The Lottery Initialization and Generalizability Analysis of Deep Recurrent Networks

Haoran You
haoran.you@rice.edu

Tony Yu
by12@rice.edu

Abstract

Deep recurrent networks (DRNs) demonstrate superior capability and performance for reasoning tasks than standard feed-forward deep networks. In particular, DRNs trained on easy problems can extrapolate their knowledge to solve harder problems by increasing their test time iteration budgets (i.e., recurrence), drawn an analogy from that humans tend to solve harder problems by thinking for a longer time. Recent studies reveal that DRNs are hard to train and scale to highly complex problems, requiring a large amount of recurrence during test time. To this end, we explore a lottery initialization of DRNs for achieving better accuracy with even less recurrence. In particular, we prune the given DRNs once prior to training according to a saliency criterion to identify important connections (i.e., lottery initialization) with only a mini-batch of data and no pretraining thus negligible overhead. Experiments on two reasoning tasks consistently demonstrate the superior accuracy, fewer parameters, and less recurrence of DRNs with lottery initialization. Moreover, we train DRNs on one dataset and transfer them to another with different data distributions, i.e., out-of-distribution (OOD), e.g., CIFAR-10 to SVHN, providing more insights for understanding the generalizability of DRNs. Codes and instructions are provided at `https://github.com/ranery/ELEC631`.

1 Introduction

While feed-forward deep networks have achieved unprecedented performance in various tasks and applications like computer vision (CV) and natural language processing (NLP), they fall short when solving reasoning tasks. Instead, deep recurrent networks (DRNs) stand out due to their reasoning capability. DRNs trained only on small or simple reasoning problems can synthesize and handle large or complex problems. Specifically, DRNs achieve such logical extrapolation by increasing the iterations/recurrences at test time and thus are also referred to as "deep thinking models" [1, 2], drawn an analogy from that humans tend to solve harder problems by thinking for a longer time. However, DRNs are hard to converge especially for highly complex problems and often require a large amount of recurrence during test time, making them difficult to be deployed to resource-constrained devices and thus limiting their pervasive applications. Prior work [2] explores from the model architecture perspective and makes DRNs more stable to train by introducing residual connections, also referred to as "recall mechanism". But they still do not consider how to make them more efficient, e.g., less recurrences or fewer parameters, for facilitating their deployment in resource-constrained devices.

To further improve both the achievable accuracy and the efficiency of DRNs, we advocate to find a **lottery initialization** as a sparse regularization term for facilitating the DRNs' stable and efficient training. In particular, we first identify the important weight connections according to predefined pruning criteria, and then use a binary mask m to construct sparse DRNs where "1" means reserved connections while "0" denotes pruned connections. In this way, the lottery initialization can be represented as $m \odot W$ where W denotes the original initialized weights of DRNs and m is the learned mask. Notably, we derive such lottery initialization with at most a mini-batch of data samples

ELEC 631 Project Report.

and without any pretraining, incurring little overhead as compared to the whole training process (i.e., $< 0.01\%$). We make non-trivial efforts to explore and validate the potential of such a lottery initialization by answering the key question: *whether such lottery initialization helps to improve the accuracy and inference efficiency of DRNs?* To the best of our knowledge, this is the first attempt taken towards discovering the lottery initialization for DRNs via pruning at initialization methods. Our empirical contributions can be summarized as follows:

1. We for the first time discover the lottery initialization of DRNs on reasoning tasks via pruning at initialization methods. Such lottery initialization helps DRNs to better extrapolate from easy problems to harder problems, achieving comparable or better accuracy at even less recurrences and number of parameters during test time.
2. We provide an empirical trial of transferring the DRNs trained on one dataset to another with different data distributions, e.g., CIFAR-10 to SVHN, providing more insights for understanding the generalizability of DRNs to out-of-distribution (OOD) tasks or problems.

2 Related Works

Deep Recurrent Networks (DRNs). Our discussion of Deep Recurrent Networks are based on the Existing work on algorithm learning involves recurrent neural network (RNN) based approaches. For modeling variable-length sequences, recurrent neural networks have recently become a popular choice. RNNs have been used successfully for a variety of tasks. The conventional Recurrent neural network accepts current input data as well as previously received inputs and can handle sequential data. Because of their internal memory, RNNs can remember previous inputs. Nevertheless, it has been discovered that training RNNs to deal with long-term sequential data is difficult because the gradients tend to vanish. To address this issue, LSTM is reduces the effects of vanishing and exploding gradients. The structure of hidden units in LSTM is changed from "sigmoid" or "tanh" to memory cells, with gates controlling their inputs and outputs. These gates regulate the flow of information to hidden neurons while preserving previously extracted features. To improve on the performance of base LSTM models, deep LSTM-based(stacked LSTM) RNN models was proposed [3]. Deep neural network architectures can represent functions that are exponentially more efficient than shallow architectures. While recurrent networks are inherently deep in time because each hidden state is a function of all previous hidden states, the internal computation has been shown to be relatively shallow. The addition of one or more nonlinear layers in the transition stages of an RNN can improve overall performance by better disentangling the underlying variations in the original input, according to.

Algorithm learning refers to data-driven models that learn scalable processes. The ability of recurrent neural networks (RNNs) to process input strings of arbitrary lengths was studied in the early works on this topic [4] propose neural Turing machines inspired by massively parallel graphical processing units, and Kaiser, Sutskever et al.[5] propose a parallel version inspired by massively parallel graphical processing units. These methods, as well as various improvements to them, produce promising results on bit string to bit string tasks such as copying inputs and adding integers, and even show the ability to generalize from shorter training strings to longer ones at testing [6]. Because they are based on traditional RNNs,However, because the amount of computation they perform is directly proportional to the length of the input string, they cannot perform more or less computation regardless of the input size (or corresponding to the difficulty of the problem).

The task of generalizing to test sets that contain more computationally complex samples than the training data is known as logical extrapolation. [7] claims that "neural network models with end-to-end training pipelines... cannot, on their own, successfully perform algorithmic reasoning," and proposes a hybrid hand-crafted and learned approach instead. Palm et al.[2] propose recurrent relational networks, which operate on graphs and pass messages iteratively. They also claim that classical architectures lack the inductive bias that allows them to reason about object relationships. Several recent studies have cast doubt on these claims. Recurrent networks based on weight-sharing architectures can be made deeper at test time regardless of the input size. In a variety of domains, these systems exhibit logical extrapolation behavior.

Pruning at Initialization. As modern deep neural networks grows deeper with more hidden layers and recurrences, so do their computational complexities, often containing millions of weights and parameters.Pruning at initialization is a technique to optimizing the computation complexity of the

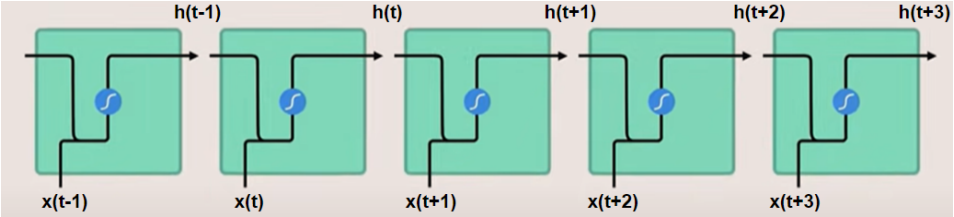

Figure 1: An illustration of DRNs.

DNN training by removing a percentages of "useless" weights at initialization to accelerating the learning process without sacrificing too much overall training accuracy, producing a smaller scaled neural network that performs similarly to the original model. Conventional pruning techniques assign scores to parameters in neural networks after training and remove the parameters with the lowest scores. While these pruning algorithms can compress neural networks at test time, they have no effect on training costs as it fails to reduce the total number of weights while initially training. More recent discussions on this topic indicates that randomly initialized neural networks can be pruned before training with little or no loss in the final test accuracy [8, 9, 10].The Iterative Magnitude Pruning (IMP) algorithm [8, 11] in particular repeats multiple cycles of training, pruning, and weight rewinding to find extremely sparse neural networks at initialization that can be trained to match the original network's test accuracy. While IMP is powerful, it necessitates multiple cycles of costly training and pruning using hyperparameters that are very specific. A different approach avoids these issues by pruning the network in a single shot using the gradients of the training loss at initialization[12], which often times suffer from layer-collapse, the premature pruning of an entire layer disconnect the neural networks, forbidding the forward and backwards propagation. This effect is very evident in training results as sudden collapse of training accuracy to 0. Current pruning algorithm such as **Synflow** [13]tackles this problem through assigning scores to individual weights with respect to Synaptic Saliency, a gradient based scores scoring metric that ensures layer-wise conservation of pruning scores,and iterative pruning,a sequential pruning technique that gradually increases the weights score in larger layers while pruning to reach maximal critical compression while ensuring information flow from end to end. As synaptic saliency conserves the scores of each layers, larger layers weights are significantly pruned due to lower average score compared to smaller layers, iterative pruning guarantees that these weights do not get extincted through over-targeting.

3 Methods

In this section, we first present the explored lottery initialization of DRNs via pruning at initialization methods, as described in Sec. 3.1, and then provide an empirical analysis of the generalizability of DRNs between different data distributions, as shown in Sec. 3.2.

3.1 Lottery Initialization of DRNs

Preliminary of DRNs. A conventional DRN is constructed by defining the transition function and the output function as shown in Eq. 1 and Eq. 2, respectively.

$$\mathbf{h}_t = f_h(\mathbf{x}_t, \mathbf{h}_{t-1}) = \phi_h\left(\mathbf{W}^\top \mathbf{h}_{t-1} + \mathbf{U}^\top \mathbf{x}_t\right) \tag{1}$$

$$\mathbf{y}_t = f_o(\mathbf{h}_t, \mathbf{x}_t) = \phi_o\left(\mathbf{V}^\top \mathbf{h}_t\right), \tag{2}$$

where $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{V}$ are the transition, input and output matrices, respectively, and ϕ_h and ϕ_o are element-wise nonlinear functions. It is usual to use a saturating nonlinear function such as a logistic sigmoid function or a hyperbolic tangent function for ϕ_h [14]. An illustration of this DRN is in Fig. 1.

Pruning Criterion. There are many ways to generate a pruned network $f(x; m \odot W)$ from an initially untrained DRN $f(x, W)$, note that here W denotes all trainable parameters at initialization. Next, we present four kinds of pruning criteria that can be applied to DRNs at initialization.

Random Pruning. The easiest method is to randomly drop some weight connections according to a predefined pruning ratio. This method assigns a random score $s \sim \text{Uniform}(0, 1)$ to each weight and removes the weights with the lowest scores. Random pruning is a naive early pruning baseline that any new proposal should benchmark and outperform.

Magnitude Pruning. This method assigns each weight a score based on its magnitude, $s = |W|$, and discards the ones with the lowest scores. Magnitude pruning is a common method for pruning after inference training [15], and it provides another naive point of comparison for early pruning.

Single-shot Network Pruning (SNIP). This method [9] takes a single iteration on sampled mini-batch training data $\mathcal{D}$, in which way we can prune unimportant ones in single-shot, disentangling the pruning process from iterative optimization. In particular, we take the magnitude of the derivatives $\mathbf{g}_j$ as the saliency criterion. If the magnitude of the derivatives is high (regardless of the sign), it means that the corresponding connection m_j has a considerable effect on the loss, and it should be preserved to allow W_j to be updated. As such, the connection sensitivity score is defined as:

$$s_j = \frac{|\mathbf{g}_j(W; \mathcal{D})|}{\sum_k |\mathbf{g}_k(W; \mathcal{D})|}. \tag{3}$$

Such an method is justified by the fact that it preserves weights with the greatest effect on the loss (No matter positive or negative).

Gradient Signal Preservation (GraSP). GraSP[16] also consider gradients as a metric to measure the connection sensitivity to remove unimportant connections as SNIP [9]. But it further argues that the first order of gradients indicate gradient flow after pruning. Suppose the pruning as adding a perturbation δ to the initial weights. We can use a Taylor approximation to characterize how removing one weight will affect the gradient flow:

$$s(\delta) = \Delta\mathcal{L}(W + \delta) - \Delta\mathcal{L}(W) = 2\delta^T \mathbf{H}\mathbf{g} + \mathcal{O}(\|\delta\|_2^2), \tag{4}$$

where $s(\delta)$ approximately measures the gradient changes after pruning, the Hessian matrix $\mathbf{H}$ captures the dependencies between different weights, and thus helps predict the effect of removing multiple weights. If $\mathbf{H}$ is the identify matrix, then GraSP will be degraded to SNIP. The pruning criterion of GraSP is that weights that reduce gradient flow are removed, i.e., smaller $s(\delta)$, while weights that increase gradient flow are preserved, i.e., larger $s(\delta)$.

In our experiments, we adopt the above four pruning criteria for DRNs on two reasoning tasks, and find that some of them can even boost the model accuracy with less recurrences and fewer parameters. Thus, we term those pruned subnetworks at initialization, i.e., $m \odot W$, as the **lottery initialization**.

3.2 Generalization Analysis

1.In this paper, we establish and extend on testing DTnet's ability to generalize in vision-oriented tasks utilizing the Cifar-10 and SVHN datasets. Previous, Bansal et al. experimented with the Dtnet on extrapolating reasoning-oriented tasks such as the prefix sum dataset and maze dataset from the easier tasks (smaller input sequence size) to harder/larger samples with promising results. However, the paper didn't experient vision-oriented tasks like object recognition or digit detection which are one of the most popular tasks that benefit significantly from DNN. We attempted to migrate the current DTnet to perform CV tasks but discovered that, unlike reasoning-oriented problems, the system fail to achieve greater extrapolation and suffered from the overthinking problem, failing to scale to highly complex problems because behavior degenerates when many iterations are applied, thus producing no interpretable outputs. We suspect that further implementation of recall architecture(concatenating the input problem to the output of the feature from each instance of the recurrent block.) or a progressive training routine that pushes the model to learn behaviors that can be repeated indefinitely rather than behaviors that are specific to iteration number might be able to solve this problem.

2. We then put our Cifar10-trained modified DRN model to the test on a different data distribution (SVHN-dataset) to see how well it generalized. In comparison to the previous instance, where we deployed the DRN model on the same data distribution as the training sample, we saw a severe drop in overall detection accuracy.

3.To investigate our DRN's ability to generalize further, we increased the recurrences between the blocks in the modified architecture, which resulted in an overall accuracy improvement. Although the modified DRN architecture still suffers from the overthinking problem, we noticed that as the number of iterations increases, the accuracy increases, which is in line with Bansal et al.'s discussion.[2].

4 Experiments

In this section, we first articulate the experiment settings in Sec. 4.1, and then evaluate DRNs with lottery initialization in Sec. 4.2 and evaluate DRNs on OOD distributions in Sec. 4.3, respectively.

4.1 Experiment Settings

4.1.1 Models and Datasets

DTnet [2] is a deep-thinking architecture that uses specially designed neural architectures and specialized training loops to train systems that avoid the overthink problem and instead converge to a fixed point after thousands of iterations. Deep-thinking systems can then demonstrate extreme logical extrapolation behaviors, leaping from solving small/simple training problems with tens of iterations to solving large and complex problem instances at test time with thousands of iterations.

Prefix Sum. A binary string represents each training sample. The goal is to generate an equal-length binary string, with each bit representing the cumulative sum of input bits modulo two. The deep-thinking models[2] accept strings of any length as input, but longer strings are more difficult to process than shorter strings. There are 10,000 uniform random binary strings in each dataset, with no duplicates. We use datasets with 32, 44, and 48 bit input lengths. An instance input and the corresponding target output are shown in Fig. 2.

$$\begin{array}{ll} \text{Input:} & [1,0,0,1,0,1,0,1,0,1,0,1,1,0,1,1,0,1] \\ \text{Target:} & [1,1,1,0,0,1,1,0,0,1,1,0,1,1,0,1,1,0] \end{array}$$

Figure 2: Prefix sum input & target example

Maze dataset consist of 50000 two-dimensional square images with black walls, white permissible paths, and red and green squares denoting the start and end positions. The problem's targets are maps of the same dimension as the input maze, but with ones on the best path and zeros everywhere else. By increasing the size of the mazes, we can create more difficult datasets.For training, we used 9x9 sized mazes from the data set. We then benchmark extrapolation performance on mazes of size 59x59, and observe models solving mazes as large as 201x201 with high success rates, despite the small training size.

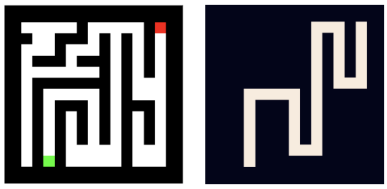

Figure 3: Maze dataset

CIFAR-10 dataset was created with the goal of creating a cleanly labeled subset of Tiny Images. The researchers created a dataset with ten classes and 6,000 images per class to achieve this goal. A plane, a car, a bird, a cat, a deer, a dog, a frog, a horse, a ship, and a truck are among these classes. There are 50,000 training images and 10,000 test images in the standard train/test split, which is class-balanced.

SVHN (The Street View House Numbers) [17] contains 630,420 color images with a 32 x 32 pixel resolution. Each image revolves around a number ranging from one to ten, which must be identified. There are 73,257 training images and 26,032 test images in the official dataset split, but there are also 531,131 additional training images available. When training our models, we use both available training sets and do not use any data augmentation, as is standard procedure for this dataset. The per-channel mean and standard deviation are used to normalize all images.

4.1.2 Training and Testing Settings.

We train the DTNets following the default setting as used in [2]. In particular, we adopt recall mechanism and progressive training techniques with $\alpha = 0.01$ to overcome the "over-thinking" problem. The width of DTNets is set as 400 and 128 for Prefix Sum task and Maze task, respectively. For training, the recurrences or iterations are 30, while for testing, we sweep all recurrences ranges from 30 to 600 to generate a complete accuracy-recurrence trade-off curve. Also, for the Prefix Sum task, we train DTNets with 32-bit data while test them on harder 512-bit data; for the Maze task, we train DTNets with 9×9 mazes while test them on harder 13×13 mazes.

For evaluating the generalizability of DRNs, we develop the recurrent ResNet based on this codebase [1] and follow their default training hyper-parameters. Note that we do not incorporate the recall

[1] https://github.com/kuangliu/pytorch-cifar

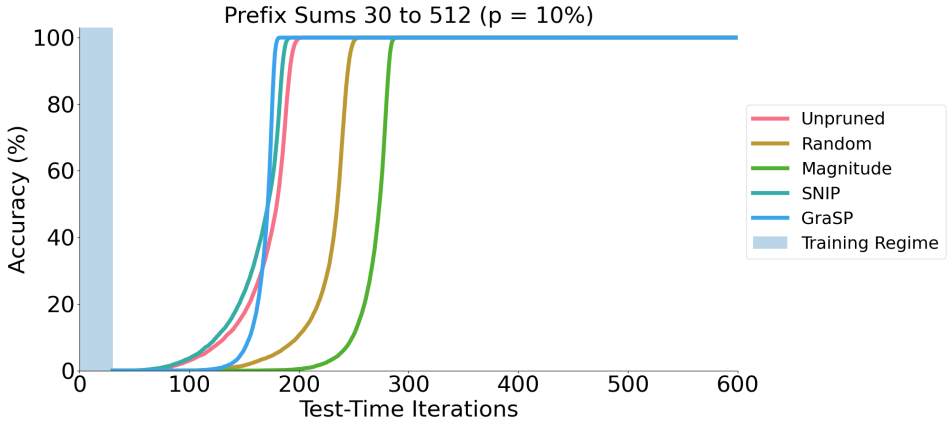

Figure 4: Comparison between DRNs without pruning and DRNs with various pruning methods (i.e., Random, Magnitude, SNIP, and GraSP) on the Prefix Sum task.

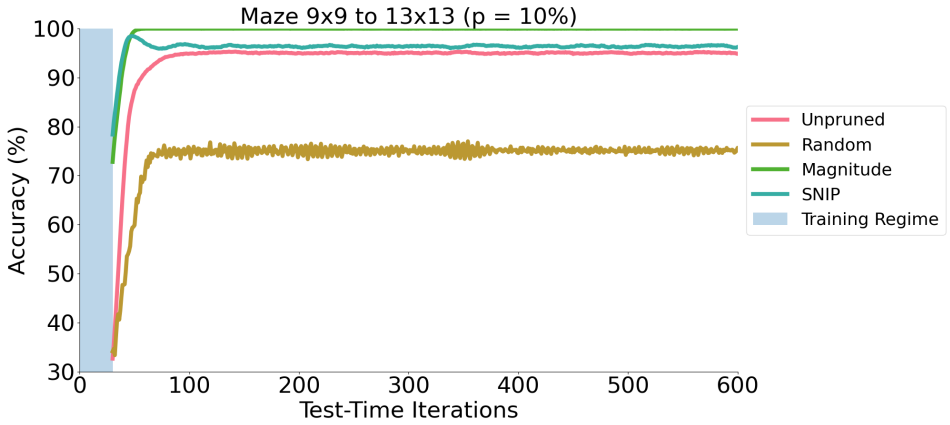

Figure 5: Comparison between DRNs without pruning and DRNs with various pruning methods (i.e., Random, SNIP) on the Maze task.

mechanism and progressive training techniques as proposed in [2] so the recurrent ResNet suffers from "overthinking" problem, we will consider such techniques in our further work.

4.2 Evaluation of DRNs with Lottery Initialization

We conduct experiments of DRNs with various pruning methods at initialization on two reasoning tasks, Prefix Sum and Maze, and compare their achievable accuracy and required recurrences with vanilla DRNs without pruning, as shown in Fig. 4 and Fig. 5, respectively, where the x-axis represents the recurrences or iterations we use during test, y-axis denotes the achievable accuracy at given recurrences, and shade area means the iterations we adopt during training DRNs. We observe that (1) DRNs with SNIP pruning methods consistently generate better accuracy (+1.2% $\sim$ +12.9%) than vanilla DRNs without pruning, where the +12.9% occurs when the test iteration is around 50 for the maze task; (2) DRNs with other pruning criterion, e.g., magnitude pruning, fails to be comparable with unpruned DRNs on Prefix Sum task, while can be better on Maze tasks, indicating that the lottery initialization is task dependent; (3) The accuracy of DRNs with GraSP tends to remain low and suddenly increase as the testing iterations increasing as shown in Fig. 4, we interpret and attribute such sudden increases to the adopted gradient flow criterion, which measures not only the magnitude of gradients but also the second-order Hessian information of loss; (4) DRNs with advanced pruning criteria, e.g., GraSP [16] and SynFlow [13], often suffer from non-convergence problem. We believe that this set of experiments helps validate the effectiveness of our proposed lottery initialization of DRNs, which helps to achieve advanced accuracy and efficiency trade-offs.

Table 1: Result of DRN's generalizability

Dataset	Recurrences/Iterations						
	5	7	9	11	13	15	20
CIFAR-10	**94.82%**	92.69%	87.78%	77.63%	62.36%	44.96%	16.38%
SVHN	8.34%	7.89%	7.89%	7.96%	8.64%	9.64%	**10.65%**

4.3 Evaluation of DRNs' Generalizability

Table 1 above shows the outcome of our experiments, we can observe that the modified DTnet model generalized well across the Cifar-10 dataset while training, with an overall accuracy of around 94 percent; nevertheless, as the number of iterations increases, it begins to suffer from the overthinking problem, as evident in the first row of table one. As the number of iterations increases, the training accuracy decreases. When using the Cifar-10 trained DRN model on the SVHN dataset, we can see a significant decrease in training accuracy; but even so, as more recurrences and iterations are incorporated, the accuracy grows positively. This result hints the possibility of implementing more complex recurrence architecture in DRN, similar publishing such as kim et al.[18] proposed a deeply-recursive convolutional network (DRCN) that also shows potential applicability in imaging tasks.

5 Conclusion

In this project, we explore the lottery initialization of DRNs for achieving better accuracy with even less recurrence by pruning DRNs at initialization with various criteria. Experiments on two reasoning tasks consistently demonstrate the superior accuracy, fewer parameters, and less recurrence of DRNs with lottery initialization. Moreover, we train DRNs on one dataset and transfer them to another with different data distributions, i.e., out-of-distribution (OOD), e.g., CIFAR-10 to SVHN, providing more insights for understanding the generalizability of DRNs.

References

[1] Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[2] Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Logical extrapolation without overthinking. *arXiv preprint arXiv:2202.05826*, 2022.

[3] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.

[4] Felix A Gers and E Schmidhuber. Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340, 2001.

[5] Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

[6] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[7] Kwabena Nuamah. Deep algorithmic question answering: Towards a compositionally hybrid ai for algorithmic reasoning. *arXiv preprint arXiv:2109.08006*, 2021.

[8] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[9] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

[10] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[11] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611*, 2019.

[12] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark? *arXiv preprint arXiv:2009.08576*, 2020.

[13] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems*, 33:6377–6389, 2020.

[14] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

[15] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

[16] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020.

[17] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[18] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Deeply-recursive convolutional network for image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1637–1645, 2016.