THE LOTTERY TICKET HYPOTHESIS: FINDING SPARSE, TRAINABLE NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural network pruning techniques can reduce the parameter counts of trained networks by over 90%, decreasing storage requirements and improving computational performance of inference without compromising accuracy. However, contemporary experience is that the sparse architectures produced by pruning are difficult to train from the start, which would similarly improve training performance.

We find that a standard technique for pruning weights naturally uncovers subnetworks whose initializations made them capable of training effectively. Based on these results, we articulate the *lottery ticket hypothesis*: unpruned, randomly-initialized feed-forward networks contain subnetworks (*winning tickets*) that—when trained in isolation—converge in a comparable number of iterations to comparable generalization accuracy. The winning tickets we find have won the initialization lottery: their connections have initial weights that make training particularly effective.

We present an algorithm to identify winning tickets and a series of experiments that support the lottery ticket hypothesis and the importance of these fortuitous initializations. We consistently find winning tickets that are less than 10% of the size of several fully-connected and convolutional feed-forward architectures for MNIST and CIFAR10. Furthermore, the winning tickets we find above that size converge faster than the original network and exhibit higher test accuracy.

1 INTRODUCTION

Techniques for eliminating unnecessary weights from neural networks (*pruning*) (LeCun et al., 1990; Hassibi & Stork, 1993; Han et al., 2015; Li et al., 2016) can reduce parameter-counts by more than 90% while maintaining accuracy. Doing so diminishes the size (Han et al., 2015; Hinton et al., 2015) or energy consumption (Yang et al., 2017; Molchanov et al., 2016; Luo et al., 2017) of trained networks, making inference more efficient. If the unpruned networks have such excess capacity, why do we not train the smaller, pruned architectures instead in the interest of more efficient training? Contemporary experience is that the sparse architectures uncovered by pruning are harder to optimize, reaching lower generalization accuracy than the original networks when trained from the start.[1]

Consider an example. In Figure 1, we randomly sample and train sparse subnetworks of decreasing size from a fully-connected network for MNIST and convolutional networks for CIFAR10. Dashed lines trace the average (ten trials) convergence times and test accuracy at convergence of networks trained at various sizes. The sparser the network, the worse the convergence times and test accuracy.

In this paper, we show that there consistently exist sparse subnetworks that train from the start and at least match the test accuracy and convergence times of the original network. The solid lines in Figure 1 show some of these networks. Based on these results, we articulate *the lottery ticket hypothesis*.

The Lottery Ticket Hypothesis. *Any randomly-initialized feed-forward neural network that trains to convergence and reaches a particular generalization accuracy contains a subnetwork that is*

[1] "Training a pruned model from scratch performs worse than retraining a pruned model, which may indicate the difficulty of training a network with a small capacity." (Li et al., 2016) "During retraining, it is better to retain the weights from the initial training phase for the connections that survived pruning than it is to re-initialize the pruned layers...gradient descent is able to find a good solution when the network is initially trained, but not after re-initializing some layers and retraining them." (Han et al., 2015)

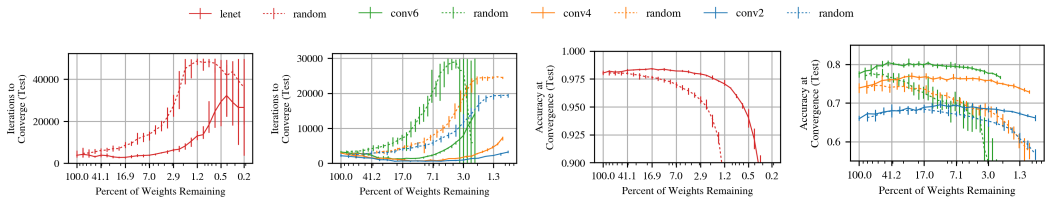

Figure 1: The convergence times (left) and test accuracies at convergence-time (right) of the lenet architecture for MNIST and the conv2, conv4, and conv6 architectures for CIFAR10 (see Figure 2) when trained starting at various sizes. The dashed lines are randomly sampled sparse networks (averaged across ten trials). The solid lines are winning tickets (averaged across five trials).

initialized such that—when trained in isolation—it can learn to match the generalization accuracy of the original network in at most the same number of training iterations.

We find that a standard pruning technique automatically uncovers such subnetworks from fully-connected and convolutional feed-forward networks. We designate these subnetworks *winning tickets*, since those that we find have won the initialization lottery with a combination of weights and connections capable of training. When randomly reinitialized, our winning tickets no longer match the performance of the original network, explaining the difficulty of training pruned networks from scratch and the importance of the original initialization.

Finding winning tickets. We identify winning tickets by training networks and subsequently pruning their smallest-magnitude weights. The set of connections that survives this process is the architecture of a winning ticket. Unique to our work, the winning ticket's weights are the values to which these connections were initialized *before training*. This forms our central experiment:

1. Randomly initialize a neural network.
2. Train the network until it converges.
3. Prune a fraction of the network.
4. To extract the winning ticket, reset the weights of the remaining portion of the network to their values from (1)—the initializations they received before training began.

If large networks contain winning tickets and pruning reveals them, then the pruned network—when reset to the original initializations and trained to convergence—will maintain competitive accuracy and convergence times at sizes too small for a randomly-initialized network to do the same.

Research questions and results. In this paper, we investigate the following questions:

How does the lottery ticket hypothesis manifest for different networks? We identify winning tickets in a fully-connected architecture for MNIST and convolutional architectures for CIFAR10 across several optimization strategies (SGD, momentum, and Adam) and dropout.

How large are winning tickets? The winning tickets we find are 10% (or less) of the size of the "original network and match its convergence times and accuracy.

How important are the structure and initialization of a winning ticket? When randomly reinitialized, winning tickets perform far worse than the original network, meaning structure alone cannot explain a winning ticket's success.

The Lottery Ticket Conjecture. Returning to our motivating question, we extend our hypothesis into an untested conjecture that gradient descent seeks out and trains a subset of well-initialized weights: Randomly-initialized, unpruned networks are easier to train because they have more possible subnetworks from which training can recover a winning ticket.

Contributions.

Network	lenet	conv2	conv4	conv6
Convolutions		64, 64, pool	64, 64, pool 128, 128, pool	64, 64, pool 128, 128, pool 256, 256, pool
FC Layers	300, 100, 10	256, 256, 10	256, 256, 10	256, 256, 10
All/Conv Weights	266K	4.3M / 38K	2.4M / 260K	1.7M / 1.1M
Iterations	50K	20K	25K	30K
Optimizer	Adam 12e-3	Adam 2e-3	Adam 3e-3	Adam 3e-3
Pruning Rate	20%	conv10% fc20%	conv10% fc20%	conv15% fc20%

Figure 2: Architectures tested in this paper. Convolutions are 3x3. Lenet is from LeCun et al. (1998). Conv2/4/6 are variants of VGG (Simonyan & Zisserman, 2014). Initializations are Gaussian Glorot (Glorot & Bengio, 2010).

- We demonstrate that pruning uncovers sparse, trainable networks that generalize as well as the original, feed-forward networks from which they derived.
- We show that winning tickets at moderate levels of pruning converge in fewer iterations and reach higher accuracy than the original network.
- We propose the *lottery ticket hypothesis* as a new perspective on the composition of neural networks to explain these findings.

Implications. In this paper, we empirically study the lottery ticket hypothesis. Now that we have demonstrated the existence of winning tickets, we hope to exploit this knowledge to:

Improve the performance of training. Since winning tickets can be trained from the start in isolation, we can design training schemes that search for winning tickets and prune as early as possible.

Design better networks. Winning tickets reveal combinations of sparse architectures and initializations that are particularly adept at learning. We can find, study, and take inspiration from winning tickets to design new architectures with the same inductive biases conducive to learning.

Improve our theoretical understanding of neural networks. We can study why feed-forward networks trained with gradient descent seem to consistently contain winning tickets and improve our understanding of the way neural networks learn.

2 Winning Tickets in Fully-Connected Networks

In this Section, we assess the lottery ticket hypothesis as applied to fully-connected networks trained on MNIST. We use the lenet-300-100 architecture (LeCun et al., 1998) as described in Figure 2. We follow the outline from Section 1: after randomly initializing and training a network, we prune the network and reset the remaining connections to their original initializations. We use a simple pruning heuristic: remove a percentage of the weights with the lowest magnitudes within each layer (as in Han et al. (2015)). Connections to outputs are pruned at half of the rate of the rest of the network. We present the results of an exploration of the hyperparameters in Appendix B, including other learning rates, optimization strategies (SGD, momentum), initialization strategies, and network sizes.

We test two pruning strategies: *one-shot pruning* and *iterative pruning*. One-shot pruning prunes all at once in a single step after training. Iterative pruning repeatedly trains, prunes, and resets the weights, removing more of the network on each iteration of the process.

Overview of results. Winning tickets found via pruning converge faster than the original network. Figure 3 plots the test set accuracy and convergence behavior during training of winning tickets iteratively pruned to different levels. We define convergence as the iteration of minimal test loss. Each curve is the average of five runs; error bars are the minimum and maximum of any run. For the first few pruning steps, convergence times decrease and test accuracy increases (left graph in Figure 3). A winning ticket comprising 51.3% of the weights from the original network converges faster

than the original network but slower than when pruned to 21.1%. After 21.1%, convergence times increase (middle graph). When pruned to about 7%, a winning ticket regresses to the performance of the original network. A similar pattern repeats throughout this paper.

Oneshot pruning. The top of Figure 4 summarizes this behavior for all pruning levels under oneshot pruning. On the left are convergence times in relation to the percent of weights remaining after pruning; on the right is accuracy at convergence. When pruned to 22% of the original network size, the average winning tickets continue to converge as fast as the original network; when pruned to between 52% and 10%, test accuracy is higher than the original network. Further pruning beyond these thresholds causes convergence times and accuracy to degrade.

Iterative pruning. Figure 4 (bottom) shows the results of iteratively pruning by 20% per iteration (blue). One-shot data from Figure 4 (top) is reproduced in green. (The x-axis is now logarithmic.) Iteratively-pruned winning tickets converge faster and reach higher test accuracy at smaller network sizes. Average convergence times decrease until 21% of the network remains, at which point the winning ticket converges on average 27% faster than the original network. After this point, further pruning causes convergence times to increase, reaching the convergence time of the original network when 7% of the network remains. Test accuracy increases with pruning, improving by more than 0.4 percentage points when the network is pruned to 17% of its original size; after this point, accuracy decreases, returning to the level of the original network when 3.5% of weights remain. Although iterative pruning extracts smaller winning tickets, repeated training means they are costlier to find. However, we aim to analyze the behavior of winning tickets rather than to find them efficiently. Iterative pruning's advantage is that it puts a tighter upper-bound on the size of a winning ticket.

Random reinitialization. To measure the importance of a winning ticket's initialization, we retain the structure of a winning ticket but randomly sample new initializations from the original distribution. We randomly reinitialize each winning ticket three times, making 15 total per point in Figure 4. We find that initialization is crucial for the efficacy of a winning ticket.

The right graph in Figure 3 shows this experiment for iterative pruning. In addition to the original network and winning tickets at 51% and 21%, it has two curves for the random reinitialization experiments. Where convergence times improve for the winning tickets, they progressively worsen when randomly reinitialized. The broader results of this experiment are the red line and orange line (random reinitialization of one-shot and iterative winning tickets, respectively) in Figure 4. Unlike winning tickets, the reinitialized networks converge increasingly slower than the original network and lose test accuracy after little pruning. The average reinitialized iterative winning ticket's test accuracy drops off from the original accuracy when the network is pruned to about 26.3%, compared to 2.9% for the winning ticket. When pruned to 21%, the winning ticket converges 2.44x faster than when reinitialized and is half a percentage point more accurate. This experiment supports the lottery ticket hypothesis' emphasis on initialization: the original initialization withstands and benefits from pruning, while the random reinitialization's performance immediately suffers and diminishes steadily.

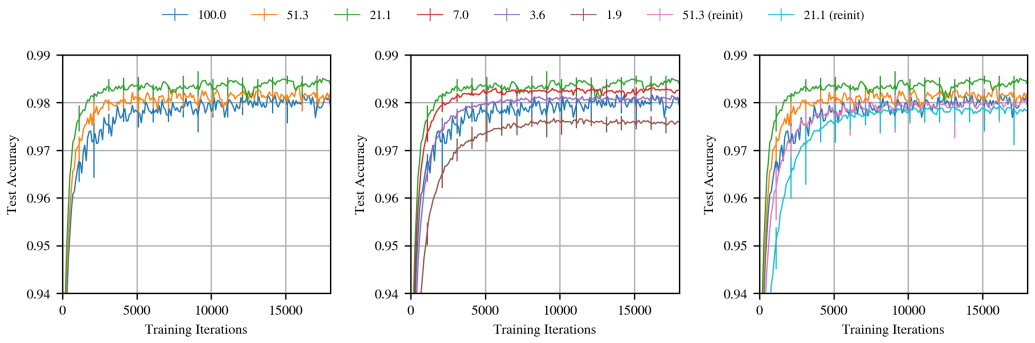

Figure 3: Test accuracy on lenet (iterative pruning) as training proceeds. Each curve is the average of five trials. Labels are the fraction of weights remaining in each layer after pruning. Error bars are the minimum and maximum of any trial.

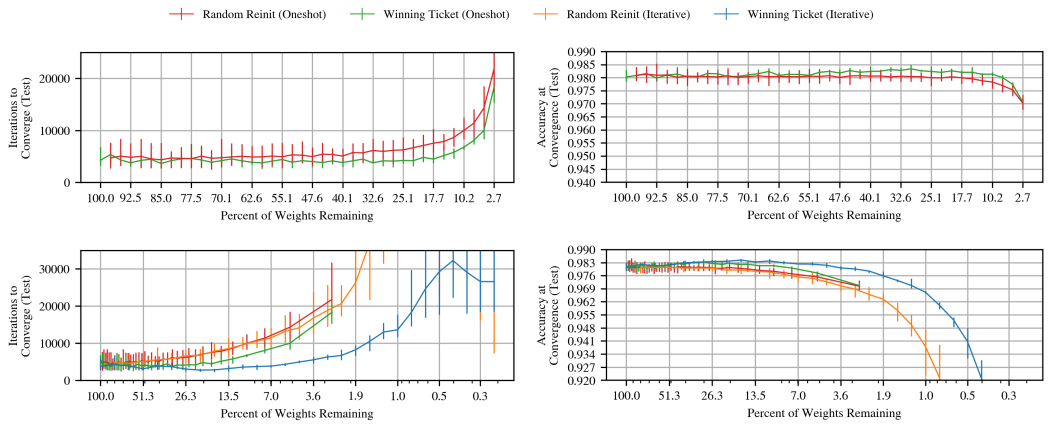

Figure 4: Convergence behavior and accuracy of lenet under one-shot (top) and iterative (bottom) pruning. Each line is the average of five trials; error bars are the minimum and maximum values.

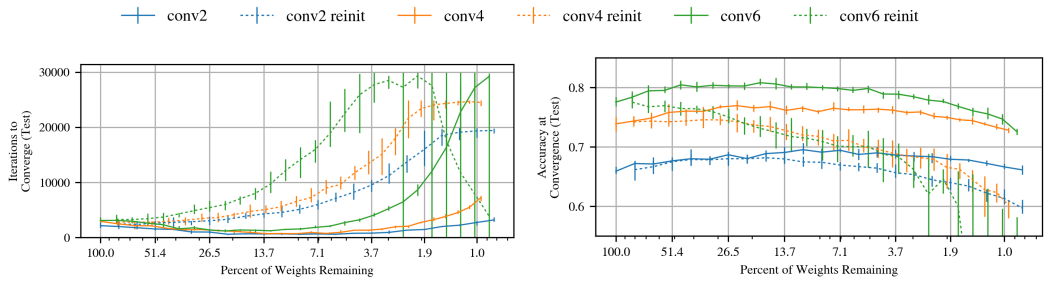

Figure 5: Convergence behavior and test accuracy of the conv2/4/6 architectures when iteratively pruned and when randomly reinitialized. Each solid line is the average of five trials; each dashed line is the average of fifteen reinitializations (three per lottery ticket experiment trial).

3 Winning Tickets in Convolutional Networks

Here, we apply the lottery ticket hypothesis to convolutional networks on CIFAR10. We consider the conv2, conv4, and conv6 architectures in Figure 2, which are scaled-down variants of the VGG (Simonyan & Zisserman, 2014) family. The networks have two, four, and six convolutional layers, respectively, followed by two fully-connected layers. The networks cover the range from fully-connected to convolutional networks, with less than 1% of parameters in convolutional layers in conv2 to nearly two thirds in conv6. We present the results of an exploration of the hyperparameters in Appendix C, including other learning rates and optimization strategies (SGD, momentum). We investigate three questions:

Question 1: How does the lottery ticket hypothesis manifest for convolutional networks?

Question 2: What is the effect of randomly reinitializing winning tickets?

Question 3: What is the effect of pruning convolutions and fully-connected layers alone and together?

Question 1. The solid lines in Figure 5 show the iterative lottery ticket experiment on conv2 (blue), conv4 (orange), and conv6 (green) at the pruning rates from Figure 2. The pattern from Section 2 repeats: as the network is pruned, convergence times drop and test accuracy rises as compared to the original network. In this case, the results are more pronounced. Winning tickets converge at best 3.91x faster for conv2 (5.7% of weights remaining), 4.17x for conv4 (9.2% remaining), and 2.41x for conv6 (12.6% remaining). Test accuracy improves at best 3.5 percentage points for conv2

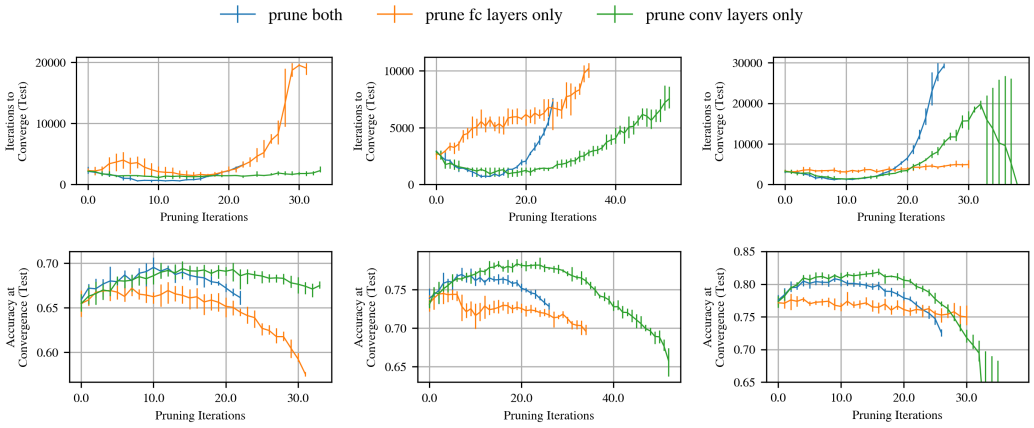

Figure 6: Convergence times and accuracy of the conv2 (left), conv4 (middle), and conv6 (right) networks when only convolutions are pruned, only fully-connected layers are pruned, and both are pruned. The x-axis measures the number of pruning iterations, making it possible to see the relative contributions to the overall network made by pruning FC layers and convolutions individually.

(11.0% remaining), 2.6 for conv4 (7.7% remaining), and 3.5 for conv6 (15.1% remaining). All three networks remain above their original average test accuracy until 2% or less of weights remain.

Question 2. We repeat the random reinitialization trial from Section 2, which appears as the dashed lines in Figure 5. These experiments again take increasingly longer to converge upon continued pruning. As in Section 2, test accuracy drops off more quickly for the random reinitialization experiments. However, unlike Section 2, test accuracy at convergence time initially remains steady and even improves for conv2 and conv4, indicating that—at moderate levels of pruning—the structure of the winning tickets alone may lead to better generalization.

Question 3. Figure 6 shows the effect of pruning just convolutions (green), just fully-connected layers (orange) and pruning both (blue). The x-axis measures the number of pruning iterations to emphasize the relative contributions made by pruning convolutions and fully-connected layers to the overall network. In all three cases, pruning convolutions alone leads to improvements in test accuracy and convergence times; pruning fully-connected layers alone generally causes test accuracy to worsen and convergence times to increase. Pruning convolutional layers alone improves the best average test accuracy by 1.3 and 0.8 percentage points for conv4 and conv6, respectively. However, pruning convolutions alone has limited ability to reduce the overall parameter-count of the network, since fully-connected layers comprise 99%, 89%, and 35% of the parameters in conv2, conv4, and conv6.

4 Winning Tickets and Dropout

Dropout (Srivastava et al., 2014; Hinton et al., 2012) improves network accuracy by randomly disabling a fraction of the units (i.e., randomly sampling a subnetwork) on each training iteration. Baldi & Sadowski (2013) characterize dropout as simultaneously training the ensemble of all subnetworks. Since the lottery ticket hypothesis suggests that one of these subnetworks comprises a winning ticket, it is natural to ask whether dropout and our strategy for finding winning tickets interact.

Figure 7 shows the results of training conv2, conv4, and conv6 with a dropout rate of 0.5. Dashed lines are the network performance without dropout (the solid lines in Figure 5).[2] We continue to find winning tickets when training with dropout. Dropout increases initial test accuracy (3.6, 2.7, and 2.0 percentage points on average for conv2, conv4, and conv6, respectively), and iterative pruning increases it further (up to 3.7, 4.8, and 4.9 percentage points, respectively, on average). Convergence times improve with iterative pruning as before, but less dramatically in the case of conv2.

[2]We choose new learning rates for the networks as trained with dropout—see Appendix C.5.

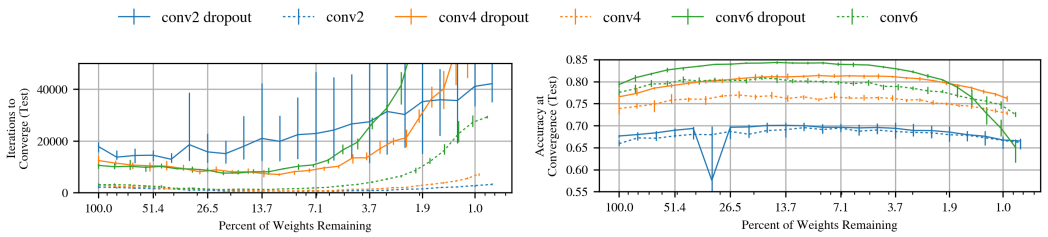

Figure 7: Convergence behavior and test accuracy of conv2, conv4, and conv6 when iteratively pruned and trained with dropout. The dashed lines are the same networks trained without dropout (the solid lines in Figure 5. Learning rates are 0.0003 for conv2 and 0.0002 for conv4 and conv6.

These improvements suggest that our iterative pruning strategy interacts with dropout in a complementary way. Srivastava et al. (2014) observe that dropout induces sparse activations in the final network; it is possible that dropout-induced sparsity primes a network to be pruned. If so, dropout techniques that target weights (Wan et al., 2013) or learn per-weight dropout probabilities (Molchanov et al., 2017; Louizos et al., 2018) could make winning tickets even easier to find.

5 Discussion

Neural network pruning (e.g., Han et al. (2015)) asks whether the function learned by the original, unpruned network can be represented by a smaller network. It answers this question affirmatively, constructing such a network by training the original network, pruning connections, and further training the already-trained weights. In effect, the inital stage of training the unpruned network serves to warm up the weights of the pruned network so that it can be optimized. Whether such a smaller representation can be trained from the start has remained an open question; we answer that question affirmatively for the feed-forward architectures evaluated in this paper, and the lottery ticket hypothesis contends that this property applies to feed-forward networks more generally.

We extend the lottery ticket hypothesis into a conjecture that winning tickets are not just an artifact that we uncover but a central element of the neural network optimization process. Namely, gradient descent seeks out and trains fortuitously-initialized subcomponents of unpruned networks; by this logic, unpruned networks are easier to optimize than pruned networks because they have more combinations of subcomponents (that is, more "lottery tickets") that have the potential to become winning tickets through lucky initialization. From this point of view, training could be seen as a process of both subnetwork search and optimization. This remains a conjecture, and we have not justified it with experimental evidence: although we show that many networks contain winning tickets, we do not show that containing a winning ticket is necessary or sufficient for a network to learn successfully. We leave consideration of this question to future work.

6 Limitations

We show that iterative pruning recovers winning tickets from fully-connected and convolutional feed-forward networks trained on MNIST and CIFAR10. We acknowledge several limitations of our experiments. We only consider vision-centric datasets and networks, and our experiments use smaller datasets (MNIST, CIFAR10) and networks. We do not investigate larger networks for larger datasets (namely Imagenet (Russakovsky et al., 2015)) because iterative pruning is exceedingly computationally expensive, requiring training a network 15 or more times consecutively for multiple trials; hyperparameter search multiplicatively increases this cost.

We only consider fully-connected and convolutional architectures. We do not consider RNNs, GNNs, etc. In Appendix D, we apply this technique to resnet18; iterative pruning does not find winning tickets at the learning rate in He et al. (2016) but finds winning tickets of commensurate accuracy at lower learning rates.

The winning tickets we find have initializations that allow them to match the performance of the unpruned networks at sizes far smaller than randomly-initialized networks can do the same. We do not explore the properties of these initializations that, in concert with the inductive biases of the pruned network architectures, makes these networks particularly adept at learning.

7 Related Work

In practice, neural networks tend to be dramatically overparameterized. Distillation (Ba & Caruana, 2014; Hinton et al., 2015) and pruning (LeCun et al., 1990; Han et al., 2015) rely on the fact that parameters can be reduced while preserving accuracy. Even with sufficient capacity to memorize training data, networks naturally learn simpler functions (Zhang et al., 2016; Neyshabur et al., 2014; Arpit et al., 2017). Contemporary experience (Bengio et al., 2006; Hinton et al., 2015; Zhang et al., 2016) and Figure 1 suggest that overparameterized networks are easier to train. We show that dense networks contain sparse subnetworks capable of learning on their own starting from their original initializations. Several other research directions aim to train small or sparse networks.

Prior to training. Squeezenet (Iandola et al., 2016) and MobileNets (Howard et al., 2017) are specifically engineered image-recognition networks that are an order of magnitude smaller than standard architectures. Denil et al. (2013) represent weight matrices as products of lower-rank factors. Li et al. (2018) restrict optimization to a small, randomly-sampled subspace of the parameter space (meaning all parameters can still be updated); they successfully train networks under this restriction. We show that one need not even update all parameters to optimize a network, and we find winning tickets though a principled search process involving pruning. Our contribution to this class of approaches is to demonstrate that sparse, trainable networks exist within larger networks.

After training. Distillation (Ba & Caruana, 2014; Hinton et al., 2015) trains small networks to mimic the behavior of large networks; small networks are easier to train in this paradigm. Recent pruning work aims to compress large models into forms that run with limited resources (e.g., on mobile devices). Although pruning is central to our experiments, we aim to gain insight into why training needs the overparameterized networks that make pruning necessary. LeCun et al. (1990) and Hassibi & Stork (1993) first explored pruning based on second derivatives. More recently, Han et al. (2015) showed per-weight magnitude-based pruning substantially reduces the size of image-recognition networks. Han et al. iteratively train to convergence, prune, and continue training. Guo et al. (2016) restore pruned connections as they become relevant again. Han et al. (2017) and Jin et al. (2016) restore pruned connections to increase network capacity after small weights have been pruned and surviving weights fine-tuned. Other proposed pruning heuristics include pruning based on activations (Hu et al., 2016), redundancy (Mariet & Sra, 2016; Srinivas & Babu, 2015a), per-layer second derivatives (Dong et al., 2017), and energy/computation efficiency (Yang et al., 2017) (e.g., pruning convolutional filters (Li et al., 2016; Molchanov et al., 2016; Luo et al., 2017) or channels (He et al., 2017)). Cohen et al. (2016) observe that convolutional filters are sensitive to initialization ("The Filter Lottery"); after training, they randomly reinitialize unimportant filters.

During training. Bellec et al. (2018) train with sparse networks and replace weights that reach zero with new random connections. Srinivas et al. (2017) and Louizos et al. (2018) learn gating variables that minimize the number of nonzero parameters. Narang et al. (2017) integrate magnitude-based pruning into training. Gal & Ghahramani (2016) show that dropout approximates Bayesian inference in Gaussian processes. Bayesian perspectives on dropout learn dropout probabilities during training (Gal et al., 2017; Kingma et al., 2015; Srinivas & Babu, 2016). Techniques that learn per-weight, per-unit (Srinivas & Babu, 2016), or structured dropout probabilities naturally (Molchanov et al., 2017; Neklyudov et al., 2017) or explicitly (Louizos et al., 2017; Srinivas & Babu, 2015b) prune and sparsify networks during training as dropout probabilities for some weights reach 1. In contrast, we train networks at least once to find winning tickets. These techniques might also find winning tickets, or, by inducing sparsity, might beneficially interact with our methods.

References

Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at

memorization in deep networks. In *International Conference on Machine Learning*, pp. 233–242, 2017.

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pp. 2654–2662, 2014.

Pierre Baldi and Peter J Sadowski. Understanding dropout. In *Advances in neural information processing systems*, pp. 2814–2822, 2013.

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. *Proceedings of ICLR*, 2018.

Yoshua Bengio, Nicolas L Roux, Pascal Vincent, Olivier Delalleau, and Patrice Marcotte. Convex neural networks. In *Advances in neural information processing systems*, pp. 123–130, 2006.

Joseph Paul Cohen, Henry Z Lo, and Wei Ding. Randomout: Using a convolutional gradient norm to win the filter lottery. *ICLR Workshop*, 2016.

Misha Denil, Babak Shakibi, Laurent Dinh, Nando De Freitas, et al. Predicting parameters in deep learning. In *Advances in neural information processing systems*, pp. 2148–2156, 2013.

Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pp. 4860–4874, 2017.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059, 2016.

Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. In *Advances in Neural Information Processing Systems*, pp. 3584–3593, 2017.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pp. 1379–1387, 2016.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015.

Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Shijian Tang, Erich Elsen, Bryan Catanzaro, John Tran, and William J Dally. Dsd: Regularizing deep neural networks with dense-sparse-dense training flow. *Proceedings of ICLR*, 2017.

Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pp. 164–171, 1993.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV)*, volume 2, pp. 6, 2017.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Xiaojie Jin, Xiaotong Yuan, Jiashi Feng, and Shuicheng Yan. Training skinny deep neural networks with iterative hard thresholding methods. *arXiv preprint arXiv:1607.05423*, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pp. 2575–2583, 2015.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

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.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. *Proceedings of ICLR*, 2018.

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

Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pp. 3290–3300, 2017.

Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. *Proceedings of ICLR*, 2018.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. *arXiv preprint arXiv:1707.06342*, 2017.

Zelda Mariet and Suvrit Sra. Diversity networks. *Proceedings of ICLR*, 2016.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*, 2017.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *arXiv preprint arXiv:1611.06440*, 2016.

Sharan Narang, Erich Elsen, Gregory Diamos, and Shubho Sengupta. Exploring sparsity in recurrent neural networks. *Proceedings of ICLR*, 2017.

Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry P Vetrov. Structured bayesian pruning via log-normal multiplicative noise. In *Advances in Neural Information Processing Systems*, pp. 6778–6787, 2017.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. *arXiv preprint arXiv:1412.6614*, 2014.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015a.

Suraj Srinivas and R Venkatesh Babu. Learning neural network architectures using backpropagation. *arXiv preprint arXiv:1511.05497*, 2015b.

Suraj Srinivas and R Venkatesh Babu. Generalized dropout. *arXiv preprint arXiv:1611.06791*, 2016.

Suraj Srinivas, Akshayvarun Subramanya, and R Venkatesh Babu. Training sparse neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 138–145, 2017.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pp. 1058–1066, 2013.

Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. *arXiv preprint*, 2017.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

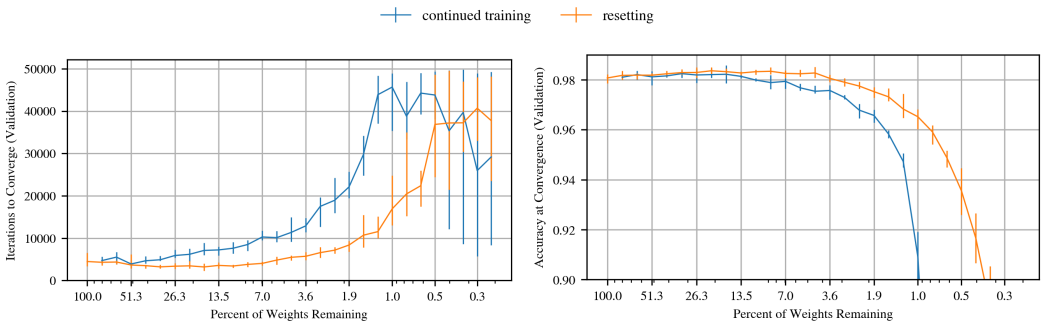

Figure 8: The convergence times and accuracy at convergence-time of the iterative lottery ticket experiment on the mnist architecture when iteratively pruned using the resetting and continued training strategies.

A ITERATIVE PRUNING STRATEGIES

In this Appendix, we examine two different ways of structuring the iterative pruning strategy that we use throughout the main body of the paper to find winning tickets.

Strategy 1: Resetting.

1. Randomly initialize a neural network.
2. Train the network until it converges.
3. Prune a fraction of the network.
4. Reset the weights of the remaining portion of the network to their values from (1)
5. Repeat steps 2 through 4 until a sufficiently pruned network has been obtained.

Strategy 2: Continued Training.

1. Randomly initialize a neural network.
2. Train the network until it converges.
3. Prune a fraction of the network.
4. Repeat steps 2 and 3 until a sufficiently pruned network has been obtained.
5. Reset the weights of the remaining portion of the network to their values from (1)

The difference between these two strategies is that, after each round of pruning, Strategy 2 retrains using the already-trained weights, whereas Strategy 1 resets the network weights back to their initial values before retraining. In both cases, after the network has been sufficiently pruned, its weights are reset back to the original initializations.

Figures 8 and 9 compare the two strategies on the lenet and conv2/4/6 architectures on the hyperparameters we select in Appendices B and C. In all cases, the Strategy 1 maintains higher validation accuracy and lower convergence times to smaller network sizes.

B HYPERPARAMETER EXPLORATION FOR FULLY-CONNECTED NETWORKS

This Appendix accompanies Section 2 of the main paper. It explores the space of hyperparameters for the lenet architecture evaluated in Section 2 with two purposes in mind:

1. To explain the hyperparameters selected in the main body of the paper.
2. To evaluate the extent to which the lottery ticket experiment patterns extend to other choices of hyperparameters.

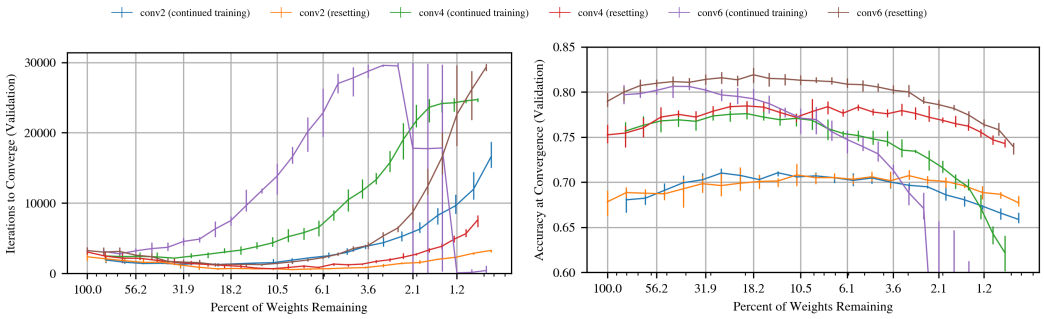

Figure 9: The convergence times and accuracy at convergence-time of the iterative lottery ticket experiment on the conv2, conv4, and conv6 architectures when iteratively pruned using the resetting and continued training strategies.

B.1 Experimental Methodology

This Section considers the fully-connected lenet architecture (LeCun et al., 1998), which comprises two fully-connected hidden layers and a ten unit output layer, on the MNIST dataset. Unless otherwise stated, the hidden layers have 300 and 100 units each.

The MNIST dataset consists of 60,000 training examples and 10,000 test examples. We randomly sampled a 5,000-example validation set from the training set and used the remaining 55,000 training examples as our training set for the rest of the paper (including Section 2). The hyperparameter selection experiments throughout this Appendix are evaluated on the validation set, and the examples in the main body of this paper (which make use of these hyperparameters) are evaluated on test set. The training set is presented to the network in mini-batches of 60 examples; at each epoch, the entire training set is shuffled.

Unless otherwise noted, each line in each graph comprises data from three separate experiments. The line itself traces the average performance of the experiments and the error bars indicate the minimum and maximum performance of any one experiment.

Throughout this Appendix, we perform the lottery ticket experiment iteratively with a pruning rate of 20% per iteration (10% for the output layer); we justify the choice of this pruning rate later in this Appendix. On each iteration of the lottery ticket experiment, the network is trained for 50,000 training iterations regardless of when it converges; in other words, no validation or test data is taken into account during the training process, and convergence times are determined retroactively by examining validation performance. We sample validation and test performance every 100 iterations.

For the main body of the paper, we opt to use the Adam optimizer (Kingma & Ba, 2014) and Gaussian Glorot initialization (Glorot & Bengio, 2010). Although we can achieve more impressive results on the lottery ticket experiment with other hyperparameters, we intend these choices to be as generic as possible in an effort to minimize the extent to which our main results depend on hand-chosen hyperparameters. In this Appendix, we select the learning rate for Adam that we use in the main body of the paper.

In addition, we consider a wide range of other hyperparameters, including other optimization algorithms (SGD with and without momentum), initialization strategies (Gaussian distributions with various standard deviations), network sizes (larger and smaller hidden layers), and pruning strategies (faster and slower pruning rates). In each experiment, we vary the chosen hyperparameter while keeping all others at their default values (Adam with the chosen learning rate, Gaussian Glorot initialization, hidden layers with 300 and 100 units). The data presented in this appendix was collected by training variations of the lenet architecture more than 3,000 times.

Since we are focused on generalization behavior, we define convergence to occur at the iteration of minimum validation loss (for hyperparameter selection) or test loss (for experiments in the main body of the paper). Validation and test loss follow a pattern where they decrease early in the training process, reach a minimum, and then begin to increase as the model overfits to the training data. Figure

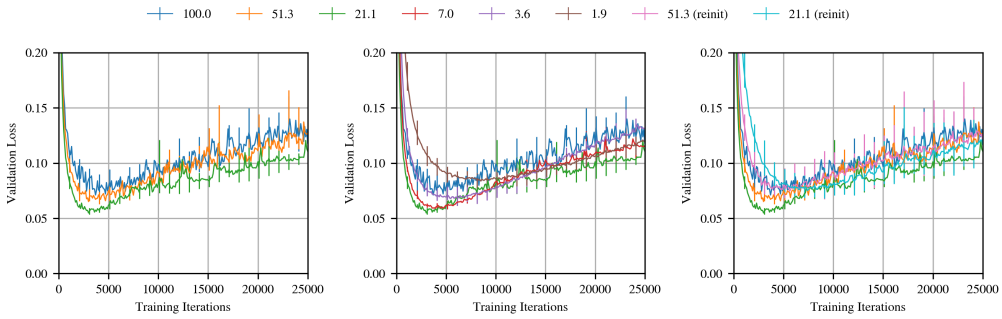

Figure 10: The validation loss as training progresses for several different levels of pruning in the iterative pruning experiment. Each line is the average of five training runs at the same level of iterative pruning; the labels are the percentage of weights from the original network that remain after pruning. Each network was trained with Adam at a learning rate of 0.0012. The left graph shows winning tickets that converge increasingly faster than the original network and reach lower loss. The middle graph shows winning tickets that converge increasingly slower after the fastest convergence time has been reached. The right graph contrasts the loss of winning tickets to the loss of randomly reinitialized networks.

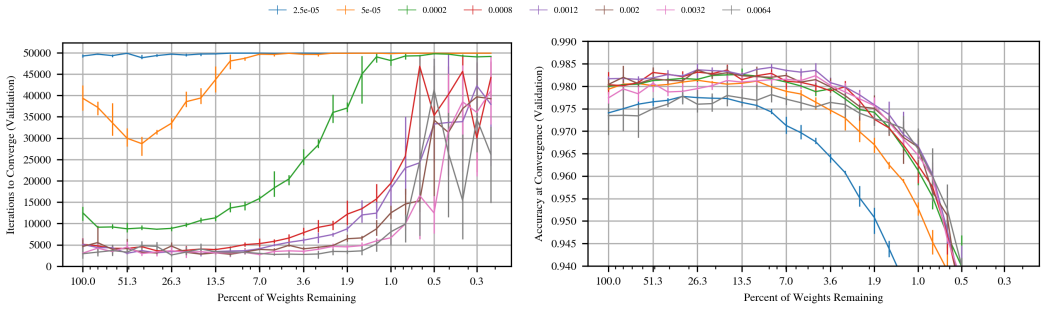

Figure 11: The convergence times and validation accuracy at convergence-time of the iterative lottery ticket experiment on the lenet architecture trained with MNIST using the Adam optimizer at various learning rates. Each line represents a different learning rate.

10 shows an example of the validation loss as training progresses; these graphs use iterative pruning and Adam with a learning rate of 0.0012 (the learning rate we will select in the following subsection).

B.2 Learning Rate

In this Subsection, we perform the lottery ticket experiment on the lenet architecture as optimized with Adam, SGD, and SGD with momentum at various learning rates.

Here, we select the learning rate that we use for Adam in the main body of the paper. Our criteria for selecting the learning rate are as follows:

1. On the unpruned network, it should minimize training iterations necessary to converge and maximize validation accuracy at convergence time. That is, it should be a reasonable hyperparameter for optimizing the unpruned network even if we are not running the lottery ticket experiment.
2. When running the iterative lottery ticket experiment, it should make it possible to match the convergence times and accuracy of the original network with as few parameters as possible.
3. Of those options that meet (1) and (2), it should be on the conservative (slow) side so that it is more likely to productively optimize heavily pruned networks under a variety of conditions with a variety of hyperparameters.

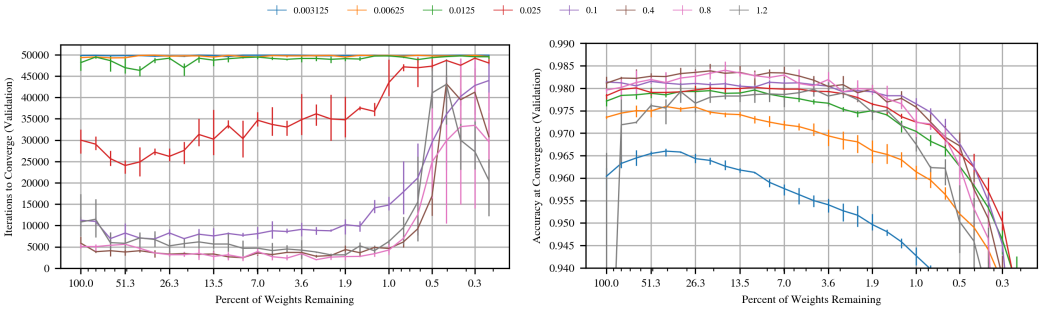

Figure 12: The convergence times and validation accuracy at convergence-time of the iterative lottery ticket experiment on the lenet architecture trained with MNIST using stochastic gradient descent at various learning rates.

Figure 11 shows the convergence times and validation accuracy at convergence-time of performing the iterative lottery ticket experiment with the lenet architecture optimized with Adam at various learning rates. According to the graph on the right of Figure 11, several learning rates between 0.0002 and 0.002 achieve similar levels of validation accuracy on the original network and maintain that performance to similar levels as the network is pruned. Of those learning rates, 0.0012 and 0.002 produce the fastest convergence times and maintain them to the smallest network sizes. We choose 0.0012 due to its higher validation accuracy on the unpruned network and in consideration of criterion (3) above.

We note that, across all of these learning rates, the lottery ticket pattern (in which convergence times decrease and validation accuracy increases with iterative pruning) remains present. Even for those learning rates that did not converge within 50,000 iterations (2.5e-05 and 0.0064) still showed accuracy improvements with pruning.

B.3 Other Optimization Algorithms

B.3.1 SGD

Here, we explore the behavior of the lottery ticket experiment when the network is optimized with stochastic gradient descent (SGD) at various learning rates. The results of doing so appear in Figure 12. The lottery ticket pattern appears across all learning rates, including those that fail to converge within 50,000 iterations. SGD learning rates 0.4 and 0.8 converge in a similar number of iterations as the best Adam learning rates (0.0012 and 0.002) but maintain this performance when the network has been pruned further (to less than 1% of its original size for SGD vs. about 5% of the original size for Adam). Likewise, on pruned networks, these SGD learning rates achieve equivalent accuracy to the best Adam learning rates, and they maintain that high accuracy when the network is pruned as much as the Adam learning rates.

B.3.2 Momentum

Here, we explore the behavior of the lottery ticket experiment when the network is optimized with SGD with momentum (0.9) at various learning rates. The results of doing so appear in Figure 13. Once again, the lottery ticket pattern appears across all learning rates, with learning rates between 0.025 and 0.1 maintaining high validation accuracy and low convergence times for the longest number of pruning iterations. Learning rate 0.025 achieves the highest validation accuracy on the unpruned network and achieves faster convergence times when pruned to less than 1% of its original size; however, its validation accuracy never increases as it is pruned, instead decreasing gradually.

B.4 Iterative Pruning Rate

Figure 14 shows the convergence times and validation accuracy when the lenet architecture is trained with Adam at our chosen learning rate and pruned iteratively at different rates. There is a tangible

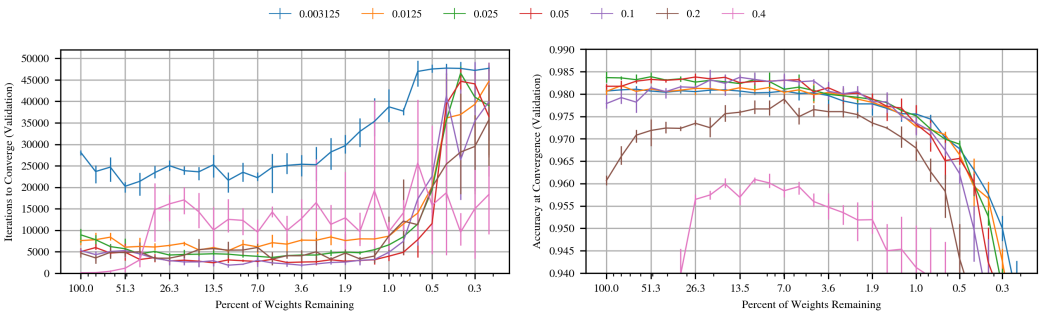

Figure 13: The convergence times and validation accuracy at convergence-time of the iterative lottery ticket experiment on the lenet architecture trained with MNIST using stochastic gradient descent with momentum (0.9) at various learning rates.

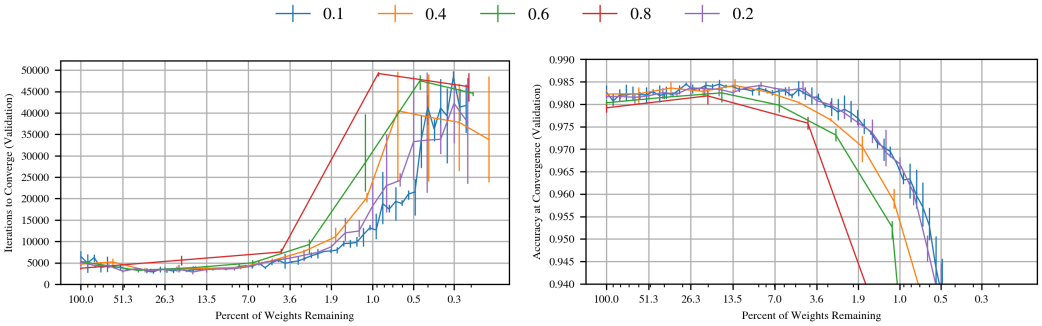

Figure 14: The convergence times and validation accuracy at convergence-time of the iterative lottery ticket experiment when pruned at different rates. Each line represents a different *pruning rate*—the percentage of lowest-magnitude weights that are pruned after each training iteration.

difference in the convergence performance and validation accuracy at convergence time between the lowest pruning rates (0.1 and 0.2) and higher pruning rates (0.4 and above). The lowest pruning rates reach higher validation accuracy and maintain that validation accuracy to smaller network sizes; they also maintain fast convergence times to smaller network sizes. For the experiments throughout the main body of the paper and this Appendix, we use a pruning rate of 0.2, which mantains much of the accuracy and convergence behavior of 0.1 while reducing the number of training iterations necessary to get to smaller network sizes.

In all of the lenet experiments, we prune the output layer at half the rate of the rest of the network. Since the output layer is so small (1,000 weights out of 266,000 for the overall lenet architecture), we found that pruning it reaches a point of diminishing returns much before the other layers. In future work, it is worth considering strategies for finding the ideal pruning rates for each individual layer of the network.

B.5 Initialization Distribution

To this point, we have considered only a Gaussian Glorot (Glorot & Bengio, 2010) initialization scheme for the network. Figure 15 performs the lottery ticket experiment while initialializing the lenet architecture from Gaussian distributions with a variety of standard deviations. The networks were optimized with Adam at the learning rate chosen earlier. The lottery ticket pattern continues to appear across all standard deviations. When initialized from a Gaussian distribution with standard deviation 0.1, the lenet architecture maintained high validation accuracy and low convergence times for the longest, approximately matching the performance of the Glorot-initialized network.

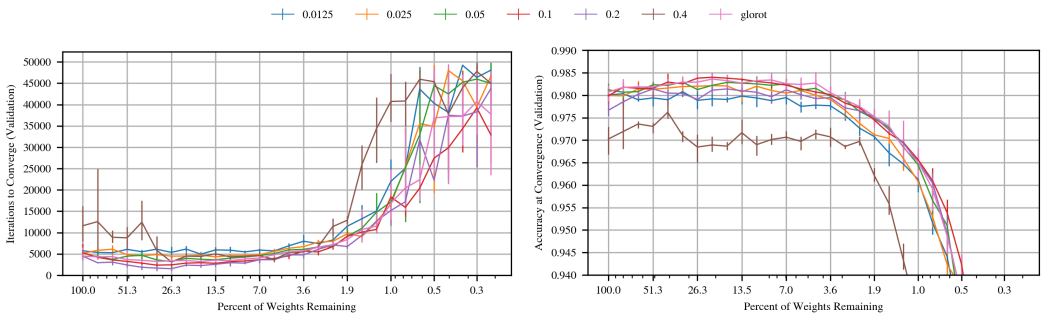

Figure 15: The convergence times and validation accuracy at convergence-time of the iterative lottery ticket experiment initialized with Gaussian distributions with various standard deviations. Each line is a different standard deviation for a Gaussian distribution centered at 0.

B.6 Network Size

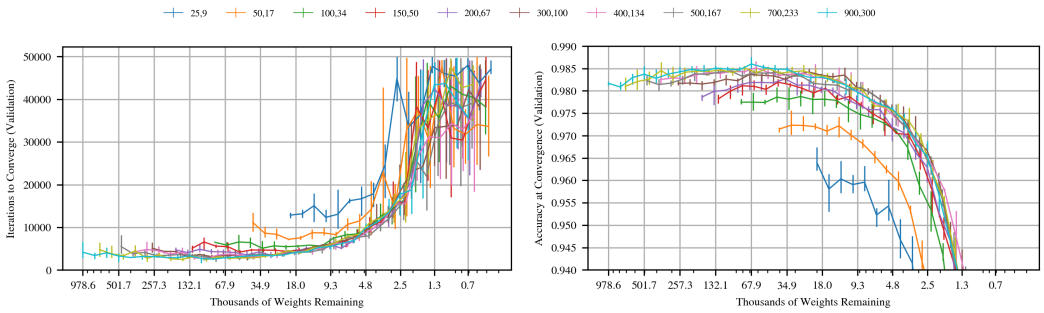

Figure 16: The convergence times and validation accuracy at convergence-time of the iterative lottery ticket experiment on the lenet architecture with various layer sizes. The label for each line is the size of the first and second hidden layers of the network. All networks had Gaussian Glorot initialization and were optimized with Adam (learning rate 0.0012). Note that the x-axis of this plot charts the number of *weights* remaining, while all other graphs in this section have charted the *percent* of weights remaining.

Throughout this section, we have considered the lenet architecture with 300 units in the first hidden layer and 100 units in the second hidden layer. Figure 16 shows the convergence times and validation accuracy at convergence-time of the lenet architecture with several other layer sizes. All networks we tested maintain the 3:1 ratio between units in the first hidden layer and units in the second hidden layer.

The lottery ticket hypothesis naturally invites a collection of questions related to network size. Generalizing, those questions tend to take the following form: according to the lottery ticket hypothesis, do larger networks, which contain more subnetworks, find "better" winning tickets? In line with the generality of this question, there are several different answers.

If we evaluate a winning ticket by the accuracy it achieves, then larger networks do find better winning tickets. The right graph in Figure 16 shows that, for any particular number of weights (that is, any particular point on the x-axis), winning tickets derived from initially larger networks reach higher accuracy. Put another way, in terms of accuracy, the lines are approximately arranged from bottom to top in increasing order of network size. It is possible that, since larger networks have more subnetworks, gradient descent found a better winning ticket. Alternatively, the initially larger networks have more units even when pruned to the same number of weights as smaller networks, meaning they are able to contain sparse subnetwork configurations that cannot be expressed by initially smaller networks.

If we evaluate a winning ticket by the time necessary for it to converge, then larger networks have less of an advantage. The left graph in Figure 16 shows that, in general, convergence times do not vary greatly between networks of different initial sizes that have been pruned to the same number of weights. Upon exceedingly close inspection, winning tickets derived from initially larger networks tend to converge marginally faster than winning tickets derived from initially smaller networks, but these differences are slight.

If we evaluate a winning ticket by the size at which it returns to the same accuracy as the original network, the large networks do not have an advantage. Regardless of the initial network size, the right graph in Figure 16 shows that winning tickets return to the accuracy of the original network when they are pruned to between about 9,000 and 15,000 weights.

C HYPERPARAMETER EXPLORATION FOR CONVOLUTIONAL NETWORKS

This Appendix accompanies Sections 3 and 4 of the main paper. It explores the space of optimization algorithms and hyperparameters for the conv2, conv4, and conv6 architectures evaluated in Section 3 and 4 with the same two purposes as Appendix B: explaining the hyperparameters used in the main body of the paper and evaluating the lottery ticket experiment on other choices of hyperparameters.

C.1 EXPERIMENTAL METHODOLOGY

The conv2, conv4, and conv6 architectures are variants of the VGG (Simonyan & Zisserman, 2014) network architecture scaled down for the CIFAR10 (Krizhevsky & Hinton, 2009) dataset. Like VGG, the networks consist of a series of modules. Each module has two layers of 3x3 convolutional filters followed by a maxpool layer with stride 2. After all of the modules are two fully-connected layers of size 256 followed by an output layer of size 10; in VGG, the fully-connected layers are of size 4096 and the output layer is of size 1000. Like VGG, the first module has 64 convolutions in each layer, the second has 128, the third has 256, etc. The conv2, conv4, and conv6 architectures have 1, 2, and 3 modules, respectively.

The CIFAR10 dataset consists of 50,000 32x32 color (three-channel) training examples and 10,000 test examples. We created a 5,000-example validation set from the training set and used the remaining 45,000 training examples as our training set for the rest of the paper. The hyperparameter selection experiments throughout this Appendix are evaluated on the validation set, and the examples in the main body of this paper (which make use of these hyperparameters) are evaluated on test set. The training set is presented to the network in mini-batches of 60 examples; at each epoch, the entire training set is shuffled.

The conv2, conv4, and conv6 networks are initialized with Gaussian Glorot initialization (Glorot & Bengio, 2010) and are trained for the number of iterations specified in Figure 2. The number of training iterations was selected such that heavily-pruned networks could still converge in the time provided. On dropout experiments, the number of training iterations is tripled to provide enough time for the dropout-regularized networks to converge. We optimize these networks with Adam, and select the learning rate for each network in this Appendix.

As with the MNIST experiments, validation and test performance is only considered retroactively and has no effect on the progression of the lottery ticket experiments; we measure convergence as the moment when validation loss (for hyperparameter selection) or test loss (for the main body of the paper) is minimal. We measure validation and test loss and accuracy every 100 training iterations.

Each line in each graph of this section represents the average of three separate experiments, with error bars indicating the minimum and maximum value that any experiment took on at that point. (Experiments in the main body of the paper are conducted five times.)

We allow convolutional layers and fully-connected layers to be pruned at different rates; we select those rates for each network in this Appendix. The output layer is pruned at half of the rate of the fully-connected layers for the reasons described in Appendix B.

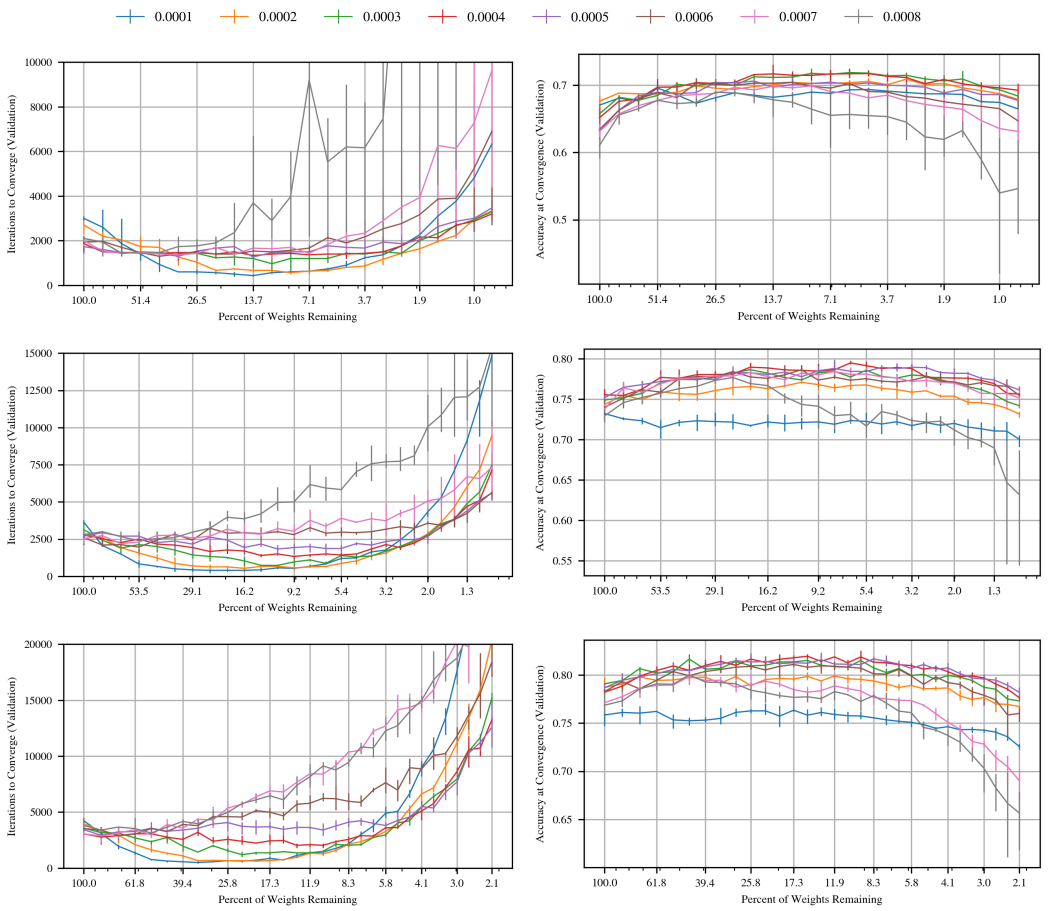

Figure 17: The convergence times and accuracy at convergence-time of the iterative lottery ticket experiment on the conv2 (top), conv4 (middle), and conv6 (bottom) architectures trained using the Adam optimizer at various learning rates. Each line represents a different learning rate.

C.2 Learning Rate

In this Subsection, we perform the lottery ticket experiment on the the conv2, conv4, and conv6 architectures as optimized with Adam at various learning rates.

Here, we select the learning rate that we use for Adam in the main body of the paper. Our criteria for selecting the learning rate are the same as in Appendix B: minimizing training iterations and maximizing accuracy at convergence, finding winning tickets containing as few parameters as possible, and remaining conservative enough to apply to a range of other experiments.

Figure 17 shows the results of performing the iterative lottery ticket experiment on the conv2 (top), conv4 (middle), and conv6 (bottom) architectures. Since we have not yet selected the pruning rates for each network, we temporarily pruned fully-connected layers at 20% per iteration, convolutional layers at 10% per iteration, and the output layer at 10% per iteration; we explore this part of the hyperparameter space in a later subsection.

For conv2, we selct a learning rate of 0.0002, which has the highest initial validation accuracy, maintains both high validation accuracy and low convergence times for the among the longest, and reaches the fastest convergence times. This learning rate also leads to a 3.3 percentage point improvement in validation accuracy when the network is pruned to 3% of its original size. Other learning rates, such 0.0004, have lower initial validation accuracy (65.2% vs 67.6%) but eventually reach higher absolute levels of validation accuracy (71.7%, a 6.5 percentage point increase, vs. 70.9%,

a 3.3 percentage point increase). However, learning rate 0.0002 shows the highest proportional decrease in convergence times: 4.8x (when pruned to 8.8% of the original network size).

For conv4, we select learning rate 0.0003, which has among the highest initial validation accuracy, maintains high validation accuracy and fast convergence times when pruned by among the most, and balances improvements in validation accuracy (3.7 percentage point improvement to 78.6% when 5.4% of weights remain) and improvements in convergence time (4.27x when 11.1% of weights remain). Other learning rates reach higher validation accuracy (0.0004—3.6 percentage point improvement to 79.1% accuracy when 5.4% of weights remain) or show better improvements in convergence times (0.0002—5.1x faster when 9.2% of weights remain) but not both.

For conv6, we also select learning rate 0.0003 for similar reasons to those provided for conv4. Validation accuracy improves by 2.4 percentage points to 81.5% when 9.31% of weights remain and convergence times improve by 2.61x when pruned to 11.9%. Learning rate 0.0004 reaches high final validation accuracy (81.9%, an increase of 2.7 percentage points, when 15.2% of weights remain) but with smaller improvements in convergence performance, and learning rate 0.0002 shows greater improvements in convergence times (6.26x when 19.7% of weights remain) but reaches lower overall validation accuracy.

We note that, across nearly all combinations of learning rates, the lottery ticket pattern—where convergence times were maintain or decreased and validation accuracy was maintained or increased during the course of the lottery ticket experiment—continued to hold. This pattern failed to hold at the very highest learning rates: convergence times decreased only briefly (in the case of conv2 or conv4) or not at all (in the case of conv6), and accuracy increased only briefly (in the case of all three networks). This pattern is similar to that which we observed with resnet18 in Appendix D: at the highest learning rates, our iterative pruning algorithm fails to find winning tickets.

C.3 Other Optimization Algorithms

C.3.1 SGD

Here, we explore the behavior of the lottery ticket experiment when the conv2, conv4, and conv6 networks are optimized with stochastic gradient descent (SGD) at various learning rates. The results of doing so appear in Figure 18. In general, these networks—particularly conv2 and conv4—proved challenging to train with SGD and Glorot initialization. As Figure 18 reflects, we could not find SGD learning rates for which the unpruned networks matched the validation accuracy of the same networks when trained with Adam; at best, the SGD-trained unpruned networks were typically 2-3 percentage points less accurate. At higher learning rates than those in Figure 17, gradients tended to explode when training the unpruned network; at lower learning rates, the networks often failed to learn at all.

At all of the learning rates depicted, we found winning tickets. In all cases, convergence times initially decreased with pruning before eventually increasing again, just as in other lottery ticket experiments. The conv6 network also exhibited the same accuracy patterns as other experiments, with validation accuracy initially increasing with pruning before eventually decreasing again.

However, the conv2 and conv4 architectures exhibited a different validation accuracy pattern from other experiments in this paper. Accuracy initially declined with pruning before rising as the network was further pruned and eventually matching or surpassing the accuracy of the unpruned network. When they reached this accuracy level, the pruned networks still converged at about the same or fewer iterations than the unpruned network, constituting a winning ticket by our definition. Interestingly, this pattern also appeared for conv6 networks at slower SGD learning rates, suggesting that faster learning rates for conv2 and conv4 than those in Figure 17 might cause the usual lottery ticket accuracypattern to reemerge. Unfortunately, at these higher learning rates, gradients exploded on the unpruned networks, preventing us from running these experiments.

C.3.2 Momentum

Here, we explore the behavior of the lottery ticket experiment when the network is optimized with SGD with momentum (0.9) at various learning rates. The results of doing so appear in Figure 19. In

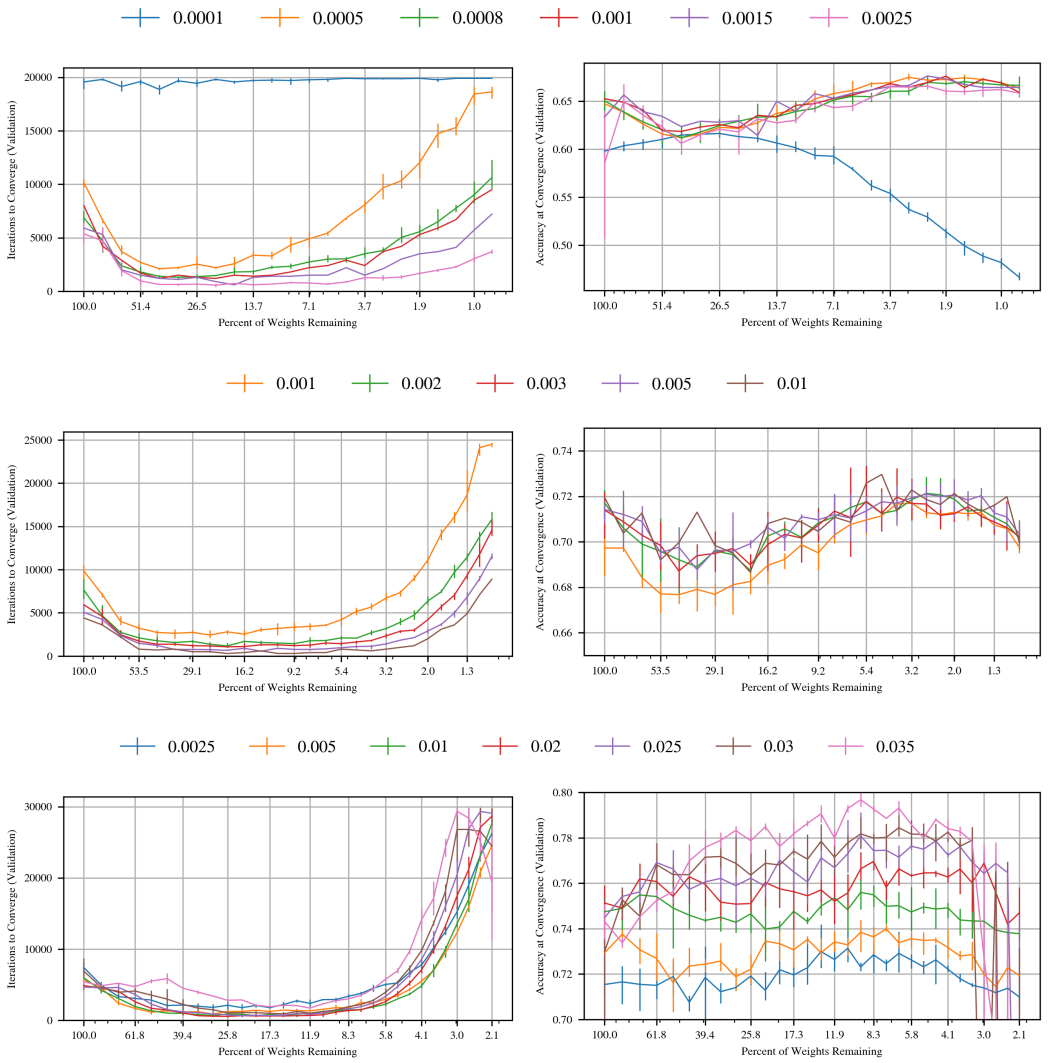

Figure 18: The convergence times and accuracy at convergence-time of the iterative lottery ticket experiment on the conv2 (top), conv4 (middle), and conv6 (bottom) architectures trained using SGD at various learning rates. Each line represents a different learning rate. The legend for each pair of graphs is above the graphs.

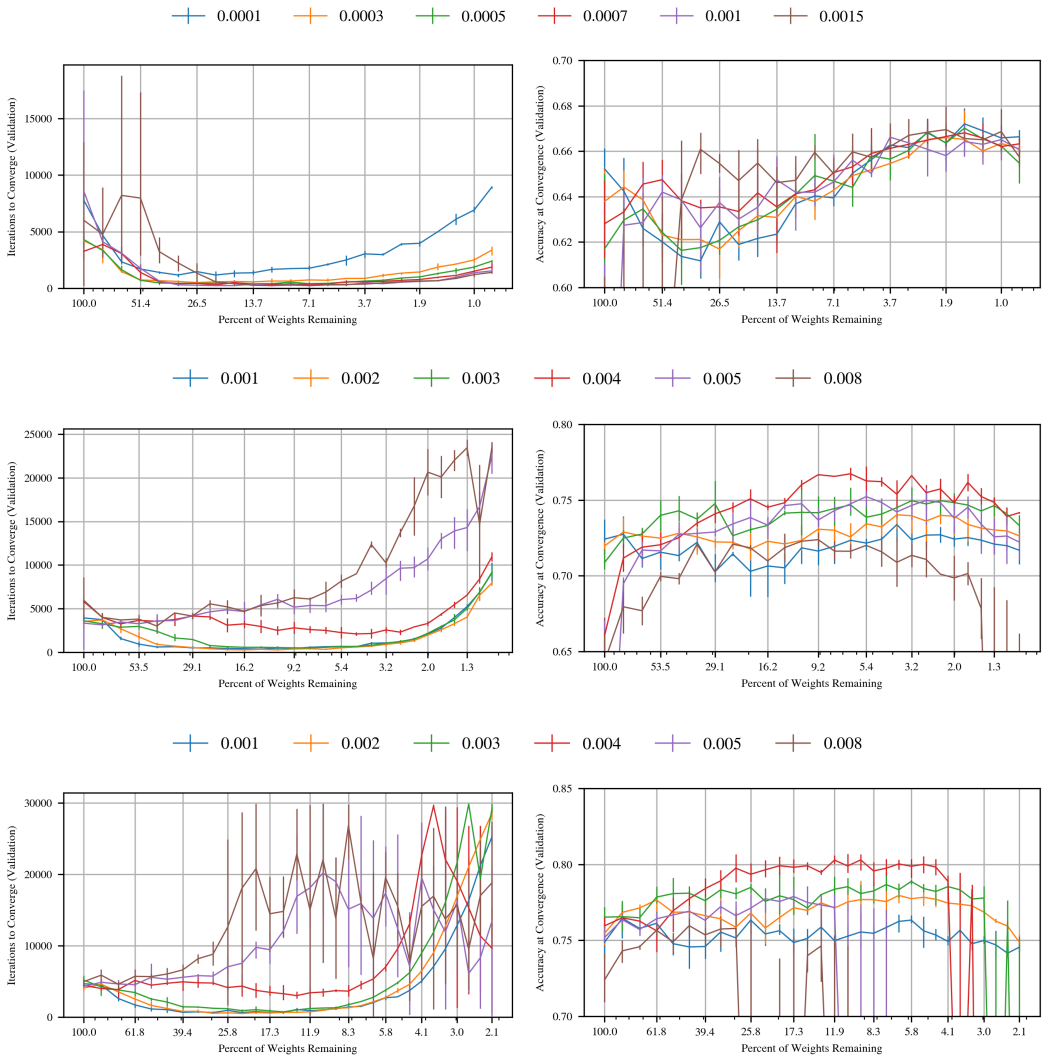

Figure 19: The convergence times and accuracy at convergence-time of the iterative lottery ticket experiment on the conv2 (top), conv4 (middle), and conv6 (bottom) architectures trained using SGD with momentum (0.9) at various learning rates. Each line represents a different learning rate. The legend for each pair of graphs is above the graphs. Lines that are unstable and contain large error bars (large vertical lines) indicate that some experiments failed to learn effectively, leading to very low accuracy and very high convergence times; these experiments reduce the averages that the lines trace and lead to much wider error bars.

general, the lottery ticket pattern continues to apply, with convergence times decreasing and accuracy increasing as the networks are pruned. However, there were two exceptions to this pattern:

1. At the very lowest learning rates (e.g., learning rate 0.001 for conv4 and all but the highest learning rate for conv2), accuracy initially decreased before increasing to higher levels than reached by the unpruned network; this is the same pattern we observed when training these networks with SGD.

2. At the very highest learning rates (e.g., learning rates 0.005 and 0.008 for conv2 and conv4), convergence times never decreased and instead remained stable before increasing; this is the same pattern we observed for the highest learning rates when training with Adam.

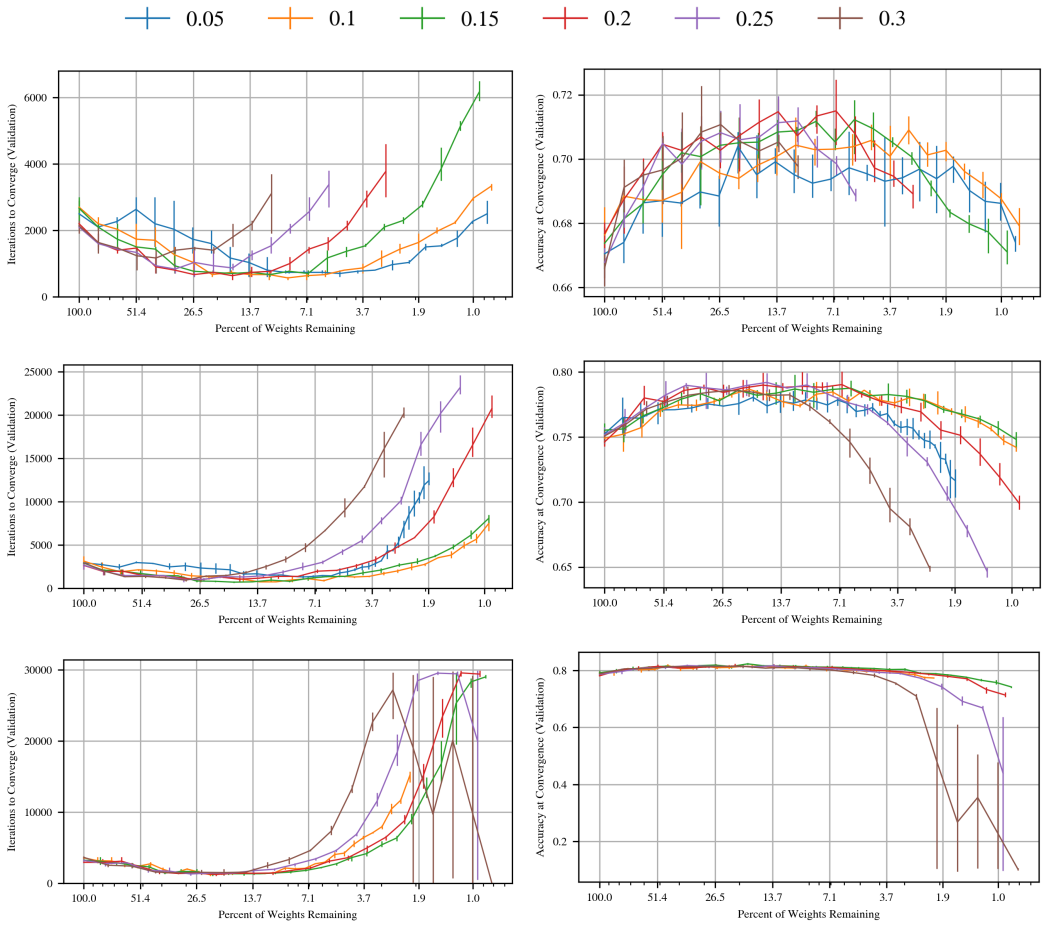

Figure 20: The convergence times and accuracy at convergence-time of the iterative lottery ticket experiment on the conv2 (top), conv4 (middle), and conv6 (bottom) architectures with an iterative pruning rate of 20% for fully-connected layers. Each line represents a different iterative pruning rate for convolutional layers.

C.4 Iterative Pruning Rate

For the convolutional network architectures, we select different pruning rates for convolutional and fully-connected layers. In the conv2 and conv4 architectures, convolutional parameters make up a relatively small portion of the overall number of parameters in the models. By pruning convolutions more slowly, we are likely to be able to prune the model further while maintaining performance. In other words, we hypothesize that, if all layers were pruned evenly, convolutional layers would become a bottleneck that would make it more difficult to find lower parameter-count models that are still able to learn. For conv6, the opposite may be true: since nearly two thirds of its parameters are in convolutional layers, pruning fully-connected layers could become the bottleneck.

Our criterion for selecting hyperparameters in this section is to find a combination of pruning rates that allows networks to reach the lowest possible parameter-counts while maintaining validation accuracy at or above the original accuracy and convergence times at or below the original convergence time.

Figure 20 shows the results of performing the iterative lottery ticket experiment on conv2 (top), conv4 (middle), and conv6 (bottom) with different combinations of pruning rates.

According to our criteria, we select an iterative convolutional pruning rate of 10% for conv2, 10% for conv4, and 15% for conv6. However, across all convolutional pruning rates, the lottery ticket pattern continued to appear.

C.5 LEARNING RATES (DROPOUT)

In order to train the conv2, conv4, and conv6 architectures with dropout, we repeated the exercise from the previous Subsection to select appropriate learning rates. Figure 17 shows the results of performing the iterative lottery ticket experiment on conv2 (top), conv4 (middle), and conv6 (bottom) with dropout and Adam at various learning rates. A network trained with dropout takes longer to converge, so we trained each architecture for three times as many iterations as in the experiments without dropout: 60,000 iterations for conv2, 75,000 iterations for conv4, and 90,000 iterations for conv6. We iteratively pruned these networks at the rates determined in Section C.4.

The conv2 network proved to be difficult to consistently train with dropout. The top right graph in Figure 21 contains wide error bars and low average accuracy for many learning rates, especially early in the lottery ticket experiments. This indicates that some or all of the training runs failed to learn; when they were averaged into the other results, they produced the aforementioned pattern in the graphs. At learning rate 0.0001, none of the three trials learned productively until pruned to more than 26.5%, at which point all three trials started learning. At learning rate 0.0002, some of the trials failed to learn productively until several rounds of iterative pruning had passed. At learning rate 0.0003, all three networks learned productively at every pruning level. At learning rate 0.0004, one network occasionally failed to learn. We selected learning rate 0.0003, which seemed to allow networks to learn productively most often while achieving among the highest initial accuracy.

It is interesting to note that networks that were unable to learn at a particular learning rate (for example, 0.0001) eventually began learning after several rounds of the lottery ticket experiment (that is, training, pruning, and resetting repeatedly). It is worth investigating whether this phenomenon was entirely due to pruning (that is, removing any random collection of weights would put the network in a configuration more amenable to learning) or whether training the network provided useful information for pruning, even if the network did not show improved accuracy.

For both the conv4 and conv6 architectures, a slightly slower learning rate (0.0002 as opposed to 0.0003) leads to the highest accuracy on the unpruned networks in addition to the highest sustained accuracy and fastest sustained convergence times as the networks are pruned during the lottery ticket experiment.

With dropout, the unpruned conv4 architecture reaches an average validation accuracy of 77.6%, a 2.7 percentage point improvement over the unpruned conv4 network trained without dropout and one percentage point lower than the highest average validation accuracy attained by a winning ticket. The dropout-trained winning tickets reach 82.6% average validation accuracy when pruned to 7.6%. Convergence times improve by up to 1.58x (when pruned to 7.6%), a smaller improvement than then 4.27x achieved by a winning ticket obtained without dropout.

With dropout, the unpruned conv6 architecture reaches an average validation accuracy of 81.3%, an improvement of 2.2 percentage points over the accuracy without dropout; this nearly matches the 81.5% average accuracy obtained by conv6 trained without dropout and pruned to 9.31%. The dropout-trained winning tickets further improve upon these numbers, reaching 84.8% average validation accuracy when pruned to 10.5%. Improvements in convergence times are less dramatic than without dropout: a 1.5x average improvement when the network is pruned to 15.1%.

At all learning rates we tested, the lottery ticket pattern generally holds for accuracy, with improvements as the networks are pruned. However, not all learning rates show the decreases in convergence times. To the contrary, none of the learning rates for conv2 show clear improvements in convergence times as seen in the other lottery ticket experiments. Likewise, the faster learning rates for conv4 and conv6 maintain the original convergence times until pruned to about 40%, at which point convergence times steadily increase.

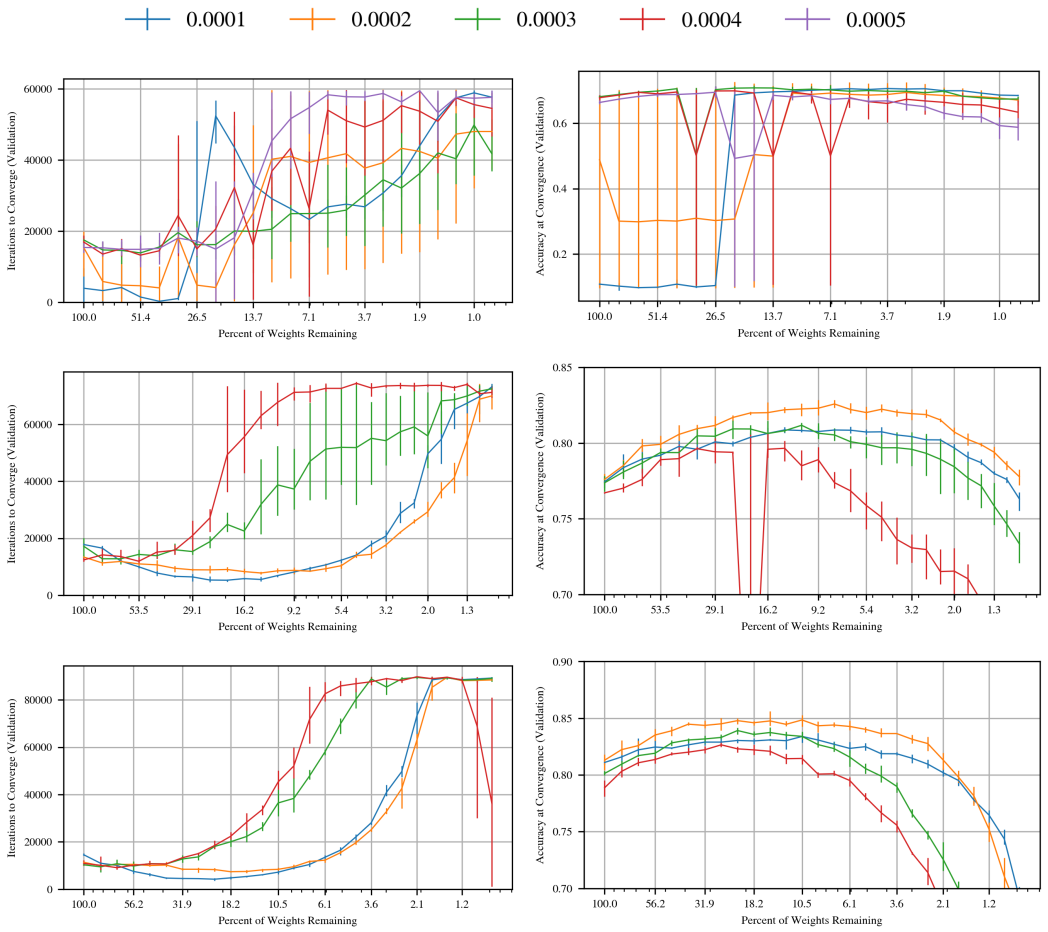

Figure 21: The convergence times and accuracy at convergence-time of the iterative lottery ticket experiment on the conv2 (top), conv4 (middle), and conv6 (bottom) architectures trained using dropout and the Adam optimizer at various learning rates. Each line represents a different learning rate.

Network	resnet18
Convolutions	16, 3x[16, 16] 3x[32, 32] 3x[64, 64]
FC Layers	avg-pool, 10
All/Conv Weights	274K / 270K
Iterations	30K
Optimizer	SGD 0.1-0.01-0.001 (momentum 0.9)
Pruning Rate	10%

Figure 22: An addendum to Figure 2 for the resnet18 architecture (He et al., 2016). Brackets denote residual connections around layers.

D WINNING TICKETS IN RESIDUAL NETWORKS

In this Appendix, we investigate the lottery ticket hypothesis as applied to residual networks, specifically resnet18 (see Figure 22) on CIFAR10. In Section D.1, we describe the details of the experiment we conduct with resnet18. In Section D.2, we describe the hyperparameter search process for our experiments. In Section D.3, we display and discuss the results of this experiment.

We find that, at the learning rate used in the paper that introduced resnet18 (He et al., 2016), iterative pruning does not find winning tickets. However, at slower learning rates for both SGD and SGD with momentum, we find winning tickets. These winning tickets nearly match the accuracy of the unpruned network with the hyperparmeters in (He et al., 2016) when pruned to 33% or less of the original network size. This suggests a number of questions for future work, including characterizing whether or how the lottery ticket hypothesis applies beyond the feedforward networks we study in this paper and whether other strategies for finding winning tickets will be work for resnet18 at a higher learning rate.

D.1 EXPERIMENTAL METHODOLOGY

The resnet18 architecture was first introduced by He et al. (2016). The architecture comprises 20 total layers as described in Figure 22: a convolutional layer followed by nine pairs of convolutional layers (with residual connections around the pairs) and a fully-connected output layer.

We follow the experimental design of He et al. (2016):

- We divide the training set into 45,000 training examples and 5,000 validation examples. We use the validation set to select hyperparameters in Section D.2 and the test set to evaluate in Section D.3.
- We augment training data using random flips and random four pixel pads and crops.
- We use a batch size of 128.
- We use batch normalization.
- We use weight decay of 0.0001.
- We use three stages of training at decreasing learning rates. Our stages last for 20,000, 5,000, and 5,000 iterations each, shorter than the 32,000, 16,000, and 16,000 used in He et al. (2016). Since each of our iterative pruning experiments requires training the network 30 times consecutively, we select this abbreviated training schedule to make it possible to explore a wider range of hyperparameters.

We use Gaussian Glorot initialization.

Since more than 98% of the weights in the network are in convolutions, we prune convolutions at a rate of 10% per iteration. We do not prune the 2560 parameters used to downsample residual connections or the 640 parameters in the fully-connected output layer, as they comprise such a small portion of the overall network.

For this experiment, we consider three sets of learning rate hyperparameters:

1. The learning rate hyperparameters from He et al. (2016): SGD with momentum (0.9) with learning rates of 0.1, 0.01, and 0.001 at the three training stages.
2. An alternate set of learning rates for SGD with momentum that are designed to find small, high-accuracy winning tickets.
3. An alternate set of learning rates for SGD (without momentum) that are designed to find small, high-accuracy winning tickets.

In Section D.2, we describe the hyperparameter search process used to select the rates for (2) and (3). For the hyperparameter search, we prune convolutions at 20% per iteration in order to make it possible to explore the space of hyperparameters efficiently.

D.2 LEARNING RATE

In He et al. (2016), the network is trained with stochastic gradient descent (SGD) with momentum (0.9) at a learning rate of 0.1 for the first training stage, 0.01 for the second training stage, and 0.001 for the third training stage. We found that these hyperparameters produce the highest accuracy on the unpruned network of any hyperparameters we explored. However, our method for finding winning tickets (iterative pruning and resetting back to the original initalizations) is not successful with these hyperparameters. In this Subsection, we search for hyperparameters for SGD with and without momentum for which our method finds winning tickets.

We do by greedily selecting the learning rates that produce the highest-accuracy winning tickets at the smallest network sizes at the end of each stage of pruning. In other words, we select the learning rate for the first stage of training by training the network only for the first stage at various learning rates. Once we select a learning rate for the first stage, we train for the first two stages with various second-stage learning rates. We follow the same procedure for the third stage once we have selected learning rates for the first two stages.

D.2.1 SGD

Here, we select learning rates for the lottery ticket experiment on resnet18 when the network is optimized with SGD. Figure 23 shows the results of performing the lottery ticket experiment on resnet18 at various learning rates. The top row shows the convergence times and accuracy when training the network only for the first stage at various learning rates.The second row shows the convergence times and accuracy when training the network for the first two stages with our selected rate for the first stage (0.08) and various rates for the second stage. The third row shows the convergence times and accuracy when training the network for all three stages with our selected rates for the first two stages (0.08, 0.008) and various rates for the third stage.

In the first stage, learning rates between 0.04 and 0.1 show the typical lottery ticket pattern and reach accuracy levels above that of the original network until the network has been pruned to approximately 10%. At learning rates above 0.16, accuracy never increases as the network is iteratively pruned. We select a rate in the middle of the range of rates that produce improvements in accuracy: 0.08.

In the second stage, the range of learning rates we consider all evidence the lottery ticket pattern for accuracy. Since the second stage is a fine-tuning step that builds on the first stage, it is unsurprising that it mimics the accuracy pattern we observe for our selected learning rate during the first stage. We select learning rate 0.008 for the second stage, although there are many valid learning rates from among those we considered.

The third stage behaves in the same manner as the second stage for the same reasons. We select learning rate 0.0012, but there are many choices that result in similar behavior.

D.2.2 MOMENTUM

Here, we select learning rates for the lottery ticket experiment on resnet18 when the network is optimized with SGD with momentum (0.9), the same framework as used in He et al. (2016). The results of doing so are in Figure 24, which has the same structure as Figure 23.

In the first stage, learning rates between 0.006 and 0.01 produce the highest accuracy sustained to the smallest winning ticket sizes. Lower learning rates result in lower accuracy. Higher learning rates (such as 0.1, the rate used in He et al. (2016)) never lead to higher accuracy as the network is pruned. We select learning rate 0.008, which is in the middle of this range. Note that this learning rate is nearly an order of magnitude lower than the first stage learning rate in He et al. (2016). This learning rate produces peak validation accuracy of about 88%, about three percentage points higher than the validation accuracy learning rate 0.1 produces on the unpruned network. However, this accuracy advantage eventually disappears during the further pruning stages.

In the second stage, the range of learning rates we consider all evidence the lottery ticket pattern for accuracy. As with SGD, the second stage is a fine-tuning step, so it is unsurprising that it reflects the accuracy pattern of the first stage. We select learning rate 0.0008, which is in the middle of the range we consider.

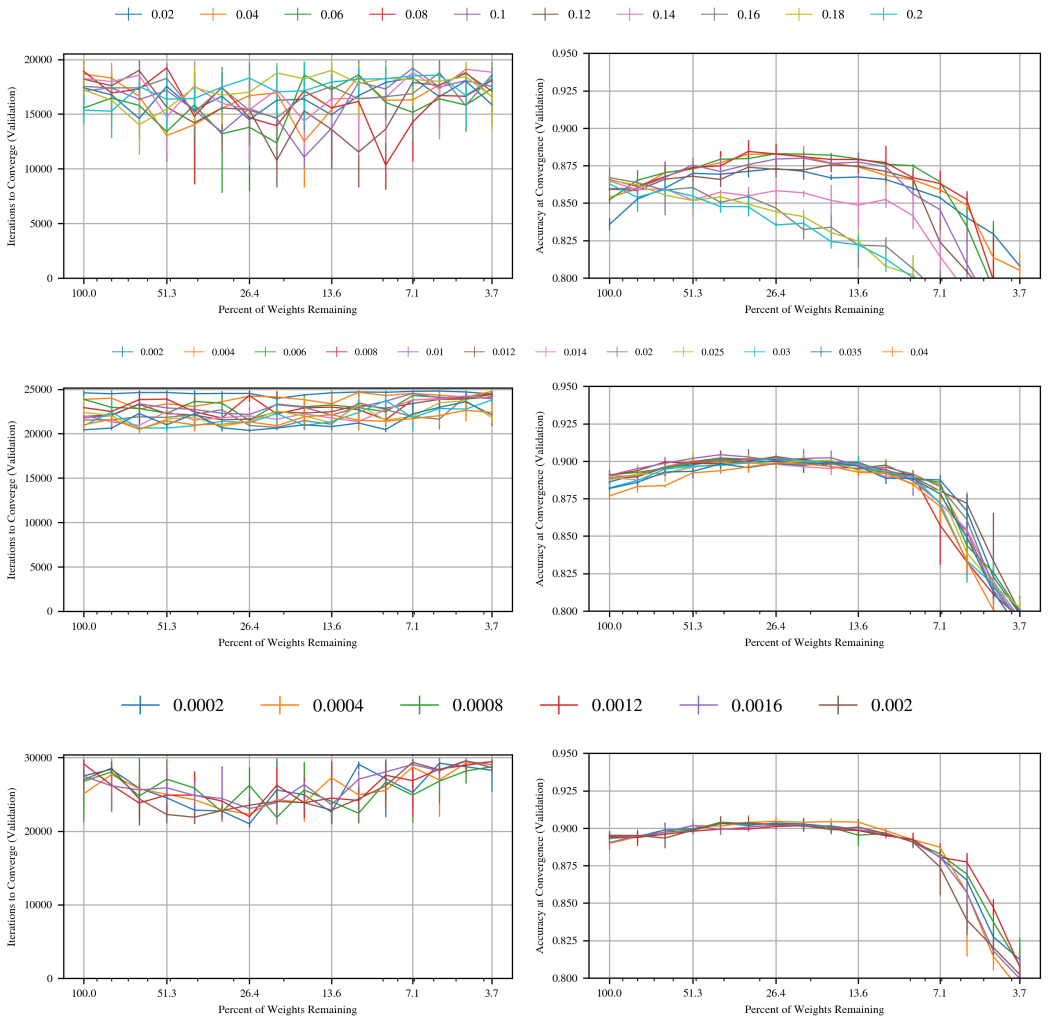

Figure 23: The convergence times and accuracy at convergence-time of the iterative lottery ticket experiment on the resnet18 architecture as trained with SGD. The top row shows the convergence times and accuracy when training the network only for the first stage at various learning rates. The second row shows the convergence times and accuracy when training the network for the first two stages with our selected rate for the first stage (0.08) and various rates for the second stage. The third row shows the convergence times and accuracy when training the network for all three stages with our selected rates for the first two stages (0.08, 0.008) and various rates for the third stage. Each line corresponds to a particular learning rate, and the legends are above the corresponding graphs.

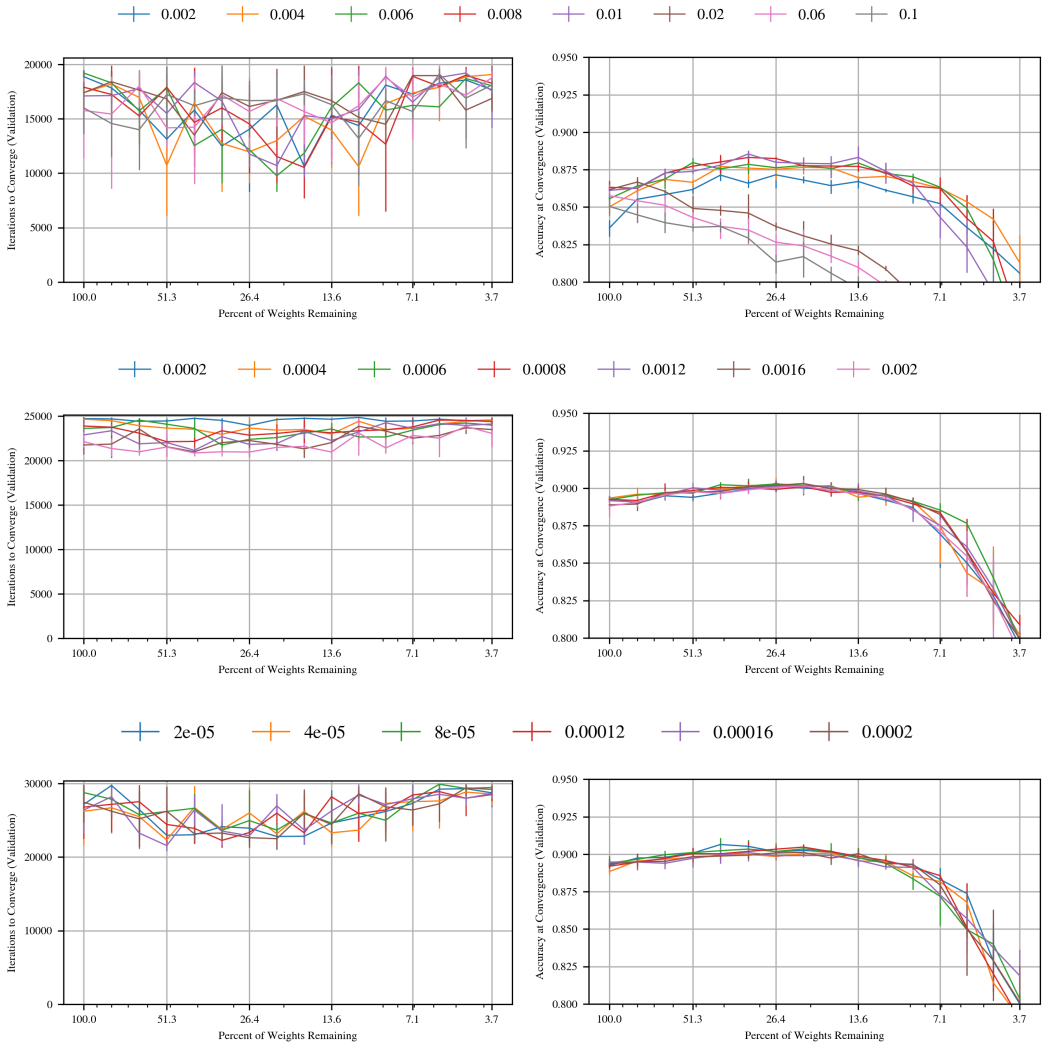

Figure 24: The convergence times and accuracy at convergence-time of the iterative lottery ticket experiment on the resnet18 architecture as trained with SGD with momentum (0.9). The top row shows the convergence times and accuracy when training the network only for the first stage at various learning rates. The second row shows the convergence times and accuracy when training the network for the first two stages with our selected rate for the first stage (0.008) and various rates for the second stage. The third row shows the convergence times and accuracy when training the network for all three stages with our selected rates for the first two stages (0.008, 0.0006) and various rates for the third stage. Each line corresponds to a particular learning rate, and the legends are above the corresponding graphs.

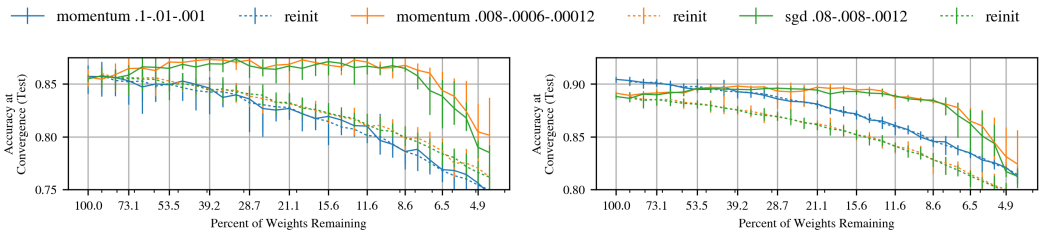

Figure 25: Test accuracy for the resnet18 model after 20,000 (left) and 30,000 (right) iterations of training as it is iteratively pruned. Dashed liens are the random reinitialization experiment.

The third stage has little, if any impact on accuracy. Since we have chosen learning rates that are an order of magnitude slower than He et al. (2016), this third stage likely learns at too slow of a rate to tangibly influence validation accuracy. We select 0.00012, which leads to accuracy representative of the range of learning rates we considered.

D.3 Experimental Results

Here, we evaluate the result of iteratively pruning resnet18 using both our chosen hyperparameters and those from He et al. (2016). Figure 25 shows the test accuracy at the end of 20,000 (left) and 30,000 (right) iterations as we iteratively prune the resnet by 20% per iteration. Using the hyperparameters from He et al. (2016) (momentum 0.9 with learning rates 0.1, 0.01, and 0.001), the network reaches an average test accuracy of 90.3% without pruning. Iterative pruning steadily reduces accuracy, and the lottery ticket experiment produces networks that perform no better than when they are randomly reinitialized.

At slower learning rates, the lottery ticket pattern reemerges and we find winning tickets. When the momentum learning rates are adjusted to 0.008, 0.0006, and 0.00012, average test accuracy when the network is pruned to 29.1% of its original size reaches 89.8%. Likewise, when the network is trained with SGD at learning rates 0.08, 0.008, and 0.0012, average test accuracy when the network is pruned to 32.2% of its original size reaches 89.7%. With both our optimization schemes, the networks continue to surpass their original accuracies (89.0% and 88.9%) when pruned by more than 88%. When randomly reinitialized, the winning tickets steadily decrease in accuracy. In sum, with appropriate hyperparameters, we find winning tickets more than 67% smaller than the unpruned network that come within half a percentage point of the test accuracy of the original learning rate.

These experiments demonstrate that the iterative pruning method for generating winning tickets is sensitive to learning rates on resnet18. It can find winning tickets at lower learning rates, but it fails to do so at the higher rate from He et al. (2016). Our experiments in Section D.2 confirm this behavior for the highest SGD and momentum learning rates for resnet18.

There are several possible explanations for this behavior.

- At higher learning rates, the iterative pruning method for finding winning tickets may not be effective on residual networks. As discussed in our future work, it is worth exploring other methods for finding winning tickets, particularly those with the residual context in mind.
- Throughout this paper, we have optimized winning tickets using the same strategy as used to optimize the original network. It is possible that iterative pruning does uncover subnetworks of resnet18 that comprise winning tickets, but that they need to be optimized in a different fashion than the original network, e.g., altering the learning rate or re-scaling the weights.
- As framed, the lottery ticket hypothesis applies only to feedforward neural networks. It is possible that it does not apply in the context of residual networks.