Compressing Recurrent Neural Networks Using Hierarchical Tucker Tensor Decomposition

Miao Yin[1] , **Siyu Liao**[1] , **Xiao-Yang Liu**[2] , **Xiaodong Wang**[2] , **Bo Yuan**[1]
[1]Department of Electrical and Computer Engineering, Rutgers University
[2]Department of Electrical Engineering, Columbia University
{miao.yin, siyu.liao}@rutgers.edu, {xl2427, xw2008}@columbia.edu, bo.yuan@soe.rutgers.edu

Abstract

Recurrent Neural Networks (RNNs) have been widely used in sequence analysis and modeling. However, when processing high-dimensional data, RNNs typically require huge model sizes, thereby bringing a series of deployment challenges. Although the state-of-the-art tensor decomposition approaches can provide good model compression performance, these existing methods are still suffering some inherent limitations, such as restricted representation capability and insufficient model complexity reduction. To overcome these limitations, in this paper, we propose to develop compact RNN models using Hierarchical Tucker (HT) decomposition. HT decomposition brings a strong hierarchical structure to the decomposed RNN models, which is very useful and important for enhancing the representation capability. Meanwhile, HT decomposition provides higher storage and computational cost reduction than the existing tensor decomposition approaches for RNN compression. Our experimental results show that, compared with the state-of-the-art compressed RNN models, such as TT-LSTM, TR-LSTM and BT-LSTM, our proposed HT-based LSTM (HT-LSTM), consistently achieves simultaneous and significant increases in both compression ratio and test accuracy on different datasets.

1 Introduction

Recurrent Neural Networks (RNNs), especially their advanced variants such as Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU), have achieved unprecedented success in sequence analysis and processing. Thanks to their powerful capability of capturing and modeling the temporary dependency and correlation in the sequential data, the state-of-the-art RNNs have been widely deployed in many important artificial intelligence (AI) fields, such as natural language processing (NLP) [Sutskever *et al.*, 2011], speech recognition [Mikolov *et al.*, 2011], and computer vision [Yu *et al.*, 2016].

Despite their current prosperity, the efficient deployment of RNNs is still facing several challenges, especially the *large model size problem*. Due to the widespread existence of high-dimensional input data in many applications, e.g. NLP and video processing, the input-to-hidden weight matrices of RNNs are often extremely large. For instance, as pointed out in [Yang *et al.*, 2017], even with small-size hidden layer such as 256 hidden states, an LSTM working on UCF11 video recognition dataset [Liu *et al.*, 2009] already requires more than 50 million parameters. Such ultra-high model size, consequently, brings a series of deployment challenges for RNNs, including but not limited to high difficulty of training, susceptibility to overfitting, long processing latency and inefficient energy consumption etc.

Prior Work on RNN Compression. To address RNNs' ultra-large model size problem, several *model compression* approaches, such as pruning and quantization [Song *et al.*, 2016], have been proposed and studied in prior work. Among them, the most promising solution is *Tensor Decomposition*, a technique that represents a large tensor with the combination of multiple small tensor cores. By its nature, tensor decomposition approach is inherently a powerful tool for identifying and exploring the higher-order data correlation. From the perspective of model compression, such strong correlation-capturing capability makes tensor decomposition very attractive and well-suited for exploiting and reducing the model redundancy in large-scale RNNs. Recent advances in model compression research already show that, various tensor decomposition-based compression methods, including tensor train (TT) [Yang *et al.*, 2017], tensor ring (TR) [Pan *et al.*, 2019] and block-term (BT) [Ye *et al.*, 2018], can bring several orders-of-magnitude fewer parameters for large-size RNNs with still maintaining high classification/prediction performance.

Limitations of Prior Work. Although the existing tensor decomposition-based RNN compression approaches already show their promising potentials, these state-of-the-art methods are still facing two inherent limitations: 1) the tensor decomposition approaches used in [Yang *et al.*, 2017], [Ye *et al.*, 2018] and [Pan *et al.*, 2019] have strict constraints on either the shapes or the combination manners of the component tensor cores, thereby limiting the representation ability of the corresponding compressed RNN models. For instance, TT decomposition requires the border tensor cores have to be rank-1, which directly hinders the representation power of TT-based RNNs. More generally, when using TT, TR or

BT, the important hierarchical structure, which is important to capture many inherent hierarchical patterns or representation in the data, is missing at the inter-tensor-core level, thereby limiting the representation capability of the entire neural network models; and 2) from the perspective of complexity analysis, TT, TR and BT are not the best tensor decomposition approaches that provide the most promising space or computational complexity reduction. For instance, executing BT-based RNN models suffers computation overhead due to the extra flatten and permutation operations. More generally, when we consider to apply tensor decomposition to compress RNN models, in many settings the state-of-the-art TT, TR and BT solutions are inferior to some other types of tensor decomposition option in terms of numbers of parameters and/or operations saving. Consequently, the existing tensor decomposition-based compression methods are not the optimal solutions to fully exploit and minimize the RNN model redundancy.

Technical Preview & Benefits. To overcome these limitations and fully unlock the potentials of tensor decomposition in model compression, in this work we propose to develop compact RNN models by using *Hierarchical Tucker* (HT) decomposition [Hackbusch and Kühn, 2009], a little explored but powerful tool for capturing and modeling the correlation and structure in the high-dimensional data. Unlike other popularly used tensor decomposition methods such as TT, TR and BT, HT decomposition enables the decomposed tensors exhibit strong hierarchical structure, which is very useful and important for enhancing the representation capability of RNN models. Meanwhile, comparing to its well-explored counterparts, HT decomposition can bring more complexity reduction with the same rank setting, thereby enabling HT-based RNN models have lower storage and computational costs than the prior work with the same or even higher accuracy. In overall, the features and benefits of HT-based RNN models are summarized as follows:

- HT-based RNNs exhibit stronger representation power than the existing tensor decomposition-based models. More specifically, the hierarchical structure imposed on the input-to-hidden layers makes RNNs can exploit and extract the important representation and pattern from high-dimensional data in a much more hierarchical, precise and comprehensive way, thereby significantly improving RNN models' representation capability. This benefit on representation capability is also verified by the empirical experiments. Our proposed HT-LSTM, as the the compressed LSTM models using HT decomposition, achieves higher accuracy than vanilla RNN, the state-of-the-art compressed RNNs as well as non-RNN models on various video recognition datasets.
- HT-based RNN models have much lower storage and computational costs than the state of the art. Compared with TT, TR and BT decomposition adopted in prior work, HT decomposition inherently provides higher complexity reduction on the same-size tensor data with the same selected rank. By leveraging such theoretical advantage, we can compress large-size RNN models in the HT format with requiring very few parameters. Experimental results show, compared with vanilla LSTM, HT-LSTM achieves very high compression ratio with even higher accuracy. Meanwhile, compared with the state-of-the-art compressed LSTM such as TT-LSTM, TR-LSTM and BT-LSTM, HT-LSTM consistently achieve simultaneous and significant increase in both compression ratio and test accuracy on different datasets.

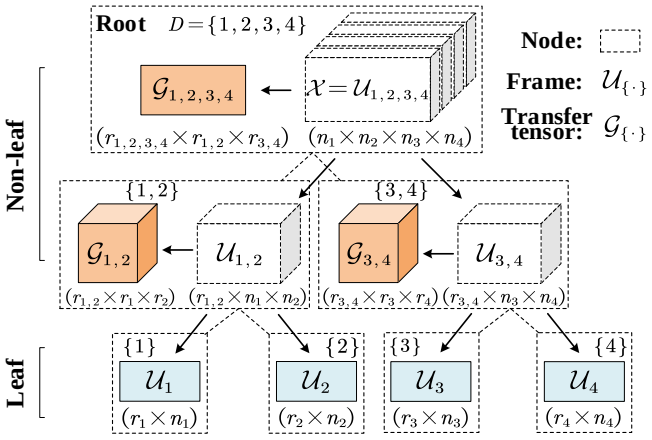

Figure 1: An illustration of HT decomposition with d=4. All the dashed lines and boxes describe a binary tree with root $D = \{1, 2, 3, 4\}$, where the dashed boxes represent the nodes. Here node $\{1\}$ is a leaf node, whose father and brother are node $\{1, 2\}$ and node $\{2\}$, respectively. For leaf nodes, they are only associated with leaf frame, and for non-leaf nodes, they are associated with transfer tensors and non-leaf frames. Here $\boldsymbol{\mathcal{X}}$ is decomposed to a set of orange-colored transfer tensors and blue-colored leaf frames.

2 Hierarchical Tucker-based RNN Models

In this section, we describe the details of HT-based RNN models. First, we introduce the preliminaries of tensor basics, tensor computation and hierarchical tucker decomposition. Then, we reformulate the forward propagation and backward propagation procedure on the original input-to-hidden layer to the HT format, and thereby forming a new HT layer, which is the building component of our proposed compact HT-based RNN models. Furthermore, in order to evaluate the efficiency of HT-based compression approach, we analyze the computational and storage complexity of HT-based RNN models and make comparison with the other methods.

2.1 Preliminaries

Notation. Throughout the paper we use boldface calligraphic script letters, boldface capital letters, and boldface lower-case letters to represent tensors, matrices, and vectors, respectively, e.g. $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$, $\boldsymbol{X} \in \mathbb{R}^{n_1 \times n_2}$, and $\boldsymbol{x} \in \mathbb{R}^{n_1}$. In addition, $\boldsymbol{\mathcal{X}}_{(i_1,\cdots,i_d)} \in \mathbb{R}$ denotes the entry of tensor $\boldsymbol{\mathcal{X}}$. Similarly, $\boldsymbol{X}_{(i,j)}$ represents the entry of matrix $\boldsymbol{X}$.

Tensor Contraction. An HT-decomposed tensor is essentially the consecutive product of multiple tensor contraction results, where tensor contraction is executed between two tensors with at least one matched dimension. For instance, given two tensors $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times n_2 \times l}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{l \times m_1 \times m_2}$, where the 3rd dimension of $\boldsymbol{\mathcal{A}}$ matches the 1st dimension

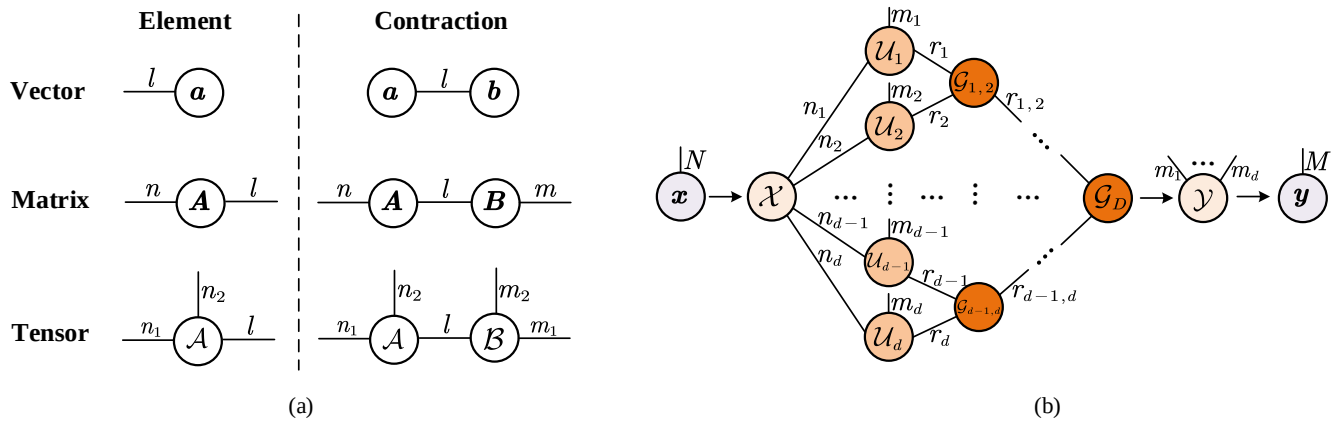

Figure 2: (a) Graphical representation of element and computation in the tensor diagram. Here the each line represents one dimension, and the variable near the line is the size of that dimension. (b) Representation of matrix-vector multiplication on a HT-decomposed layer using tensor diagram. Here weight matrix is already decomposed to the HT format with a set of 2-D leaf frames and 3-D transfer tensor.

of $\boldsymbol{\mathcal{B}}$ with length l, the tensor contraction result is a size-$n_1 \times n_2 \times m_1 \times m_2$ tensor as

$$(\boldsymbol{\mathcal{A}} \times_1^3 \boldsymbol{\mathcal{B}})_{(i_1,i_2,j_1,j_2)} = \sum_{\alpha=1}^{l} \boldsymbol{\mathcal{A}}_{(i_1,i_2,\alpha)} \boldsymbol{\mathcal{B}}_{(\alpha,j_1,j_2)}.$$

Hierarchical Tucker Decomposition. The Hierarchical Tucker decomposition is a special type of tensor decomposition approach with hierarchical levels with respect to the order of the tensor. As illustrated in Figure 1, an HT-decomposed tensor can be recursively decomposed into intermediate components, referred as *frames*, from top to bottom in a binary tree, where each frame corresponds to a unique *node*, and each node is associated with a *dimension set*. In general, for a HT-decomposed tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, we can build a binary tree with a root node associated with $D = \{1, 2, \cdots, d\}$ and $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{U}}_D$ as the root frame. For each non-leaf frame $\boldsymbol{\mathcal{U}}_s \in \mathbb{R}^{r_s \times n_{\mu_s} \times \cdots \times n_{\nu_s}}$, where $s \subsetneq D$ is associated with the node corresponding to $\boldsymbol{\mathcal{U}}_s$, $s_1, s_2 \subsetneq s$ are associated with the left and right child nodes of the s-associated node, and $\mu_s = \min(s), \nu_s = \max(s), \boldsymbol{\mathcal{U}}_s$ can be recursively decomposed to its left and right child frames ($\boldsymbol{\mathcal{U}}_{s_1}$ and $\boldsymbol{\mathcal{U}}_{s_2}$) and transfer tensor $\boldsymbol{\mathcal{G}}_s \in \mathbb{R}^{r_s \times r_{s_1} \times r_{s_2}}$ as

$$\boldsymbol{\mathcal{U}}_s = \boldsymbol{\mathcal{G}}_s \times_1^2 \boldsymbol{\mathcal{U}}_{s_1} \times_1^2 \boldsymbol{\mathcal{U}}_{s_2}. \tag{1}$$

Consequently, by performing this recursive decomposition till the bottom of the binary tree, we can decompose the original $n_1 \times \cdots \times n_d$-order tensor $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{U}}_D$ into the combination of the 2-order leaf frames and 3-order transfer tensors. Notice that here r_s, as *hierarchical rank*, is an important parameter that determines the decomposition effect.

2.2 HT-based RNN Models

This subsection describes the details of compressing the large-size RNN models to the compact ones using HT decomposition, including the key steps as well as the important HT-based forward and backward propagation schemes.

Tensorization. In general, the key idea of building HT-RNN is to transform the weight matrix $\boldsymbol{W} \in \mathbb{R}^{M \times N}$ to the HT-based format. Considering $\boldsymbol{W}$ is a 2-D matrix, while HT decomposition is mainly performed on high-order tensor, we first need to reshape $\boldsymbol{W}$ as well as its affiliated input vector $\boldsymbol{x} \in \mathbb{R}^N$ and output vector $\boldsymbol{y} \in \mathbb{R}^M$ to tensor format as $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{m_1 \times \cdots \times m_d \times n_1 \times \cdots \times n_d}$, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ and $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{m_1 \times \cdots \times m_d}$, respectively, where $M = \prod_{i=1}^d m_i$ and $N = \prod_{j=1}^d n_j$.

Decomposing $\boldsymbol{W}$. Given a tensorized $\boldsymbol{W}$ as $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{m_1 \times \cdots \times m_d \times n_1 \times \cdots \times n_d}$, we can now leverage HT decomposition to represent the large-size $\boldsymbol{W}$ using a set of small-size matrices and tensors. In general, following Equation (1), $\boldsymbol{\mathcal{W}}$ can be decomposed as

$$\begin{aligned} \boldsymbol{\mathcal{W}}_{(i_1,\cdots,i_d,j_1,\cdots,j_d)} = \sum_{k=1}^{r_D} \sum_{p=1}^{r_{D_1}} \sum_{q=1}^{r_{D_2}} (\boldsymbol{\mathcal{G}}_D)_{(k,p,q)} \\ \cdot (\boldsymbol{\mathcal{U}}_{D_1})_{(p,\varphi_{D_1}(\boldsymbol{i},\boldsymbol{j}))} (\boldsymbol{\mathcal{U}}_{D_2})_{(q,\varphi_{D_2}(\boldsymbol{i},\boldsymbol{j}))}, \end{aligned} \tag{2}$$

where $\varphi_s(\boldsymbol{i}, \boldsymbol{j})$ is a mapping function that produces the correct indices $\boldsymbol{i} = (i_1, \cdots, i_d)$ and $\boldsymbol{j} = (j_1, \cdots, j_d)$ for a specified frame $\boldsymbol{\mathcal{U}}_s$ with the given s and d. For instance, with $d = 6$ and $s = \{3, 4\}$, the output of $\varphi_s(\boldsymbol{i}, \boldsymbol{j})$ is (i_3, i_4, j_3, j_4). In addition, $\boldsymbol{\mathcal{U}}_{D_1}$ and $\boldsymbol{\mathcal{U}}_{D_2}$ can be recursively computed as

$$\begin{aligned} (\boldsymbol{\mathcal{U}}_s)_{(k,\varphi_s(\boldsymbol{i},\boldsymbol{j}))} = \sum_{p=1}^{r_{s_1}} \sum_{q=1}^{r_{s_2}} (\boldsymbol{\mathcal{G}}_s)_{(k,p,q)} \\ \cdot (\boldsymbol{\mathcal{U}}_{s_1})_{(p,\varphi_{s_1}(\boldsymbol{i},\boldsymbol{j}))} (\boldsymbol{\mathcal{U}}_{s_2})_{(q,\varphi_{s_1}(\boldsymbol{i},\boldsymbol{j}))}, \end{aligned} \tag{3}$$

where $D = \{1, 2, \cdots, d\}$, $D_1 = \{1, \cdots, \lfloor d/2 \rfloor\}$ and $D_2 = \{\lceil d/2 \rceil, \cdots, d\}$ are associated with left and right child nodes of the root node.

HT Layer. With the HT-decomposed weight matrix, the component HT layer of HT-based RNN models can now be developed. Specifically, the HT-format matrix-vector multiplication, as the kernel computation in the forward propagation procedure, is performed as follows:

$$\begin{aligned} \boldsymbol{\mathcal{Y}}_{(\boldsymbol{i})} = \sum_{\boldsymbol{j}} \sum_{k=1}^{r_D} \sum_{p=1}^{r_{D_1}} \sum_{q=1}^{r_{D_2}} (\boldsymbol{\mathcal{G}}_D)_{(k,p,q)} \\ \cdot (\boldsymbol{\mathcal{U}}_{D_1})_{(p,\varphi_{D_1}(\boldsymbol{i},\boldsymbol{j}))} (\boldsymbol{\mathcal{U}}_{D_2})_{(q,\varphi_{D_2}(\boldsymbol{i},\boldsymbol{j}))} \boldsymbol{\mathcal{X}}_{(\boldsymbol{j})}. \end{aligned} \tag{4}$$

Considering the desired output $\boldsymbol{y}$ of HT-layer is a vector, the calculated $\boldsymbol{\mathcal{Y}}$ needs to be re-shaped again to the 1-D format. Consequently, we denote the entire forward computing

Model	Space	Time
RNN FP	$\mathcal{O}(NM)$	$\mathcal{O}(NM)$
RNN BP		$\mathcal{O}(NM)$
TT-RNN FP	$\mathcal{O}(dmnr^2)$	$\mathcal{O}(dmr^2N)$
TT-RNN BP		$\mathcal{O}(d^2mr^4N)$
TR-RNN FP	$\mathcal{O}(dmnr^2)$	$\mathcal{O}(dr^3N + dr^3M)$
TR-RNN BP		$\mathcal{O}(d^2r^5N + nd^2r^5M)$
BT-RNN FP	$\mathcal{O}(dmnr + r^d)$	$\mathcal{O}(dmr^dNC)$
BT-RNN BP		$\mathcal{O}(d^2mr^dNC)$
HT-RNN FP	$\mathcal{O}(dmnr + dr^3)$	$\mathcal{O}(dmr^2N + dr^3N)$
HT-RNN BP		$\mathcal{O}(d^2mr^5N + d^2r^6N)$

Table 1: Comparison of complexity with different tensor decomposition-based RNNs. Here FP and BP mean forward and backward propagation, respectively. C is the CP rank-value defined in BT decomposition, and $r = \max_{s\subsetneq D} r_s$, $m = \max_{k\in D} m_k$, $n = \max_{k\in D} n_k$,

procedure from input $\boldsymbol{x}$ to output $\boldsymbol{y}$ as

$$\boldsymbol{y} = HTL(\boldsymbol{W}, \boldsymbol{x}). \tag{5}$$

Remark-1: Hierarchical Structure in HT-layer. In tensor theory both the tensors and their computations can be graphically represented using *tensor diagram* (see Figure 2(a)). For HT decomposition, its inherent hierarchical characteristics make the corresponding HT-layer exhibit strong multi-level hierarchical structure, which is visualized in the specific tensor diagram of HT-layer (see Figure 2(b)). Considering that a well-known feature and advantage of deep neural network is its strong ability of capturing hierarchical pattern and representation via its multi-layer structure, the existence of such hierarchical structure in HT-layer can effectively improve the representation capability of the overall neural network models. In Section 3, empirical experiments on different datasets demonstrate that HT-based RNN models indeed outperform many other types of RNN models in terms of accuracy. Besides, it is worth noting that some recent advances in learning theory [Nadav *et al.*, 2018] also indicates the strong connection and correlation between HT modeling and expressive power of deep neural networks.

HT-LSTM. With the HT-based component layers, a HT-based RNN model can be simply constructed by replacing the original uncompressed layer with the HT-layer. Considering LSTM is the most popular and advanced variant of RNNs, we develop the corresponding HT-LSTM model as follows:

$$\begin{aligned}
\boldsymbol{u}[t] =& \sigma(HTL(\boldsymbol{W}_u, \boldsymbol{x}[t]) + \boldsymbol{V}_u\boldsymbol{h}[t-1] + \boldsymbol{b}_u)\\
\boldsymbol{f}[t] =& \sigma(HTL(\boldsymbol{W}_f, \boldsymbol{x}[t]) + \boldsymbol{V}_f\boldsymbol{h}[t-1] + \boldsymbol{b}_f)\\
\boldsymbol{o}[t] =& \sigma(HTL(\boldsymbol{W}_o, \boldsymbol{x}[t]) + \boldsymbol{V}_o\boldsymbol{h}[t-1] + \boldsymbol{b}_o)\\
\boldsymbol{c}[t] =& \boldsymbol{f}[t] \odot \boldsymbol{c}[t-1] + \boldsymbol{u}[t] \odot\\
& \tanh(HTL(\boldsymbol{W}_c, \boldsymbol{x}[t]) + \boldsymbol{V}_c\boldsymbol{h}[t-1] + \boldsymbol{b}_c)\\
\boldsymbol{h}[t] =& \boldsymbol{o}[t] \odot \tanh(\boldsymbol{c}[t]),
\end{aligned} \tag{6}$$

where σ, $\tanh$ and $\odot$ are the sigmoid function, hyperbolic function and element-wise product, respectively.

HT-based Gradient Calculation. To ensure the valid training on HT-RNN, the gradient calculation in the backward propagation should also be accordingly reformulated to HT-based format. In general, considering for HT layer $\boldsymbol{\mathcal{W}} = \boldsymbol{\mathcal{U}}_D$ and $\frac{\partial \boldsymbol{\mathcal{Y}}}{\partial \boldsymbol{\mathcal{U}}_D} = \boldsymbol{\mathcal{X}}$, assuming s is associated with a left node, as we denote that $F(s)$ and $B(s)$ are the sets associated with the father and brother nodes of the s-associated node in the binary tree, respectively, and define $\mu_s = \min(s), \nu_s = \max(s)$, the partial derivative of output tensor with respect to frames can be calculated in the following recursive way until $F(s)$ is equal to D:

$$\begin{aligned}
\frac{\partial \boldsymbol{\mathcal{Y}}}{\partial \boldsymbol{\mathcal{U}}_s} =& \boldsymbol{\mathcal{G}}_{F(s)} \times_1^3 \boldsymbol{\mathcal{U}}_{B(s)}\\
& \times_{\substack{1,\cdots,\nu_{B(s)}-\mu_{B(s)}+2,\nu_{F(s)}-\mu_{F(s)}+3,\\ \cdots,\nu_{F(s)}-\mu_{F(s)}+\nu_{B(s)}-\mu_{B(s)}+3}}^{1,3,\cdots,2\nu_{B(s)}-2\mu_{B(s)}+4} \frac{\partial \boldsymbol{\mathcal{Y}}}{\partial \boldsymbol{\mathcal{U}}_{F(s)}}.
\end{aligned} \tag{7}$$

Based on Equation (7), the gradients for leaf frames and transfer tensors can be computed as follows:

$$\frac{\partial L}{\partial \boldsymbol{\mathcal{U}}_s} = \frac{\partial \boldsymbol{\mathcal{Y}}}{\partial \boldsymbol{\mathcal{U}}_s} \times_{1,\cdots,\mu_s-1,\nu_s+1,\cdots,d}^{\nu_s-\mu_s+3,\cdots,d+1} \frac{\partial L}{\partial \boldsymbol{\mathcal{Y}}}, \tag{8}$$

$$\begin{aligned}
\frac{\partial L}{\partial \boldsymbol{\mathcal{G}}_s} =& \frac{\partial \boldsymbol{\mathcal{Y}}}{\partial \boldsymbol{\mathcal{U}}_s} \times_{2,\cdots,\nu_{s_1}-\mu_{s_1}+2}^{2,\cdots,\nu_{s_1}-\mu_{s_1}+2} \boldsymbol{\mathcal{U}}_{s_1}\\
& \times_{2,\cdots,\nu_{s_2}-\mu_{s_2}+2}^{3,\cdots,\nu_{s_2}-\mu_{s_2}+3} \boldsymbol{\mathcal{U}}_{s_2} \times_{1,\cdots,d}^{4,\cdots,d+3} \frac{\partial L}{\partial \boldsymbol{\mathcal{Y}}}.
\end{aligned} \tag{9}$$

2.3 Complexity Analysis

To better understand the impact of HT decomposition on RNNs, we analyze the theoretical complexity of HT-RNN, and compare it with the vanilla uncompressed RNN as well as other tensor decomposition-based RNN models. Table 1 summarizes the space complexity and time complexity of different RNN models. It is seen that compared with the other compressed RNN models using tensor decomposition, HT-RNN has the lowest space complexity to store the model parameters. Meanwhile, HT-RNN also enjoys less time complexity than most listed models for both forward and backward propagation. It is worth noting that though TT-RNN has even less time complexity than HT-RNN, it has weaker representation capability, which translates to the lower accuracy, as demonstrated via the experimental results in Section 3.

Remark-2. Besides theoretical complexity analysis, we also verify the low-cost benefits of HT-based compression via empirical experiments. Figure 3(a) shows the number of parameters to store a compressed weight matrix, and Figure 3(b) shows the number of needed operations for multiplication between the compressed weight matrix and vector. Here we adopt the size-$57,600 \times 256$ weight matrix used in [Yang *et al.*, 2017] [Ye *et al.*, 2018] [Pan *et al.*, 2019] for evaluation. From Figure 3 it is seen that HT-based approach indeed achieves lower costs, especially on storage requirement, than other tensor decomposition-based methods. Next, the experiments in Section 3 further show the advantages of HT-based RNN on compression ratios over various datasets.

3 Experiments

In this section, we evaluate the performance of HT-based RNN on different datasets, and compare them with the state-of-the-art with respect to compression ratio and test accuracy.

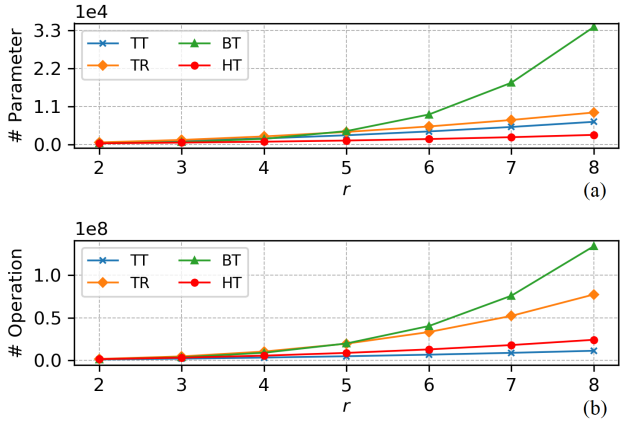

Figure 3: Top: Comparison on number of required parameters of weight matrix. Bottom: Comparison on number of required operations of matrix-vector multiplications. All the tensor decomposition methods use the same setting $d = 5$, $(n_1, \cdots, n_5) = (8, 10, 10, 9, 8)$, $(m_1, \cdots, m_5) = (4, 4, 2, 4, 2)$, and r is the rank.

Considering LSTM is the current most commonly used RNN variant in both academia and industry, our experiments focus on HT-LSTM, and compare it with the vanilla uncompressed LSTM and recent advances in compressed RNN such as TT-LSTM, BT-LSTM and TR-LSTM. Besides, we also compare HT-LSTM with other reported models that can be evaluated on the same datasets.

Training Strategy. Following the similar setting in prior work, we adopt two types of training strategy: end-to-end direct training and training with pre-trained CNN. In the end-to-end direct training the input of LSTM is the raw data, e.g. video clips; while training with pre-trained CNNs means the back-end LSTM receives the compact features extracted by a front-end pre-trained CNN. Next we describe our experiments belonging to these two categories, respectively.

3.1 End-to-End Direct Training

Hyperparameter Setting. We train the models using ADAM optimizer with L2 regularization of coefficient 0.001. Also, dropout rate is set as 0.25 and batch size is 16.

UCF11 Dataset. The UCF11 dataset [Liu *et al.*, 2009] consists of 11-class human action (e.g. biking, diving, basketball) videos with totally 1,600 video clips. Each class is assembled by 25 video groups, where each group contains at least 4 action clips. For each clip, the resolution is 320×240.

At data pre-processing stage we choose the same settings used in the related work [Ye *et al.*, 2018] [Pan *et al.*, 2019] for fair comparison. Specifically, the resolution of video clips is first scaled down to 160×120, and then 6 frames from each clip are randomly sampled to form the sequential input for our HT-LSTM model.

For the baseline vanilla uncompressed LSTM model, it contains 4 input-to-hidden layers, where the size of its input vector is $160 \times 120 \times 3 = 57,600$, and the number of hidden states in each layer is 256. For our proposed HT-LSTM, the input vector is reshaped to a tensor of shape $8 \times 10 \times 10 \times 9 \times 8$, and the output tensor of the input-to-hidden layer is of shape $4 \times 4 \times 2 \times 4 \times 2$. All leaf ranks are set as 4, and all non-leaf ranks are set as 5.

Model	CR	# Param.	Accuracy (%)
LSTM	1	59M	69.7
TT-LSTM (ICML-17)	17,554×	3,360	79.6
BT-LSTM (CVPR-18)	17,414×	3,387	85.3
TR-LSTM (AAAI-19)	34,193×	1,725	86.9
HT-LSTM (Ours)	**47,375×**	**1,245**	**87.2**

Table 2: Performance of different RNN compression work on UCF11 dataset using end-to-end direct training. CR stands for compression ratios. Results of TT-LSTM, BT-LSTM and TR-LSTM are reported in [Yang *et al.*, 2017], [Ye *et al.*, 2018] and [Pan *et al.*, 2019], respectively.

Table 2 summarizes the performance of our HT-LSTM on UCF11 dataset and compare it with the related work. It is seen that compared with vanilla LSTM using 59 million parameters, HT-LSTM only needs 47,375× fewer parameters with 17.5% accuracy increase. Compared with the recent advances on compressing RNNs using other tensor decomposition methods, including TT-LSTM, BT-LSTM and TR-LSTM, our proposed HT-LSTM requires at least 1.38× fewer parameters with at least 0.3% increase in accuracy.

Youtube Celebrities Face Dataset. Youtube dataset [Kim *et al.*, 2008] contains 1,910 video clips from 47 subjects, where each of it is a celebrated individual such as movie star. Also, the resolutions of the frames vary for different video clips. Being consistent with prior work, for data pre-processing the resolution of the input data to HT-LSTM is re-scaled as 160×120. Also, 6 frames in each video clips are randomly sampled to form the input sequence.

In this experiment we build a HT-LSTM with the similar setting with the one used in UCF11 dataset. To be specific, 4 input-to-hidden layers are equipped, and the shapes of tensorized input and output vectors are $8 \times 10 \times 10 \times 9 \times 8$ and $4 \times 4 \times 2 \times 4 \times 2$, respectively. A slight difference is in this HT-LSTM all leaf ranks are set as 3, and all non-leaf ranks are set as 4.

Table 3 compares the performance of HT-LSTM with prior work on Youtube Celebrities Face dataset. Considering among the state-of-the-art work, only TT-RNN [Yang *et al.*, 2017], including TT-GRU and TT-LSTM, reports both the accuracy and compression ratio on this dataset, the comparison in Table 3 is mainly between HT-LSTM and TT-based solutions. From this table it is seen that HT-LSTM achieves 72,818× compression ratio over the original uncompressed LSTM with much higher accuracy. Compared with the existing high-compression model TT-GRU, HT-LSTM has 4.1× fewer parameters but offering 8.1% accuracy increase.

Besides, on the same Youtube dataset we also compare HT-LSTM with several other reported works without using tensor decomposition method. As shown in Table 4, among those works the state-of-the-art model is [Ortiz *et al.*, 2013], which has the highest reported accuracy (80.8%). Compared with that model, HT-LSTM achieves 7.3% higher test accuracy.

Model	CR	# Param.	Accuracy (%)
LSTM	1×	59M	33.2
TT-GRU	17,723×	3,328	80.0
TT-LSTM	17,388×	3,392	75.5
HT-LSTM (Ours)	**72,818**×	**810**	**88.1**

Table 3: Performance of different RNN compression work on Youtube celebrities face dataset using end-to-end direct training. CR stands for compression ratios. Results of TT-LSTM and TT-GRU are reported in [Yang *et al.*, 2017].

Model	Accuracy (%)
[Kim *et al.*, 2008]	71.2
[Harandi *et al.*, 2013]	73.9
[Ortiz *et al.*, 2013]	80.8
[Lu *et al.*, 2015]	78.5
HT-LSTM (Ours)	**88.1**

Table 4: Comparison between HT-LSTM using end-to-end direct training and other models without using tensor decomposition on Youtube celebrities face dataset.

3.2 Training with Pre-trained CNNs

Another set of our experiments is based on training strategy using pre-trained CNNs. To be specific, the pre-trained CNN first extracts the useful and compact features from the large-size raw input data, and then sends those captured features to RNN, As indicated in [Donahue *et al.*, 2015], using the front-end CNN can not only reduce the required input vector size of RNN, but can also significantly improve the overall performance of the entire CNN+RNN model.

Hyperparameter Setting. In this part of experiments dropout rate is set as 0.5. The L2 regularization with coefficient 0.0001 is used, and the entire HT-LSTM model is trained using ADAM optimizer with batch size 16.

UCF11 Dataset. Consistent with [Pan *et al.*, 2019], Inception-V3 is selected as the the front-end CNN feature extractor, whose output is a flattened size-2,048 feature vector. For our proposed HT-LSTM, this feature vector, as the input to the model, is reshaped to a tensor of size $8 \times 8 \times 8 \times 4$. Similarly, the output vector is reshaped to a tensor of size $4 \times 8 \times 8 \times 8$. In addition, the ranks for all the nodes are set as 4.

Table 5 summarizes the test accuracy of different models over UCF11 dataset. It is seen that with pre-trained CNN model as front-end feature extractor, HT-LSTM achieves 98.1% accuracy, which is 3.5% higher than the best reported accuracy from the state of the art. Compared with the TR-LSTM using the same front-end CNN, HT-LSTM achieves 4.3% accuracy increase. Meanwhile, being compressed from the same vanilla LSTM, HT-LSTM model brings very high compression ratio as 12,945; while the compression ratio of TR-LSTM is only 25.

HMDB51 Dataset. HMDB51 dataset [Kuehne *et al.*, 2011] contains 6,849 video clips that belong to 51 action categories, where each of them consists of more than 101 clips. Again, for the experiment on this dataset we use Inception-V3 as the pre-trained CNN model, whose extracted feature is flattened to a length-2,048 vector. Reshaped from this vector, the input tensor has size of $8 \times 8 \times 8 \times 4$. We also reshape the output vector to a tensor of size $4 \times 8 \times 8 \times 8$. Meanwhile, the ranks of all the nodes are set as 4.

Model	Accuracy (%)
[Wang *et al.*, 2015]	84.2
[Sharma *et al.*, 2015]	86.0
[Cho *et al.*, 2014]	88.0
[Gammulle *et al.*, 2017]	94.6
CNN + LSTM [Pan *et al.*, 2019]	92.3
CNN + TR-LSTM [Pan *et al.*, 2019]	93.8
CNN + HT-LSTM (Ours)	**98.1**

Table 5: Comparison between HT-LSTM using front-end pre-trained CNN and other related work on UCF11 dataset.

Table 6 summarizes the performance of CNN-aided HT-LSTM and other related work on this dataset. It is seen that HT-LSTM achieves 64.2% accuracy, which obtains 0.4% increase than the recent TR-LSTM. Meanwhile, the compression ratio brought by HT-LSTM is 12,945, which is much higher than the 25× parameter reduction in TR-LSTM.

It is worth noting that the state-of-the-art work [Carreira and Zisserman, 2017] achieves a higher accuracy (66.4%) than HT-LSTM. This performance is based on using two streams of input (RGB images and optical flow); while HT-LSTM does not utilize optical flow information of the video. When only RGB information of the video is sent to the models, HT-LSTM can achieve very competitive classification performance as compared to the prior work.

Model	Accuracy (%)
[Wang *et al.*, 2015]	63.2
[Feichtenhofer *et al.*, 2016]	56.8
[Carreira and Zisserman, 2017]	RGB + Flow: **66.4** RGB: 49.8
CNN + LSTM [Pan *et al.*, 2019]	62.9
CNN + TR-LSTM [Pan *et al.*, 2019]	63.8
CNN + HT-LSTM (Ours)	64.2

Table 6: Comparison between HT-LSTM using front-end pre-trained CNN and other related work on HMDB51 dataset.

4 Conclusion

In this paper, we propose a new RNN compression approach using Hierarchical Tucker (HT) decomposition. The HT-based RNN models exhibit strong hierarchical structure as well as low storage and computational costs. Our experiments on different datasets show that, our proposed HT-LSTM models significantly outperform the state-of-the-art compressed RNN models in terms of both compression ratio and test accuracy.

References

[Carreira and Zisserman, 2017] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

[Cho *et al.*, 2014] Jungchan Cho, Minsik Lee, Hyung Jin Chang, and Songhwai Oh. Robust action recognition using local motion and group sparsity. *Pattern Recognition*, 47(5):1813–1825, 2014.

[Donahue *et al.*, 2015] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2625–2634, 2015.

[Feichtenhofer *et al.*, 2016] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1933–1941, 2016.

[Gammulle *et al.*, 2017] Harshala Gammulle, Simon Denman, Sridha Sridharan, and Clinton Fookes. Two stream lstm: A deep fusion framework for human action recognition. In *IEEE Winter Conference on Applications of Computer Vision*, pages 177–186. IEEE, 2017.

[Hackbusch and Kühn, 2009] Wolfgang Hackbusch and Stefan Kühn. A new scheme for the tensor representation. *Journal of Fourier Analysis and Applications*, 15(5):706–722, 2009.

[Harandi *et al.*, 2013] Mehrtash Harandi, Conrad Sanderson, Chunhua Shen, and Brian C Lovell. Dictionary learning and sparse coding on grassmann manifolds: An extrinsic solution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3120–3127, 2013.

[Kim *et al.*, 2008] Minyoung Kim, Sanjiv Kumar, Vladimir Pavlovic, and Henry Rowley. Face tracking and recognition with visual constraints in real-world videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.

[Kuehne *et al.*, 2011] Hildegard Kuehne, Hueihan Jhuang, Estíbaliz Garrote, Tomaso Poggio, and Thomas Serre. Hmdb: a large video database for human motion recognition. In *International Conference on Computer Vision*, pages 2556–2563. IEEE, 2011.

[Liu *et al.*, 2009] Jingen Liu, Jiebo Luo, and Mubarak Shah. Recognizing realistic actions from videos "in the wild". In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1996–2003. IEEE, 2009.

[Lu *et al.*, 2015] Jiwen Lu, Gang Wang, Weihong Deng, Pierre Moulin, and Jie Zhou. Multi-manifold deep metric learning for image set classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1137–1145, 2015.

[Mikolov *et al.*, 2011] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5528–5531. IEEE, 2011.

[Nadav *et al.*, 2018] Cohen Nadav, Sharir Or, Levine Yoav, Tamari Ronen, Yakira David, and Shashua Amnon. Analysis and design of convolutional networks via hierarchical tensor decompositions. *arXiv preprint arXiv:1705.02302*, 2018.

[Ortiz *et al.*, 2013] Enrique G Ortiz, Alan Wright, and Mubarak Shah. Face recognition in movie trailers via mean sequence sparse representation-based classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3531–3538, 2013.

[Pan *et al.*, 2019] Yu Pan, Jing Xu, Maolin Wang, Jinmian Ye, Fei Wang, Kun Bai, and Zenglin Xu. Compressing recurrent neural networks with tensor ring for action recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4683–4690, 2019.

[Sharma *et al.*, 2015] Shikhar Sharma, Ryan Kiros, and Ruslan Salakhutdinov. Action recognition using visual attention. In *Neural Information Processing Systems: Time Series Workshop*, 2015.

[Song *et al.*, 2016] Han Song, Mao Huizi, and Dally William J. Deep compression: compressing deep neural network with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*, 2016.

[Sutskever *et al.*, 2011] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *International Conference on Machine Learning*, pages 1017–1024, 2011.

[Wang *et al.*, 2015] Limin Wang, Yu Qiao, and Xiaoou Tang. Action recognition with trajectory-pooled deep-convolutional descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4305–4314, 2015.

[Yang *et al.*, 2017] Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification. In *International Conference on Machine Learning*, pages 3891–3900. JMLR. org, 2017.

[Ye *et al.*, 2018] Jinmian Ye, Linnan Wang, Guangxi Li, Di Chen, Shandian Zhe, Xinqi Chu, and Zenglin Xu. Learning compact recurrent neural networks with block-term tensor decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9378–9387, 2018.

[Yu *et al.*, 2016] Haonan Yu, Jiang Wang, Zhiheng Huang, Yi Yang, and Wei Xu. Video paragraph captioning using hierarchical recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4584–4593, 2016.