Learning Compact Recurrent Neural Networks with Block-Term Tensor Decomposition

Jinmian Ye[1], Linnan Wang[2], Guangxi Li[1], Di Chen[1], Shandian Zhe[3], Xinqi Chu[4], Zenglin Xu[1]

[1]SMILE Lab, University of Electronic Science and Technology of China [2]Brown University [3]University of Utah [4]Xjera Labs pte.ltd

2018 CVPR SALT LAKE CITY • JUNE 18-22

Introduction

RNNs are powerful sequence modeling tools. We introduce a compact architecture with Block-Term tensor decomposition to address the redundancy problem.

Challenges: 1) traditional RNNs suffer from an excess of parameters; and 2) Tensor-Train RNN has limited representation ability and flexibility due to the difficulty of searching the optimal setting of ranks.

Contributions: 1) introduces a new sparsely connected RNN architecture with Block-Term tensor decomposition; and 2) achieves better performance while maintaining fewer parameters.

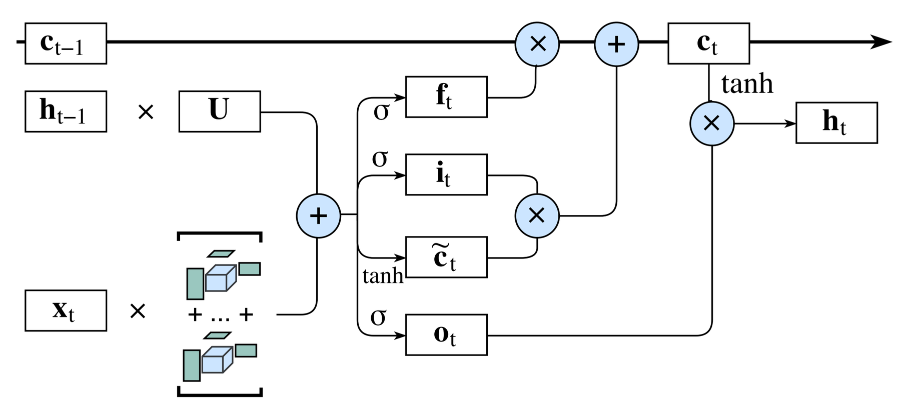


Figure: Architecture of BT-LSTM. The redundant dense connections between input and hidden state is replaced by low-rank BT representation.

Should be noted that we only substitute the input-hidden matrix multiplication while retaining the current design philosophy of LSTM.

$$\left(\mathbf{f}_t', \mathbf{i}_t', \tilde{\mathbf{c}}_t', \mathbf{o}_t'\right) = \mathbf{W} \cdot \mathbf{x}_t + \mathbf{U} \cdot \mathbf{h}_{t-1} + \mathbf{b} \quad (1)$$

$$(\mathbf{f}_t, \mathbf{i}_t, \tilde{\mathbf{c}}_t, \mathbf{o}_t) = \left(\sigma(\mathbf{f}_t'), \sigma(\mathbf{i}_t'), \tanh(\tilde{\mathbf{c}}_t'), \sigma(\mathbf{o}_t')\right) \quad (2)$$

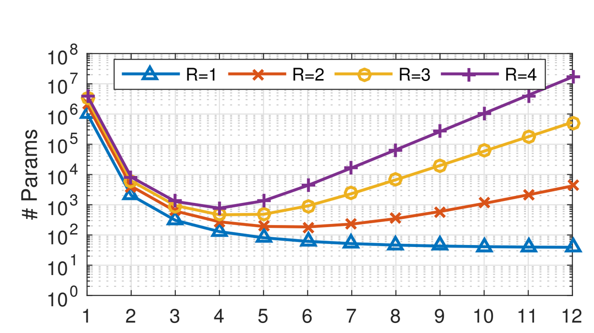
Analysis

Comparison of complexity and memory usage of vanilla RNN, Tensor-Train RNN (TT-RNN) and our BT-RNN. In this table, the weight matrix's shape is $I \times J$. Here, $J_{max} = \max_k(J_k), k \in [1, d]$.

Method	Time	Memory
RNN forward	$\mathcal{O}(IJ)$	$\mathcal{O}(IJ)$
RNN backward	$\mathcal{O}(IJ)$	$\mathcal{O}(IJ)$
TT-RNN forward	$\mathcal{O}(dIR^2 J_{max})$	$\mathcal{O}(RI)$
TT-RNN backward	$\mathcal{O}(d^2 IR^4 J_{max})$	$\mathcal{O}(R^3 I)$
BT-RNN forward	$\mathcal{O}(NdIR^d J_{max})$	$\mathcal{O}(R^d I)$
BT-RNN backward	$\mathcal{O}(Nd^2 IR^d J_{max})$	$\mathcal{O}(R^d I)$

Total #Parameters: $P_{BTD} = N(\sum_{k=1}^{d} I_k J_k R + R^d)$

Figure: The number of parameters w.r.t Core-order d and Tucker-rank R, in the setting of $I = 4096, J = 256, N = 1$. While the vanilla RNN contains $I \times J = 1048576$ parameters. When d is small, the first part $\sum_1^d I_k J_k R$ does the main contribution to parameters. While d is large, the second part R^d does. So we can see the number of parameters will go down sharply at first, but rise up gradually as d grows up (except for the case of $R = 1$).



Method

Background 1: Tensor Product Given $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ and $\mathcal{B} \in \mathbb{R}^{J_1 \times \cdots \times J_d}$, while $I_k = J_k$. To simplify, i_k^- denotes indices $(i_1, \ldots, i_{k-1})$, while i_k^+ denotes $(i_{k+1}, \ldots, i_d)$:

$$(\mathcal{A} \bullet_k \mathcal{B})_{i_k^-, i_k^+, j_k^-, j_k^+} = \sum_{p=1}^{I_k} \mathcal{A}_{i_k^-, p, i_k^+} \mathcal{B}_{j_k^-, p, j_k^+} \quad (3)$$

Background 2: Block-Term Decomposition

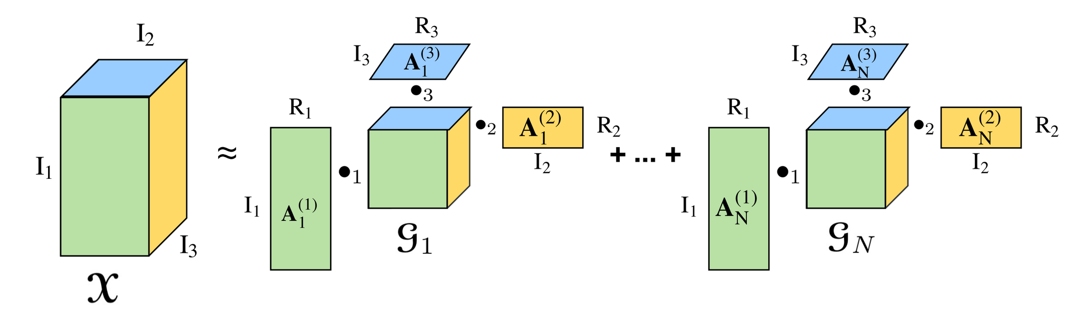
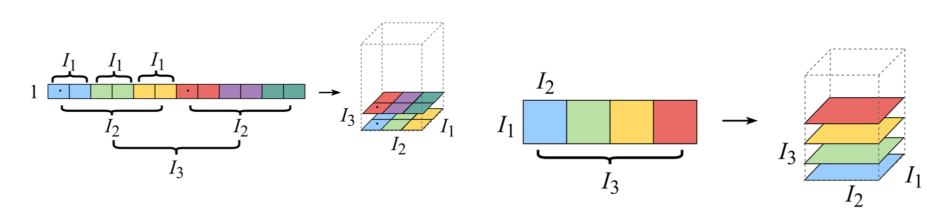


Figure: $\mathcal{X} = \sum_{n=1}^{N} \mathcal{G}_n \bullet_1 \mathcal{A}_n^{(1)} \bullet_2 \mathcal{A}_n^{(2)} \bullet_3 \cdots \bullet_d \mathcal{A}_n^{(d)}$

BT-RNN Model: 1) Tensorizing W and x; 2) Decomposing W with BTD; 3) Computing W · x; 4) Traing from scratch via BPTT!



(a) vector to tensor (b) matrix to tensor

Figure: Step 1: Tensorization operation in a case of 3-order tensors.

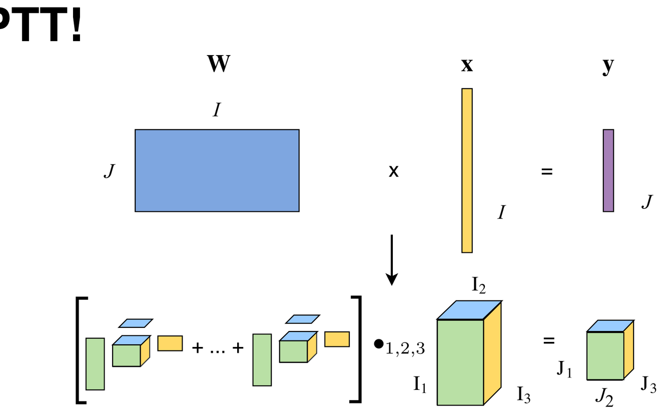


Figure: Step 2 & 3: Decomposing weight matrix and computing $\mathbf{y} = \mathbf{W}\mathbf{x}$.

Implementation: $\mathbf{W} \in \mathbb{R}^{J \times I}$, $\mathcal{W} \in \mathbb{R}^{J_1 \times I_1 \times J_2 \times \cdots \times J_d \times I_d}$, where $I = I_1 I_2 \cdots I_d$ and $J = J_1 J_2 \cdots J_d$. $\mathcal{G}_n \in \mathbb{R}^{R_1 \times \cdots \times R_d}$ denotes the core tensor, $\mathcal{A}_n^{(d)} \in \mathbb{R}^{I_d \times J_d \times R_d}$ denotes the factor tensor.

$$\mathbf{W} \cdot \mathbf{x}_t = \sum_{n=1}^{N} \mathcal{X}_t \bullet_1 \mathcal{A}_n^{(1)} \bullet_2 \cdots \bullet_d \mathcal{A}_n^{(d)} \bullet_{1,2,\ldots,d} \mathcal{G}_n \quad (4)$$

Conclusion

We proposed a Block-Term RNN architecture to address the redundancy problem in RNNs. Experiment results on 3 challenge tasks show that our BT-RNN architecture can not only consume several orders fewer parameters but also improve the model performance over standard traditional LSTM and the TT-LSTM.

References

[1] L. De Lathauwer. Decompositions of a higher-order tensor in block termspart ii: Definitions and uniqueness. SIAM SIMAX' 2008.

[2] Y. Yang, D. Krompass, and V. Tresp. Tensor-Train Recurrent Neural Networks for Video Classification. In ICML' 2017.

[3] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra. Draw: A recurrent neural network for image generation. In ICML' 2015.

[4] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In CVPR' 2015.

Experiments

Tasks: 1. Action Recognition in Videos; 2. Image Generation; 3. Image Captioning; 4. Sensitivity Analysis on Hyper-Parameters.

Datasets: 1. UCF11 YouTube Action dataset; 2. MNIST; 3. MSCOCO.

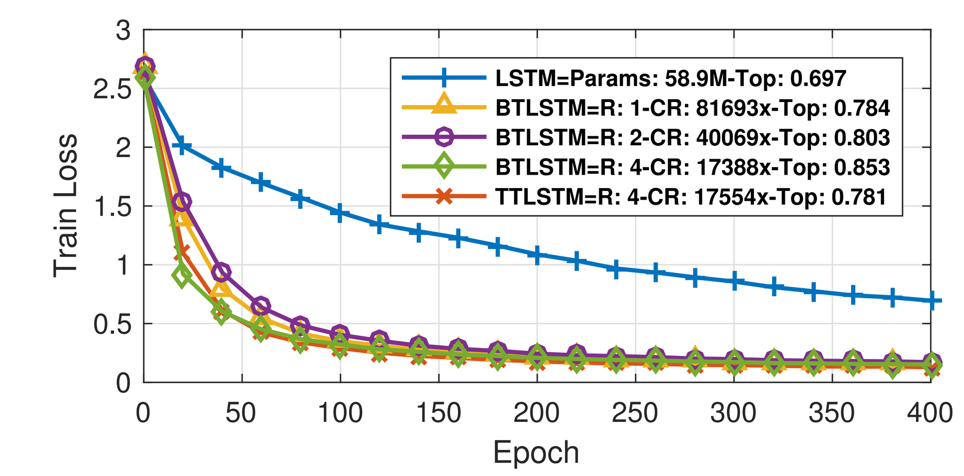
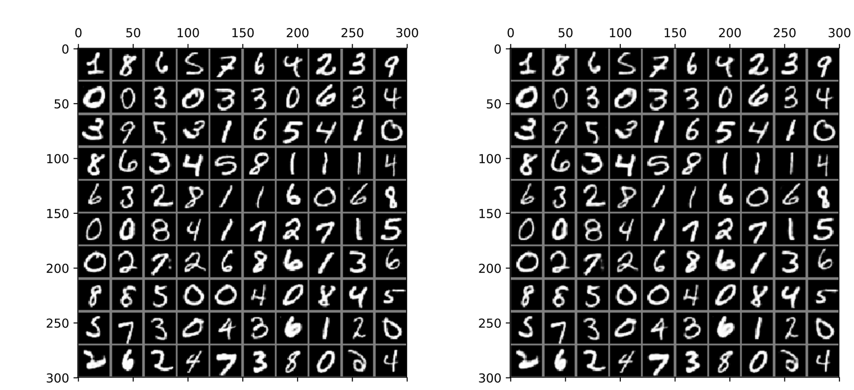


Figure: Performance of different RNN models on the Action Recognition task trained with UCF11.

	Method	Accuracy
Orthogonal Approaches	Original	0.712
	Spatial-temporal	0.761
	Visual Attention	0.850
RNN Approaches	LSTM	0.697
	TT-LSTM	0.796
	BT-LSTM	**0.853**

Table: State-of-the-art results on UCF11 dataset reported in literature, in comparison with our best model.

Task 1: We use a single LSTM cell as the model architecture to evaluate BT-LSTM against LSTM and TT-LSTM. The frames in video are directly input to the LSTM cell. The figure in left demonstrates the training loss of different models. From these experiments, we claim that our BT-LSTM has: 1) **8 × 10⁴** *times parameter reductions*; 2) *faster convergence*; 3) *better model efficiency*.



(a) LSTM, #Params:1.8M (b) BT-LSTM, #Params:1184

Task 2: In this experiment, we use an encoder-decoder architecture to generate images. We only substitute the encoder network to qualitatively evaluate the LSTM and BT-LSTM. The result shows that both LSTM and BT-LSTM can generate comparable images.

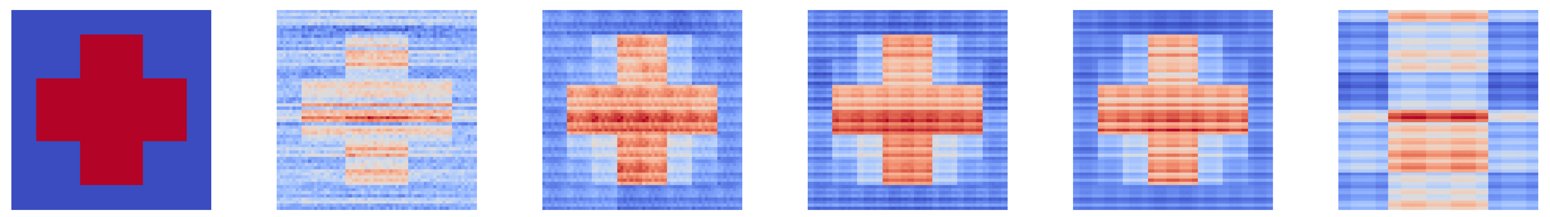
(c) **LSTM**: A train traveling down tracks next to a forest. **TT-LSTM**: A train traveling down train tracks next to a forest. **BT-LSTM**: A train traveling through a lush green forest.

(d) **LSTM**: A group of people standing next to each other. **TT-LSTM**: A group of men standing next to each other. **BT-LSTM**: A group of people posing for a photo.

(e) **LSTM**: A man and a dog are standing in the snow. **TT-LSTM**: A man and a dog are in the snow. **BT-LSTM**: A man and a dog playing with a frisbee.

(f) **LSTM**: A large elephant standing next to a baby elephant. **TT-LSTM**: An elephant walking down a dirt road near trees. **BT-LSTM**: A large elephant walking down a road with cars.

Task 3: We use the architecture described in [4], all the three models can generate proper sentences but with little improvement in BT-LSTM.



(g) Truth:W' P=4096 (h) $\mathbf{y} = \mathbf{W} \cdot \mathbf{x}$, P=4096 (i) d=2, R=1, N=1, P=129 (j) d=2, R=4, N=1, P=528 (k) d=2, R=1, N=2, P=258 (l) d=4, R=4, N=1, P=384

Figure: The trained $\mathbf{W}$ for different BT-LSTM settings. The closer to (a), the better $\mathbf{W}$ is.

Task 4: In this experiment, we use a single LSTM cell as the model architecture to evaluate BT-LSTM against LSTM and TT-LSTM.