LEARNING INTRINSIC SPARSE STRUCTURES WITHIN LONG SHORT-TERM MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Model compression is significant for the wide adoption of *Recurrent Neural Networks* (RNNs) in both user devices possessing limited resources and business clusters requiring quick responses to large-scale service requests. This work aims to learn structurally-sparse *Long Short-Term Memory* (LSTM) by reducing the sizes of *basic structures* within LSTM units, including input updates, gates, hidden states, cell states and outputs. Independently reducing the sizes of basic structures can result in inconsistent dimensions among them, and consequently, end up with invalid LSTM units. To overcome the problem, we propose *Intrinsic Sparse Structures* (ISS) in LSTMs. Removing a component of ISS will simultaneously decrease the sizes of all basic structures by one and thereby always maintain the dimension consistency. By learning ISS within LSTM units, the obtained LSTMs remain regular while having much smaller basic structures. Based on group Lasso regularization, our method achieves $10.59\times$ speedup without losing any perplexity of a language modeling of Penn TreeBank dataset. It is also successfully evaluated through a compact model with only 2.69M weights for machine Question Answering of SQuAD dataset.

1 INTRODUCTION

Model Compression (Jaderberg et al. (2014), Han et al. (2015a), Wen et al. (2017), Louizos et al. (2017)) is a class of approaches of reducing the size of *Deep Neural Networks* (DNNs) to accelerate inference. *Structure Learning* (Zoph & Le (2017), Philipp & Carbonell (2017), Cortes et al. (2017)) emerges as an active research area for DNN structure exploration, potentially replacing human labor with machine automation for design space exploration. In the intersection of both techniques, an important area is to learn compact structures in DNNs for efficient inference computation using minimal memory and execution time without losing accuracy. Learning compact structures in *Convolutional Neural Networks* (CNNs) have been widely explored in the past few years. Han et al. (2015b) proposed connection pruning for sparse CNNs. Pruning method also works successfully in coarse-grain levels, such as pruning filters in CNNs (Li et al. (2017)) and reducing neuron numbers (Alvarez & Salzmann (2016)). Wen et al. (2016) presented a general framework to learn versatile compact structures (neurons, filters, filter shapes, channels and even layers) in DNNs.

Learning the compact structures in *Recurrent Neural Networks* (RNNs) is more challenging. As a recurrent unit is shared across all the time steps in sequence, compressing the unit will aggressively affect all the steps. A recent work by Narang et al. (2017) proposes a pruning approach that deletes up to 90% connections in RNNs. Connection pruning methods sparsify weights of recurrent units but cannot explicitly change basic structures, *e.g.*, the number of input updates, gates, hidden states, cell states and outputs. Moreover, the obtained sparse matrices have an irregular/non-structured pattern of non-zero weights, which is unfriendly for efficient computation in modern hardware systems (Lebedev & Lempitsky (2016)). Previous study (Wen et al. (2016)) on sparse matrix multiplication in GPUs showed that the speedup[1] was either counterproductive or ignorable. More specific, with sparsity[2] of 67.6%, 92.4%, 97.2%, 96.6% and 94.3% in weight matrices of *AlexNet*, the speedup was $0.25\times$, $0.52\times$, $1.38\times$, $1.04\times$, and $1.36\times$, respectively. This problem also exists in CPUs. Fig. 1 shows that non-structured pattern in sparsity limits the speedup. We only

[1] Defined as (new speed)/(original speed).

[2] Defined as the percentage of zeros.

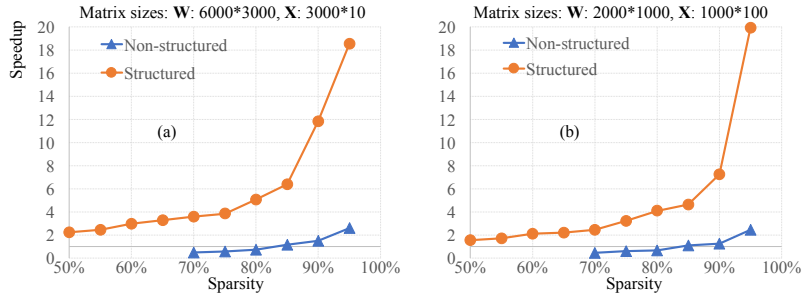

Figure 1: Speedups of matrix multiplication using non-structured and structured sparsity. Speeds are measured in Intel MKL implementations in Intel Xeon CPU E5-2630 v3 @ 2.40GHz. General matrix-matrix multiplication (GEMM) of $\mathbf{W} \cdot \mathbf{X}$ is implemented by `cblas_sgemm`. The matrix sizes are selected to reflect commonly used GEMMs in LSTMs. For example, (a) represents GEMM in LSTMs with hidden size 1500, input size 1500 and batch size 10. To accelerate GEMM by sparsity, $\mathbf{W}$ is sparsified. In non-structured sparsity approach, $\mathbf{W}$ is randomly sparsified and encoded as Compressed Sparse Row format for sparse computation (using `mkl_scsrmm`); in structured sparsity approach, $2k$ columns and $4k$ rows in $\mathbf{W}$ are removed to match the same level of sparsity (*i.e.,* the percentage of removed parameters) for faster GEMM under smaller sizes.

starts to observe speed gain when the sparsity is beyond 80% and the speedup is only around $2.5\times$ even when the sparsity is 95%. In this work, we focus on learning structurally sparse LSTMs for computation efficiency. More specific, we aim to reduce the number of input updates, gates, hidden states, cell states and outputs simultaneously during learning, such that the obtained LSTMs have the original schematic with dense connections but smaller sizes of these basic structures. Such compact models have structured sparsity, with columns and rows in weight matrices removed, whose computation efficiency is shown in Fig. 1. Moreover, off-the-shelf libraries in deep learning frameworks can be directly utilized to deploy the reduced LSTMs. Details should be explained.

There is a vital challenge originated from recurrent units: as the structures of inputs, gates, states and outputs interweave with each other, independently removing these structures can result in mismatch of their dimensions and then inducing invalid recurrent units. The problem does not exist in CNNs, where neurons (or filters) can be independently removed without violating the usability of the final network structure. we propose *Intrinsic Sparse Structures* (ISS) to overcome the unique challenge in LSTMs. By removing weights associated with one component of ISS, the sizes/dimensions (of input updates, gates, hidden states and cell states in current layer, and outputs to the next stacked layers) are *simultaneously* reduced by one. Therefore, learning ISS can directly reduce the size of structures in LSTMs while maintaining the dimension consistency.

We evaluated our method by LSTMs in language modeling of Penn Treebank dataset (Marcus et al. (1993)) and machine Question Answering of SQuAD dataset (Rajpurkar et al. (2016)). Our approach works both in fine-tuning and in training from scratch. In a RNN with two stacked LSTM layers with hidden sizes of 1500 (*i.e.*, 1500 components of ISS) for language modeling (Zaremba et al. (2014)), our method learns that the sizes of 373 and 315 in the first and second LSTMs, respectively, are sufficient for the same perplexity. It achieves $10.59\times$ speedup of inference time. The result is obtained by training from scratch with the same number of epochs. Directly training LSTMs with sizes of 373 and 315 cannot achieve the same perplexity, which proves the advantage of learning ISS for model compression. Promising results are also obtained in a compact model – the state-of-the-art BiDAF model (Seo et al. (2017)) for Question Answering in SQuAD dataset. BiDAF has only 2.69M weights. Our source code is publicly available (*URL is removed for double blind review*).

2 Related Work

A major approach in DNN compression is to reduce the complexity of structures within DNNs. The studies can be categorized to three classes: removing redundant structures in original DNNs, approximating the original function of DNNs (Denil et al. (2013), Jaderberg et al. (2014), Hinton et al. (2015), Lu et al. (2016), Prabhavalkar et al. (2016), Molchanov et al. (2017)), and designing

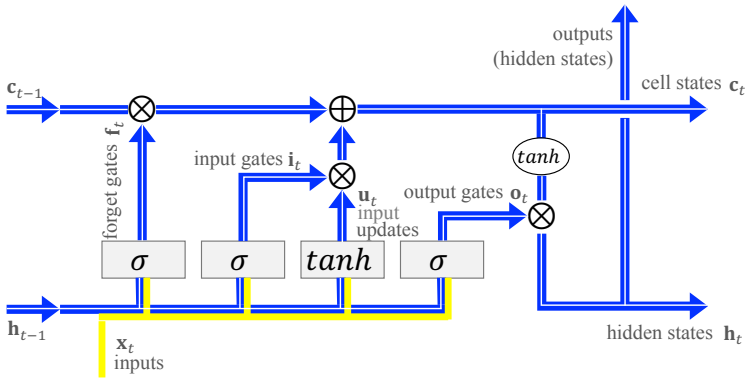

Figure 2: Intrinsic Sparse Structures (ISS) in LSTM units.

DNNs with inherently compact structures (Szegedy et al. (2015), He et al. (2016), Wu et al. (2017), Bradbury et al. (2016)). Our method belongs to the first category.

Research on removing redundant structures in *Feed-forward Neural Networks* (FNNs), typically in CNNs, has been extensively studied. Based on ℓ_1 regularization (Liu et al. (2015), Park et al. (2017)) or connection pruning (Han et al. (2015b), Guo et al. (2016)), the number of connections/parameters can be dramatically reduced. Group Lasso based methods were proved to be effective in reducing coarse-grain structures (*e.g.,* neurons, filters, channels, filter shapes, and even layers) in CNNs (Wen et al. (2016), Alvarez & Salzmann (2016), Lebedev & Lempitsky (2016), Yoon & Hwang (2017)). For instance, Wen et al. (2016) reduced the number of layers from 32 to 18 in *ResNet* without any accuracy loss for CIFAR-10 dataset. A recent work by Narang et al. (2017) advances connection pruning techniques in CNNs and extends it to RNNs. It compresses the size of *Deep Speech 2* (Amodei et al. (2016)) from 268 MB to around 32 MB. However, to the best of our knowledge, little work has been carried out to reduce coarse-grain structures beyond fine-grain connections in RNNs, possibly because of the sophisticated structures of RNNs, especially in LSTMs. To fill this gap, our work targets to develop a method that can learn to reduce the number of input updates, gates, hidden states, cell states and outputs within LSTM units. After learning those structures, final LSTMs are still regular LSTMs with the same connectivity, but have the sizes reduced.

Another line of related research is *Structure Learning* of FNNs or CNNs. Zoph & Le (2017) uses reinforcement learning to search good neural architectures. Philipp & Carbonell (2017) dynamically adds and eliminates neurons in FNNs by using group Lasso regularization. Cortes et al. (2017) gradually adds sub-networks to current networks to incrementally reduce the objective function. All these works focused on finding optimal structures in FNNs or CNNs for classification accuracy. In contrast, this work aims at learning compact structures in LSTMs for model compression.

3 Learning Intrinsic Sparse Structures

3.1 Intrinsic Sparse Structures

The computation within LSTMs is (Hochreiter & Schmidhuber (1997))

$$\begin{aligned}
\mathbf{i}_t &= \sigma\left(\mathbf{x}_t \cdot \mathbf{W}_{xi} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{hi} + \mathbf{b}_i\right) \\
\mathbf{f}_t &= \sigma\left(\mathbf{x}_t \cdot \mathbf{W}_{xf} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{hf} + \mathbf{b}_f\right) \\
\mathbf{o}_t &= \sigma\left(\mathbf{x}_t \cdot \mathbf{W}_{xo} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{ho} + \mathbf{b}_o\right) \\
\mathbf{u}_t &= tanh\left(\mathbf{x}_t \cdot \mathbf{W}_{xu} + \mathbf{h}_{t-1} \cdot \mathbf{W}_{hu} + \mathbf{b}_u\right), \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{u}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot tanh\left(\mathbf{c}_t\right)
\end{aligned} \tag{1}$$

where $\odot$ is element-wise multiplication, $\sigma(\cdot)$ is sigmoid function, and $tanh(\cdot)$ is hyperbolic tangent function. Vectors are row vectors. $\mathbf{W}$s are weight matrices, which transform the concatenation (of hidden states $\mathbf{h}_{t-1}$ and inputs $\mathbf{x}_t$) to input updates $\mathbf{u}_t$ and gates ($\mathbf{i}_t$, $\mathbf{f}_t$ and $\mathbf{o}_t$). Fig. 2 is the schematic

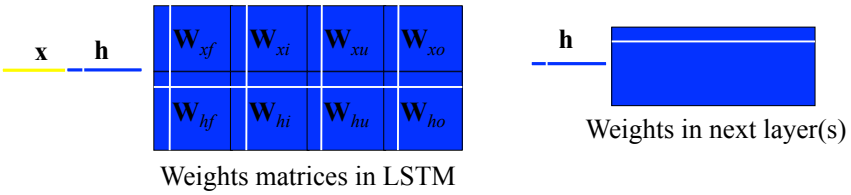

Figure 3: Applying Intrinsic Sparse Structures in weight matrices.

of LSTMs in the layout of Olah (2015). The transformations by $\mathbf{W}$s and the corresponding nonlinear functions are illustrated in rectangle blocks. Our goal is to reduce the size of this sophisticated structure within LSTMs, meanwhile maintaining the original schematic.

Because of element-wise operators ("$\oplus$" and "$\otimes$"), all vectors along the blue band in Fig. 2 must have the same dimension. We call this constraint as "*dimension consistency*". The vectors required to obey the dimension consistency include input updates, all gates, hidden states, cell states, and outputs. Note that hidden states are usually outputs connected to classifier layer or stacked LSTM layers. Due to the dimension consistency, it is difficult to reduce the structures in LSTMs: as can be seen in Fig. 2, vectors (along the blue band) interweave with each other so removing an individual component from one or a few vectors *independently* can result in the violation of dimension consistency. To overcome the difficulty, we propose *Intrinsic Sparse Structures* (ISS) within LSTMs as shown by the blue band in Fig. 2. One component of ISS is highlighted as the white strip. By decreasing the size of ISS (*i.e.*, the width of the blue band), we are able to simultaneously reduce the dimensions of input updates, gates, hidden states, cell states and outputs.

To learn sparse ISS, we turn to weight sparsifying. There are totally eight weight matrices in Eq. (1). We organize them in the form of Fig. 3 as basic LSTM cells in TensorFlow. We can remove one component of ISS by zeroing out all associated weights in the white rows and white columns in Fig. 3. Why? Suppose the k-th hidden state of $\mathbf{h}$ is removable, then the k-th row in the lower four weight matrices can be all zeros (as shown by the left white horizontal line in Fig. 3), because those weights are on connections receiving the k-th *useless* hidden state. Likewise, all connections receiving the k-th hidden state in next layer(s) can be removed as shown by the right white horizontal line. Note that next layer(s) can be an output layer, LSTM layers, fully-connected layers, or a mix of them. ISS overlay two or more layers, without explicit explanation, we refer to the first LSTM layer as the ownership of ISS. Furthermore, when the k-th hidden state turns useless, the k-th output gate and k-th cell state generating this hidden state are removable. As the k-th output gate is generated by the k-th column in $\mathbf{W}_{xo}$ and $\mathbf{W}_{ho}$, these weights can be zeroed out (as shown by the fourth vertical white line in Fig. 3). Tracing back against the computation flow in Fig. 2, we can reach similar conclusions for forget gates, input gates and input updates, as respectively shown by the first, second and third vertical line in Fig. 3. For convenience, we call the weights in white rows and columns as an "*ISS weight group*". Although we focus on learning ISS in LSTMs, variants of ISS for vanilla RNNs, *Gated Recurrent Unit* (GRU) (Cho et al. (2014)), and *Recurrent Highway Networks* (RHNs) (Zilly et al. (2017)) can also be realized based on the same philosophy.

For even a medium-scale LSTM, the number of weights in one ISS weight group can be very large. It seems to be very aggressive to simultaneously slaughter so many weights to maintain the original recognition performance. However, the proposed ISS intrinsically exists within LSTMs and can even be unveiled by *independently* sparsifying each weight using ℓ_1-norm regularization. We take the large stacked LSTMs by Zaremba et al. (2014) for language modeling as the example. The network has two stacked LSTM layers whose dimensions of inputs and states are both 1500, and an output layer with a vocabulary of 10000 words. The sizes of "ISS weight groups" of two LSTM layers are as large as 24000 and 28000. The perplexities of validation set and test set are respectively 82.57 and 78.57. We fine-tune this baseline LSTMs with ℓ_1-norm regularization. The same training hyper-parameters as the baseline are adopted, except a bigger dropout keep ratio of 0.6 (original 0.35). A weaker dropout is used because ℓ_1-norm is also a regularization to avoid overfitting. A too strong dropout plus ℓ_1-norm regularization can result in underfitting. The weight decay of ℓ_1-norm regularization is 0.0001. The sparsified network has validation perplexity and test perplexity of 82.40 and 78.60, respectively, which is approximately the same with the baseline. The sparsity of weights in the first LSTM layer, the second LSTM layer and the last output layer is 91.66%, 90.32%

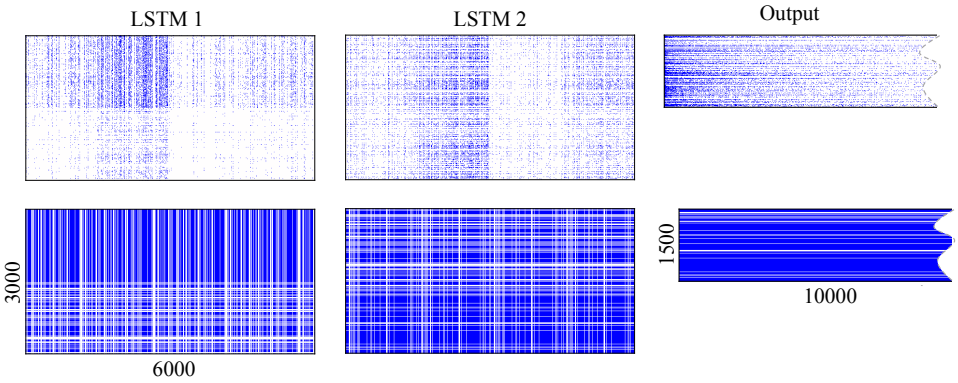

Figure 4: Intrinsic Sparse Structures unveiled by ℓ_1 regularization (zoom in for a better view). The top row shows the original weight matrices, where blue dots are nonzero weights and white ones refer zeros; the bottom row are the weight matrices in the format of Fig. 3, where white strips are ISS components whose weights are all zeros. For better visualization, the original matrices are evenly down-sampled by 10×10.

and 90.22%, respectively. Fig. 4 plots the learned sparse weight matrices. The sparse matrices in the top row reveal some interesting patterns: there are lots of all-zero columns and rows, and their positions are highly correlated. Those patterns are profiled in the bottom row. Much to our surprise, sparsifying individual weight *independently* can converge to sparse LSTMs with many ISS removed—504 and 220 ISS components in the first and second LSTM layer are all-zeros. It unveils that sparse ISS *intrinsically* exist in LSTMs and the learning process can easily converge to the status with a high ratio of ISS removed. In Section 3.2, we propose a learning method to *explicitly* remove much more ISS than the implicit ℓ_1-norm regularization.

3.2 Learning Method

Suppose $\mathbf{w}_k^{(n)}$ is a vector of all weights in the k-th component of ISS in the n-th LSTM layer ($1 \leq n \leq N$ and $1 \leq k \leq K^{(n)}$), where N is the number of LSTM layers and $K^{(n)}$ is the number of ISS components (*i.e.*, hidden size) of the n-th LSTM layer. The optimization goal is to remove as many weight groups $\mathbf{w}_k^{(n)}$ as possible without losing accuracy. Methods to remove weight groups (such as filters, channels and layers) have been successfully studied in CNNs as summarized in Section 2. However, how these methods perform in RNNs is unknown. Here, we extend the group Lasso based methods (Yuan & Lin (2006)) to RNNs for ISS sparsity learning. More specific, the group Lasso regularization is added to the minimization function in order to encourage sparsity in ISS. Formally, the ISS regularization is

$$R(\mathbf{w}) = \sum_{n=1}^{N} \sum_{k=1}^{K^{(n)}} \left|\left|\mathbf{w}_k^{(n)}\right|\right|_2, \tag{2}$$

where $\mathbf{w}$ is the vector of all weights and $||\cdot||_2$ is ℓ_2-norm (*i.e.*, Euclidean length). In *Stochastic Gradient Descent* (SGD) training, the step to update each ISS weight group becomes

$$\mathbf{w}_k^{(n)} \leftarrow \mathbf{w}_k^{(n)} - \eta \cdot \left(\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}_k^{(n)}} + \lambda \cdot \frac{\mathbf{w}_k^{(n)}}{\left|\left|\mathbf{w}_k^{(n)}\right|\right|_2} \right), \tag{3}$$

where $E(\mathbf{w})$ is data loss, η is learning rate and $\lambda > 0$ is the coefficient of group Lasso regularization to trade off recognition accuracy and ISS sparsity. The regularization gradient, *i.e.*, the last term in Eq. (3), is a unit vector. It constantly squeezes the Euclidean length of each $\mathbf{w}_k^{(n)}$ to zero, such that, a high portion of ISS components can be enforced to fully-zeros after learning. To avoid division by zero in the computation of regularization gradient, we can add a tiny number ϵ in $||\cdot||_2$, that is,

$$\left|\left|\mathbf{w}_k^{(n)}\right|\right|_2 \triangleq \sqrt{\epsilon + \sum_j \left(w_{kj}^{(n)}\right)^2}, \tag{4}$$

where $w_{kj}^{(n)}$ is the j-th element of $\mathbf{w}_k^{(n)}$. We set $\epsilon = 1.0e-8$ in this work. The learning method can effectively squeeze many groups near zeros, but it is very hard to exactly stabilize them as zeros because of the always-present fluctuating weight updates. Fortunately, the fluctuation is within a tiny ball centered at zero. To stabilize the sparsity during training, we zero out the weights whose absolute values are smaller than a pre-defined threshold τ. The process of thresholding is applied per mini-batch.

4 EXPERIMENTS

Our experiments use published models as baselines. The application domains include language modeling of Penn TreeBank and machine Question Answering of SQuAD dataset. For more comprehensive evaluation, we sparsify ISS in models with both a large hidden size of 1500 and a small hidden size of 100. We attempt to maximize threshold τ to fully exploit the benefit. For a specific application, we preset τ by cross validation. The maximum τ which sparsifies the dense model (baseline) without deteriorating its performance is selected. The validation of τ is performed only once and no training effort is needed. τ is $1.0e-4$ and $4.0e-4$ for Penn TreeBank and SQuAD, respectively. We used HyperDrive by Rasley et al. (2017) to explore the hyperparameter of λ. More details can be found in our source code.

To measure the inference speed, the experiments were run on a dual socket Intel Xeon CPU E5-2673 v3 @ 2.40GHz processor with a total of 24 cores (12 per socket) and 128GB of memory. Intel MKL library 2017 update 2 was used for matrix-multiplication operations. OpenMP runtime was utilized for parallelism. We used Intel C++ Compiler 17.0 to generate executables that were run on Windows Server 2016. Each of the experiments was run for 1000 iterations, and the execution time was averaged to find the execution latency.

4.1 LANGUAGE MODELING

A RNN with two stacked LSTM layers for language modeling (Zaremba et al. (2014)) is selected as the baseline. It has hidden sizes of 1500 (*i.e.*, 1500 components of ISS) in both LSTM units. The output layer has a vocabulary of 10000 words. The dimension of word embedding in the input layer is 1500. Word embedding layer is not sparsified because the computation of selecting a vector from a matrix is very efficient. The same training scheme as the baseline is adopted to learn ISS sparsity, except a larger dropout keep ratio of 0.6 versus 0.35 in the baseline for the reason explained in Section 3.1. All models are trained from scratch for 55 epochs. The results are shown in Table 1. Note that, when trained using dropout keep ratio of 0.6 without adopting group Lasso regularization, the baseline over-fits and the lowest validation perplexity is only 97.73. The trade-off of perplexity and sparsity is controlled by λ. In the second row, with tiny perplexity difference from baseline, our approach can reduce the number of ISS in the first and second LSTM unit from 1500, down to 373 and 315, respectively. It reduces the model size from 66.0M to 21.8M and achieves $10.59\times$ speedup without losing perplexity. Remarkably, the practical speedup ($10.59\times$) even goes beyond theoretical mult-add reduction ($7.48\times$) as shown in Table 1 —which comes from the increased computational efficiency. When applying structural sparsity, the underlying weight matrices become smaller so as to fit into the L3 cache with good locality, which improves the FLOPS (floating point operations per second). This is a key advantage of our approach over non-structurally sparse RNNs generated by connection pruning (Narang et al. (2017)), which suffers from irregular memory access pattern and inferior-theoretical speedup. At last, when learning a compact structure, our method can perform as structure regularization to avoid overfitting. As shown in the third row in Table 1, lower perplexity is achieved by even a smaller (25.2M) and faster ($7.10\times$) model. Its learned weight matrices are visualized in Fig. 5, where 1119 and 965 ISS components shown by white strips are removed in the first and second LSTM, respectively.

A straightforward way to reduce model complexity is to directly design a RNN with smaller hidden size and train from scratch. Compare with the direct design approach, our ISS method can automatically determine appropriate structures within LSTMs while maintaining an expected perplexity. More importantly, compact models learned by ISS method have lower perplexity, comparing with direct design method. To evaluate it, we directly design a RNN with exactly the same structure of the second RNN in Table 1 and train it from scratch instead of learning ISS from a larger RNN. The result is included in the last row of Table 1. We tuned dropout keep ratio to get best perplexity

Table 1: Learning ISS sparsity from scratch in stacked LSTMs.

Method	Dropout keep ratio	Perplexity (validate, test)	ISS # in (1st , 2nd) LSTM	Weight #	Total time*	Speedup	Mult-add reduction†
baseline	0.35	(82.57, 78.57)	(1500, 1500)	66.0M	157.0ms	1.00×	1.00×
ISS	0.60	(82.59, 78.65)	(373, 315)	21.8M	14.82ms	10.59×	7.48×
		(80.24, 76.03)	(381, 535)	25.2M	22.11ms	7.10×	5.01×
direct design	0.55	(90.31, 85.66)	(373, 315)	21.8M	14.82ms	10.59×	7.48×

* Measured with 10 batch size and 30 unrolled steps.
† The reduction of multiplication-add operations in matrix multiplication. Defined as (original Mult-add)/(left Mult-add)

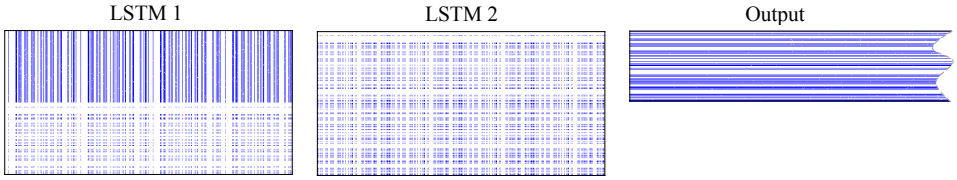

Figure 5: Intrinsic Sparse Structures learned by group Lasso regularization (zoom in for better view). Original weight matrices are plotted, where blue dots are nonzero weights and white ones refer zeros. For better visualization, original matrices are evenly down-sampled by 10×10.

for the directly-designed RNN. The final test perplexity is 85.66, which is 7.01 higher that our ISS method.

4.2 Machine Reading Comprehension

We evaluate ISS method by state-of-the-art dataset (SQuAD) and state-of-the-art model (BiDAF) with small ISS sizes. SQuAD (Rajpurkar et al. (2016)) is a recently released reading comprehension dataset, crowdsourced from $100,000+$ question-answer pairs on $500+$ Wikipedia articles. Exact-Match (EM) and F1 scores are two major metrics for the task[3]. The higher those scores are, the better the model is. We adopt BiDAF (Seo et al. (2017)) to evaluate how ISS method works in small LSTM units. BiDAF is a compact machine Question Answering model with totally 2.69M weights. The ISS sizes are only 100 in all LSTM units. The implementation of BiDAF is made available by its authors [4].

BiDAF has character, word and contextual embedding layers to extract representations from input sentences, following which are bi-directional attention layer, modeling layer, and final output layer. LSTM units are used in contextual embedding layer, modeling layer, and output layer. All LSTMs are bidirectional (Schuster & Paliwal (1997)). In a bidirectional LSTM, there are one forward plus one backward LSTM branch. The two branches share inputs and their outputs are concatenated for next stacked layers. We found that it is hard to remove ISS components in contextual embedding layer, because the representations are relatively dense as it is close to inputs and the original hidden size (100) is relatively small. In our experiments, we exclude LSTMs in contextual embedding layer and sparsify all other LSTM layers. Those LSTM layers are the computation bottleneck of BiDAF. We profiled the computation time on CPUs, and find those LSTM layers (excluding contextual embedding layer) consume 76.47% of total inference time. There are three bi-directional LSTM layers we will sparsify, two of which belong to the modeling layer, and one belongs to the output layer. More details of BiDAF are covered by Seo et al. (2017). For brevity, we mark the forward (backward) path of the 1st bi-directional LSTM in the modeling layer as `ModFwd1` (`ModBwd1`). Similarly, `ModFwd2` and `ModBwd2` are for the 2nd bi-directional LSTM. Forward (backward) LSTM path in the output layer are marked as `OutFwd` and `OutBwd`.

As discussed in Section 3.1 and shown in Fig. 3, multiple parallel layers can receive the hidden states from the same LSTM layer and all connections (weights) receive those hidden states belong to ISS of the shared LSTM. For instance, `ModFwd2` and `ModBwd2` both receive hidden states of

[3]Refer to Rajpurkar et al. (2016) for the definition of EM and F1 scores.
[4]`https://github.com/allenai/bi-att-flow/tree/dev`

Table 2: Remaining ISS components in BiDAF by fine-tuning.

EM	F1	ModFwd1	ModBwd1	ModFwd2	ModBwd2	OutFwd	OutBwd	weight #	Total time*
67.98	77.85	100	100	100	100	100	100	2.69M	6.20ms
67.21	76.71	100	95	78	82	71	52	2.08M	5.79ms
66.59	76.40	84	90	38	46	34	21	1.48M	4.52ms
65.29	75.47	54	47	22	30	18	12	1.03M	3.54ms
64.81	75.22	52	50	19	26	15	12	1.01M	3.51ms

* Measured with batch size 1.

Table 3: Remaining ISS components in BiDAF by training from scratch.

EM	F1	ModFwd1	ModBwd1	ModFwd2	ModBwd2	OutFwd	OutBwd	weight #	Total time*
67.98	77.85	100	100	100	100	100	100	2.69M	6.20ms
67.36	77.16	87	81	87	92	74	96	2.29M	5.83ms
66.32	76.22	51	33	42	58	37	26	1.17M	4.46ms
65.36	75.78	20	33	40	38	31	16	0.95M	3.59ms
64.60	74.99	23	22	35	35	25	14	0.88M	2.74ms

* Measured with batch size 1.

`ModFwd1` as inputs, therefore the k-th "ISS weight group" includes the k-th rows of weights in both `ModFwd2` and `ModBwd2`, plus the weights in the k-th ISS component within `ModFwd1`. For simplicity, we use "ISS of `ModFwd1`" to refer to the whole group of weights. Structures of six ISS are included in Table 4 in Appendix A. We learn ISS sparsity in BiDAF by both fine-tuning the baseline and training from scratch. All the training schemes keep as the same as the baseline except applying a higher dropout keep ratio. After training, we zero out weights whose absolute values are smaller than 0.02. This does not impact EM and F1 scores, but increase sparsity.

Table 2 shows the EM, F1, the number of remaining ISS components, model size, and inference speed. The first row is the baseline without ISS learning, and ISS sizes are all 100. The validation EM and F1 is 67.98 and 77.85, respectively. Other rows are obtained by fine-tuning baseline using ISS regularization. By learning ISS, with small EM and F1 loss, we can reduce ISS in all LSTMs except `ModFwd1` (the second row). For example, almost half of the ISS components are removed in `OutBwd`. By increasing the strength of group Lasso regularization (λ), we can increase the ISS sparsity by losing some EM/F1 scores. The trade-off is listed in Table 2. With only 2.63 F1 score loss, the sizes of `OutFwd` and `OutBwd` can be reduced from original 100 to 15 and 12, respectively. At last, we find it hard to reduce ISS sizes without losing any EM/F1 score. This implies that BiDAF is compact enough and its scale is suitable for both computation and performance (EM and F1 scores). However, our method can still significantly compress this compact model under acceptable performance loss.

At last, instead of fine-tuning baseline, we train BiDAF from scratch with ISS learning. The results are summarized in Table 3. Our approach also works well when training from scratch. Overall, training from scratch balances the sparsity across all layers better than fine-tuning, which results in even better compression of model size and speedup of inference time. The histogram of vector lengths of "ISS weight groups" is plotted in Appendix B.

5 Conclusion

We proposed *Intrinsic Sparse Structures* (ISS) within LSTMs and its learning method to simultaneously reduce the sizes of input updates, gates, hidden states, cell states and outputs within the sophisticated LSTM structure. By learning ISS, a structurally sparse LSTM can be obtained, which essentially is a regular LSTM with reduced hidden dimension. Thus, no software or hardware specific customization is required to get storage saving and computation acceleration. Though ISS is proposed with LSTMs, it can be easily extended to vanilla RNNs, *Gated Recurrent Unit* (GRU) (Cho et al. (2014)), and *Recurrent Highway Networks* (RHNs) (Zilly et al. (2017)).

REFERENCES

Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, 2016.

Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*, pp. 173–182, 2016.

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. Quasi-recurrent neural networks. *arXiv:1611.01576*, 2016.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv:1406.1078*, 2014.

Corinna Cortes, Xavi Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. Adanet: Adaptive structural learning of artificial neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 874–883, 2017.

Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, 2013.

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, 2016.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv:1510.00149*, 2015a.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, 2015b.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

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

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv:1405.3866*, 2014.

Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2554–2564, 2016.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations (ICLR)*, 2017.

Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 806–814, 2015.

Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. *arXiv:1705.08665*, 2017.

Zhiyun Lu, Vikas Sindhwani, and Tara N Sainath. Learning compact recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pp. 5960–5964. IEEE, 2016.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations (ICLR)*, 2017.

Sharan Narang, Gregory Diamos, Shubho Sengupta, and Erich Elsen. Exploring sparsity in recurrent neural networks. *arXiv:1704.05119*, 2017.

Christopher Olah. Understanding lstm networks. *GITHUB blog, posted on August*, 27:2015, 2015.

Jongsoo Park, Sheng Li, Wei Wen, Ping Tak Peter Tang, Hai Li, Yiran Chen, and Pradeep Dubey. Faster cnns with direct sparse convolutions and guided pruning. In *International Conference on Learning Representations (ICLR)*, 2017.

George Philipp and Jaime G Carbonell. Nonparametric neural networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Rohit Prabhavalkar, Ouais Alsharif, Antoine Bruguier, and Lan McGraw. On the compression of recurrent neural networks with an application to lvcsr acoustic modeling for embedded speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pp. 5970–5974. IEEE, 2016.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv:1606.05250*, 2016.

Jeff Rasley, Yuxiong He, Feng Yan, Olatunji Ruwase, and Rodrigo Fonseca. HyperDrive: Exploring Hyperparameters with POP Scheduling. In *Proceedings of the 18th International Middleware Conference*, Middleware '17. ACM, 2017.

Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. In *International Conference on Learning Representations (ICLR)*, 2017.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, 2016.

Wei Wen, Cong Xu, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Coordinating filters for faster deep neural networks. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2017.

Chunpeng Wu, Wei Wen, Tariq Afzal, Yongmei Zhang, Yiran Chen, and Hai Li. A compact dnn: Approaching googlenet-level accuracy of classification and domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.

Jaehong Yoon and Sung Ju Hwang. Combined group and exclusive sparsity for deep neural networks. In *International Conference on Machine Learning*, pp. 3958–3966, 2017.

Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv:1409.2329*, 2014.

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 4189–4198, 2017.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

APPENDIX A ISS IN BIDAF

Table 4: The ISS in BiDAF.

LSTM name	Dimensions of weight matrix	Receivers of hidden states	Size of "ISS weight group"
`ModFwd1`	900×400	`ModFwd2` `ModBwd2`	4800
`ModBwd1`	900×400	`ModFwd2` `ModBwd2`	4800

`ModFwd2`	300×400	`OutFwd`
		`OutBwd`
		logit layer for start index
			3201

`ModBwd2`	300×400	`OutFwd`
		`OutBwd`
		logit layer for start index
			3201

`OutFwd`	1500×400	logit layer for end index	6401
`OutBwd`	1500×400	logit layer for end index	6401

APPENDIX B HISTOGRAM OF VECTOR LENGTHS OF ISS WEIGHT GROUPS IN BIDAF

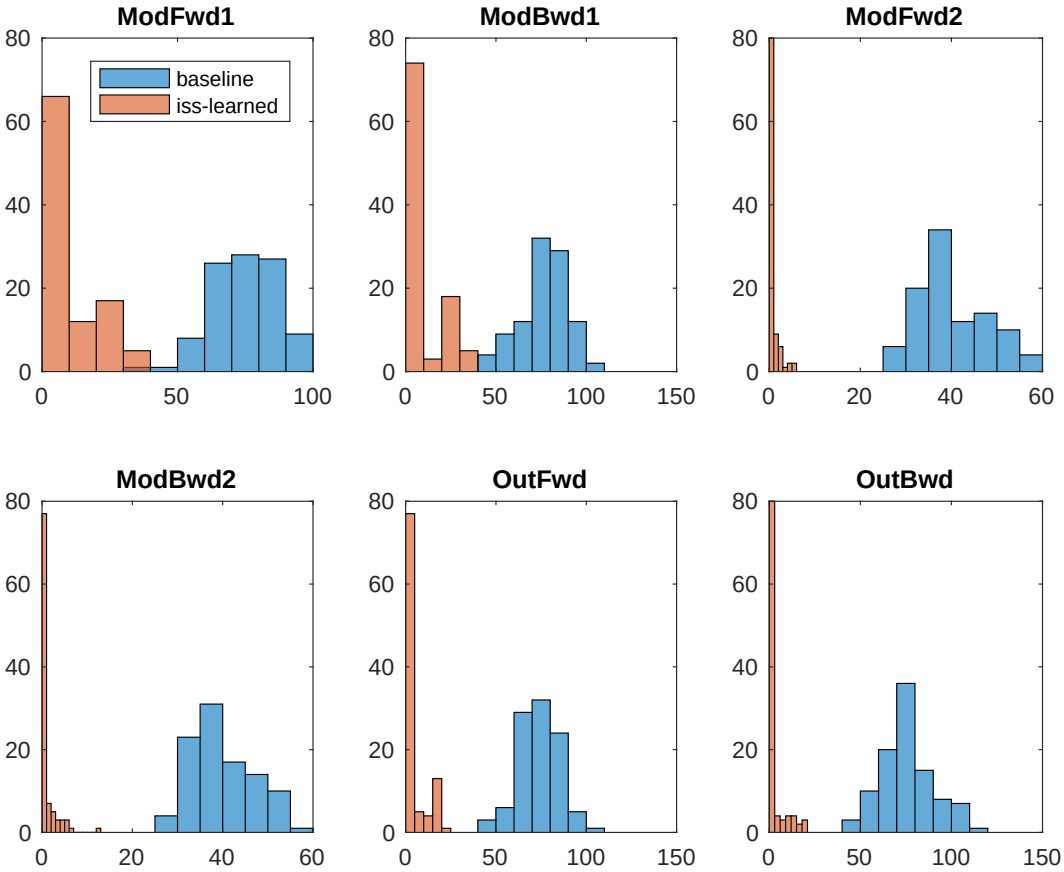

Figure 6: Histogram of vector lengths of "ISS weight groups" in BiDAF. The ISS-learned BiDAF is the one in the third row of Table 3 with EM 66.32 and F1 76.22. Using our approach, the lengths are regularized closer to zeros with a peak at the zero, resulting in high ISS sparsity.