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 decrease the sizes of all basic structures by one simultaneously and thereby always maintain the dimension consistency. By learning ISS within LSTM units, the obtained LSTMs remain regular while having much smaller basic structures. Our method achieves $10.59 \times$ speedup in state-of-the-art LSTMs, without losing any perplexity of language modeling of Penn TreeBank dataset. It is also successfully evaluated through a compact model with only $2.69$M 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 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 was either counterproductive or ignorable. More specific, with sparsity 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.

1 Defined as (new speed)/(original speed).
2 Defined as the percentage of zeros.
In this work, we focus on learning structurally sparse LSTMs for computation efficiency. More specifically, we aim to reduce the number of input updates, gates, hidden states, cell states and outputs simultaneously during learning, such that the obtained LSTMs can maintain the original schematic with dense connections and smaller sizes of these basic structures. Such compact models help save memory and execution time. Moreover, off-the-shelf libraries in deep learning frameworks can be directly utilized to deploy the reduced LSTMs.

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 a mismatch of their dimensions and then inducing invalid recurrent units. In essence, the dimension consistency is introduced by the element-wise operations in recurrent units, e.g., element-wise multiplication/addition among input updates, gates, states and outputs in LSTMs. 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 meanwhile 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× 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 public 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 CNNs 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 ℓ1 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, compact LSTM units remain original structural schematic 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)

\[
\begin{align*}
    i_t &= \sigma (x_t \cdot W_{xi} + h_{t-1} \cdot W_{hi} + b_i) \\
    f_t &= \sigma (x_t \cdot W_{xf} + h_{t-1} \cdot W_{hf} + b_f) \\
    o_t &= \sigma (x_t \cdot W_{xo} + h_{t-1} \cdot W_{ho} + b_o) \\
    u_t &= \tanh (x_t \cdot W_{xu} + h_{t-1} \cdot W_{hu} + b_u) \\
    c_t &= f_t \odot c_{t-1} + i_t \odot u_t \\
    h_t &= o_t \odot \tanh (c_t)
\end{align*}
\]

Figure 1: Intrinsic Sparse Structures (ISS) in LSTM units.

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 first need to determine how to sparsify weights. In LSTMs, input updates, gates, states and outputs are dynamically activated by stochastic inputs. To permanently remove a component of ISS, the associated weights of those activations should be all zeros. As such, the problem is converted to how to sparsify weight matrices when learning ISS. There are totally eight
Weights matrices in LSTMs

Figure 2: Applying Intrinsic Sparse Structures in weight matrices.

weights in Eq. (1). We organize them in the form of Fig. 2 as basic LSTM cells in TensorFlow. We can remove one component of ISS by zeroing out all weights in the white rows and white columns in Fig. 2. Here, biases are omitted for simple discussion without loss of generality.

Suppose the \( k \)-th hidden state of \( h \) 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. 2), 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 \( W_{xo} \) and \( W_{ho} \), these weights can be zeroed out (as shown by the fourth vertical white line in Fig. 2). Tracing back against the computation flow in Fig. 1 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. 2. 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. For example, the weight number can be 3200 for an ISS weight group of the first layer in a network with two stacked LSTM layers, whose input and hidden dimensions are both 200. However, the proposed ISS intrinsically exists within LSTMs and can be unveiled by even independently sparsifying each weight using \( \ell_1 \)-norm regularization. We take the large stacked LSTMs by Zaremba et al. (2014) for language modeling as the example of sparsifying weights by \( \ell_1 \)-norm regularization. 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 the two stacked 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 \( \ell_1 \)-norm regularization. The same training hyper-parameters with the baseline are adopted, except a bigger dropout keep ratio of 0.6 (original 0.35). A weaker dropout is used because \( \ell_1 \)-norm is also a regularization to avoid overfitting. A too strong dropout plus \( \ell_1 \)-norm regularization can result in underfitting. The weight decay of \( \ell_1 \)-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\% and 90.22\%, respectively. Fig. 5 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 \( \ell_1 \)-norm regularization.

3.2 Learning Method

Suppose \( w_k^{(n)} \) is a vector of all weights in the \( k \)-th component of ISS in the \( n \)-th LSTM layer (1 ≤ \( n \) ≤ \( N \) and 1 ≤ \( k \) ≤ \( K^{(n)} \)), 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 \( w_k^{(n)} \) 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

\[
R(w) = \sum_{n=1}^{N} \sum_{k=1}^{K^{(n)}} \|w_k^{(n)}\|_2^2, 
\]

(2)

where \( w \) is the vector of all weights and \( \| \cdot \|_2 \) is \( \ell_2 \)-norm (i.e., Euclidean length). In Stochastic Gradient Descent (SGD) training, the step to update each ISS weight group becomes

\[
w_k^{(n)} \leftarrow w_k^{(n)} - \eta \cdot \left( \sum_{j=1}^{n} \frac{\partial E(w)}{\partial w_k^{(n)}} + \lambda \cdot \frac{\|w_k^{(n)}\|_2}{\|w_k^{(n)}\|_2} \right),
\]

(3)

where \( E(w) \) is data loss, \( \eta \) 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 \( w_k^{(n)} \) to zero, such that, a high proportion 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 \( \epsilon \) in \( \| \cdot \|_2 \), that is,

\[
\|w_k^{(n)}\|_2 \triangleq \sqrt{\epsilon + \sum_{j} w_{kj}^{(n)}},
\]

(4)

where \( w_{kj}^{(n)} \) is the \( j \)-th element of \( w_k^{(n)} \). We set \( \epsilon = 1.0 \times 10^{-8} \) 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 \( \tau \). The process of thresholding is applied after each iteration.

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 com-
prehensive 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 \( \tau \) to fully exploit the benefit. For a specific application, we preset \( \tau \) by cross validation. The maximum \( \tau \) which sparsifies the baseline model without deteriorating its performance is selected. The validation of \( \tau \) is performed only once and no training effort is needed. \( \tau \) 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 \( \lambda \). 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 exactly same training scheme of 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. The trade-off of perplexity and sparsity is controlled by \( \lambda \). 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× speedup without losing perplexity. Remarkably, the practical speedup (10.59×) even goes beyond theoretical mult-add reduction (7.48×) 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×) model. Its learned weight matrices are visualized in Fig. 4, 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 smaller RNN 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 for the directly-designed RNN. The final test perplexity is 85.06, which is 7.01 higher that our ISS method.

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Dropout keep ratio</th>
<th>Perplexity (validate, test)</th>
<th>ISS # in (1st, 2nd) LSTM</th>
<th>Weight #</th>
<th>Total time</th>
<th>Speedup</th>
<th>Mult-add reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td>0.35</td>
<td>(82.57, 78.57)</td>
<td>(1500, 1500)</td>
<td>66.0M</td>
<td>157.0ms</td>
<td>1.00×</td>
<td>1.00×</td>
</tr>
<tr>
<td>ISS</td>
<td>0.60</td>
<td>(82.59, 78.65)</td>
<td>(373, 315)</td>
<td>21.8M</td>
<td>14.82ms</td>
<td>10.59×</td>
<td>7.48×</td>
</tr>
<tr>
<td>direct design</td>
<td>0.55</td>
<td>(90.31, 85.66)</td>
<td>(373, 315)</td>
<td>21.8M</td>
<td>14.82ms</td>
<td>10.59×</td>
<td>7.48×</td>
</tr>
</tbody>
</table>

1. Measured with 10 batch size and 30 unrolled steps.
2. The reduction of multiplication-add operations in matrix multiplication. Defined as (original Mult-add)/(left Mult-add).
Figure 4: 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 $10 \times 10$.

<table>
<thead>
<tr>
<th>LSTM name</th>
<th>Dimensions of weight matrix</th>
<th>Receivers of hidden states</th>
<th>Size of “ISS weight group”</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModFwd1</td>
<td>900 $\times$ 400</td>
<td>ModFwd2, ModBwd2</td>
<td>4800</td>
</tr>
<tr>
<td>ModBwd1</td>
<td>900 $\times$ 400</td>
<td>ModFwd2, ModBwd2</td>
<td>4800</td>
</tr>
<tr>
<td>ModFwd2</td>
<td>300 $\times$ 400</td>
<td>OutFwd, OutBwd</td>
<td>3201</td>
</tr>
<tr>
<td></td>
<td></td>
<td>logit layer for start index</td>
<td></td>
</tr>
<tr>
<td>ModBwd2</td>
<td>300 $\times$ 400</td>
<td>OutFwd, OutBwd</td>
<td>3201</td>
</tr>
<tr>
<td></td>
<td></td>
<td>logit layer for start index</td>
<td></td>
</tr>
<tr>
<td>OutFwd</td>
<td>1500 $\times$ 400</td>
<td>logit layer for end index</td>
<td>6401</td>
</tr>
<tr>
<td>OutBwd</td>
<td>1500 $\times$ 400</td>
<td>logit layer for end index</td>
<td>6401</td>
</tr>
</tbody>
</table>

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. 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.[3]

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. 2, 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 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 2. We learn ISS sparsity in BiDAF by both fine-tuning the baseline and training from scratch. All the training schemes keep 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 3 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 increasing the strength of group Lasso regularization ($\lambda$), we can increase the ISS sparsity by losing some EM/F1 scores. The trade-off is listed in Table 3. 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 4. 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.

## 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


