Neural Computing and Applications (2021) 33:9625–9636
https://doi.org/10.1007/s00521-021-05727-y

ORIGINAL ARTICLE

Efficient and effective training of sparse recurrent neural networks

Shiwei Liu[1] · Iftitahu Ni'mah[1] · Vlado Menkovski[1] · Decebal Constantin Mocanu[1,2] · Mykola Pechenizkiy[1]

Received: 28 October 2020 / Accepted: 8 January 2021 / Published online: 26 January 2021

Abstract
Recurrent neural networks (RNNs) have achieved state-of-the-art performances on various applications. However, RNNs are prone to be memory-bandwidth limited in practical applications and need both long periods of training and inference time. The aforementioned problems are at odds with training and deploying RNNs on resource-limited devices where the memory and floating-point operations (FLOPs) budget are strictly constrained. To address this problem, conventional model compression techniques usually focus on reducing inference costs, operating on a costly pre-trained model. Recently, dynamic sparse training has been proposed to accelerate the training process by directly training sparse neural networks from scratch. However, previous sparse training techniques are mainly designed for convolutional neural networks and multi-layer perceptron. In this paper, we introduce a method to train intrinsically sparse RNN models with a fixed number of parameters and floating-point operations (FLOPs) during training. We demonstrate state-of-the-art sparse performance with long short-term memory and recurrent highway networks on widely used tasks, language modeling, and text classification. We simply use the results to advocate that, contrary to the general belief that training a sparse neural network from scratch leads to worse performance than dense networks, sparse training with adaptive connectivity can usually achieve better performance than dense models for RNNs.

Keywords Dynamic sparse training · Sparse recurrent neural networks · Long short-term memory · Recurrent highway networks

1 Introduction

Recurrent neural networks (RNNs), as one of the most widespread neural network architectures, mainly focus on a wide variety of applications where data is sequential, such as text classification [38], language modeling [60], speech recognition [64], machine translation [56]. On the other hand, RNNs have also shown their notable success in image processing [17], including but not limited to text recognition in scenes [55], facial expression recognition [6], visual question answering [4], handwriting recognition [52]. As a well-known architecture of RNNs, LSTM [27] has been widely utilized to encode various input (e.g., image, text, audio and video) to improve the recognition performance [60]. LSTM's success is due to its twofold properties. The first one is its natural ability to memorize long-term information, like being deep in time, which fits very well with sequential data [61]. This ability is its main advantage compared with other mainstream networks such as Multilayer Perceptron (MLP), CNNs. Second, the exploding and vanishing gradient problems are mitigated by the multiplicative gates regularizing the information over different time steps.

Recently, ensemble models, obtained by stacking RNN layers together with other types of neural network architectures (MLP and CNN), have been shown to further improve the prediction accuracy [5, 8, 11, 13, 14, 16, 26, 41]. CNN is usually used as the feature extractor to encode spatial data and RNN is used to handle the temporal correlation of the encoded features. However, while these RNN-based models keep refreshing the records of various competitions, the cost required to train and deploy them is also increasing. Their success is

✉ Shiwei Liu
s.liu3@tue.nl

[1] Department of Mathematics and Computer Science, Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands

[2] Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, Twente, The Netherlands

often associated with expensive computations, large memory requests and slow processes in both phases, training and inference. For example, around 30% of the Tensor Processing Unit (TPU) workload in the Google cloud is caused by LSTMs [29]. The computation-intensive and memory-intensive make it very difficult to train and deploy these models on resource-limited devices. Different from other neural networks, RNNs are relatively more challenging to be compressed [18, 57]. To tackle this problem, many researchers have already proposed various methods, such as Sparse Variational Dropout (Sparse VD) [48], sparse regularization [57], distillation [10] and pruning [24, 50]. While all of them can achieve promising compression rates with negligible performance loss, efficiency can only be obtained for inference because very expensive dense networks are still required in the beginning.

Only very recently, dynamic sparse training (DST) was begun to be studied to enable training sparse neural networks from scratch, with a few approaches including Sparse Evolutionary Training (SET) [47], Dynamic Sparse Reparameterization (DSR) [49], Sparse Networks from Scratch (SNFS) [15], Rigged Lottery (RigL) [18]. Due to the limited support for sparse operations in the existing frameworks [1, 51], the sparse structures were enforced with binary masks in the above-mentioned works. Liu et al. [36] developed a truly sparse implementation of SET, which demonstrates practical values of DST by being able to train a sparse MLP model with more than one million neurons on commodity hardware. However, these methods mainly focus on feedforward networks (e.g., convolutional neural networks (CNNs), multi-layer perceptrons (MLPs)). The performance of DST in RNNs is still unclear.

In this paper, we propose sparse training of recurrent neural networks (ST-RNNs) to gain effectiveness and efficiency both on training and inference. Concretely, we initialize the network with a sparse topology and then apply an adaptive sparse connectivity technique to optimize the sparse topology during the training phase. In comparison with all conventional compression techniques discussed above, our method is sparse from the design phase, before training. Evaluated on both language modeling and text classification, our proposed method can achieve good performance under a strict parameter budget. The major contributions of this paper are:

- *Training and inference efficiency* We propose an algorithm to enable training intrinsically sparse RNNs with a fixed number of parameters without involving any dense pre-training steps. The training floating-point operations (FLOPs) and the parameter count are strictly proportional to the dense model throughout training.
- *Better performance* We empirically demonstrate that even with much smaller training FLOPs and inference FLOPs, the sparse RNN models yielded by our method are able to achieve state-of-the-art sparse training performance, even better than the dense models.

2 Related work

2.1 Text classification and language modeling

In this paper, we evaluate our method on two widely used tasks, text classification and language modeling. Text classification is a well-known topic for natural language processing to classify free-text documents into categories by detecting and extracting information in the text. Viewed as a classification problem, text classification can be dealt with by various machine learning techniques such as Naive Bayes [20], support vector machine [9] and neural networks [38]. Language modeling is also a well-explored evaluation of the model's ability to estimate the probability distribution of various linguistic units based on the previous sequence of input [30]. The conventional technique to solve this application is the n-gram, a model that assigns probabilities to sentences and sequences of words. Recently, recurrent neural networks have been applied to language modeling with surprising performance [60].

2.2 Sparse neural networks

There are various effective techniques to yield sparse neural networks, while preserving competitive performance. Here, we briefly discuss some of them below.

2.2.1 Pruning methods

Pruning as a classical model compression method has been widely used to produce different types of sparse neural networks such as restricted Boltzmann machines (RBMs) [45], MLPs, CNNs, and RNNs. By zeroing out the unimportant weights to the function computed by the neural network, pruning is able to achieve a proper compression ratio without substantial loss in performance. LeCun et al. [33] used second-derivative information to prune the unimportant weight whose deletion has the lowest damage to the model performance. Giles et al. [22] proposed a simple pruning and retraining strategy to get sparse recurrent neural networks (RNNs). However, prohibitive computation and many training iterations are the main disadvantages of these methods. Han et al. [25] enabled high compression ratios via a simple heuristic based on a magnitude pruning and retraining strategy. Based on the

pruning approach of [25], an efficient compression method focusing on LSTMs was introduced [24]. It can compress the LSTM model size by 20 × (10 × from pruning and 2 × from quantization) with negligible loss of the prediction accuracy. Moreover, Narang et al. [50] shrunk the post-pruning sparse LTSM size by 90% through a monotonically increasing threshold. Using a set of hyperparameters can determine the specific pruning thresholds for different layers. Besides, large-sparse models were demonstrated to outperform comparably sized small-dense models on deep CNNs and LSTM-based models [43], which empirically shows the benefits of sparse neural networks. There are also some works trying to analyze sparse neural networks [21, 65]. Frankle et al. [21] proposed the "lottery ticket hypothesis", claiming that pruning makes it possible to find the "winning tickets" that can reach the same or even better performance as the dense networks from randomly initialized, dense neural networks. And the initialization of the "winning tickets" contributes mainly to its success. Zhou et al. [65] further found that the sign of the initialization of "winning tickets" actually results in a good performance, not the corresponding weight values.

2.2.2 Sparse neural networks throughout training

Recently, some works of dynamic sparse training have emerged, allowing train sparse neural networks from the beginning. Mocanu et al. [46] introduced restricted Boltzmann machines with a fixed sparse scale-free and small-world connectivity to obtain sparse models. However, the connectivity graph does not evolve towards the optimal one without a topology optimization method. Based on Bayesian sampling, DeepR [7] can optimize both the training weights and the connectivity topology simultaneously. The convergence of the connectivity topology is guaranteed theoretically. Simple but effective magnitude-based weights pruning and random-based weights growing were proposed in SET [47]. In addition to the proper classification performance, it also helps to detect important input features. Cross-layer reallocation of weights was introduced in DSR [49]. Dynamic parameter reallocation is able to break the limit of the parameter budget of each layer and allocate more parameters to layers where the training loss is more quickly decreased. Proposed by Dettmers et al. [15], SNFS obtains consistent improvement in various CNN models by using the momentum information of zero-valued weights. However, calculating and storing the momentum of all weights result in extra computation and memory requests. More recently, RigL [18] introduced gradient-based regrowing to get rid of the extra computation and storage caused by SNFS. It demonstrates state-of-the-art sparse training performance in various CNNs. Reference [35] proposed a method to measure the distance between sparse connectivities obtained with dynamic sparse training from the perspective of graph theory. They empirically show that there are many different sparse connectivities achieving equally good performance.

Unfortunately, RNNs are out of the consideration of the approaches above-mentioned.

2.2.3 Other methods

Many other approaches also make significant contribution to sparse neural networks, including group Lasso regularization [57], L_0 Regularization [37], Variational Dropout [48] etc.

3 ST-RNNs

In this section, we introduce sparse training of recurrent neural networks (ST-RNNs), the new class of sparse recurrent neural network models which we are proposing in this paper. Without loss of generality, we choose ST-LSTM as a specific case to discuss here. Please note that our method can also be easily generalized to other RNNs variants.

The general architecture of ST-LSTM is the same as the one of the dense LSTM. The only difference is that we replace the dense weights of the gates with sparse ones. The gates (f_t, i_t and o_t) are the keys to optimally regulate the flow of information insides LSTM cells which can be formulated by Eq. 1.

$$\begin{aligned} i_t &= \sigma(x_t \cdot W_{xi} + h_{t-1} \cdot W_{hi} + b_i) \\ f_t &= \sigma(x_t \cdot W_{xf} + h_{t-1} \cdot W_{hf} + b_f) \\ o_t &= \sigma(x_t \cdot W_{xo} + h_{t-1} \cdot W_{ho} + b_o) \\ g_t &= \tanh(x_t \cdot W_{xg} + h_{t-1} \cdot W_{hg} + b_g) \\ c_t &= f_t \otimes c_{t-1} + i_t \otimes g_t \\ h_t &= o_t \otimes \tanh(c_t) \end{aligned} \tag{1}$$

where W_{xi}, W_{xf}, W_{xo}, W_{hi}, W_{hf} and W_{ho} are weight matrices; b_i, b_f, b_o and b_g are bias weights; x_t, h_t, g_t refer to the input, hidden state, memory cell state at step t; h_{t-1}, g_{t-1} refer to the hidden state, memory cell state at step $t-1$; $\otimes$ is element-wise multiplication and $\cdot$ is matrix multiplication; $\sigma(\cdot)$ is the sigmoid function and $\tanh(\cdot)$ is the hyperbolic tangent function.

We illustrate the ST-LSTM cell in Fig. 1. The connections within the LSTM cell are initialized to be sparse with a certain sparsity level under the parameters budget. Then, the sparse structure is optimized gradually during the training, since not every naively initialized sparse topology

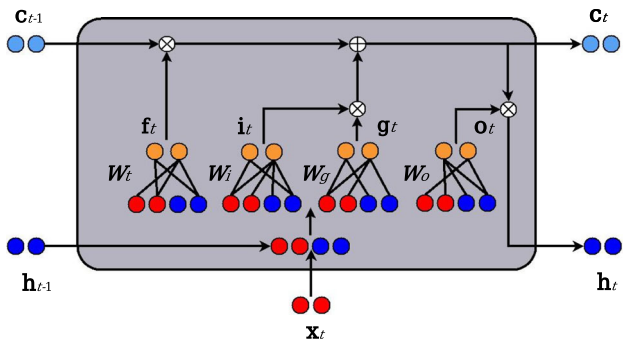

Fig. 1 Schematic diagram of the ST-LSTM (sparse training RNN) cell

is capable of reaching the competitive performance to the dense network [21]. In this paper, we apply a two-step strategy: weight pruning and weight regrowing, to optimize the sparse structure during the training. The cross-layer weights redistribution proposed in [49] is not applied in ST-RNNs, since it focuses on a fixed number of parameters rather than a fixed number of FLOPs. The full ST-RNNs pseudocode is shown in Algorithm 1. Every part of our algorithm is explained below.

limited support for sparse operations in the existing frameworks [1, 51]. Formally, the network is initialized by:

$$\theta_s = \theta * M \tag{3}$$

where M is the binary mask where the nonzero elements are initialized by a random distribution or the Erdős–Rényi distribution. Erdős–Rényi is introduced in [47] where the connection (M_{ij}^k) between neuron h_j^{k-1} and h_i^k exists with the probability:

$$p(M_{ij}^k) = \frac{\epsilon(n^k + n^{k-1})}{n^k n^{k-1}} \tag{4}$$

where n^k, n^{k-1} are the number of neurons of layer h^k and h^{k-1}, respectively; ϵ is a parameter determining the sparsity level s. The smaller ϵ is, the more sparse the topology is. Initialized by Erdős–Rényi topology, layers with larger weights matrices will have higher sparsity than the smaller ones. Another approach is uniform sparse initialization which initializes every layer with the same sparsity as the total sparsity s.

Algorithm 1: Sparse Traing RNNs

Data: A RNN model M with L layers, Weight W_i, sparsity s, pruning rate p, training epoch n

```
% Sparse topology initialization
for i ← 0 to L do
    W_i ← SparseInitialization(W_i, s_i) (refer Eq. (3))
end
% Training
for epoch ← 0 to n do
    W ← NormalTraining(W)
    % Weights pruning
    for i ← 0 to L do
        W_i ←MagnitudeWeightsPrune(W_i, p_i) (refer Eq. (5) and Eq. (6))
    end
    % Weights regrowing
    for i ← 0 to L do
        W_i ←RegrowWeightsAmongLayers(W_i) (refer Eq. (7))
    end
    p ←DecayPruningRate(p)
end
```

3.1 Sparse topology initialization

A network can be denoted by:

$$y = f(x; \theta) \tag{2}$$

where $\theta \in \mathbf{R}$ is the dense parameter of the network. Instead of starting with dense parameters, our method enables the network starting with sparse parameters θ_s. In this paper, we use masks to enforce the sparse structure due to the

3.2 Pruning strategy

Different from the magnitude-based pruning used in SET which removes a fraction ζ of the smallest positive weights and of the largest negative weights of each layer after each training epoch, we choose another variant, pruning weights with the smallest absolute values. The effectiveness of such magnitude-based pruning has been shown in many works

[23, 25]. For each individual weight θ_s^i in every layer, we define its importance as its absolute values:

$$S(\theta_s^i) = |\theta_s^i| \tag{5}$$

Given a certain pruning rate p, we find the pth percentile of $S(\theta_s)$ with an ascending order as γ. Then the new mask is given by:

$$M = S(\theta_s) > \gamma \tag{6}$$

Additionally, we decay the pruning rate p iteratively to zero during training, so that the sparse topology would converge to the optimal one. This is another difference between ST-RNNs and SET.

3.3 Regrowing strategy

To keep a pure sparse structure both for the forward and the backward process, we randomly regrow new weights rather than utilizing any information of nonzero parameters. This is the main difference between ST-RNNs with gradient-based sparse training techniques such as RigL and SNFS. Gradient-based regrowing heavily depends on the gradient of every parameter and they still require a dense forward pass at least once per ΔT iterations, whereas our method keeps a clearly sparse backward pass and requires smaller FLOPs. The random regrow is given by:

$$M = M + R \tag{7}$$

R is a binary tensor in which the nonzero elements are randomly distributed. The total number of newly activated connections is the same as the number of removed connections to maintain the same sparsity level. As we keep the sparsity level of each layer strictly fixed, the FLOPs needed for training a model are proportional to their dense counterpart.

3.4 Computational complexity reduction

In the standard architecture of LSTM networks, there are an input layer, a recurrent LSTM layer and an output layer. Therefore, the learning computational complexity per time step is $O(W)$, where W is the total number of parameters, which can be calculated as: $W = 4 \times (n_i + n_h) \times n_h + n_h \times n_o = 4 \times (n_h^2 + n_i \times n_h) + n_h \times n_o$ whereas the number of ST-LSTM is $W = 4 \times \epsilon \times (n_i + n_h + n_h) + \epsilon \times (n_h + n_o)$ where n_h is the number of hidden units; n_i and n_o are the number of input and the number of output, respectively. ϵ is the hyperparameter to control the sparsity level, which usually is a small integer. Approximately, we reduce the complexity of LSTM from $O(n_h^2 + n_h \times n_k)$ to $O(\epsilon \times (n_h + n_o))$.

Our method differs from SET in several important aspects. (1) We use uniform sparse initialization instead of Erdős–Rényi. The former consistently achieves better performance with various RNN models than Erdős–Rényi. (2) Instead of pruning a fraction ζ of the smallest positive weights and of the largest negative weights, we prune the weights with the least absolute values. (3) We apply a cosine decay schedule to decay the pruning rate gradually to zero. As demonstrated in Table 2, these components help ST-LSTM outperform SET by a large margin.

4 Experimental results

To evaluate the effectiveness of our approach, we perform language modeling with stacked LSTM and recurrent highway networks (RHN) on the Penn Treebank (PTB) dataset [40, 44] and also perform text classification with one layer LSTM on six large-scale datasets.

Penn Treebank A corpus consisting of over 4.5 million words of American English, is one of the most frequently used data in language modeling. The number of unique tokens in the vocabulary is 10,000.

Text classification datasets For text classification, we evaluate ST-LSTM on AG's News [63], Yelp Review Polarity [63], IMDB [39], Sanders Corpus Twitter,[1] Yelp 2018,[2] Amazon Fine Food Reviews,[3] as shown in Table 1.

4.1 Language modeling

For the language modeling task, we choose two RNN models: stacked LSTM [61] and recurrent highway networks (RHN) [66]. For the language modeling task, we implement ST-RNNs on PyTorch. We compare ST-RNNs with strong state-of-the-art DST baselines including SET, SNFS, and RigL and a dense-to-sparse method, ISS.

4.1.1 Stacked LSTMs

The stacked LSTM is a standard RNN with two stacked LSTM layers with 1500 hidden units per layer. The size of each minibatch is 20 and the unrolled step is 35. We evaluate two sparse initialization methods, Erdős–Rényi and uniform. We train all models with momentum SGD for 100 epochs and evaluate the best model on the test set. We set the dropout rate to 0.65, the learning rate to 2 and decay it by a factor of 1.33 once the validation loss stops decreasing. We empirically find that a high clip norm of 10 brings large benefits to stacked LSTM over a small clip

[1] http://www.sananalytics.com/lab/twitter-sentiment/.

[2] https://www.yelp.com/dataset/challenge.

[3] https://snap.stanford.edu/data/web-FineFoods.html.

Table 1 Datasets for sentiment analysis experiments

Dataset	Classes	Train samples	Test samples
Sanders Corpus Twitter	4	4410	1103
IMDB	2	25,000	25,000
Yelp 2018	5	120,000	30,000
AG's news	4	120,000	7600
Amazon fine food reviews	5	454,763	113,691
Yelp review polarity	2	560,000	70,000

norm of 0.25 as used in [3, 42, 53, 59]. Our method is evaluated at two sparsity levels, 0.67 and 0.80. Results are shown in Table 2.

We can see that, with 33% parameters, all gradient-based methods (SNFS and RigL) fail to match the performance of the dense-to-sparse method (ISS), whereas random-based methods (SET, ST-LSTM) can all outperform ISS and the dense model. This observation contradicts the common belief that gradient-based weight regrowth achieves better performance than random-based regrowth [15, 18]. ST-LSTM initialized by Erdős–Rényi and uniform distribution all outperform ISS and also dense model by around 2 and 3 perplexity, respectively. Even with 80% parameters, ST-LSTM can still outperform its dense counterpart. Moreover, the uniform distribution consistently achieves slightly better performance than the Erdős–Rényi redistribution in the RNN settings.

It is interesting to evaluate if the dense LSTM trained from scratch with the same number of parameters can reach the performance of ST-LSTM or not. To make a fair comparison, the number of hidden units is set to be 700 and 490, respectively. We name this small dense LSTM as Small Dense. The result shows that directly training a small dense model cannot reach the same perplexity achieved by our method. This highlights the effectiveness of our method as neither the large dense LSTM nor the small dense LSTM can achieve the same performance as ST-LSTM.

We also report the FLOPs required to train one stacked LSTMs by different methods. As we mentioned, one advantage of our method is the training FLOPs is strictly constrained throughout the training, which allows us to choose suitable sparsity levels for different applications. Although the dense-to-sparse methods like ISS can discover sparse neural networks with smaller inference FLOPs, it starts from a highly over-parameterized dense model which is not memory efficient. Different from sparse training methods, ISS doesn't sparsify the embedding layer of stacked LSTMs, leading to a lower number of FLOPs.

4.1.2 Recurrent highway networks

Highway layers [54] are able to build very deep feedforward networks easily via a *transform* gate and a *carry* gate, which carries the original input directly to the next layer. Inspired by highway layers, Zilly et al. [66] introduced Recurrent Highway Networks (RHN) in which one or multiple Highway layers are stacked together to increase the recurrence depth. It allows each time step inside RNN layers to have deeper architecture.

We choose the same RHN variant used in ISS [57], that is, "Variational RHN + WT", which refers to a dropout regularization approximating variational Bayesian inference and weight-tying. And the carry gate is also coupled with the transform gate to reduce the number of parameters. Each time step stacks 10 Highway layers together with 830 hidden units for each. The total number of parameters is 23.5 M.

Similar to LSTM, inside the RHN layer, there are matrices W and R represent the weights matrices of the *transform* gate T and the H nonlinear transform. We first initialize the gate weights with Erdős–Rényi topology or uniform Sparsity and then apply prune-and-grow strategy to optimize the sparse topology towards an optimal one. We set the batch size to 20 and the sequence length to 35. We also use momentum SGD for ST-RHN with a learning rate of 2. Additionally, the learning rate will be decayed by 1.11 when the validation loss fails to decrease. Dropout rates are set to be 0.20 for the embedding layer, 0.65 for the input layer, 0.25 for the hidden units and 0.65 for the output layer. We train the model for 200 epochs.

The RHN experimental results are shown in Table 3. ST-RHN outperforms the dense model by about 2 test perplexity points with only 50% training FLOPs. Besides, like stacked LSTM, the small dense RHN trained from scratch fails to match the performance of ST-RHN. This stands as empirical evidence regarding the benefit of ST-RNNs. Furthermore, similar to stacked LSTM, uniform Sparsity also has slightly better performance than Erdős–Rényi for RHN.

4.2 Text classification

For text classification, we provide another implementation of ST-RNNs based on Keras with Tensorflow backend. We choose Erdős–Rényi topology to initialize sparse layers. We compare ST-LSTM to the one layer standard dense LSTM on four public datasets, Sanders Corpus Twitter, IMDB, Yelp 2018 and Amazon Fine Food Reviews. We choose the first 150,000 samples from Yelp 2018 and split them into training data and test data. To offer fair comparisons, all the hyperparameters of these two models are the same except the sparse connectivity. For the sake of

Table 2 Single model perplexity on validation and test sets in stacked LSTM on the Penn Treebank dataset

Method	Initialization	FLOPs (train)	FLOPs (test)	#Param (M)	Val	Test
Dense [61]	Dense	1 × (3.1e16)	1 × (7.2e10)	66.0	82.57	78.57
Small dense	Dense	0.330 ×	0.330 ×	21.8	84.39	81.12
SNFS	Uniform	0.614 ×	0.363 ×	21.8	84.05	80.60
RigL	Erdős–Rényi	0.333 ×	0.333 ×	21.8	82.24	79.06
ISS [57]	Dense	0.280 ×	0.200 ×	21.8	82.59	78.65
SET	Erdős–Rényi	0.333 ×	0.333 ×	21.8	80.67	77.83
ST-LSTM	Erdős–Rényi	0.333 ×	0.333 ×	21.8	79.10	76.13
ST-LSTM	Uniform	0.330 ×	0.330 ×	21.8	**78.77**	**75.84**
SNFS	Uniform	0.529 ×	0.223 ×	13.2	90.73	87.07
RigL	Erdős–Rényi	0.202 ×	0.202 ×	13.2	86.79	82.66
Small dense	Dense	0.200 ×	0.200 ×	13.2	85.22	82.00
SET	Erdős–Rényi	0.202 ×	0.202 ×	13.2	84.16	81.31
ST-LSTM	Erdős–Rényi	0.202 ×	0.202 ×	13.2	80.11	76.95
ST-LSTM	Uniform	0.200 ×	0.200 ×	13.2	**80.82**	**76.74**

The best performance shown in bold

Flops required to train a network and Flops required to inference on a single data are reported by normalizing with the FLOPs of a dense model

"Small Dense" represents a small dense LSTM trained from scratch with the same number of parameters as ST-LSTM

Table 3 Single model perplexity on validation and test sets of RHN on the Penn Treebank dataset

Method	Initialization	FLOPs Train	FLOPs Test	#Param (M)	Val	Test
Dense [66]	Dense	1 × (6.5e16)	1 × (3.3e10)	23.5	67.9	65.4
Small dense	Dense	0.472 ×	0.472 ×	11.1	70.1	68.4
ISS [57]	Dense	0.503 ×	0.472 ×	11.1	68.1	65.4
ST-RHN	Erdős–Rényi	0.474 ×	0.474 ×	11.1	66.51	63.55
ST-RHN	Uniform	0.472 ×	0.472 ×	11.1	**66.08**	**63.19**

The best performance shown in bold

Flops required to train a network and Flops required to inference on a single data are reported by normalizing with the FLOPs of a dense model

"Small Dense" represents a small dense RHN trained from scratch with the same number of parameters as ST-LSTM

convenience, on all datasets, we set the sparsity hyperparameter to be $\epsilon = 10$, with a sparsity level of 95.69%. The dimension of word embedding and the hidden state of LSTM unit are both 256. The number of words in each sentence is 100, and the total number of words in the embedding vector is 20,000. The rewire rate $\zeta = 0.2$ for Yelp 2018 and Amazon, $\zeta = 0.6$ for Twitter and $\zeta = 0.4$ for IMDB; Additionally, the mini-batch size is 64 for Twitter, Yelp and Amazon, and 256 for IMDB. We train the models using Adam optimizer with the learning rate of 0.001 for Twitter and Amazon, 0.01 for Yelp 2018, and 0.0005 for IMDB.

Table 4 is the test accuracy for the four datasets on which ST-LSTM is compared with dense LSTM. We can see that ST-LSTM can outperform dense LSTM on three datasets, while decreases the number of parameters from 5.6M to 0.2M. The only dataset that ST-LSTM does not increase the accuracy is Amazon, with a 1.36% loss of accuracy. We mention here that the accuracy on Amazon can be improved by searching for the best hyperparameters, but it is out of the goal of this paper.

Moreover, to sanity check the effectiveness of ST-RNNs and to provide a comparison with existing traditional classifiers, we choose two datasets AG's News and Yelp Review Polarity from published literature [28, 63]. For these two experiments, the sparsity level ϵ is 20 with corresponding sparsity level 92.2%. The optimizer we use is Adam with learning rates 0.01 and 0.001 for AG's News and Yelp Review Polarity, respectively. As shown in Table 5, our method is still competitive among the state-of-the-art methods on AG's News and Yelp Review Polarity. Note that outperforming every other model reported in the

Table 4 Test accuracy (%) on sentiment classification compared with dense LSTM

Methods	Twitter	IMDB	Yelp	Amazon
Dense	77.8	85.3	63.4	**81.9**
ST-LSTM	**79.2** ($\pm$0.03)	**86.0** ($\pm$0.02)	**68.0** ($\pm$0.03)	80.5 ($\pm$0.01)

The sparsity level is 95.69% for all datasets

The number of parameters is 0.2M for ST-LSTM, while the dense LSTM contains 5.6M parameters

Each accuracy is collected and averaged from five different ST-LSTM runnings

We mark the best performance in bold

Table 5 Test accuracy (%) on AG's news and Yelp review polarity

Methods	AG	Yelp P.
BoW [63]	88.8	92.2
ngrams [63]	92.0	95.6
char-CNN [62]	87.2	94.7
char-CRNN [58]	91.4	94.5
VDCNN [12]	91.3	**95.7**
fastText, bigram [28]	**92.5**	**95.7**
Dense-LSTM	88.4	94.8
ST-LSTM	90.9 ($\pm$0.20)	95.3 ($\pm$0.13)

The sparsity level is 92.2%

Each accuracy is collected and averaged from five different ST-LSTM trials, as the topology and weights are initialized randomly

We mark the best performance in bold

literature on these datasets is outside the scope of this paper, as our aim is to demonstrate the ability of sparse recurrent neural networks to reach (and most of the times to outperform) the performance of their dense counterparts.

Besides this, we are also interested in if ST-LSTM is still trainable under the extreme sparsity. To do this, we set the sparsity to an extreme level (99.1%), and we compare our algorithm with the dense LSTM. We test our approach on five datasets. The results are shown in Table 6. With only 0.04M parameters, our method is still able to find a good sparse topology with competitive performance.

5 Analysis

5.1 Sensitivity analysis

To understand better how the two hyperparameters, pruning rate and sparsity, influence the performance of ST-RNNs, we make extra analysis on Sanders Corpus Twitter dataset and Penn Treebank dataset.

Table 6 Sentiment analysis test accuracy (%) of ST-LSTM under extreme sparsity (99.1%) on IMDB, Twitter, Yelp 2018, AG's news and Yelp. P

Methods	IMDB	Twitter	Yelp	AG's news	Yelp. P
Dense	85.26	77.79	63.36	87.50	94.80
ST-LSTM	85.05	78.85	67.82	90.00	94.22

The number of parameters of ST-LSTM is 0.04M and the dense LSTM counterpart is 5.6M

5.1.1 Pruning rate

As a hyperparameter of ST-RNNs, the pruning rate determines how many connections should be removed after each epoch. We conduct experiments with various pruning rates from 0.1 to 0.9 to study the influence of different pruning rates. The test accuracy is reported in Fig. 2. We can see that the ST-RNNs have better performance with relatively higher pruning rates between 0.5 and 0.9 than the lower options. This is held for both two datasets. Note that, while varying the pruning rate results in the different performance of ST-RNNs, the difference is very small when the pruning rate locates in a proper interval, especially for Twitter.

5.1.2 Sparsity

For sparse training networks, there is a trade-off between the sparsity level and the performance of sparse neural networks. If the network is too sparse, it will not have sufficient capacity to learn the dataset, whereas if the network is too dense, the benefits provided by sparse neural networks will be too subtle. In order to analyze the sensitivity between model performance and sparsity, we perform an experiment with different sparsity levels. The results are reported in Fig. 3. We can see that the one-layer ST-LSTM can outperform the dense counterpart with sparsity in an interval between 52.5% to 99.1%. It is worth noting that, for Twitter with extreme sparsity, when $\epsilon = 2$ (sparsity is equal to 99.1%), the accuracy of ST-LSTM

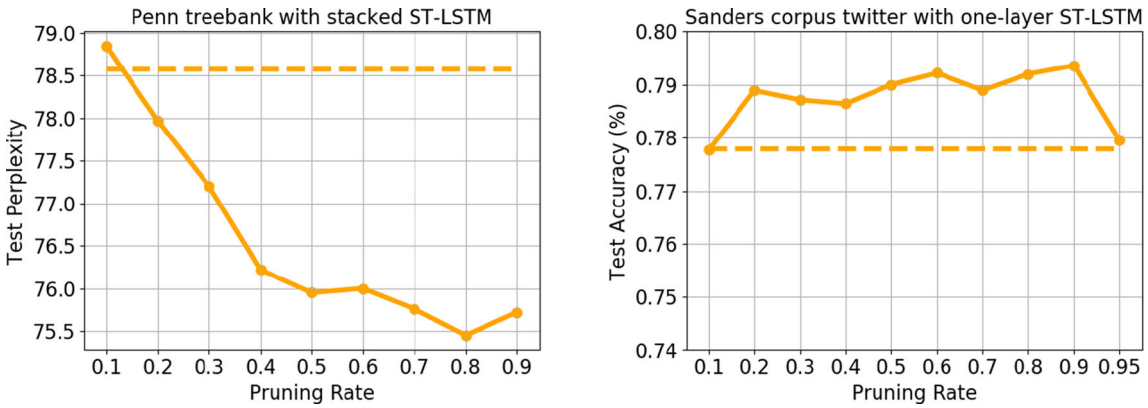

Fig. 2 Performance of ST-RNNs with various pruning rates on Sanders Corpus Twitter and Penn Treebank datasets. The performance of the dense model is indicated with a dashed orange line (colour figure online)

(78.75%) is still higher than the accuracy of Dense-LSTM (77.19%). Moreover, it is interesting to see that when the sparsity level goes down under 90% the accuracy is also going down, this being in line with our observation that usually sparse networks with adaptive sparse connectivity perform better than fully connected networks. For Penn Treebank, as long as the sparsity is below 80%, ST-LSTM can reach the dense performance.

5.2 Analyze the effect of cross-layer parameter reallocation

Cross-layer parameter reallocation has shown impressive improvement in convolutional neural networks [15, 49]. To verify the effect of parameter reallocation on RNNs, we compare the perplexity between stacked ST-LSTM and ST-RHN on Penn Treebank with and without parameter reallocation, respectively. The parameter reallocation method we choose is Sparse Momentum proposed in [15], which redistributes weights according to the mean of momentum magnitude m_i of all nonzero weights in each layer i. More specifically, first, the momentum contribution of layer i is obtained by normalizing m_i through all layers $\sum_{i=0}^{k} m_i$ after each epoch. Then, the number of parameters to be regrown in each layer is the multiplication of the total number of regrow weights with the momentum contribution of each layer.

The results can be seen in Table 7. Parameter reallocation among layers results in performance improvement for ST-LSTM steadily for both initialization, whereas the performance degrades dramatically if cross-layer parameter reallocation is used for RHN. These results suggest parameter reallocation is not pivotal for RNNs. At least, proper reallocation strategies are needed to design for sparse RNNs.

6 Conclusions and future work

In this paper, we propose ST-RNNs to train intrinsically sparse recurrent neural networks efficiently under the situation where the training FLOP budget is strictly limited.

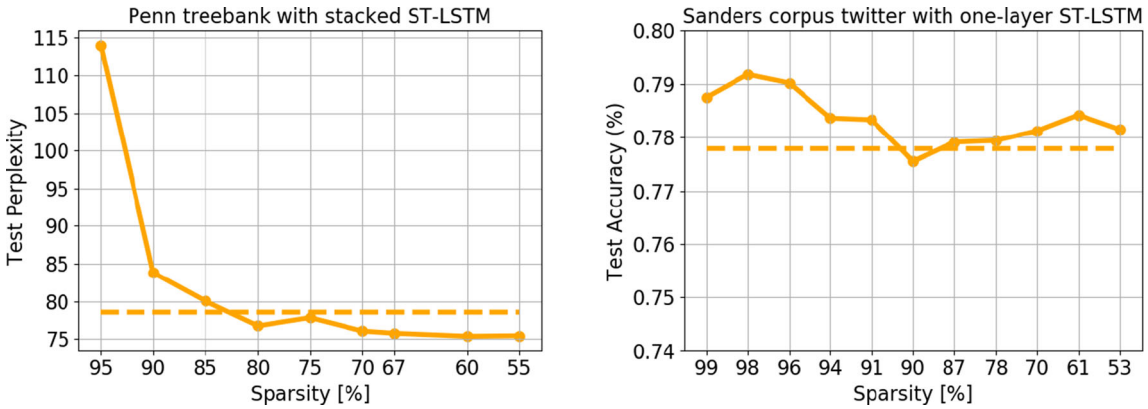

Fig. 3 Performance of ST-RNNs with various sparsity levels on Sanders Corpus Twitter and Penn Treebank. The performance of the standard dense model is indicated with a dashed orange line (colour figure online)

Table 7 Single model perplexity on validation and test sets of ST-LSTM and ST-RHN on PTB with and without cross-layer parameter reallocation

Initialization	Redistribution	Perplexity (valid, test)	
		ST-LSTM	ST-RHN
Erdős–Rényi	Sparse momentum	(78.15, 75.53)	(71.94, 69.03)
Erdős–Rényi	None	(79.44, 76.06)	(66.51, 63.55)
Uniform	Sparse momentum	(78.08, 74.36)	(73.05, 70.22)
Uniform	None	(78.83, 75.76)	(65.71, 63.04)

The advantages of our method are threefold. (1) Training efficiency: ST-RNN trains a sparse RNN from scratch and maintains a fixed number of training FLOPs throughout training, getting rid of training a dense model in the beginning; (2) inference efficiently: our method yields sparse RNNs that can achieve better performance than their dense counterpart with much fewer FLOPs; (3) state-of-the-art sparse RNN performance: with the same number of parameters, ST-RNNs achieves state-of-the-art performance with LSTMs and RHN on language modeling and text classification.

One possible direction to improve the performance of sparse training is to apply the advanced activation function e.g., SPOCU [31] and SELU [32], to increase the gradient flow. As noted by [2, 19, 34], sparse neural networks suffer from a poor gradient flow which limits the learning capability of the sparse networks. Compared with ReLU which has zero derivatives for all negative inputs, activation functions such as SPOCU and SELU might help to increase the gradient flow of sparse training. Additionally, the ability of SPOCU to alleviate the vanishing gradient problem can also improve the performance of sparse LSTMs.

Besides, the results demonstrated in this paper are achieved by masked weights, since GPU-accelerated libraries have limited support for sparse operations. Our paper provides motivation for new types of hardware accelerators and libraries with better support for sparse neural networks. In future work, we intend to develop more efficient methods to achieve sparse neural networks that can be optimized by contemporary hardware. We are also interested to modify modern libraries so that the advantages of sparse networks can be made full use of. Furthermore, the explanation underlying the observation that sparse networks with adaptive sparse connectivity can generalize better than their dense counterpart has not been fully studied yet. Our work points out important directions for future research.

Compliance with ethical standards

Conflict of interest All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

References

1. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M et al (2016) Tensorflow: a system for large-scale machine learning. In: 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16), pp 265–283
2. Tessera k, Hooker S, Rosman B (2021) Keep the gradients flowing: using gradient flow to study sparse network optimization. https://openreview.net/forum?id=HI0j7omXTaG
3. Liu S, Mocanu DC, Pei Y, Pechenizkiy M (2021) Selfish sparse RNN training. In: Submitted to international conference on learning representations. https://openreview.net/forum?id=5wmNjjvGOXh
4. Antol S, Agrawal A, Lu J, Mitchell M, Batra D, Lawrence ZC, Parikh D (2015) VQA: visual question answering. In: Proceedings of the IEEE international conference on computer vision, pp 2425–2433
5. Aquino G, Rubio JDJ, Pacheco J, Gutierrez GJ, Ochoa G, Balcazar R, Cruz DR, Garcia E, Novoa JF, Zacarias A (2020) Novel nonlinear hypothesis for the delta parallel robot modeling. IEEE Access 8:46324–46334
6. Baddar WJ, Ro YM (2020) Encoding features robust to unseen modes of variation with attentive long short-term memory. Pattern Recognit 100:107159
7. Bellec G, Kappel D, Maass W, Legenstein R (2018) Deep rewiring: training very sparse deep networks. In: International conference on learning representations. https://openreview.net/forum?id=BJ_wN01C-
8. Bhunia AK, Konwer A, Bhunia AK, Bhowmick A, Roy PP, Pal U (2019) Script identification in natural scene image and video frames using an attention based convolutional-LSTM network. Pattern Recognit 85:172–184
9. Bhushan SB, Danti A (2017) Classification of text documents based on score level fusion approach. Pattern Recognit Lett 94:118–126
10. Chebotar Y, Waters A (2016) Distilling knowledge from ensembles of neural networks for speech recognition. In: Interspeech, pp 3439–3443
11. Chiang HS, Chen MY, Huang YJ (2019) Wavelet-based EEG processing for epilepsy detection using fuzzy entropy and associative petri net. IEEE Access 7:103255–103262

12. Conneau A, Schwenk H, Barrault L, Lecun Y (2017) Very deep convolutional networks for text classification. In: Proceedings of the 15th conference of the European chapter of the association for computational linguistics: volume 1, long papers. Association for Computational Linguistics, Valencia, Spain, pp 1107–1116. https://www.aclweb.org/anthology/E17-1104
13. de Jesús Rubio J (2009) Sofmls: online self-organizing fuzzy modified least-squares network. IEEE Trans Fuzzy Syst 17(6):1296–1309
14. de Rubio JJ (2020) Stability analysis of the modified Levenberg–Marquardt algorithm for the artificial neural network training. IEEE Trans Neural Netw Learn Syst
15. Dettmers T, Zettlemoyer L (2019) Sparse networks from scratch: faster training without losing performance. arXiv preprint arXiv:1907.04840
16. Donahue J, Anne HL, Guadarrama S, Rohrbach M, Venugopalan S, Saenko K, Darrell T (2015) Long-term recurrent convolutional networks for visual recognition and description. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2625–2634
17. Egmont-Petersen M, de Ridder D, Handels H (2002) Image processing with neural networks—a review. Pattern Recognit 35(10):2279–2301
18. Evci U, Gale T, Menick J, Castro, PS, Elsen, E (2019) Rigging the lottery: making all tickets winners. arXiv preprint arXiv:1911.11134
19. Evci U, Ioannou YA, Keskin C, Dauphin Y (2020) Gradient flow in sparse neural networks and how lottery tickets win. arXiv preprint arXiv:2010.03533
20. Feng G, Guo J, Jing BY, Sun T (2015) Feature subset selection using Naive Bayes for text classification. Pattern Recognit Lett 65:109–115
21. Frankle J, Carbin M (2019) The lottery ticket hypothesis: finding sparse, trainable neural networks. In: International conference on learning representations. https://openreview.net/forum?id=rJl-b3RcF7
22. Giles CL, Omlin CW (1994) Pruning recurrent neural networks for improved generalization performance. IEEE Trans Neural Netw 5(5):848–851
23. Guo Y, Yao A, Chen Y (2016) Dynamic network surgery for efficient DNNs. In: Advances in neural information processing systems, pp 1379–1387
24. Han S, Kang J, Mao H, Hu Y, Li X, Li Y, Xie D, Luo H, Yao S, Wang Y et al (2017) ESE: efficient speech recognition engine with sparse LSTM on FPGA. In: Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays. ACM, pp 75–84
25. Han S, Pool J, Tran J, Dally W (2015) Learning both weights and connections for efficient neural network. In: Advances in neural information processing systems, pp 1135–1143
26. Hernández G, Zamora E, Sossa H, Téllez G, Furlán F (2020) Hybrid neural networks for big data classification. Neurocomputing 390:327–340
27. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
28. Joulin A, Grave E, Bojanowski P, Mikolov T (2017) Bag of tricks for efficient text classification. In: Proceedings of the 15th conference of the European chapter of the association for computational linguistics: volume 2, short papers. Association for Computational Linguistics, Valencia, Spain, pp 427–431. https://www.aclweb.org/anthology/E17-2068
29. Jouppi NP, Young C, Patil N, Patterson D, Agrawal G, Bajwa R, Bates S, Bhatia S, Boden N, Borchers A et al (2017) In-datacenter performance analysis of a tensor processing unit. In: 2017 ACM/IEEE 44th annual international symposium on computer architecture (ISCA). IEEE, pp 1–12
30. Juan A, Vidal E (2002) On the use of Bernoulli mixture models for text classification. Pattern Recognit 35(12):2705–2710
31. Kisel'ák J, Lu Y, Švihra J, Szépe P, Stehlík M (2020) "SPOCU": scaled polynomial constant unit activation function. Neural Comput Appl 1–17
32. Klambauer G, Unterthiner T, Mayr A, Hochreiter S (2017) Self-normalizing neural networks. Adv Neural Inf Process Syst 30:971–980
33. LeCun Y, Denker JS, Solla, SA (1990) Optimal brain damage. In: Advances in neural information processing systems, pp 598–605
34. Lee N, Ajanthan T, Gould S, Torr PH (2019) A signal propagation perspective for pruning neural networks at initialization. arXiv preprint https://openreview.net/forum?id=HI0j7omXTaG0
35. Liu S, van der Lee T, Yaman A, Atashgahi Z, Ferrar D, Sokar G, Pechenizkiy M, Mocanu D (2020) Topological insights into sparse neural networks. In: Joint European conference on machine learning and knowledge discovery in databases
36. Liu S, Mocanu DC, Matavalam ARR, Pei Y, Pechenizkiy M (2020) Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. Neural Comput Appl 1–16
37. Louizos C, Welling M, Kingma, DP (2018) Learning sparse neural networks through l_0 regularization. In: International conference on learning representations. https://openreview.net/forum?id=H1Y8hhg0b
38. Lu G, Zhao X, Yin J, Yang W, Li B (2018) Multi-task learning using variational auto-encoder for sentiment classification. Pattern Recognit Lett
39. Maas AL, Daly RE, Pham PT, Huang D, Ng AY, Potts C (2011) Learning word vectors for sentiment analysis. In: Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies, vol 1. Association for Computational Linguistics, pp 142–150
40. Marcus MP, Santorini B, Marcinkiewicz MA (1993) Building a large annotated corpus of English: the Penn Treebank. Comput Linguist 19(2):313–330
41. Meda-Campaña JA (2018) On the estimation and control of nonlinear systems with parametric uncertainties and noisy outputs. IEEE Access 6:31968–31973
42. Merity S, Keskar NS, Socher, R (2017) Regularizing and optimizing LSTM language models. arXiv preprint arXiv:1708.02182
43. Michael H, Zhu SG (2018) To prune, or not to prune: exploring the efficacy of pruning for model compression. In: International conference on learning representations. https://openreview.net/forum?id=S1lN69AT-
44. Mikolov T, Karafiát M, Burget L, Černockỳ J, Khudanpur S (2010) Recurrent neural network based language model. In: Eleventh annual conference of the international speech communication association
45. Mocanu DC, Ammar HB, Puig L, Eaton E, Liotta A (2017) Estimating 3D trajectories from 2D projections via disjunctive factored four-way conditional restricted Boltzmann machines. Pattern Recognit 69:325–335
46. Mocanu DC, Mocanu E, Nguyen PH, Gibescu M, Liotta A (2016) A topological insight into restricted Boltzmann machines. Mach Learn 104(2):243–270
47. Mocanu DC, Mocanu E, Stone P, Nguyen PH, Gibescu M, Liotta A (2018) Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. Nat Commun 9(1):2383
48. Molchanov D, Ashukha A, Vetrov D (2017) Variational dropout sparsifies deep neural networks. In: Proceedings of the 34th international conference on machine learning, vol 70. JMLR.org, pp 2498–2507

49. Mostafa H, Wang X (2019) Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In: Proceedings of the 36th international conference on machine learning, vol 97. JMLR.org, pp 4646–4655
50. Narang S, Elsen E, Diamos G, Sengupta S (2017) Exploring sparsity in recurrent neural networks. In: International conference on learning representations. https://openreview.net/forum?id=BylSPv9gx
51. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L et al (2019) Pytorch: an imperative style, high-performance deep learning library. In: Advances in neural information processing systems, pp 8026–8037
52. Ren H, Wang W, Liu C (2019) Recognizing online handwritten Chinese characters using RNNs with new computing architectures. Pattern Recognit 93:179–192
53. Shen Y, Tan S, Sordoni A, Courville A (2018) Ordered neurons: integrating tree structures into recurrent neural networks. arXiv preprint arXiv:1810.09536
54. Srivastav, RK, Greff K, Schmidhuber, J (2015) Highway networks. arXiv preprint https://openreview.net/forum?id=HI0j7omXTaG7
55. Su B, Lu S (2017) Accurate recognition of words in scenes without character segmentation using recurrent neural network. Pattern Recognit 63:397–405
56. Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In: Advances in neural information processing systems, pp 3104–3112
57. Wen W, He Y, Rajbhandari S, Zhang M, Wang W, Liu F, Hu B, Chen Y, Li H (2018) Learning intrinsic sparse structures within long short-term memory. In: International conference on learning representations. https://openreview.net/forum?id=rk6cfpRjZ
58. Xiao Y, Cho K (2016) Efficient character-level document classification by combining convolution and recurrent layers. arXiv preprint https://openreview.net/forum?id=HI0j7omXTaG9
59. Yang Z, Dai Z, Salakhutdinov R, Cohen WW (2017) Breaking the softmax bottleneck: a high-rank RNN language model. arXiv preprint arXiv:1711.03953
60. Yousfi S, Berrani SA, Garcia C (2017) Contribution of recurrent connectionist language models in improving LSTM-based Arabic text recognition in videos. Pattern Recognit 64:245–254
61. Zaremba W, Sutskever I, Vinyals O (2014) Recurrent neural network regularization. arXiv preprint https://openreview.net/forum?id=5wmNjjvGOXh1
62. Zhang X, LeCun Y (2015) Text understanding from scratch. arXiv preprint https://openreview.net/forum?id=5wmNjjvGOXh2
63. Zhang X, Zhao J, LeCun Y (2015) Character-level convolutional networks for text classification. In: Advances in neural information processing systems, pp 649–657
64. Zhang Y, Chen G, Yu D, Yaco K, Khudanpur S, Glass J (2016) Highway long short-term memory RNNs for distant speech recognition. In: 2016 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, pp 5755–5759
65. Zhou H, Lan J, Liu R, Yosinski J (2019) Deconstructing lottery tickets: zeros, signs, and the supermask. In: Advances in neural information processing systems, pp 3592–3602
66. Zilly JG, Srivastava RK, Koutník J, Schmidhuber, J (2017) Recurrent highway networks. In: Proceedings of the 34th international conference on machine learning, vol 70. JMLR.org, pp 4189–4198