Merging Lottery Tickets

1st Hayden Coffey
EECS Dept.
University of Tennessee, Knoxville
Knoxville, TN
hcoffey1@vols.utk.edu

2nd Carl Edwards
EECS Dept.
University of Tennessee, Knoxville
Knoxville, TN
cedwar45@utk.edu

3rd Landen McDonald
MABE Dept.
University of Tennessee, Knoxville
Knoxville, TN
lmcdon14@vols.utk.edu

***Abstract*—Sparse neural networks have the capability to drastically decrease the storage requirement and computational intensity of making predictions while also maintaining accuracy. Training such sparse networks from the start, however, has been proven to be difficult without proper initialization.**
The lottery ticket theorem states that given a dense neural network, it is possible to find a subnetwork which learns in commensurate time or better, with commensurate accuracy or better, and fewer parameters. By utilizing traditional iterative pruning techniques of near-zero weights and then retraining from the starting initializations, it is possible to produce "winning ticket" subnetworks consisting of 10-15% of the original weights in approximately the same number of training iterations.
To further improve upon the algorithms currently implemented for finding such winning tickets, we propose combining the initializations of multiple winning tickets for better training performance and higher test accuracy. We explored multiple ticket combining algorithms in an effort to create more accurate networks with comparable sparsity and training times to the individual constituent winning tickets.

I. Introduction

Many neural networks are wildly over-parameterized and contain vastly more connections than are necessary for obtaining accurate predictions. These networks require a means of compression in order to remain computationally and energy efficient when making inferences. However, it has been shown that training sparse network architectures with only a fraction of the number of parameters from scratch often yields far less accurate results than the corresponding larger networks. In 2018, Frankle & Carbin published shocking results [1] that presented a very basic algorithm with the capability of producing extremely sparse subnetworks within larger networks that are not only trainable from scratch, but also often outperform their larger counterparts.

This key insight was dubbed "The Lottery Ticket Hypothesis" which claims that there exists some sparse subnetwork in a larger dense network that, when trained in isolation with the same original initializations used to find that subnetwork, can achieve comparable accuracy to the original larger network in at most the same number of training iterations [1]. These sparse networks have been generated with $\geq$95% of their original weights pruned before experiencing diminishing accuracy. Such extremely sparse networks are known as "Winning Tickets" due to the fact that they have effectively won the initialization lottery.

The approach utilized for generating such tickets in [1] was actually quite simple and elegant. They began by training a large network. Once trained, they then removed all of the weights in the network that did not have a magnitude greater than some threshold. The weights of this slightly sparser network were then reset to their original initialization values prior to any training. This iterative process was then repeated until the desired level of sparsity was obtained.

The underlying mechanics of this technique has been brought under extreme scrutiny in recent research [2]–[4], in which the merit of keeping the original initialization values and applying proper masks have been questioned. With such a simple technique for generating extraordinarily sparse networks that are trainable from scratch, the next logical question to ask is how we can further improve upon these findings. We believe that there are means of building upon this research in order to generate comparably sparse networks with improved prediction accuracy that can be trained from scratch in nearly the same number of iterations. Specifically, we think it may prove beneficial to combine the initializations of multiple lottery tickets into a single sparse network that retains some of the learned information from each ticket.

In this document, we investigate multiple ticket combining techniques in an effort to generate measurable accuracy improvements in the sparse networks generated using Frankle & Carbin's lottery ticket approach. The two approaches that we have chosen to employ for combining these network initializations are summing the pruned initial weights onto a "dominant" lottery ticket and averaging the pruned initial weights. The experiments that we conducted with these two approaches yielded interesting results that are further expanded upon below.

II. Previous Work

The original paper in which the Lottery Ticket Hypothesis was coined was Frankle & Carbin's 2018 paper mentioned previously [1]. In this work, unequivocal evidence was shown for the existence of extremely sparse networks that retain all of the predicting accuracy of their fully-connected counterparts, even when up to 99% of their weights had been masked. Additionally, such networks were shown to achieve this accuracy in no more than the same number of training

iterations as the dense network required as long as the original initializations were preserved. This led to the realization that a network's "structure alone does not define a winning ticket's success," but its ability to win the initialization lottery is also critical for peak performance. Moreover, they showed that the approach of iterative pruning, only removing a small fraction of the weights before rewinding the initialization and training again, yielded far better results than one-shot pruning methods that sought to achieve the same level of sparsity in a single iteration.

Useful observations of the underlying mechanics of the lottery ticket approach are further explored in Uber Engineering's 2019 paper [2]. In this paper, numerous combinations and criteria were employed for generating both the initial weight distributions and masks utilized for initializing new sparse networks. They ultimately were able to uncover a few methods that generated superior results when compared to the original lottery ticket approach used by Frankle & Carbin. Furthermore, it was also observed that the specific value of the re-initialized weights were not quite as important as maintaining their original sign was. This was an encouraging discovery that helped fuel our desire to test combining methods for multiple tickets since the exact initializations need not be retained for comparable training and testing results. Finally, Zhou et al. also discovered that the untrained re-initialization of a masked subnetwork often yielded uncharacteristically high predicting power, even when untrained. This led to the conclusion that "masking" or setting pruned weights to zero in itself is a form of training.

The works of Desai et al. (2019) and Liu et al. (2018) further investigate the capability of transfer learning with winning lottery tickets to test their ability to effectively generalize [3], [4]. In addition, both of these papers concluded that, in fact, the original initialization values were not necessary for producing accurate sparse networks, but they do help lower variance in testing accuracy. The experiments conducted in [4] also explored different methods for structured pruning that, unlike the lottery ticket approach used by Frankle & Carbin, produced sparse architectures (masks) that were far more important than the weight initializations when attempting to preserve accuracy when training from scratch.

Although lottery tickets produce sparse networks that require far less memory and computation than a fully-connected network when making inferences, the iterative training and pruning process of generating these tickets is usually extremely slow. However, experiments conducted in this 2019 paper [5] elucidate the existence of so-called "Early-Bird Tickets" that can be found rather early in the iterative training process. These subnetworks are generated under the principle that the general structure of a neural network is usually created in only the first few steps of training, whereas the specific weights needed for accuracy take longer to be divulged. When finding the proper initializations and masks for lottery tickets however, the exact weights are not necessary until the final iterations of training. With aggressive learning rates and training schemes, functional lottery tickets can be generated in only a fraction of the time necessary to create the "ground-truth" lottery tickets proposed by Frankle & Carbin. This ultimately means that creating accurate sparse networks are not only more efficient than fully-connected networks during final training and making inferences, but also throughout the entire training and ticket generation process.

Finally, there have been extensive strides made in evaluating the performance of sparse networks. The work conducted by Venkatesh et al. earlier this year (2020) generated a number of performance-predicting metrics for lottery tickets that can evaluate existing networks as well as help generate new networks with improved generalization and decreased overfitting. One such metric seeks networks that not only produce accurate testing results, but also show improved confidence in correct results and decreased confidence in incorrect results by probing their softmax output layers. Along with many other calibration criterion, this paper presented a comprehensive suite of evaluation tools for ensuring the robustness of sparse lottery ticket networks of various kinds.

III. Technical Approach

The ultimate objective of this experiment is to discover novel means of generating high-performance, sparse networks by assigning proper weight initializations to heavily pruned fully-connected networks. As mentioned previously, by means of iterative training, pruning, rewinding initializations to their original values, and retraining, it is possible to generate extremely sparse networks with less than 5% of the weights in a corresponding fully-connected network while also incurring little to no performance degradation. The problem that we seek to solve is in what ways we can make these sparse networks perform even better by tweaking their initializations and masks.

We propose combining multiple lottery tickets generated in the traditional approach used by Frankle & Corbin [1] in order to accomplish this task. The original network architecture utilized to generate our winning lottery tickets is the Lenet-300-100 architecture. This fully-connected network is then iteratively pruned until we obtain a sparsely-connected network with a fraction of the original size. The negative log-likelihood loss function is used in conjunction with the softmax activation function on our output layer in order to train and prune the weights of our networks. After each iteration of training, we prune the $p^{\frac{1}{n}}\%$ of smallest weights from the network, where p is the final percentage of original weights desired and n is the number of pruning iterations used. A mask of binary values is then applied to the initializations saved from the very beginning of training corresponding to the near-zero weights that should be removed from the network.

Once we have generated a number of individual winning tickets whose performance is on par with the original fully-connected network, we sought to combine networks for improved test accuracy and training speeds. In our experiments, we investigate two unique methods for combining the weights of multiple lottery tickets. The first method that we em-

ployed was the Dominant Ticket Merging method (DTM). This merging algorithm picks a hierarchy of dominant tickets ranked from 1-N. It then merges tickets in ascending order of rank from the lowest rank (N) to the most dominant (1). If the weight of the higher ranked ticket has not been pruned (i.e. is not zero), the summation of the lower ranked tickets will be added to the higher ranked ticket's weights. This summation will then become the new lower ranked ticket and will be similarly combined with the proceeding higher ranked ticket. This is repeated until the N-1 lower ranked tickets are combined with the most dominant ticket. With this method, the mask of the most dominant ticket is preserved, while some of the information from the subservient tickets is transferred to that dominant ticket in an effort to improve the initialization of the combined ticket. An example of a DTM combination with N=3 is shown below in Fig. 1.

Dominant Ticket	Second ticket	Third ticket	Merged Initialization
0	0	0.15	0
0	0.25	0	0
0.25	0.5	0	0.75
0.5	0	0	0.5
0	0	0	0
0	0.75	0.25	0
0.75	0	0.5	0.75
0	0.75	0	0
0	0	0.5	0
0.5	0.5	0	1.0
.	.	.	.
.	.	.	.
.	.	.	.
0	0	0	0

Fig. 1: Example of Dominant Ticket Merging (DTM).

One unique aspect of this merging technique is displayed in the 7^{th} merged weight in Fig. 1. Since the second ticket is more dominant than the third ticket and the 7^{th} weight in the second ticket has been pruned, the weight from the third ticket is neglected from the final summation. This hierarchy that has been created implies that not only do the combinations of our tickets matter for the DTM technique, but also the permutations.

The second combination method that we have implemented is the Average Ticket Merging method (ATM). This technique simply takes an average of all of the non-zero weights at each weight location. The idea behind implementing this merging method is that no ticket will retain a higher precedent over the others, and more information can be transferred between the tickets upon combination. An example of such a combining technique is shown below in Fig. 2.

First Ticket	Second ticket	Third ticket	Merged Initialization
0	0	0.15	0.15
0	0.25	0	0.25
0.25	0.5	0	0.375
0.5	0	0	0.5
0	0	0	0
0	0.75	0.25	0.5
0.75	0	0.5	0.625
0	0.75	0	0.75
0	0	0.5	0.5
0.5	0.5	0	0.5
.	.	.	.
.	.	.	.
.	.	.	.
0	0	0	0

Fig. 2: Example of Average Ticket Merging (ATM).

As displayed in Fig. 2 above, the ATM approach combines each weight from the constituent tickets into the final merged ticket. This leaves us with a slightly less sparse network, but retains all of the structural information acquired when generating each of the tickets being combined. Moreover, since each weight is combined, this method is not concerned with the order in which tickets are merged. When testing all possible orientations of this method, we simply have to test the number of combinations of lottery tickets rather than permutations as we did for the DTM approach.

IV. Dataset and Implementation

For these experiments, we chose to train on the MNIST and Fashion MNIST datasets. These relatively simple datasets of images are not only conducive to rapid training, but also allow for clear comparisons in network quality and training speed. Since the sparse pruning technique is often quite computationally intensive and requires numerous iterations of retraining (often 15 or more), we chose to implement this experiment on smaller datasets that would allow us to garner meaningful results quickly and effectively. The code is also implemented to run the experiments on the CIFAR10 and CIFAR100 datasets, but we chose to stick with MNIST for our application.

Both MNIST datasets have 60000 examples of 28x28 pixel grayscale images. Torchvision automatically divides

these into 50000 training examples and 10000 test examples. The network architecture that we use is the Lenet-300-100 architecture [6]; it is a simple, 3-layer fully-connected network with 300, 100, and 10 neurons respectively in the layers. We chose to stick with the default hyper-parameters and Adam optimizer used for this experiment for each test in order to maintain a direct comparison between the performances of each model.

Those parameters were as follows:

- batch size = 60
- epochs = 20/iteration
- learning rate = 1.2e-3
- betas = 0.9, 0.999
- eps = 1e-8
- weight decay = 1e-4
- prune percent = 10%/iteration
- iterations = 22
- near-zero weights $\leq$ 1e-6

The training time for generating winning tickets varied greatly depending on the hardware used, but in general 22 iterations of pruning and retraining took roughly 2-3 hours on a CUDA-capable GPU and between 7-8 hours on a CPU. Once the tickets had been generated, combining and retraining for a single iteration took on the order of 15 minutes.

V. Experiments and Results Analysis

A. Testing Design

We created 7 MNIST tickets and 5 Fashion-MNIST tickets. From these, we arbitrarily selected four of each to test with. The reduced selection was done in order to make testing computationally feasible. Computing each of the tests involved training the network, so in total it took more than 18 hours on a GPU. Increasing the number of tickets involved by 1 would have exponentially increased this value. Since DTM is order sensitive, permutations of size two and three of the four tickets for MNIST and Fashion-MNIST were evaluated under DTM. A size four permutation was determined to be too computationally expensive to carry out under the workload. When testing four tickets merged under DTM, the four tickets were tested in one arbitrary order, and then tested in the reverse order. Note: our script had a bug which caused us to run dominant merging tests with extra tickets for permutations of size 2 and 3 in MNIST. ATM is not order sensitive, and as a result, combinations of size two, three, and four were tested for the four tickets for each data set. We extracted parameters from each of these trails and then we use the average values to compare different techniques.

B. Learning Time - Exponential Fit

We decided to measure learning time by fitting an exponential function to the epoch vs. accuracy data.

$$f(x) = ae^{-c(x-b)} + d$$

Params Remaining	Avg. Best Accuracy	Avg. C Parameter	# Tickets
	MNIST		
100%	96.688	3.041	4
53.2%	96.908	3.516	4
47.9%	96.932	3.740	4
31.5%	96.948	3.928	4
20.7%	96.928	3.405	4
11.1%	96.835	3.442	4
	FMNIST		
100%	85.555	2.875	4
53.2%	85.63	3.685	4
47.9%	85.927	3.705	4
31.5%	85.985	3.382	4
20.7%	85.915	3.623	4
11.1%	85.973	4.092	4

TABLE I: This table shows the average results of tickets with different percent of parameters remaining. We compare our merging algorithms to these values as a baseline.

We fit the data to the above equation using SciPy [7]'s curve_fit function with an initial guess of $[-1, 1, 4, 95]$, as shown in Figure 3. Then, we use the optimized value of c to compare the speed the algorithm learns. A larger "C parameter" indicates a faster learning network.

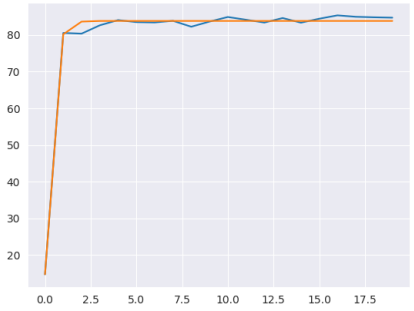
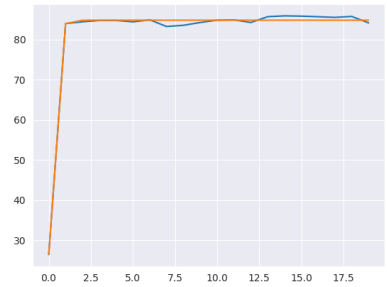


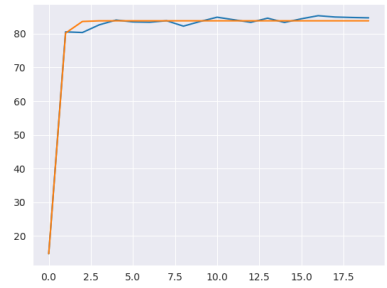
Fig. 3: Epochs vs. Accuracy on one of the fashion MNIST tickets. The exponential fit (orange) attempts to model how fast the network improved (blue).

C. Ticket Results

In Table I, we can see that the tickets we extract behave as predicted in the paper. The accuracy for both MNIST and Fashion-MNIST stays similar up to a 90% reduction of parameters, and it even improves, with a maximum value of 31% parameters remaining. Based on the "C parameter", the network training speed appears to be faster for smaller networks. This is less clear for MNIST, which may be too easy of a dataset to compare with. On FMNIST, the parameter increases from under 3.0 for the full network to above 4.0 for the 11% network. This indicates that the parameter is correctly measuring the speed of training as well, since [1] reports increased training speed. This difference can be seen in Figure 4, where the pruned model reaches a higher accuracy faster.



(a) 11.1% parameters epoch vs. accuracy



(b) 100% parameters epoch vs. accuracy

Fig. 4: The pruned 11.1% ticket reaches ~83% accuracy after epoch 1, but the 100% parameter ticket is slower and only reaches 80% accuracy. These tickets were trained on Fashion-MNIST. The 11.1% has C value 3.442 and the 100% has C value 3.041.

D. Merging Results

We found that ticket parameter masks tended to be mostly disjoint for lower numbers of tickets. For example, when merging Fashion-MNIST with tickets A, B, and C, we find that the number of parameters in the resulting network is 29.01%. Other tickets were also very close to this value, which makes sense if we assume that the active parameters are distributed uniformly throughout the thousands of model parameters. Without any overlap, three merged tickets would be pruned to a size of 33.3%. This value being close to 29 indicates that the tickets were relatively disjoint. This was even more so the case with only 2 tickets being merged.

1) Dominant Ticket Merging: Table II displays the results of our experiments using dominant ticket merging. More MNIST trials were conducted than originally planned due to the bug encountered in the first rendition of the testing script. The issue was resolved, leading to the smaller number of trials for the tickets in the FMNIST data set. From the table, we can see that the results are difficult to interpret. On the MNIST dataset, merging 2 tickets learns the fastest, but on FMNIST, 4 does. We potentially attribute this to an insignificant number of trials using the 4 tickets that we trained on. If we ignore the 4-ticket merges, we can see that the training speed C decreased going from 2 to 3 merges. This might be because the effect of the dominant ticket was removed as parts of other tickets were added.

Tickets Merged	Avg. Best Accuracy	Avg. C Parameter	# Trials
	MNIST		
2	96.635	3.497	42
3	96.724	3.405	27
4	96.64	3.170	2
	FMNIST		
2	85.736	3.295	12
3	85.659	3.254	24
4	85.735	3.460	2

TABLE II: Average best accuracy and exponential C parameter for tickets combined via dominant merging.

Tickets Merged	Avg. Best Accuracy	Avg. C Parameter	# Trials
	MNIST		
2	96.668	3.558	6
3	96.72	3.189	4
4	96.85	3.331	1
	FMNIST		
2	85.832	3.419	6
3	85.45	3.477	4
4	85.76	2.916	1

TABLE III: Average best accuracy and exponential C parameter for tickets combined via averaging.

2) Average Ticket Merging: Table III displays the results of the experiment. Less trials were conducted for ATM than DTM as ATM did is not sensitive to ticket ordering. From the table, we can see that the smaller networks had the largest C parameter values, and therefore learned the quickest for both MNIST and FMNIST. Like in DTM, the best accuracy didn't elucidate any potential differences in the algorithm.

3) Merging All Tickets: We experimented in Table IV with merging all of the tickets we had generated together. Since our experiments above did not produce positive results, we hypothesized that the relatively large network from merging several tickets might perform better than a similarly sized network. For MNIST, for example, we merge all 7 tickets. This results in a 54.73% network, which means there was more substantial overlap. Without overlap, there would be 77.7% parameters remaining, so roughly 20% of total parameters overlapped. Table IV shows that with 7 tickets we achieved an accuracy of 98.63% and a C parameter of 3.068. In comparison, the 53.2% ticket network had an average best accuracy of 96.908% and a C parameter of 3.516. Since we only have one trial, it's difficult to determine, but our technique appears to perform worse in this experiment as well.

We also expected ATM to breakdown much faster than DTM as more tickets were merged (since DTM retains a dominant ticket). This appears to be the case, since in Table IV, DTM performs better than ATM on both datasets in best accuracy and higher C parameter. However, merging all our tickets prevented us from being able to perform more trials so this can't be stated as a fact.

Merge Type	# Tickets	Avg. Best Accuracy	Avg. C Parameter
MNIST			
Dom. (DTM)	7	96.83	3.068
Avg. (ATM)	7	96.66	2.907
FMNIST			
Dom. (DTM)	5	85.74	3.295
Avg. (ATM)	5	85.12	3.035

TABLE IV: Results for merging all tickets. The number of trials is 1.

VI. Conclusion

We found that the combined tickets perform slightly worse than a similar percent pruned ticket for low number of tickets merged. For example, a 20% ticket is better than two 10% tickets merged. When we merge large amounts of tickets (for example, 7), we also perform worse and learn slower. However, we suspect that the combined tickets may still be better than random networks which are the same percent pruned; for example, what might happen if we merge tickets until the network has near 100% active parameters? We were unable to test this theory due to the time it would take to create enough tickets for merging. However, based on merging all our MNIST tickets this also appears to result in a negative result. This test, however, did hint that dominant ticket merging performs better than average ticket merging. Future work could examine building some type of initialization scheme around one ticket. Perhaps an iterative parameter growth could be employed starting from some base ticket.

While the results are not what we may have hoped for, this work has given us an opportunity to examine current research in the field of Deep Learning and attempt to build upon it. In order to properly tackle this project, we had to read through literature related to the lottery ticket hypothesis and study other techniques aimed at improving it. Since the lottery ticket hypothesis deals with optimizing the training process, the computational workload for this project was much higher than previous workloads in the course and required additional consideration when planning the experiment. While accuracy was a reported metric, we also wanted to capture the rate of learning for the networks, and needed an appropriate way to measure it, ultimately leading to the select of exponential parameter C, illustrating that evaluating an Deep Learning technique or model is often more complicated than just examining accuracy.

VII. Acknowledgements

We would like to acknowledge rahulvigneswaran for their PyTorch implementation of [1] on Github. We used this code as a starting point to construct our experiments.

References

[1] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018.

[2] H. Zhou, J. Lan, R. Liu, and J. Yosinski, "Deconstructing lottery tickets: Zeros, signs, and the supermask," in *Advances in Neural Information Processing Systems*, pp. 3592–3602, 2019.

[3] S. Desai, H. Zhan, and A. Aly, "Evaluating lottery tickets under distributional shifts," *arXiv preprint arXiv:1910.12708*, 2019.

[4] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv preprint arXiv:1810.05270*, 2018.

[5] H. You, C. Li, P. Xu, Y. Fu, Y. Wang, X. Chen, Y. Lin, Z. Wang, and R. G. Baraniuk, "Drawing early-bird tickets: Towards more efficient training of deep networks," *arXiv preprint arXiv:1909.11957*, 2019.

[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[7] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

APPENDIX A: CODE DESIGN

Our code has been adapted from rahulvigneswaran's Pytorch implementation on GitHub (https://github.com/rahulvigneswaran/Lottery-Ticket-Hypothesis-in-Pytorch) of Frankle & Carbin's original lottery ticket experiments (2018). On top of the original lottery ticket generation and testing infrastructure, we also have implemented a considerable amount of functionality that allows us to perform the ticket combining study discussed in this report.

The code implements 7 key functions:

1) Lottery Ticket generation (main.py)
2) Dominant Ticket Merging and retraining (merge_tickets.py)
3) Average Ticket Merging and retraining (merge_tickets_avg.py)
4) Merge scripting (runTests.py)
5) Exponential fitting (C parameter) (exp_fit_example.py)
6) Evaluate merging results (evaluate_results.py)
7) Evaluate single ticket results (evaluate_tickets.py)

Example commands for these programs can be found in "commands_to_run.txt"

The main.py code essentially mirrors the original repository's main functionality, but there is additional support for CPU training for machines that do not have a CUDA-capable GPU. Furthermore, the function now saves significantly more intermediate data that is necessary for evaluating the progress of the pruning and training processes.

The merge_tickets scripts both implement a slightly different version of the main script that essentially combines the given lottery tickets and then runs a single iteration of the training from the main script only with no pruning at the end. These functions slightly alter the weight initialization dictionaries in order to reset the initializations based on the desired combination of lottery tickets.

The runTests script runs all of the combinations and permutations of the given tickets and dataset based on the given method of ticket combining and saves all test results for further evaluation.

The exponential fitting function fits an exponential function to the training results in order to estimate the "C-parameter" which essentially defines the rate at which our networks train.

The evaluate results functions are then utilized to further investigate the results of training, combination, and retraining to determine the most optimized initializations for our sparse networks.

APPENDIX B: WORKLOAD DISTRIBUTION

Hayden Coffey:

- Trained 3 mnist winning tickets on CPU.
- Wrote scripts for generating merging tests.
- Wrote scripts for creating results from tests and collating them.
- Wrote script for evaluating tickets.

Carl Edwards:

- Trained 4 fmnist, 3 mnist winning tickets on GPU.
- Got winning ticket code to work using PyTorch repo.
- Wrote dominate merging code.
- Trained merging tests on GPU.
- Implemented exponential fit code.

Landen McDonald:

- Trained 2 winning tickets (1 MNIST, 1 Fashion MNIST) on CPU
- Wrote weight averaging method
- Trained 2 combinations of tickets on CPU
- Helped write proposal
- Wrote Abstract-Dataset and Implementation for final report