University of California

Santa Barbara

Adaptive-rank Fusion for Sentiment Analysis: end-to-end training and generalization behavior

A Thesis submitted in partial satisfaction of the
requirements for the degree Master of Science
in Electrical and Computer Engineering

by

Keon Yong Lee

Committee in charge:

Professor Zheng Zhang, Chair

Professor Kenneth Rose

Professor B.S. Manjunath

September 2022

The thesis of Christian Lee is approved.

_____

B.S. Manjunath

_____

Kenneth Rose

_____

Zheng Zhang, Committee Chair

May 2022

ABSTRACT

Adaptive-rank Fusion for Sentiment Analysis:

end-to-end training and generalization behavior

by

Keon Yong Lee

In multimodal sentiment analysis, a model learns to predict one's opinion towards a certain entity after observing data from multiple sources. As humans merge information from all five senses to read someone, the model needs a proper fusion mechanism to utilize the data from multiple sources. Tensor fusion networks have been proposed and have shown to perform well for multimodal sentiment analysis. However, it suffers from the curse of dimensionality because they use the full tensor format. As a solution, low-rank fusion networks have been proposed, but they require us to pre-determine the rank, which is an ill-posed problem. Instead, we propose adaptive-rank fusion networks that can learn the rank in a data-driven way. We compare the three models in terms of accuracy, complexity, and runtime. Then, we test how adaptive-rank fusion networks react to being trained with a subset of the original training dataset.

1 Introduction

Deep neural networks are demonstrating the states of art performance in many machine learning tasks such as computer vision and natural language processing. However, billions of model parameters in modern deep neural networks are becoming problems in training and deploying on devices with limited-resources such as mobile phones. Additionally, every floating point operations (FLOPs) use electrical power and potentially release some carbon, causing environmental concerns as well. Therefore, the motivation to train smaller networks that can perform comparably to the larger networks is higher than ever.

A tensor fusion network is a deep neural network that is designed to process input features from multiple modalities [17]. Humans express their feelings through various channels including voice and facial expressions. Therefore, tensor fusion networks that can utilize information from multiple input channels through tensor fusion have demonstrated excellent performance in the machine learning task of understanding human opinions. A downside of tensor fusion networks is that they suffer from the curse of dimensionality, which worsens as the number of input modalities increases [6]. The low-rank fusion network is proposed to remedy the curse, but the rank of its fusion layer should be pre-determined, which is an ill-posed problem [3],[12].

The low-rank tensor decomposition of the model parameters is one of many proposed compression methods for the deep neural networks [1],[2],[6],[7],[9],[10]. Most low-rank compression algorithms for deep neural networks approximate the model parameters of the pre-trained models into low-rank factors using tensor decomposition to remove redundancies in the parameters while compromising on generalization performance [7]. We can set the rank of tensor decomposition to control the trade-off between the accuracy and the complexity. Can we rather learn the rank of tensor decomposition from data in an end-to-end training fashion? [2] proposed a training algorithm with a Bayesian approach to the automatic rank determination for tensorized deep neural networks. We propose adaptive-rank fusion networks and modify and adopt the training algorithm to learn the rank of fusion layer through a data-driven approach.

We compare adaptive-rank fusion networks, tensor fusion networks, and low-rank fusion networks in terms of accuracy, number of parameters, training time, and test time. Interestingly, compressed adaptive-rank fusion networks perform better than uncompressed tensor fusion networks. Moreover, we test the generalization performance of adaptive-rank fusion networks and witness that, compared to other two networks, they are more immune to being trained with a subset of original training dataset. Finally, we try to find the explanation for these findings from the information theory.

2 Background

2.1 Tensor

Tensors are multi-way collections of numbers that can be thought of as matrices extended to higher dimensions [3],[12]. The number of dimension of a tensor is called the order, also known as ways or modes. Therefore, scalars can be considered as zeroth-order tensors, vectors as first-order tensors, and matrices as second-order tensors. Tensors with three or more modes are referred to as higher-order tensors. The scalars are denoted by lower case letters, e.g., a, the vectors by boldface low case letters, e.g., $\mathbf{a}$, the matrices by boldface upper case letters, e.g., $\mathbf{A}$, and the higher-order tensors by boldface Euler script, e.g., $\boldsymbol{\mathcal{A}}$. The i-th element of the vector is denoted by $\mathbf{a}(i)$, the (i,j)-th element of the matrix by $\mathbf{A}(i,j)$, and the (i,j,k)-th element of the tensor by $\boldsymbol{\mathcal{A}}(i,j,k)$ given that $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I \times J \times K}$. Fibers refer to the one-dimensional sub-arrays of tensors and slices represent the two-dimensional sub-arrays of tensors. For the 3-way tensor $\boldsymbol{\mathcal{A}}$, $\boldsymbol{\mathcal{A}}(:,j,k)$ is a mode-1 fiber, $\boldsymbol{\mathcal{A}}(i,:,k)$ is a mode-2 fiber, and $\boldsymbol{\mathcal{A}}(i,j,:)$ is a mode-3 fiber. Its slices are given as $\boldsymbol{\mathcal{A}}(:,:,k)$, $\boldsymbol{\mathcal{A}}(:,j,:)$, and $\boldsymbol{\mathcal{A}}(i,:,:)$ [3],[12]. More generally, for N-way tensors, their n-way subsections are created by fixing all but n indices.

2.2 Tensor Computations

Let us go over some tensor computations that are used in this paper.

2.2.1 Tensor inner Product

The inner product between an N-way tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \cdots \times I_N}$ and an M-way tensor $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{I_1 \times \cdots \times I_N \times \cdots \times I_M}$ is defined as

$$\boldsymbol{\mathcal{A}} \cdot \boldsymbol{\mathcal{B}} = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} \boldsymbol{\mathcal{A}}(i_1, ..., i_N) \boldsymbol{\mathcal{B}}(i_1, ..., i_N, :, ..., :) \tag{1}$$

where the resulting tensor is an $(M-N)$-way tensor in $\mathbb{R}^{I_{N+1} \times \cdots \times I_M}$. Therefore, the result is a scalar if $M = N$.

2.2.2 Tensor-matrix Product

The n-mode product between an N-way tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times \cdots \times I_N}$ and a matrix $\mathbf{B} \in \mathbb{R}^{J \times I_n}$ is denoted as $\boldsymbol{\mathcal{A}} \times_n \mathbf{B}$ where its output is an N-way tensor in $\mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N}$ [3],[12].

Format	CP	TT	Full
Memory Cost	$R\sum_{n=1}^{N} I_n$	$\sum_{n=1}^{N} R_{n-1} I_n R_n$	$\prod_{n=1}^{N} I_n$

Table 1: The memory cost of storing $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ in CP format with the CP rank of R (CP), in TT format with the TT rank of $(R_0, R_1, ..., R_N)$ (TT), and in full tensor format (Full).

Elementwise, it is given as

$$(\boldsymbol{\mathcal{A}} \times_n \mathbf{B})(i_1, ..., i_{n-1}, j, i_{n+1}, ..., i_N) = \sum_{i_n=1}^{I_n} \boldsymbol{\mathcal{A}}(i_1, ..., i_N)\mathbf{B}(j, i_n). \tag{2}$$

Intuitively, each mode-n fiber of $\boldsymbol{\mathcal{A}}$ is multiplied by $\mathbf{B}$. Note that if $J = 1$, this becomes a tensor-vector product that results in an $(N-1)$-way tensor in $\mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N}$.

2.3 Tensor Decomposition

The motivation behind tensor decomposition comes from matrix factor analysis, which breaks down a matrix into two factor matrices that contain latent information [12]. Formally, a matrix $\mathbf{M} \in \mathbb{R}^{I \times J}$ can be described by two matrices $\mathbf{A} \in \mathbb{R}^{I \times R}$ and $\mathbf{B} \in \mathbb{R}^{J \times R}$ as $\mathbf{M} = \mathbf{A}\mathbf{B}^\top$, where R is the number of latent variables or the rank of $\mathbf{M}$. This type of matrix factorization is called the matrix rank decomposition. Additionally, $\mathbf{M}$ can be approximated by or compressed to $\hat{\mathbf{M}}$ with a low-rank [12].

2.3.1 Rank-1 Tensor

An N-way tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is a rank-1 tensor if it can be expressed as the outer product of N vectors:

$$\boldsymbol{\mathcal{A}} = \bigotimes_{n=1}^{N} \mathbf{u}^{(n)} \tag{3}$$

where $\mathbf{u}^{(n)} \in \mathbb{R}^{I_n}$ are called the factor vectors [3],[12]. Elementwise,

$$\boldsymbol{\mathcal{A}}(i_1, ..., i_N) = \mathbf{u}^{(1)}(i_1) \cdots \mathbf{u}^{(N)}(i_N). \tag{4}$$

Note that a rank-1 tensor can be stored with linearly increasing $\sum_{n=1}^{N} I_n$ compared to the exponentially increasing $\prod_{n=1}^{N} I_n$ elements of its full tensor form.

2.3.2 CP Decomposition

CP decomposition expresses an N-way tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ as the sum of R, the rank-1

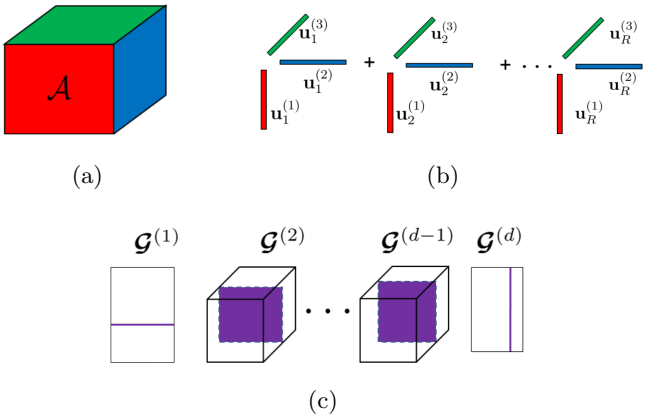


Figure 1: The representations of (a) full tensor, (b) CP, and (c) TT decomposition. In (b), the vectors in different colors represent $\mathbf{U}^{(n)}(:, r)$ from different factor matrices. In (c), the purple lines and squares indicate different $\boldsymbol{\mathcal{G}}^{(n)}(:, i_n, :)$ (original figures from [2]).

tensors [3],[12]:

$$\boldsymbol{\mathcal{A}} = \sum_{r=1}^{R} \bigotimes_{n=1}^{N} \mathbf{U}^{(n)}(:, r) \tag{5}$$

where R is called the CP rank of $\boldsymbol{\mathcal{A}}$ and $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R}$. A tensor in CP format can be stored with $R\sum_{n=1}^{N} I_n$ elements (Table 1).

The minimum value of R that achieves equation (5) is called the rank of $\boldsymbol{\mathcal{A}}$ [3],[12]. Determining the rank of a given tensor is an NP-hard problem. In addition, CP decomposition at a fixed CP rank can be ill posed. Therefore, CP decomposition algorithms are not guaranteed to converge for a given tensor [3],[11],[12].

2.3.3 TT Decomposition

The TT decomposition of an N-way tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \cdots \times I_N}$ expresses its elements as a train of matrix products:

$$\boldsymbol{\mathcal{A}}(i_1, ..., i_N) = \boldsymbol{\mathcal{G}}^{(1)}(:, i_1, :)\boldsymbol{\mathcal{G}}^{(2)}(:, i_2, :) \cdots \boldsymbol{\mathcal{G}}^{(N)}(:, i_N, :) \tag{6}$$

where $\boldsymbol{\mathcal{G}}^{(n)}(:, i_n, :)$ is an $R_{n-1} \times R_n$ slice of the mode-n factor tensors (cores) $\boldsymbol{\mathcal{G}}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times R_n}$. A tuple $(R_0, ..., R_N)$ is called the TT-rank of $\boldsymbol{\mathcal{A}}$, where by default $R_0 = R_n = 1$, and its maximum is called the maximal TT rank of $\boldsymbol{\mathcal{A}}$. A tensor in TT format can be stored with with $\sum_{n=1}^{N} R_{n-1} I_n R_n$ elements (Table 1) [1],[9],[11].

TT decomposition is a generalization of matrix rank decomposition to tensors, and a TT decomposition algorithm based on singular value decomposition is very stable compared to the CP decomposition algorithms [1],[9],[11].

2.4 Tensorized Neural Network

Modern neural networks contain more than billion parameters which lead to high storage and computational cost [2],[7]. With the increased focus on deployment on edge devices, the incentive to compress neural networks has grown in recent years. There are various approaches to reduce the size of neural networks such as weight sharing, network pruning, knowledge distillation, quatization, and low-rank tensorization [7]. The goal of tensorizing neural networks is to decompose their weights into low-rank factors [2],[7] and to directly use the low-rank factors during forward propagation [1],[9].

2.4.1 Tensorized Linear Layer

A linear layer is defined as

$$\mathbf{h} = \mathbf{z} \cdot \mathbf{W} + \mathbf{b} \tag{7}$$

where $\mathbf{h} \in \mathbb{R}^J$ is an output, $\mathbf{z} \in \mathbb{R}^I$ is an input, $\mathbf{W} \in \mathbb{R}^{I \times J}$ is a weight, and $\mathbf{b} \in \mathbb{R}^J$ is a bias. To tensorize the linear layer, we reshape $\mathbf{z}$ into $\boldsymbol{\mathcal{Z}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ and $\mathbf{W}$ into $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{I_1 \times \cdots \times \cdots I_d \times J_1 \times \cdots \times \cdots \times J_d}$ and $\mathbf{b}$ into $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{J_1 \times \cdots \times J_d}$ where $I = \prod_{n=1}^{d} I_n$ and $J = \prod_{n=1}^{d} J_n$. The reshaping involves two bi-injective mapping functions

$$u : \mathbb{R}^I \leftrightarrow \mathbb{R}^{I_1 \times \cdots \times I_d} \tag{8}$$

$$v : \mathbb{R}^J \leftrightarrow \mathbb{R}^{J_1 \times \cdots \times J_d} \tag{9}$$

that uniquely map i to $(i_1, ..., i_d)$ and j to $(j_1, ..., j_d)$, and visa-versa. Then, the output of tensorized linear layer is given as

$$\boldsymbol{\mathcal{H}} = \boldsymbol{\mathcal{Z}} \cdot \boldsymbol{\mathcal{W}} + \boldsymbol{\mathcal{B}} \tag{10}$$

where $\boldsymbol{\mathcal{H}} \in \mathbb{R}^{J_1 \times \cdots \times J_d}$ [9]. To compress the linear layer, $\boldsymbol{\mathcal{W}}$ can be saved as its low-rank factors after performing CP or TT decomposition (Table 2) [1],[2],[9].

Note that both equations (7) and (10) have the computational complexity of $\mathcal{O}(IJ)$ (Table 2) [9]. To speed up equation (10), we directly use the low-rank factors instead of reconstructing $\boldsymbol{\mathcal{W}}$ from them.

2.4.2 Tensorized Convolutional Layer

In many deep learning frameworks, to improve their computational efficiency, convolution operations are reformulated as matrix-matrix products, where their weight (kernel) tensors are converted to Toeplitz matrices. Therefore, convolutional layers can be tensorized as linear layers [1].

Format	CP	TT	Full (Recon.)
Memory Cost	$\mathcal{O}(dRI_{\max})$	$\mathcal{O}(R\ \max(I,J))$	$\mathcal{O}(IJ)$
Comp. Cost	$\mathcal{O}(dRI_{\max}J)$	$\mathcal{O}(dR^2I_{\max}\max(I,J))$	$\mathcal{O}(IJ)$

Table 2: The memory and computational cost of linear layers in the CP format (CP), the TT format (TT), and the full tensor format, i.e., the original weight is reconstructed, (Full (Recon.)), where $I_{\max} = \max_{n=1}^{d} I_n$.

A convolutional layer transforms the 3-dimensional input tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{W\times H\times C}$ into the output tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{W'\times H'\times C'}$, $H' = H - l + 1$ and $W' = W - l + 1$, by convolving $\boldsymbol{\mathcal{X}}$ with the kernel tensor $\boldsymbol{\mathcal{K}} \in \mathbb{R}^{l\times l\times C\times C'}$:

$$\boldsymbol{\mathcal{Y}}(w', h', c') = \sum_{i=1}^{l}\sum_{j=1}^{l}\sum_{c=1}^{C} \boldsymbol{\mathcal{K}}(i, j, c, c')\boldsymbol{\mathcal{X}}((w'+i-1), (h'+j-1), c). \tag{11}$$

First, we introduce a matrix $\mathbf{X} \in \mathbb{R}^{W'H'\times l^2C}$ with its row corresponding to the $l \times l \times C$ patch of $\boldsymbol{\mathcal{X}}$:

$$\mathbf{X}(w' + W'(h'-1), i + l(j-1) + l^2(c-1)) = \boldsymbol{\mathcal{X}}(w'+i-1, h'+j-1, c) \tag{12}$$

where $i, j = 1, ..., l$. Then, we reshape the kernel tensor $\boldsymbol{\mathcal{K}}$ into a matrix $\mathbf{K} \in \mathbb{R}^{l^2C\times C'}$:

$$\mathbf{K}(i + l(j-1) + l^2(c-1), s) = \boldsymbol{\mathcal{K}}(i, j, c, c'). \tag{13}$$

Finally, we compute $\mathbf{Y} = \mathbf{XK}$ and reshape $\mathbf{Y} \in \mathbb{R}^{W'H'\times C'}$ into the output tensor $\boldsymbol{\mathcal{Y}}$:

$$\boldsymbol{\mathcal{Y}}(w', h', c') = \mathbf{Y}(w' + W'(h'-1), c'). \tag{14}$$

2.4.3 CP Forward Propagation

If a layer is in CP format, its weight is decomposed into a set of factor matrices $\{\mathbf{U}^{(n)}\}_{n=1}^{2d}$. The factorized forward propagation of the layer in CP format is given as

$$\begin{aligned}\boldsymbol{\mathcal{H}} = \sum_{r=1}^{R} \Big(\boldsymbol{\mathcal{Z}} \times_1 \mathbf{U}^{(1)}(:, r) \times_2 \mathbf{U}^{(2)}(:, r) \times_3 \cdots \times_d \mathbf{U}^{(d)}(:, r)\Big) \\ \otimes \mathbf{U}^{(d+1)}(:, r) \otimes \mathbf{U}^{(d+2)}(:, r) \otimes \cdots \otimes \mathbf{U}^{(2d)}(:, r) + \boldsymbol{\mathcal{B}}\end{aligned} \tag{15}$$

and its computational complexity is $\mathcal{O}(dRI_{\max}J)$ where $I_{\max} = \max_{n=1}^{d} I_n$ (Table 2).

2.4.4 TT Forward Propagation

If a layer is in TT format, its factorized forward propagation is given as

$$\begin{aligned}\boldsymbol{\mathcal{H}}(j_1,...,j_d) = \sum_{i_1=1}^{I_1}\cdots\sum_{i_d=1}^{I_d}\boldsymbol{\mathcal{Z}}(i_1,...,i_d)\boldsymbol{\mathcal{G}}^{(1)}(:,i_1,:)\cdots\boldsymbol{\mathcal{G}}^{(d)}(:,i_d,:)\boldsymbol{\mathcal{G}}^{(d+1)}(:,j_1,:)\cdots\boldsymbol{\mathcal{G}}^{(2d)}(:,j_d,:) \\ + \boldsymbol{\mathcal{B}}(j_1,...,j_d)\end{aligned} \tag{16}$$

where $\{\boldsymbol{\mathcal{G}}^{(n)}\}_{n=1}^{2d}$ are the TT cores of $\boldsymbol{\mathcal{W}}$. The computational complexity of TT forward propagation is $\mathcal{O}(dR^2 I_{\max}\max(I,J))$ (Table 2) where R is the maximal TT rank of $\boldsymbol{\mathcal{W}}$ [11].

2.5 Tensorized LSTM

Long-short term memory (LSTM) is a recurrent neural network designed to process a sequence of data $[\mathbf{x}_1\ \mathbf{x}_2\cdots\ \mathbf{x}_T]$ [15],[17]. To have a sense of time, LSTMs have feedback connections from their memory cell states $\mathbf{c}$ and hidden cell states $\mathbf{h}$ from the previous sequence $t-1$. For each element in the input sequence, each layer computes

$$\begin{aligned}\mathbf{i}_t &= \sigma(\mathbf{W}_{\text{ii}}\mathbf{x}_t + \mathbf{b}_{\text{ii}} + \mathbf{W}_{\text{hi}}\mathbf{h}_{t-1} + \mathbf{b}_{\text{hi}}) \\ \mathbf{f}_t &= \sigma(\mathbf{W}_{\text{if}}\mathbf{x}_t + \mathbf{b}_{\text{if}} + \mathbf{W}_{\text{hf}}\mathbf{h}_{t-1} + \mathbf{b}_{\text{hf}}) \\ \mathbf{g}_t &= \tanh(\mathbf{W}_{\text{ig}}\mathbf{x}_t + \mathbf{b}_{\text{ig}} + \mathbf{W}_{\text{hg}}\mathbf{h}_{t-1} + \mathbf{b}_{\text{hg}}) \\ \mathbf{o}_t &= \sigma(\mathbf{W}_{\text{io}}\mathbf{x}_t + \mathbf{b}_{\text{io}} + \mathbf{W}_{\text{ho}}\mathbf{h}_{t-1} + \mathbf{b}_{\text{ho}}) \\ \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \\ \mathbf{o}_t &= \mathbf{o}_t \odot \tanh \mathbf{c}_t\end{aligned} \tag{17}$$

where $\mathbf{i}_t$, $\mathbf{f}_t$, $\mathbf{g}_t$, and $\mathbf{o}_t$ are input, forget, cell, and output gates, respectively. σ is the sigmoid function, and $\odot$ is the elementwise (Hadamard) product. In many deep learning frameworks, the input-to-hidden weights and biases and the hidden-to-hidden weights and biases for all gate are concatenated [10]:

$$\begin{aligned}\mathbf{W}_{\text{i}} &= [\mathbf{W}_{\text{ii}}\ \mathbf{W}_{\text{if}}\ \mathbf{W}_{\text{ig}}\ \mathbf{W}_{\text{io}}]^\top \\ \mathbf{W}_{\text{h}} &= [\mathbf{W}_{\text{hi}}\ \mathbf{W}_{\text{hf}}\ \mathbf{W}_{\text{hg}}\ \mathbf{W}_{\text{ho}}]^\top \\ \mathbf{b}_{\text{i}} &= [\mathbf{b}_{\text{ii}}\ \mathbf{b}_{\text{if}}\ \mathbf{b}_{\text{ig}}\ \mathbf{b}_{\text{io}}]^\top \\ \mathbf{b}_{\text{h}} &= [\mathbf{b}_{\text{hi}}\ \mathbf{b}_{\text{hf}}\ \mathbf{b}_{\text{hg}}\ \mathbf{b}_{\text{ho}}]^\top.\end{aligned} \tag{18}$$

Then, equation 17 can be expressed as a sequence of two linear layers:

$$\begin{pmatrix} \mathbf{i}_t \\ \mathbf{f}_t \\ \mathbf{g}_t \\ \mathbf{o}_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \tanh \\ \sigma \end{pmatrix} \mathbf{W}_{\mathrm{h}} \begin{pmatrix} \mathbf{W}_{\mathrm{i}}\mathbf{x}_t + \mathbf{b}_{\mathrm{i}} \\ \mathbf{h}_{t-1} \end{pmatrix} + \mathbf{b}_{\mathrm{h}}. \tag{19}$$

Therefore, tensorizing a single layered LSTM is equivalent to tensorizing two linear layers [10].

2.6 Bayesian Low-rank Neural Network

Bayesian low-rank neural networks are tensorized neural networks trained with a Bayesian approach to automatic rank determination from [2]. While most tensorized neural network training algorithms try to tensorize a pre-trained neural network or train at a fixed rank [1],[2],[6],[7],[9],[10], this algorithm let neural networks to learn the ranks during training [2]. First, we initialize a tensorized neural network with "high"-rank tensorized layers. Then during training, we regularize the ranks through a loss function that penalizes "high"-ranks.

In a Bayesian framework, a model learns to represent a posterior distribution over its parameters after observing some data. The posterior distribution $p(\theta|\mathcal{D})$ is given by the Bayes' rule:

$$p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta) \tag{20}$$

where $p(\mathcal{D}|\theta)$ is the likelihood of dataset $\mathcal{D}$ given by the model with parameters θ and $p(\theta)$ is a prior distribution that reflect our prior knowledge about θ.

Given that our model f is a tensorized neural network with L layers, θ is given as $\{\Phi_l, \mathbf{b}_l, \Lambda_l\}_{l=1}^{L}$ where Φ_l is the set of low-rank factors, $\mathbf{b}_l$ is the bias, and Λ_l is the set of rank controlling parameters corresponding to the layer l. Then, the prior distribution over θ is given as

$$p(\theta) = \prod_{l=1}^{L} p(\Phi_l, \Lambda_l)p(\mathbf{b}_l) \tag{21}$$

where

$$p(\mathbf{b}_l) \propto \prod_j \frac{1}{\sigma_0^2} \exp\left(\frac{\mathbf{b}_l(j)}{2\sigma_0^2}\right) \tag{22}$$

is a weak normal prior distribution that is not significant during optimization [2].

The joint factor prior distribution $p(\Phi_l, \Lambda_l)$ depends on the decomposition type of layer l [2].

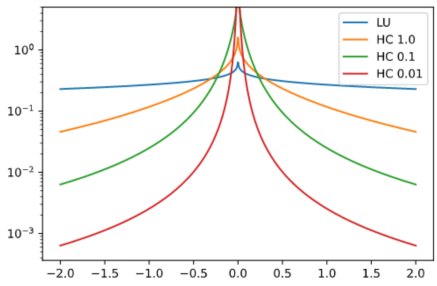
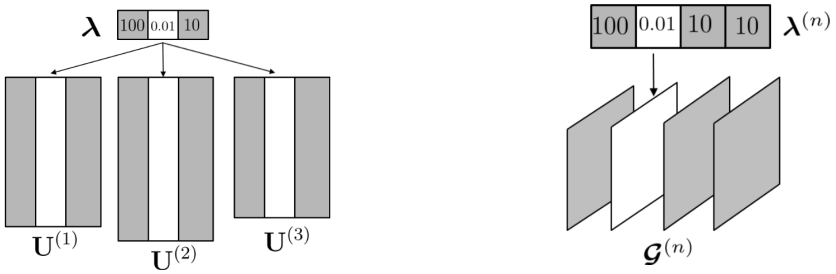


Figure 2: The distribution of $\mathbf{U}^{(n)}(i,r)$ when $p(\boldsymbol{\lambda}(r))$ is LogUniform$(\sqrt{\boldsymbol{\lambda}(r)}|0,\infty)$ (LU), HalfCauchy$(\sqrt{\boldsymbol{\lambda}(r)}|0,\eta)$ where η is 1.0 (HC 1.0), 0.1 (HC 0.1), and 0.01 (HC 0.01) (original figures from [2]).



(a) (b)

Figure 3: The rank pruning of (a) $\{\mathbf{U}^{(n)}\}_{n=1}^{N}$, if in the CP format, and (b) $\boldsymbol{\mathcal{G}}^{(n)}$, if in the TT format (original figures from [2]).

In case of CP decomposition, from equation (5), we can notice that R can be decreased by 1 by reducing the number of rank-1 tensors in the sum, which is equivalent to setting the r-th columns of $\{\mathbf{U}^{(n)}\}_{i=1}^{N}$ to zero. Therefore, we assign prior distributions that have high probability around zero to the columns to reflect our prior belief that the model can be compressed. More specifically, $\{\mathbf{U}^{(n)}\}_{n=1}^{N}$, are initialized as matrices with $R \geq R^*$ columns, i.e., $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R}$, where R^* is the optimal CP rank of the weight. Its rank controlling parameter is initialized as a vector with R positive elements: $\Lambda = \boldsymbol{\lambda} \in \mathbb{R}_{+}^{R}$. As a rank pruning prior, we place a zero-mean Gaussian prior distribution with variance $\boldsymbol{\lambda}(r)$ for $\{\mathbf{U}^{(n)}(:,r)\}_{i=1}^{N}$ where $\boldsymbol{\lambda}(r)$ has a prior distribution that prioritizes small positive values near zero (Figure 2). Then, the joint factor prior distribution is given as [2]:

$$
\begin{aligned}
p(\Phi,\Lambda) &= p(\Lambda)p(\Phi|\Lambda) \\
&= p(\boldsymbol{\lambda}) \prod_{n=1}^{N} p(\mathbf{U}^{(n)}|\boldsymbol{\lambda}) \\
&= \prod_{r=1}^{R} p(\boldsymbol{\lambda}(r)) \prod_{n=1}^{N} \prod_{i=1}^{I_n} \prod_{r=1}^{R} p(\mathbf{U}^{(n)}(i,r)|\boldsymbol{\lambda}(r)) \\
&= \prod_{r=1}^{R} p(\boldsymbol{\lambda}(r)) \prod_{n=1}^{N} \prod_{i=1}^{I_n} \prod_{r=1}^{R} \mathcal{N}(\mathbf{U}^{(n)}(i,r)|0,\boldsymbol{\lambda}(r))
\end{aligned}
\tag{23}
$$

where

$$p(\boldsymbol{\lambda}(r)) = \begin{cases} \text{HalfCauchy}(\sqrt{\boldsymbol{\lambda}(r)}|0,\eta) \text{ or} \\ \text{LogUniform}(\sqrt{\boldsymbol{\lambda}(r)}|0,\infty). \end{cases} \tag{24}$$

Note that

$$\text{HalfCauchy}(x|0,\eta) \propto \left(1+\frac{x^2}{\eta^2}\right)^{-1} \tag{25}$$

$$\text{LogUniform}(x|0,\infty) \propto x^{-1} \tag{26}$$

return high values if x is near 0. As $\boldsymbol{\lambda}(r)$ approaches zero, the entries in $\{\mathbf{U}^{(n)}(:,r)\}_{n=1}^{N}$ goes to zero leading to the decrease in CP rank (Figure 3a) [2].

If our layer in TT format, it requires mode-specific rank controlling vectors because each TT core has its own rank. The TT cores $\{\boldsymbol{\mathcal{G}}^{(n)}\}_{n=1}^{N}$ are initialized as $R_{n-1} \times I_n \times R_n$ tensors where $R_n \geq R_n^*$, and $(R_0, R_1^*, ..., R_{n-1}^*, R_N)$ is the optimal TT rank of the weight [2]. Since $R_0 = R_N = 1$ by default, we need $N-1$ positive-valued rank controlling vectors: $\Lambda = \{\boldsymbol{\lambda}_n \in \mathbb{R}_+^{R_n}\}_{n=1}^{N-1}$. Then, the TT layer's joint factor prior distribution is given as

$$\begin{aligned} & p(\Lambda)p(\Phi|\Lambda) \\ &= p(\boldsymbol{\lambda}_{N-1})p(\boldsymbol{\mathcal{G}}^{(N)}|\boldsymbol{\lambda}_{N-1}) \prod_{n=1}^{N-1} p(\boldsymbol{\lambda}_n)p(\boldsymbol{\mathcal{G}}^{(n)}|\boldsymbol{\lambda}_n) \\ &= \prod_{r=1}^{R_{N-1}} \prod_{i=1}^{I_N} \prod_{k=1}^{R_N} p(\boldsymbol{\lambda}_{N-1}(r))p(\boldsymbol{\mathcal{G}}^{(N)}(r,i,k)|\boldsymbol{\lambda}_{N-1}(r)) \\ & \quad \prod_{n=1}^{N-1} \prod_{k=1}^{R_{n-1}} \prod_{i=1}^{I_n} \prod_{r=1}^{R_n} p(\boldsymbol{\lambda}_n(r))p(\boldsymbol{\mathcal{G}}^{(n)}(k,i,r)|\boldsymbol{\lambda}_n(r)) \\ &= \prod_{r=1}^{R_{N-1}} \prod_{i=1}^{I_N} \prod_{k=1}^{R_N} p(\boldsymbol{\lambda}_{N-1}(r))\mathcal{N}(\boldsymbol{\mathcal{G}}^{(N)}(r,i,k)|0,\boldsymbol{\lambda}_{N-1}(r)) \\ & \quad \prod_{n=1}^{N-1} \prod_{k=1}^{R_{n-1}} \prod_{i=1}^{I_n} \prod_{r=1}^{R_n} p(\boldsymbol{\lambda}_n(r))\mathcal{N}(\boldsymbol{\mathcal{G}}^{(n)}(k,i,r)|0,\boldsymbol{\lambda}_n(r)) \end{aligned} \tag{27}$$

where the prior distribution for $\boldsymbol{\lambda}_n(r)$ is equation (24). As $\boldsymbol{\lambda}_n(r)$ approaches zero, the $\boldsymbol{\mathcal{G}}^{(n)}(:,:,r)$ becomes extremely sparse leading to the TT rank pruning in the n-th mode (Figure 3b) [2].

2.7 Multimodal Sentiment Analysis

Sentiment analysis is the study on the underlying disposition that one holds towards a certain entity [6],[13],[17]. It has many practical applications such as recommendation system, stock market prediction, and election outcome prediction [13]. Text-based deep neural networks have been dominating the field due to a large corpus of research available on natural language processing.

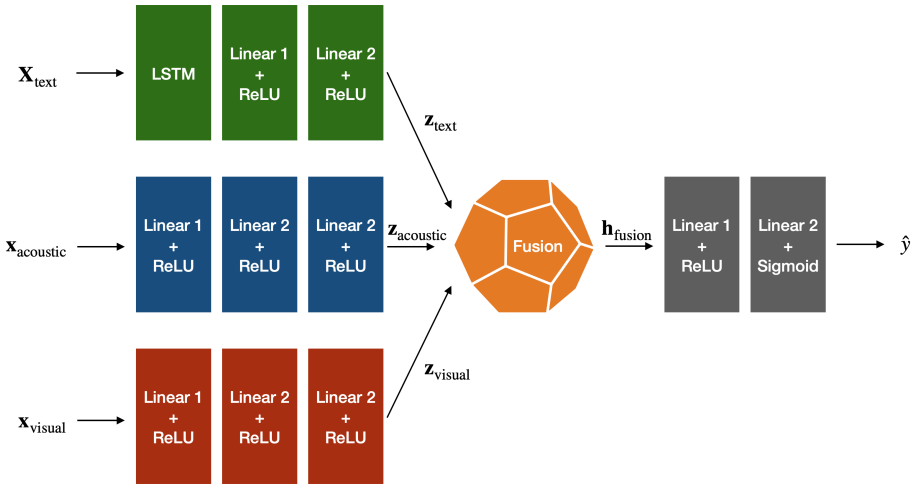


Figure 4: The network architecture of tensor fusion networks.

However, text alone has its limits as some words have different meanings depending on a person's facial expression or intonation [13].

Recently, with the rich availability of data from various sources (modalities), multimodal sentiment analysis is emerging. The motivation behind multimodal sentiment analysis is to combine information from multiple modalities to make better predictions [6],[13],[17]. Thus, the main challenge of multimodal sentiment analysis is in modeling two types of interactions: the intermodal interactions between different modalities and intramodal interactions within each modality [17].

Most works on multimodal sentiment analysis do early-fusion at the input layer or late-fusion at the decision layer [8],[16],[17]. Two approaches involve simple concatenation of vectors and fail to learn either inter or intramodal dynamics. The first approach makes it hard to learn itra-modal relationship because the training is dominated by intermodal interactions. The later limits the neural network's ability to learn intermodal dynamics. To resolve this issue, Zadeh et al. proposed tensor fusion network that can effectively learn both inter and intramodal dynamics [17].

2.8 Tensor Fusion Network

Tensor fusion networks are deep neural networks designed for end-to-end multimodal tensor fusion [17]. A tensor fusion network consists of three major building blocks: embedding subnetworks for each modality, a tensor fusion layer, and a sentiment inference subnetwork (Figure 4). The embedding subnetworks take unimodal features as inputs and each outputs a vector that contain intramodal dynamics. Given the unimodal vectors, the tensor fusion layer extracts a vector that encodes both intramodal and intermodal dynamics. Then, the extracted vector is fed to the inference subnetwork to make a prediction.

Type	TFL	LRFL	ARFL
Memory Cost	$\mathcal{O}(d_{M+1}(d_{\max})^M)$	$\mathcal{O}(d_{M+1}d_{\max}RM)$	$\mathcal{O}(d_{M+1}d_{\max}RM)$
Comp. Cost	$\mathcal{O}(d_{M+1}(d_{\max})^M)$	$\mathcal{O}(d_{M+1}d_{\max}RM)$	$\mathcal{O}(d_{M+1}d_{\max}RM)$

Table 3: The momory and computational cost of tensor fusion layers (TFL), low-rank fusion layers (LRFL), and adaptive-rank fusion layers (ARFL), where $d_{\max} = \max_{m=1}^{M} d_m$.

2.8.1 Tensor Fusion Layer

Given a set of vectors $\{\mathbf{z}_m\}_{m=1}^{M}$ from M different input modalities, a tensor fusion layer first creates an M-way fusion tensor:

$$\boldsymbol{\mathcal{Z}}_{\text{fusion}} = \bigotimes_{m=1}^{M} \mathbf{z}_m^{'} \tag{28}$$

where $\mathbf{z}_m^{'} = [\mathbf{z}_m^{\top}\ 1]^{\top} \in \mathbb{R}^{d_m}$ and $\boldsymbol{\mathcal{Z}} \in \mathbb{R}^{d_1 \times \cdots \times d_M}$ [17]. Note that by appending 1, we allow $\boldsymbol{\mathcal{Z}}$ to model every intra and intermodal interactions in $\{\mathbf{z}_m\}_{m=1}^{M}$ that are given as $\{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_M, \mathbf{z}_1 \otimes \mathbf{z}_2, \mathbf{z}_1 \otimes \mathbf{z}_3, ..., \bigotimes_{i=m}^{M} \mathbf{z}_m\}$. In other words, if each $\mathbf{z}_m$ represents a probability distribution, $\boldsymbol{\mathcal{Z}}_{\text{fusion}}$ contains every marginal and joint probability distributions we can derive out of them. Then, $\boldsymbol{\mathcal{Z}}_{\text{fusion}}$ is passed through a linear layer to produce a vector representation that summarizes the embedding vectors:

$$\mathbf{h}_{\text{fusion}} = \boldsymbol{\mathcal{Z}}_{\text{fusion}} \cdot \boldsymbol{\mathcal{W}}_{\text{fusion}} + \mathbf{b}_{\text{fusion}} \tag{29}$$

where $\mathbf{h}_{\text{fusion}} \in \mathbb{R}^{d_{M+1}}$ is a fusion output, $\boldsymbol{\mathcal{W}}_{\text{fusion}} \in \mathbb{R}^{d_1 \times \cdots \times d_M \times d_{M+1}}$ is a fusion weight, and $\mathbf{b}_{\text{fusion}} \in \mathbb{R}^{d_{M+1}}$ is a fusion bias.

Despite their powerful modeling ability, tensor fusion layers have a major drawback [6]. The computational complexity of tensor fusion layers, which combines equations (28) and (29), is given by $2\prod_{m=1}^{M+1} d_m$ and increases exponentially with the number of input modalities (Table 3). As a result, tensor fusion layers become less efficient as we try to fuse with more modalities.

2.9 Low-rank Fusion Network

Low-rank fusion networks are tensor fusion networks with their fusion layers decomposed in CP format. In addition, they perform factorized forward propagation to improve the computational efficiency of multimodal tensor fusion [6].

2.9.1 Low-rank Fusion Layer

Given $\mathcal{W}_{\text{fusion}}$ from equation (29), it is decomposed into $M+1$ low-rank factor matrices:

$$\mathcal{W}_{\text{fusion}} = \sum_{r=1}^{R_{\text{fusion}}} \bigotimes_{m=1}^{M+1} \mathbf{U}_{\text{fusion}}^{(m)}(:, r) \tag{30}$$

where $\mathbf{U}_{\text{fusion}}^{(m)} \in \mathbb{R}^{d_m \times R_{\text{fusion}}}$ [6]. Consequently, equation (29) can be rewritten into its factorized version as

$$\begin{aligned}
\mathbf{h}_{\text{fusion}} &= \sum_{r=1}^{R_{\text{fusion}}} \bigotimes_{m=1}^{M} \mathbf{z}_m^{'} \cdot \bigotimes_{m=1}^{M+1} \mathbf{U}_{\text{fusion}}^{(m)}(:, r) + \mathbf{b}_{\text{fusion}} \\
&= \sum_{r=1}^{R_{\text{fusion}}} \left(\bigotimes_{m=1}^{M} \mathbf{z}_m^{'} \cdot \mathbf{U}_{\text{fusion}}^{(m)}(:, r) \right) \otimes \mathbf{U}_{\text{fusion}}^{(M+1)}(:, r) + \mathbf{b}_{\text{fusion}} \\
&= \sum_{r=1}^{R_{\text{fusion}}} \left(\bigodot_{m=1}^{M} \mathbf{z}_m^{'} \cdot \mathbf{U}_{\text{fusion}}^{(m)}(:, r) \right) \otimes \mathbf{U}_{\text{fusion}}^{(M+1)}(:, r) + \mathbf{b}_{\text{fusion}} \\
&= \left(\bigodot_{m=1}^{M} \mathbf{z}_m^{'} \cdot \mathbf{U}_{\text{fusion}}^{(m)} \right) \otimes \mathbf{U}_{\text{fusion}}^{(M+1)} + \mathbf{b}_{\text{fusion}}
\end{aligned} \tag{31}$$

where $\bigodot$ denotes elementwise product. It has the computational and the memory cost of $R_{\text{fusion}} \sum_{m=1}^{M+1} d_m$ (Table 3) [6].

3 Adaptive-rank Fusion Network

Low-rank fusion layers can reduce the computational complexity and memory cost of tensor fusion but have a major drawback that R_{fusion} from equation (31) needs to be pre-determined. As a solution to the problem, we propose adaptive-rank fusion networks that have adaptive-rank fusion layers. Compared to low-rank fusion layers, adaptive-rank fusion layers can learn the optimal rank of their weights.

3.1 Adaptive-rank Fusion Layer

An adaptive-rank fusion layer can be seen as a low-rank fusion layer that can learn its rank with the algorithm from [2]. Therefore, it has a rank controlling parameter vector $\boldsymbol{\lambda}_{\text{fusion}} \in \mathbb{R}^{M+1}$ and its forward propagation is identical to that of a low-rank fusion layer in equation (31) (Table 3). Since adaptive rank fusion layers are essentially linear layers in CP format, their joint factor prior distribution is given by equation (23).

3.2 TT Text Embedding Subnetwork

TT text embedding subnetworks are tensorized text embedding subnetworks with their LSTMs and linear layers tensorized in TT format.

3.3 Prior Distribution

Adaptive-rank fusion networks are not fully tensorized neural networks in a sense that they have some layers that are not tensorized. Let θ be the parameters of an adaptive-rank fusion network f with L non-tensorized layers. Then, its prior distribution is given as

$$p(\theta) = p(\theta_{\text{fusion}})p(\theta_{\text{TT}}) \prod_{l=1}^{L} p(\theta_l) \tag{32}$$

where θ_{fusion} is the adaptive-rank fusion layer parameters, θ_{TT} is the TT text embedding subnetwork parameters, if applicable, and θ_l is the parameters of non-tensorized layer l. For layer l that is not tensorized, its parameters are given as $\theta_l = \{\mathbf{W}_l, \mathbf{b}_l\}$, and their joint prior distribution is defined as

$$p(\mathbf{W}_l)p(\mathbf{b}_l) = \prod_{i,j} p(\mathbf{W}_l(i,j)) \prod_{j} p(\mathbf{b}_l(j)) \tag{33}$$

where $p(\mathbf{W}_l(i,j)) \propto \frac{1}{\sigma_0^2} \exp\left(\frac{\mathbf{W}_l(i,j)}{2\sigma_0^2}\right)$ and $p(\mathbf{b}_l(j)) \propto \frac{1}{\sigma_0^2} \exp\left(\frac{\mathbf{b}_l(j)}{2\sigma_0^2}\right)$ are the weak normal prior distributions that are not significant during optimization.

3.4 Maximum a Posteriori

The maximum a posteriori (MAP) optimization finds the set of model parameters that maximize a point estimate of the posterior distribution from equation (20). The MAP optimization for the adaptive-rank fusion network f is given as

$$\begin{aligned}
\theta_{MAP} &= \arg\max_{\theta} \; P(\theta|\mathcal{D}) \\
&= \arg\min_{\theta} \; -\log P(\theta|\mathcal{D}) \\
&\propto \arg\min_{\theta} \; -\log P(\mathcal{D}|\theta)P(\theta) \\
&\propto \arg\min_{\theta} \; -\log P(\mathcal{D}|\theta) - \log P(\theta) \\
&\propto \arg\min_{\theta} \; -\log P(\mathcal{D}|\theta) - \log P(\theta_{\text{fusion}}) - \log P(\theta_{\text{TT}}) - \sum_{l=1}^{L} \log P(\theta_l) \\
&\propto \arg\min_{\theta} \; -\log P(\mathcal{D}|\theta) - \log P(\theta_{\text{fusion}}) - \log P(\theta_{\text{TT}})
\end{aligned} \tag{34}$$

where $-\log P(\mathcal{D}|\theta)$ is also known as the negative log likelihood. Given that f is designed for binary classification with logistic regression, its prediction is given as $\hat{y} = f((x, y)|\theta) = p(y|x, \theta)$ where $(x, y) \in \mathcal{D}$ is an input-label pair. Then, the negative log likelihood is computed as

$$\begin{aligned}
-\log P(\mathcal{D}|\theta) &= -\log \prod_{(x,y)\in\mathcal{D}} \hat{y}^y (1-\hat{y})^{1-y} \\
&= -\sum_{(x,y)\in\mathcal{D}} y \log \hat{y} + (1-y)\log(1-\hat{y}),
\end{aligned} \tag{35}$$

which is also known as the binary cross entropy (BCE) loss.

3.5 Training Tips

There are factors that may cause the training to fail. Since the domain of the priors for the rank controlling vectors in equation (24) is positive, the rank controlling vectors should be clamped to a very small number such as 1×10^{-10} if their elements become negative during training. The initialization of low-rank factors is also very important. If a layer is in CP format, we calculate σ_{f} the target standard deviation for its factor matrices

$$\sigma_{\text{f}} = \sqrt[4N]{\frac{\sigma_{\text{w}}^2}{R}} \tag{36}$$

where σ_{w}^2 is the target variance of the original weight matrix. Otherwise, if a layer is in TT format,

the target standard deviation is given as

$$\sigma_{\mathrm{f}} = \sqrt[2N]{\frac{\sigma_{\mathrm{w}}^2}{R_{\mathrm{prod}}}} \tag{37}$$

where R_{prod} is the product of its TT rank. Then, we initialize all the factors $\sim \mathcal{N}(0, \sigma_{\mathrm{f}}^2)$ [15].

As mentioned before, it is important to set the rank of a layer high enough so that the rank is not lower than the true rank of its weight. A safe bet is to initialize the rank with a value that preserves the number of parameters after decomposition.

The negative log prior terms, $-\log p(\theta_{\mathrm{fusion}})$ and $-\log p(\theta_{\mathrm{TT}})$, in equation (34) can be overwhelming and put too much emphasis on sparsity during training. Therefore, we multiply a hyperparameter β to them to control the accuracy-sparsity trade-off.

4 Experiments

4.1 Dataset

Multimodal Opinion Sentiment Intensity (CMU- MOSI) dataset is an annotated dataset of 93 opinion video opinions from YouTube movie reviews [6],[17]. Each video consists of multiple opinion segments, and each segment is annotated with the sentiment in the range $[-3, 3]$. The train dataset contains 1284 samples, the validation dataset contains 229 samples, and the test dataset contains 686 samples. Acoustic and visual features of CMU-MOSI are extracted at respective sampling rates, while text data are segmented per word. The text features are given by $\mathbf{X}_{\text{text}} \in \mathbb{R}^{300\times T}$ where T is the number of words in an utterance and $\mathbf{x}_{\text{text},t} \in \mathbb{R}^{300}$ is a GloVe word vector embedding. For each opinion utterance audio, a set of acoustic features are extracted using COVAREP acoustic analysis framework. These extracted features capture different characteristics of human voice and have been shown to be related to emotions. For each opinion segment with multiple audio frames, we perform mean pooling over frames to obtain the expected acoustic features $\mathbf{x}_{\text{acoustic}} \in \mathbb{R}^5$. The speaker's facial emotion indicators for each frame of video are extracted using FACET facial expression analysis framework. A set of 20 Facial Action Units indicating detailed muscle movements on the face, are also extracted using FACET. We perform mean pooling over the frames to obtain the expected visual features $\mathbf{x}_{\text{visual}} \in \mathbb{R}^{20}$ [6],[17].

4.2 Network Architecture

A general tensor fusion network for multimodal sentiment analysis with CMU-MOSI dataset consists of a text embedding subnetwork, an acoustic embedding subnetwork, a visual embedding subnetwork, a fusion layer of your choice (tensor fusion, low-rank fusion, or adaptive-rank fusion), and an inference subnetwork (Figure 4). It takes inputs from three modalities: text, acoustic, and visual, which are processed through the corresponding subnetworks. The outputs from the subnetworks are merged together by the fusion layer into a single output that contains multimodal information. Then, the fusion layer output is used by the inference subnetwork to make a prediction (Figure 4).

The text embedding subnetwork is composed of a long short-term memory (LSTM) followed by 2 linear layers with rectified linear unit (ReLU) activation functions [17]. The input to the LSTM is $\mathbf{X}_{\text{text}}$. The LSTM is single layered and has the input size of 300, the sequence length of 20, and the hidden size of 128. Input to the first linear layer is the hidden state output of the LSTM. The first linear layer has the size of (128×128). The second linear layer has the size of (128×128) as well [17]. If it is tensorized in to a TT text embedding subnetwork, the two linear layers in the LSTM are tensorized into TT layers with the sizes of (4×5×15×4×8×16) and (4×5×8×4×8×16), and the

other two linear layers into TT linear layers with the size of (4×4×8×4×4×8).

The acoustic embedding subnetwork is a 3-layer fully-connected neural network with ReLU activation functions after each layer [17]. Input to the network is $\mathbf{x}_{\text{acoustic}}$. Its input layer has the size of (5×32). The following layers have the size of (32×32). The visual embedding subnetwork is identical to the acoustic embedding subnetwork except for the first layer due to the difference in input feature sizes. Input for the visual embedding subnetwork is $\mathbf{x}_{\text{visual}}$, and its input layer has the size of (20×32). Since they are so small in size, it is hard to find the benefit of tensorizing the acousting and visual embedding subnetworks [17].

Once, $\mathbf{z}_{\text{text}}$, $\mathbf{z}_{\text{acoustic}}$, and $\mathbf{z}_{\text{visual}}$, the outputs of the embedding subnetworks, are gathered, they are concatenated with 1 into $\mathbf{z}'_{\text{text}} = [\mathbf{z}^{\top}_{\text{text}}\ 1]^{\top} \in \mathbb{R}^{129}$, $\mathbf{z}'_{\text{acoustic}} = [\mathbf{z}^{\top}_{\text{acoustic}}\ 1]^{\top} \in \mathbb{R}^{33}$, and $\mathbf{z}'_{\text{visual}} = [\mathbf{z}^{\top}_{\text{visual}}\ 1]^{\top} \in \mathbb{R}^{33}$. Then, they are fed forward through the fusion layer with the size of $(129 \times 33 \times 33 \times 128)$ to output $\mathbf{h}_{\text{fusion}} \in \mathbb{R}^{128}$ that encodes intra and intermodal interactions in $\mathbf{z}_{\text{text}}$, $\mathbf{z}_{\text{acoustic}}$, and $\mathbf{z}_{\text{visual}}$. Finally, the inference subnetwork has an input layer of size (128×128) with the ReLU and an output layer of size (128×1) with the sigmoid activation [17]. It takes $\mathbf{h}_{\text{fusion}}$ as an input to make a prediction $\hat{y} \in \mathbb{R}$ for binary classification task.

4.3 Training Adaptive Rank Fusion Networks

Let us look into the rank-adaptive training of an adaptive-rank fusion network with the initial fusion rank of 100 (ARFN 100) to gain better insight to the rank reduction mechanism and training tips.

Figure 5a is the validation loss and Figure 5b is the estimated rank during its training. We can notice that the training does converge to a model with the lowest validation loss but to a model with a higher validation loss but less number of parameters (lower rank). This is because equation (34) involves minimizing the loss and the rank at the same time.

In Figure 5d, the average column-wise $l2$-norm (Avg. Col. L2-norm) $\bar{\mathbf{v}} = \sum_{n=1}^{N} \mathbf{v}^{(n)}$ is computed, where $\mathbf{v}^{(n)} \in \mathbb{R}^{R_{\text{fusion}}}$ is the column-wise $l2$-norm of each factor matrix $\mathbf{U}^{(n)}$. The histogram of $\bar{\mathbf{v}}$ shows the distribution of the elements of $\bar{\mathbf{v}}$ at epochs 1, 5, and 10. From Figure 5b, we can guess that the number of columns with $\bar{\mathbf{v}} \approx 0$ should increase as the training proceeds. Indeed, the histogram gets more concentrated around 0 as we progress from epoch 1 to epoch 10.

Similarly, $\bar{\mathbf{g}} \in \mathbb{R}^{R_{\text{fusion}}}$, the average column-wise $l2$-norm of their gradient (Avg. Col. Grad. L2-norm) is computed. Figure 5c plots the number of columns with high gradient norm, or the number of elements with high value in $\bar{\mathbf{g}}$, and the amount of change in R_{fusion} ($|\Delta R_{\text{fusion}}|$) per epoch. We can notice that the two proxies are highly correlated indicating that our rank controlling framework

	Acc-2 (%)	F1 (%)	# Params	R_{fusion}	Tra. Time (s)	Inf. Time (s)
TFN	74.74	55.79	1.83×10^7	-	5.31	.0292
LRFN 100	74.64	64.49	3.07×10^5	100	1.31	.0050
LRFN 79	73.47	65.92	3.01×10^5	79	1.31	.0044
ARFN 100	75.22	64.73	3.01×10^5	79	1.46	.0045
ARFN-T 100	72.74	65.43	7.24×10^4	78	17.04	.0474

Table 4: The binary accuracy (Acc-2), the f1 score (F1), the number of model parameters (# Params), the fusion rank (R_{fusion}), the training time per epoch (Tra. Time), and the inference time per sample (Inf. Time) are measured for TFN, LRFN 100, LRFN 79, ARFN 100, and ARFN-T 100.

	TT	Full (Recon.)
Fwd. Time (s)	2.06×10^{-3}	7.01×10^{-4}

Table 5: The forward propagation time per sample for the linear layers (128×128)in TT format and in original format (Full (Recon.)).

is working as expected.

Figure 5e is the bar graph of the average of $\bar{\mathbf{g}}$ (Avg. Avg. Col. Grad. L2-norm) over the training epochs. Therefore, it shows the average amount of gradient that columns experience over the training. Columns 0 and 35 experienced the most gradient during the training, so we can guess that those columns are likely to be subject to the rank reduction. In Figure 5f, we compare their average column-wise $l2$-norms to those of columns 2 and 40, which experience small amount of gradient during the training according to Figure 5e. By epochs 9, the average column-wise $l2$-norms of columns 0 and 35 go to 0 while those of column 2 and 40 converge to approximately 0.75 and 0.2, respectively, by the end of the training. Indeed, columns 2 and 40 were not pruned out in the final compressed model. Additionally, the evolution of column 2 shows that some columns may come back to life, and so, it is wise to wait till the convergence to prune the model (Figure 5e).

After the training of adaptive-rank fusion networks, we prune the sparsified factors for insignificant columns or slices and discard the rank controlling vectors to get compressed models for inference.

4.4 Comparison with other Models

In this experiment, a tensor fusion network (TFN), a low-rank fusion network with fixed fusion rank 100 (LRFN 100), a low-rank fusion network with fixed fusion rank 79 (LRFN 79), and an adaptive-rank fusion network with the initial fusion rank of 100 (ARFN 100), and an ARFN 100 with TT text embedding subnetwork (ARFN-T 100). They are compared in terms of binary accuracy, f1 score, the number of model parameters, fusion rank, train time per epoch, and inference time per

sample, which are measured on a CPU (Table 4). The TFN is trained at the learning rate of 5×10^{-4}. The low-rank fusion networks are trained at the learning rate of 1×10^{-3}. The ARFN 100 are trained at the learning rate of 1×10^{-2} with the negative log prior coefficient β of 1×10^{-1}. The ARFN-T 100 is trained with the learning rate of 5×10^{-4} for the TT text embedding subnetwork and 5×10^{-3} for the fusion layer and 1×10^{-3} for the rest with β of 1×10^{-3}. All networks are trained using the Adam optimizer with the batch size of 32 and the learning rate scheduler that reduce learning rates by $\times 10$ if a model's validation loss does not decrease for two training epochs in a row.

The TFN contains 1.83 $\times 10^7$ parameters, where 1.80 $\times 10^7$ (98%) belongs to its tensor fusion layer. Since tensor fusion layers are so big, adaptive-rank fusion networks with $R_{\text{fusion}} \approx 60000$ have the same number of parameters as the tensor fusion network (Table 4). Therefore, the ARFN 100 with $R_{\text{fusion}} = 100$ is already a very compressed model compared to the TFN. However, we were able to confidently set $R_{\text{fusion}} = 100$ since the prior work reported that low-rank fusion networks perform well at $R_{\text{fusion}} \leq 100$ [6]. The ARFN 100 is compressed to a model with $\times 60$ less parameters at better binary accuracy and f1 score than the TFN . Additionally, its training time per epoch is reduced by more than $\times 2$ compared to that of the TFN. Compared to LRFN 100, the ARFN 100, despite having the same number of parameters, trains slightly slower due to the time spent on computing the negative log prior term in equation 34 (Table 4).

The ARFN-T 100 was trained to test the impact of having more tensorized layers compared to the ARFN 100. We were able to train a model with $\times 4$ less parameters than the ARFN 100 due to the compression from the TT linear layers of the TT text embedding subnetwork (Table 4). However, a downside of adding TT linear layers is that it takes considerable time to compute the negative log prior over their parameters because their factors are order-3 tensors instead of order-2 tensors in CP linear layer cases. Therefore, the training time per epoch of the ARFN-T 100 turns out to be slower than that of the TFN (Table 4). Its slower inference time can be explained by Table 5 where we measure the forward propagation time of linear layers in TT format (TT) and in original format (Full (Recon.)) for a single data sample, i.e., the batch size is 1. The original linear layer has the shape of (128×128), and the TT linear has the shape of $(4 \times 4 \times 8 \times 4 \times 4 \times 8)$ with the TT rank of $(1, 21, 32, 32, 21, 32, 1)$, so that it has the same complexity (number of elements) as the original linear layer. In Table 5, we can witness that the linear layers TT formats are significantly slower than the ones in the original format due to many implementation tricks behind the modern deep learning frameworks that are evolved to speed-up large-sized matrix-matrix multiplication (especially on GPUs). In order to reflect the gain in computational complexity of the TT forward propagation, we need to develop deep learning frameworks that are specialized in accelerating the computations in the units of low-rank factors.

4.5 The Generalization of Adaptive-rank Fusion Networks

In this experiment, we train all the models from the previous experiment (TFN, LRFN 100, ARFN 100, and ARFN-T 100), a low-rank fusion network with fixed rank 1000 (LRFN 1000), and an adaptive-rank fusion network with initial fusion rank 1000 (ARFN 1000). The LRFN 1000 and the ARFN 1000 were tested as models with higher capacities than their lower rank versions, LRFN 100 and ARFN 100. All models are trained under the same optimization settings from the previous experiment. To test their generalization performance, we measure their accuracy when they are trained with 100%, 90%, 80%, and 70% of the training dataset.

Table 6 summarizes 5 different training results and their average for each case. In Figure 6a, we plot the average binary accuracy of the models versus the proportion of the original training dataset used to train the model. Our obvious expectation is that all models experience drop in their test accuracy as we provide them with less data. However, when we have 70% of training dataset, the ARFN-T 100 actually converged to a better model (Figure 6a). This observation was unexpected, and so, we repeated the experiment and still got the same result. Nevertheless, the accuracy decays with 90% and 80% of training dataset are much less compare to other models (Figure 6b), which shows that the ARFN-T 100 has an excellent generalization ability. Generally, the models with adaptive-rank layers (ARFN 1000, ARFN 100, and ARFN-T 100) seem to be more robust to being trained with less data than the one without adaptive-rank layers (Figure 6b).

We also wanted to see if the compression ratio of the models are affected by the decrease in the number of training samples. The compression ratio is measured by dividing the final number of model parameters by the initial number of model parameters. In Figure 6c, we plot the compression ratio of ARFN 1000, ARFN 100, and ARFN-T 100 at different training dataset settings. The models tend to compress less when trained with less training samples (Figure 6c).

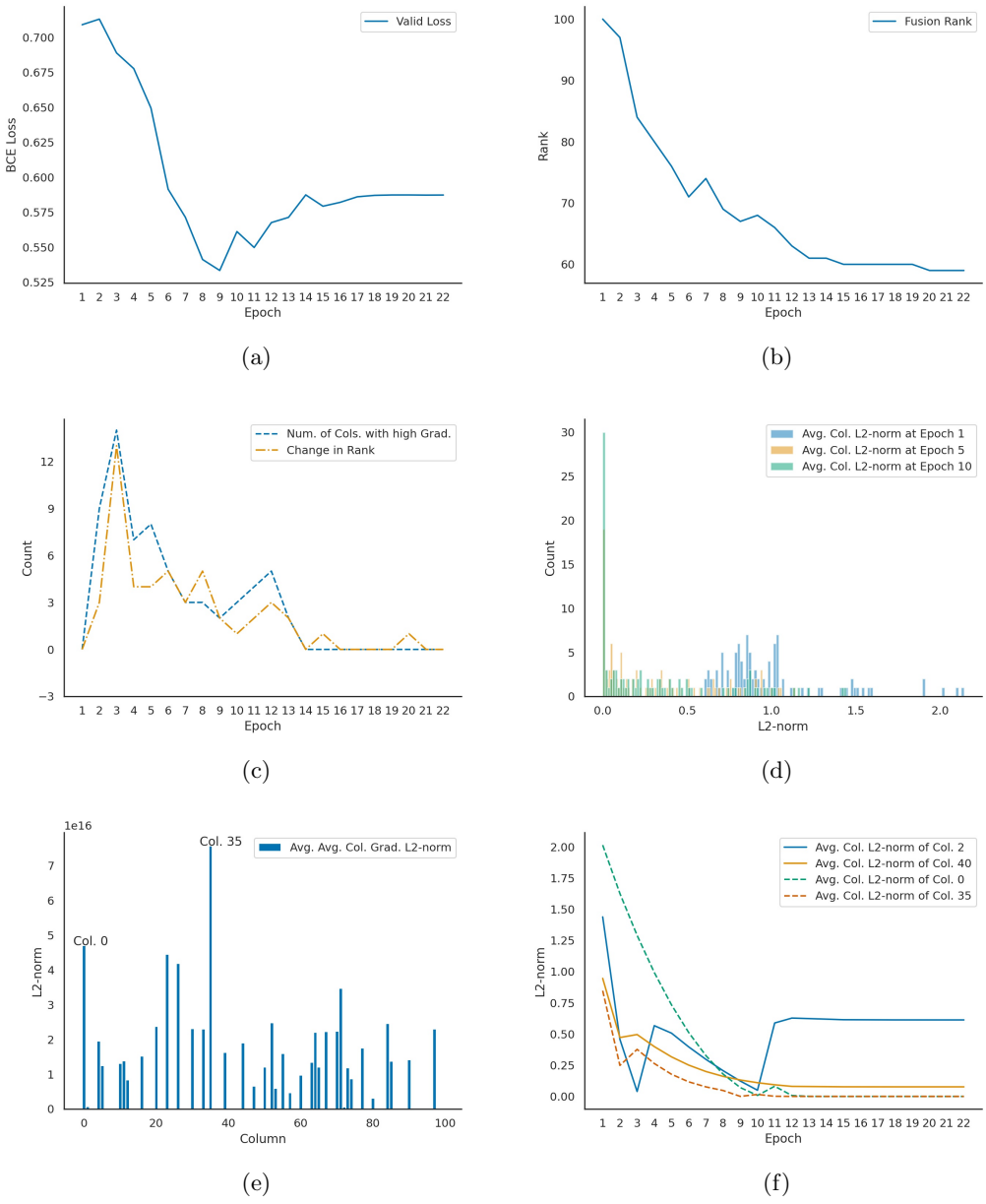
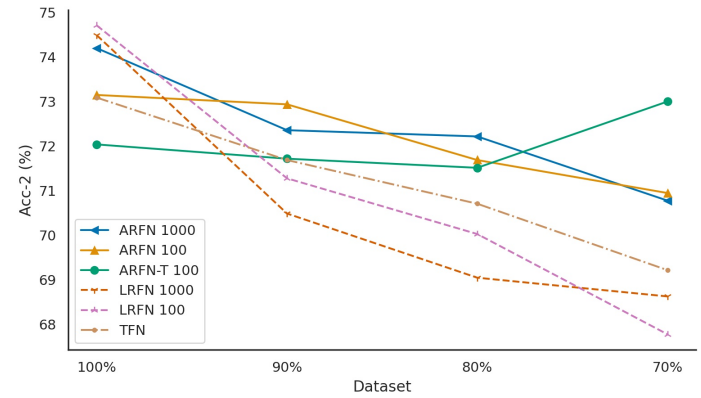
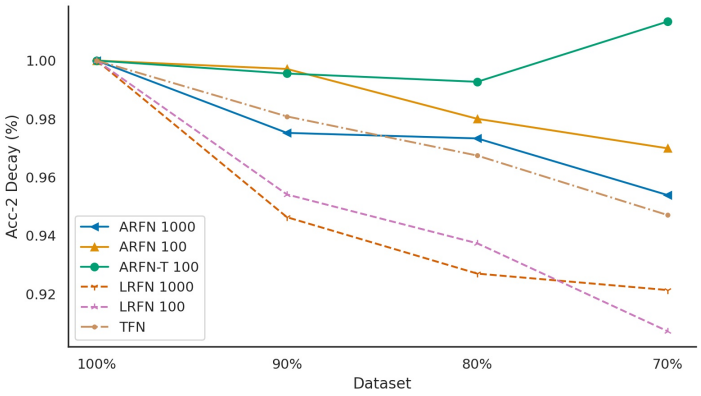


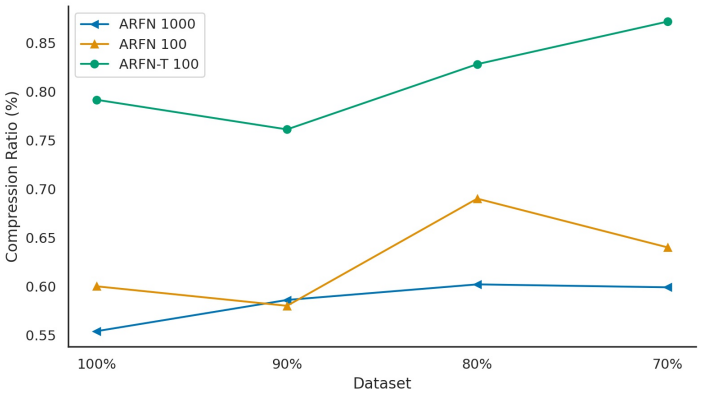
Figure 5: The training of ARFN 100. (a) The validation loss (BCE loss) of ARFN at different training epochs. (b) The fusion rank of ARFN 100 at different training epochs. (c) The number of columns (Num. of Cols.) with high gradient (Grad.) and $|\Delta R_{\text{fusion}}|$ at different training epochs. (d) The histogram of average column-wise $l2$-norm (Avg. Col. L2-norm) of different columns at epochs 1, 5, 10. (e) The bar graph of the average average column-wise $l2$-norm of gradient (Avg. Avg. Col. L2-norm) over training epochs. (f) The column-wise $l2$-norm of gradient of columns (Col.) 2, 40, 0, and 35 at each epoch is plotted.



(a)



(b)



(c)

Figure 6: The binary accuracy (Acc-2) (a) and the decay in binary accuracy (b) of TFN, LRFN 1000, LRFN 100, ARFN 1000, ARFN 100, and ARFN-T 100 trained with 100%, 90%, 80%, and 70% of the training dataset. The compression ratio (c) of ARFN 1000, ARFN 100, and ARFN-T 100 trained with 100%, 90%, 80%, and 70% of the training dataset.

	TFN	LRFN 1000	LRFN 100	ARFN 1000	ARFN 100	ARFN-T 100
100%	73.04	75.51	73.75	74.49(.53)	75.35(.54)	73.90(.81)
	73.76	75.07	76.09	75.07(.60)	74.78(.53)	70.85(.77)
	73.03	74.05	75.21	72.59(.54)	71.87(.57)	72.45(.86)
	74.05	73.62	74.34	74.34(.55)	71.87(.66)	69.97(.74)
	71.59	74.20	74.20	74.93(.55)	72.01(.68)	73.03(.78)
	73.09	**74.49**	**74.71**	**74.20(.54)**	**73.15(.60)**	**72.04(.79)**
90%	73.47	72.01	71.57	71.57(.59)	72.01(.69)	73.33(.71)
	72.59	72.01	70.99	75.07(.59)	72.16(.65)	74.34(.88)
	71.72	69.38	70.55	72.01(.65)	73.47(.50)	70.26(.88)
	69.97	68.80	70.84	72.74(.59)	73.32(.50)	70.84(.59)
	70.70	70.26	72.44	70.41(.52)	73.76(.57)	69.83(.74)
	71.69	**70.49**	**71.28**	**72.36(.59)**	**72.94(.58)**	**71.72(.76)**
80%	71.43	67.78	70.70	70.99(.58)	71.57(.72)	70.70(.72)
	68.95	72.44	69.97	73.91(.62)	71.57(.65)	69.97(.77)
	71.28	70.12	71.43	72.30(.59)	71.28(.71)	73.03(.78)
	72.89	69.96	69.68	71.14(.60)	73.03(.71)	75.35(.95)
	68.99	67.20	68.37	72.74(.62)	70.99(.69)	73.18(.92)
	70.71	**69.05**	**70.03**	**72.22(.60)**	**71.69(.70)**	**71.52(.83)**
70%	68.99	69.24	69.83	70.12(.60)	70.55(.63)	73.03(.90)
	71.57	70.41	67.06	71.14(.61)	69.68(.68)	73.90(.88)
	68.22	68.95	66.18	71.82(.62)	72.24(.60)	75.32(.94)
	68.66	66.90	67.49	70.26(.60)	70.99(.61)	71.72(.77)
	68.68	67.64	68.08	70.55(.61)	71.28(.69)	73.03(.91)
	69.22	**68.63**	**67.78**	**70.78(.60)**	**70.95(.64)**	**73.03(.82)**

Table 6: The binary accuracy of TFN, LRFN, LRFN 100, ARFN 1000, ARFN 100, and ARFN-T 100 are measured when they are trained with 100%, 90%, 80%, and 70% of the training dataset. The bold numbers are the average accuracy over 5 different runs, and the compression ratio is in the parenthesis, if applicable.

5 Discussion

Surprisingly, the ARFN 100, which is $\times 60$ compressed compared to the TFN, scored higher testing accuracy than the TFN (Table 4). We also found out that the models trained with adaptive-rank training, including the ARFN 100, are less dependent on the size of the dataset (Figure 6b). Therefore, we conjecture that the CMU-MOSI being a small dataset with 1284 training samples could have played a role in the increased testing accuracy of the compressed model.

Most intelligent people think abstractly and simply. Therefore, the hidden motivation for compression is in finding models that are smaller in size but generalize better. Previous works found that training a smaller network from scratch has poorer generalization performance than training a larger network and pruning it to the same size [5],[4],[18]. Intuitively, larger networks have larger capacity, i.e., more degrees of freemdom, and offer larger sets of possible solutions. Moreover, more parameters means smoother loss function leading to better generalization performance [4]. To support these observations theoretically, the information theory is used to show that compression is required for good generalization if a model is trained with a finite set of training samples [14]. Therefore, we should try to optimize for the generalization gap and the complexity gap during the training of deep neural networks [14]. While other compression/pruning algorithms reduce pre-trained networks, during the training of our adaptive-rank fusion networks, we optimize jointly for the generalization gap, represented by the BCE loss, and the complexity gap, represented by the log prior over tensorized layer parameters, in equation (34). Therefore, we think that the models with adaptive-rank layers have better generalization performance than the ones without because they are optimized for both generalization and complexity gaps at the same time.

6 Conclusion

This work proposes adaptive-rank fusion networks that are trained with the MAP version of the training algorithm proposed in [2]. We show that adaptive-rank fusion networks can perform better than tensor fusion networks and low-rank fusion networks. Additionally, our work profiles the training and inference time of different adaptive-rank networks and report that the gains in FLOPs of the factorized forward propagation are not reflected in the training and the inference time of the adaptive-rank networks due to the way that the deep learning libraries are implemented. Finally, we observe that the generalization power of the models trained with the rank-adaptive training algorithm is better than the models that did not. In the future, we would like to test if our observation extends to other dataset and other architectures. Also, we want to compare the gain in generalization performance of general pruning and of the rank pruning in order to see the role of tensor decomposition in the gain.

References

[1] Timur Garipov et al. "Ultimate tensorization: compressing convolutional and fc layers alike". In: *arXiv preprint arXiv:1611.03214* (2016).

[2] Cole Hawkins, Xing Liu, and Zheng Zhang. "Towards compact neural networks via end-to-end training: A bayesian tensor approach with automatic rank determination". In: *SIAM Journal on Mathematics of Data Science* 4.1 (2022), pp. 46–71.

[3] Tamara G Kolda and Brett W Bader. "Tensor decompositions and applications". In: *SIAM review* 51.3 (2009), pp. 455–500.

[4] Hao Li et al. "Visualizing the loss landscape of neural nets". In: *Advances in neural information processing systems* 31 (2018).

[5] Ji Lin et al. "Runtime neural pruning". In: *Advances in neural information processing systems* 30 (2017).

[6] Zhun Liu et al. "Efficient low-rank multimodal fusion with modality-specific factors". In: *arXiv preprint arXiv:1806.00064* (2018).

[7] Rahul Mishra, Hari Prabhat Gupta, and Tanima Dutta. "A survey on deep neural network compression: Challenges, overview, and solutions". In: *arXiv preprint arXiv:2010.03954* (2020).

[8] Behnaz Nojavanasghari et al. "Deep multimodal fusion for persuasiveness prediction". In: (2016), pp. 284–288.

[9] Alexander Novikov et al. "Tensorizing neural networks". In: *Advances in neural information processing systems* 28 (2015).

[10] Charles C Onu, Jacob E Miller, and Doina Precup. "A fully tensorized recurrent neural network". In: *arXiv preprint arXiv:2010.04196* (2020).

[11] Ivan V Oseledets. "Tensor-train decomposition". In: *SIAM Journal on Scientific Computing* 33.5 (2011), pp. 2295–2317.

[12] Stephan Rabanser, Oleksandr Shchur, and Stephan Günnemann. "Introduction to tensor decompositions and their applications in machine learning". In: *arXiv preprint arXiv:1711.10781* (2017).

[13] Mohammad Soleymani et al. "A survey of multimodal sentiment analysis". In: *Image and Vision Computing* 65 (2017), pp. 3–14.

[14] Naftali Tishby and Noga Zaslavsky. "Deep learning and the information bottleneck principle". In: *2015 ieee information theory workshop (itw)*. IEEE. 2015, pp. 1–5.

[15] Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. "Tensor decomposition for compressing recurrent neural network". In: *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2018, pp. 1–8.

[16] Jennifer Williams et al. "Dnn multimodal fusion techniques for predicting video sentiment". In: (2018), pp. 64–72.

[17] Amir Zadeh et al. "Tensor fusion network for multimodal sentiment analysis". In: *arXiv preprint arXiv:1707.07250* (2017).

[18] Michael Zhu and Suyog Gupta. "To prune, or not to prune: exploring the efficacy of pruning for model compression". In: *arXiv preprint arXiv:1710.01878* (2017).