Deconstructing Lottery Tickets: Zeros, Signs, and the Supermask

Hattie Zhou
Uber
hattie@uber.com

Janice Lan
Uber AI Labs
janlan@uber.com

Rosanne Liu
Uber AI Labs
rosanne@uber.com

Jason Yosinski
Uber AI Labs
yosinski@uber.com

Abstract

The recent "Lottery Ticket Hypothesis" paper by Frankle & Carbin showed that a simple approach to creating sparse networks (keep the large weights) results in models that are trainable from scratch, but only when starting from the same initial weights. The performance of these networks often exceeds the performance of the non-sparse base model, but for reasons that were not well understood. In this paper we study the three critical components of the Lottery Ticket (LT) algorithm, showing that each may be varied significantly without impacting the overall results. Ablating these factors leads to new insights for why LT networks perform as well as they do. We show why setting weights to zero is important, how signs are all you need to make the re-initialized network train, and why masking behaves like training. Finally, we discover the existence of Supermasks, or masks that can be applied to an untrained, randomly initialized network to produce a model with performance far better than chance (86% on MNIST, 41% on CIFAR-10).

1 Introduction

Many neural networks are over-parameterized [1, 2], enabling compression of each layer [2, 17, 5] or of the entire network [11]. Some compression approaches enable more efficient computation by pruning parameters, by factorizing matrices, or via other tricks [5, 7, 10, 13–19]. Unfortunately, although sparse networks created via pruning often work well, training sparse networks directly often fails, with the resulting networks underperforming their dense counterparts [13, 5].

A recent work by Frankle & Carbin [3] was thus surprising to many researchers when it presented a simple algorithm for finding sparse subnetworks within larger networks that *are* trainable. Their approach for finding these sparse, performant networks is as follows: after training a network, set all weights smaller than some threshold to zero, pruning them (similarly to other pruning approaches [6, 5, 12]), rewind the rest of the weights to their initial configuration, and then retrain the network from this starting configuration but with the zero weights frozen (not trained). Using this approach, they obtained two intriguing results.

First, they showed that the pruned networks performed well. Aggressively pruned networks (with 99.5 percent to 95 percent of weights pruned) showed no drop in performance compared to the much larger, unpruned network. Moreover, networks only moderately pruned (with 50 percent to 90 percent of weights pruned) often outperformed their unpruned counterparts.

Second, compelling though these results were, the characteristics of the remaining network structure and weights were just as interesting. Normally, if you take a trained network, re-initialize it with random weights, and then re-train it, its performance will be about the same as before. But with their pruned networks, they showed that this property does not hold. The network trains well only if it is rewound to its initial state, including the specific initial weights that were used. Reinitializing the same network topology with new weights causes it to train poorly. As pointed out in [3], it

Preprint. Work in progress.

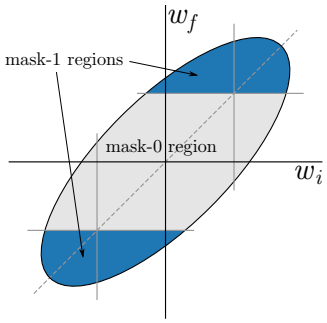

Figure 1: Different mask criteria can be thought of as segmenting the 2D (w_i = initial weight value, w_f = final weight value) space into regions corresponding to mask values of 1 vs 0. The ellipse represents in cartoon form the area occupied by the positively correlated initial and final weights from a given layer. The mask criterion shown, identified by two horizontal lines that separate the whole region into mask-1 (blue) regions and mask-0 (grey) regions, corresponds to a large_final criterion used in [3]: weights with large final magnitude are kept, and weights with final values near zero are pruned. Note that this criterion ignores the initial magnitude of the weight.

appears that the specific combination of pruning mask and weights underlying the mask form a lucky, trainable sub-network found within the larger network, or, as named by the original study, a lucky winning "Lottery Ticket," or LT. Their associated "Lottery Ticket Hypothesis" states that inside the original, large network, a sub-network together with its initialization forms the winning ticket that results in a more efficient network.

Although [3] clearly demonstrated lottery tickets to be effective, as is often the case with intriguing research, the results raised as many questions as they answered, and many of the underlying mechanics were not yet well understood. Remaining questions include: what about LT networks causes them to show better performance? Why are the mask and the initial set of weights so tightly coupled, such that re-initializing the network makes it less trainable? Why does simply selecting large weights constitute an effective criterion for choosing a mask? Would other criteria for mask selection work too?

Our paper proposes explanations behind these mechanisms, finds curious quirks of these subnetworks, introduces competitive variants of the pruning algorithm, and derives a surprising by-product: the Supermask.

We begin by briefly describing the lottery ticket algorithm:

0. Initialize a mask m to all ones. Randomly initialize the parameters w of a network $f(x; w \odot m)$
1. Train the parameters w of the network $f(x; w \odot m)$ to completion. Denote the initial weights before training w_i and the final weights after training w_f.
2. *Mask Criterion.* Use the mask criterion $M(w_i, w_f)$ to produce a masking score for each currently unmasked weight. Rank the weights in each layer by their scores, set the mask value for the top $p\%$ to 1, the bottom $(100 - p)\%$ to 0, breaking ties randomly. Here p may vary by layer, and we follow the ratios chosen in [3], summarized in Table S1. In [3] the mask selected weights with large final value corresponding to $M(w_i, w_f) = |w_f|$. In Section 2 we consider other mask criteria.
3. *Mask-1 Action.* Take some action with the weights with mask value 1. In [3] these weights were reset to their initial values and marked for training in the next round. We consider this and other mask-1 actions in Section 3.
4. *Mask-0 Action.* Take some action with the weights with mask value 0. In [3] these weights were pruned: set to 0 and frozen during any subsequent training. We consider this and other mask-0 actions in Section 4.
5. Repeat from 1 if performing iterative pruning.

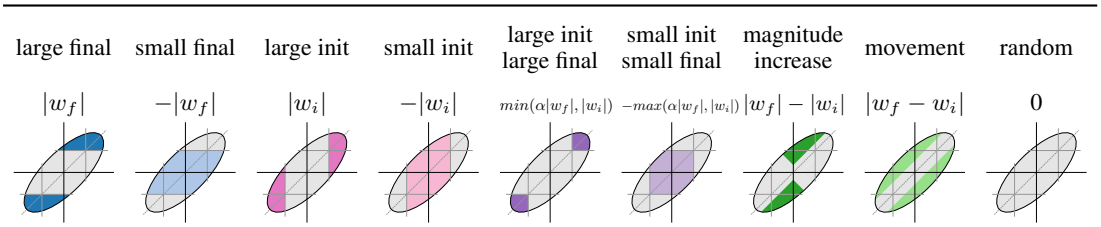

Figure 2: Mask criteria studied in this section, starting with large_final that was used in [3]. Names we use to refer to the various methods are given along with the formula that projects each (w_i, w_f) pair to a score. Weights with the largest scores (colored regions) are kept, and weights with the smallest scores (gray regions) are pruned. The x axis in each small figure is w_i and the y axis is w_f. In two methods, α is adjusted as needed to align percentiles between w_i and w_f. When masks are created, ties are broken randomly, so the random method just assigns a score of 0 to every weight.

In this paper we perform ablation studies along the above three dimensions of variability, considering alternate mask criteria (Section 2), alternate mask-1 actions (Section 3), and alternate mask-0 actions (Section 4). These studies in aggregate reveal new insights for why lottery ticket networks work as they do. Along the way we also discover the existence of Supermasks—masks that produce above-chance performance when applied to untrained networks (Section 5).

2 Mask criteria

We begin our investigation with a study of different *Mask Criteria*, or functions that decide which weights to keep vs. prune. In this paper, we define the mask for each individual weight as a function of the weight's values both at initialization and after training: $M(w_i, w_f)$. We can visualize this function as a set of decision boundaries in a 2D space as shown in Figure 1. In [3], the mask criterion simply keeps weights with large final value; we refer to this as the large_final mask, $M(w_i, w_f) = |w_f|$.

In addition to large_final, we also experimented with the inverse, small_final, versions that evaluate weights based on their initial magnitudes instead, large_init and small_init, versions that select for both, large_init_large_final and small_init_small_final, two mask criteria that evaluate how far weights moved, magnitude_increase and movement, and a control, random, that chooses masks randomly. These nine masks are depicted along with their associated equations in Figure 2. Note that magnitude_increase keeps weights whose magnitude increased the most, whereas movement keeps weights that changed the most during training, the main difference between the two being that those weights that change sign are more likely to be kept in the movement criterion than the magnitude_increase criterion.

In this section and throughout the remainder of the paper, we follow the experimental framework from [3] and perform iterative pruning experiments on a fully-connected network (FC) trained on MNIST [9] and on three convolutional networks (Conv2, Conv4, and Conv6) trained on CIFAR-10 [8]. For more achitecture and training details, see Section S1.

Results of these pruning experiments are shown in Figure 3. In this figure along with all others in this paper, the line depicts the mean over five runs, and the band (if shown) depicts the min and max obtained over five runs. In some cases the band is omitted for visual clarity. As can be seen, the magnitude_increase criterion turns out to work just as well as the large_final criterion, and in some cases significantly better. As a baseline, we also show results on a random pruning criterion that simply chooses a random mask with the desired pruning percentage. Note that the first six criteria out of the eight form three opposing pairs; in each case, we see when one member of the pair performs better than the random baseline, the opposing member performs worse than it.

The magnitude_increase criterion turns out to work just as well as the large_final criterion, and in some cases significantly better. Results of all criteria are shown in Figure 3 for the four networks (FC, Conv2, Conv4, Conv6). As a baseline, we also show results on a random pruning criterion that simply chooses a random mask with the desired pruning percentage. Note that the first six criteria out

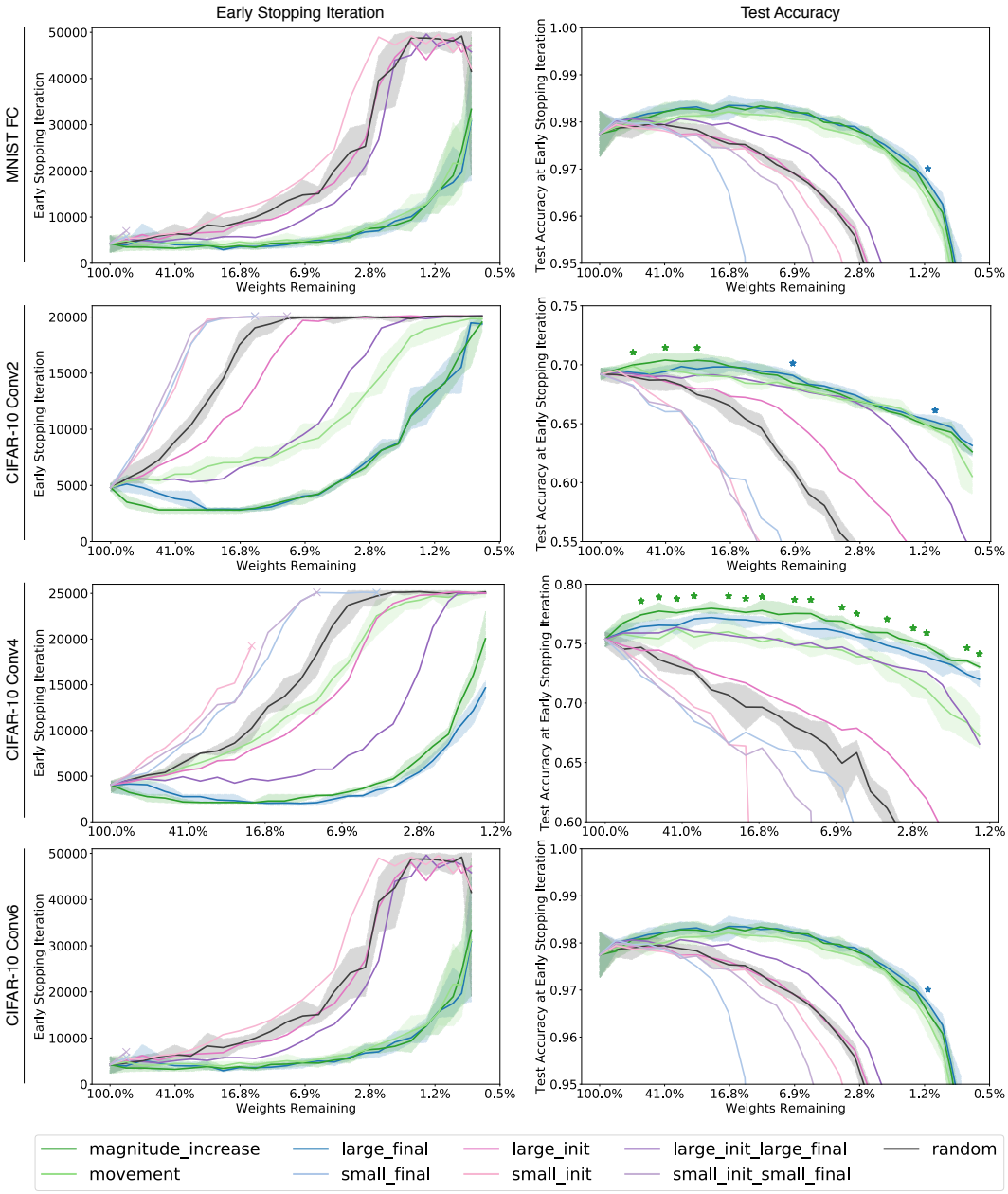

Figure 3: Performance of different mask criteria for four networks at various pruning rates. We show early stopping iteration on the left and test accuracy on the right. Each line is a different mask criteria, with bands around magnitude_increase, large_final, movement, and random depicting the min and max over 5 runs. Stars represent points that are significantly above all other lines at a $p = 0.05$ level. large_final and magnitude_increase show the best convergence speed and accuracy, with magnitude_increase having slightly higher accuracy in Conv2 and Conv4. As expected, criteria using small weight values consistently perform worse than random.

of the eight form three opposing pairs; in each case, we see when one member of the pair performs better than the random baseline, the opposing member performs worse than it.

The conclusion so far is that large_final is a quite competitive mask criterion, although in some cases magnitude_increase has an edge. This partially answers our question about the efficacy of different

mask criteria. Still unanswered: why either of these two front-running criteria should work well in the first place. We'll have to wait until Section 4 to uncover those details.

3 Mask-1 actions: show me a sign

Now that we have explored various ways of choosing which weights to keep and prune, we will consider what values to set for the kept weights. In particular, we want to explore an interesting observation in [3] which showed that the pruned, skeletal LT networks train well when you rewind to its original initialization, but degrades in performance when you randomly reinitialize the network.

Why does reinitialization cause LT networks to train poorly? Which components of the original initialization are important? We evaluate a number of variants of reinitialization to investigate:

- "Reinit" experiments: reinitialize kept weights based on the original initialization distribution
- "Reshuffle" experiments: reinitialize while respecting the original distribution of remaining weights in that layer by reshuffling the kept weights' initial values
- "Constant" experiments: reinitialize by setting mask-1 weight values to a positive or negative constant, with the constant set to be the standard deviation of each layer's original initialization. Thus every weight on a layer becomes one of three values: $-\alpha$, 0, or α.

All of the reinitialization experiments are based on the same original networks and use the large_final mask criterion with iterative pruning. We include the original LT network ("rewind, large final") and the randomly pruned network ("random") as baselines for comparison.

We find that none of these three variants alone are able to train as well as the original LT network, shown as dashed lines in Figure 4. However, all three variants work better when we ensure that the new values of the kept weights are of the same sign as their original initial values. These are shown as solid color lines in Figure 4. Clearly, the common factor in all working variants including the original rewind action is the sign. As long as you keep the sign, reinitialization is not a deal breaker; in fact, even setting all kept weights to a constant value consistently performs well!

4 Mask-0 actions: masking is training

What should we do with weights that are pruned? This question may seem trivial, as deleting them (equivalently: setting them to zero) is the standard practice. The term "pruning" implies the dropping of connections by setting weights to zero, and these weights are thought of as unimportant. However, if pruned weights are truly unimportant, we would expect that setting them to any other sensible values should lead to a similarly performing network. This turns out to not be the case. We show in this section that zero values actually matter, alternative freezing approach results in better performing networks, and masking can be viewed as a way of training.

Typical network pruning procedures perform two actions on pruned weights: set them to zero, and freeze them in subsequent training. It is unclear which of these two components leads to the increased performance in LT networks. To separate the two factors, we run a simple experiment: we reproduce the LT iterative pruning experiments in which network weights are masked out in alternating train/mask/rewind cycles, but try an additional treatment: freeze masked weights at their initial values instead of at zero. If zero isn't special, both should perform similarly.

Figure 5 shows the results for this experiment. We find that networks perform significantly better when weights are frozen specifically at zero than at random initial values. For these networks masked via the LT large_final criterion, zero would seem to be a particularly good value to set weights to when they had small final values. At high levels of pruning, freezing at the initial values may perform better, which makes sense since having a large number of zeros means having lots of dead connections.

So why does zero work better than initial values? One hypothesis is that the mask criterion we use *tends to mask to zero those weights that were headed toward zero anyway*.

To test out this hypothesis, let's consider a new approach to freezing. We run another experiment interpolated between the previous two: for any weight to be frozen, we freeze it to zero if it moved toward zero over the course of training, and we freeze it at its random initial value if it moved away

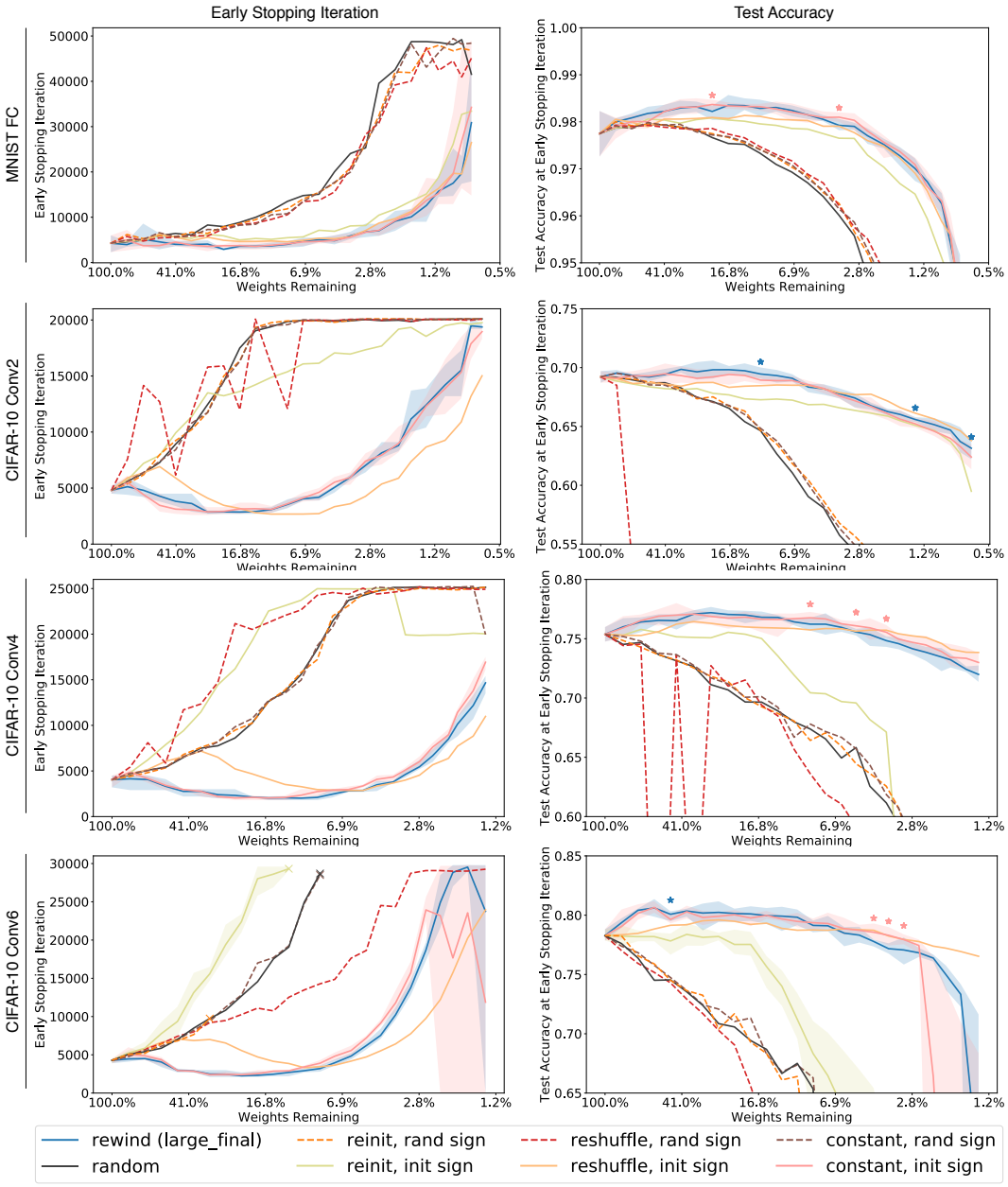

Figure 4: The effects of various 1-actions for the four networks and various pruning rates. Dotted lines represent the three described methods, and solid lines are those three except with each weight having the same sign as its original initialization. Shaded bands around notable runs depict the min and max over 5 runs. Stars represent points that are significantly above all other lines at a $p = 0.05$ level. We also include the original rewinding method and random reinitialization as baselines. "Reshuffle, init sign" and "constant, init sign" perform similarly to the "rewind" baseline.

from zero. Results are shown in Figure 5. We see that performance is indeed better with this treatment than when freezing all pruned weights to zero or all to initial values. This supports our hypothesis that the benefit derived from freezing values to zero comes from the fact that those values were moving toward zero anyway.

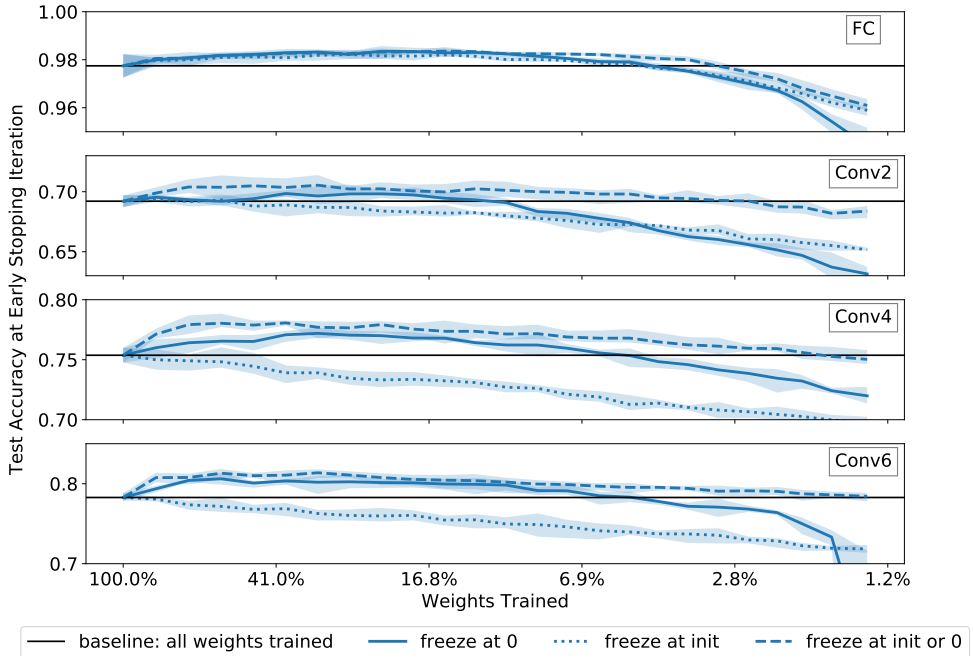

Figure 5: Performance of pruned networks using different treatments of pruned weights. Horizontal black lines represent the performance of the original, unpruned networks, averaged over five runs. Solid blue lines represent networks trained using the LT algorithm, which sets pruned weights to zero and freezes them. Dotted blue lines represent networks where pruned weights are frozen to their initial values. Dashed blue lines represent networks trained using the new proposed scheme for pruned weights: freeze pruned weights at zero if they decreased in magnitude by the end of training, otherwise freeze them at their initialization values.

Figure S1 illustrates why the large_final criterion biases weights that were moving toward zero during training toward zero in the mask, effectively pushing them further in the direction they were headed. Additional variants of this experiment can be seen in Supplementary Information Section S2.

5 Supermasks

The hypothesis above suggests that for certain mask criteria, like large_final, that masking is training: *the masking operation tends to move weights in the direction they would have moved during training*. If so, just how powerful is this training operation? To answer this question, we can start all the way from the beginning—not training the network at all, but simply applying a mask to the randomly initialized network.

It turns out that with a well-chosen mask, an untrained network can already attain a test accuracy far better than chance. This might come as a surprise, because if you use a randomly initialized and untrained network to, say, classify images of handwritten digits from the MNIST dataset, you would expect accuracy to be no better than chance (about 10%). But now imagine you multiply the network weights by a mask containing only zeros and ones. In this instance, weights are either unchanged or deleted entirely, but the resulting network now achieves nearly 40 percent accuracy at the task! This is strange, but it is exactly what we observe with masks created using the large_final criterion.

As depicted in Figure 6, in randomly-initialized networks and randomly-initialized networks with random masks, neither weights nor the mask contain any information about the labels, so accuracy cannot reliably be better than chance. In randomly-initialized networks with large_final masks, it is

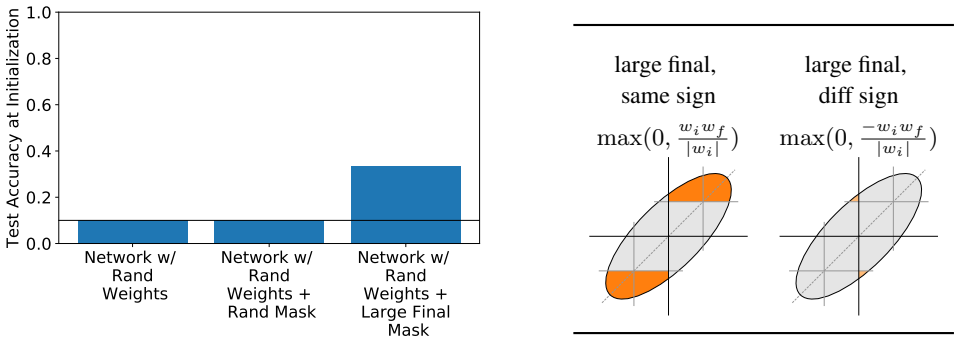

Figure 6: **(left)** Untrained networks perform at chance (10% accuracy, for example, on the MNIST dataset as depicted), if they are randomly initialized, or randomly initialized and randomly masked. However, applying the large_final mask improves the network accuracy beyond the chance level. **(right)** The large_final_same_sign mask criterion (left) that tends to produce the best Supermasks. In contrast to the large_final mask in Figure 1, this criterion masks out the quadrants where the sign of w_i and w_f differ. We include large_final_diff_sign (right) as a control.

not entirely implausible to have better-than-chance performance since the masks are derived from the training process. The large improvement in performance is still surprising, however, since the only transmission of information from the training back to the initial network is via a zero-one mask based on a simple criterion. We call these masks that can produce better-than-chance accuracy without training of the underlying weights "Supermasks".

We now turn our attention to finding better Supermasks. First, we simply gather all masks instantiated in the process of creating the networks shown in Figure 2 and evaluate them without training the network. Next, compelled by the demonstration in Section 3 of the importance of signs and of keeping large weights, we define a new large_final_same_sign mask criterion that selects for weights with large final magnitudes that also maintained the same sign by the end of training. This criterion is depicted in Figure 6. Also included as a control is the large_final_diff_sign. Performances of Supermasks produced by all 10 criteria are included in Figure 7, compared with two baselines: networks untrained and unmasked (untrained_baseline) and networks fully trained (trained_baseline).

By using this simple mask criterion of large_final_same_sign, we can create networks that obtain a remarkable 80% test accuracy on MNIST and 24% on CIFAR-10 without training. Another curious observation is that if we apply the mask to a signed constant (as described in Section 3) rather than the actual initial weights, we can produce even higher test accuracy of up to 86% on MNIST and 41% on CIFAR-10! Detailed results across network architectures, pruning percentages, and these two treaments, are shown in Figure 7. The criterion large_final_same_sign outperforms other criteria consistently by a large margin. And the performance boost from setting mask-1 weights from original initialization value to a same-signed constant is evident, especially for CIFAR-10.

We find it fascinating that these Supermasks exist and can be found via such simple criteria. As an aside, they also present a method for network compression, since we only need to save a binary mask and a single random seed to reconstruct the full weights of the network.

5.1 Learning to find the best Supermasks

Now that we know Supermasks exist, and those derived from simple heuristics work remarkably well, we might wonder how far we can push the performance of Supermasks for a given network. One can search, in the search space of all 2^n possible masks, where n is the number of parameters in that network.

We can also try learning it with regular optimizers. To do that, we create a trainable mask variable for each layer while freezing all original parameters for that layer at their random initialization values. For an original weight tensor w and a mask tensor m of the same shape, we have as the effective weight $w' = w_i \odot g(m)$, where w_i denotes the initial values weights are frozen at, $\odot$ is element-wise multiplication and g is a point-wise function that transform a matrix of continuous values into binary

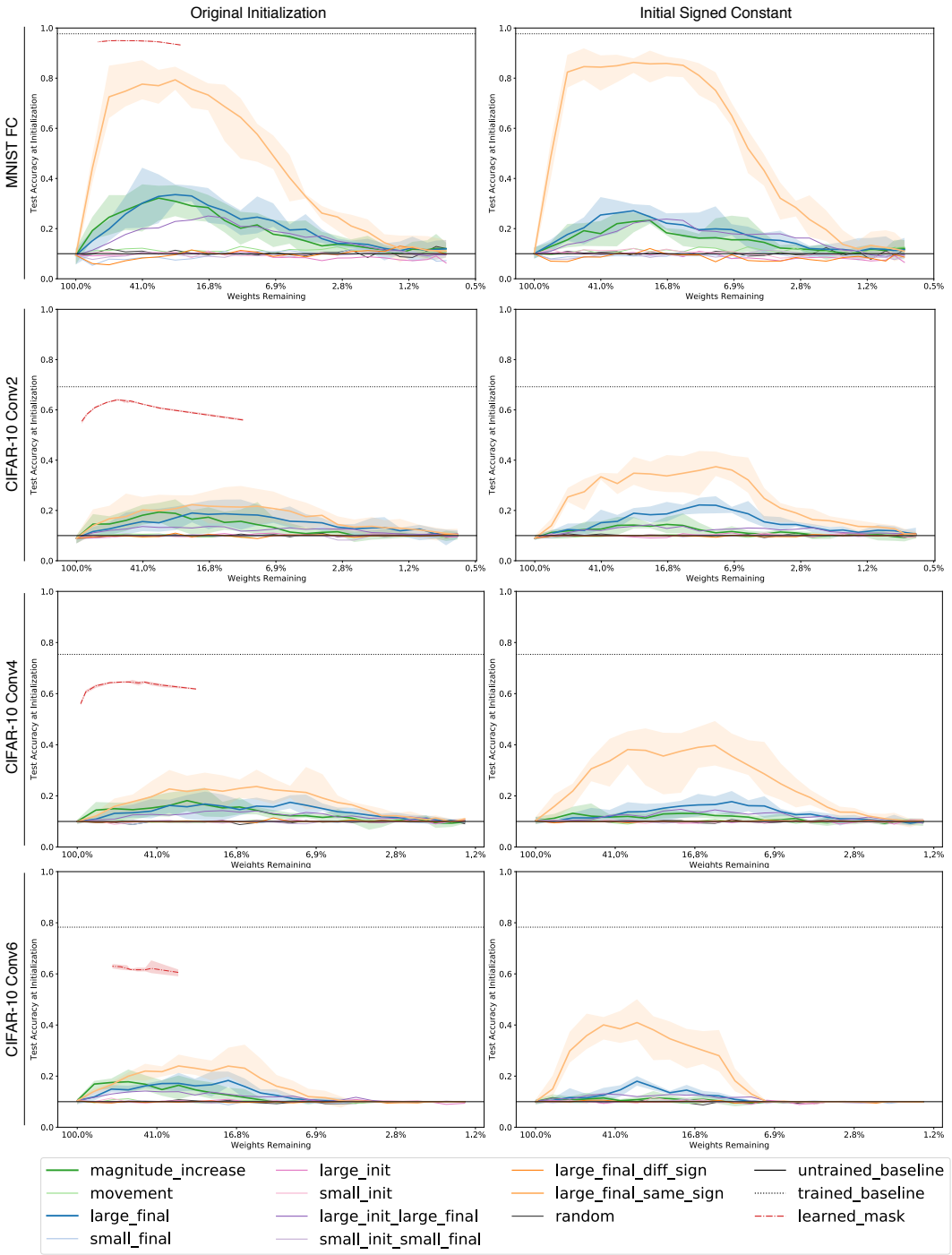

Figure 7: Comparision of Supermask performances in terms of test accuracy on MNIST and CIFAR-10 classification tasks. Subfigures are across various network structures (from top row to bottom: FC on MNIST, Conv2 on CIFAR-10, Conv4 on CIFAR-10, Conv6 on CIFAR-10), as well as 1-action treatments (left: weights are frozen at their original initialization, right: weights are frozen at a signed constant). No training is performed in any network. Weights are frozen at either initialization or constant and various masks are applied. Within heuristic based Supermasks (excluding learned_mask), the large_final_same_sign mask creates the highest performing Supermask by a wide margin. Note that aside from the five independent runs performed to generate uncertainty bands for this plot, every data point on the plot is the same underlying network, just with different masks.

Table 1: Test accuracy of the best supermasks with various initialization treatments. Values shown are max over any prune percentage. The left two columns show untrained network with heuristic based mask. The third column shows untrained network with learned mask. The last column is from training the full network.

Network	mask $\odot$ initial weights	mask $\odot$ signed constant	learned mask $\odot$ initial weights	trained network
MNIST FC	79.3	86.3	95.3	97.7
CIFAR Conv2	22.3	37.4	64.4	69.2
CIFAR Conv4	23.7	39.7	65.4	75.4
CIFAR Conv6	24.0	41.0	65.3	78.3

values. One example of g is $\lfloor (S(m)) \rceil$, where S is the sigmoid function and $\lfloor \rceil$ means rounding. Bias terms are added as usual to the product of w' with the inputs as per the usual fully connected or convolutional kernels.

We train the masks with $g(m) = \mathrm{Bern}(S(m))$, where $\mathrm{Bern}(p)$ is the bernoulli sampler with probability p. It works slightly better than $\lfloor (S(m)) \rceil$. The bernoulli sampling adds some stochasticity that helps with training, mitigates the bias of all things starting at the same value, and uses in effect the expected value of $S(m)$, which is especially useful when they are close to 0.5.

By training the m matrix with SGD, we obtained up to 95.3% test accuracy on MNIST and 65.4% on CIFAR-10. Results are shown in Figure 7, along with all the heuristic based, unlearned Supermasks. Note that there is no straightforward way to control for the pruning percentage. What we do is initializing m with a constant of different magnitudes, whose value nudges the network toward pruning more or less. With this tactic we are able to produce masks with the amounts of pruning (percentages of zeros) ranging from 7% to 89%. Further details about the training can be seen in Section S3.

Table 1 summarizes the best test accuracy obtained through different treatments. The result shows striking improvement of learned Supermasks over heuristic based ones. And learning Supermasks results in performance not too far from training the full network. It suggests that a network upon initialization has already contained powerful subnetworks that work well. Additionally, the learning of Supermask allows identifying a possibly optimal pruning rate for each layer, since each layer is free to learn the distribution of 0s in m on their own. For instance, in [3] the last layer of each network is designed to be pruned approximately half as much as the other layers, in our setting this ratio is automatically adjusted.

6 Conclusion

In this paper, we have studied how three components to LT-style network pruning—mask criterion, treatment of kept weights during retraining (mask-1 action), and treatment of pruned weights during retraining (mask-0 action)—come together to produce sparse and performant sub-networks. We proposed the hypothesis that networks work well when pruned weights are set close to their final values. Building on this hypothesis, we introduced alternative freezing schemes and other mask criteria that meet or exceed current approaches by respecting this basic rule. In addition, we demonstrate that although it is important to keep the original initialization of the weights during retraining, the only element of the initialization that is crucial is the sign, not the relative magnitude of the weights. We also showed that the masking procedure can be thought of as a training operation, and consequently we uncovered the existence of Supermasks, which can produce partially working networks without training.

Acknowledgments

The authors would like to acknowledge Jonathan Frankle, Joel Lehman, and Sam Greydanus for combinations of helpful discussion and comments on early drafts of this work.

References

[1] Yann Dauphin and Yoshua Bengio. Big neural networks waste capacity. *CoRR*, abs/1301.3583, 2013.

[2] Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pages 2148–2156, 2013.

[3] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations (ICLR)*, volume abs/1803.03635, 2019.

[4] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[5] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.

[6] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1135–1143. Curran Associates, Inc., 2015.

[7] Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 164–171. Morgan-Kaufmann, 1993.

[8] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan-Kaufmann, 1990.

[11] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the Intrinsic Dimension of Objective Landscapes. In *International Conference on Learning Representations*, April 2018.

[12] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[13] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations (ICLR)*, volume abs/1608.08710, 2017.

[14] C. Louizos, K. Ullrich, and M. Welling. Bayesian Compression for Deep Learning. *ArXiv e-prints*, May 2017.

[15] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision (ICCV)*, pages 5058–5066, 2017.

[16] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations (ICLR)*, 2017.

[17] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *CoRR*, abs/1608.03665, 2016.

[18] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5687–5695, 2017.

[19] Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. Deep fried convnets. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1476–1483, 2015.

Supplementary Information for: Deconstructing Lottery Tickets: Zeros, Signs, and the Supermask

S1 Architectures and training hyperparameters

Table S1 contains the architectures used in this study, together with relevant training hyperparameters, based off of experiments in [3].

Table S1: The architectures used in this paper. Table reproduced and modified from [3]. Conv networks use 3x3 convolutional layers with max pooling followed by fully connected layers. FC layer sizes are from [9]. Initializations are Glorot Normal [4] and activations are ReLu.

Network	MNIST FC	CIFAR-10 Conv2	CIFAR-10 Conv4	CIFAR-10 Conv6
Convolutional Layers		64, 64, pool	64, 64, pool 128, 128, pool	64, 64, pool 128, 128, pool 256, 256, pool
FC Layers None	300, 100, 10	256, 256, 10	256, 256, 10	256, 256, 10
All/Conv Weights	266K	4.3M / 38K	2.4M / 260K	2.3M / 1.1M
Iterations/Batch	50K / 60	20K / 60	25K / 60	30K / 60
Optimizer	Adam 1.2e-3	Adam 2e-4	Adam 3e-4	Adam 3e-4
Pruning Rates	fc20%	conv10% fc20%	conv10% fc20%	conv15% fc20%

S2 Further mask-0 action details

Figure S1 shows why the large_final criterion creates Supermasks.

S3 Further training details for learning Supermasks

We train the networks with mask m for each layer (and all regular kernels and biases frozen) with SGD, 0.9 momentum. The {FC, Conv2, Conv4, Conv6} networks respectively had {100, 100, 50, 20} for learning rates and trained for {2000, 2000, 1000, 800} iterations. These hyperparameters may seem absurd, but a network of masks is quite different and cannot train well with typical learning rates. Conv4 and Conv6 showed significant overfitting, thus we used early stopping as we are unable to use standard regularizing techniques. For evaluation, we also use Bernoulli sampling, but average the accuracies over 10 independent samples.

For adjusting the amount pruned, we initialized m in every layer to be the same constant, which ranged from -5 to 5. In the future it may be worth trying different initializations of m for each layer for more granular control over per-layer pruning rates. A different method to try would be to add an L1 loss to influence layers to go toward certain values, which may alleviate the cold start problems of some networks not learning anything due to mask values starting too low (effectively having the entire network start at zero).

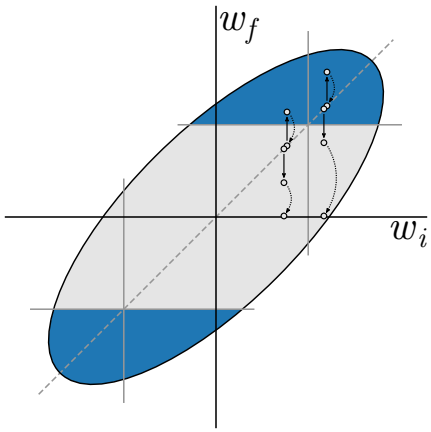

Figure S1: The motion selectivity of the large_final mask criterion. Although large_final only selects for large final weights, it biases weights near the selection threshold toward larger values (their init value) if they increased during training and toward a smaller value (0) if they decreased during training. This explains why the simple large_final criterion creates Supermasks.