Learning Block-Sparse Neural Networks

Huanru Henry Mao*, Shuyang Li*, Yiting Ethan Li*
Computer Science and Engineering
University of California, San Diego
9500 Gilman Dr, La Jolla, CA 92093

***Abstract*—Over-parameterized neural networks tend to have redundant parameters [1] and can be pruned and compressed without loss in model performance. In this work, we experiment with different methods to achieve sparsity constraints on over-parameterized LSTM recurrent networks on the language modeling task. We divide the weight matrices of the neural network into pre-defined blocks and apply block-sparsity constraints to the optimization objective. We demonstrate that pruning a neural network while training when coupled with a block lasso constraint achieves the best performance.**

I. Introduction

Deep neural networks (DNNs) are a class of powerful function approximators that have seen success across a broad range of learning tasks in computer vision and natural language processing. Much of the success from deep learning can be attributed to allocating a large number of parameters to learn a particular task [2]. It is conjectured that increasing the parameters of a network can lead to easier optimization as there are more saddle points in high dimensional space, and thus lead to better performance on a variety of tasks. However, large DNNs are typically over-paramaterized and, after training, have many redundant parameters [1]. Work in neural pruning after training has demonstrated that a majority of parameters can be removed with little impact on a model's performance. There are also many work that demonstrate DNNs can be compressed after training without loss in accuracy [3], [4], [5], [6], [7], [8]. This phenomenon suggests that, although over-parameterization eases optimization, there is an innate inefficiency in doing so.

One potential direction to improve this inefficiency is to investigate sparser neural networks. A sparse neural network has many weights that are exactly zero. Sparsity, when constructed in a structured manner, also has benefits during training and inference by significantly reducing the computation time for running a neural network [9]. One prime example of structured sparsity is block sparsity [10], which treats blocks (contiguous sub-matrices, as in figure 1) within the weight matrix in a neural network as a collective group. These groups can be set to exact zeros. This sparse connectivity graph can be represented as a matrix with "blocks" of zero entries, which is core to the idea of block-sparse DNNs [10]. Block sparsity enables efficient computation as GPU kernels can be designed to skip the computation of these blocks [11].

Creating sparser neural networks is also of interest due to its connection with biological neural networks. While typical DNNs represent information flow as a densely-connected graph, human neural connections form a sparse pattern exhibiting small-world connectivity [12]. There is evidence that during human brain development and adolescence, a large amount of synapses are eliminated similar to the process of pruning [13], [14]. This pruning process occurs while we learn and is not necessarily done in a post-training manner similar to most methods in DNN pruning.

In this paper, we explore learning block-sparse neural network by defining block sparsity as part of the constraint of the training objective function. While Gray, et. al [10] defines a random block-sparse matrix and fixes these blocks to zero upon network creation, we propose to learn an optimal set of zero blocks over time (which may be more biologically plausible), developing a sparse network topology as we train. We experiment with a block-sparse variant of the Long Short-Term Memory (LSTM) model. We aim to significantly reduce training time for neural networks by pruning while training. This enables us to reduce the model size the more we train, similar to a greedy architectural search. Our experiments demonstrate that the block lasso objective is capable of pruning a model while achieving similar performance to its dense counterpart when coupled with gradual pruning during the training process.

II. Related Work

Coefficient group sparsity has a long history in statistical learning problems, stemming from Group Lasso to regularize groups of factors in regression tasks [15]. Here, as in [10], we take groups of factors to be 2-dimensional sections of connection weights in a feed-forward neural network. We seek to induce contiguous sparse regions for significant computational benefits in both training and inference, while maintaining generalization power.

We take inspiration from similar work in model compression via compressing basic structures within recurrent long short-term memory (LSTM) units [16]. There, the authors use group lasso to squeeze weights in components of each LSTM unit near zero, and then threshold them to zero-out entire components. We adapt this approach but use sparse group lasso to force both intra- and inter-component sparsity in each layer of our feed-forward network.

Group lasso has also been applied to convolutional neural networks (CNNs) to induce sparsity at the filter (feature) level [17]. Lym, et al. [17] take their groups to be channels (convolutional features), while we seek to sparsify contiguous blocks of weights. Block-level sparsity in recurrent neural

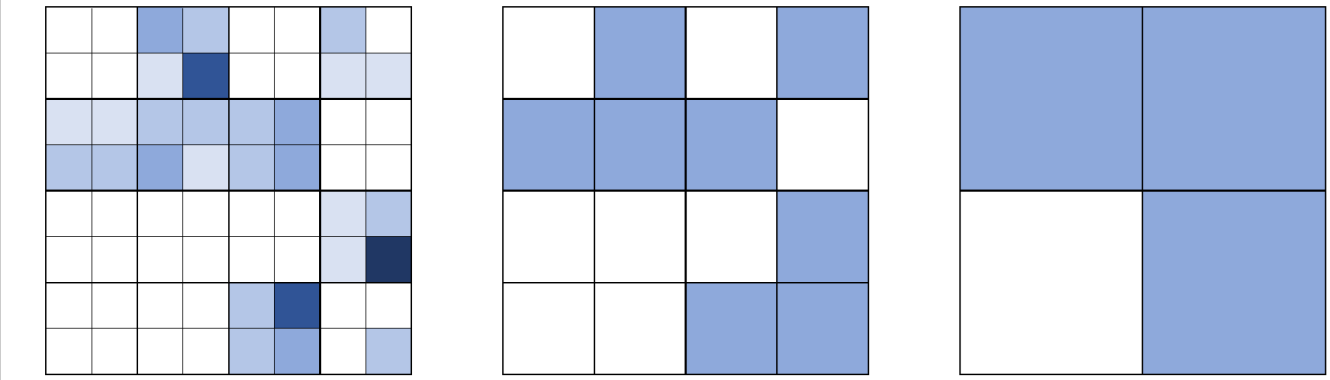

Fig. 1. Block sparsity, with white squares representing zero weights. From left to right: original weights, 2×2 blocks, 4×4 blocks

networks (RNNs) has been explored in [18], which focuses on adaptive pruning rates through the training process.

Another proposed pruning approach to sparsifying connection weights comes from targeted dropout [19], which applies the idea of dropout regularization [20] - originally a method to prevent overfitting - to improve post-hoc pruning (e.g pruning a model after training). Instead of randomly sampling dropout neurons from the entire weight set, the network estimates a set of neurons of low significance and randomly drops from that set during training.

III. Problem Statement

Given a DNN with weights W and n hidden layers (with input weights W_0), we wish to optimize some task-specific objective function $f(\cdot)$ given a dataset of size N, subject to block sparsity constraints.

Let us evenly divide the set of all weights W into contiguous square blocks of size $d \times d$, with a total of $b = \frac{1}{d^2} \sum_{h=1}^{n} |W^h|$ blocks where W^h is the connection weight matrix from layer $h-1$ to h. Let $\| \cdot \|_0$ be the L_0 norm of a tensor, indicating the number of nonzero elements.

We constrain the network to contain no more than k blocks with any nonzero weight:

$$\begin{aligned} \min_{W} \quad & f(W) \\ \text{s. t.} \quad & \|\hat{W}\|_0 \leq k \end{aligned}$$

where $\hat{W}$ consists of the L_0 norms of each block W_i in the network:

$$\hat{W} = \{\|W_i\|_0, \quad i = 1, \dots b\}$$

The L_0 norm is discrete and non-differentiable, with 2^b possible states across all b blocks. Our optimization problem is thus intractable, and we must reasonably relax our problem to allow for continuous optimization as in [6]. We propose a simple relaxation taking advantage of the fact that the L_1 norm is a good approximation for the L_0 norm [21] (e.g the L_1 unit sphere is the convex hull of the L_0 unit sphere). We thus relax our primal problem:

$$\begin{aligned} \min_{W} \quad & f(W) \\ \text{s. t.} \quad & \|\hat{W}\|_1 \leq k \\ & \hat{W} = \{\|W_i\|_2, \quad i = 1, \dots b\} \end{aligned}$$

Here we note that the internal L_0 norm exists only to track if a block is nonzero. Thus we are able to replace it with the block-wise L_2 norm with no penalty, as the L_2 Frobenius norm is only 0 for the 0 matrix. We are able to thus formulate the constraint as a Group Lasso problem:

$$\begin{aligned} \min_{W} \quad & f(W) \\ \text{s. t.} \quad & \sum_{i=1}^{b} \|W_i\|_2 \leq k \end{aligned}$$

with associated Lagrangian

$$\mathcal{L}(W, \lambda) = f(x, W) + \lambda \cdot (\sum_{i=1}^{b} \|W_i\|_2 - k)$$

and dual problem formulation:

$$\begin{aligned} \max_{\lambda} \quad & g(\lambda) = \inf_{W} \mathcal{L}(W, \lambda) \\ & = \inf_{W} \{f(W) + \lambda \sum_{i=1}^{b} \|W_i\|_2\} - k\lambda \\ \text{s. t.} \quad & \lambda \geq 0 \end{aligned}$$

We see that the dual objective is affine in λ, and the constraint is also affine. We are thus maximizing a concave function subject to a convex constraint, which is equivalent to minimizing its negative, making this dual problem convex. We are thus able to approximate a solution to the original problem via regularized empirical risk minimization on the parameters of a neural network $f(\cdot, W)$:

$$W^* = \arg\min_{W} \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i, W)) + \lambda(\sum_{i=1}^{b} \|W_i\|_2 - k)$$

where $L(\cdot)$ represents our cross-entropy objective function, and λ is the regularization weight. This is the final training objective of our model, which we call the *block lasso objective*.

IV. Sparse Computational Efficiency

Block sparsity can grant significant computational benefits given optimized GPU Kernels. Take the case of modifying an $n \times 1$ input vector with an $m \times n$ weight matrix: $Wx = b \in \mathcal{R}^{m \times 1}$. The sparsity ratio of the weight matrix is $\rho = \frac{K}{mn}$, where K represents the number of total zero weights, and K_i represents the number of zero weights in row i.

In standard dense matrix multiplication, each element of the resulting vector can be calculated as: $b_i = \sum_{j=0}^{n} W_{ij} x_i$. This gives us $\sum_{i}^{m} n = mn$ total operations for dense matrix multiplication.

In the case of a sparse matrix, any zero weights in the weight matrix do not contribute to the element sum: $b_i = \sum_{j, W_{ij} \neq 0}^{n} W_{ij} a_i$. giving a total number of multiply-add operations of: $\sum_{i}^{m} (n - K_i) = mn - K$. We see here that the computational savings is equal to the number of sparse weights. These derivations hold whether we calculate a matrix with $m \times n$ weights, or $m \times n$ square blocks of weights.

However, unstructured sparsity cannot provide such theoretical savings, as we would need to check if a weight was sparse before deciding whether to include it in our multiply-sum. With structured group sparsity, as in contiguous blocks/submatrices of weights, we can use masked (sparsity-aware) multiplication kernels on GPUs [10]. Here, we can provide a mask indicating which blocks are sparse, and the GPU will skip those indices in matrix multiplication. We can derive the theoretical speedup for a single layer of weights:

$$\frac{O(dense)}{O(sparse)} = \frac{mn}{mn - K} = \frac{1}{1 - \frac{K}{mn}} = \frac{1}{1 - \rho}$$

Although we can theoretically achieve these speedup factors, in this paper we did not have access to these GPU kernels and we leave integration with efficient GPU kernels for future work.

V. Methods

We experiment with three different models to explore the role of training with sparsity in recurrent neural networks for language modeling.

A. Long Short-Term Memory (LSTM) Networks

LSTM is a type of recurrent neural network (RNN) [22] architecture for sequence modeling. As the network consumes sequential inputs, it also maintains an internal hidden state that remembers aspects of historical inputs. The LSTM consists of three parts: the input x_t, hidden state h_t, and cell state C_t. The *forget gate* f_t determines what information to forget from the prior hidden state. The *input gate* i_t determines how much new information is obtained from the input. The *output gate* o_t determines how the new cell state translates into the new hidden state.

The LSTM equations are as follows [23]:

$$\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
\tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
h_t &= o_t * \tanh(C_t)
\end{aligned}$$

We optimize over the weights W_f, W_i, W_o, W_C and biases b_f, b_i, b_o, b_C in the above equations.

B. Models

In order to explore sparse LSTM models, we compare a baseline vanilla LSTM with LSTM trained with several variants of the *block lasso objective*. Because the *block lasso objective* introduces additional regularization to the model, we choose our experiments to approximately compare different LSTMs with similar parameter count.

Baseline (LSTM). LSTM network with 1024 hidden units and 2496 units. We experimented with two LSTMs of different sizes to show that larger model capacity can give better performance. The 2496 unit is chosen to approximately match the number of parameters in the block lasso models after 50% pruning (post-hoc pruning).

Block Lasso LSTM (BL-LSTM). 3296 hidden unit LSTM with block lasso objective with maximum sparsity of $k = 90\%$ and $\lambda = 5 \times 10^{-4}$. After training, we prune the 50% of blocks with the smallest absolute values and fix them to 0.

Block Lasso with Gradual Pruning (BL-GP-LSTM). The exact same model as BL-LSTM, but we perform gradual pruning during training. After each epoch, the set of small magnitude trainable weights are fixed to exact zero, until 50% of blocks are fixed at 0. Once fixed to zero, these blocks can no longer be trained. This method aims to mimic synaptogenesis in human brain development - specifically, activity-dependent synapse elimination [13]. The added benefit of this training procedure is that it enables training speedup as the model becomes smaller as it trains. In this case, post-hoc pruning does not do anything as 50% of the blocks are already zero,

For block-sparse methods, we set our block size to 32×32, and base our regularization weight λ heuristically via the ratio of regularization loss to total loss before training: $\rho_R = \frac{\lambda \mathcal{L}_R}{L(\cdot) + \lambda \mathcal{L}_R}$ with $\rho_R = 0.01$. This hyperparameter can obviously be tuned for better result, but due to computational resource constraints, we were unable to tune this hyperparamter.

C. Task and Dataset

We investigate the performance of block-sparse Long Short-Term Memory (LSTM) networks for character language modeling on the WikiText-103 dataset (see Table I) [24].

Character language modeling is a task where the goal is to predict the next character in a sequence of text given previous characters in the texts. We can represent each character in a string of text as a token x_n. The goal of language modeling

TABLE I
STATISTICS FOR WIKITEXT-103 LANGUAGE MODELING DATASET.

	Train	Validation	Test
Articles	28,475	60	60
Words	103,227,021	217,646	245,569
Characters	537,936,515	1,139,711	1,278,143
Vocab (Words)	267,735	-	-
Vocab (Chars)	256	-	-

TABLE II
EXPERIMENT RESULTS FOR CHARACTER LEVEL LANGUAGE MODELING ON THE TEST SET OF WIKITEXT-103 DATASET.

Model	Parameters	Iterations	BPC
1024 LSTM	5M	36	2.39
2496 LSTM	26M	82	1.93
BL-LSTM	43M	108	1.98
50% Prune BL-LSTM	22M	-	61.82
BL-GP-LSTM	43M	57	2.05
50% Prune BL-GP-LSTM	**22M**	-	**2.05**

is to learn the conditional probability X_{n+1} given previous tokens within a sequence of $n+1$ characters.

$$P(X_{n+1} = x_{n+1} | X_n = x_n, X_{n-1} = x_{n-1}, ..., X_1 = x_1)$$

In practice, character language modeling is treated as a classification task, where at each step we want to classify the next character that should appear. This classification task is trained by maximum likelihood estimation using the cross entropy loss. It can be trained in a semi-supervised manner as all it requires is a corpus of text data. The performance of a character language model is typically measured using bits per character (BPC), which is defined as 2 to the power of the cross entropy between the correct characters and the predicted characters. The lower the bits per character, the better the model.

VI. EXPERIMENTS

We performed our experiments by training our models on a single 1080Ti GPU. Training each model ranges from 8 to 16 hours. We train our model by using a variant of stochastic gradient descent method called Adam [25], which provides faster convergence. All our models are trained with a learning rate of 10^{-3}, batch size of 32 and back propagation through time sequence length of 1024. Early stopping is performed on the validation set and we report the result on the test set in Table II.

When comparing the LSTM model with 1024 and 2496 units, our results support the obvious fact that given sufficient data, a larger model capacity is better in obtaining stronger performance than a smaller model. The BL-LSTM (with 3296 units) is able to achieve comparable performance to that of the 2496 LSTM, which indicates that the block lasso loss adds a regularization penalty to the model and the BL-LSTM model, despite having significantly more parameters, is heavily constrained. One would expect the BL-LSTM to be susceptible for naive neural pruning after training because many blocks would have been set to zero due to the block lasso term in the objective function, but we found that this is not the case. After training, we pruned 50% of BL-LSTM's parameters by setting the smallest magnitude blocks to zero. This resulted in a jump in BPC to 61.82, causing the model to result in degenerate performance and output. While our BL-GP-LSTM model gives slightly higher BPC, is amenable to post hoc pruning. In fact, the training process itself already enforces pruning so post hoc pruning has no effect on the model. It shows equal performance with up to 50% of the lowest magnitude weights set to zero. This is likely because the pruning occurs gradually, and thus the model has the opportunity to adapt its available parameters as it trains.

A. Qualitative Analysis

We now examine the outputs from our LSTM language models. An interesting result of a character language model is that, after training, we can sample from the model's probability distribution and generate text. We sample text by randomly sampling a character according to the probability distribution produced by the model, then feeding this character back into the model itself to generate the next character. Using this sampling method, we can visualize the model output and gauge its performance qualitatively. We show the output samples from our different models below (after some minor text cleaning).

1024 LSTM Output

```
All tracks began. Other Kapplee Exlectors
have been excluded to Jake Stein , born on
RusanHigh SEE Russia. It was a falling town
@-@ force cleared of 40 @,@ 022 : 03 @,@ 000
FU Vale.
```

BL-LSTM Output

```
Victoria and her fans stage has been done
over the entire feature. The second one and
three bad sights (attached from a pair of
a cheaper pristine plaza and the meter to
fleeing over).
```

BL-GP-LSTM

```
England was forced to acquire extreme
Commons under Benjamin Jese own,lft to forming
the French medieval situated under Russia and
Captain Victoria Lundman. At this point, his
brother Henry married Bodrh in the castle and
doved, by him, with Arnold L©n. TÃ©on Brown,
daughter of "High, Bay" and his wife.
```

As we can see from the output samples, the vanilla LSTM model produces the least coherent text. The text generated by the LSTM is somewhat grammatically correct. Note that the `@-@` is an artifact of the training set, so it is reasonable for the model to generate this. The BL-LSTM and BL-GP-LSTM generates outputs that are somewhat reasonable

B. Connectivity Visualization

See Appendix for visualizations. From the visualizations, it is clear that W_C weights are more prominent in sparse networks, whereas they are much less prominent in the baseline

LSTM model, indicating that baseline LSTM is relying a lot more on new inputs rather than the past. The dark stripes in the weight matrix mark entire neurons being pruned, thus the BL-GP-LSTM model is effectively learning a small and dense network in a large parameter space. The BL-LSTM visualization also shows that Block-Lasso regularization does lead to a reasonable approximation of sparsity constraint, and that pruning during training barely changes the model topology, as the difference between the BL-LSTM and the BL-GP-LSTM weight matrices is visually unnoticeable.

VII. Conclusions and Future Work

We have shown that it is feasible to train a block-sparse neural network for language modeling by incorporating the block lasso constraint into the objective function and applying a simple continuous relaxation to the constraint. Our work demonstrates that training a block sparse neural network enforces heavy regularization on the network. A naive pruning method applied to a model trained with *block lasso objective* yields poor performance. However, when we couple gradual pruning while training, our sparse model yields comparable performance to its dense counterpart with similar number of parameters.

Our work shows that there is promise to developing block-sparse models. However, it is unclear whether block-sparsity is better than other forms of sparsity (such as feature sparsity and neuron pruning). Contrary to [10], our learned structured sparsity does not exhibit small world patterns and, instead, seem to exhibit neural pruning behavior. Future work should investigate and further compare different types of sparsity to understand its efficacy.

References

[1] M. Denil, B. Shakibi, L. Dinh, N. De Freitas *et al.*, "Predicting parameters in deep learning," in *Advances in neural information processing systems*, 2013, pp. 2148–2156.

[2] S. Alford, R. A. Robinett, L. Milechin, and J. Kepner, "Pruned and structurally sparse neural networks," *CoRR*, vol. abs/1810.00299, 2018. [Online]. Available: http://arxiv.org/abs/1810.00299

[3] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *CoRR*, vol. abs/1506.02626, 2015. [Online]. Available: http://arxiv.org/abs/1506.02626

[4] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[5] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 806–814.

[6] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through l_0 regularization," *arXiv preprint arXiv:1712.01312*, 2017.

[7] S. Srinivas, A. Subramanya, and R. V. Babu, "Training sparse neural networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 455–462. [Online]. Available: https://doi.org/10.1109/CVPRW.2017.61

[8] E. Tartaglione, S. Lepsøy, A. Fiandrotti, and G. Francini, "Learning sparse neural networks via sensitivity-driven regularization," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, 2018, pp. 3882–3892. [Online]. Available: http://papers.nips.cc/paper/7644-learning-sparse-neural-networks-via-sensitivity-driven-regularization

[9] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nature communications*, vol. 9, no. 1, p. 2383, 2018.

[10] S. Gray, A. Radford, and D. P. Kingma, "Gpu kernels for block-sparse weights," 2017.

[11] N. Bell and M. Garland, "Efficient sparse matrix-vector multiplication on cuda," Tech. Rep., 2008.

[12] D. Smith Bassett and E. Bullmore, "Small-world brain networks," *The Neuroscientist : a review journal bringing neurobiology, neurology and psychiatry*, vol. 12, pp. 512–23, 01 2007.

[13] H. T. Cline, "Dendritic arbor development and synaptogenesis," *Current opinion in neurobiology*, vol. 11, no. 1, pp. 118–126, 2001.

[14] A. L. Tierney and C. A. Nelson III, "Brain development and the role of experience in the early years," *Zero to three*, vol. 30, no. 2, p. 9, 2009.

[15] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 68, no. 1, pp. 49–67, 2006.

[16] W. Wen, Y. He, S. Rajbhandari, M. Zhang, W. Wang, F. Liu, B. Hu, Y. Chen, and H. Li, "Learning intrinsic sparse structures within long short-term memory," *arXiv preprint arXiv:1709.05027*, 2017.

[17] S. Lym, E. Choukse, S. Zangeneh, W. Wen, M. Erez, and S. Shanghavi, "Prunetrain: Gradual structured pruning from scratch for faster neural network training," *arXiv preprint arXiv:1901.09290*, 2019.

[18] S. Narang, E. Undersander, and G. Diamos, "Block-sparse recurrent neural networks," *arXiv preprint arXiv:1711.02782*, 2017.

[19] A. N. Gomez, I. Zhang, K. Swersky, Y. Gal, and G. E. Hinton, "Targeted dropout," 2018.

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] C. Ramirez, V. Kreinovich, and M. Argaez, "Why l1 is a good approximation to l0: A geometric explanation," *Journal of Uncertain Systems*, vol. 7, no. 3, pp. 203–207, 2013.

[22] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[23] C. Olah, "Understanding lstm networks," 2015. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[24] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

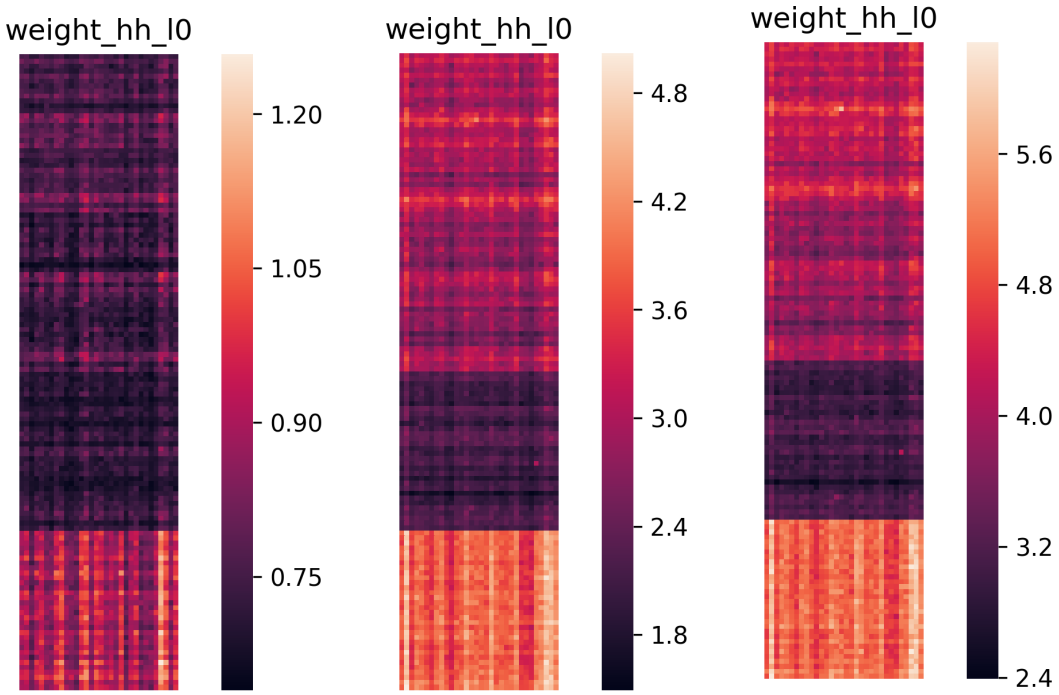

Fig. 2. LSTM hidden state weight matrices (W_i, W_f, W_C, W_o square matrices stacked from top to bottom) on epoch 1, 19 and 37 (left to right).

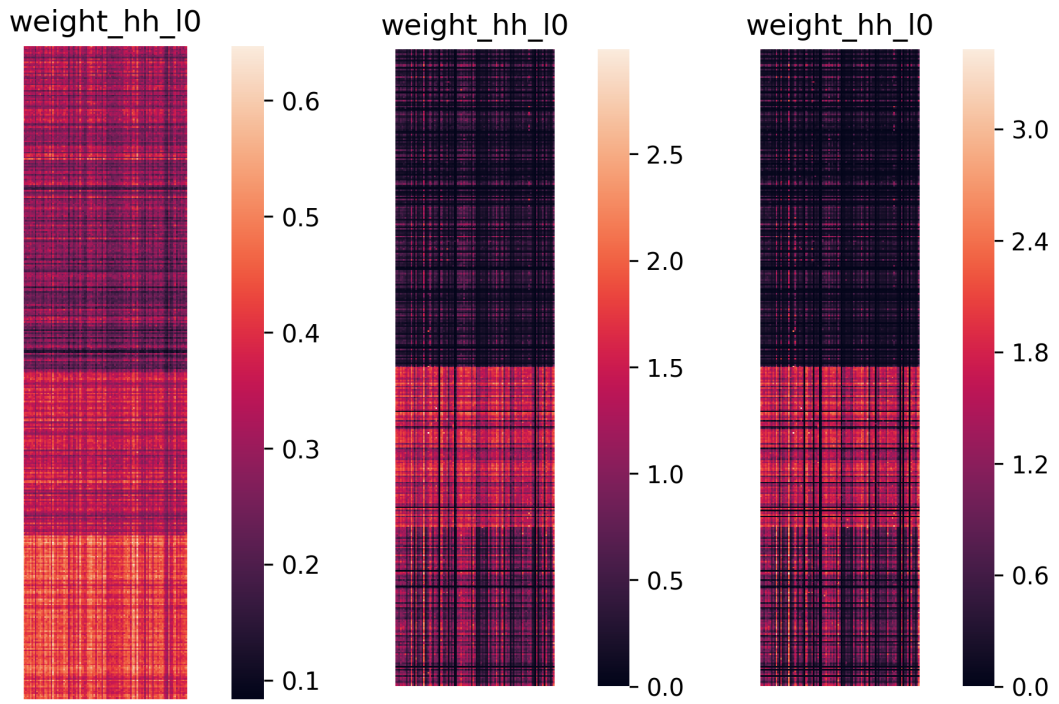

Fig. 3. BL-LSTM hidden state weight matrices (W_i, W_f, W_C, W_o square matrices stacked from top to bottom) on epoch 1, 55 and 109 (left to right).

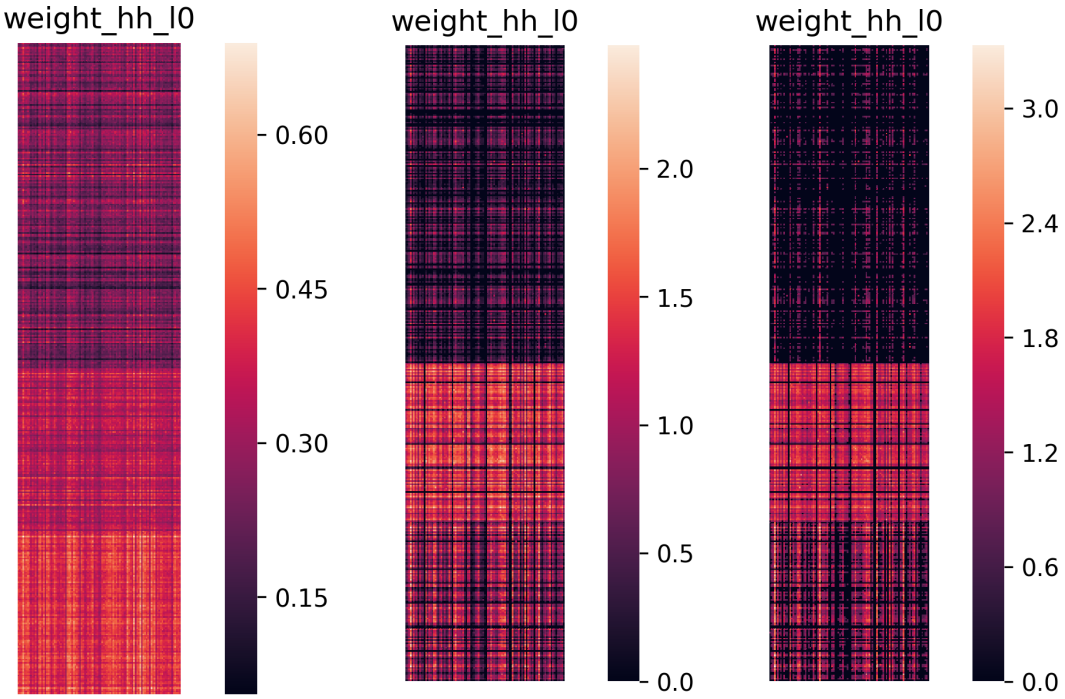

Fig. 4. BL-GP-LSTM hidden state weight matrices (W_i, W_f, W_C, W_o square matrices stacked from top to bottom) on epoch 1, 29 and 58 (left to right).