, which do not

Table 4: **ResNet20×{1}/CIFAR-10**. Results using the ResNet20×{1} trained on CIFAR-10, from a rewind point k, using various methods of sparse training with sparsity S. LTH trains within the original dense/pruned solution basin, while naive/permuted train from a new random initialization.

		Rewind Epoch k								
S	Method	$k=0$	5	10	15	20	25	50	75	100
80%	LTH	90.41 ± 0.14	92.12 ± 0.25	92.08 ± 0.36	92.10 ± 0.27	92.25 ± 0.14	92.32 ± 0.26	92.15 ± 0.13	92.26 ± 0.19	92.21 ± 0.16
	naive	89.67 ± 0.35	89.74 ± 0.69	90.16 ± 0.14	90.07 ± 0.09	90.13 ± 0.11	90.40 ± 0.11	90.66 ± 0.12	90.31 ± 0.27	90.45 ± 0.22
	perm.	89.74 ± 0.05	90.15 ± 0.16	90.26 ± 0.08	90.72 ± 0.12	90.68 ± 0.18	90.72 ± 0.28	90.76 ± 0.27	**91.13 ± 0.06**	90.82 ± 0.21
90%	LTH	89.45 ± 0.10	91.27 ± 0.37	91.34 ± 0.29	91.34 ± 0.09	91.18 ± 0.27	91.43 ± 0.22	91.44 ± 0.12	91.36 ± 0.18	91.68 ± 0.28
	naive	88.47 ± 0.21	88.70 ± 0.14	88.77 ± 0.21	88.84 ± 0.43	88.83 ± 0.27	88.78 ± 0.02	88.99 ± 0.08	88.81 ± 0.17	88.82 ± 0.07
	perm.	88.59 ± 0.11	89.09 ± 0.22	89.56 ± 0.28	89.71 ± 0.12	89.50 ± 0.27	89.97 ± 0.13	89.84 ± 0.15	**90.03 ± 0.07**	89.77 ± 0.15
95%	LTH	87.83 ± 0.38	90.33 ± 0.22	90.39 ± 0.28	90.37 ± 0.21	90.58 ± 0.26	90.43 ± 0.20	90.56 ± 0.29	90.44 ± 0.26	90.40 ± 0.19
	naive	86.89 ± 0.21	87.01 ± 0.23	86.88 ± 0.13	87.28 ± 0.19	87.31 ± 0.36	87.00 ± 0.19	86.88 ± 0.08	86.99 ± 0.29	86.50 ± 0.22
	perm.	87.24 ± 0.22	87.70 ± 0.08	87.92 ± 0.25	88.23 ± 0.52	**88.29 ± 0.52**	88.24 ± 0.20	88.21 ± 0.30	88.21 ± 0.20	88.04 ± 0.22
97%	LTH	86.03 ± 0.22	88.00 ± 0.02	88.73 ± 0.05	89.00 ± 0.24	89.21 ± 0.23	89.27 ± 0.14	89.03 ± 0.27	89.12 ± 0.25	89.06 ± 0.21
	naive	85.60 ± 0.38	85.43 ± 0.40	85.89 ± 0.37	85.48 ± 0.13	85.36 ± 0.14	85.70 ± 0.21	85.30 ± 0.32	85.14 ± 0.29	84.64 ± 0.34
	perm.	85.61 ± 0.48	85.93 ± 0.34	86.26 ± 0.40	**86.48 ± 0.39**	86.12 ± 0.27	86.16 ± 0.14	86.43 ± 0.27	86.06 ± 0.26	85.95 ± 0.14

Table 5: **ResNet20×{4}/CIFAR-10**. Results using the ResNet20×{4} trained on CIFAR-10, from a rewind point k, using various methods of sparse training with sparsity S. LTH trains within the original dense/pruned solution basin, while naive/permuted train from a new random initialization. Note this table is the same setting as Table 4 except $w=4$.

		Rewind Epoch k								
S	Method	k=0	5	10	15	20	25	50	75	100
80%	LTH	94.67 ± 0.14	95.57 ± 0.05	95.84 ± 0.15	95.80 ± 0.12	95.88 ± 0.20	95.72 ± 0.09	95.81 ± 0.10	95.83 ± 0.21	95.71 ± 0.16
	naive	94.36 ± 0.04	94.55 ± 0.14	94.59 ± 0.29	94.74 ± 0.13	94.69 ± 0.09	94.81 ± 0.06	95.07 ± 0.17	95.02 ± 0.11	94.97 ± 0.21
	perm.	94.39 ± 0.19	94.88 ± 0.28	95.15 ± 0.14	95.20 ± 0.16	95.17 ± 0.21	95.28 ± 0.29	**95.43 ± 0.14**	95.40 ± 0.10	95.30 ± 0.08
90%	LTH	94.43 ± 0.17	95.53 ± 0.21	95.63 ± 0.07	95.65 ± 0.30	95.66 ± 0.07	95.61 ± 0.14	95.56 ± 0.16	95.62 ± 0.14	95.50 ± 0.04
	naive	93.79 ± 0.15	93.96 ± 0.05	94.09 ± 0.11	94.20 ± 0.29	94.35 ± 0.25	94.20 ± 0.13	94.27 ± 0.19	94.23 ± 0.08	94.19 ± 0.27
	perm.	93.97 ± 0.29	94.64 ± 0.13	94.73 ± 0.17	94.93 ± 0.12	94.92 ± 0.11	94.90 ± 0.07	95.04 ± 0.14	**95.07 ± 0.18**	94.91 ± 0.19
95%	LTH	93.65 ± 0.12	95.26 ± 0.08	95.39 ± 0.05	95.32 ± 0.18	95.26 ± 0.03	95.33 ± 0.07	95.40 ± 0.14	95.19 ± 0.05	95.37 ± 0.21
	naive	93.27 ± 0.07	93.30 ± 0.11	93.63 ± 0.04	93.61 ± 0.21	93.66 ± 0.13	93.67 ± 0.14	93.43 ± 0.21	93.51 ± 0.32	93.14 ± 0.03
	perm.	93.54 ± 0.24	94.17 ± 0.07	94.46 ± 0.10	94.27 ± 0.19	94.61 ± 0.07	94.54 ± 0.07	**94.75 ± 0.11**	**94.75 ± 0.09**	94.54 ± 0.27
97%	LTH	93.00 ± 0.11	94.77 ± 0.09	94.86 ± 0.06	94.94 ± 0.17	94.96 ± 0.06	94.89 ± 0.21	95.00 ± 0.24	94.94 ± 0.10	94.97 ± 0.13
	naive	92.63 ± 0.12	92.80 ± 0.10	92.85 ± 0.21	92.66 ± 0.21	92.74 ± 0.11	92.69 ± 0.14	92.28 ± 0.09	92.02 ± 0.18	91.87 ± 0.10
	perm.	92.81 ± 0.27	93.54 ± 0.08	93.83 ± 0.12	93.75 ± 0.34	94.00 ± 0.33	94.12 ± 0.04	94.07 ± 0.31	**94.32 ± 0.24**	94.14 ± 0.04

involve matching steps. However, it's important to note that the primary goal of this study is not to improve training efficiency but rather to investigate why the LTH framework fails when applied to sparse training from new random initializations (not associated with the winning ticket's mask).

C. Full Symmetry Figure including Lottery Ticket Hypothesis

In Fig. 12 we include the full version of Fig. 1, including an illustration of the LTH in Fig. 12b.

Table 6: **ResNet20×{8}/CIFAR-10**. Results using the ResNet20×{8} trained on CIFAR-10, from a rewind point k, using various methods of sparse training with sparsity S. LTH trains within the original dense/pruned solution basin, while naive/permuted train from a new random initialization. Note this table is the same setting as Table 4 except $w=8$.

		Rewind Epoch k				
S	Method	k=0	10	25	50	100
80%	LTH	95.35 ± 0.07	95.98 ± 0.14	96.12 ± 0.04	96.10 ± 0.20	96.21 ± 0.06
	naive	95.17 ± 0.17	95.32 ± 0.13	95.63 ± 0.13	95.62 ± 0.08	95.79 ± 0.15
	perm.	95.36 ± 0.14	95.60 ± 0.15	95.89 ± 0.19	**95.94 ± 0.17**	**95.94 ± 0.06**
90%	LTH	94.96 ± 0.18	95.97 ± 0.15	96.02 ± 0.05	96.00 ± 0.19	96.12 ± 0.10
	naive	95.05 ± 0.07	95.12 ± 0.03	95.20 ± 0.22	95.44 ± 0.14	95.06 ± 0.25
	perm.	95.05 ± 0.05	95.58 ± 0.06	95.78 ± 0.12	**95.87 ± 0.13**	95.85 ± 0.11
95%	LTH	94.86 ± 0.08	95.90 ± 0.15	95.93 ± 0.26	96.07 ± 0.25	96.00 ± 0.25
	naive	94.60 ± 0.14	94.84 ± 0.13	94.93 ± 0.17	95.01 ± 0.33	94.59 ± 0.52
	perm.	94.85 ± 0.19	95.29 ± 0.27	95.63 ± 0.11	**95.67 ± 0.16**	95.59 ± 0.22
97%	LTH	94.54 ± 0.23	95.79 ± 0.14	95.87 ± 0.03	95.78 ± 0.21	95.90 ± 0.04
	naive	94.39 ± 0.04	94.39 ± 0.04	94.49 ± 0.18	94.19 ± 0.11	93.83 ± 0.08
	perm.	94.46 ± 0.14	95.26 ± 0.10	95.16 ± 0.26	**95.56 ± 0.06**	95.45 ± 0.05

Table 7: **ResNet20×{16}/CIFAR-10**. Results using the ResNet20×{8} trained on CIFAR-10, from a rewind point k, using various methods of sparse training with sparsity S. LTH trains within the original dense/pruned solution basin, while naive/permuted train from a new random initialization. Note this table is the same setting as Table 4 except $w=16$.

		Rewind Epoch k				
S	Method	k=0	10	25	50	100
80%	LTH	95.62 ± 0.19	95.84 ± 0.36	96.05 ± 0.34	96.31 ± 0.18	96.36 ± 0.24
	naive	95.47 ± 0.15	95.71 ± 0.22	95.71 ± 0.26	96.09 ± 0.04	95.99 ± 0.21
	perm.	95.77 ± 0.11	95.79 ± 0.29	96.00 ± 0.14	**96.24 ± 0.11**	96.21 ± 0.06
90%	LTH	95.59 ± 0.22	96.10 ± 0.48	96.19 ± 0.49	96.18 ± 0.20	96.41 ± 0.14
	naive	95.37 ± 0.09	95.47 ± 0.13	95.66 ± 0.01	95.70 ± 0.13	95.76 ± 0.14
	perm.	95.58 ± 0.22	95.80 ± 0.14	96.11 ± 0.13	**96.17 ± 0.17**	96.04 ± 0.05
95%	LTH	95.08 ± 0.21	95.96 ± 0.39	96.12 ± 0.21	96.16 ± 0.30	96.26 ± 0.23
	naive	95.27 ± 0.13	95.43 ± 0.09	95.57 ± 0.37	95.63 ± 0.25	95.27 ± 0.55
	perm.	95.39 ± 0.26	96.02 ± 0.22	96.12 ± 0.18	**96.18 ± 0.18**	96.06 ± 0.09
97%	LTH	95.19 ± 0.27	95.84 ± 0.25	96.14 ± 0.30	96.12 ± 0.27	96.17 ± 0.33
	naive	94.94 ± 0.04	95.06 ± 0.17	95.29 ± 0.15	95.13 ± 0.19	94.35 ± 0.45
	perm.	95.07 ± 0.06	95.51 ± 0.22	95.88 ± 0.14	**95.90 ± 0.24**	95.88 ± 0.09

Table 8: **VGG11**$\times\{1\}$**/CIFAR-10**. Results using the VGG11 trained on CIFAR-10, from a rewind point k, using various methods of sparse training with sparsity S. LTH trains within the original dense/pruned solution basin, while naive/permuted train from a new random initialization.

		Rewind Epoch k						
S	Method	$k=0$	5	10	15	20	25	50
80%	LTH	89.94 ± 0.06	90.44 ± 0.17	90.91 ± 0.12	90.87 ± 0.16	91.14 ± 0.28	91.11 ± 0.08	91.22 ± 0.08
	naive	89.70 ± 0.13	89.90 ± 0.18	90.04 ± 0.07	90.34 ± 0.16	90.48 ± 0.19	90.55 ± 0.17	90.87 ± 0.19
	perm.	89.94 ± 0.1	90.18 ± 0.08	90.52 ± 0.17	90.71 ± 0.22	90.77 ± 0.19	90.81 ± 0.19	**91.07 ± 0.21**
90%	LTH	89.33 ± 0.16	90.82 ± 0.09	90.97 ± 0.14	91.05 ± 0.04	91.15 ± 0.11	90.91 ± 0.17	91.08 ± 0.31
	naive	89.17 ± 0.2	89.55 ± 0.02	89.81 ± 0.02	89.49 ± 0.05	89.68 ± 0.11	89.80 ± 0.03	89.80 ± 0.05
	perm.	89.30 ± 0.02	90.33 ± 0.08	90.44 ± 0.14	90.46 ± 0.04	90.75 ± 0.22	90.76 ± 0.12	**91.01 ± 0.06**

Table 9: **ResNet50**$\times\{1\}$**/ImageNet**. Top-1 and Top-5 Accuracies of ResNet50 trained on ImageNet, from a rewind point k, using various methods of sparse training with sparsity S.

		Top-1 Accuracy			Top-5 Accuracy		
S	Method	$k=10$	25	50	$k=10$	25	50
80%	LTH	72.87	72.16	65.23	91.13	90.66	86.65
	naive	69.13	68.94	60.30	88.85	88.1	83.22
	perm.	**69.87**	69.85	61.14	89.16	**89.45**	84.04
90%	LTH	71.40	70.74	60.62	90.27	90.00	83.94
	naive	65.49	64.77	54.46	86.55	86.26	79.07
	perm.	66.25	**66.37**	57.40	87.23	**87.37**	81.45
95%	LTH	68.61	68.07	59.83	89.03	88.25	82.96
	naive	61.39	60.77	51.78	83.79	83.58	76.79
	perm.	62.48	**62.77**	52.98	84.51	**84.79**	78.11

Table 10: **ResNet20**$\times\{1\}$**/CIFAR-100**. Results using the ResNet20$\times\{1\}$ trained on CIFAR-100, from a rewind point k, using various methods of sparse training with sparsity S. LTH trains within the original dense/pruned solution basin, while naive/permuted train from a new random initialization.

		Rewind Epoch k				
S	Method	k=0	10	25	50	100
80%	LTH	63.69 ± 0.41	67.67 ± 0.08	67.66 ± 0.25	67.82 ± 0.17	67.73 ± 0.38
	naive	62.89 ± 0.16	63.37 ± 0.09	63.07 ± 0.44	63.36 ± 0.27	63.33 ± 0.35
	perm.	63.04 ± 0.24	64.07 ± 0.15	**64.71 ± 0.10**	64.52 ± 0.78	64.57 ± 0.49
90%	LTH	59.81 ± 0.29	65.21 ± 0.17	65.15 ± 0.28	65.10 ± 0.30	65.17 ± 0.21
	naive	58.77 ± 0.28	59.59 ± 0.18	59.44 ± 0.27	59.19 ± 0.41	58.58 ± 0.16
	perm.	59.32 ± 0.32	60.60 ± 0.79	61.32 ± 0.33	**61.53 ± 0.65**	60.93 ± 0.51
95%	LTH	55.71 ± 0.52	61.08 ± 0.54	61.73 ± 0.18	61.65 ± 0.37	61.68 ± 0.18
	naive	54.04 ± 0.29	55.20 ± 0.39	54.65 ± 0.38	54.96 ± 0.57	53.97 ± 0.91
	perm.	55.12 ± 0.17	56.93 ± 0.26	**57.64 ± 0.36**	57.47 ± 0.66	57.13 ± 0.34
97%	LTH	51.10 ± 0.34	56.14 ± 0.56	56.92 ± 0.25	56.94 ± 0.13	56.93 ± 0.06
	naive	49.70 ± 0.64	49.60 ± 0.25	49.49 ± 0.32	49.16 ± 0.21	47.70 ± 0.83
	perm.	50.34 ± 0.21	51.55 ± 0.69	51.88 ± 1.08	**52.64 ± 0.34**	50.96 ± 1.15

Table 11: **ResNet20×{4}/CIFAR-100**. Results using the ResNet20×{4} trained on CIFAR-100, from a rewind point k, using various methods of sparse training with sparsity S. LTH trains within the original dense/pruned solution basin, while naive/permuted train from a new random initialization. Note this table is the same setting as Table 10 except $w=4$.

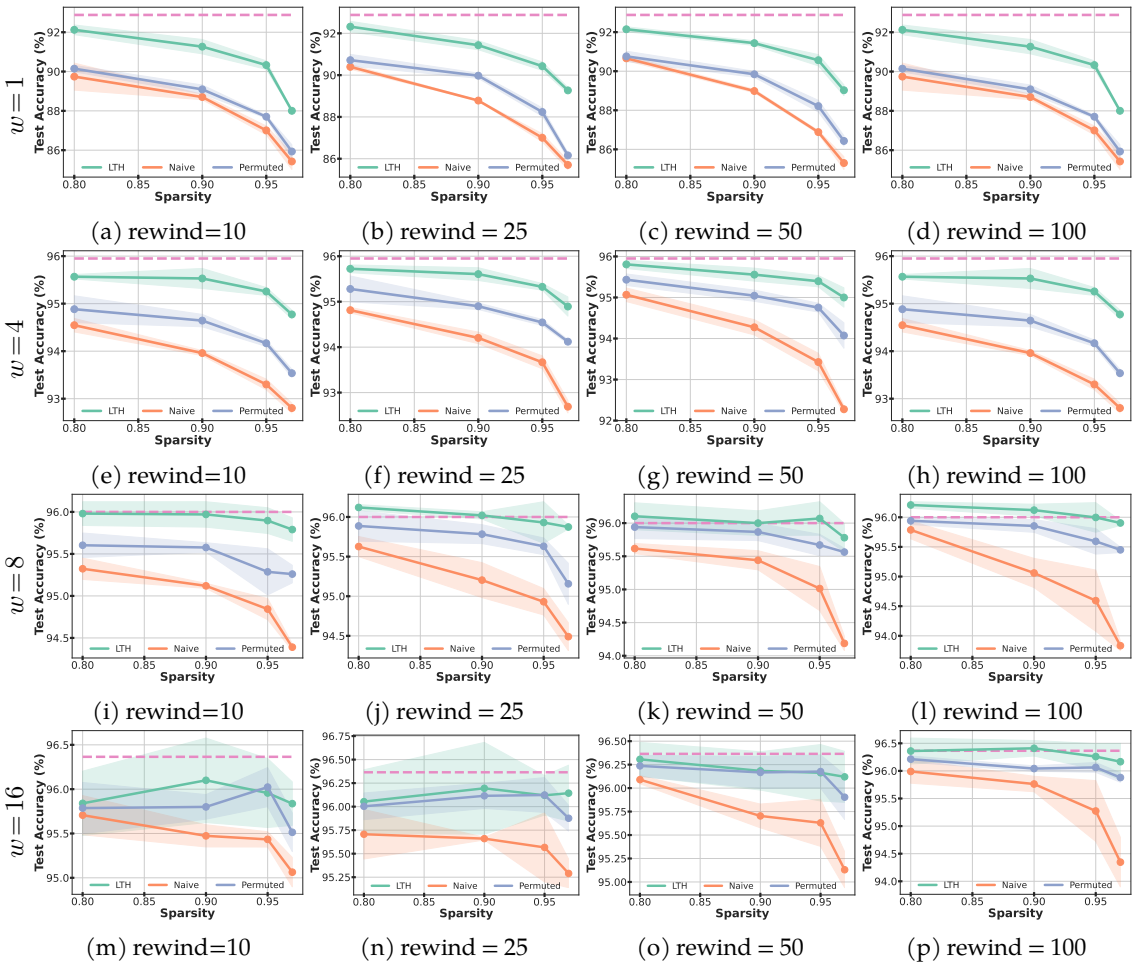
		Rewind Epoch k				
S	Method	k=0	10	25	50	100
80%	LTH	74.46 ± 0.12	77.57 ± 0.06	77.35 ± 0.31	77.75 ± 0.26	77.64 ± 0.14
	naive	73.30 ± 0.08	74.10 ± 0.12	74.98 ± 0.17	75.21 ± 0.12	75.20 ± 0.16
	perm.	73.68 ± 0.09	75.24 ± 0.31	75.74 ± 0.41	76.12 ± 0.37	**76.19 ± 0.39**
90%	LTH	72.54 ± 0.57	76.56 ± 0.11	76.56 ± 0.32	76.80 ± 0.34	76.80 ± 0.21
	naive	71.97 ± 0.30	72.56 ± 0.22	72.89 ± 0.27	72.59 ± 0.15	72.54 ± 0.33
	perm.	72.18 ± 0.23	74.17 ± 0.35	74.21 ± 0.23	74.45 ± 0.27	**74.89 ± 0.47**
95%	LTH	71.16 ± 0.23	75.41 ± 0.18	75.53 ± 0.11	75.68 ± 0.17	75.76 ± 0.17
	naive	70.17 ± 0.47	70.95 ± 0.50	70.90 ± 0.18	71.21 ± 0.26	69.95 ± 0.42
	perm.	70.41 ± 0.07	72.70 ± 0.21	72.92 ± 0.39	**73.65 ± 0.28**	73.41 ± 0.18
97%	LTH	69.06 ± 0.03	74.00 ± 0.39	74.08 ± 0.37	74.18 ± 0.18	74.29 ± 0.31
	naive	68.40 ± 0.21	69.26 ± 0.19	69.06 ± 0.11	68.67 ± 0.47	68.42 ± 0.78
	perm.	69.08 ± 0.22	71.41 ± 0.54	71.49 ± 0.32	71.92 ± 0.17	**72.20 ± 0.08**

Table 12: **ResNet20×{8}/CIFAR-100**. Results using the ResNet20×{8} trained on CIFAR-100, from a rewind point k, using various methods of sparse training with sparsity S. LTH trains within the original dense/pruned solution basin, while naive/permuted train from a new random initialization. Note this table is the same setting as Table 10 except $w=8$.

		Rewind Epoch k				
S	Method	k=0	10	25	50	100
80%	LTH	78.09 ± 0.28	80.63 ± 0.32	80.83 ± 0.39	80.92 ± 0.06	80.66 ± 0.34
	naive	76.86 ± 0.17	77.47 ± 0.35	78.20 ± 0.61	78.65 ± 0.33	78.74 ± 0.39
	perm.	77.34 ± 0.26	78.82 ± 0.34	79.20 ± 0.16	**79.55 ± 0.38**	79.54 ± 0.39
90%	LTH	76.47 ± 0.43	80.02 ± 0.07	80.10 ± 0.13	79.98 ± 0.33	79.98 ± 0.20
	naive	75.68 ± 0.23	76.36 ± 0.21	76.80 ± 0.14	77.27 ± 0.12	76.55 ± 0.49
	perm.	76.17 ± 0.26	77.99 ± 0.17	78.22 ± 0.15	78.62 ± 0.19	**78.82 ± 0.17**
95%	LTH	75.38 ± 0.02	79.42 ± 0.06	79.24 ± 0.19	79.35 ± 0.06	79.29 ± 0.13
	naive	74.78 ± 0.15	75.48 ± 0.18	75.53 ± 0.15	75.27 ± 0.15	74.38 ± 0.65
	perm.	75.07 ± 0.14	76.97 ± 0.46	77.80 ± 0.14	77.74 ± 0.51	**78.04 ± 0.42**
97%	LTH	73.97 ± 0.21	78.63 ± 0.25	78.65 ± 0.50	78.74 ± 0.49	78.47 ± 0.16
	naive	73.13 ± 0.26	73.73 ± 0.12	73.76 ± 0.27	73.26 ± 0.07	72.79 ± 0.46
	perm.	73.81 ± 0.67	76.29 ± 0.14	76.38 ± 0.57	76.57 ± 0.29	**76.79 ± 0.76**

Table 13: **ResNet20$\times\{16\}$/CIFAR-100**. Results using the ResNet20$\times\{16\}$ trained on CIFAR-100, from a rewind point k, using various methods of sparse training with sparsity S. LTH trains within the original dense/pruned solution basin, while naive/permuted train from a new random initialization. Note this table is the same setting as Table 10 except $w=16$.

		Rewind Epoch k				
S	Method	k=0	10	25	50	100
80%	LTH	80.21 ± 0.18	82.32 ± 0.34	82.40 ± 0.26	82.48 ± 0.38	82.16 ± 0.30
	naive	79.31 ± 0.06	79.50 ± 0.09	80.24 ± 0.17	81.02 ± 0.11	81.01 ± 0.07
	perm.	79.35 ± 0.11	80.44 ± 0.40	81.15 ± 0.48	81.57 ± 0.38	**81.81 ± 0.21**
90%	LTH	79.31 ± 0.16	82.26 ± 0.18	82.14 ± 0.08	81.95 ± 0.03	82.11 ± 0.12
	naive	78.78 ± 0.37	79.26 ± 0.11	79.42 ± 0.51	79.56 ± 0.26	79.57 ± 0.13
	perm.	79.20 ± 0.09	80.49 ± 0.32	80.59 ± 0.15	81.12 ± 0.05	**81.24 ± 0.09**
95%	LTH	78.32 ± 0.34	81.57 ± 0.09	81.57 ± 0.32	81.47 ± 0.25	81.63 ± 0.07
	naive	78.01 ± 0.02	78.53 ± 0.10	78.45 ± 0.21	78.38 ± 0.43	77.49 ± 0.06
	perm.	78.25 ± 0.20	79.76 ± 0.20	**80.50 ± 0.04**	80.47 ± 0.08	80.25 ± 0.21
97%	LTH	77.49 ± 0.27	81.07 ± 0.07	81.06 ± 0.11	81.11 ± 0.18	81.14 ± 0.32
	naive	76.46 ± 0.44	76.71 ± 0.41	77.19 ± 0.09	76.93 ± 0.36	75.53 ± 0.40
	perm.	77.04 ± 0.38	79.14 ± 0.17	79.30 ± 0.21	79.62 ± 0.14	**79.63 ± 0.06**



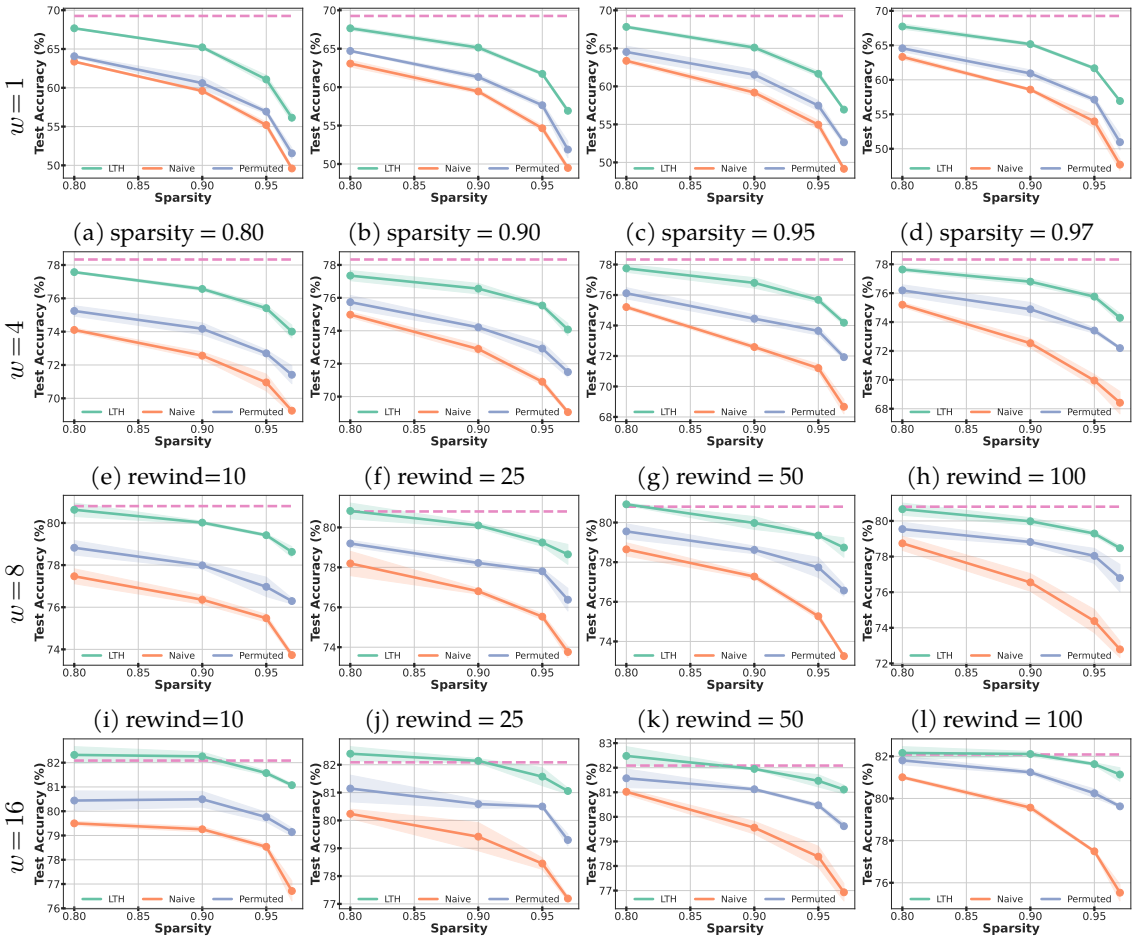
(a) rewind=10 (b) rewind = 25 (c) rewind = 50 (d) rewind = 100

(e) rewind=10 (f) rewind = 25 (g) rewind = 50 (h) rewind = 100

(i) rewind=10 (j) rewind = 25 (k) rewind = 50 (l) rewind = 100

(m) rewind=10 (n) rewind = 25 (o) rewind = 50 (p) rewind = 100

Figure 9: **Accuracy vs sparsity trend for ResNet20$\times\{w\}$/CIFAR-10**.As the width increases, the gap between permuted and naive solutions increases, showing permuted masks help with sparse training. With increased width, we observe a more significant gap seen throughout Figs. 9d, 9h, 9l and 9p and the permuted solution approaches the LTH solution. The dashed (**- -**) line shows the dense model accuracy.



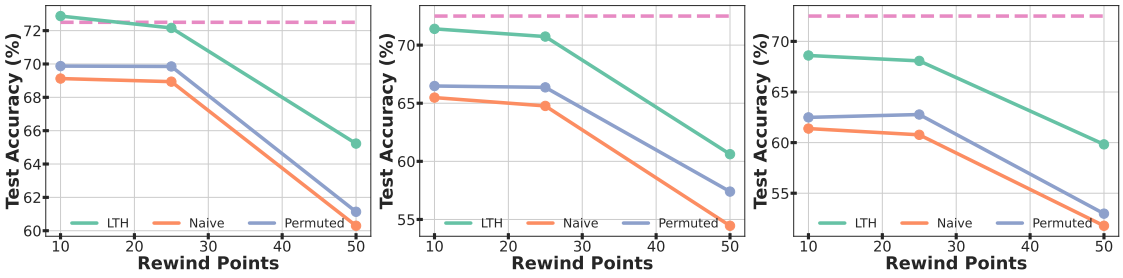
(a) sparsity = 0.80 (b) sparsity = 0.90 (c) sparsity = 0.95 (d) sparsity = 0.97

(e) rewind=10 (f) rewind = 25 (g) rewind = 50 (h) rewind = 100

(i) rewind=10 (j) rewind = 25 (k) rewind = 50 (l) rewind = 100

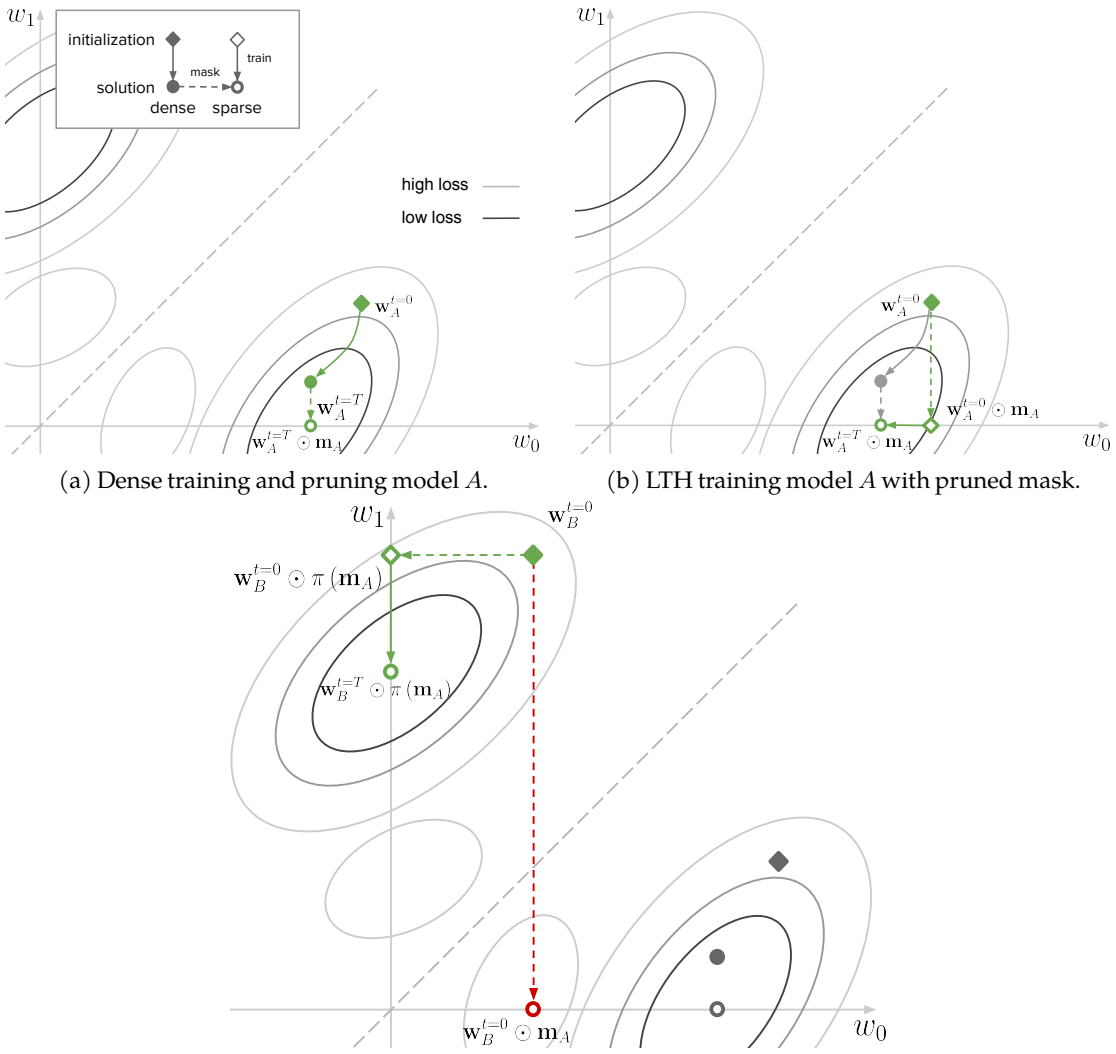
(m) rewind=10 (n) rewind = 25 (o) rewind = 50 (p) rewind = 100

Figure 10: **Accuracy vs sparsity trend for ResNet20$\times\{w\}$/CIFAR-100**. Similar to the phenomenon seen in Fig. 9, with higher width the gap between permuted and naive solutions increases. As seen in Figs. 10d, 10h, 10l and 10p and the permuted solution approaches the LTH solution. The dashed (**- -**) line shows the dense model accuracy.



(a) sparsity = 0.80 (b) sparsity = 0.90 (c) sparsity = 0.95

Figure 11: **ResNet50$\times\{1\}$/ImageNet**. Top-1 test accuracy vs rewinds points of sparse network solutions at various sparsity levels. We observe the permuted solution consistently peroforming better than the naive solution for all sparsities. The dashed (**- -**) line shows the dense model accuracy.



(a) Dense training and pruning model A.

(b) LTH training model A with pruned mask.

(c) Sparse training model B with A mask.

Figure 12: **Weight Symmetry and the Sparse Training Problem (Full Figure)**. A model with a single layer and only two parameters, $\mathbf{w} = (w_0, w_1)$, operating on a single input scale x_0 has the weight symmetry in the 2D loss landscape as illustrated above. In (a) the original dense model, $\mathbf{w}_A$, is trained from a random dense initialization, $\mathbf{w}_A^{t=0}$ to a dense solution, $\mathbf{w}_A^{t=T}$, which is then pruned using weight magnitude resulting in the mask $\mathbf{m}_A = (1,0)$. In (b) we re-use the init. $\mathbf{w}_A^{t=0}$, to train model A with the pruned mask from (a), $\mathbf{m}_A$, as in LTH. In (c), naively using the same mask to train a model, B, from a different random initialization will likely result in the initialization being far from a good solution. Permuting the mask to match the (symmetric) basin in which the new initialization is in will enable sparse training.