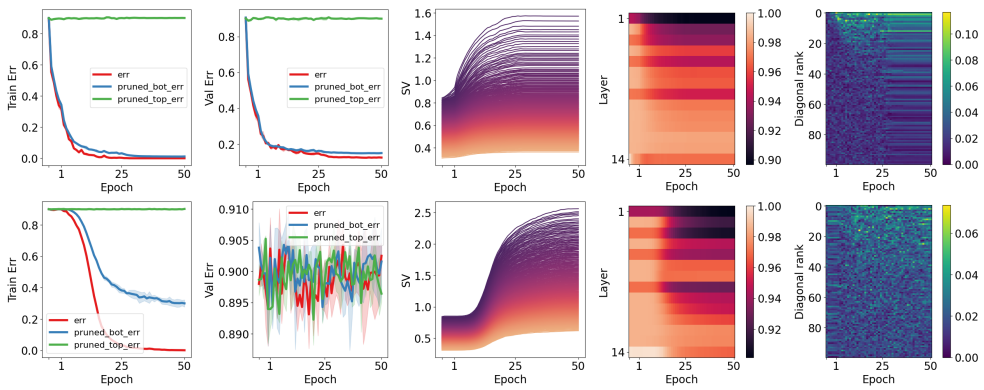


(a) Train Err. (b) Val. Err. (c) SVs (d) Eff. Rank (e) Alignment

Figure 8: Dynamics with random labels for VGG. **Top row:** results with true labels. **Bottom row:** results with random labels. We see that the middle layers have a lower effective rank when using true labels and that alignment in the middle layers persists throughout training. The results are less stark in the VGG case, but similar to the MLP.

A Explanation of Balancedness

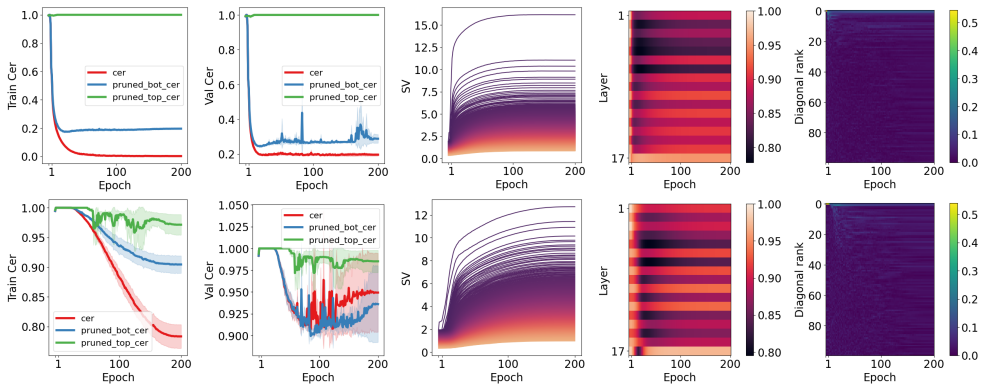
Prior work on deep linear networks (Arora et al., 2019; Milanesi et al., 2021) suggests that rank minimization may describe implicit regularization in deep matrix factorization better than simple matrix norms. See Arora et al. (2018) (Appendix A) for a detailed argument. However, a critical assumption used in these works is "balanced initialization." This means that for consecutive matrices W_i and W_{i+1} in the product matrix $\prod_j W_j$, we have $W_{i+1}^\top W_{i+1} = W_i W_i^\top$ at initialization. Decomposing these matrices with SVDs and leveraging orthogonality, this simplifies to $V_{i+1}\Sigma_{i+1}^2 V_{i+1}^\top = U_i \Sigma_i^2 U_i^\top$ where U_i and V_{i+1} are orthogonal matrices. Since these are orthogonal decompositions of the same matrix, their diagonals must be equivalent, allowing for the permutation of elements with the same value. This leads to $U_i = V_{i+1}O$ up to signs, where O is a block diagonal permutation matrix that may permute the rows of equivalent diagonal elements. Notably, if all diagonal elements are distinct and U_i and V_{i+1} are square matrices, then $U_i = V_{i+1}$ up to signs. This gives us matching singular vectors for consecutive matrices.

B Spectral Dynamics with Random Labels

Given the observations connecting generalization and rank thus far, and the enlightening view on the implicit effects of weight decay, we are interested in seeing whether the perspective developed sheds any light on the classic random label memorization experiments of Zhang et al. (2021).

Similar to Zhang et al. (2021), we train a MLP, VGG and an LSTM to fit random or true labels. Please see Appendix D for the details regarding the experimental setup. Zhang et al. (2021) decay the learning rate to zero, and the random label experiments only converge late in training. Consequently, we use a constant learning rate to control this phenomenon. We see in Figure 7 that both cases are able to achieve zero error, though with different singular value evolution and alignment in the middle layer.

Surprisingly in Figure 7, we see that with true labels the inner layers are low rank, while with random labels they are much higher rank. This may be explained by the shared structure in the true classes of the dataset, which manifests in the parameters. Even more surprisingly, we find here that even without weight decay, inner layers align with true labels, while with random labels, this alignment occurs and then disappears with more training. This is particularly intriguing as there are non-linearities that could theoretically separate the network from the linear case, and yet strong alignment occurs despite that. Such alignment has not yet been leveraged by existing theory, and might provide structured assumptions for new understanding. Results on the VGG (Figure 8) are qualitatively quite similar, including on the alignment point. Results on the LSTM (Figure 9)



(a) Train Err. (b) Val. Err. (c) SVs (d) Eff. Rank (e) Alignment

Figure 9: Dynamics with random labels for LSTM. **Top row:** results with true labels. **Bottom row:** results with random labels. We see that the middle layers have a lower effective rank when using true labels and that alignment in the middle layers persists throughout training. Though the LSTM doesn't fit the random labels perfectly, the results are qualitatively similar to the other cases, except alignment is almost nonexistent.

are weakly similar, though the alignment is much weaker. In summary, these results suggest that viewing generalization through the lens of rank and alignment may be fruitful.

C Beyond Generalization

We have seen over the course of many experiments that deep models are biased toward low rank, and that there is a tempting connection between rank minimization and generalization. Still, the lens of spectral dynamics can be applied more broadly. In the following subsections, we explore two phenomena: lottery tickets (Frankle & Carbin, 2018) and linear mode connectivity (Frankle et al., 2020). Beyond shedding further light on neural networks, these phenomena have implications for more efficient inference and storage, as well as understanding the importance of pretraining (Neyshabur et al., 2020). We find that lottery tickets are a sparse approximation of final-checkpoint top singular vectors. The ability to linearly interpolate between faraway checkpoints and improve performance coincides strongly with top singular vector sharing between checkpoints. Such observations may form a foundation for a better understanding compression and model averaging (Wortsman et al., 2022; Ilharco et al., 2022).

C.1 Top Singular Vectors Become Stable Earlier

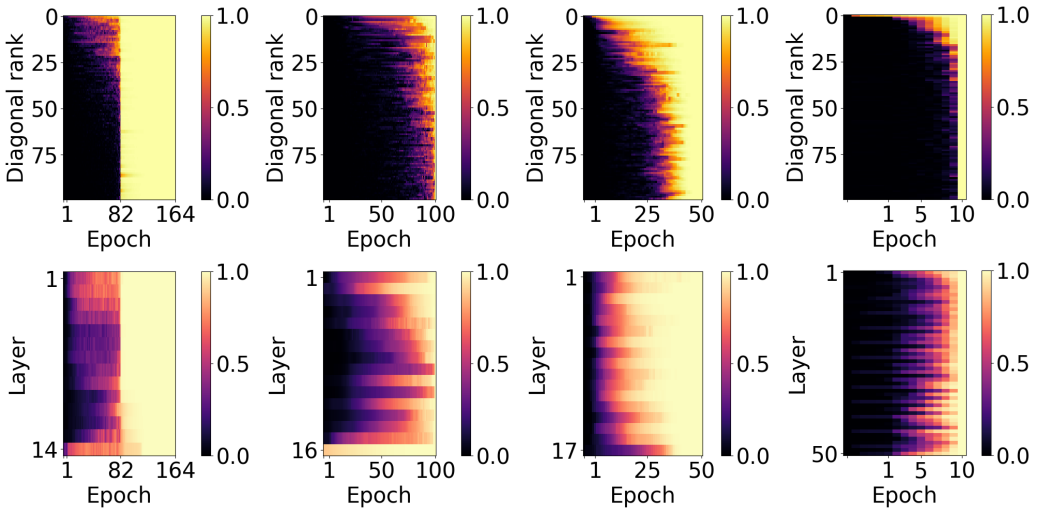
Before we explore the phenomena, we first make another observation that will be helpful. As top singular values grow disproportionately large, it would be natural that top singular vectors become stable in direction as the gradients remain small. To demonstrate this, for a given matrix in the network $W_i(t) = \sum_{j=1}^{R} \sigma_j(t) u_j(t) v_j(t)^\top$ at training time t, we compute

$$S(t)_{jk} = |\langle u_j(t) v_j(t)^\top, u_k(T) v_k(T)^\top \rangle|, \tag{5}$$

where T is the final step of training, and the absolute value is taken to ignore sign flips in the SVD computation. We then plot the diagonal of this matrix $S(t)_{ii}\ \forall\ i \leq 100$ over time. We also use a scalar measure of the diagonal to summarize like in the alignment case: $s(t) = \frac{1}{10}\sum_i S(t)_{ii}$. In Figure 10, we see that top singular vectors converge in direction earlier than bottom vectors.

C.2 Lottery Tickets Preserve Final Top Singular Vectors

As large singular vectors will become stable late in training, we wonder about the connection to magnitude pruning and the lottery ticket hypothesis. Frankle & Carbin (2018) first showed evidence for the lottery ticket hypothesis, the idea that there exist sparse subnetworks of neural networks that



(a) VGG (b) UNet (c) LSTM (d) Transformer

Figure 10: **Top row:** Singular vector agreement for a single matrix in the middle of each model (diagonal of Eqn. 5). Notice top singular vectors become stable in direction earlier. **Bottom row:** Summary score for each matrix across architectures. As we move down the y-axis, the depth of the parameters in the model increases, while the x-axis tracks training time. The sharp transition midway through training in the VGG case is likely due to a 10x learning rate decay.

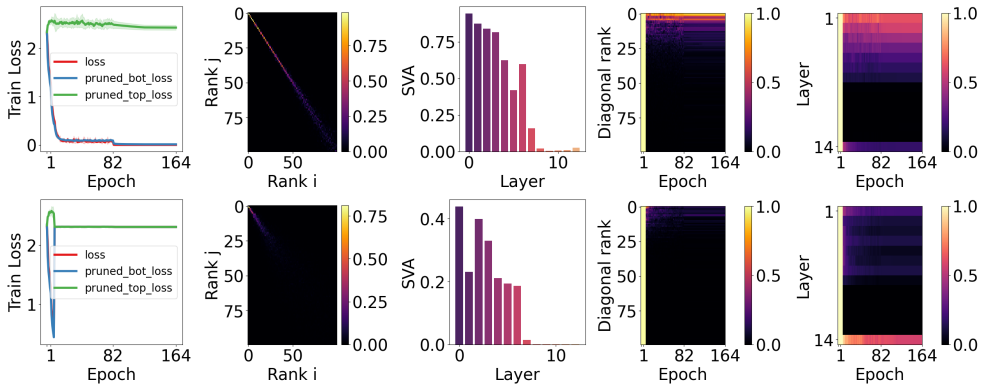
can be trained to a comparable performance as the full network, where the sparse mask is computed from the largest magnitude weights of the network at the end of training. Frankle et al. (2020) build further on this hypothesis and notice that, for larger networks, the masking cannot begin at initialization, but rather at some point early in training. Still, the mask must come from the end of training.

The reason for this particular choice of mask may be connected to the dynamics we previously observed. Specifically, at the end of training large singular values are disproportionately larger, so high-magnitude weights may correspond closely to weights in the top singular vectors at the end of training. If magnitude masks were computed at the beginning, the directions that would become the top singular vectors might be prematurely masked as they have not yet stabilized, which may prevent learning on the task.

Here we train an unmasked VGG-16 (Simonyan & Zisserman, 2014) on CIFAR10, then compute either a random mask, or a global magnitude mask from the end of training, and rewind to an early point (Frankle et al., 2020) to start sparse retraining. We also do the same with an LSTM (Hochreiter & Schmidhuber, 1997b) on LibriSpeech (Panayotov et al., 2015). Please see Appendix D for details. In Figures 11 and 12, we plot the singular vector agreement (SVA, Eqn. 5) between the final model, masked and unmasked, where we see exactly that magnitude masks preserve the top singular vectors of parameters, and with increasing sparsity fewer directions are preserved. Even though prior work has remarked that it is possible to use low-rank approximations for neural networks (Yu et al., 2017), and others have explicitly optimized for low-rank lottery tickets (Wang et al., 2021; Schotthöfer et al., 2022), we rather are pointing out that the magnitude pruning procedure seems to recover a low-rank approximation.

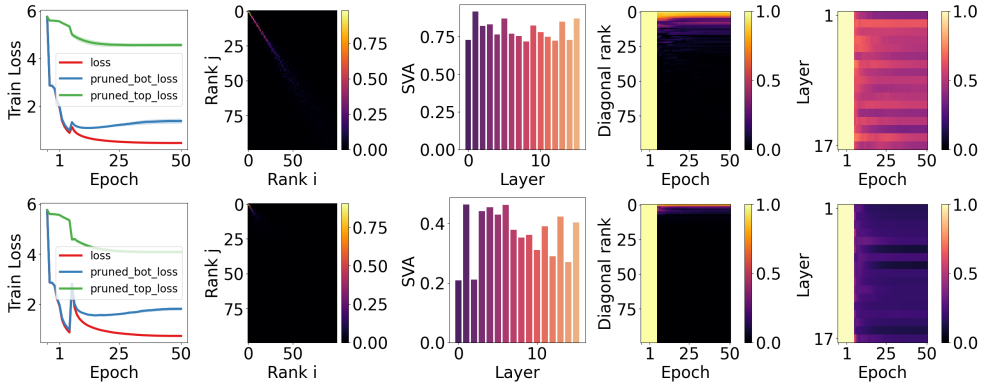
We also compute the singular vector agreement (SVA) between the masked model trajectory and the original unmasked model trajectory (diagonal of Eqn. 5). We see in Figures 11 and 12 that there is no agreement between the bottom singular vectors at all, but there is still loose agreement in the top singular vectors. Thus, it seems the mask allows the dynamics of only the top singular vectors to remain similar, which we know are most important from the pruning analysis in Figure 4.

Preserving top singular vectors by pruning seems like a natural outcome of large matrices, so as a control, we follow exactly the same protocol except we generate the mask randomly with the same layerwise sparsity. We can see in Figures 11 and 12 that this results in much lower preservation of



(a) Loss (b) Pruned SVA (c) All Layers (d) SVA evol. (e) All Layers

Figure 11: Pruning results for VGG. **Top row:** Magnitude pruning. **Bottom row:** random pruning. **First column:** Training loss. We see that at 5% sparsity magnitude pruning is significantly better than random pruning of the same layerwise sparsity. **2nd column:** Singular vector alignment pre- and post-pruning at the end of training for a single layer (the 3rd convolution). We see that magnitude pruning approximates the top singular vectors, while random pruning at the same level does not. **3rd column:** Singular vector alignment score pre- and post-pruning across all layers. Agreement is higher across all layers for magnitude pruning, though later layers do not agree, likely as later layers are wider so weights are lower magnitude. **4th column:** Singular vector alignment between the pruned and unpruned models along the training trajectory. We see that the magnitude pruning still has similar dynamics in its top singular vectors, while random pruning does not. **Last column:** Singular vector alignment score between pruned and unpruned models across layers and time. Again evolution is similar for early layers with magnitude pruning, and completely different for random pruning.



(a) Loss (b) Pruned SVA (c) All Layers (d) SVA evol. (e) All Layers

Figure 12: Pruning results for LSTM. **Top row:** Magnitude pruning. **Bottom row:** random pruning. See Figure 11 for details. Results are quite similar for the LSTM at 25% pruning as the VGG in Figure 11.

top singular vector dynamics, and also performs worse, as in (Frankle et al., 2020). It would not be surprising that random pruning is worse if simply evaluated at the end of training, but masking is applied quite early in training at epoch 4 of 164 long before convergence, so it's striking that the network now fails to learn further even though it is far from convergence. We interpret this as evidence that the mask has somehow cut signal flow between layers, so it is now impossible for the network to learn further, while magnitude pruning and rewinding still allows signals to pass that eventually become important.

C.3 Spectral Dynamics and Linear Mode Connectivity

We come to the final phenomenon that we seek to describe: linear mode connectivity. Linear mode connectivity (LMC) is the property that one can interpolate linearly between two different minima in weight space and every parameter set along that path performs well, which gives the impression that the loss surface of neural networks is somehow convex despite its theoretical nonconvexity. This was first demonstrated in small networks with the same initialization (Nagarajan & Kolter, 2019), then expanded to larger networks and connected to lottery tickets (Frankle et al., 2020; Paul et al., 2022). Entezari et al. (2021) first conjecture that two arbitrary minima show LMC up to permutation, and demonstrate it in simple models. This was expanded to wide models (Ainsworth et al., 2022; Jordan et al., 2022; Qu & Horvath, 2024), and can be proven in various ways (Kuditipudi et al., 2019; Brea et al., 2019; Simsek et al., 2021; Ferbach et al., 2023), but it does not hold for standard models (Qu & Horvath, 2024). LMC has also been exploited for model-averaging and performance gains (Wortsman et al., 2022; Ilharco et al., 2022; Rame et al., 2022). Still despite all of this work, we lack a description for why LMC occurs. In particular: why is there a convex, high dimensional (Yunis et al., 2022) basin that models find shortly in training (Frankle et al., 2020), or after pretraining (Neyshabur et al., 2020; Sadrtdinov et al., 2023)? We do not answer this question in full, but find an interesting view through the singular vectors.

C.3.1 Linear Mode Connectivity Correlates with Top Singular Vector Agreement

As we saw earlier directional convergence of top singular vectors in Figure 10, it suggests the dynamics of those components are more stable, so we might expect mode-connected solutions to share these components. To examine this, we plot agreement between the singular vectors of the weight matrices at either endpoint of branches:

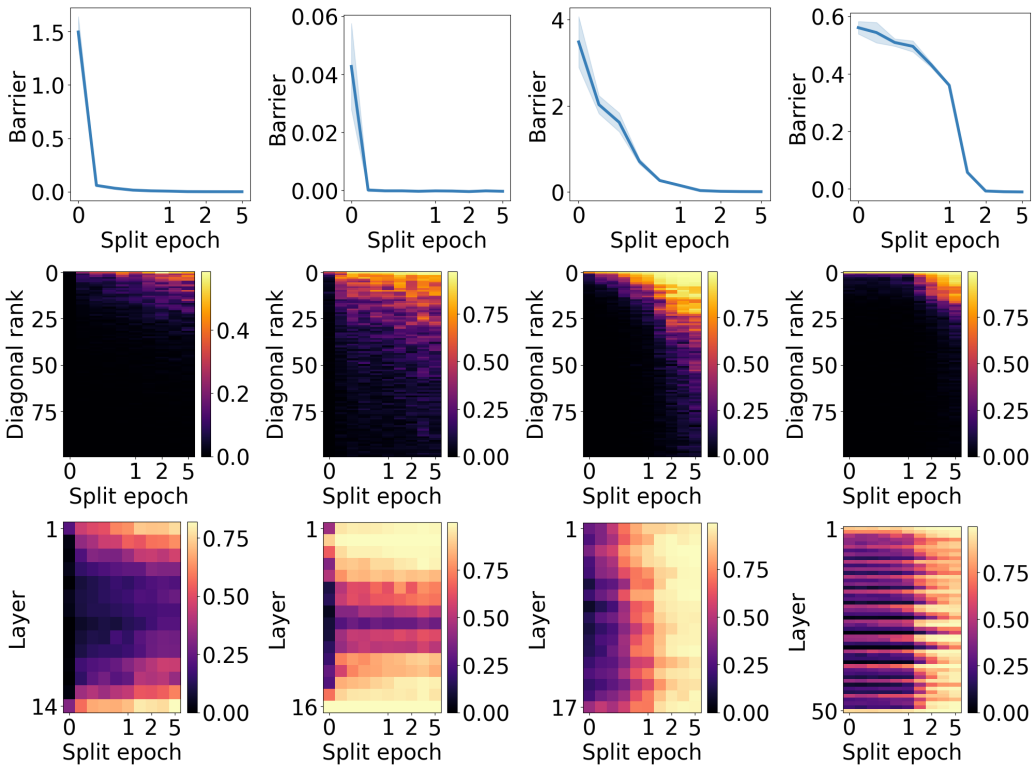
$$W^{(1)}(T) = \sum_{j}^{R} \sigma_j(T) u_j(T) v_j(T)^\top \quad ,$$

$$W^{(2)}(T) = \sum_{k}^{R} \sigma'_k(T) u'_k(T) v'_k(T)^\top \quad ,$$

spawned from the same initialization in training. If the branches are split from an initialization on a trunk trajectory $W(t)$, we call t the split point or epoch. We visualize the diagonal of $|\langle u_j(T)v_j(T)^\top, u'_k(T)v'_k(T)^\top\rangle|_{jk}$ vs. split epoch, where the absolute value is taken to ignore sign flips in SVD computation.

To remind the reader, LMC only occurs after a small amount of training time has passed. Too early and the final models of each branch will show a bump, or barrier, in the loss surface along the linear interpolation (Frankle et al., 2020). To measure this precisely, we use the definition from Neyshabur et al. (2020), which is the maximum deviation from a linear interpolation in the loss, an empirical measure for convexity in this linear direction. When this deviation is 0, we consider the checkpoints to exhibit LMC. Please see Appendix D.10 for details on the calculation. Given evidence in Figure 4 that top components are the most important for prediction, and that top components become stable before training has finished, it is plausible that LMC is connected to the stability of top singular vectors in the later portion of training.

This would mean that checkpoints that do not exhibit the LMC property should not share top singular vectors, while checkpoints that do exhibit the LMC property should share top singular vectors. We see in Figure 13 that this is the case across models and tasks, where the alignment between endpoints is much stronger in top singular vectors. We also see no LMC and poor agreement in top components between branches that have initializations from different trunk trajectories, but with the same split epoch t and the same branch data order in Figure 14. Thus, these top directions are not a unique property of the architecture and data, but rather are dependent on initialization. It is notable that concurrent work (Ito et al., 2024) arrives at a similar conclusion: permutation solvers between optima match top singular vectors. Though the conclusions are similar, their experiments are primarily conducted on smaller scale settings, and only for permutation matching at the end of training. Here we connect these observations to the optimization behavior of networks throughout training.



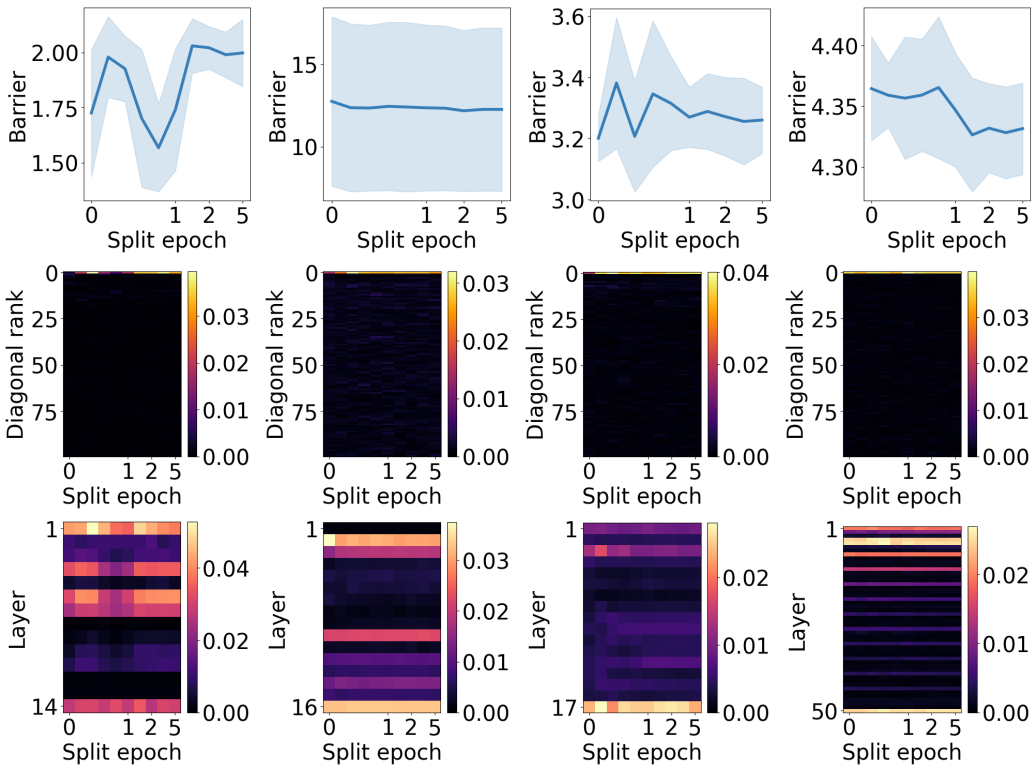
(a) VGG (b) UNet (c) LSTM (d) Transformer

Figure 13: **Top row:** Barrier size vs. split step. **Middle row:** singular vector agreement for a single matrix parameter between branch endpoints that share a common trunk. **Bottom row:** summary statistic for singular vector agreement across layers vs. split step. We see that as models exhibit LMC, they also share top singular vectors.

C.3.2 Perturbing Breaks Linear Mode Connectivity and Singular Vector Agreement Simultaneously

To make the connection between top singular vectors and LMC even tighter, we intervene in the normal training process. If we add random perturbations to destabilize the components that will become the top components long before they have converged, and if singular vector agreement is tied to LMC, we would like to see that final models no longer exhibit the LMC property. Indeed this is the case. In Figure 15, when increasingly large random perturbations are applied, the barrier between final checkpoints increases and the LMC behavior disappears. Please see Appendix D for details. In addition, the previously-strong singular vector agreement disappears simultaneously. Thus it seems this agreement is tied to linear mode connectivity.

We speculate that, due to the results in Figure 4 that show the top half of the SVDs are much more critical for performance, if these components are shared then interpolating will not affect performance much. Rather, interpolation will eliminate the orthogonal bottom components which may only make a minor impact on performance. If however the top components are not shared, then interpolating between two models will remove these components, leading to poor performance in between. Such observations may help in explaining the utility of pretraining (Neyshabur et al., 2020), weight averaging (Rame et al., 2022; Wortsman et al., 2022; Ilharco et al., 2022) or the use of LoRA (Huh et al., 2022) to replace full finetuning.



(a) VGG (b) UNet (c) LSTM (d) Transformer

Figure 14: **Top row:** Barrier size vs. split step. **Middle row:** singular vector agreement for a single matrix parameter between branch endpoints that do not share a common trunk, but do share split time and branch data order. **Bottom row:** summary statistic for singular vector agreement across layers. We see that when branches do not share a common trunk, there is neither LMC nor singular vector agreement, even though the optimization is otherwise the same.

D Experimental Details

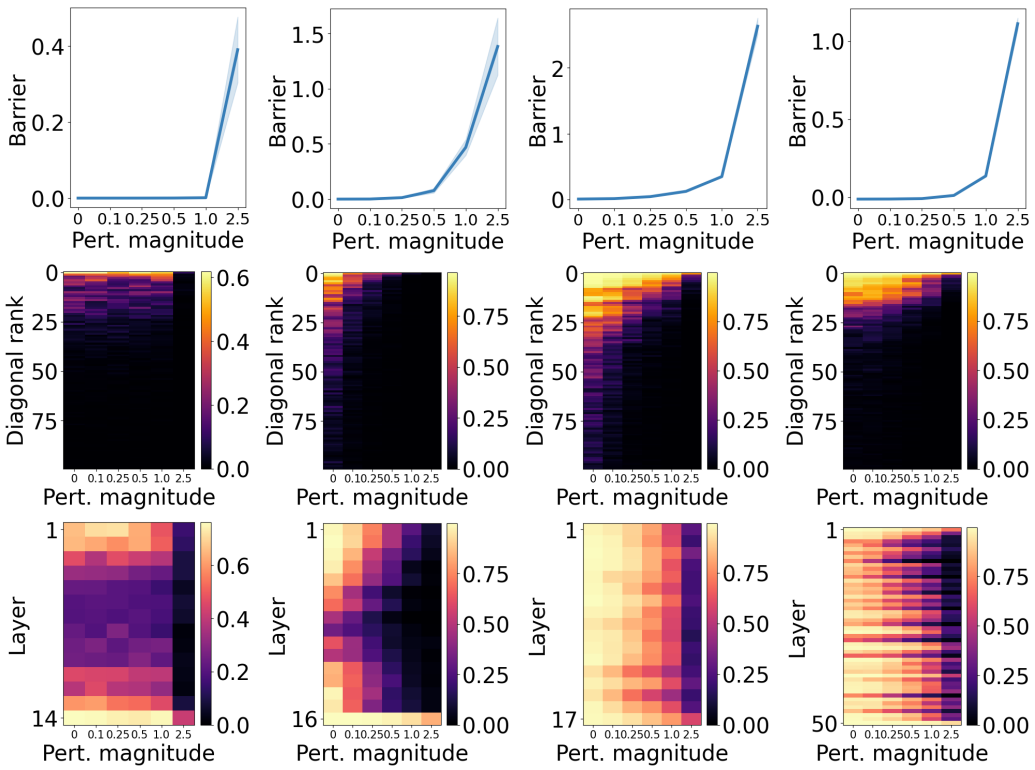
For all experiments, we use 3 random seeds and average all plots over those 3. This is relatively small, but error bars tend to be very tight, and due to the high volume of runs required for this work we lack the resources to run much more.

In order to compute alignment we consider only pairs of layers that directly feed into each other, and ignore the influence of residual connections so as to cut down on the number of comparisons. Specifics on individual architectures are given below.

D.1 Image Classification with VGG

We train a VGG-16 (Simonyan & Zisserman, 2014) on CIFAR-10 (Krizhevsky, 2009) for 164 epochs, following hyperparameters and learning rate schedule in (Frankle et al., 2020), but without data augmentation. For the optimizer we use SGD with batch size 128, initial learning rate 0.1 and momentum of 0.9. We also decay the learning rate 3 times by a factor of 10 at epoch 82, epoch 120, and finally at epoch 160. We also use a minor amount of weight decay with coefficient 0.0001.

VGG-16 uses ReLU activations and batch normalization (Ioffe & Szegedy, 2015), and includes both convolutional and linear layers. For linear layers we simply compute the SVD of the weight matrix. For convolutional layers, the parameters are typically stored as a 4D tensor of shape $(c_{\text{out}}, c_{\text{in}}, h, w)$ for the output channels, input channels, height and width of the filters respectively. As the filters compute a transformation from each position and input channel to an output channel, we compute the SVD of the flattened tensor $(c_{\text{out}}, c_{\text{in}} \cdot h \cdot w)$, which maps all inputs to outputs, similar to Praggastis



(a) VGG (b) UNet (c) LSTM (d) Transformer

Figure 15: **Top row:** Barrier size vs. perturbation magnitude. **Middle row:** singular vector agreement for a single matrix parameter between branch endpoints vs. perturbation magnitude. **Bottom row:** summary statistic for singular vector agreement across layers with perturbation magnitude. We see that whereas without perturbation models would exhibit LMC after training, with increasing perturbations the LMC property disappears simultaneously with the agreement in top singular vectors.

et al. (2022). This is not the SVD of the entire transformation of the feature map to the next feature map, but rather the transformation from a set of adjacent positions to a particular position in the next layer. For the individual SV evolution plot, we use the 12th convolutional layer.

In order to compute alignment of bases between consecutive convolutional layers, $V_{i+1}^{\top} U_i$ we need to match the dimensionality between U_i and V_{i+1}. For convolutional layers we are presented with a question as to how to handle the spatial dimensions h and w as naively the input dimension of the next layer will be a factor of $h \cdot w$ larger dimension. We experimented with multiple cases, including aligning at each spatial position individually or averaging over the alignment at all spatial positions, and eventually settled at aligning the output of one layer to the center spatial input of the next layer. That is, for a 3x3 convolution mapping to a following 3x3 convolution, we compute the alignment only for position (1,1) of the next layer. This seemed reasonable to us as on average the edges of the filters showed poorer alignment overall. For the individual alignment plot, we use the alignment between the 11th and 12th convolutional layers at the center spatial position of the 12th convolutional layer.

D.2 Image Generation with UNets

We train a UNet (Ronneberger et al., 2015) diffusion model (Sohl-Dickstein et al., 2015; Ho et al., 2020) on MNIST (LeCun, 1998) generation. We take model design and hyperparameters from (Wang & Vastola, 2022). In particular we use a 4-layer residual UNet and train with AdamW (Loshchilov & Hutter, 2017) with batch size 128, and learning rate of 0.0003 for 100

epochs. This model uses swish (Ramachandran et al., 2017) activations and a combination of linear and convolutional, as well as transposed convolutional layers.

Computing SVDs and alignment is similar to the image classification case described above, except in the case of the transposed convolutions where an extra transpose of dimensions is needed as parameters are stored with the shape $(c_{\text{in}}, c_{\text{out}}, h, w)$. For the individual SV evolution plot, we use the 3rd convolutional layer. For the alignment plot, we use the alignment between the 3rd and 4th convolutional layers at the center spatial position of the 4th convolutional layer.

D.3 Speech Recognition with LSTMs

We train a bidirectional LSTM (Hochreiter & Schmidhuber, 1997a) for automatic speech recognition on LibriSpeech (Panayotov et al., 2015). We tune for a simple and well-performing hyperparameter setting. We use AdamW (Loshchilov & Hutter, 2017) with batch size 32, learning rate 0.0003 and weight decay 0.1 for 50 epochs. We also use a cosine annealing learning rate schedule from 1 to 0 over the entire 50 epochs.

The LSTM only has matrix parameters and biases, so it is straightforward to compute SVDs of the matrices. For individual SV evolution plots, we plot the 3rd layer input parameter. In the case of alignment, we make a number of connections: first down depth for the input parameters, then connecting the previous input parameter to the current hidden parameter in both directions, then connecting the previous hidden parameter to the current input parameter. In particular the LSTM parameters are stored as a stack of 4 matrices in PyTorch, and we find alignment is highest for the "gate" submatrix, so we choose that for all plots. For the individual layer alignment, we plot alignment between the 3rd and 4th layer input parameters.

D.4 Language Modeling with Transformers

We train a Transformer (Vaswani et al., 2017) language model on Wikitext-103 (Merity et al., 2016). We base hyperparameter choices on the Pythia suite (Biderman et al., 2023), specifically the 160 million parameter configuration with sinusoidal position embeddings, 12 layers, model dimension 768, 12 attention heads per layer, and hidden dimension 768. We use AdamW (Loshchilov & Hutter, 2017) with batch size 256, learning rate 0.0006 and weight decay 0.1. We use a context length of 2048 and clip gradients to a maximum norm of 1. We also use a learning rate schedule with a linear warmup and cosine decay to 10% of the learning rate, like Biderman et al. (2023).

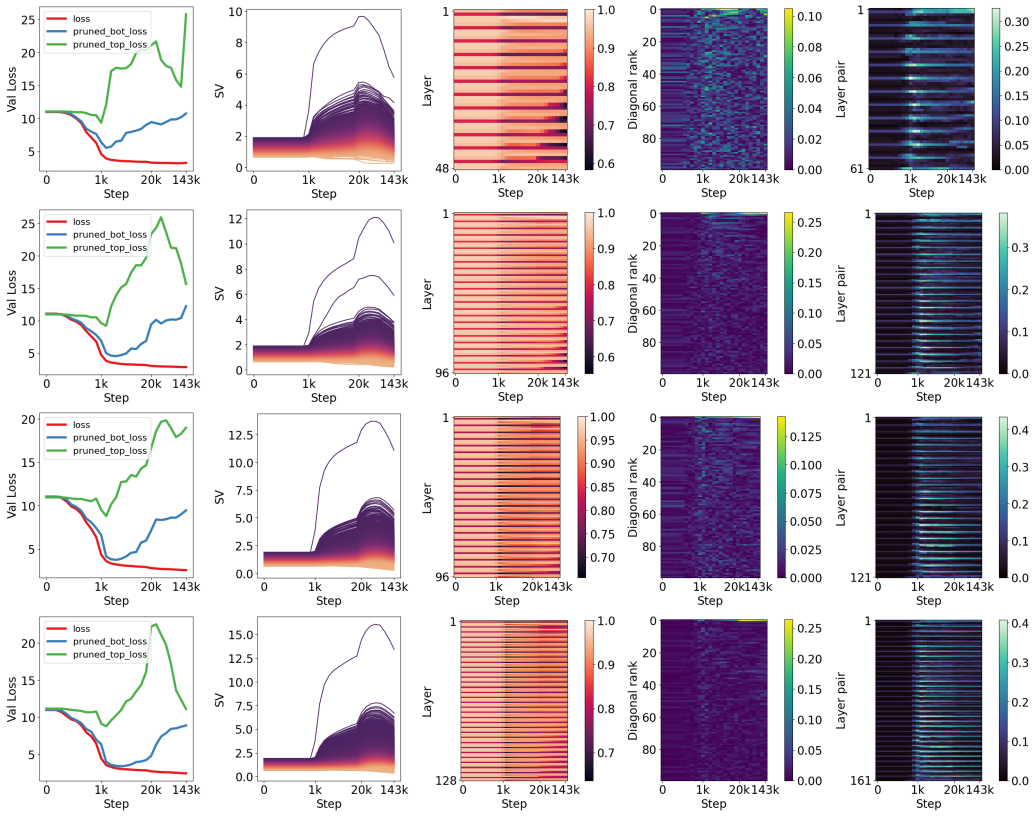
For SVDs, for simplicity we take the SVD of the entire $(3d_{\text{model}}, d_{\text{model}})$ parameter that computes queries, keys and values from the hidden dimension inside the attention layer, without splitting into individual heads. This is reasonable as the splitting is done after the fact internally. We also take the SVD of the output parameters, and linear layers of the MLPs, which are 2 dimensional matrices. For the individual SV evolution plot, we plot the SVs of W_1 of the 8th layer MLP

For alignment, we consider the alignment of W_Q and W_K matrices, W_V and W_O matrices, computing alignment between heads individually then averaging over all heads. We also consider the alignment between W_O and W_1 of the MLP block, between W_1 and W_2 of the MLP block, and between W_2 and the next attention layer. For the individual layer alignment, we plot alignment between W_1 and W_2 of the 8th layer MLP.

D.5 Spectral Dynamics with Scale (Pythia)

Here we apply the perspective developed in Section 4 to larger scale models. As we lack the resources to train these models ourselves, we leverage the Pythia (Biderman et al., 2023) family which provides training trajectories for language models across a range of scales (70m to 12b parameters). We are further constrained to the 2.8b parameter model at the largest due to memory requirements when computing SVDs and alignment.

In Figure 16, we see similar rank dynamics across a variety of scales. We choose to select the 7th layer MLP to compare between models as it is present at all scales. We do see an unequal evolution in singular values, but also a contraction as training proceeds for longer. The difference between scales is not very obvious, but slightly fewer of the singular values evolve to be large in the 2.8b model as opposed to the 410m model, which one can see from the thickness of the light magenta



(a) Val. Loss (b) SVs (c) Eff. Rank (d) Alignment (e) Alignment Score

Figure 16: Spectral dynamics of Pythia suite. From top to bottom we examine the 160m, 410m, 1.4b and 2.8b parameter models. Notably, much less noise appears in the alignment plot with increasing scale. Presumably this could be due to the fact that larger dimensional vectors have higher probability to be orthogonal, which may play a role in making optimization easier. We see stronger alignment score (Eqn. 4) in all layers in the larger model, perhaps because of that cleaner signal.

color. The lack of alignment except for the top rank is quite consistent with earlier observations, and such alignment happens much later for the largest model.

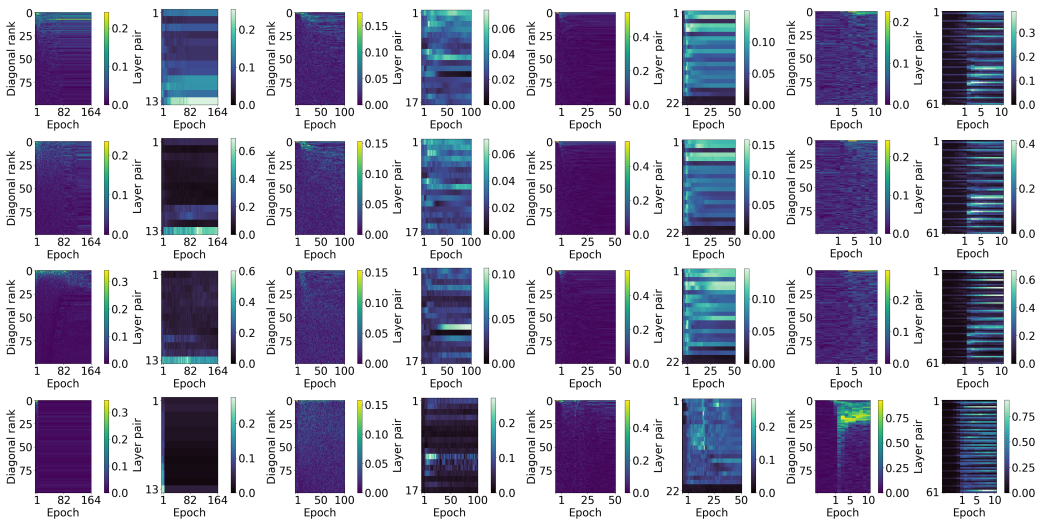
D.6 Weight Decay Experiments

All tasks are trained in exactly the same fashion as mentioned previously, with increasing weight decay in the set $\{0, 0.0001, 0.001, 0.01, 0.1, 1.0, 10.0\}$. For ease of presentation we consider a subset of settings across tasks. In Figure 18 we include trained model performance and pruned model performance to show that, even with high levels of weight decay, models do not entirely break down. More so, the approximation of the pruned model to the full model gets better with higher weight decay.

D.7 Grokking Experiments

For the Trasnformer, we mostly follow the settings and architecture of Nanda et al. (2023), except we use sinusoidal positional encodings instead of learned.

For the slingshot case we follow hyperparameter settings in Thilak et al. (2022), Appendix B except with the 1-layer architecture from Nanda et al. (2023) instead of the 2-layer architecture specified. W perform addition modulo 97. The original grokking plot in Thilak et al. (2022) appears much more dramatic as it log-scales the x-axis, which we do not do here for clarity.



(a) VGG (b) UNet (c) LSTM (d) Transformer

Figure 17: Diagonal of alignment for a single pair over time (Eqn. 3) and alignment metric across pairs of matrices over time (Eqn. 4) where the y-axis represents depth. From top to bottom, for VGG we use coefficients $\{0, 0.001, 0.01, 0.1\}$, while for other networks we use coefficients $\{0, 0.1, 1, 10\}$. We see that the maximum alignment magnitude is higher with large weight decay, and in particular, the Transformer has the strongest alignment even when nonlinearities separate the MLP layers.

In the case of the deep MLP, we follow Fan et al. (2024), where we use a 12-layer MLP with ReLU activations and width 400, trained on MSE loss on MNIST (LeCun, 1998). We use 2000 examples, a batch size of 100, weight decay 0.01, and initialization scale 8 (Liu et al., 2023).

D.8 Random Label Experiments

We train a 4-layer MLP on CIFAR10 (Krizhevsky, 2009) with either completely random labels, or the true labels. We use SGD with momentum of 0.9 and constant learning rate of 0.001, and train for 300 epochs to see the entire trend of training. The major difference to the setting of Zhang et al. (2021) is the use of a constant learning rate, as their use of a learning rate schedule might conflate the results.

For the VGG case, we follow our previous hyperparameters, except we leave out weight decay and learning rate scheduling, instead using a constant learning rate of 0.01.

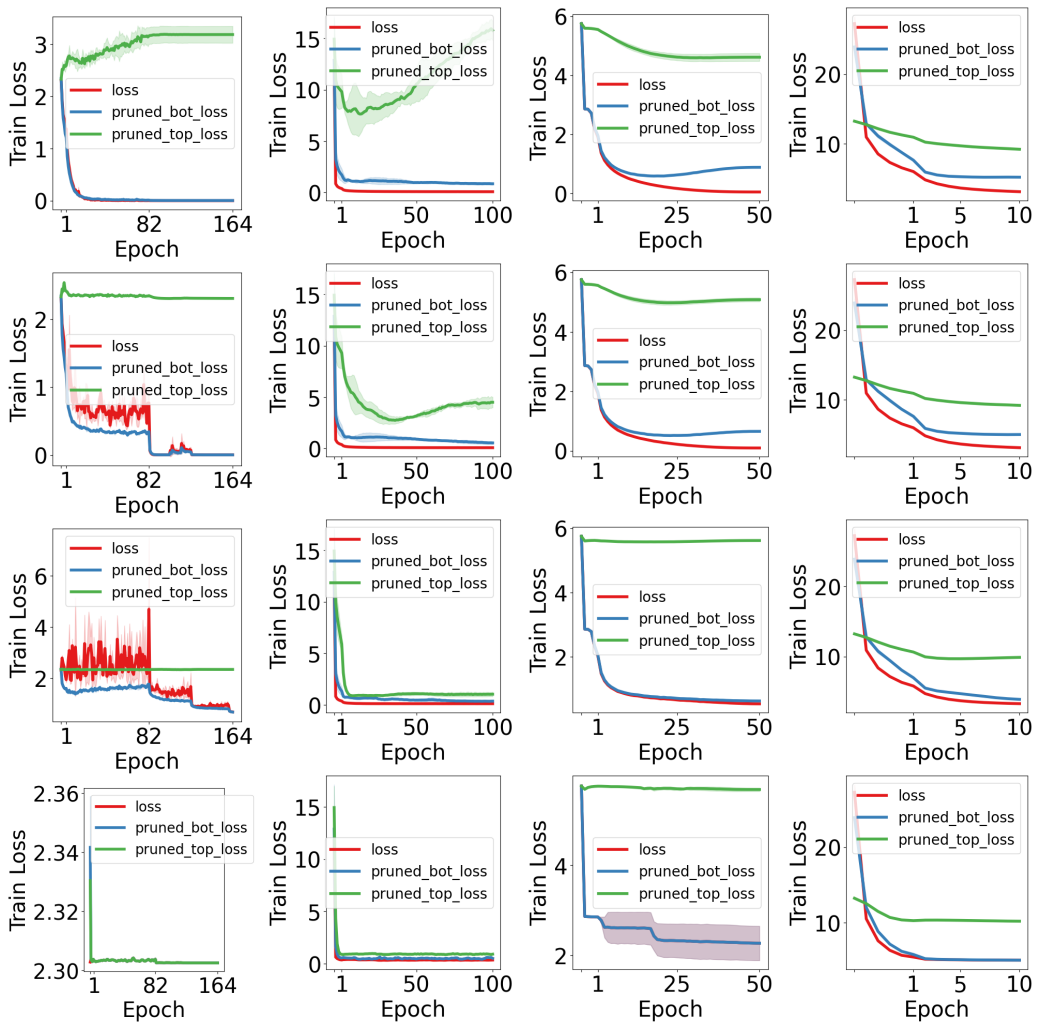
For the LSTM case, we follow our previous hyperparameters, and extend the training budget to 200 epochs allow for the random label setting to train longer. In this case, our network does not have sufficient capacity to memorize the data completely.

D.9 Magnitude Pruning Experiments

We use the same VGG setup as described previously. In this case we train til the end, then compute a global magnitude mask. To do this we flatten all linear and convolutional weights into a single vector, except for the last linear layer, and sort by magnitude. Then we keep the top 5% of weights globally, and reshape back to the layerwise masks. This results in different sparsity levels for different layers, so when generating the random masks, we use the per-layer sparsities that resulted from the global magnitude mask.

To retrain the network, we rewind to epoch 4, then continue training with the mask, always setting other weights and their gradients to 0. We average all results over 3 random seeds.

For the LSTM we follow exactly the same procedure, except our mask only reaches a level of 25% sparsity, due to large performance degradations past that.



(a) VGG (b) UNet (c) LSTM (d) Transformer

Figure 18: Training loss over time, where the rows use differing amounts of weight decay. From top to bottom, for VGG we use coefficients $\{0, 0.001, 0.01, 0.1\}$, while for other networks we use coefficients $\{0, 0.1, 1, 10\}$. We see that it is still possible to achieve low training loss under high weight decay, and as we increase the amount of weight decay, the gap between pruned and unpruned parameters closes, lending support to the idea that the parameters become lower rank.

D.10 LMC Experiments

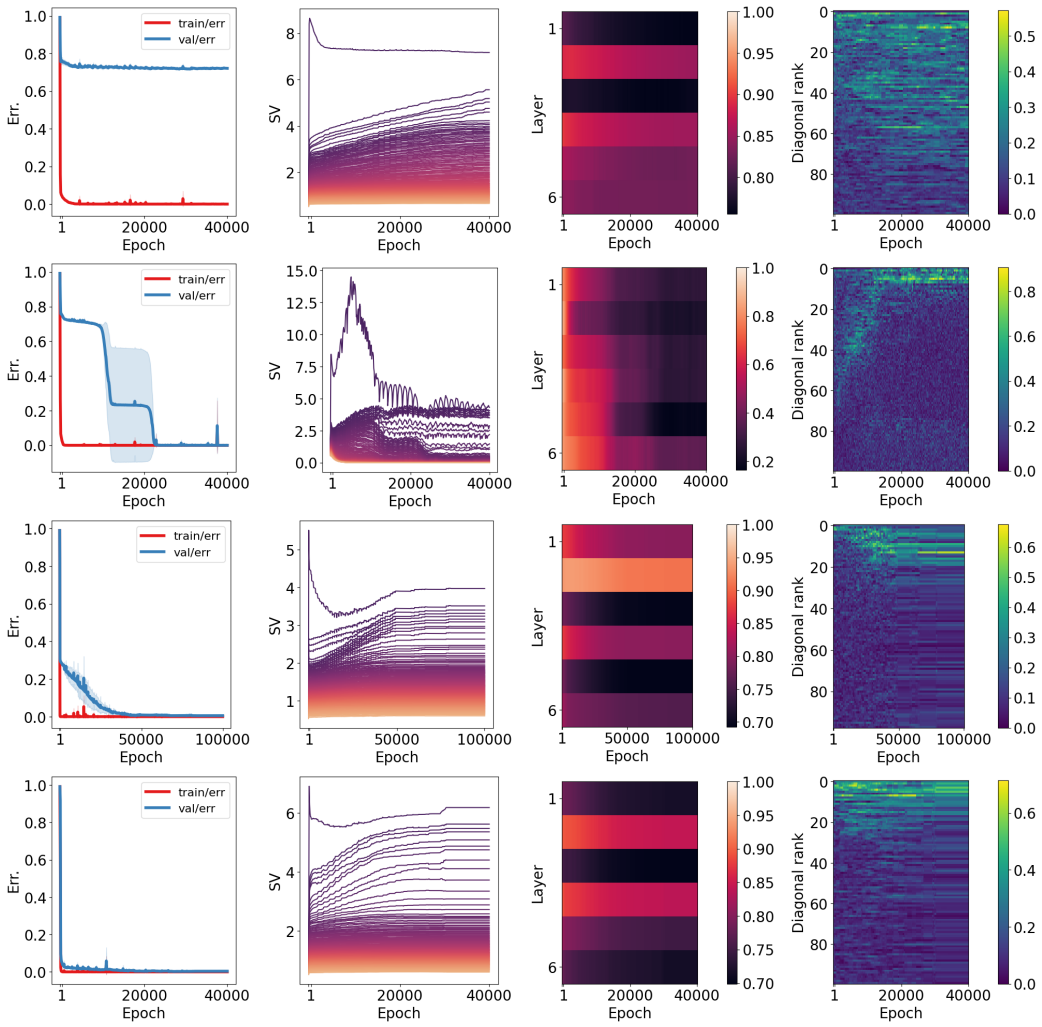
We save 5 evenly-spaced checkpoints in the first epoch, as well as at the end of the next 4 epochs for 10 intializations in total. We train 3 trunks, and split 3 branches from each trunk for a total of 9 branches which we average all plots over.

Following Neyshabur et al. (2020), we compute the barrier between checkpoints as follows: given $W^{(1)}(T)$ and $W^{(2)}(T)$ that were branched from $W(t)$ we compute

$$b(t) = (\max_{\alpha \in [0,1]} \mathcal{L}((1-\alpha)W^{(1)}(T) + \alpha W^{(2)}(T)) - ((1-\alpha)\mathcal{L}(W^{(1)}(T)) + \alpha \mathcal{L}(W^{(2)}(T))) \quad (6)$$

when this quantity is 0, we consider the checkpoints to exhibit LMC.

We recompute batch normalization parameters after interpolating for VGG-16, and group normalization parameters for the UNet, as these do not necessarily interpolate well (Frankle et al., 2020). We also compute singular vector agreement for the same parameter between either branch endpoint.



(a) Error (b) SV Evolution (c) Effective Rank (d) Alignment

Figure 19: **Grokking and Spectral Dynamics in Modular addition. Top row:** 30% data and no weight decay. **2nd row:** 30% data and weight decay 1.0 (grokking), using hyperparameters from Nanda et al. (2023). **3rd row:** 70% data with no weight decay (slingshot), using hyperparameters from Thilak et al. (2022). **Bottom row:** 90% data and no weight decay. **1st column:** Training and validation error. **2nd column:** Singular value evolution is visualized for the first attention parameter, where each line represents a single singular value and the color represents the rank. **3rd column:** Effective rank of all layers (Eqn. 1). **4th column:** Alignment (Eqn. 3) between the embedding and the first attention parameter is also visualized, where the y-axis corresponds to index i of the diagonal. One can see that grokking co-occurs with low-rank weights. In addition, there is an alignment that begins early in training that evolves up the diagonal. Without weight decay and with less data, neither grokking nor the other phenomena occur during the entire training budget, but using more data, even without weight decay, leads to low-rank solutions from the beginning of training. The slingshot case follows a similar trend, though the validation loss is gradually fit. Across cases with good generalization, parameters are lower rank, and alignment is also more prevalent in the top ranks.

To plot the singular vector (dis)agreement and LMC between different modes, we make 11 evenly spaced measurements interpolating between branch endpoints that had the same split epoch, and the same branch seed, but different trunk initializations.

D.11 Perturbed LMC Experiments

We perturb all weights W after the point of dynamics stability where we expect to see LMC at the end of training (epoch 4 is sufficiently late in all cases) using randomly sampled normal perturbations $\epsilon \sim \mathcal{N}(0, I)$ with $\|\epsilon\| = \eta\|W\|$ where $\eta \in \{0.0, 0.1, 0.25, 0.5, 1.0, 2.5\}$. We do not perturb the output layer, as this has a very substantial effect on the optimization. We also do not perturb the input layer for the Transformer as it is too computationally expensive for our resources.

E Limitations

There are a few key limitations to our study. As mentioned, we lack the computational resources to run more than 3 random seeds per experiment, though we do find error bars to be quite tight in general (except for the generalization epoch in the grokking experiments). In addition, as discussed we ignore 1D parameters in the neural networks, which may be particularly crucial (especially normalization). In addition, due to computational constraints we do not consider alignment of layers across residual connections as this quickly becomes combinatorial in depth, thus there may be other interesting interactions that we do not observe. Finally, due to computational constraints we are unable to investigate results on larger models than the 12 layer Transformer, which may have different behavior.

F Compute Resources

All experiments are performed on an internal cluster with on the order of 100 NVIDIA 2080ti GPUs or newer. All experiments run on a single GPU in less than 8 hours, though it is extremely helpful to parallelize across machines. We estimate that end-to-end it might take a few days on these resources to rerun all of the experiments in this paper. Additionally, the storage requirements for all of the checkpoints will take on the order of 5 terabytes.

G Code Sources

We use PyTorch (Paszke et al., 2019) and NumPy (Harris et al., 2020) for all experiments and Weights & Biases (Biewald, 2020) for experiment tracking. We make plots with Matplotlib (Hunter, 2007) and Seaborn (Waskom, 2021). We also use HuggingFace Datasets (Lhoest et al., 2021) for Wikitext-103 (Merity et al., 2016).