Learning Sparse Hidden States in Long Short-Term Memory*

Niange Yu[1], Cornelius Weber[2], and Xiaolin Hu[1]**

[1] State Key Laboratory of Intelligent Technology and Systems, Beijing National Research Center for Information Science and Technology, Department of Computer Sicence and Technology, Tsinghua University, Beijing 100084, China
niangeyu@gmail.com, xlhu@tsinghua.edu.cn
[2] Department of Informatics, University of Hamburg, Hamburg, Germany
weber@informatik.uni-hamburg.de

Abstract. Long Short-Term Memory (LSTM) is a powerful recurrent neural network architecture that is successfully used in many sequence modeling applications. Inside an LSTM unit, a vector called "memory cell" is used to memorize the history. Another important vector, which works along with the memory cell, represents hidden states and is used to make a prediction at a specific step. Memory cells record the entire history, while the hidden states at a specific time step in general need to attend only to very limited information thereof. Therefore, there exists an imbalance between the huge information carried by a memory cell and the small amount of information requested by the hidden states at a specific step. We propose to explicitly impose sparsity on the hidden states to adapt them to the required information. Extensive experiments show that sparsity reduces the computational complexity and improves the performance of LSTM networks [3].

Keywords: Recurrent neural network (RNN) · Long short-term memory (LSTM) · Language modeling · Image captioning · Network acceleration

1 Introduction

Recurrent neural networks (RNNs) are widely used in sequence modeling problems. Generally, a recurrent unit takes actions along a given sequence of inputs and, step-by-step, produces a sequence of outputs. The current actions attend to the current inputs or predictions, while a memory records history. Modern recurrent units like LSTM [8] or Gated Recurrent Unit (GRU) [5] use gate mechanisms to control the communication between the memory and the externals. By selectively retrieving messages from the inputs, and writing and erasing some contents in

* We acknowledge support by NSFC (61621136008) and German Research Foundation (DFG) under project CML (TRR 169).
** Corresponding author.
[3] The source code is available at https://github.com/feiyuhug/SHS_LSTM/tree/master

the memory with gates, those recurrent units can learn to process sequences with complex interdependencies. Their applications span a wide range such as language modeling [15], machine translation [20], speech recognition [6] and image captioning [21].

The memory in a recurrent unit is in charge of the whole history of the sequence, but taking an action (such as prediction) at one step should only require to attend to a specific part of the memory. In an LSTM unit, a hidden state h is learned to carry the message from the memory c and used to make the prediction. Since h should learn to get rid of a large part of the information in c that is unrelated to the current step, a general priority can be set that h should be "lighter" than c. However, the hidden state h and memory c are two vectors that share the same dimension. "Lighter" in this context has two meanings: one is that h should carry less information than c, reflecting the focus of the current step, which may not require all information that is kept in memory. The other is that the computational complexity with h should be lower, since h will be passed on to further processing steps. We achieve both targets by imposing sparsity on h.

Sparse coding has been subject to many modelling studies, and physiological recordings from sensory neurons indicate that it is employed in several modalities in the sensory cortex [17]. Forming part of efficient coding [4], sparse distributions have also been found in the underlying causes of natural stimuli [2]. Hence, imposing sparse coding on a network state representation may bias the network to encode compact representations of the causes of sensory stimuli.

Let us refer to the dimensions of the memory cell as "channels". In vanilla LSTM units, the output gates modulate the channels softly by multiplying with a value in $(0, 1)$, which does not affect the dimensionality of h. We zero out many output gates that are lower than a threshold, as to close many channels in h. In this way, sparsity is imposed on h. While the hidden state h is central in the computational graph of the LSTM unit, the computational complexity of a time step is reduced in proportion to the sparsity ratio of h.

We conduct extensive experiments on language modeling and image captioning tasks. By adjusting the sparsification strength, an optimal sparsity ratio on h can always be found that improves both the prediction accuracy and inference efficiency over the baseline. Unlike weight regularization, the proposed hidden state sparsification method does not reduce the number of adaptable network parameters θ, i.e. weights. We will show in Section 4.1 that hidden state sparsification can be combined with weight sparsification methods [23,16] for additional merit.

2 Related Works

Many works have been proposed to accelerate neural networks. Dynamic networks with conditional computations inside reduce the computational cost by exploring certain computation paths for each input [14,19,3]. Lin et al. propose to selectively prune some channels in a feed-forward convolutional neural network (CNN)

according to the current inputs. They use an external network to generate pruning actions that are learned by a reinforcement learning algorithm [11]. For RNN acceleration, Jernite et al. propose to update only a fraction of hidden states to reduce computational cost at each step. An external module is learned to determine which hidden states to update [9]. This method has only been successfully applied to the vanilla RNN and GRU. In our method, a subset of channels in the hidden state is selected to open for each input, and this selection process is modulated by the output gate of an LSTM unit, which incurs no extra computational cost.

Another line of works proposed sparsity on weight matrices to reduce computational cost [7,22,16], which is different from the proposed hidden states sparsification method.

3 Methods

The computation for an LSTM unit can be formulated as:

$$\text{LSTM} : x_t, h_{t-1}, c_{t-1} \rightarrow h_t, c_t \tag{1}$$

where subscripts denote time steps. At time step t, it takes an input x_t and updates the hidden state ($h_{t-1} \rightarrow h_t$) and memory cell ($c_{t-1} \rightarrow c_t$). The computation inside is with four gates: "input", "forget", "output" and "input modulation" (denoted as i_t, f_t, o_t, g_t respectively),

$$i_t = \text{sigm}(W_{ix}x_t + W_{ih}h_{t-1} + b_i), \tag{2}$$
$$f_t = \text{sigm}(W_{fx}x_t + W_{fh}h_{t-1} + b_f), \tag{3}$$
$$o_t = \text{sigm}(W_{ox}x_t + W_{oh}h_{t-1} + b_o), \tag{4}$$
$$g_t = \tanh(W_{gx}x_t + W_{gh}h_{t-1} + b_g), \tag{5}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t, \tag{6}$$
$$h_t = o_t \odot \tanh(c_t). \tag{7}$$

The memory cell $c_t \in \mathbb{R}^n$ is a vector that stores the long-term memory, and $h_t \in \mathbb{R}^n$ is the hidden state that is usually used to predict the output y_t (see Eq. (16)). "sigm" is the sigmoid function, "tanh" is the hyperbolic tangent function and $\odot$ denotes element-wise multiplication.

Multiple LSTM units can be stacked into a hierarchical network, in which the output of each unit is fed into the next upper unit. The original input x_t is fed into the bottom-most unit, and the final output is taken from the top-most unit. Let superscripts denote layers, the computation inside the LSTM unit in the l-th layer at time step t ($1 \leq l \leq L, 1 \leq t \leq T$) can be formulated as [24], where T

denotes the unrolled time steps.

$$\text{LSTM} : h_t^{l-1}, h_{t-1}^l, c_{t-1}^l \rightarrow h_t^l, c_t^l, \tag{8}$$

$$\begin{pmatrix} i_t^l \\ f_t^l \\ o_t^l \\ g_t^l \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} T_{2n,4n} \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}, \tag{9}$$

$$c_t^l = f_t^l \odot c_{t-1}^l + i_t^l \odot g_t^l, \tag{10}$$

$$h_t^l = o_t^l \odot \tanh(c_t^l). \tag{11}$$

The input is h_t^{l-1} which is from the lower layer for $l > 1$, or x_t for $l = 1$. The computations with four gates are fused in (9), where $T_{2n,4n} : \mathbb{R}^{2n} \rightarrow \mathbb{R}^{4n}$ is an affine transform ($Wx + b$) which contributes the most to the computational costs.

Eqn. (10) is the memory update function. Note that there is no nonlinear or parameterized transformation at the memory update. Some parts of the memory may be kept over many steps, which add LSTM model long-term dependencies. However, in many scenarios, such as in sequence generation tasks, the LSTM needs to make a prediction at every time step. The prediction targets at different steps are usually different, which may be in conflict with the shared memory. Since the output gate o_t^l connects the memory and the prediction layer, our idea is that it should learn to attend to different parts of the memory for different predictions, so to resolve the conflicts.

We empirically find that the output gate values learned implicitly in LSTM are widely distributed ("baseline" shown in the left column of Fig. 2), which may not produce distinct attention over the memory for different predictions.

In order to impose sparsity, we introduce an L1-norm loss term over the output gate at layer l as

$$\mathcal{L}^S(o^l(x, \theta)) = \sum_{1 \leq b \leq B} \sum_{1 \leq t \leq T} |o_t^l(x^{(b)})|. \tag{12}$$

Unlike weight regularization, the output gate sparsification is data-dependent. Sparsification refers to a training batch $x^{(b)} (1 \leq b \leq B)$, where B is the batch size. $o_t^l(x^{(b)})$ is computed in Eq. (9) and T denotes the unrolled time steps. The sparsity loss term is added to the classification loss term $\mathcal{L}^{XE}(\theta)$ for the adaptation of model parameters θ by supervised learning, which will be described in the following sections. Denoting the model weights as w, the updating function for w via stochastic gradient decent (SGD) is

$$w \leftarrow w - \Big(\eta \frac{\partial \mathcal{L}^{XE}(\theta)}{\partial w} + \sum_{1 \leq l \leq L} \lambda_l \frac{\partial \mathcal{L}^S(o^l(x, \theta))}{\partial w}\Big), \tag{13}$$

where η is the learning rate and λ_l are the coefficients that control the strength of the sparsity term at each LSTM layer ($1 \leq l \leq L$).

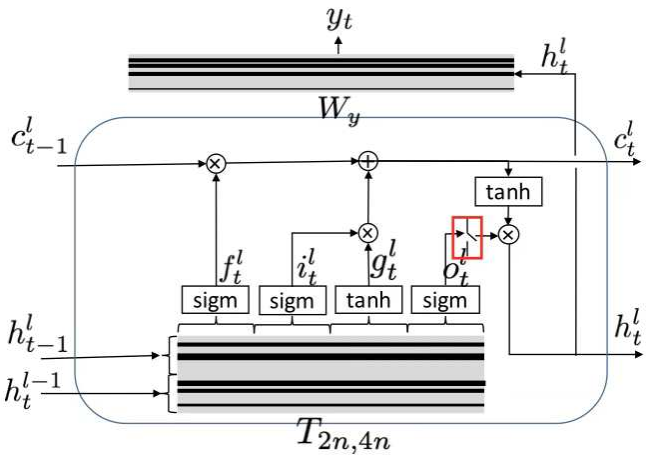

Fig. 1. Sparse LSTM. Given sparse hidden state vectors (h_{t-1}^l, h_t^{l-1}) and h_t^l, only a few rows of the respective transformation matrices $T_{2n,4n}$ and W_y are used.

By increasing λ_l, more output gate channels would be driven to zero. We empirically found that introducing a threshold function ψ after the gates could obtain better performance:

$$\widetilde{o}_t^l(x_t^{(b)}) = \psi(o_t^l(x_t^{(b)}), \xi) = \begin{cases} o_t^l(x_t^{(b)}), & o_t^l(x_t^{(b)}) > \xi \\ 0 \quad , & o_t^l(x_t^{(b)}) <= \xi \end{cases}, \tag{14}$$

where $\xi \geq 0$. The threshold function is appended after the output gates, both in training and testing. As a consequence, the hidden state function defined in Eq. (11) is reformulated as:

$$h_t^l = \widetilde{o}_t^l \odot \tanh(c_t^l). \tag{15}$$

During training, the sparsity of the output gates would increase gradually and become stable after several epochs. The final sparsity that can be reached is determined by $\lambda_l (1 \leq l \leq L)$ and ξ. Higher values for λ_l and ξ both increase the sparsity.

Taking the unit in the top-layer of the LSTM network as an example, the computational graph of the LSTM unit with the sparse hidden state is shown in Fig. 1. Compared with a classical LSTM unit, a threshold layer is appended behind the output gate (shown inside the red box). The major computational burden in the LSTM unit is two affine transformations: one is defined in Eq. (9) and the other is the output layer

$$y_t = W_y h_t^l. \tag{16}$$

The input vectors of both affine transformations are from the sparse hidden states. The dimensions set to zero in the input vector would zero out the corresponding

rows in the weight matrices ($T_{2n,4n}$ and W_y) as shown in the two shaded blocks in Fig. 1, where gray indicates zero rows and black indicates nonzero rows. Only nonzero rows in the weight matrices take part in the affine transformations Eq. (9) and Eq. (16). The computational complexity of the two affine transformations is reduced in proportion to the sparsity ratio of h_t^l and (h_{t-1}^l, h_t^{l-1}) respectively.

4 Experiments

4.1 Language Modeling

The language modeling experiments are conducted on the Penn Treebank dataset [13]. We start by implementing the proposed method in a two-layer LSTM network proposed in [24]. The dimensions of both the LSTM memory and the word embedding in the input layer are 1500. Given the next word y_t^* as target output, the cross-entropy loss that we minimize as part of Eq. (13) is:

$$\mathcal{L}^{XE}(\theta) = -\sum_{t=1}^{T} \log(p_t^{(\theta)}(y_t^*|(y_{1:t-1}^*))), \tag{17}$$

We use the PyTorch implementation of the publicly available baseline model[4]. The distribution of the output gates o in the baseline model (i.e. $\xi = 0, \lambda = 0$) is shown in Fig. 2. At the training, we set $\xi = 0.1$, and use the output gate sparsification to squeeze more gates under ξ as to increase sparsity. The distributions of the output gates with $\lambda_1/\lambda_2 = 1e^{-6}$ and $\lambda_1/\lambda_2 = 1.6e^{-6}/4.8e^{-6}$ are shown in Fig. 2. The distribution can be further pushed close to zero if λ continues to increase.

Table 1. Learning sparse output gates from scratch.

Method	$\xi, \lambda_1/\lambda_2(10^{-6})$	Test Perplexity	Output Width (1st, 2nd) LSTM	Mult-add* Reduction	Time(ms)/ Speed-Up
baseline	$(0.0, 0.0/0.0)$	77.88	(1500, 1500)	1.00×	81.7/1×
Ours	$(0.1, 0.4/0.4)$	77.14	(720, 953)	1.56×	53.3/1.53×
	$(0.1, 1.0/1.0)$	**76.97**	(436, 797)	1.89×	44.6/1.83×
	$(0.1, 1.6/1.6)$	77.05	(344, 706)	2.09×	38.2/2.14×
	$(0.1, 1.6/4.8)$	77.85	**(261, 344)**	**2.75**×	**29.9/2.73**×

*The reduction of multiplication-add operations in matrix multiplications.

With ξ fixed to 0.1, we search for the optimal λ. The results are shown in Table 1. All the models are trained from scratch in the same setting with the baseline. "Test Perplexity" denotes the perplexity on the test set; lower is better. We find that $\lambda_1/\lambda_2 = 1e^{-6}$ gets the best perplexity (the improvement is -0.91 compared to the baseline), with λ_1/λ_2 higher or lower than this value would

[4] `https://github.com/pytorch/examples/tree/master/word_language_model`

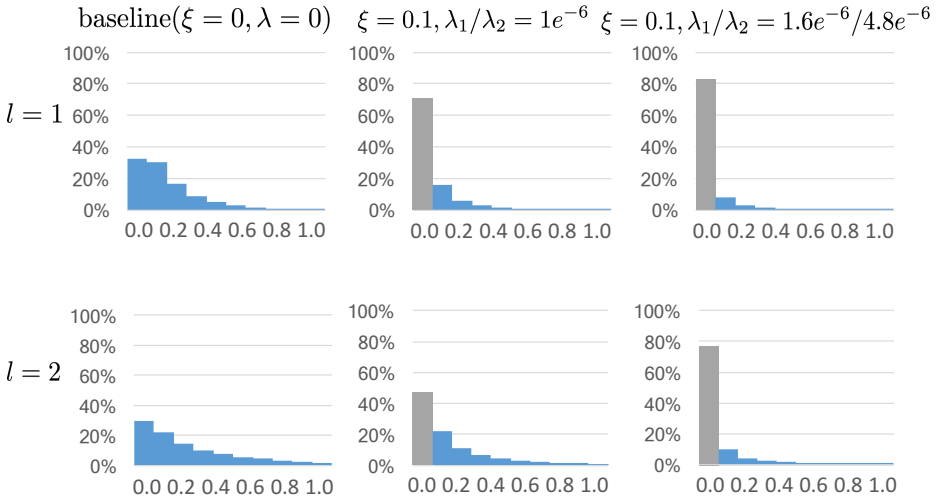

Fig. 2. Distributions of the output gate values o_t^l as defined in Eq. (9) for three pre-trained models. $l = 1$ and $l = 2$ stands for the first and second layers respectively. The values are calculated with a batch size of 50 and with 35 unrolled steps ($t = 1, ..., 35$) on the test set.

lead to inferior results. "Output width" denotes the remaining (non-zero) output channels at each LSTM layer. Notably, models with sparse output largely reduce the computational complexity. For $\lambda_1/\lambda_2 = 1.6e^{-6}/4.8e^{-6}$, the output channel dimension is reduced to 17%(261/1500) in the first layer and to 23%(344/1500) in the second layer, thereby totally reducing Multi-add operations by a factor of 2.75. We measure the inference time on an Intel® Core™ i7-6850K @ 3.60GHz CPU processor with OpenBLAS library for matrix-multiplication operations. Testing the time by unrolling the LSTM unit for 100 steps, the maximum speed-up is 2.73×.

Next, we evaluate whether our state sparsification method combined with weight sparsification yields additional benefit. Intrinsic Sparse Structures (ISS) [22] is a recently proposed weight-sparsifying LSTM acceleration method, which yields an impressive performance on the Penn Treebank dataset. Based on group Lasso regularization, it reduces the dimension of the memory, the hidden state and the related weight matrices by structured weight pruning. Our method can be embedded into ISS easily by adjusting ξ to control the output width. The experimental results are listed in Table 2. The third column lists the memory size and the output width of the learned model. ISS+Ours ($\xi = 0.20$) achieves the highest speed-up ratio of 6.28× with slightly lower test perplexity (78.37) than the baseline(ISS) (78.57) or ISS alone (78.65).

Table 2. ISS with sparse hidden states

Method	Test Perplexity	Memory/Output Width (1st, 2nd) LSTM	Mult-add reduction	Time(ms)/ Speed-Up
baseline*	78.57	(1500, 1500)/(1500, 1500)	1.00×	81.7/1.00×
ISS	78.65	(373, 315)/(373, 315)	7.48×	14.1/5.79×
ISS+Ours ($\xi = 0.10$)	**76.27**	(379, 536)/(229, 445)	5.99×	17.2/4.75×
ISS+Ours ($\xi = 0.20$)	78.37	(194, 542)/(63, 375)	**8.63**×	**13.0/6.28**×

*The baseline is different from the baseline in Tab 1.

4.2 Image Captioning

We experiment our methods with the image captioning model *Google NIC* [21] as a baseline. Google NIC is an encoder-decoder framework which uses a CNN as an encoder to extract features from images and then uses an LSTM decoder to generate sentences. The CNN image features are input to the LSTM at the beginning step. The LSTM network generates the sentence word by word with one word per step and stops when the "stop token" is predicted or when it reaches the maximum length.

Formally, the inputs for the LSTM are

$$x_1 = \text{CNN}(I), \tag{18}$$

$$x_t = W_e y_{t-1}, (t \in 2, 3, ..., T), \tag{19}$$

where I denotes the raw image and x_1 is input once at $t = 1$. y_{t-1} is the one-hot representation of the input word (from the target sentence at training or the previous prediction at inference). $W_e \in \mathcal{R}^{V \times E}$ is the word embedding matrix for a vocabulary of size V. Words are predicted using Softmax:

$$p_t = \text{Softmax}(h_t). \tag{20}$$

The predicted word y_t is sampled from p_t. h_t is the hidden state defined in Eq. (7). Given a ground-truth sentence $y^*_{1:T}$, the cross-entropy loss that we minimize as part of Eq. (13) is:

$$\mathcal{L}^{XE}(\theta) = -\sum_{t=1}^{T} \log(p_t^{(\theta)}(y_t^*|(\text{CNN}(I), y^*_{1:t-1}))). \tag{21}$$

The model is trained and evaluated on MSCOCO [12], which is a widely used benchmark for image captioning. MSCOCO has 123,000 images with five captions annotated for each image. We use the publicly available split [10], where the validation set and test set each has 5,000 images. The remaining 113,000 images are used for training.

Table 3 shows the results. All listed models use ResNet101 or ResNet152 as an encoder. Compared with state-of-the-art methods [18,1], our Google NIC implementations (NIC 512/NIC 1024/NIC 2048) achieve new best results on

BLEU4 and METEOR scores, on par with ROUGE-L, but not winning on CIDEr. Note that while the other works involve dedicated attention networks with learnt attention weights, our model uses only simple sparsification, which by focusing activations has an attention-like effect.

Table 3. The results of Google NIC with different memory size and ξ on the MSCOCO Karpathy test split. B4, M, R-L and C stand for BLEU4, METEOR, ROUGE-L and CIDEr respectively. Time is in ms.

Model	Time	B4	M	R-L	C
[18]	-	31.3	26.0	54.3	101.3
[1]	-	33.4	26.1	**54.4**	**105.4**
NIC 512	239	**33.5**	25.8	**54.4**	101.7
NIC 1024	503	32.6	26.0	54.1	101.8
NIC 2048	1070	32.3	26.1	53.9	102.1
NIC 2048 ($\xi = 0.3$)	721	33.3	**26.4**	**54.4**	103.0
NIC 2048 ($\xi = 0.4$)	616	32.9	**26.4**	**54.4**	103.3
NIC 2048 ($\xi = 0.5$)	436	32.7	26.0	53.9	102.9

Finally, we adjust the parameter ξ directly to close more channels in the output gates without using the output gate sparsification of Eq. (12). The results with $\xi = 0.3, 0.4, 0.5$ on the image caption task are listed in Table 3. It is found that the optimal setting for ξ is 0.4 which improves the CIDEr from 102.1 to 103.3 while speeding up by a factor of $1.74 \times (1070ms/616ms)$.

5 Conclusion

We explore novel sparse LSTM networks in which the activations of the output gates are sparsified. This reduces the amount of information that is passed on for further processing, while not impacting on the memory of the LSTM cells. Experiments were conducted on three tasks including language modeling and image captioning. The proposed method obtains better performance on all tasks at lower computational costs. On the Penn Treebank language modeling experiments, we found that sparse hidden states can work together with weight sparsifying regularization methods to achieve better results than when using them individually.

References

1. Anderson, P., He, X., Buehler, C., Teney, D., Johnson, M., Gould, S., Zhang, L.: Bottom-up and top-down attention for image captioning and visual question answering. In: Computer Vision and Pattern Recognition (CVPR), IEEE Conference on (2018)

2. Barlow, H.: What is the computational goal of the neocortex? In: Koch, C., Davis, J. (eds.) Large-scale neuronal theories of the brain. pp. 1–22. Cambridge, MA, US: The MIT Press (1994)
3. Campos, V., Jou, B., Giró-i Nieto, X., Torres, J., Chang, S.F.: Skip RNN: Learning to skip state updates in recurrent neural networks. arXiv preprint arXiv:1708.06834 (2017)
4. Chalk, M., Marre, O., Tkačik, G.: Toward a unified theory of efficient, predictive, and sparse coding. Proceedings of the National Academy of Sciences **115**(1), 186–191 (2018)
5. Cho, K., van Merriënboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: Encoder–decoder approaches. In: Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation. pp. 103–111 (2014)
6. Graves, A., Mohamed, A.r., Hinton, G.: Speech recognition with deep recurrent neural networks. In: Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. pp. 6645–6649. IEEE (2013)
7. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems. pp. 1135–1143 (2015)
8. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**(8), 1735–1780 (1997)
9. Jernite, Y., Grave, E., Joulin, A., Mikolov, T.: Variable computation in recurrent neural networks. arXiv preprint arXiv:1611.06188 (2016)
10. Karpathy, A., Fei-Fei, L.: Deep visual-semantic alignments for generating image descriptions. In: Computer Vision and Pattern Recognition (CVPR), IEEE Conference on. pp. 3128–3137 (2015)
11. Lin, J., Rao, Y., Lu, J., Zhou, J.: Runtime neural pruning. In: Advances in Neural Information Processing Systems. pp. 2178–2188 (2017)
12. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft COCO: Common objects in context. In: European Conference on Computer Vision. pp. 740–755. Springer (2014)
13. Marcus, M.P., Marcinkiewicz, M.A., Santorini, B.: Building a large annotated corpus of English: The Penn Treebank. Computational Linguistics **19**(2), 313–330 (1993)
14. McGill, M., Perona, P.: Deciding how to decide: Dynamic routing in artificial neural networks. arXiv preprint arXiv:1703.06217 (2017)
15. Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., Khudanpur, S.: Recurrent neural network based language model. In: Eleventh Annual Conference of the International Speech Communication Association (2010)
16. Narang, S., Elsen, E., Diamos, G., Sengupta, S.: Exploring sparsity in recurrent neural networks. arXiv preprint arXiv:1704.05119 (2017)
17. Olshausen, B.A., Field, D.J.: Sparse coding of sensory inputs. Current Opinion in Neurobiology **14**(4), 481–487 (2004)
18. Rennie, S.J., Marcheret, E., Mroueh, Y., Ross, J., Goel, V.: Self-critical sequence training for image captioning. In: Computer Vision and Pattern Recognition (CVPR), IEEE Conference on. pp. 7008–7024 (2017)
19. Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., Dean, J.: Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. arXiv preprint arXiv:1701.06538 (2017)

Learning Sparse Hidden States in Long Short-Term Memory 11

20. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems. pp. 3104–3112 (2014)
21. Vinyals, O., Toshev, A., Bengio, S., Erhan, D.: Show and tell: A neural image caption generator. In: Computer Vision and Pattern Recognition (CVPR), IEEE Conference on. pp. 3156–3164. IEEE (2015)
22. Wen, W., He, Y., Rajbhandari, S., Wang, W., Liu, F., Hu, B., Chen, Y., Li, H.: Learning intrinsic sparse structures within long short-term memory. arXiv preprint arXiv:1709.05027 (2017)
23. Yu, N., Qiu, S., Hu, X., Li, J.: Accelerating convolutional neural networks by group-wise 2D-filter pruning. In: Neural Networks (IJCNN), 2017 International Joint Conference on. pp. 2502–2509. IEEE (2017)
24. Zaremba, W., Sutskever, I., Vinyals, O.: Recurrent neural network regularization. arXiv preprint arXiv:1409.2329 (2014)