Selfish Sparse RNN Training

Shiwei Liu [1] **Decebal Constantin Mocanu** [1 2] **Yulong Pei** [1] **Mykola Pechenizkiy** [1]

Abstract

Sparse neural networks have been widely applied to reduce the computational demands of training and deploying over-parameterized deep neural networks. For inference acceleration, methods that discover a sparse network from a pre-trained dense network (dense-to-sparse training) work effectively. Recently, dynamic sparse training (DST) has been proposed to train sparse neural networks without pre-training a dense model (sparse-to-sparse training), so that the training process can also be accelerated. However, previous sparse-to-sparse methods mainly focus on Multilayer Perceptron Networks (MLPs) and Convolutional Neural Networks (CNNs), failing to match the performance of dense-to-sparse methods in the Recurrent Neural Networks (RNNs) setting. In this paper, we propose an approach to train intrinsically sparse RNNs with a fixed parameter count in one single run, without compromising performance. During training, we allow RNN layers to have a non-uniform redistribution across cell gates for better regularization. Further, we propose SNT-ASGD, a novel variant of the averaged stochastic gradient optimizer, which significantly improves the performance of all sparse training methods for RNNs. Using these strategies, we achieve state-of-the-art sparse training results, better than the dense-to-sparse methods, with various types of RNNs on Penn TreeBank and Wikitext-2 datasets. Our codes are available at https://github.com/Shiweiliuiiiiiii/Selfish-RNN.

[1]Department of Computer Science, Eindhoven University of Technology, the Netherlands
[2]Faculty of Electrical Engineering, Mathematics, and Computer Science at University of Twente, the Netherlands
. Correspondence to: Shiwei Liu <s.liu3@tue.nl>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

1. Introduction

Recurrent neural networks (RNNs) (Elman, 1990), with a variant of long short-term memory (LSTM) (Hochreiter & Schmidhuber, 1997), have been highly successful in various fields, including language modeling (Mikolov et al., 2010), machine translation (Kalchbrenner & Blunsom, 2013), question answering (Hirschman et al., 1999; Wang & Jiang, 2017), etc. As a standard task to evaluate models' ability to capture long-range context, language modeling has witnessed great progress in RNNs. Mikolov et al. (2010) demonstrated that RNNs perform much better than backoff models for language modeling. After that, various novel RNN architectures such as Recurrent Highway Networks (RHNs) (Zilly et al., 2017), Pointer Sentinel Mixture Models (Merity et al., 2017), Neural Cache Model (Grave et al., 2017), Mixture of Softmaxes (AWD-LSTM-MoS) (Yang et al., 2018), Ordered Neurons LSTM (ON-LSTM) (Shen et al., 2019), and effective regularization like Variational Dropout (Gal & Ghahramani, 2016), Weight Tying (Inan et al., 2017), DropConnect (Merity et al., 2018) have been proposed to improve the performance of RNNs on language modeling.

At the same time, as the performance of deep neural networks (DNNs) improves, the resources required to train and deploy these deep models are becoming prohibitively large. To tackle this problem, various dense-to-sparse methods have been developed, including but not limited to pruning (LeCun et al., 1990; Han et al., 2015; Molchanov et al., 2016), variational dropout (Kingma et al., 2015; Molchanov et al., 2017), distillation (Hinton et al., 2015), L_1 regularization (Wen et al., 2018), and low-rank decomposition (Jaderberg et al., 2014). Given a pre-trained model, these methods work effectively to accelerate the inference process.

Recently, some dynamic sparse training (DST) approaches (Mocanu et al., 2018; Mostafa & Wang, 2019; Dettmers & Zettlemoyer, 2019; Evci et al., 2020) have been proposed to bring efficiency to the training phase as well. However, previous approaches are mainly for CNNs and MLPs. The long-term dependencies and repetitive usage of recurrent cells make RNNs more difficult to be sparsified (Kalchbrenner et al., 2018; Evci et al., 2020). More importantly, the state-of-the-art performance achieved by RNNs on language modeling is mainly associated with the optimizer, averaged

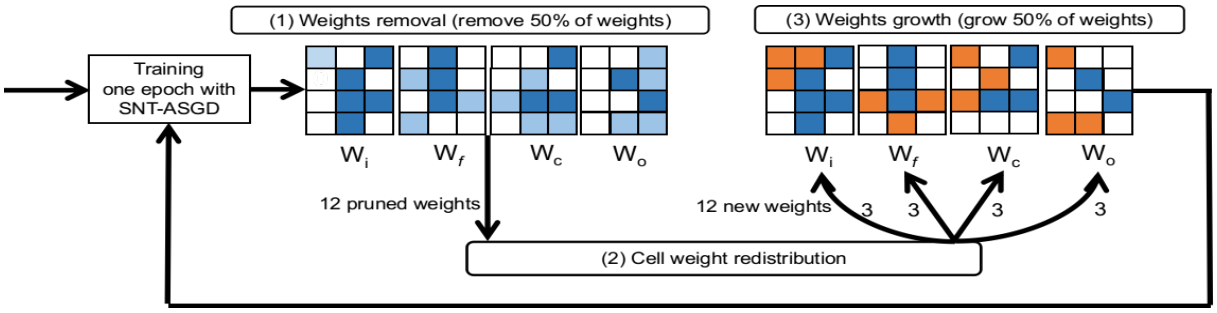

Figure 1. Schematic diagram of the Selfish-RNN. W_i, W_f, W_c, W_o refer to LSTM cell gates. Colored squares and white squares refer to nonzero weights and zero weights, respectively. Light blue squares are weights to be removed and orange squares are weights to be grown.

stochastic gradient descent (ASGD) (Polyak & Juditsky, 1992), which is not compatible with the existing DST approaches. The abovementioned problems heavily limit the performance of the off-the-shelf sparse training methods in the RNN field. For instance, while "The Rigged Lottery" (RigL) achieves state-of-the-art sparse training results with various CNNs, it fails to match the performance of the iterative pruning method (Gale et al., 2019) in the RNN setting (Evci et al., 2020).

In this paper, we propose an algorithm to train initially sparse RNNs with a fixed number of parameters throughout training. We abbreviate our sparse RNN training method as Selfish-RNN because our method encourages cell gates to obtain their parameters selfishly. The main contributions of this work are four-fold:

- We propose an algorithm to train sparse RNNs from scratch with a fixed number of parameters. Our method has two novelty components: (1) we propose SNT-ASGD, a sparse variant of the non-monotonically triggered averaged stochastic gradient descent optimizer (NT-ASGD), which improves the performance of all sparse training methods for RNNs; (2) we allow RNN layers to have a non-uniform redistribution across cell gates during training for a better regularization.

- We demonstrate state-of-the-art sparse training performance, better than the dense-to-sparse methods, with various RNN models, including stacked LSTMs (Zaremba et al., 2014), RHNs, ON-LSTM on Penn TreeBank (PTB) dataset (Marcus et al., 1993) and AWD-LSTM-MoS on WikiText-2 dataset (Melis et al., 2018).

- We present an approach to measure the topological distance between different sparse connectivities from the perspective of graph theory. Recent works (Garipov et al., 2018; Draxler et al., 2018; Fort & Jastrzebski, 2019) on understanding dense loss landscapes find that many independent optima are located in different low-loss tunnels. We complement these works by showing that there exist many low-loss sparse networks which are very different in the topological space.

- Our analysis shows two surprising phenomena in the setting of RNNs contrary to CNNs (1) random-based weight growth performs better than gradient-based weight growth, and (2) uniform sparse distribution performs better than *Erdős-Rényi* (ER) sparse distribution. These results highlight the need to choose different sparse training methods for different architectures.

2. Related Work

Dense-to-Sparse. There are a large number of works operating on a dense network to yield a sparse network. We divide them into three categories based on the training cost in terms of memory and computation.
(1) *Iterative Pruning and Retraining.* To the best of our knowledge, pruning was first proposed by Janowsky (1989) and Mozer & Smolensky (1989) whose goal is to yield a sparse network from a pre-trained network for sparse inference. Han et al. (2015) brought it back to people's attention based on the idea of iterative pruning and retraining with modern architectures. Gradual Magnitude Pruning (GMP) (Zhu & Gupta, 2017; Gale et al., 2019) was further proposed to obtain the target sparse model in one running. Recently, Frankle & Carbin (2019) proposed the Lottery Ticket Hypothesis showing that the sub-networks ("winning tickets") obtained via iterative magnitude pruning combined with their "lucky" initialization can outperform the dense networks. Zhou et al. (2019) found that the sign of the initial weights is the key factor that makes the "winning tickets" work. Our work shows that there exists a much more efficient and robust way to find those "winning tickets" without any pre-training steps and any specific initialization. Overall, iterative pruning and retraining requires at least the same training cost as training a dense model, sometimes even

Table 1. Comparison of different sparsity-inducing approaches in RNNs. ER and ERK refer to the *Erdős-Rényi* distribution and the *Erdős-Rényi-Kernel* distribution, respectively. *Backward Sparse* means a clean sparse backward pass and no need to compute or store any information of the non-existing weights. *Sparse Opt* indicates whether a specific optimizer is proposed for sparse RNN training.

Method	Initialization	Removal	Growth	Weight Redistribution	Backward Sparse	Sparse Opt
Iterative Pruning	dense	$min(\|\theta\|)$	none	no	no	no
ISS	dense	Lasso	none	no	no	no
SET	ER	$min(\|\theta\|)$	random	no	yes	no
DSR	uniform	$min(\|\theta\|)$	random	across all layers	yes	no
SNFS	uniform	$min(\|\theta\|)$	momentum	across all layers	no	no
RigL	ERK	$min(\|\theta\|)$	gradient	no	no	no
Selfish-RNN	uniform	$min(\|\theta\|)$	random	across RNN cell gates	yes	yes

more, as a pre-trained dense model is involved. We compare our method with the state-of-the-art pruning method proposed by Zhu & Gupta (2017) in Appendix I. With fewer training costs, our method is able to discover sparse networks with lower test perplexity.

(2) *Learning Sparsity During Training.* Some works attempt to learn the sparse networks during training. Louizos et al. (2017) and Wen et al. (2018) were examples that gradually enforce the network weights to zero via L_0 and L_1 regularization, respectively. Dai et al. (2018) proposed the idea of using singular value decomposition (SVD) to accelerate the training process for LSTMs. Recent works (Liu et al., 2020a; Srivastava et al., 2015; Xiao et al., 2019; Kusupati et al., 2020) induce sparsity by jointly learning masks and model weights during training. These methods start with a fully dense network, and hence are not memory efficient.

(3) *Pruning at Initialization.* Some works aim to find sparse neural networks by pruning once prior to the main training phase based on some salience criteria, including connection sensitivity (Lee et al., 2019; 2020), synaptic flow (Tanaka et al., 2020), gradient signal preservation (Wang et al., 2020), and iterative pruning (de Jorge et al., 2020). These techniques can find sparse networks before the standard training, but at least one iteration of dense training is involved to identify these trainable sparse networks. Additionally, pruning at initialization generally cannot match the performance of dynamic sparse training, especially at extreme sparsity levels (Wang et al., 2020).

Sparse-to-Sparse. Recently, many works have emerged to train intrinsically sparse neural networks from scratch to obtain efficiency both for training and inference.

(1) *Static Sparse Training.* Mocanu et al. (2016) developed intrinsically sparse networks by exploring the scale-free and small-world topological properties in Restricted Boltzmann Machines. Later, some works focus on designing sparse CNNs based on Expander graphs and show comparable performance against the corresponding dense models (Prabhu et al., 2018; Kepner & Robinett, 2019).

(2) *Dynamic Sparse Training.* Mocanu et al. (2018) proposed Sparse Evolutionary Training (SET) allowing sparse training MLPs to match the performance of dense MLPs by dynamically changing the sparse connectivity based on a simple remove-and-regrow strategy. Shortly after, DeepR was introduced by Bellec et al. (2018) to train sparse networks by sampling the sparse connectivity from the posterior distribution. Mostafa & Wang (2019) introduced Dynamic Sparse Reparameterization (DSR) to train sparse neural networks while dynamically adjusting the sparse distribution during training. Sparse Networks from Scratch (SNFS) (Dettmers & Zettlemoyer, 2019) improved the sparse training performance by introducing the idea of growing free weights according to their momentum. While effective, it requires extra computation and memory to update the dense momentum tensor for every iteration. RigL (Evci et al., 2020) went one step further by activating new weights with the highest magnitude gradient. It amortizes the significant amount of overhead by updating the sparse connectivity infrequently. In addition, some recent works (Jayakumar et al., 2020; Raihan & Aamodt, 2020) attempt to explore more sparse space during training to improve the sparse training performance. Due to the inherent limitations of deep learning software and hardware libraries, all of those works simulate sparsity using a binary mask over weights. More recently, Liu et al. (2020b) proved the practical values of DST by developing for the first time an independent software framework to train truly sparse MLPs with over one million neurons on a typical laptop. However, all these works mainly focus on CNNs and MLPs, and they are not designed to match state-of-the-art performance for RNNs.

We summarize the properties of all approaches compared in this paper in Table 1. Additionally, we provide a high-level overview of the difference between Selfish-RNN and iterative pruning and re-training in Figure 2. FLOPs required by Selfish-RNN is much smaller than iterative pruning and re-training, as it starts with a sparse network and without any retraining phases. See Appendix H for more differences.

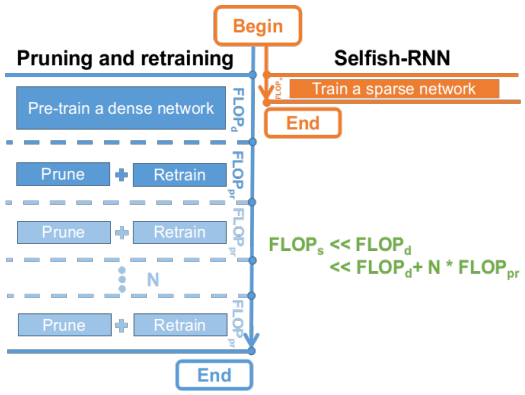

Figure 2. A high-level overview of the difference between Selfish-RNN and iterative pruning and re-training techniques. Blocks with light blue color represent optional pruning and retraining steps chosen depending on specific approaches.

Algorithm 1 Selfish-RNN

Input: Model weight θ, number of layer L, sparsity S, pruning rate p, training epoch n
for $i = 1$ **to** L **do**
  Initialize the network with Eq. (1)
end for
for $epoch = 1$ **to** n **do**
  Training: $\theta_s \leftarrow$ SNT-ASGD(θ_s)
  for $i = 1$ **to** L **do**
    if RNN layer **then**
      Cell Gate redistribution with Eq. (4)
    else
      Weight removal with Eq. (2)
      Weight growth with Eq. (3)
    end if
  end for
  $p \leftarrow$DecayRemovingRate(p)
end for

3. Sparse RNN Training

Our sparse RNN training method is illustrated in Figure 1 with LSTM as a specific case. Note that our method can be easily applied to any other RNN variants. The only difference is the number of cell gates. Before training, we randomly initialize each layer at the same sparsity (the fraction of zero-valued weights), so that the training costs are proportional to the dense model at the beginning. To explore more sparse structures, while to maintain a fixed sparsity level, we need to optimize the sparse connectivity together with the corresponding weights (a combinatorial optimization problem). We use dynamic sparse connectivity and SNT-ASGD together to handle this combinatorial optimization problem. The pseudocode of the full training procedure of our algorithm is shown in Algorithm 1.

3.1. Dynamic Sparse Connectivity

We consider uniform sparse initialization, magnitude weight removal, random weight growth, and cell gate redistribution together as main components of our dynamic sparse training method, which can ensure a fixed number of parameters and a clean sparse backward pass, as discussed next.

Notation. Given a dataset of N samples $\mathbf{D} = \{(x_i, y_i)\}_{i=1}^{N}$ and a network $f(x;\theta)$ parameterized by θ. We train the network to minimize the loss function $\sum_{i=1}^{N} L(f(x_i;\theta), y_i)$. The basic mechanism of sparse training is to train with a fraction of parameters θ_s, while preserving the performance as much as possible. Hence, a sparse neural network can be denoted as $f_s(x;\theta_s)$ with a sparsity level $S = 1 - \frac{\|\theta_s\|_0}{\|\theta\|_0}$, where $\|\cdot\|_0$ is the ℓ_0-norm.

Uniform Sparse Initialization. First, the network is randomly initialized with a uniform sparse distribution in which the sparsity level of each layer is the same S. More precisely, the network is initialized by:

$$\theta_s = \theta \odot M \tag{1}$$

where θ is a dense weight tensor initialized in a standard way; M is a binary tensor, in which nonzero elements are sampled uniformly based on the sparsity S; $\odot$ refers to the Hadamard product.

Magnitude Weight Removal. For non-RNN layers, we use magnitude weight removal followed by random weight growth to update the sparse connectivity. We remove a fraction p of weights with the smallest magnitude after each training epoch. This step is performed by changing the binary tensor M, as follows:

$$M = M - P \tag{2}$$

where P is a binary tensor with the same shape as M, in which the nonzero elements have the same indices with the top-p smallest-magnitude nonzero weights in θ_s, with $||P||_0 = p||M||_0$.

Random Weight Growth. To keep a fixed parameter count, we randomly grow the same number of weights immediately after weight removal, by:

$$M = M + R \tag{3}$$

where R is a binary tensor where the nonzero elements are randomly located at the position of zero elements of M. We choose random growth to get rid of using any information of the non-existing weights, so that both feedforward and backpropagation are completely sparse. It is more desirable to have such pure sparse structures as it enables the possibility of conceiving in the future specialized hardware accelerators for sparse neural networks. Besides, our analysis of

growth methods in Section 5.2 shows that random growth can explore more sparse structural degrees of freedom than gradient growth, which might be crucial to sparse training.

Cell Gate Redistribution. Our dynamic sparse connectivity differs from previous methods mainly in cell gate redistribution. For non-RNN layers, we use magnitude weight removal followed by random weight growth to update the sparse connectivity. For RNN layers, we use cell gate redistribution to update their sparse connectivities. The naive approach is to sparsify all cell gates independently at the same sparsity, as used in Liu et al. (2019) which is a straightforward SET extension to RNNs. Essentially, it is more desirable to redistribute new weights to cell gates dependently, as all gates collaborate together to regulate information. Intuitively, we redistribute new weights in a way that cell gates containing more large-magnitude weights should have more weights. Large-magnitude weights indicate that their loss gradients are large and few oscillations occur. Therefore, gates with more large-magnitude weights should be reallocated with more parameters to accelerate training. Concretely, for each RNN layer l, we remove weights dependently given by an ascending sort:

$$Sort_p(|\theta_1^l|, |\theta_2^l|, .., |\theta_t^l|) \tag{4}$$

where $\{\theta_1^l, \theta_2^l, ..., \theta_t^l\}$ are all gate tensors within RNN cell, and $Sort_p$ returns p indices of the smallest-magnitude weights. After weight removal, new weights are uniformly grown to each gate, so that gates with more large-magnitude weights will gradually obtain more weights. We further demonstrate the final sparsity breakdown of cell gates learned by our method in Appendix M and observe that the forget gates are consistently sparser than other gates for all models.

Similar with SNFS, We also decay the pruning rate p to zero with a cosine annealing. We further use Eq. (1) to enforce the sparse structure before the forward pass and after the backward pass, so that zero weights will not contribute to the loss. And all newly activated weights are initialized to zero.

3.2. Sparse Non-monotonically Triggered ASGD

Non-monotonically Triggered ASGD (NT-ASGD) has shown its ability to achieve remarkable performance with various RNNs (Merity et al., 2018; Yang et al., 2018; Shen et al., 2019). However, it becomes less appealing for sparse RNNs training. Unlike dense networks in which every parameter in the model is updated at each iteration, the non-active weights remain zero for sparse training. Once these non-active weights are activated, the original averaging operation of standard NT-ASGD will immediately bring them close to zero. Thereby, after the averaging operation is triggered, the number of valid weights will decrease sharply. To alleviate this problem, we describe SNT-ASGD as following:

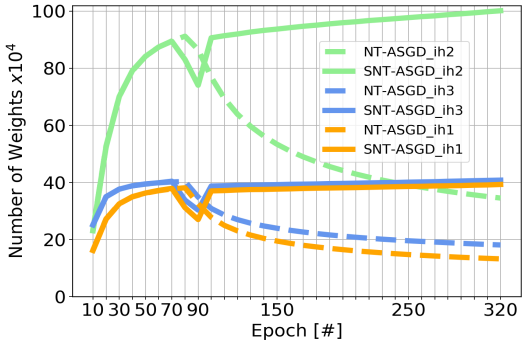

Figure 3. The number of weights whose magnitude is larger than 0.1 during training for ON-LSTM. The solid lines represent SNT-ASGD and dashed lines represent standard NT-ASGD. *ih1*, *ih2*, and *ih3* refer to input weights in the first, second and third LSTM layer.

$$\tilde{w}_i = \begin{cases} 0 & if\ m_i = 0, \forall i, \\ \frac{\sum_{t=T_i}^{K} w_{i,t}}{(K - T_i + 1)} & if\ m_i = 1, \forall i. \end{cases} \tag{5}$$

where $\tilde{w}_i$ is the value returned by SNT-ASGD for weight w_i; $w_{i,t}$ represents the actual value of weight w_i at the t^{th} iteration; $m_i = 1$ if the weight w_i exists and $m_i = 0$ if not; T_i is the iteration in which the weight w_i grows most recently; and K is the total number of iterations. We demonstrate the effectiveness of SNT-ASGD in Figure 3. At the beginning when trained with SGD, the number of weights with high magnitude increases fast. However, the trend of NT-ASGD starts to descend significantly once the averaging is triggered at epoch 80, whereas the trend of SNT-ASGD continues to rise after a small drop caused by the averaging operation.

Besides, the constant learning rate of SNT-ASGD helps to prevent the negative effect of learning rate decay on dynamic sparse training. Since most dynamic sparse training methods initialize the newly activated weights as zero, the magnitude of new weights can barely catch up with the unpruned weights once the learning rate is decayed to small values. To better understand how proposed components, cell gate redistribution and SNT-ASGD, improve the sparse RNN training performance, we further conduct an ablation study in Appendix B. It is clear to see that both of them lead to significant performance improvement.

4. Experimental Results

We evaluate Selfish-RNN with various models including stacked LSTMs, RHNs, ON-LSTM on the Penn TreeBank dataset and AWD-LSTM-MoS on the WikiText-2 dataset. The performance of Selfish-RNN is compared with 6 state-of-the-art sparse inducing techniques, including 2 dense-to-sparse methods, ISS and GMP; 4 sparse-to-sparse methods,

Table 2. Single model perplexity on validation and test sets for the Penn Treebank language modeling task with stacked LSTMs and RHNs. '*' indicates the reported results from the original papers: "Dense" is obtained from Zaremba et al. (2014) and ISS is obtained from Wen et al. (2018). "Static-ER" and "Static-Uniform" are the static sparse network trained from scratch with ER distribution and uniform distribution, respectively. "Small Dense" refers to the small dense network with the same number of parameters as Selfish-RNN.

	Stacked LSTMs			RHNs		
Models	#Parameters	Validation	Test	#Parameters	Validation	Test
Dense*	66.0M	82.57	78.57	23.5M	67.90	65.40
Dense (NT-ASGD)	66.0M	74.51	72.40	23.5M	63.44	61.84
	Sparsity=0.67			Sparsity=0.53		
Small Dense (NT-ASGD)	21.8M	88.67	86.33	11.1M	70.10	68.40
Static-ER (SNT-ASGD)	21.8M	81.02	79.30	11.1M	75.74	73.21
Static-Uniform (SNT-ASGD)	21.8M	80.37	78.61	11.1M	74.11	71.83
ISS*	21.8M	82.59	78.65	11.1M	68.10	65.40
DSR (Adam)	21.8M	89.95	88.16	11.1M	65.38	63.19
GMP (Adam)	21.8M	89.47	87.97	11.1M	63.21	61.55
SNFS (Adam)	21.8M	88.31	86.28	11.1M	74.02	70.99
RigL (Adam)	21.8M	88.39	85.61	11.1M	67.43	64.41
SET (Adam)	21.8M	87.30	85.49	11.1M	63.66	61.08
Selfish-RNN (Adam)	21.8M	85.70	82.85	11.1M	63.28	60.75
RigL (SNT-ASGD)	21.8M	78.31	75.90	11.1M	64.82	62.47
GMP (SNT-ASGD)	21.8M	76.78	74.84	11.1M	65.63	63.96
Selfish-RNN (SNT-ASGD)	21.8M	**73.76**	**71.65**	11.1M	**62.10**	**60.35**
	Sparsity=0.62			Sparsity=0.68		
ISS*	25.2M	80.24	76.03	7.6M	70.30	67.70
RigL (SNT-ASGD)	25.2M	77.16	74.76	7.6M	69.32	66.64
GMP (SNT-ASGD)	25.2M	74.86	73.03	7.6M	66.61	64.98
Selfish-RNN (SNT-ASGD)	25.2M	**73.50**	**71.42**	7.6M	**66.35**	**64.03**

SET, DSR, SNFS, and RigL. Intrinsic Sparse Structures (ISS) (Wen et al., 2018) is a method that uses Lasso regularization to explore sparsity inside RNNs. GMP is the state-of-the-art unstructured pruning method in DNNs. For fair comparison, we use the exact same hyperparameters and regularization introduced in ON-LSTM (Shen et al., 2019) and AWD-LSTM-MoS (Yang et al., 2018). We then extend these similar settings to stacked LSTMs and RHNs. We choose Adam (Kingma & Ba, 2014) and SNT-ASGD as the optimizers of different DST methods. Due to the space limitation, we put the results of sparse ON-LSTM on PTB and sparse AWD-LSTM-MoS on Wikitext-2 in Appendix D and Appendix E, respectively. See Appendix A for hyperparameters.

4.1. Stacked LSTMs

As introduced by Zaremba et al. (2014), stacked LSTMs (large) is a two-layer LSTM model with 1500 hidden units for each LSTM layer. We choose the same sparsity as ISS, 67% and 62%. Results are shown in the left side of Table 2 (see Appendix L for the version with estimated FLOPs). We provide a new dense baseline trained with the standard NT-ASGD, achieving 6 lower test perplexity than the widely-used baseline. When optimized by Adam, while Selfish-RNN achieves the lowest perplexity, all sparse training techniques fail to match the performance of ISS and dense models. On the other hand, training sparse RNNs with SNT-ASGD substantially improves the performance of all sparse methods, and Selfish-RNN achieves the best one, even better than the new dense baseline. Note that even starting from a sparse network (sparse-to-sparse), our method can discover a better solution than the state-of-the-art dense-to-sparse method, GMP. We also test whether a small dense network and a static sparse network with the same number of parameters as Selfish-RNN can match the performance of Selfish-RNN. We train a dense stacked LSTMs with 700 hidden units, named as "Small Dense". In line with the previous studies (Mocanu et al., 2018; Mostafa & Wang, 2019; Evci et al., 2020), both static sparse networks and small dense networks fall short of Selfish-RNN. Moreover, training a static sparse network from scratch with uniform distribution performs better than the one with *ER* distribution.

To understand better the effect of different optimizers on different DST methods, we report the performance of all DST methods trained with Adam, momentum SGD, and SNT-ASGD. For SNFS (SNT-ASGD), we replace momentum of weights with their gradients, as SNT-ASGD does not

Table 3. Effect of different optimizers on different DST methods. We choose stacked LSTMs on PTB dataset at a sparsity level of 67%.

MODELS	#PARAMETERS	VALIDATION	TEST
DENSE	66.0M	82.57	78.57
		ADAM	
DSR	21.8M	89.95	88.16
SNFS	21.8M	88.31	86.28
RIGL	21.8M	88.39	85.61
SET	21.8M	87.30	85.49
SELFISH-RNN	21.8M	**85.70**	**82.85**
		SGD WITH MOMENTUM	
SNFS	21.8M	90.09	87.98
SET	21.8M	85.73	82.52
RIGL	21.8M	84.78	80.81
DSR	21.8M	82.89	80.09
SELFISH-RNN	21.8M	**82.48**	**79.69**
		SNT-ASGD	
SNFS	21.8M	82.11	79.50
RIGL	21.8M	78.31	75.90
SET	21.8M	76.78	74.84
SELFISH-RNN	21.8M	73.76	71.65
DSR	21.8M	**72.30**	**70.76**

involve any momentum terms. We use the same hyperparameters for all DST methods. The results are shown in Table 3. It is clear that SNT-ASGD brings significant perplexity improvements to all sparse training techniques. This further stands as empirical evidence that SNT-ASGD is crucial to improve the sparse training performance in the RNN setting. Moreover, compared with other DST methods, Selfish-RNN is quite robust to the choice of optimizers likely due to its simple scheme to update the sparse connectivity. Advanced strategies such as across-layer weight redistribution used in DSR and SNFS, gradient-based weight growth used in RigL and SNFS heavily depend on optimizers. They might work decently for some optimization methods but may not for others.

4.2. Recurrent Highway Networks

Recurrent Highway Networks (Zilly et al., 2017) is a variant of RNNs allowing RNNs to explore deeper architectures inside the recurrent transition. The results are shown in the right side of Table 2. Again, Selfish-RNN achieves the lowest perplexity with both Adam and SNT-ASGD, better than the dense-to-sparse methods (ISS and GMP). Surprisingly, random-based growth methods (SET, DSR, and Selfish-RNN) consistently have the lower perplexity than the gradient-based growth methods (RigL and SNFS). We further analyze the effect of different weight growth methods on DST in Section 5.2.

5. Analyzing the Performance of Selfish-RNN

5.1. Analysis of Topological Distance between Sparse Connectivities

Prior works on understanding dense loss landscapes have shown the existence of diverse low-loss solutions on the manifold of dense networks (Goodfellow et al., 2014; Garipov et al., 2018; Draxler et al., 2018; Fort & Jastrzebski, 2019). Here, the fact that Selfish-RNN consistently achieves good performance with different runs naturally raises some questions: e.g., are final sparse connectivities obtained by different runs similar or very different? Is the distance between the original sparse connectivity and the final sparse connectivity large or small? To answer these questions, we investigate a method based on graph edit distance (GED) (Sanfeliu & Fu, 1983) to measure the topological distance between different sparse connectivities. The distance is scaled between 0 and 1. The smaller the distance is, the more similar the two sparse topologies are (See Appendix J for details on how we measure the sparse connectivity distance).

The results are demonstrated in Figure 4. Figure 4-left shows how the topology of one random-initialized (random sparse connectivity and random weight values) network evolves when trained with Selfish-RNN. We compare the sparse connectivity at 5 epoch with those at the following epochs. We can see that the distance gradually increases from 0 to a very high value 0.8, meaning that Selfish-RNN optimizes the initial topology to a very different one. Moreover, Figure 4-right illustrates that the topological distance between two same-initialized (same sparse connectivity and same weight values) networks trained with different seeds after the 4^{th} epoch. Started from the same sparse topology, they evolve to completely different sparse topologies. While topologically different, they have very similarly performance. This indicates that in the case of RNNs there exist many low-dimensional sub-networks that can achieve similarly low loss. This phenomenon complements the findings of Liu et al. (2020c) which shows that there are numerous sparse sub-networks performing similarly well in the context of sparse MLPs.

5.2. Analysis of Weight Growth Methods

Methods that leverage gradient-based weight growth (SNFS and RigL) have shown superiority on performance over the methods using random-based weight growth for CNNs (Evci et al., 2020). However, we observe a different behavior with RNNs. We set up a controlled experiment to compare these two methods with SNT-ASGD and momentum SGD. We report the results with various update intervals (the number of iterations between sparse connectivity updates) in Figure 5. Surprisingly, gradient-based growth performs worse than random-based growth in most cases. Our hypothesis

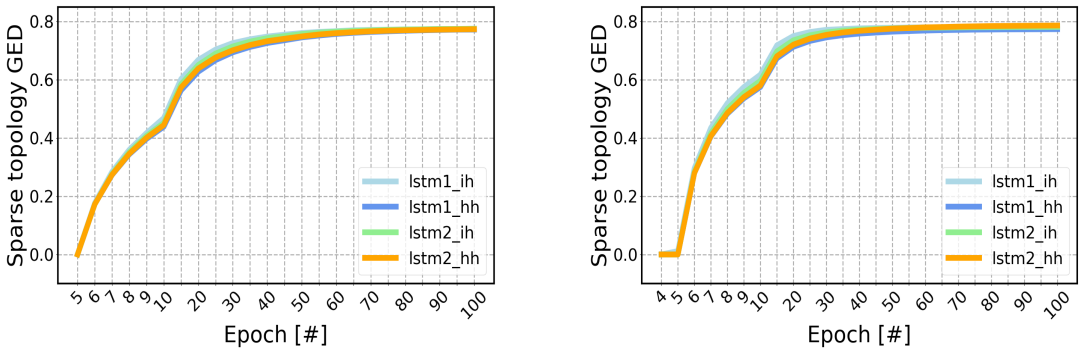

Figure 4. (**left**) One random-initialized sparse stacked LSTMs trained with Selfish-RNN end up with a very different sparse connectivity topology. (**right**) Two same-initialized sparse stacked LSTMs trained with different random seeds end up with very different sparse connectivity topologies. ih is the input weight tensor comprising four cell gates and hh is the hidden state weight tensor comprising four cell gates.

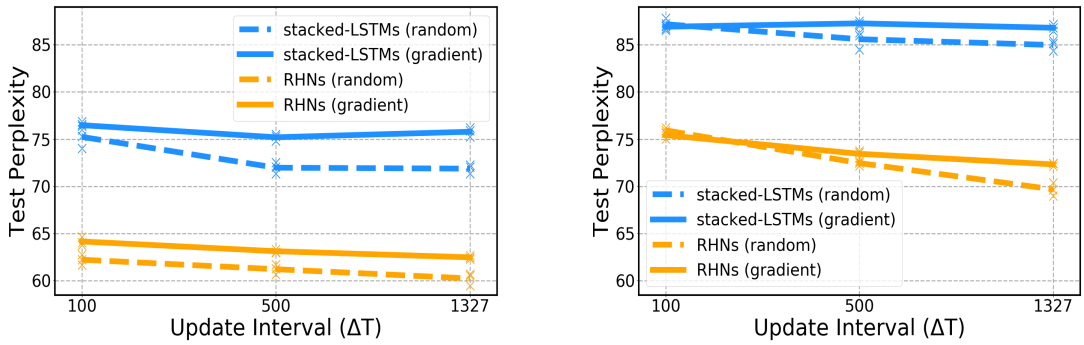

Figure 5. Comparison between random-based growth and gradient-based growth. (**left**) Models trained with SNT-ASGD. (**right**) Models trained with momentum SGD.

is that random growth helps in exploring better the search space, as it naturally considers a large number of various sparse connectivities during training, which is crucial to the performance of dynamic sparse training. Differently, gradient growth drives the network topology towards some similar local optima for the sparse connectivity as it uses a greedy search strategy (highest gradient magnitude) at every topological change. However, benefits provided by high-magnitude gradients might change dynamically afterwards due to complicated interactions between weights. We empirically illustrate our hypothesis via the proposed sparse connectivity distance measurement in Appendix K.

5.3. Analysis of Sparse Initialization

It has been shown that the choice of sparse initialization (sparsity distribution) is important for sparse training in Frankle & Carbin (2019); Kusupati et al. (2020); Evci et al. (2020). Here, we compare two types of sparse initialization for RNNs, ER distribution and uniform distribution. Uniform distribution namely enforces the sparsity level of each layer to be the same as S. ER distribution allocates higher sparsity to larger layers than smaller ones. Note that its variant *Erdős-Rényi-Kernel* proposed by Evci et al. (2020) scales back to ER for RNNs, as no kernels are involved. The results are shown as the *Static* group in Table 2. We can see that uniform distribution outperforms ER distribution consistently for RNNs.

5.4. Analysis of DST Hyperparameters

The sparsity S and the initial pruning rate p are two hyperparameters of our DST method. We show their sensitivity analysis in Appendix F and Appendix G. We find that Selfish Stacked LSTMs, RHNs, ON-LSTM, and AWD-LSTM-MoS need around 25%, 40%, 45%, and 40% parameters to reach the performance of their dense counterparts, respectively. In addition, our method is quite robust to the choice of the initial pruning rate.

6. Conclusion

In this paper, we developed an approach to train sparse RNNs from scratch with a fixed parameter count throughout training. We further proposed SNT-ASGD, a specially designed sparse optimizer for training sparse RNNs and showed that it substantially improves the performance of all DST methods in RNNs. We observed that random-based growth achieves lower perplexity than gradient-based growth in the case of RNNs. Further, we developed an approach to compare two different sparse connectivities from the perspective of graph theory. Using this approach, we found that random-based growth explores better the topological search space for optimal sparse connectivities, whereas gradient-based growth is prone to drive the network towards similar sparse connectivity patterns, opening the path for a better understanding of sparse training.

References

Bellec, G., Kappel, D., Maass, W., and Legenstein, R. Deep rewiring: Training very sparse deep networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BJ_wN01C-.

Dai, R., Li, L., and Yu, W. Fast training and model compression of gated rnns via singular value decomposition. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7. IEEE, 2018.

de Jorge, P., Sanyal, A., Behl, H. S., Torr, P. H., Rogez, G., and Dokania, P. K. Progressive skeletonization: Trimming more fat from a network at initialization. *arXiv preprint arXiv:2006.09081*, 2020.

Dettmers, T. and Zettlemoyer, L. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

Draxler, F., Veschgini, K., Salmhofer, M., and Hamprecht, F. A. Essentially no barriers in neural network energy landscape. *arXiv preprint arXiv:1803.00885*, 2018.

Elman, J. L. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Evci, U., Gale, T., Menick, J., Castro, P. S., and Elsen, E. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, 2020.

Fort, S. and Jastrzebski, S. Large scale structure of neural network loss landscapes. In *Advances in Neural Information Processing Systems*, pp. 6709–6717, 2019.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

Gal, Y. and Ghahramani, Z. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pp. 1019–1027, 2016.

Gale, T., Elsen, E., and Hooker, S. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Garipov, T., Izmailov, P., Podoprikhin, D., Vetrov, D. P., and Wilson, A. G. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Advances in Neural Information Processing Systems*, pp. 8789–8798, 2018.

Goodfellow, I. J., Vinyals, O., and Saxe, A. M. Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544*, 2014.

Grave, E., Joulin, A., and Usunier, N. Improving neural language models with a continuous cache. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B184E5qee.

Gray, S., Radford, A., and Kingma, D. P. Gpu kernels for block-sparse weights. *arXiv preprint arXiv:1711.09224*, 3, 2017.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*, 2016.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Hirschman, L., Light, M., Breck, E., and Burger, J. D. Deep read: A reading comprehension system. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics*, pp. 325–332, 1999.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Inan, H., Khosravi, K., and Socher, R. Tying word vectors and word classifiers: A loss framework for language modeling. In *International Conference on Learning Representations*, 2017.

Jaderberg, M., Vedaldi, A., and Zisserman, A. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

Janowsky, S. A. Pruning versus clipping in neural networks. *Physical Review A*, 39(12):6600, 1989.

Jayakumar, S., Pascanu, R., Rae, J., Osindero, S., and Elsen, E. Top-kast: Top-k always sparse training. *Advances in Neural Information Processing Systems*, 33, 2020.

Kalchbrenner, N. and Blunsom, P. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1700–1709, 2013.

Kalchbrenner, N., Elsen, E., Simonyan, K., Noury, S., Casagrande, N., Lockhart, E., Stimberg, F., Oord, A. v. d., Dieleman, S., and Kavukcuoglu, K. Efficient neural audio synthesis. In *International Conference on Learning Representations*, 2018.

Kepner, J. and Robinett, R. Radix-net: Structured sparse matrices for deep neural networks. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 268–274. IEEE, 2019.

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

Kingma, D. P., Salimans, T., and Welling, M. Variational dropout and the local reparameterization trick. *arXiv preprint arXiv:1506.02557*, 2015.

Krause, B., Kahembwe, E., Murray, I., and Renals, S. Dynamic evaluation of neural sequence models. In *International Conference on Machine Learning*, pp. 2766–2775, 2018.

Kusupati, A., Ramanujan, V., Somani, R., Wortsman, M., Jain, P., Kakade, S., and Farhadi, A. Soft threshold weight reparameterization for learnable sparsity. In *International Conference on Machine Learning*, pp. 5544–5555. PMLR, 2020.

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

Lee, N., Ajanthan, T., and Torr, P. SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=B1VZqjAcYX.

Lee, N., Ajanthan, T., Gould, S., and Torr, P. H. S. A signal propagation perspective for pruning neural networks at initialization. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJeTo2VFwH.

Li, Y., Yosinski, J., Clune, J., Lipson, H., and Hopcroft, J. E. Convergent learning: Do different neural networks learn the same representations? In *Iclr*, 2016.

Liu, J., XU, Z., SHI, R., Cheung, R. C. C., and So, H. K. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. In *International Conference on Learning Representations*, 2020a. URL https://openreview.net/forum?id=SJlbGJrtDB.

Liu, S., Mocanu, D. C., and Pechenizkiy, M. Intrinsically sparse long short-term memory networks. *arXiv preprint arXiv:1901.09208*, 2019.

Liu, S., Mocanu, D. C., Matavalam, A. R. R., Pei, Y., and Pechenizkiy, M. Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. *Neural Computing and Applications*, pp. 1–16, 2020b.

Liu, S., van der Lee, T., Yaman, A., Atashgahi, Z., Ferrar, D., Sokar, G., Pechenizkiy, M., and Mocanu, D. Topological insights into sparse neural networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2020c.

Louizos, C., Welling, M., and Kingma, D. P. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2017.

Ma, R., Miao, J., Niu, L., and Zhang, P. Transformed l1 regularization for learning sparse deep neural networks. *Neural Networks*, 119:286–298, 2019.

Marcus, M., Santorini, B., and Marcinkiewicz, M. A. Building a large annotated corpus of english: The penn treebank. 1993.

Melis, G., Dyer, C., and Blunsom, P. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

Melis, G., Dyer, C., and Blunsom, P. On the state of the art of evaluation in neural language models. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=ByJHuTgA-.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.

Merity, S., Keskar, N. S., and Socher, R. Regularizing and optimizing LSTM language models. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SyyGPP0TZ.

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

Mocanu, D. C., Mocanu, E., Nguyen, P. H., Gibescu, M., and Liotta, A. A topological insight into restricted boltzmann machines. *Machine Learning*, 104(2):243–270, Sep 2016. ISSN 1573-0565. doi: 10.1007/s10994-016-5570-z. URL https://doi.org/10.1007/s10994-016-5570-z.

Mocanu, D. C., Mocanu, E., Stone, P., Nguyen, P. H., Gibescu, M., and Liotta, A. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 9 (1):2383, 2018.

Molchanov, D., Ashukha, A., and Vetrov, D. Variational dropout sparsifies deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2498–2507. JMLR. org, 2017.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

Moradi, R., Berangi, R., and Minaei, B. Sparsemaps: convolutional networks with sparse feature maps for tiny image classification. *Expert Systems with Applications*, 119: 142–154, 2019.

Mostafa, H. and Wang, X. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, 2019.

Mozer, M. C. and Smolensky, P. Using relevance to reduce network size automatically. *Connection Science*, 1(1): 3–16, 1989.

NVIDIA. Nvidia a100 tensor core gpu architecture. *Retrieved from: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf*, 2020.

Polyak, B. T. and Juditsky, A. B. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

Prabhu, A., Varma, G., and Namboodiri, A. Deep expander networks: Efficient deep networks from graph theory. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 20–35, 2018.

Raihan, M. A. and Aamodt, T. M. Sparse weight activation training. *arXiv preprint arXiv:2001.01969*, 2020.

Sanfeliu, A. and Fu, K.-S. A distance measure between attributed relational graphs for pattern recognition. *IEEE transactions on systems, man, and cybernetics*, (3):353–362, 1983.

Saxe, A. M., McClelland, J. L., and Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *ICLR*, 2014.

Shen, Y., Tan, S., Sordoni, A., and Courville, A. Ordered neurons: Integrating tree structures into recurrent neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=B1l6qiR5F7.

Srivastava, R. K., Greff, K., and Schmidhuber, J. Training very deep networks. In *Advances in neural information processing systems*, pp. 2377–2385, 2015.

Tanaka, H., Kunin, D., Yamins, D. L., and Ganguli, S. Pruning neural networks without any data by iteratively conserving synaptic flow. *arXiv preprint arXiv:2006.05467*, 2020.

Wang, C., Zhang, G., and Grosse, R. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SkgsACVKPH.

Wang, S. and Jiang, J. Machine comprehension using match-lstm and answer pointer. In *International Conference on Learning Representations*, 2017.

Wen, W., He, Y., Rajbhandari, S., Zhang, M., Wang, W., Liu, F., Hu, B., Chen, Y., and Li, H. Learning intrinsic sparse structures within long short-term memory. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rk6cfpRjZ.

Xiao, X., Wang, Z., and Rajasekaran, S. Autoprune: Automatic network pruning by regularizing auxiliary parameters. *Advances in neural information processing systems*, 32, 2019.

Yang, J. and Ma, J. Feed-forward neural network training using sparse representation. *Expert Systems with Applications*, 116:255–264, 2019.

Yang, Z., Dai, Z., Salakhutdinov, R., and Cohen, W. W. Breaking the softmax bottleneck: A high-rank RNN language model. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HkwZSG-CZ.

Zaremba, W., Sutskever, I., and Vinyals, O. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

Zhou, H., Lan, J., Liu, R., and Yosinski, J. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems*, pp. 3597–3607, 2019.

Zhu, M. and Gupta, S. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

Zilly, J. G., Srivastava, R. K., Koutník, J., and Schmidhuber, J. Recurrent highway networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 4189–4198. JMLR. org, 2017.