Neural Networks 123 (2020) 134–141

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Structured pruning of recurrent neural networks through neuron selection

Liangjian Wen [a], Xuanyang Zhang [a], Haoli Bai [b], Zenglin Xu [a,c,*]

[a] SMILE Lab, School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610031, China
[b] Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin NT 999077, Hong Kong SAR
[c] Center of Artificial Intelligence, Peng Cheng Lab, Shenzhen, Guangdong, China

ARTICLE INFO

Article history:
Received 17 June 2019
Received in revised form 1 October 2019
Accepted 19 November 2019
Available online 5 December 2019

Keywords:
Feature selection
Recurrent neural networks
Learning sparse models
Model compression

ABSTRACT

Recurrent neural networks (RNNs) have recently achieved remarkable successes in a number of applications. However, the huge sizes and computational burden of these models make it difficult for their deployment on edge devices. A practically effective approach is to reduce the overall storage and computation costs of RNNs by network pruning techniques. Despite their successful applications, those pruning methods based on Lasso either produce irregular sparse patterns in weight matrices, which is not helpful in practical speedup. To address these issues, we propose a structured pruning method through neuron selection which can remove the independent neuron of RNNs. More specifically, we introduce two sets of binary random variables, which can be interpreted as gates or switches to the input neurons and the hidden neurons, respectively. We demonstrate that the corresponding optimization problem can be addressed by minimizing the L_0 norm of the weight matrix. Finally, experimental results on language modeling and machine reading comprehension tasks have indicated the advantages of the proposed method in comparison with state-of-the-art pruning competitors. In particular, nearly 20× practical speedup during inference was achieved without losing performance for the language model on the Penn TreeBank dataset, indicating the promising performance of the proposed method.

1. Introduction

Recurrent neural networks (RNNs) have recently achieved remarkable successes in multiple fields such as image captioning (Anderson et al., 2018; Vinyals, Toshev, Bengio, & Erhan, 2016), action recognition (Pan et al., 2019; Ye et al., 2018), music segmentation (Liu et al., 2018), question answering (Mao, Hao, Wang, & Huang, 2018; Sagara & Hagiwara, 2014), machine translation (Bahdanau, Cho, & Bengio, 2014; Luong, Pham, & Manning, 2015; Yang, Chen, Wang, & Xu, 2018), and language modeling (Byeon, Breuel, Raue, & Liwicki, 2015; Liu et al., 2018; Sutskever, Vinyals, & Le, 2014). These successes heavily rely on huge models trained on large datasets, especially for those RNN variants such as Long Short Term Memory (LSTM) networks (Hochreiter & Schmidhuber, 1997) and Gated Recurrent Unit (GRU) networks (Cho et al., 2014). With the increasing popularity of edge computing, a recent trend is to deploy these models onto end devices so as to allow off-line reasoning and inference. However, these models are generally of huge sizes and bring expensive computation and storage costs during inference, which makes the deployment difficult for those devices with limited resources. In order to reduce the overall computation and storage costs of these models, model compression on recurrent neural networks has been widely concerned.

Network pruning is one of the prominent approaches to tackle the compression of RNNs. Narang, Elsen, Diamos and Sengupta (2017a) presents a connection pruning method to compress RNNs efficiently. However, the obtained weight matrix via connection pruning has random and unstructured sparsity. Such unstructured sparse formats are unfriendly for efficient computation in modern hardware systems (Lebedev & Lempitsky, 2016; Zhao et al., 2017) due to irregular memory access in modern processors. Previous studies (Wen et al., 2017; Wen, Wu, Wang, Chen, & Li, 2016) have shown that speedup obtained with random sparse matrix multiplication on various hardware platforms is lower than expected. For example, varying the sparsity level in weight matrices of AlexNet in the range of 67.6%, 92.4%, 94.3%, 96.6%, and 97.2%, the speedup ratio was 0.25×, 0.52×, 1.36×, 1.04×, and 1.38×, respectively. A practical remedy to this problem is structured pruning where pruning individual neurons can directly trim weight matrix size such that structured sparse matrix multiplication efficiently utilizes the hardware resources.

* Correspondence to: School of Computer Science and Engineering, University of Electronic Science and Technology of China, No.2006, Xiyuan Ave, West Hi-Tech Zone, 611731 Chengdu, China.
E-mail address: zlxu@uestc.edu.cn (Z. Xu).

https://doi.org/10.1016/j.neunet.2019.11.018
0893-6080/

Due to the promising properties of structured pruning, the structured pruning on deep neuron networks(DNNs) has been widely explored (Ding, Ding, Guo, & Han, 2019; He, Liu, Wang, Hu, & Yang, 2019; He, Zhang, & Sun, 2017; Zhuang et al., 2018). However, compared with the structured pruning on DNNs, there is a vital challenge originated from recurrent structure of RNNs, which is shared across all the time steps in a sequence. Structured pruning methods used in DNNs cannot be directly applied to RNNs. The reason is that independently removing the links can result in a mismatch of feature dimensions and then induce invalid recurrent units. In contrast, this problem does not exist in DNNs, where neurons can be independently removed without violating the usability of the final network structure. Accordingly group sparsity (Louizos, Welling, & Kingma, 2017) is difficult to be applied in RNNs.

To address this issue, we explore a new type of method along the line of structured pruning of RNNs through neuron selection. In detail, we introduce two sets of binary random variables, which can be interpreted as gates to the neurons, to indicate the presence of the input neurons and the hidden neurons, respectively. The two sets of binary random variables are then used to generate sparse masks for the weight matrix. More specifically, the presence of the matrix entry w_{ij} depends on both the presence of the ith input unit and the jth output unit, while the value of w_{ij} indicates the strength of the connection if $w_{ij} \neq 0$. However, the optimization of these variables is computationally intractable due to the nature of $2^{|h|}$ possible states of binary gate variable vector h. We then develop an efficient L_0 inference algorithm for inferring the binary gate variables, motivated from the work of pruning DNN weights (Louizos et al., 2017; Srinivas, Subramanya, & Babu, 2017).

While previous efforts on structured pruning of RNNs resort to the group lasso (i.e., the $L_{2,1}$ norm regularization) for learning sparsity (Wen et al., 2017), the lasso based methods are shown to be insufficient in inducing sparsity for large scale non-convex problems such as the training of DNNs (Collins & Kohli, 2014; Srinivas et al., 2017). In contrast, the expected L_0 minimization closely resembles spike-and-slab priors (Mitchell & Beauchamp, 1988; Xu, Zhe, Qi, & Yu, 2016; Zhe, Xu, Qi, & Yu, 2015) used in Bayesian variable selection (Louizos et al., 2017; Srinivas et al., 2017). The spike-and-slab priors can induce high sparsity and encourage large values at the same time due to the richer parameterization of these priors, while LASSO shrinks all parameters until lot of them are close to zero. And the L_0-norm regularization explicitly penalizes parameters for being different than zero with no other restrictions. Hence compared with Intrinsic Sparse Structures (ISS) via Lasso proposed by Wen et al. (2017), our neuron selection via the L_0 norm regularization can achieve higher adequate sparsity in RNNs.

In this paper, we propose a new type of method to prune individual neurons of RNNs. Our key contribution is that we introduce binary gates on recurrent and input units such that sparse masks for the weight matrix can be generated, allowing for effective neuron selection under sparsity constraint. For the first work of neuron selection in RNNs, we attempt to employ the smoothed mechanism for the L_0 regularized objective proposed in Louizos et al. (2017), motivated from Srinivas et al. (2017).

We evaluate our structured pruning method on two tasks, i.e., language modeling and machine reading comprehension. For example, in the case of language modeling of the word level on the Penn Treebank dataset, our method achieves the state-of-the-art results, i.e., the model size is reduced by more than 10 times, and the inference of the resulted sparse model is nearly 20 times faster than that of the original model. We also achieve encouraging results for the recurrent highway networks (Zilly, Srivastava, Koutník, & Schmidhuber, 2017) on language modeling and BiDAF model (Seo, Kembhavi, Farhadi, & Hajishirzi, 2016) on machine reading comprehension.

2. Related work

Despite model compression has achieved impressive success in DNNs (e.g., CNNs) (Ayinde, Inanc, & Zurada, 2019; Han, Mao, & Dally, 2016; Mohammed & Lim, 2017), it is difficult to directly apply this technology of compressing DNNs to the compression of RNNs due to the recurrent structure in RNNs. There are some recent efforts on the compression of RNNs. Generally, the compression techniques on RNNs can be categorized into the following types: pruning (Narang, Elsen, Diamos & Sengupta, 2017a; Narang, Elsen & Diamos, 2017; Wen et al., 2017), low-rank matrix/tensor factorization (Prabhavalkar, Alsharif, Bruguier, & McGraw, 2016; Ye et al., 2018; Zilly et al., 2017) and quantization (Hubara, Courbariaux, Soudry, El-Yaniv, & Bengio, 2016; Wang et al., 2018). Wang, Lin and Zhongfeng (2018) introduce several strategies including gate activation sparsity, top-k pruning schemes and mixed quantization schemes to compress LSTMs. Our work lies in the branch of pruning.

Pruning approaches can be further divided into non-structured pruning and structured pruning. For non-structured pruning, elements of the weight matrices can be removed based on some criteria. For example, Narang, Elsen, Diamos and Sengupta (2017a) present a magnitude-based pruning approach for RNNs, i.e., at every iteration, the top-k elements of the weights are set as 0. While such an approach can achieve over 90% sparsity in RNNs of Deep Speech 2 model with a minor decrease of accuracy, the obtained non-structured sparse matrices cannot efficiently accelerate the computation in modern computing platforms due to the irregular memory access. To improve this, Narang, Elsen and Diamos (2017) further proposed block-structured pruning in RNNs via the group lasso regularization. It extends the approach in Narang, Elsen, Diamos and Sengupta (2017a) to prune blocks of a matrix instead of individual weights. Wen et al. (2017) also proposed Intrinsic Structured Sparsity (ISS) for LSTMs by collectively removing the columns and rows for the weight matrices. ISS reduces the sizes of basic structures within LSTM units and is more hardware-friendly for acceleration compared with block structure in Narang, Elsen and Diamos (2017). Similar to Narang, Elsen and Diamos (2017), ISS also relies on group lasso for sparsity learning. Nevertheless, lasso regularization is shown to be insufficient for large non-convex problems, e.g., the training of DNNs (Collins & Kohli, 2014). To alleviate this challenge in Narang, Elsen and Diamos (2017) and Wen et al. (2017), we instead present a structured sparsity learning method through L_0 regularization, which not only reduces the size of basic structures of LSTMs but also achieves higher sparsity for non-convex RNN training in a tractable way.

3. Structured pruning of LSTMs through neuron selection

Without loss of generality, we focus on the compression of LSTMs (Hochreiter & Schmidhuber, 1997), a common variant of RNN that learns long-term dependencies. Note that our method can be readily applied to the compression of GRUs and vanilla RNNs. Before presenting the proposed sparsification methods, we first introduce the LSTM network.

$$
\begin{aligned}
i_t &= \sigma(W^i x_t + U^i h_{t-1} + b_i),\\
f_t &= \sigma(W^f x_t + U^f h_{t-1} + b_f),\\
o_t &= \sigma(W^o x_t + U^o h_{t-1} + b_o),\\
u_t &= \tanh(W^u x_t + U^u h_{t-1} + b_u),\\
c_t &= i_t \odot u_t + f_t \odot c_{t-1},\\
h_t &= o_t \odot \tanh(c_t),
\end{aligned}
\tag{1}
$$

where $\sigma(\cdot)$ is the sigmoid function, $\odot$ denotes the element-wise multiplication and $\tanh(\cdot)$ is the hyperbolic tangent function. x_t

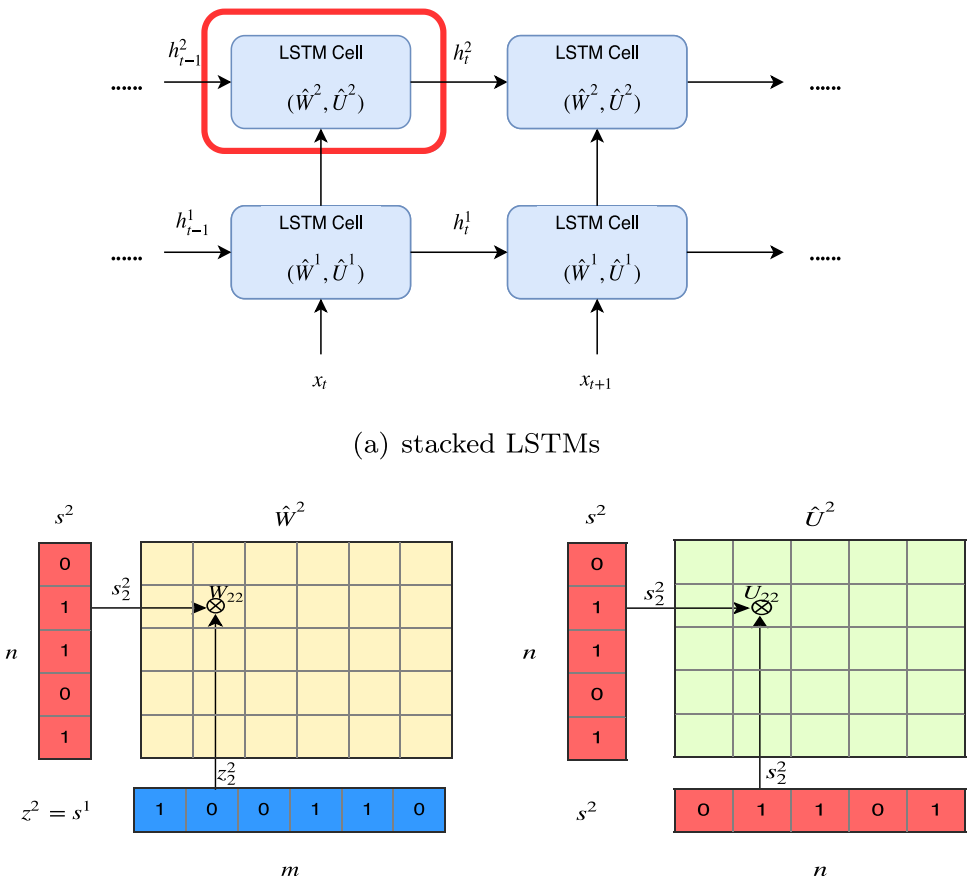

Fig. 1. Illustration on the proposed gate mechanism. (a) shows an illustration of stacked LSTMs with $\hat{W}$ and $\hat{U}$, where the parameters in the red box are illustrated in (b) and (c); (b) shows the gate z^2 and s^2 controlling the input-to-hidden matrix $\hat{W}^2$ activation of the input neurons, where $z^2 = s^1$ since the output of W^1 is the input to W^2; and (c) shows the gate s^2 controlling the hidden-to-hidden weight matrix U^2.

denotes the input vector at the time-step t, h_t denotes the current hidden state, and c_t denotes the long-term memory cell state. i_t, f_t and o_t correspond to the input gate, the forget gate and the output gate, respectively. For notation simplicity, we let $W = \{W^i, W^f, W^o, W^c\}$ be the input-to-hidden weight matrices, and $U = \{U^i, U^f, U^o, U^c\}$ be the hidden-to-hidden weight matrices.

3.1. Neuron selection via binary gates

To prune individual neurons in LSTMs, which can simultaneously reduce the size, FLOPs of LSTMs, we introduce neuron selection mechanisms into the design of LSTMs. As illustrated in Fig. 1(b) and (c), we introduce two auxiliary sets of binary "gate" or "switch" variables: one set of variables (denoted by $z = \{z_i\}$) controls the presence of the input neuron i, where $z_i \in \{0, 1\}$ and $|z| = m$ is the number of input neurons the other set of binary gate switch variables (denoted by $s = \{s_j\}$) controls the presence of the hidden neuron j, where $s_j \in \{0, 1\}$ and $|s| = n$ is the number of hidden neurons. In this way, w_{ij} controls the strength of the link from the input neuron i to the hidden neuron j, while z_i and s_j control the presence of neurons, and the mask on w_{ij} can be calculated by $z_i \times s_j$. In particular, such a gating mechanism can induce structured sparsity on the weight matrices. If $z_i = 0$, all hidden neurons connected to i will be switched off meaning that the ith column of W will be all zeros; and $s_j = 0$ will turn off each row of W. Therefore, by sharing the binary masks across all the gates, we can obtain structured sparsity on the weight matrices. For convenience, we re-parameterize the original parameter matrices W and U to $\hat{W}$ and $\hat{U}$ as follows,

$$\hat{W} = W \odot (zs^\top), \qquad \hat{U} = U \odot (ss^\top), \tag{2}$$

where $\odot$ denotes the element-wise product operation. In this way, w_{ij} controls the strength of the link from the input neuron i to the hidden neuron j, while z_i and s_j control the presence of neurons. Furthermore, our proposed switch mechanism can be directly applied to stacked LSTMs. As illustrated in Fig. 1(a), since the output of the lth layer is the input of the $l+1$th layer, we have $z^{l+1} = s^l$.

To model the uncertainty of each of random "gate" variables z_i and s_j, we let $z_i \sim \text{Bern}(\pi_{z_i})$ and $s_j \sim \text{Bern}(\pi_{s_j})$, where π_{z_i} and π_{s_j} are the parameters of the Bernoulli distributions, and denote the probability of random variables z_i and s_j taking value 1, respectively. We can optimize this objective function with π_z and π_s by minimizing the L_0 norm of the weight matrix to achieve neuron selection. The L_0 norm regularization can explicitly penalize no-zero parameters of models without further restrictions (Louizos et al., 2017), and it demonstrates superior advantages over the L_1 norm regularization in the sparse learning. A naive idea in learning sparsity is to directly utilize the L_0-norm regularization on the weights of LSTMs, which leads to the following objective function:

$$\mathcal{L}(W, U) = E_D(W, U) + \lambda_1 \|W\|_0 + \lambda_2 \|U\|_0,$$

$$W^*, U^* = \arg\min_{W,U} \mathcal{L}(W, U). \tag{3}$$

Here $\|W\|_0 = \sum_{i=1}^{m}\sum_{j=1}^{n} \mathbb{I}(w_{ij} \neq 0)$ denotes the L_0-norm, and $\|U\|_0$ follows a similar pattern. $\mathbb{I}(.)$ is the indicator function, and m and n denote the number of input units and hidden units, respectively. $E_D(W, U)$ represents the loss on the dataset D, and λ_1 and λ_2 are penalty parameters for the sparsity regularization.

Neuron selection via binary random variables generates masks for the weight matrix. With the gating mechanism, the L_0 norm in Eq. (3) can be further specified as $\|\hat{W}\|_0 = \sum_i \sum_j z_i \times s_j$ and $\|\hat{U}\|_0 = \sum_{j_1} \sum_{j_2} s_{j_1} \times s_{j_2}$. Hence we can seek to penalize the number of parameters appearing in LSTM on average. The expectation of Eq. (3) over the auxiliary masks is reformulated as follows,

$$\begin{aligned}
&\mathcal{L}(W, U, \pi_z, \pi_s) = \mathbb{E}_{\Phi(z|\pi_z)\Phi(s|\pi_s)}[\mathbb{E}_D(W, U, z, s)] \\
&+ \lambda_1 \sum_{i=1}\sum_{j=1} \pi_{z_i}\pi_{s_j} + \lambda_2 \sum_{i=1}\sum_{j=1}^{i\neq j} \pi_{s_i}\pi_{s_j} + \lambda_2 \sum_{j=1} \pi_{s_j}, \\
&(W^*, U^*, \pi_z^*, \pi_s^*) = \arg\min_{W,U,\pi_z,\pi_s} \mathcal{L}(W, U, \pi_z, \pi_s).
\end{aligned} \tag{4}$$

The item $s_j \times s_j$ in $\|\hat{U}\|_0 = \sum_{j_1} \sum_{j_2} s_{j_1} \times s_{j_2}$ denotes the mask of weight parameters U_{jj}. The term $s_j \times s_j$ only depends on the presence of the jth recurrent units. Hence the expectation of $s_j \times s_j$ is π_j. Note that to avoid quadratic terms on s_j, we write $\|\hat{U}\|_0$ as the summation of two terms as shown in Eq. (4). Since the binary gates are shared across layers in stacked RNNs, it may result in an extremely unbalanced sparse structure in different layers to penalize the gate variables of different layers with the same penalty factor. For this reason, we specify independent regularization for different layers.

3.2. Optimization

It is intractable to learn sparse parametric models by minimizing the L_0 norm based on gradient optimization. The optimization of the objective in Eq. (4) is problematic due to the discrete nature of z and s. In principle, the REINFORCE estimator (Williams, 1992) can be used to compute the gradients, but it suffers from high variance and slow convergence. On the other hand, the straight-through estimator (STE) (Bengio, Léonard, & Courville, 2013) can also be used, however, the mismatch of the parameters between the forward and backward pass in the optimization leads to biased gradients and updates.

A more appealing method is to continuously relax discrete random binary variables by a hard-sigmoid rectification of continuous random variable with a distribution (Louizos et al., 2017), as follows:

$$z = \min(\mathbf{1}, \max(\mathbf{0}, \tau_z)), \quad \tau_z \sim q(\tau_z|\phi_z), \tag{5}$$

where the $q(\tau_z|\phi_z)$ corresponds to the continue distribution with parameter ϕ_z. Then the probability of the discrete random binary variables being non-zero is computed by the cumulative distribution function(CDF) $Q(.)$ of $\mathbf{s}$, as follows:

$$\Phi_z(z \neq 0|\phi_z) = 1 - Q(\tau_z \leq 0|\phi_z). \tag{6}$$

Then it can not only enable gradient-based optimization of a generic loss by smoothing the binary gate variables $\mathbf{z}$, but also allow the variables z to be exactly zero. We can write the continuous distribution $q(\tau_z|\phi_z) = q(f(\phi_{\tau_z}, \epsilon))$ by using the reparameterization trick (Kingma & Welling, 2013; Rezende, Mohamed, & Wierstra, 2014), where $f(.)$ is a deterministic and differentiable function and ϵ denotes the uniform or Gaussian free noise. Since ϵ is independent of the parameters of models, we can directly take the gradient of the optimization target over the parameters of the distributions of random variables.

Similarly, we can apply the same procedure to smooth gate variables s with ϕ_s which denotes the parameters of the correlative continuous distribution. Armed with the above approach, we can generate z and s via the differentiable transformation in Eq. (8), and therefore shift the optimization over π_z and π_s in Eq. (4) to ϕ_z and ϕ_s in the hard concrete distributions. The original objective function in Eq. (4) can be reformulated as

$$\begin{aligned}
&\tilde{\mathcal{L}}(W, U, \phi_z, \phi_s) = \mathbb{E}_{\Phi(z|\phi_z)\Phi(s|\phi_s)}[\mathbb{E}_D(W, U, z, s)] \\
&+ \lambda_1 \sum_{i=1}\sum_{j=1} (\Phi_{z_i}(z_i \neq 0|\phi_{z_i}))(\Phi_{s_j}(s_j \neq 0|\phi_{s_j})) \\
&+ \lambda_2 \sum_{i=1}\sum_{j=1}^{i\neq j} (\Phi_{s_i}(s_i \neq 0|\phi_{s_i}))(\Phi_{s_j}(s_j \neq 0|\phi_{s_j})) \\
&+ \lambda_2 \sum_{i=1} \Phi_{s_i}(s_i \neq 0|\phi_{s_i}). \\
&(W^*, U^*, \phi_z^*, \phi_s^*) = \arg\min_{W,U,\pi_z,\pi_s} \tilde{\mathcal{L}}(W, U, \Phi_z, \Phi_s).
\end{aligned} \tag{7}$$

As shown in Louizos et al. (2017), an efficient choice of the smoothing continuous distribution is as follows; the binary concrete distribution (Jang, Gu, & Poole, 2017; Maddison, Mnih, & Teh, 2016) $q(\hat{\tau})$ with the parameters $\log\alpha$ and β, where $\log\alpha$ and β are the location and temperature parameters respectively, is stretched from the (0,1) interval to the (ζ, γ) interval,with $\zeta_z < 0$ and $\gamma_z > 1$. Then we apply a hard-sigmoid on its random samples. More specific, the procedure to smooth gate variables s is as follows:

$$\begin{aligned}
&u \sim \text{Uniform}(0, 1), \\
&\hat{\tau} = \text{ff}((\log u - \log(1-u) + \log\alpha_z)/\beta_z), \\
&\tau = \hat{\tau}(\zeta_z - \gamma_z) + \gamma_z, \\
&z = \min(1, \max(0, \tau)),
\end{aligned} \tag{8}$$

where σ is the sigmoid function as introduced before. The probability of the z being non-zero can be computed by the cumulative density function $\Phi(\cdot)$ of z as follows:

$$\Phi_z(z \neq 0|\phi_z) = \text{ff}(\log\alpha_z - \beta_z \log\frac{-\gamma_z}{\zeta_z}), \tag{9}$$

where $\phi_z = \{\alpha_z, \beta_z, \zeta_z, \gamma_z\}$ denotes the parameters of the hard concrete distribution. Similarly, we can apply the same procedure to smooth gate variables s.

During testing, we use the following estimator for the final z and s under a hard concrete smoothness:

$$z = \min(1, \max(0, \text{ff}(\log\alpha_z)(\zeta_z - \gamma_z) + \gamma_z)), \tag{10}$$

$$s = \min(1, \max(0, \text{ff}(\log\alpha_s)(\zeta_s - \gamma_s) + \gamma_s)). \tag{11}$$

4. Experiments

To compare with Intrinsic Sparse Structures (ISS) via Lasso proposed by Wen et al. (2017), we also evaluate our structured sparsity learning method with L_0 regularization on language modeling and machine reading tasks. In the case of language modeling, we seek to sparsify a stacked LSTM model (Zaremba, Sutskever, & Vinyals, 2014) and the state-of-the-art Recurrent Highway Networks (RHNs) (Zilly et al., 2017) on the Penn Treebank (PTB) dataset (Marcus, Marcinkiewicz, & Santorini, 1993). For the task of machine reading comprehension, we choose the Bi-Directional Attention Flow Model (BiDAF) (Seo et al., 2016) with a small hidden size of 100 on the SQuAD dataset (Rajpurkar, Zhang, Lopyrev, & Liang, 2016). While our structured L_0 norm imposes no shrinkage on the remaining components, the learned model could be over-fitted if weight decay is not assigned. Consequently, we follow a similar pattern in Louizos

Table 1
Learning structured sparsity from scratch in stacked LSTMs.

Method	Dropout keep ratio	Perplexity (validate, test)	LSTMs (input, L1, L2)	Size	Time (ms)	Speedup	Multi-add reduction
Vanilla model	0.35	(82.57, 78.57)	(1500, 1500, 1500)	66.0M	365.92 ± 10.3	1.00×	1.00×
ISS	0.60	(82.59, 78.65)	(1500, 373, 315)	21.8M	39.88 ± 0.7	9.17×	7.48×
Our method	0.65	(**81.62**, **78.08**)	(**251**, **296**, **247**)	**6.16M**	**18.87** ± **0.3**	**19.39**×	**13.95**×

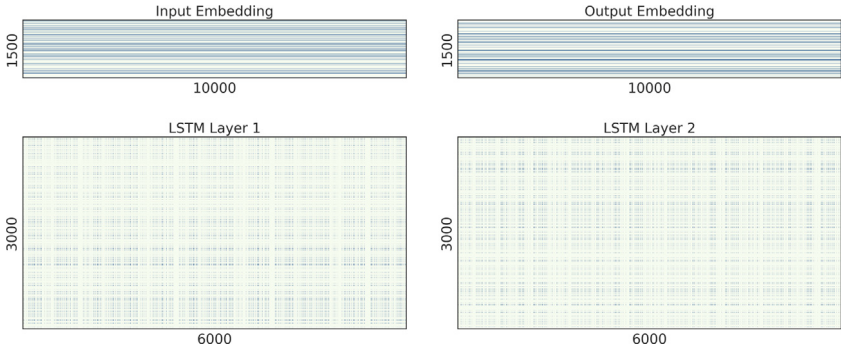

Fig. 2. Illustration of word embedding and weight matrices in the two-layer LSTM. The first row presents the embedding layer for the input and output respectively. The second row shows the weight matrices for the first and second layers. Note that we have concatenated $\{W^i, W^f, W^u, W^o\}$ and $\{U^i, U^f, U^u, U^o\}$ into a single matrix with shape 3000×6000. The blue dots are nonzero weights and the rest ones are structurally pruned to zeros. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

et al. (2017) to impose L_2 regularization on model parameters. For the setting of the hard concrete distribution, we follow the same pattern in Louizos et al. (2017) for all experiments, i.e., ζ, γ, β are taken as the hyper-parameters. For $\log \alpha$, it is updated by back-propagation of the network and initialized by samples from $\mathcal{N}(1, 0.1)$.

4.1. Language modeling

For language modeling, we evaluate two models: stacked LSTMs and recurrent highway neural networks. Both models are trained from scratch. We use the word level PTB dataset for language modeling, which consists of 929k training words, 73k validation words and 82k test words with 10,000 unique words in its vocabulary.

4.1.1. Stacked LSTMs

Baselines. We compare our proposed method against two baselines, the vanilla two-layer stacked LSTM used in Zaremba et al. (2014) and the ISS method (Wen et al., 2017), which is the state-of-the-art method in RNN compression. The dropout keep ratio is 0.35 for the vanilla model. The vocabulary size, embedding size and hidden size of the stacked LSTMs are set as 10,000, 1500 and 1500, respectively, which is consistent with the settings in Wen et al. (2017). The results of ISS are taken from the original paper.

Hyper-parameters Setting. We use NT-ASGD (Merity, Keskar, & Socher, 2017) for training with an initial learning rate equal to 20.0 and the gradient clipping set to 0.25. Similar to ISS (Wen et al., 2017), we increase the dropout keep ratio to 0.65 due to the intrinsically structured sparsity in the network. We use the default initialization strategy provided in PyTroch[1] for the input and output word embedding as well as the parameters of the LSTM.

Results. We show the results of stacked LSTMs in Table 1. It can be observed that our method finds the most compact structure of the model, i.e. the numbers of the first and second hidden units are reduced from 1500 to 296 and 247 respectively, both of which are significantly smaller than the vanilla stacked model and the ISS method. Besides, the dimension of word embedding vectors also decreases from 1500 to 251. Overall, our method reduces the model size from 66.0M to 6.16M, which is more than 10× reduction comparing to the vanilla model. Theoretically, the computation is reduced by nearly 14× in terms of multi-add operations. Compared with ISS in Wen et al. (2017), our structured sparsity learning method reduces the multi-add computation further by 1.86x and the model size further by 15.64M. The results indicate that our structured L_0 regularization can indeed sufficiently sparse the model. Additionally, despite the model size being sharply shrunk, our method still achieves the lowest perplexity on the PTB dataset. The excellent performance of our method can be explained by the superiority of our structured L_0 regularization since it poses no penalization over the remained parameters, while for ISS, the group lasso method penalizes the norms of all groups collectively, and thereon could affect the model capacity.

In order to evaluate the practical speedup of the learned structures of our proposed method and all the baseline, we measure the speedup of inference on CPU.[2] using TensorFlow with Intel MKL library[3] The time is measured with 10 batch size and 30 unrolled steps, and the result is averaged from 1000 times of inference with standard deviation reported. It can be seen from Table 1 that the practical inference time is 19.4× faster than the original model. The actual speedup is even higher than the theoretical result, and we conjecture that this could be due to some basic optimization in the MKL library.

To look into the learned structured sparsity, we further visualize the embedding and weights of the stacked LSTMs in Fig. 2 after training 200 epochs. We can see that after structured L_0 regularization, the size of word embedding is highly reduced, and similarly, most rows and columns of the weight matrices are pruned away. Therefore, such matrices can be re-arranged to a small and compact structure, leading to practical speed up during inference.

[1] https://pytorch.org/

[2] Intel CPU E5-2630 v4 @ 2.20 GHz processor with a total of 40 cores.

[3] https://www.tensorflow.org/guide/performance/overview.

Table 2
Ablative study the penalty parameters of stacked LSTMs.

The penalty scale $\lambda_1^{(1)}$, $\lambda_2^{(1)}$, $\lambda_1^{(2)}$ and $\lambda_2^{(2)}$	Perplexity (validate, test)	The dimension of input, 1st, 2nd
(1, 1, 1, 1)	(81.73, 78.33)	(1020, 215, 250)
(2, 1, 1, 1)	(82.87, 79.25)	(635, 146, 247)
(3, 1, 1, 1)	(85.49, 81.82)	(438, 110, 244)
(4, 1, 1, 1)	(87.37, 84.22)	(333, 88, 242)
(5, 1, 1, 1)	(90.83, 84.22)	(273, 70, 237)

* We use $\lambda_1^{(1)}$, $\lambda_2^{(1)}$, $\lambda_1^{(2)}$ and $\lambda_2^{(2)}$ to denote the times to $0.08/N$, e.g., $\lambda_1^{(1)} \times 0.08/N$ denotes the penalty parameter to the input neuron the first layer.

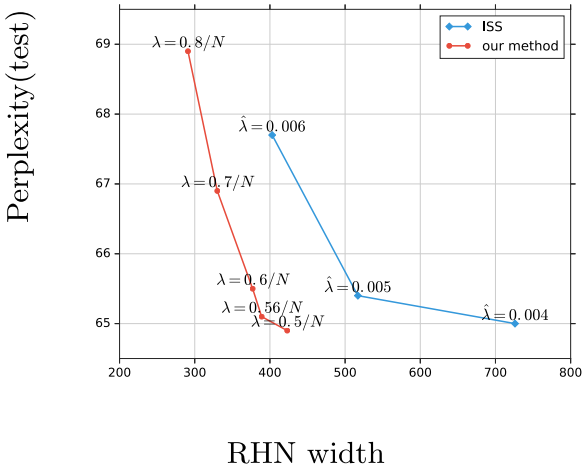

Fig. 3. Perplexity of our method and ISS under different sparsity levels with different regularization strength. For our method, we vary λ in $\{0.5/N, 0.56/N, 0.6/N, 0.7/N, 0.8/N\}$ (N denotes the size of the training data sample). In terms of ISS, we tune the group lasso regularization $\hat{\lambda}$ defined in its paper among $\{0.006, 0.005, 0.004\}$.

Ablative Study. We used a two-layer stacked LSTM to verify the sensitivity of the penalty parameters in Eq. (3). Experimentally, we find that setting all the penalty values to $0.08/N$ (where N denotes the number of training data) can lead to good performance. For convenience, we use $\lambda_1^{(1)}$, $\lambda_2^{(1)}$, $\lambda_1^{(2)}$ and $\lambda_2^{(2)}$ to denote the times to $0.08/N$, e.g., $\lambda_1^{(1)} \times 0.08/N$ denotes the penalty parameter to the input neuron the first layer. Here the super-index denotes the corresponding layer. Initially, we set all the parameters to the same value (i.e., $0.08/N$). We find that the numbers of the first-layer and second-layer hidden units are reduced from 1500 to 215 and 250 respectively, while the size of the input only decreases from 1500 to 1020.

Since the first input layer is usually much larger than the hidden layers, we increase the penalty value of $\lambda_1^{(1)}$ to drop out more input neurons and to seek much smaller networks. From Table 2, we can find that with a larger value of $\lambda_1^{(1)}$, the sparser input layer can be obtained while with a significant increase in the perplexity.

4.1.2. Recurrent highway networks

Our method can be further extended to Recurrent Highway Networks (RHNs) (Zilly et al., 2017). Due to the structure of RHNs, we only introduce the binary mask z and set its regularization as λ. During training, we impose L_0 regularization over z so as to learn structured sparsity.

Baselines. To evaluate our structured sparsity learning method, we choose "Variational RHN + WT" of Zilly et al. (2017) as our baseline model. The number of units per layer is defined as the width of RHNs. It has depth 10 and width 830, with totally $23.5M$ parameters. The implementation of RHNs is available from its authors.[4] Aside from the vanilla RHNs, we also compare to the ISS method, and its results are taken from Wen et al. (2017).

Table 3
Learning structured sparsity from scratch in RHNs.

Method	Perplexity (validate, test)	RHN width	Parameter
RNHs (Zilly et al., 2017)	(67.9, 65.4)	830	23.5M
ISS (Wen et al., 2017)	(68.1, 65.4)	517	11.1M
Our method	(**68.2, 65.1**)	**389**	**7.2M**

Hyper-parameters Setting. For our method, we use the same hyper-parameters as those in the baseline, except for that the parameters of models are initialized uniformly in $[-0.08, 0.08]$, the dropout ratios are multiplied by 0.75, and we divide the learning rate by a factor of 1.02 at every epoch after it reaches 35.

Results. Table 3 shows the results obtained by the baseline, ISS and our method. Comparing to the vanilla RHN and ISS method, our structured sparsity approach can significantly achieve a more compact model with width 387 without losing perplexity. The parameter size also decreases to $7.1M$, which is about 69.8% reduction.

We further investigate the trade-off between perplexity and sparsity of our method and ISS, and the plot is shown in Fig. 3. It can be observed that our method can achieve a higher reduction of RHNs width than ISS at the same perplexity. With the same width, our method can achieve lower perplexity as well. This again demonstrates the superiority of our structured L_0 regularization over group lasso methods in inducing sparsity for recurrent structures.

4.2. Machine reading comprehension

Machine Reading Comprehension (MRC) is one of the frontier tasks in the field of natural language processing. In MRC, the models answer a query about a given context paragraph, and Exact-Match (EM) and F1 scores are two major metrics for the evaluation (the higher the better). We use the benchmark Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016), which consists of 100,000+ questions crowd-sourced on more than 500 Wikipedia articles.

Baselines. We use the BiDirectional Attention Flow Model (BiDAF)[5] (Seo et al., 2016) as the backbone model to evaluate our structured sparsity learning method. The BiDAF is composed of bidirectional LSTMs, and our structured L_0 regularization method can be readily applied. We focus on sparsifying the two layers of the bidirectional LSTM denoted as ModFwd1, ModBwd1, ModFwd2, and ModBwd2, since they are shown to be most computationally expensive layers in Wen et al. (2017).

Hyper-parameter setting. We penalize different layers of the model in a similar pattern to the two-layer LSTMs used language modeling. We also increase the dropout keep ratio to 0.9 as the

[4] https://github.com/jzilly/RecurrentHighwayNetworks

[5] We use the code from https://github.com/allenai/bi-att-flow.

Table 4
Comparison between the vanilla trained BiDAF, ISS method and our method.

Method	EM	F1	ModFwd1	ModBwd1	ModFwd2	ModBwd2	outFwd	outBwd	Weight
Vanilla BiDAF	67.98	77.85	100	100	100	100	100	100	2.69M
ISS (Wen et al., 2017)	65.36	75.78	**20**	**33**	40	38	**31**	16	**0.95M**
Our method	65.67	75.69	26	**33**	**36**	**36**	33	**15**	0.96M

structured sparsity itself can prevent over-fitting to some extent. All the rest training schemes are the same as those in the baseline.

Results. Table 4 shows the EM, F1, the number of remaining components and model sizes obtained by the baseline, ISS and our method. As is mentioned in Wen et al. (2017), the scale of the vanilla BiDAF is compact enough on the SQuAD dataset, and it is thereon hard to reduce the hidden size of those LSTM layers in BiDAF without losing any EM/F1. Our structure sparsity method achieves competitive sparsity and performance comparing to the ISS method, both of which produce remarkably compressed models under acceptable degradation to the vanilla BiDAF. The result again demonstrates that our L_0 structured methods can be effectively used to discover the sparsity of recurrent neural networks.

5. Conclusion

In this paper, we propose a novel structured sparsity learning method for recurrent neural networks. By introducing binary gates on neurons, we penalize weight matrices through L_0 regularization, reduce the sizes of the network parameters significantly and lead to practical speedup during inference. We also demonstrate the superiority of our relaxed L_0 regularization over the group lasso used in previous methods. Our methods can be readily used in other recurrent structures such as Gated Recurrent Unit, and Recurrent Highway Networks.

For future work, we plan to explore the sparsity constraints for neuron selection for further reducing the number of model parameters, to exploit a more appealing unbiased, lower variance estimator (e.g., the unbiased ARMs-estimator recently proposed in Yin & Zhou, 2019) for neuron selection. We also plan to combine neuron selection with quantization algorithms to further reduce model sizes and FlOPs of RNNs.

Acknowledgments

This work was partially supported by NSF China (No. 61572111), a Fundamental Research Fund for the Central Universities of China (No. ZYGX2016Z003), and Startup Funding (No. G05QNQR004).

References

Anderson, P., He, X., Buehler, C., Teney, D., Johnson, M., Gould, S., et al. (2018). Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 6077–6086).

Ayinde, B. O., Inanc, T., & Zurada, J. M. (2019). Redundant feature pruning for accelerated inference in deep neural networks. *Neural Networks*, *118*, 148–158.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

Bengio, Y., Léonard, N., & Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432.

Byeon, W., Breuel, T. M., Raue, F., & Liwicki, M. (2015). Scene labeling with LSTM recurrent neural networks. In *CVPR* (pp. 3547–3555).

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., et al. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.

Collins, M. D., & Kohli, P. (2014). Memory bounded deep convolutional networks. *CoRR, abs/1412.1442*.

Ding, X., Ding, G., Guo, Y., & Han, J. (2019). Centripetal sgd for pruning very deep convolutional networks with complicated structure. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4943–4953).

Han, S., Mao, H., & Dally, W. J. (2016). Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding. *CoRR, abs/1510.00149*.

He, Y., Liu, P., Wang, Z., Hu, Z., & Yang, Y. (2019). Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4340–4349).

He, Y., Zhang, X., & Sun, J. (2017). Channel pruning for accelerating very deep neural networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*.

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

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., & Bengio, Y. (2016). Quantized neural networks: Training neural networks with low precision weights and activations. *CoRR, abs/1609.07061*.

Jang, E., Gu, S., & Poole, B. (2017). Categorical reparameterization with gumbel-softmax. In *5th international conference on learning representations, ICLR 2017, Toulon, France, April 24-26, 2017, conference track proceedings*.

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114.

Lebedev, V., & Lempitsky, V. (2016). Fast convnets using group-wise brain damage. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2554–2564).

Liu, H., He, L., Bai, H., Dai, B., Bai, K., & Xu, Z. (2018). Structured inference for recurrent hidden semi-markov model. In *Proceedings of the twenty-seventh international joint conference on artificial intelligence, ijcai 2018, july 13-19, 2018, stockholm, sweden* (pp. 2447–2453).

Louizos, C., Welling, M., & Kingma, D. P. (2017). Learning sparse neural networks through L_0 regularization. *CoRR, abs/1712.01312*.

Luong, T., Pham, H., & Manning, C. D. (2015). Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of the 2015 conference on empirical methods in natural language processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015* (pp. 1412–1421).

Maddison, C. J., Mnih, A., & Teh, Y. W. (2016). The concrete distribution: A continuous relaxation of discrete random variables. arXiv preprint arXiv:1611.00712.

Mao, X. -L., Hao, Y. -J., Wang, D., & Huang, H. (2018). Query completion in community-based question answering search. *Neurocomputing*, *274*, 3–7.

Marcus, M. P., Marcinkiewicz, M. A., & Santorini, B. (1993). Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, *19*(2), 313–330.

Merity, S., Keskar, N. S., & Socher, R. (2017). Regularizing and optimizing LSTM language models. arXiv preprint arXiv:1708.02182.

Mitchell, T. J., & Beauchamp, J. J. (1988). Bayesian variable selection in linear regression. *Journal of the American Statistical Association*, *83*(404), 1023–1032.

Mohammed, M. F., & Lim, C. P. (2017). A new hyperbox selection rule and a pruning strategy for the enhanced fuzzy min–max neural network. *Neural Networks*, *86*, 69–79.

Narang, S., Elsen, E., Diamos, G., & Sengupta, S. (2017a). Exploring sparsity in recurrent neural networks. arXiv preprint arXiv:1704.05119.

Narang, S., Undersander, E., & Diamos, G. (2017). Block-sparse recurrent neural networks. arXiv preprint arXiv:1711.02782.

Pan, Y., Xu, J., Wang, M., Ye, J., Wang, F., Bai, K., et al. (2019). Compressing recurrent neural networks with tensor ring for action recognition. In *The thirty-third aaai conference on artificial intelligence, aaai 2019, honolulu, hawaii, usa, january 27 - february 1, 2019* (pp. 4683–4690).

Prabhavalkar, R., Alsharif, O., Bruguier, A., & McGraw, I. (2016). On the compression of recurrent neural networks with an application to lvcsr acoustic modeling for embedded speech recognition. In *ICASSP 2016, Shanghai, China, March 20-25, 2016* (pp. 5970–5974). http://dx.doi.org/10.1109/ICASSP.2016.7472823.

Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. arXiv preprint arXiv:1606.05250.

Rezende, D. J., Mohamed, S., & Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. arXiv preprint arXiv:1401.4082.

Sagara, T., & Hagiwara, M. (2014). Natural language neural network and its application to question-answering system. *Neurocomputing*, *142*, 201–208.

Seo, M., Kembhavi, A., Farhadi, A., & Hajishirzi, H. (2016). Bidirectional attention flow for machine comprehension. arXiv preprint arXiv:1611.01603.

Srinivas, S., Subramanya, A., & Babu, R. V. (2017). Training sparse neural networks. In *CVPR workshops 2017, Honolulu, HI, USA, July 21-26, 2017* (pp. 455–462). http://dx.doi.org/10.1109/CVPRW.2017.61.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *NIPS* (pp. 3104–3112).

Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2016). Show and tell: Lessons learned from the 2015 mscoco image captioning challenge. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(4), 652–663.

Wang, Z., Lin, J., & Wang, Z. (2018). Hardware-oriented compression of long short-term memory for efficient inference. *IEEE Signal Processing Letters*, *25*(7), 984–988. http://dx.doi.org/10.1109/LSP.2018.2834872.

Wang, P., Xie, X., Deng, L., Li, G., Wang, D., & Xie, Y. (2018). HitNet: Hybrid Ternary Recurrent Neural Network. In *Advances in neural information processing systems 31: annual conference on neural information processing systems 2018, NeurIPS 2018, 3-8 December 2018, MontrÉAl, Canada* (pp. 602–612).

Wen, W., He, Y., Rajbhandari, S., Zhang, M., Wang, W., Liu, F., et al. (2017). Learning intrinsic sparse structures within long short-term memory. arXiv preprint arXiv:1709.05027.

Wen, W., Wu, C., Wang, Y., Chen, Y., & Li, H. (2016). Learning structured sparsity in deep neural networks. In *NIPS 2016, December 5-10, 2016, Barcelona, Spain* (pp. 2074–2082).

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*(3–4), 229–256.

Xu, Z., Zhe, S., Qi, Y., & Yu, P. (2016). Association discovery and diagnosis of Alzheimer's disease with Bayesian multiview learning. *Journal of Artificial Intelligence Research*, *56*, 247–268.

Yang, Z., Chen, W., Wang, F., & Xu, B. (2018). Generative adversarial training for neural machine translation. *Neurocomputing*, *321*, 146–155.

Ye, J., Wang, L., Li, G., Chen, D., Zhe, S., Chu, X., et al. (2018). Learning compact recurrent neural networks with block-term tensor decomposition. In *2018 IEEE conference on computer vision and pattern recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018* (pp. 9378–9387).

Yin, M., & Zhou, M. (2019). ARM: augment-REINFORCE-merge gradient for stochastic binary networks. *ICLR*.

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. arXiv preprint arXiv:1409.2329.

Zhao, Y., Wang, L., Wu, W., Bosilca, G., Vuduc, R. W., Ye, J., et al. (2017). Efficient Communications in Training Large Scale Neural Networks. In *Proceedings of the on thematic workshops of ACM Multimedia 2017, Mountain View, CA, USA, October 23 - 27, 2017* (pp. 110–116).

Zhe, S., Xu, Z., Qi, Y., & Yu, P. (2015). Sparse Bayesian Multiview Learning for Simultaneous Association Discovery and Diagnosis of Alzheimer's Disease. In *Proceedings of the twenty-ninth AAAI conference on artificial intelligence, January 25-30, 2015, Austin, Texas, USA* (pp. 1966–1972).

Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., et al. (2018). Discrimination-aware channel pruning for deep neural networks. In *Advances in neural information processing systems 31: annual conference on neural information processing systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada* (pp. 883–894).

Zilly, J. G., Srivastava, R. K., Koutník, J., & Schmidhuber, J. (2017). Recurrent highway networks. In *Proceedings of the 34th international conference on machine learning-volume 70* (pp. 4189–4198). JMLR. org.