Fast Training and Model Compression of Gated RNNs via Singular Value Decomposition

Rui Dai
Dept. Industrial Engineering
Tsinghua University
Beijing, China
dair17@mails.tsinghua.edu.cn

Lefei Li
Dept. Industrial Engineering
Tsinghua University
Beijing, China
lilefei@tsinghua.edu.cn

Wenjian Yu
BNRist, Dept. Computer Science & Tech.
Tsinghua University
Beijing, China
yu-wj@tsinghua.edu.cn

***Abstract*—Long Short-Term Memory (LSTM) network and Gated Recurrent Units (GRU) network are two widely-used gated Recurrent Neural Network (RNN) architectures. Both of them usually have a huge model size and require a long time to be trained. In this paper, we first propose a singular value decomposition (SVD) based approach for fast training of LSTM. Then, the factorized model and SVD based training approach are proposed for the GRU network, which adaptively choose the rank parameter for the matrix factorization model and reduce the training time and parameters of the gated RNNs. Experiments are carried out on the image classification and sentiment classification tasks using datasets MNIST and IMDB, respectively. The results show that the proposed LSTM-SVD approach achieves up to 3.9X speedup compared with training the original LSTM model, without loss of accuracy. The approaches for training the GRU network also have about 2X speedup. And, with the factorized models the quantity of RNN cell parameters can be significantly reduced by more than 10X.**

***Index Terms*—Gated recurrent neural networks (RNNs), model compression, singular value decomposition, training acceleration**

I. Introduction

As one of the most powerful deep neural networks (DNNs), the gated recurrent neural network (RNN), like LSTM [1] or GRU network [2], has shown promising results in many machine learning tasks, including language modeling [3], machine translation [4] and speech recognition [5]. The gated RNN usually involves millions of parameters and a large quantity of matrix multiplications both for the training and inference stages, so that its training and deployment require a lot of computation resource. To accelerate the training of large-scale DNNs and to compress the relevant model size, the approaches using GPU-based computing [6] and distributed computation [7] have been proposed. However, such high-performance computing equipments are costly, and are not always available. In fact, the hardware-based acceleration is not feasible for training the models used in intelligent household electrical appliance, mobile phones or other even smaller embedded devices. And, compression of the trained RNN model becomes crucial for the realization of edge computing.

Another line of work aims at reducing the amount of computation associated with the forward-propagation and the back-propagation process. Since floating-point arithmetic costs most time of training the deep-learning model, some work were focused on replacing floating-point multiplications by binary shifts [8] or integer shifts [9]. Lin et al. converted the multiplications to sign changes during forward propagation and performed quantized back-propagation that converts the remaining multiplication into binary shifts [10]. In [11] and [12], Han et al. proposed techniques including pruning the unimportant connections and then retraining the remaining sparse network, trained quantization and Huffman coding to accelerate the training procedure of the convolution neural network (CNN) and to reduce the storage of model. Their method also improves the energy efficiency and does not degrade the accuracy. Girshick applied truncated SVD to fully connected layers in region-based convolutional network (R-CNN) for faster detection [13]. Spring and Shrivastava proposed a randomized hashing approach to reduce the amount of computation in the training and testing procedures drastically [14].

Besides the approaches mentioned above, low-rank matrix approximation is another key that can be possibly used for the training acceleration and model compression. Denil et al. proposed a matrix-factorization based method to reduce the number of parameters in deep learning models by training only a small number of weights and predicting the rest [15]. Sindhwani et al. used structured matrix transformations with low-rank matrices to accelerate inferences and forward/backward pass [16]. Singular value decomposition (SVD) of a matrix produces the optimal low-rank approximation, and this has also been utilized for the training acceleration. In the fast learning model proposed by Cai et al. [17], SVD is applied to the multi-layer back-propagation (BP) network which has been pre-adjusted by supervised training, and then the analogous back-propagation is used for subsequent training of the factorized network. Finally, the factorized model is converted inversely to the BP network. It should be pointed out that there is few work for the fast training of gated RNNs. Most of existing work is focused on the BP network or CNNs.

For the gated RNNs, recently the training efficiency of LSTM networks was improved by leveraging factorization tricks [18]. One of the tricks is based on matrix factorization and is named F-LSTM (Factorized LSTM). In the approach, two low-rank matrices are initialized randomly to represent the weight matrix in LSTM network. This reduces the number

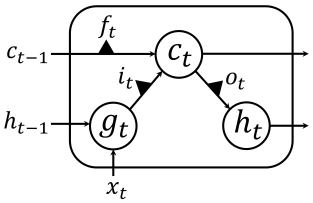
978-1-5090-6014-6/18/$31.00 ©2018 IEEE

of parameters and results in the acceleration of the training procedure.

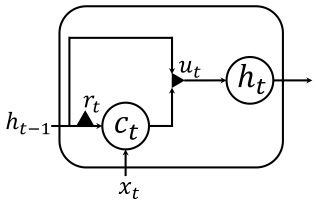
In this paper, a new training acceleration approach is proposed for the LSTM network, which combines the matrix factorization and the SVD trick used in [17]. It adaptively chooses the rank parameter according to the accuracy criterion of matrix approximation, and makes faster convergence of LSTM training than the approach in [18]. The matrix factorization based techniques are also proposed to accelerate the training of GRU networks. Experimental results on MNIST and IMDB datasets show that the proposed technique is superior to F-LSTM, and significantly reduces the training time of the original LSTM model (up to 3.9X). For the GRU network, the training time can also be reduced by about 2X. The proposed techniques also significantly compress the RNN model, with more than 10X parameter reduction shown in experiments.

II. Technical Background

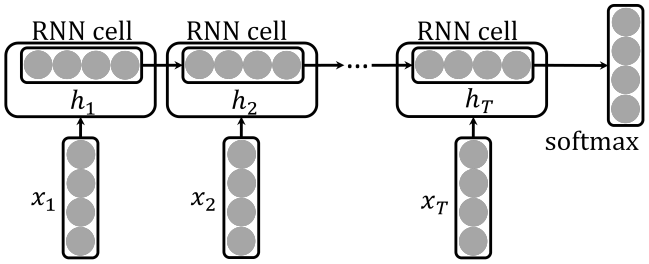
In this section, we briefly introduce the computation carried out for training LSTM network and GRU network, and the F-LSTM approach. $\sigma()$ and $\tanh()$ denote the sigmoid function and hyperbolic tangent function, respectively. Both work for vector. And, "$\odot$" stands for the operator of element-wise multiplication. The structures of an LSTM cell and a GRU cell are illustrated in Figure 1(a) and 1(b) respectively. An RNN classifier is illustrated in Figure 1(c), where the hidden state of the last time step is connected to a dense layer and softmax is used as the activation function.



(a) Illustration of an LSTM cell



(b) Illustration of a GRU cell



(c) The structure of an RNN classifier

Fig. 1: Illustration of RNN cells and an RNN classifier.

A. Long Short-Term Memory Network

The computations associated with the LSTM cell can be written in the following recurrent form:

$$LSTM : \boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1}, \boldsymbol{x}_t \rightarrow \boldsymbol{h}_t, \boldsymbol{c}_t, \tag{1}$$

where $\boldsymbol{x}_t \in \mathbb{R}^{n_i}$ is the input at time step t ($t = 1, 2, \ldots, T$), $\boldsymbol{h}_t \in \mathbb{R}^{n_h}$ is the corresponding hidden state and $\boldsymbol{c}_t \in \mathbb{R}^{n_h}$ is the corresponding memory. The states of the cell's gates before non-linearity activation at time step t are computed:

$$\boldsymbol{T}_t = \begin{bmatrix} \bar{\boldsymbol{i}}_t \\ \bar{\boldsymbol{f}}_t \\ \bar{\boldsymbol{o}}_t \\ \bar{\boldsymbol{g}}_t \end{bmatrix} = \boldsymbol{W}\boldsymbol{x}'_t + \boldsymbol{b} = \boldsymbol{W}\begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{h}_{t-1} \end{bmatrix} + \boldsymbol{b}, \tag{2}$$

where $\bar{\boldsymbol{i}}_t, \bar{\boldsymbol{f}}_t, \bar{\boldsymbol{o}}_t, \bar{\boldsymbol{g}}_t \in \mathbb{R}^{n_h}$, $\boldsymbol{W} \in \mathbb{R}^{4n_h \times (n_i + n_h)}$, and $\boldsymbol{b}$, $\boldsymbol{T}_t \in \mathbb{R}^{4n_h}$. Here n_i and n_h denote the dimensions of input sequences and hidden states, respectively. Then, at time step t the states of the input gate, forget gate, output gate and the new memory (denoted by $\boldsymbol{i}_t$, $\boldsymbol{f}_t$, $\boldsymbol{o}_t$ and $\boldsymbol{g}_t$ respectively) are computed:

$$\boldsymbol{i}_t = \sigma(\bar{\boldsymbol{i}}_t), \boldsymbol{f}_t = \sigma(\bar{\boldsymbol{f}}_t), \boldsymbol{o}_t = \sigma(\bar{\boldsymbol{o}}_t), \boldsymbol{g}_t = \sigma(\bar{\boldsymbol{g}}_t). \tag{3}$$

The final memory $\boldsymbol{c}_t$ and the hidden state $\boldsymbol{h}_t$ are computed as:

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{g}_t, \tag{4}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t). \tag{5}$$

B. Gated Recurrent Unit Network

Similar to the LSTM cell, the GRU cell has the following recurrent form:

$$GRU : \boldsymbol{h}_{t-1}, \boldsymbol{x}_t \rightarrow \boldsymbol{h}_t. \tag{6}$$

The cell's gates before nonlinearity activation at time step t are computed:

$$\boldsymbol{T}_t^g = \begin{bmatrix} \bar{\boldsymbol{r}}_t \\ \bar{\boldsymbol{u}}_t \end{bmatrix} = \boldsymbol{W}_g \boldsymbol{x}_t^g + \boldsymbol{b}_g = \boldsymbol{W}_g \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{h}_{t-1} \end{bmatrix} + \boldsymbol{b}_g, \tag{7}$$

where $\bar{\boldsymbol{r}}_t, \bar{\boldsymbol{u}}_t \in \mathbb{R}^{n_h}$, $\boldsymbol{x}_t^g \in \mathbb{R}^{n_i + n_h}$, $\boldsymbol{b}_g \in \mathbb{R}^{2n_h}$, and $\boldsymbol{W}_g \in \mathbb{R}^{2n_h \times (n_i + n_h)}$. The state of reset gate $\boldsymbol{r}_t = \sigma(\bar{\boldsymbol{r}}_t)$ and the state of update gate $\boldsymbol{u}_t = \sigma(\bar{\boldsymbol{u}}_t)$. The new memory is computed as:

$$\boldsymbol{T}_t^c = \boldsymbol{W}_c \boldsymbol{x}_t^c + \boldsymbol{b}_c = \boldsymbol{W}_c \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{r} \odot \boldsymbol{h}_{t-1} \end{bmatrix} + \boldsymbol{b}_c, \tag{8}$$

$$\boldsymbol{c}_t = \tanh(\boldsymbol{T}_t^c), \tag{9}$$

where $\boldsymbol{x}_t^c \in \mathbb{R}^{n_i + n_h}$, $\boldsymbol{W}_c \in \mathbb{R}^{n_h \times (n_i + n_h)}$, and $\boldsymbol{b}_c \in \mathbb{R}^{n_h}$. Finally, the hidden state is computed as

$$\boldsymbol{h}_t = (\boldsymbol{1} - \boldsymbol{u}_t) \odot \boldsymbol{c}_t + \boldsymbol{u}_t \odot \boldsymbol{h}_{t-1}, \tag{10}$$

where $\boldsymbol{1}$ denotes the vector with all 1 elements.

C. The F-LSTM Approach

To speed up the training of LSTM, Kuchaiev and Ginsburg proposed F-LSTM [18]. Its idea is to approximate matrix $\boldsymbol{W}$ in Eq. (2) with the product of two smaller matrices, i.e., $\boldsymbol{W} = \boldsymbol{W}_2 \cdot \boldsymbol{W}_1$. This makes the number of parameters reduced from $4n_h \times (n_i + n_h)$ to $(5n_h + n_i)r$, where r is the rank parameter. In F-LSTM, $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are initialized randomly before training.

The F-LSTM approach has the following shortages. Firstly, as $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are initialized randomly, the training procedure may converge slowly compared with other approach

which assigns better initial values to $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$. Secondly, it is not clear how to choose the parameter r. In this work, we propose a new fast training approach to overcome the above shortages. We will also extend the work to accelerate the training of GRU networks.

III. Methodology

Three training approaches for RNNs are proposed here: LSTM-SVD, Factorized GRU (F-GRU) and GRU-SVD. All of them can achieve a remarkable speedup and model-size compression without loss of performance.

A. The SVD Based Approach for Training LSTM

The SVD method [19] is one of the most powerful and widely-used tools for matrix factorization. For a weight matrix $\boldsymbol{W}_{(m\times n)}$ in an RNN model, the rank-r truncated SVD can be obtained by performing the full SVD and then keeping the r largest singular values and corresponding singular vectors.

$$\begin{aligned}\boldsymbol{W} &= \boldsymbol{U}_{(m\times m)}\cdot\boldsymbol{\Sigma}_{(m\times n)}\cdot\boldsymbol{V}^T_{(n\times n)}\\ &\approx \boldsymbol{U}_{(m\times r)}\cdot\left(\boldsymbol{\Sigma}_{(r\times r)}\cdot\boldsymbol{V}^T_{(r\times n)}\right)\\ &= \boldsymbol{W}_2\cdot\boldsymbol{W}_1=\boldsymbol{W}_f.\end{aligned} \tag{11}$$

From Eq. (11), we see that the truncated SVD can be expressed as the matrix factorization in form of $\boldsymbol{W}_2\cdot\boldsymbol{W}_1$. And, the result matrix $\boldsymbol{W}_f$ best approximates $\boldsymbol{W}$ among all matrices with rank r [20]. Suppose the singular values of $\boldsymbol{W}$ are $\sigma_1\geq\ldots\geq\sigma_n\geq 0$. The approximation error is

$$\frac{\|\boldsymbol{W}-\boldsymbol{W}_f\|_2}{\|\boldsymbol{W}\|_2}=\frac{\sigma_{r+1}}{\sigma_1}. \tag{12}$$

So, if we want to constrain the relative error of the approximation within ε, i.e.,

$$e_r=\frac{\|\boldsymbol{W}-\boldsymbol{W}_f\|_2}{\|\boldsymbol{W}\|_2}\leq\varepsilon, \tag{13}$$

the rank parameter r should be:

$$r^*=\min_{\sigma_{r+1}\leq\sigma_1\varepsilon} r. \tag{14}$$

This gives an adaptive strategy for choosing r value according to accuracy criterion.

Replacing $\boldsymbol{W}$ with $\boldsymbol{W}_2\cdot\boldsymbol{W}_1$, we obtain the factorized model of LSTM. To overcome the drawback brought by randomly initializing $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$, we instead pre-train the original model for several iterations, and then make truncated SVD on $\boldsymbol{W}$ to obtain the initial values of $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$. After that the factorized model is trained with BPTT algorithm. While doing the truncated SVD, Eq. (14) is used to choose the parameter r for the matrix factorization. This generates an SVD based approach for training LSTM (named LSTM-SVD), which includes the following steps.

- Step 1: Pre-train the LSTM network for several iterations, e.g., for 1 epoch;
- Step 2: Apply SVD to matrix $\boldsymbol{W}$ obtained from the pre-trained network, and then choose parameter r according to an accuracy ε and Eq. (14);
- Step 3: Train the factorized model with $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$, and finally obtain the converged factorized LSTM model.

For the factorized models, BPTT algorithm is still used for training. For LSTMs, the following can be inferred according to the chain rule:

$$\begin{aligned}\nabla_{\boldsymbol{T}_t}L&=\begin{bmatrix}\nabla_{\boldsymbol{i}_t}L\\ \nabla_{\boldsymbol{f}_t}L\\ \nabla_{\boldsymbol{o}_t}L\\ \nabla_{\boldsymbol{g}_t}L\end{bmatrix}\\ &=\begin{bmatrix}\nabla_{\boldsymbol{c}_t}L\odot\boldsymbol{g}_t\odot(\boldsymbol{1}-\boldsymbol{i}_t)\odot\boldsymbol{i}_t\\ \nabla_{\boldsymbol{c}_t}L\odot\boldsymbol{c}_{t-1}\odot(\boldsymbol{1}-\boldsymbol{f}_t)\odot\boldsymbol{f}_t\\ \nabla_{\boldsymbol{h}_t}L\odot\tanh(\boldsymbol{c}_t)\odot(\boldsymbol{1}-\boldsymbol{o}_t)\odot\boldsymbol{o}_t\\ \nabla_{\boldsymbol{c}_t}L\odot\boldsymbol{i}_t\odot\left(\boldsymbol{1}-\boldsymbol{g}_t^2\right)\end{bmatrix},\end{aligned} \tag{15}$$

where L is the cost function, and $\nabla_{\boldsymbol{v}}L$ stands for the gradient of L with respect to $\boldsymbol{v}$, i.e., $\nabla_{\boldsymbol{v}}L=\left[\frac{\partial L}{\partial v_1},\frac{\partial L}{\partial v_2},\ldots,\frac{\partial L}{\partial v_n}\right]^T$.

For convenience we denote:

$$\boldsymbol{L}'_t=\begin{cases}\sum_{s=t}^{T}\nabla_{\boldsymbol{h}_s}L\odot\nabla_{\boldsymbol{c}_t}\boldsymbol{h}_s & ,1\leq t\leq T\\ \boldsymbol{0} & ,t=T+1\end{cases}. \tag{16}$$

Here $\boldsymbol{1}$ and $\boldsymbol{0}$ stand for the vectors with all 1 and 0 elements, respectively. The BPTT algorithm for training the factorized LSTM model is illustrated in Algorithm 1. With the converged $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$, we do not convert them back to $\boldsymbol{W}$. And, they are used in the testing stage with a slight modification on the forward-propagation algorithm.

Algorithm 1 The BPTT Algorithm for the factorized model of LSTM

Input: $\boldsymbol{x}_t\in\mathbb{R}^{n_i},\boldsymbol{i}_t,\boldsymbol{f}_t,\boldsymbol{o}_t,\boldsymbol{g}_t,\boldsymbol{c}_t,\boldsymbol{h}_t\in\mathbb{R}^{n_h},(t=1,\ldots,T)$; $\boldsymbol{W}_1\in\mathbb{R}^{r\times(n_i+n_h)}$, $\boldsymbol{W}_2\in\mathbb{R}^{4n_h\times r}$, $\boldsymbol{b}\in\mathbb{R}^{4n_h}$; $\eta\in\mathbb{R}$

Output: $\Delta\boldsymbol{W}_1\in\mathbb{R}^{r\times(n_i+n_h)}$, $\Delta\boldsymbol{W}_2\in\mathbb{R}^{4n_h\times r}$, $\Delta\boldsymbol{b}\in\mathbb{R}^{4n_h}$

$lw1\leftarrow\boldsymbol{0}^{r\times(n_i+n_h)}$, $lw2\leftarrow\boldsymbol{0}^{4n_h\times r}$, $lb\leftarrow\boldsymbol{0}^{4n_h}$ {$lw1$ denotes $\nabla_{\boldsymbol{W}_1}L$, $lw2$ denotes $\nabla_{\boldsymbol{W}_2}L$, and lb denotes $\nabla_{\boldsymbol{b}}L$}
for $t\leftarrow T$ to 1 **do**
  $\nabla_{\boldsymbol{c}_t}L\leftarrow\nabla_{\boldsymbol{h}_t}L\odot\boldsymbol{o}_t\odot\left(\boldsymbol{1}-\tanh^2(\boldsymbol{c}_t)\right)+\boldsymbol{L}'_{t+1}$
  calculate $\nabla_{\boldsymbol{T}_t}L$ using Eq. (15)
  $lw1$ += $\boldsymbol{W}_2^T\cdot\nabla_{\boldsymbol{T}_t}L\cdot\boldsymbol{x}_t'^T$
  $lw2$ += $\nabla_{\boldsymbol{T}_t}L\cdot(\boldsymbol{W}_1\boldsymbol{x}'_t)^T$
  lb += $\nabla_{\boldsymbol{T}_t}L$
  $\boldsymbol{L}'_t\leftarrow\nabla_{\boldsymbol{c}_t}L\odot\boldsymbol{f}_t$
  $\nabla_{\boldsymbol{h}_{t-1}}L\leftarrow\left((\boldsymbol{W}_2\boldsymbol{W}_1)^T\cdot\nabla_{\boldsymbol{T}_t}L\right)[-n_h:]$ {"$[-n:]$" means the last n elements of a vector.}
end for
$\Delta\boldsymbol{W}_1\leftarrow\eta\cdot lw1$
$\Delta\boldsymbol{W}_2\leftarrow\eta\cdot lw2$
$\Delta\boldsymbol{b}\leftarrow\eta\cdot lb$
return $\Delta\boldsymbol{W}_1$, $\Delta\boldsymbol{W}_2$, $\Delta\boldsymbol{b}$.

B. The Factorization-Based Approaches for GRU

For the GRU network, both the factorization approach and the SVD based approach in last subsection can also be applied. They reduce the time for training the GRU network. The

difference to training LSTM is that there are two parameter matrices in GRU network.

The first matrix is $\boldsymbol{W}_g$ including $2n_h(n_i+n_h)$ parameters. So, if it is factorized as: $\boldsymbol{W}_g \approx \boldsymbol{W}_{gf} = \boldsymbol{W}_{g2} \cdot \boldsymbol{W}_{g1}$, where $\boldsymbol{W}_{g1} \in \mathbb{R}^{r_g \times (n_i+n_h)}$ and $\boldsymbol{W}_{g2} \in \mathbb{R}^{2n_h \times r_g}$. The number of parameters is cut down to $(n_i+3n_h)r_g$, where r_g is the rank parameter of $\boldsymbol{W}_g$. Hence, for the approach with factorized GRU model (denoted by F-GRU), the calculation of $\boldsymbol{T}_t^g$ in Eq. (7) changes to

$$\boldsymbol{T}_t^g = \boldsymbol{W}_{g2} \cdot (\boldsymbol{W}_{g1} \cdot \boldsymbol{x}_t^g) + \boldsymbol{b}_g. \tag{17}$$

The second weight matrix $\boldsymbol{W}_c$ can also be approximated by a rank-r_c matrix $\boldsymbol{W}_{cf} = \boldsymbol{W}_{c2} \cdot \boldsymbol{W}_{c1}$, where $\boldsymbol{W}_{c1} \in \mathbb{R}^{r_c \times (n_i+n_h)}$ and $\boldsymbol{W}_{c2} \in \mathbb{R}^{n_h \times r_c}$. Then, $\boldsymbol{T}_t^c$ in Eq. (8) changes to

$$\boldsymbol{T}_c^g = \boldsymbol{W}_{c2} \cdot (\boldsymbol{W}_{c1} \cdot \boldsymbol{x}_t^c) + \boldsymbol{b}_c. \tag{18}$$

Notice that $\boldsymbol{W}_c$ has $n_h(n_i+n_h)$ parameters and the factorization can reduce this number down to $(n_i+2n_h)r_c$. Compared with $\boldsymbol{W}_g$, this reduction seems not prominent. In practice, we can factorize both $\boldsymbol{W}_g$ and $\boldsymbol{W}_c$, but we prefer only do factorization for the larger matrix $\boldsymbol{W}_g$. Parameters of the F-GRU model are randomly initialized before training and the model is also trained with the BPTT method.

An alternative approach is named GRU-SVD, which has the same idea of LSTM-SVD. We first pre-train a conventional GRU model for several iterations. Then, we apply SVD to $\boldsymbol{W}_g$ or both $\boldsymbol{W}_g$ and $\boldsymbol{W}_c$, and each matrix is substitute by the product of two low-rank matrices. Thus the subsequent work is the same as the F-GRU approach. Finally, we can get the trained GRU-SVD model. Our experiment shows that the large singular values of $\boldsymbol{W}_g$ are more distinctive than those of $\boldsymbol{W}_c$. Therefore, a preferable strategy is applying SVD to $\boldsymbol{W}_g$ only while keeping the original $\boldsymbol{W}_c$ matrix in the GRU-SVD approach.

For GRUs, it's not hard to derive the following expressions:

$$\nabla_{\boldsymbol{u}_t} L = \nabla_{\boldsymbol{h}_t} L \odot (\boldsymbol{h}_{t-1} - \boldsymbol{c}_t), \tag{19}$$

$$\nabla_{\boldsymbol{r}_t} L = \left((\boldsymbol{W}_{c2}\boldsymbol{W}_{c1})^T \cdot \left(\boldsymbol{1} - \boldsymbol{c}_t^2\right) \odot \nabla_{\boldsymbol{c}_t} L\right)[-n_h:] \odot \boldsymbol{h}_{t-1}, \tag{20}$$

$$\nabla_{\boldsymbol{T}_t^g} L = \begin{bmatrix} \nabla_{\bar{\boldsymbol{u}}_t} L \\ \nabla_{\bar{\boldsymbol{r}}_t} L \end{bmatrix} = \begin{bmatrix} \nabla_{\boldsymbol{u}_t} L \odot \boldsymbol{u}_t \odot (\boldsymbol{1} - \boldsymbol{u}_t) \\ \nabla_{\boldsymbol{r}_t} L \odot \boldsymbol{r}_t \odot (\boldsymbol{1} - \boldsymbol{r}_t) \end{bmatrix}. \tag{21}$$

Here, "$[-n_h:]$" following Python notation, means the last n elements of a vector. The BPTT algorithm for a factorized GRU cell is illustrated in Algorithm 2.

Algorithm 2 The BPTT Algorithm for the factorized model of GRU

Input: $\boldsymbol{x}_t \in \mathbb{R}^{n_i}, \boldsymbol{r}_t, \boldsymbol{u}_t, \boldsymbol{c}_t, \boldsymbol{h}_t \in \mathbb{R}^{n_h} (t = 1, \ldots, T)$; $\boldsymbol{W}_{g1} \in \mathbb{R}^{r_g \times (n_i+n_h)}$, $\boldsymbol{W}_{g2} \in \mathbb{R}^{2n_h \times r_g}$, $\boldsymbol{b}_g \in \mathbb{R}^{2n_h}$, $\boldsymbol{W}_{c1} \in \mathbb{R}^{r_c \times (n_i+n_h)}$, $\boldsymbol{W}_{c2} \in \mathbb{R}^{n_h \times r_c}$, $\boldsymbol{b}_c \in \mathbb{R}^{n_h}$; $\eta \in \mathbb{R}$

Output: $\Delta\boldsymbol{W}_{g1} \in \mathbb{R}^{r_g \times (n_i+n_h)}$, $\Delta\boldsymbol{W}_{g2} \in \mathbb{R}^{2n_h \times r_g}$, $\Delta\boldsymbol{b}_g \in \mathbb{R}^{2n_h}$, $\Delta\boldsymbol{W}_{c1} \in \mathbb{R}^{r_c \times (n_i+n_h)}$, $\Delta\boldsymbol{W}_{c2} \in \mathbb{R}^{n_h \times r_c}$, $\Delta\boldsymbol{b}_c \in \mathbb{R}^{n_h}$

$lwg1 \leftarrow \boldsymbol{0}^{r_g \times (n_i+n_h)}$, $lwg2 \leftarrow \boldsymbol{0}^{2n_h \times r_g}$, $lbg \leftarrow \boldsymbol{0}^{2n_h}$

$lwc1 \leftarrow \boldsymbol{0}^{r_c \times (n_i+n_h)}$, $lwc2 \leftarrow \boldsymbol{0}^{n_h \times r_c}$, $lbc \leftarrow \boldsymbol{0}^{n_h}$ {lwg1, lwg2, lbg, lwc1, lwc2 and lbc denote $\nabla_{\boldsymbol{W}_{g1}} L$, $\nabla_{\boldsymbol{W}_{g2}} L$, $\nabla_{\boldsymbol{b}_g} L$, $\nabla_{\boldsymbol{W}_{c1}} L$, $\nabla_{\boldsymbol{W}_{c2}} L$, and $\nabla_{\boldsymbol{b}_c} L$ respectively}

for $t \leftarrow T$ to 1 **do**

  $\nabla_{\boldsymbol{c}_t} L \leftarrow \nabla_{\boldsymbol{h}_t} L \odot (\boldsymbol{1} - \boldsymbol{u}_t)$

  calculate $\nabla_{\boldsymbol{T}_t^g} L$ using Eq. (21)

  $lwg1$ += $\boldsymbol{W}_{g2}^T \cdot \nabla_{\boldsymbol{T}_t^g} L \cdot (\boldsymbol{x}_t^g)^T$

  $lwg2$ += $\nabla_{\boldsymbol{T}_t^g} L \cdot (\boldsymbol{W}_{g1}\boldsymbol{x}_t^g)^T$

  lbg += $\nabla_{\boldsymbol{T}_t^g} L$

  $lwc1$ += $\boldsymbol{W}_{c2}^T \cdot \nabla_{\boldsymbol{c}_t} L \odot \left(\boldsymbol{1} - \boldsymbol{c}_t^2\right) \cdot (\boldsymbol{x}_t^c)^T$

  $lwc2$ += $\nabla_{\boldsymbol{c}_t} L \odot \left(\boldsymbol{1} - \boldsymbol{c}_t^2\right) \cdot \left(\boldsymbol{W}_{c1}^T \boldsymbol{x}_t^c\right)^T$

  lbc += $\nabla_{\boldsymbol{c}_t} L \odot \left(\boldsymbol{1} - \boldsymbol{c}_t^2\right)$

  $\nabla_{\boldsymbol{h}_{t-1}} L \leftarrow \left((\boldsymbol{W}_{g2}\boldsymbol{W}_{g1})^T \cdot \nabla_{\boldsymbol{T}_t^g} L\right)[-n_h:]$

end for

$\Delta\boldsymbol{W}_{g1} \leftarrow \eta \cdot lwg1$, $\Delta\boldsymbol{W}_{g2} \leftarrow \eta \cdot lwg2$, $\Delta\boldsymbol{b}_g \leftarrow \eta \cdot lbg$

$\Delta\boldsymbol{W}_{c1} \leftarrow \eta \cdot lwc1$, $\Delta\boldsymbol{W}_{c2} \leftarrow \eta \cdot lwc2$, $\Delta\boldsymbol{b}_c \leftarrow \eta \cdot lbc$

return $\Delta\boldsymbol{W}_{g1}$, $\Delta\boldsymbol{W}_{g2}$, $\Delta\boldsymbol{b}_g$, $\Delta\boldsymbol{W}_{c1}$, $\Delta\boldsymbol{W}_{c2}$, $\Delta\boldsymbol{b}_c$.

IV. Experiments

The proposed approaches have been tested on two tasks - MNIST classification task and IMDB sentiment classification task. Each of the two tasks corresponds to a publicly available dataset, whose statistics are summarized in Table I. The training approaches for LSTM and GRU networks are implemented based on TensorFlow [21]. In the SVD based approach, the relative error constraint ε is set to 0.2, except explicitly stated. All experiments were run on a machine equipped with a 16-core Dual-Core AMD Opteron™ Processor 8224 SE and 64 GB memory.

TABLE I: Sizes of the datasets

Dataset	Training size	Validation size	Test size
MNIST	55,000	5,000	10,000
IMDB	30,000	10,000	10,000

A. MNIST Classification Task

MNIST is a large dataset including images of handwritten digits, and each image is represented by a 28×28 pixel matrix of gray-scale values. MNIST has been studied in many image classification tasks [22]. Considering each image as a sequence of pixel rows, each image can be handled as a sequence of vectors with 28 steps. We use the cross-entropy loss as the cost function and Adam optimizer [23] with a learning rate decay is used for training. Specifically, the initial learning rate is 0.0009 which decays by a factor of 0.95 per 100 steps.

We first test the approaches using LSTM network. Each image can be handled as a sequence of vectors with $T = 28$ steps and the input size $n_i = 28$. The size of hidden state is $n_h = 768$. With the SVD based training approach (LSTM-SVD), we first pre-train the LSTM network for one epoch.

Some leading singular values of weight matrix $\boldsymbol{W}$ after the pre-training are shown in Figure 2. Compared with the initial random weight matrix, there is distinct decay on the singular values of the pre-adjusted $\boldsymbol{W}$. Thus, the pre-adjusted $\boldsymbol{W}$ can be well approximated by a low-rank matrix. With the preset ε and (14), we can calculate the value of rank parameter r^*. In this experiment, $r^* = 48$. Then, with the factorized LSTM model, the number of parameters in the LSTM cell is reduced from 2,448,384 to 185,664 (13X reduction), and the total number of parameters is reduced from 2,456,074 to 193,354 (**13X**).

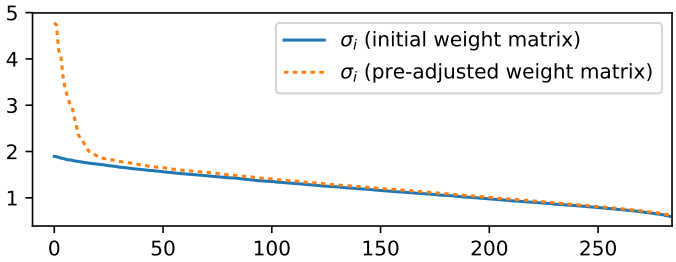


Fig. 2: Distributions of leading singular values of the initial random weight matrix $\boldsymbol{W}$ and the pre-adjusted $\boldsymbol{W}$ in the MNIST classification task.

With the same low-rank factorization, we then train the network with the F-LSTM approach. In Figure 3, we show the training loss of LSTM-SVD, F-LSTM and the conventional LSTM. From it we can see that at the same step count LSTM-SVD gains lower loss than F-LSTM even though they both have same number of parameters. More results of the three approaches are listed in Table II. For the training stage, we use the early stopping strategy for regularization, so that the training is stopped if the validation accuracy is not improved for 4 epochs. From the table, we see that the approaches converge with different speed. LSTM-SVD needs less epoch than F-LSTM for reaching convergence, which makes the latter's costs 13% more training time than the former. Compared with the conventional LSTM model, the LSTM-SVD approach reduces the training time for **3.9X**, while keeping same accuracy. For the testing stage, the model obtained with LSTM-SVD has better test accuracy than that corresponding to the F-LSTM approach.

TABLE II: Comparisons of LSTM, F-LSTM and LSTM-SVD approaches in the MNIST classification task.

Approach	LSTM	F-LSTM	LSTM-SVD
Training accuracy	100.0%	99.5%	100.0%
Test accuracy	99.0%	98.4%	98.8%
Training time (min.)	229.3	65.6	58.1
Epoch number	15	19	15
Testing time (sec.)	23.9	12.4	12.0

Then, we test the approaches using GRU network. The size of hidden state is still $n_h = 768$. In a GRU cell, for weight matrices $\boldsymbol{W}_c$ and $\boldsymbol{W}_g$, the distributions of the singular values before and after the pre-training step are shown in Figure 4. For each initial weight matrix, the singular values are relatively equally distributed. After the pre-training, more distinct decay of singular value is exhibited for the both matrices. Between the both, the decay trend of singular value for the pre-adjusted $\boldsymbol{W}_g$ is obviously more prominent than the pre-adjusted $\boldsymbol{W}_c$. This indicates $\boldsymbol{W}_g$ should be factorized preferentially.

In the experiment, we just factorize the matrix $\boldsymbol{W}_g$, and the GRU-SVD approach automatically choose a rank parameter $r_g^* = 103$. With the same low-rank factorization, the F-GRU approach is then performed. Different from LSTM-based approach, with the factorized GRU model, the training loss converges with same speed as the conventional GRU model. The experimental results also indicate that the GRU-SVD approach does not outperform the F-GRU approach on the convergence rate.

In Table III, the results of the GRU-based approaches in the MNIST classification task are listed. From it we see that, the F-GRU approach achieves the same test accuracy as the conventional GRU model, but costing only 69.7% training time. With the factorized model, the testing time can also be reduced to 69.1% of the original model. The performance of the GRU-SVD is close to that of F-GRU, showing 1.3X reduction of the training time of the conventional GRU model. The number of parameters is also reduced if the F-GRU or GRU-SVD approaches are used, as listed in Table III. It should be pointed out more speedup and model compression can be expected providing that both $\boldsymbol{W}_g$ and $\boldsymbol{W}_c$ are factorized with lower ranks. For example, the F-GRU approach with rank-60 factorizations for both $\boldsymbol{W}_g$ and $\boldsymbol{W}_c$ can achieve **2.1X** speedup while keeping high performance (98.7% test accuracy). This corresponds to a **7.6X** compression rate of the model size.

B. IMDB Sentiment Classification Task

Large Movie Review Dataset contains 50,000 highly polar movie reviews from IMDB and is widely used for sentiment classification tasks [24]. Top 20,000 words in this corpus are used to build the vocabulary dictionary. Natural Language Toolkit (NLTK) [25] is used for tokenization and sentences are truncated or padded to the length of 200.

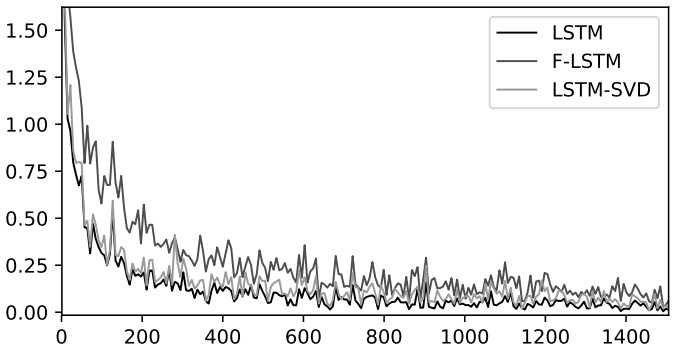


Fig. 3: Training loss versus training steps (mini-batch) for the F-LSTM, LSTM and LSTM-SVD approaches in the MNIST classification task

TABLE III: Comparisons of GRU, F-GRU and GRU-SVD in the MNIST classification task.

Approach	Training accuracy	Test accuracy	Training time (min.)	Testing time (sec.)	num of parameters
GRU	100.0%	98.9%	111.2	16.6	1,843,978
F-GRU	100.0%	98.9%	77.5	11.5	861,518
GRU-SVD	100.0%	99.0%	80.5	11.8	861,518

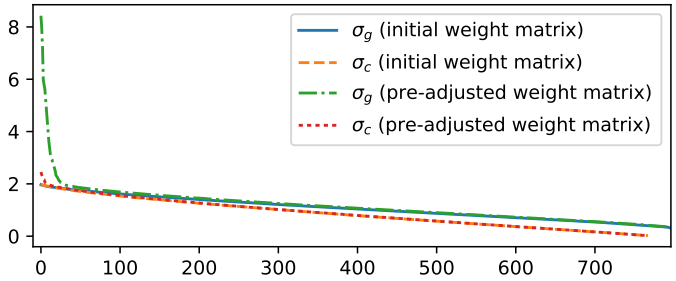


Fig. 4: Distributions of singular values in the GRU-SVD model for MNIST classification task.

In this binary classification task, whose output is either positive or negative, an embedding layer is used before the RNN cell. Dropout is used to avoid overfitting [26]. And, the binary cross-entropy is employed as the cost function. In our experiments, we set hidden size of 768 ($n_h = 768$), embedding size of 32 ($n_i = 32$) and dropout rate 0.5, while the Adam optimizer is used to train the model. The initial learning rate is 0.001, and it decays by a factor of 0.6 each epoch until reaching 0.0001.

We first test the LSTM based approaches. After pre-training the network for 1 epoch, the LSTM-SVD automatically chooses the rank parameter $r^* = 11$ (corresponding to $\varepsilon = 0.2$). Note that the singular value of $\boldsymbol{W}$ here presents a more sharp drop than that for the MNIST classification task. With the same setting for low-rank factorization, F-LSTM is also used to train the network. And, the early stopping is used for regularization and the training iteration stops when there is no improvement of validation accuracy for 2 epochs. Their results are listed in Table IV, compared with the results from the conventional LSTM.

TABLE IV: Comparison of LSTM, F-LSTM and LSTM-SVD in the IMDB sentiment classification task.

Approach	LSTM	F-LSTM	LSTM-SVD
Training accuracy	97.5%	96.0%	96.0%
Test accuracy	86.2%	86.6%	87.0%
Training time (min.)	577.8	241.5	186.9
Epoch number	11	11	7
Testing time (sec.)	180.0	88.7	89.3
Parameter number	3,102,210	687,202	687,202
RNN parameter number	2,460,672	45,664	45,664

The results show that both F-LSTM and LSTM-SVD achieve better evaluation accuracy than LSTM. The reason might be that overfitting is alleviated by factorization. F-LSTM and LSTM-SVD leads to 2.9X and **3.6X** reduction of training time respectively, compared with the LSTM model. The number of parameters also reduces to 22.2% of the original LSTM model. Note that this proportion is a little larger in comparison with the MNIST task, where the size of the factorized model is only 7.9% of the original LSTM model. The reason is the existence of the embedding layer, which also has a large quantity of parameters. Focusing on the number of parameters within the RNN cell, we can see that the factorized cell has 54X fewer parameters.

Compared with the MNIST task, the pre-adjusted GRU-SVD model in the IMDB task demonstrate different distributions of singular values. Both $\boldsymbol{W}_g$ and $\boldsymbol{W}_c$ have a few large singular values that are very distinctive, as illustrated in Figure 5. Hence we factorize both $\boldsymbol{W}_g$ and $\boldsymbol{W}_c$ in F-GRU and GRU-SVD approaches. Given $\varepsilon = 0.2$, we have $r_g^* = 99$ and $r_c^* = 315$.

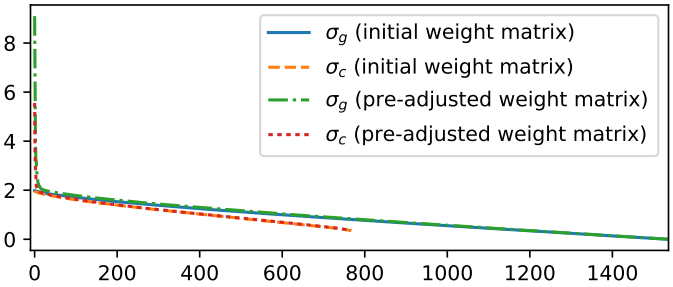


Fig. 5: Distributions of singular values in GRU-SVD model for IMDB sentiment classification task.

The computational results of the GRU-based approaches are listed in Table V. Similar to the results on the MNIST classification task, the results indicate that F-GRU performs a little bit better than GRU-SVD. The F-GRU model achieves the same test accuracy as the original GRU model, but only costs 72.0% training time. The GRU-SVD approach has comparable performance as F-GRU. If manually setting the rank parameters, we find out that F-GRU could perform even better. For example, if setting the rank parameters for $\boldsymbol{W}_g$ and $\boldsymbol{W}_c$ to 15 and 30 respectively, the training time can be reduced to 247 minutes (2.1X reduction if compared with the original GRU). And, the total number of parameters can be reduced to 29.2% of that in the original GRU.

V. Conclusions

The training acceleration and model compression of gated RNNs is explored in this work. With the ideas of matrix factorization and SVD, several approaches are proposed for reducing the training time of LSTM and GRU networks. Among them, LSTM-SVD automatically chooses the rank parameter and results in up to **3.9X** speedup for the training of LSTM

TABLE V: Comparison of different GRU-based approaches for the IMDB sentiment classification task.

Approach	Training accuracy	Test accuracy	Training time (min.)	Testing time (sec.)	Total number of parameters	Parameters in GRU cell
GRU	99.5%	86.5%	521.5	123.5	2,487,042	1,845,504
F-GRU	98.0%	86.5%	375.6	86.3	1,369,026	727,488
GRU-SVD	97.5%	86.8%	390.4	85.6	1,369,026	727,488

network in the MINST classification and IMDB sentiment classification problems. It outperforms the existing F-LSTM approach in terms of runtime and accuracy. F-GRU and GRU-SVD approaches are for the training of GRU network and bring about 2X reduction of the training time, as shown in the experiments. With some setting of rank parameters, F-GRU performs better and achieves up to 2.1X speedup. The developed approaches induce less error, and can also compress the RNN model by up to **13X**.

An interesting phenomenon is that the obtained factorized models often show better performance on the testing set. A possible reason is that the factorization has the effect of regularization and thus alleviates overfitting. This remains a topic of future research. Besides, how to further improve the proposed technique and make it adapted to actual edge-computing scenarios will be explored in the future.

ACKNOWLEDGMENT

This work was partially supported by the National Key Research and Development Program of China (grant No. 2016YFF0204100). W. Yu is the corresponding author.

REFERENCES

[1] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[2] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv:1412.3555*, 2014.

[3] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM neural networks for language modeling," in *Proceedings of Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[4] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv:1409.0473*, 2014.

[5] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2013),*. IEEE, 2013, pp. 6645–6649.

[6] Z. Luo, H. Liu, and X. Wu, "Artificial neural network computation on graphic process unit," in *Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN'05)*, vol. 1. IEEE, 2005, pp. 622–626.

[7] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1223–1231.

[8] H. K. Kwan and C. Tang, "Multiplierless multilayer feedforward neural network design suitable for continuous input-output mapping," *Electronics Letters*, vol. 29, no. 14, pp. 1259–1260, 1993.

[9] E. L. Machado, C. J. Miosso, R. von Borries, M. Coutinho, P. d. A. Berger, T. Marques, and R. P. Jacobi, "Computational cost reduction in learned transform classifications," *arXiv:1504.06779*, 2015.

[10] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural networks with few multiplications," *arXiv:1510.03009, in Proc. ICLR 2016*, 2015.

[11] 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.

[12] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv:1510.00149, in Proc. ICLR 2016*, 2015.

[13] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2015, pp. 1440–1448.

[14] R. Spring and A. Shrivastava, "Scalable and sustainable deep learning via randomized hashing," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 445–454.

[15] 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.

[16] V. Sindhwani, T. Sainath, and S. Kumar, "Structured transforms for small-footprint deep learning," in *Advances in Neural Information Processing Systems*, 2015, pp. 3088–3096.

[17] C. Cai, D. Ke, Y. Xu, and K. Su, "Fast learning of deep neural networks via singular value decomposition," in *Proceedings of Pacific Rim International Conference on Artificial Intelligence*. Springer, 2014, pp. 820–826.

[18] O. Kuchaiev and B. Ginsburg, "Factorization tricks for LSTM networks," *arXiv:1703.10722, in Proc. ICLR 2017*, 2017.

[19] V. Klema and A. Laub, "The singular value decomposition: Its computation and some applications," *IEEE Transactions on Automatic Control*, vol. 25, no. 2, pp. 164–176, 1980.

[20] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936.

[21] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv:1603.04467*, 2016.

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[23] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, 2014.

[24] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. [Online]. Available: http://www.aclweb.org/anthology/P11-1015

[25] S. Bird, "NLTK: the natural language toolkit," in *Proceedings of the COLING/ACL on Interactive presentation sessions*. Association for Computational Linguistics, 2006, pp. 69–72.

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