# Appendices

## A. Experimental Settings of Hypothesis Evaluation

In this Appendix, we describe the experimental settings of the hypothesis evaluation in Section 3.2.

### A.1. Models

We use MLP on CIFAR-10, VGG-16 on CIFAR-10, ResNet-34 on CIFAR-100 to work through our hypothesis. We describe these models in detail as follows:

**MLP.** MLP is a clean three-layer MLP with ReLU activation for CIFAR-10. The number of neurons of each layer is 1024, 512, 10, respectively. No other regularization such as dropout or batch normalization is used further.

**VGG-16.** VGG-16 is a modified CIFAR-10 version of the original VGG model introduced by Lee et al. (2019). The size of the fully-connected layer is reduced to 512 and the dropout layers are replaced with batch normalization to avoid any other sparsification.

**ResNet-34.** ResNet-34 is the CIFAR-100 version of ResNet with 34 layers introduced by He et al. (2016).

### A.2. Algorithm

We choose Sparse Evolutionary Training (SET) (Mocanu et al., 2018) as the DST method to evaluate our hypothesis. SET helps to avoid the dense over-parameterization bias introduced by the gradient-based methods e.g., RigL and SNFS, as the latter utilize dense gradients in the backward pass to explore new weights. SET starts from a random sparse topology (*Erdős-Rényi*), and optimize the sparse connectivity towards a scale-free topology during training.

This algorithm contains three key steps:

1. Initializing a sparse neural network with *Erdős-Rényi* random graph at a sparsity of S.
2. Training the sparse neural network for $\Delta T$ iterations.
3. Removing weights according to the standard magnitude pruning and growing new weights in a random fashion.

Steps 2 and 3 will be repeated iteratively until the end of the training. By doing this, SET maintains a fixed parameter count throughout training.

### A.3. Training

*Table 3.* Experiment hyperparameters of the hypothesis evaluation in Section 3.2. The hyperparameters include Learning Rate (LR), Batch Size (BS), Typical Training Epochs (TT Epochs), Learning Rate Drop (LR Drop), Weight Decay (WD), Sparse Initialization (Sparse Init), Update Interval of the Extended Training ($\Delta T$), Pruning Rate Schedule (Sched), Initial Pruning Rate (P), etc.

| Model | Data | Methods | LR | BS | TT Epochs | LR Drop | WD | Sparse Init | $\Delta T$ | Sched | P |
|---|---|---|---|---|---|---|---|---|---|---|---|
| MLP | CIFAR-10 | RigL | 0.01 | 128 | 200 | 10x | 5e-4 | ER | 4000 | Cosine | 0.5 |
| MLP | CIFAR-10 | SET | 0.01 | 128 | 200 | 10x | 5e-4 | ER | 1500 | Cosine | 0.5 |
| VGG-16 | CIFAR-10 | SET | 0.1 | 128 | 250 | 10x | 5e-4 | ERK | 2000 | Cosine | 0.5 |
| ResNet-34 | CIFAR-100 | SET | 0.1 | 128 | 200 | 10x | 1e-4 | ERK | 1000 | Cosine | 0.5 |

We basically follow the experimental settings from Dettmers & Zettlemoyer (2019).

For models trained for a typical time, we train them with various update interval $\Delta T$ reported in Figure 3. We use a set of 10% training data as the validation set and train on the remaining training data. Weight growth is guided by random sampling and weight pruning is guided by magnitude. We do not specifically finetune the starting point and the finishing point of the parameter exploration. The exploring operation is performed throughout training. The initial sparse connectivity is sampled by the *Erdős-Rényi* distribution introduced in Mocanu et al. (2018). We set the initial pruning rate as 0.5 and

gradually decay it to 0 with a cosine annealing, as introduced in Dettmers & Zettlemoyer (2019). The remaining training hyperparameters are set as follows:

**MLP.** We train sparse MLPs for 200 epochs by momentum SGD with a learning rate of 0.01 and a momentum coefficient of 0.9. We use a small learning rate 0.01 rather than 0.1, as the dense MLP doesn't converge with a learning rate of 0.1. We decay the learning rate by a factor of 10 every 24000 iterations. We set the batch size as 128. The weight decay is set as 5.0e-4.

**VGG-16.** We strictly follow the experimental settings from Dettmers & Zettlemoyer (2019) for VGG-16. All sparse models are trained with momentum SGD for 250 epochs with a learning rate of 0.1, decayed by 10 every 30000 mini-batches. We use a batch size of 128 and weight decay to 5.0e-4.

**ResNet-34.** We train sparse ResNet-34 for 200 epochs with momentum SGD with a learning rate of 0.1, decayed by 10 at the 100 and 150 epoch. We use a batch size of 128 and weight decay to 1.0e-4.

For models trained for an extended training time, we simply extend the training time and the anchor epochs of the learning rate schedule, while using a large $\Delta T$. The update interval $\Delta T$ is chosen according to the trade-off shown in Figure 3. Besides the learning steps, the anchor epochs of the learning rate schedule and the pruning rate schedule are also scaled by the same factor. For each training time, the accuracy are averaged over 3 seeds with mean and standard deviation. More detailed training hyperparameters are shared in Table 3.

## B. Implementation Details of RigL-ITOP in Section 4.2

*Table 4.* Experiment hyperparameters in Section 4.2 and Section 5. The hyperparameters include Learning Rate (LR), Batch Size (typical training time / extended training time) (BS), Training Epochs (typical training time / extended training time) (Epochs), Learning Rate Drop (LR Drop), Weight Decay (WD), Sparse Initialization (Sparse Init), Update Interval ($\Delta T$), Pruning Rate Schedule (Sched), Initial Pruning Rate (P), etc.

| Model | Data | Methods | LR | BS | Epochs | LR Drop | WD | Sparse Init | $\Delta T$ | Sched | P |
|---|---|---|---|---|---|---|---|---|---|---|---|
| MLP | CIFAR-10 | SET-ITOP | 0.01 | 32 / 128 | 200 / 4000 | 10x | 5e-4 | ER | 1500 | Cosine | 0.5 |
| MLP | CIFAR-10 | RigL-ITOP | 0.01 | 32 / 128 | 200 / 4000 | 10x | 5e-4 | ER | 4000 | Cosine | 0.5 |
| ResNet-34 | CIFAR-100 | SET-ITOP | 0.1 | 32 / 128 | 200 / 4000 | 10x | 1e-4 | ERK | 1500 | Cosine | 0.5 |
| ResNet-34 | CIFAR-100 | RigL-ITOP | 0.1 | 32 / 128 | 200 / 4000 | 10x | 1e-4 | ERK | 4000 | Cosine | 0.5 |
| ResNet-50 | ImageNet | RigL-ITOP | 0.1 | 64 / - | 100 / - | 10x | 1e-4 | ERK | 4000 | Cosine | 0.5 |

In this Appendix, we describe our replication of RigL (Evci et al., 2020a) and the hyperparameters we used for RigL-ITOP.

RigL is a state-of-the-art DST method growing new weights that are expected to receive gradient with high magnitude in the next iteration. Besides, it shows the proposed sparse distribution *Erdős-Rényi-Kernel* (ERK) improves the sparse performance over the *Erdős-Rényi* (ER). Since RigL is originally implemented with TensorFlow, we replicate it with PyTorch based on the implementation from Dettmers & Zettlemoyer (2019). We note that RigL tunes the starting epoch and the ending point of the mask update. To encourage more exploration, we do not follow this strategy and explore sparse connectivities throughout training. We train sparse ResNet-50 for 100 epochs, the same as Dettmers & Zettlemoyer (2019); Evci et al. (2020a). The learning rate is linearly increased to 0.1 with a warm-up in the first 5 epochs and decreased by a factor of 10 at epochs 30, 60, and 90. To reach a high and reliable In-Time Over-Parameterization rate, we use a small batch size of 64 and an update interval of 4000. Batch sizes lower than 64 lead to worse test accuracy. ImageNet experiments were run on 2x NVIDIA Tesla V100. With more fine-tuning, the results of RigL-ITOP (e.g. extended training time) can likely be improved, but we lack the resources to do it. We share the hyperparameters of RigL-ITOP in Table 4.

## C. Implementation Details in Section 5

In this Appendix, we describe the hyperparameters of SET-ITOP used in 5 in Table 4. The replication details of LTH and SNIP are given below.

**LTH.** Lottery Ticket Hypothesis (LTH) (Frankle & Carbin, 2019) shows that there exist sub-networks that can match the accuracy of the dense network when trained with their original initializations. We follow the PyTorch implementation

provide by Liu et al. (2019) on GitHub[1] to replicate LTH.

Give the fact that the iterative pruning process of LTH would lead to much larger training resource costs than SNIP and static sparse training, we use one-shot pruning for LTH. For the typical training time setting, we first train a dense model for 200 epochs, after which we use global and one-shot magnitude pruning to prune the model to the target sparsity and retrain the pruned model with its original initializations for 200 epochs.

**SNIP.** Single-shot network pruning (SNIP) proposed in Lee et al. (2019), is a method that attempts to prune at initialization before the main training based on the connection sensitivity score $s_i = |\frac{\partial L}{\partial w_i} w_i|$. The weights with the smallest score are pruned. We replicate SNIP based on the PyTorch implementation on GitHub[1]. Same as Lee et al. (2019), we use a mini-batch of data to calculate the important scores and obtain the sparse model in a one-shot fashion before the initialization. After that, we train the sparse model without any sparse exploration for 200 epochs.

## D. Extended Training Performance of RigL with $\Delta T = 1500$

According to the results from Figure 5, we can see the $\Delta T = 4000$ is a good choice for the update interval of RigL. What if we choose a small update interval, e.g., $\Delta T = 1500$? Here we compare the extended training performance of RigL with two different update intervals 1500 and 4000. The results are shown in Figure 9. It is clear to see models trained with $\Delta T = 1500$ fall short of models trained with $\Delta T = 4000$, which indicates small update intervals is not sufficient for newly weights to catch up the existing weights in terms of magnitude. More importantly, although expected to perform sparse exploration more frequently, models trained with $\Delta T = 1500$ end up with a lower $R_s$ than the ones trained with $\Delta T = 4000$. These results highlight the importance of the sufficient training time for the new weights.

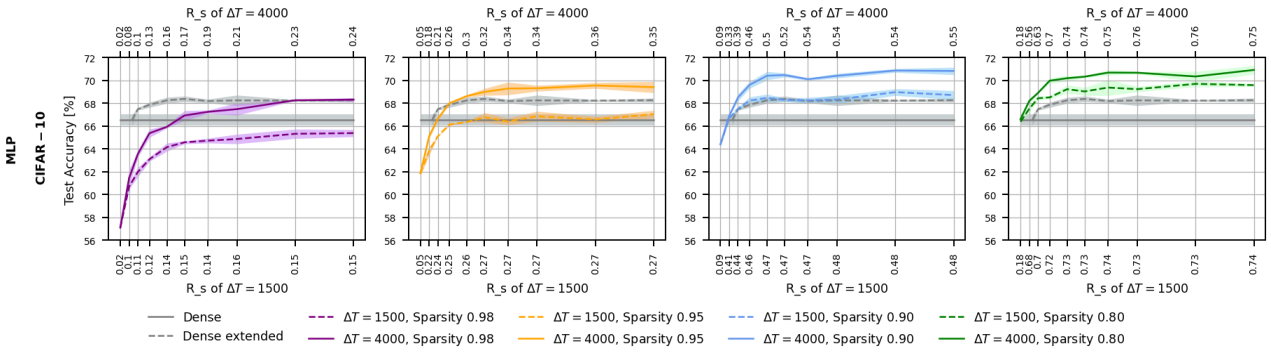


*Figure 9.* Extended training performance of RigL with update interval $\Delta T = 1500$ and $\Delta T = 4000$.

## E. Test Accuracy of RigL with Various Batch Sizes

In this Appendix, we evaluate the performance of RigL with different batch sizes. We choose MLP as our model and the update interval $\Delta T = 4000$. The results are shown in Figure 10. Similar with SET, the performance of RigL also increases as the batch size decrease from 256 to 32. After that, the performance starts to drop due to the noisy input caused by the extreme small batch sizes. The In-Time Over-Parameterization rate ($R_s$) of RigL is again bounded up to some values. We also provide the comparison between RigL (solid lines) and SET (dashed lines) in this setting. We find a similar pattern with the extended training time, that is, RigL outperforms SET when $R_s$ is small but falls short of SET when sufficient parameters have been reliably explored.

## F. Regrowing from the Non-Activated Weights First

One direct way to increase the In-Time Over-Parameterization rate during a typical training time is to sample from the non-activated weights first when performing weight growing. We evaluate this idea with SET by regrowing the non-activated weights first and report the results as SET+ with $(\text{mean} \pm \text{std}, R_s)$ in Table 5. We training sparse ResNet-18 on CIFAR-10

[1] https://github.com/Eric-mingjie/rethinking-network-pruning
[1] https://github.com/mil-ad/snip

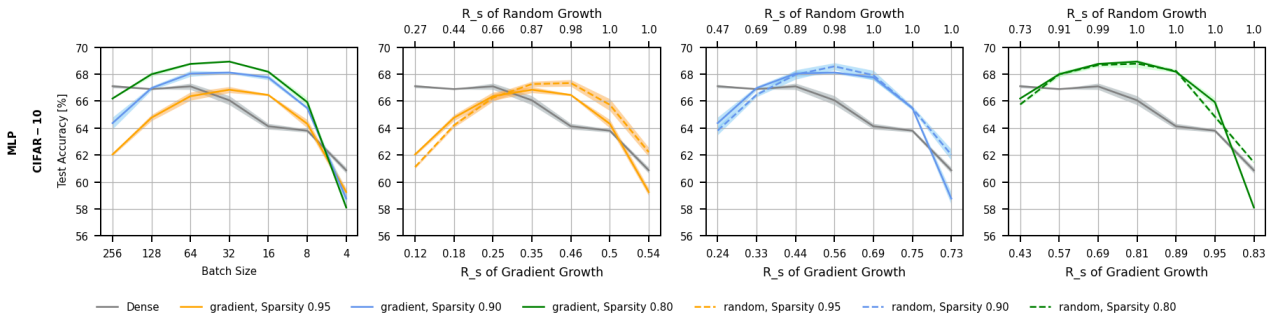


*Figure 10.* Test accuracy of RigL with various batch sizes. The update interval $\Delta T$ is set as 4000.

for 250 epochs with a learning rate of 0.1 decayed by 10x at 125, 187 epochs, a batch size of 128, a pruning rate of 0.5.

When the parameter exploration is insufficient (small $R_s$), SET+ consistently achieves higher accuracy and higher $R_s$ than SET. Whiling effective, the $R_s$ increase achieved by this modification is relatively limited. This observation highlights an important direction for future work to achieve high $R_s$ within a typical training time.

*Table 5.* Performance of sparse ResNet-18 on CIFAR-10 with various pruning rates. The results are run three times and reported with (mean $\pm$ std, $R_s$). The highest test accuracies are marked in bold.

| Sparsity | Method | P | $\Delta T = 15000$ | $\Delta T = 10000$ | $\Delta T = 8000$ | $\Delta T = 5000$ | $\Delta T = 3000$ |
|---|---|---|---|---|---|---|---|
| 0.9 | SET | 0.5 | (94.30 $\pm$ 0.16, 0.162) | (94.47 $\pm$ 0.14, 0.201) | (94.25 $\pm$ 0.10, 0.228) | (94.36 $\pm$ 0.08, 0.302) | (**94.54** $\pm$ 0.05, 0.411) |
| | SET+ | 0.5 | (**94.43** $\pm$ 0.14, 0.169) | (**94.59** $\pm$ 0.11, 0.215) | (**94.54** $\pm$ 0.28, 0.247) | (**94.38** $\pm$ 0.07, 0.342) | (94.53 $\pm$ 0.03, 0.492) |
| 0.95 | SET | 0.5 | (93.57 $\pm$ 0.16, 0.086) | (93.46 $\pm$ 0.04, 0.108) | (**93.67** $\pm$ 0.04, 0.124) | (93.60 $\pm$ 0.04, 0.170) | (93.61 $\pm$ 0.09, 0.241) |
| | SET+ | 0.5 | (**93.66** $\pm$ 0.14, 0.088) | (**93.70** $\pm$ 0.03, 0.114) | (93.66 $\pm$ 0.15, 0.133) | (**93.78** $\pm$ 0.04, 0.186) | (**94.00** $\pm$ 0.08, 0.272) |