Tensor Regression Networks

Jean Kossaifi
Amazon AI
Imperial College London
jean.kossaifi@imperial.ac.uk

Zachary Lipton
Amazon AI
University of California, San Diego
zlipton@cs.ucsd.edu

Aran Khanna
Amazon AI
arankhan@amazon.com

Tommaso Furlanello
Amazon AI
University of Southern California
furlanel@usc.edu

Anima Anandkumar
Amazon AI
California Institute of Technology
anima@amazon.com

Abstract

To date, most convolutional neural network architectures output predictions by flattening 3rd-order activation tensors, and applying fully-connected output layers. This approach has two drawbacks: (i) we lose rich, multi-modal structure during the flattening process and (ii) fully-connected layers require many parameters. We present the first attempt to circumvent these issues by expressing the output of a neural network directly as the the result of a multi-linear mapping from an activation tensor to the output. By imposing low-rank constraints on the regression tensor, we can efficiently solve problems for which existing solutions are badly parametrized. Our proposed tensor regression layer replaces flattening operations and fully-connected layers by leveraging multi-modal structure in the data and expressing the regression weights via a low rank tensor decomposition. Additionally, we combine tensor regression with tensor contraction to further increase efficiency. Augmenting the VGG and ResNet architectures, we demonstrate large reductions in the number of parameters with negligible impact on performance on the ImageNet dataset.

1 Introduction

Many natural datasets exhibit rich multi-modal structure. We can express audio spectrograms as 2nd-order tensors (matrices) in which the modes correspond to frequency and time. Images are third-order tensors with modes corresponding to height, width and the filter channel. Videos could be expressed as 4th-order tensors, and the signal processed by an array of video sensors could be described as a 5th-order tensor. Owing to the abundance of multi-modal data, tensor methods, which extend linear algebra to multilinear structures, are promising tools for manipulating and analyzing this data.

Although the mathematical properties of tensors have long been the subject of theoretical study, until recently most machine learning techniques have regarded points data as vectors and datasets as matrices. However, over the past decade, tensor methods have come to prominence in the machine learning community. One class of broadly useful techniques within tensor methods are tensor decompositions, which extend the familiar linear algebraic matrix decompositions to to higher-order

tensors. Tensor methods have recently been developed for learning latent variable models [1], and developing recommender systems [2].

In the past several years, several papers have investigated the intersection of tensor methods and deep learning in a variety of ways. Some papers use tensor as tools of analysis in order to gain insight into the representational expressivity of CNNs. Deep neural networks have demonstrated remarkable breakthroughs for a variety of tasks in the domains of natural language processing, computer vision and speech recognition. Yet, despite this success, there remain many open questions as to why they work so well and whether they really require so many parameters in order to achieve state-of-the-art performance, Tensor methods have emerged as promising tools of analysis to address these questions and to better understand the success of deep neural networks [3, 4]. In another line of research, tensor methods are investigated as tools for devising neural network learning algorithms with theoretical guarantees of convergence [5, 6, 7].

Generally, two families of tensor methods have been studied in connection with deep learning: tensor decomposition and tensor regression. While tensor decomposition has been applied in deep learning for several applications including multi-task learning [8], sharing residual units [9] and speeding up convolutional neural networks [10], most of these approaches are simply used for initialization [8], require fine tuning [11] or use the Tensor-Train format [12] which presents several issues due to the arbitrary reshaping of matrices into tensors. One promising direction is to incorporate tensor regressions into deep neural networks. Tensor regressions leverage the natural multi-modal structure of its inputs to learn compact predictive models [13, 14, 15, 16]. To our knowledge, no previous attempt has been made to incorporate either tensor regressions or tensor contractions as end-to-end trainable components of neural networks themselves.

In this paper, we pursue this new research direction and investigate tensor operations, as pluggable components of neural networks. Our aim is to use tensor methods to exploit the rich multilinear structure in our data without giving up the the power and flexibility conferred by modern deep learning methods. Consider that modern deep neural networks make frequent use of high-order tensors for representing spatial and sequential data. In a standard deep Convolutional Neural Network (CNN), the inputs and the activations of hidden convolutional layers are all 3^{rd}-order tensors. And yet, to wit, most architectures output predictions by first flattening the activations tensors, before connecting to the output neurons via a fully-connected layer. This approach presents several issues: we lose rich, multi-modal information during the flattening process and the fully-connected layers require many parameters (typically the product of the input's length by the number of hidden layers/outputs).

Specifically, we propose incorporating tensor regression into the neural network architecture by leveraging a low-rank representation of the regression weights expressed as the factors of a tensor decomposition. Our proposed Tensor Regression Layer (TRL) replaces flattening operations and fully-connected layers by leveraging multi-dimensional structure in the data and expressing the regression weights via a low rank tensor decomposition. Additionally, we combine tensor regression with tensor contraction to further increase efficiency. Augmenting the VGG and ResNet architectures, we demonstrate parameter reduction and improved performance on the ImageNet dataset.

2 Mathematical background

Notation: Throughout the paper, we define tensors as multidimensional arrays, with indexing starting at 0. First order tensors are vectors, denoted $\mathbf{v}$. Second order tensors are matrices, denoted $\mathbf{M}$ and Id is the identity matrix. We denote $\tilde{\mathcal{X}}$ tensors of order 3 or greater. The transpose of $\mathbf{M}$ is denoted $\mathbf{M}^\top$ and its pseudo-inverse $\mathbf{M}^\dagger$. Finally, for any $i, j \in \mathbb{N}$, $[i \,..\, j]$ denotes the set of integers $\{i, i+1, \cdots, j-1, j\}$.

Tensor unfolding: Given a tensor, $\tilde{\mathcal{X}} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_N}$, its mode-$n$ unfolding is a matrix $\mathbf{X}_{[n]} \in \mathbb{R}^{I_n, I_M}$, with $M = \prod_{\substack{k=0,\\ k\neq n}}^{N} I_k$ and is defined by the mapping from element $(i_0, i_1, \cdots, i_N)$ to (i_n, j), with $j = \sum_{\substack{k=0,\\ k\neq n}}^{N} i_k \times \prod_{m=k+1}^{N} I_m$.

Tensor vectorization: Given a tensor, $\tilde{\mathcal{X}} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_N}$, it can be flattened into a vector $\text{vec}(\tilde{\mathcal{X}})$ of size $(I_0 \times \cdots \times I_N)$ defined by the mapping from element $(i_0, i_1, \cdots, i_N)$ of $\tilde{\mathcal{X}}$ to element j of $\text{vec}(\tilde{\mathcal{X}})$, with $j = \sum_{k=0}^{N} i_k \times \prod_{m=k+1}^{N} I_m$.

n-mode product: For a tensor $\tilde{\mathcal{X}} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_N}$ and a matrix $\mathbf{M} \in \mathbb{R}^{R \times I_n}$, the n-mode product of a tensor is a tensor of size $(I_0 \times \cdots \times I_{n-1} \times R \times I_{n+1} \times \cdot \times I_N)$ and can be expressed using unfolding of $\tilde{\mathcal{X}}$ and the classical dot product as:

$$\tilde{\mathcal{X}} \times_n \mathbf{M} = \mathbf{M}\tilde{\mathcal{X}}_{[n]} \in \mathbb{R}^{I_0 \times \cdots \times I_{n-1} \times R \times I_{n+1} \times \cdot \times I_N} \tag{1}$$

Tucker decomposition: Given a tensor $\tilde{\mathcal{X}} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_N}$, it can be decomposed into a low rank core $\tilde{\mathcal{G}} \in \mathbb{R}^{R_0 \times R_1 \times \cdots \times R_N}$ through projection along each of its modes by projection factors $\left(\mathbf{U}^{(0)}, \cdots, \mathbf{U}^{(N)}\right)$, with $\mathbf{U}^{(k)} \in \mathbb{R}^{R_k, I_k}, k \in (0, \cdots, N)$.

In other words, we can write:

$$\tilde{\mathcal{X}} = \tilde{\mathcal{G}} \times_0 \mathbf{U}^{(0)} \times_1 \mathbf{U}^{(2)} \times \cdots \times_N \mathbf{U}^{(N)} = [\![\tilde{\mathcal{G}}; \mathbf{U}^{(0)}, \cdots, \mathbf{U}^{(N)}]\!] \tag{2}$$

Typically, the factors and core of the decomposition are obtained by solving a least squares problem. In particular, closed form solutions can be obtained for the factor by considering the $n-$mode unfolding of $\tilde{\mathcal{X}}$ that can be expressed as:

$$\mathbf{X}_{[n]} = \mathbf{U}^{(n)}\mathbf{G}_{[n]} \left(\mathbf{U}^{(0)} \otimes \cdots \mathbf{U}^{(n-1)} \otimes \mathbf{U}^{(n+1)} \otimes \cdots \otimes \mathbf{U}^{(N)}\right)^T \tag{3}$$

Similarly, we can optimize the core in a straightforward manner by isolating it using the equivalent rewriting of the above equality:

$$vec(\mathbf{X}) = \left(\mathbf{U}^{(0)} \otimes \cdots \otimes \mathbf{U}^{(N)}\right) vec(\mathbf{G}) \tag{4}$$

The interested reader is referred to the thorough review of the literature on tensor methods by Kolda and Bader [17].

3 Tensor Contraction and Tensor Regression

In this section, we explain how to incorporate tensor contractions and tensor regressions into neural networks as differentiable layers.

3.1 Tensor Contraction

The most obvious way to incorporate tensor operations into a neural network is to apply tensor contraction to an activation tensor in order to obtain a low-dimensional representation [18]. We call this technique the Tensor Contraction layer (TCL). Compared to performing a similar rank reduction with a fully-connected layer, TCLs require fewer parameters and less computation.

Tensor contraction layers Given an activation tensor $\tilde{\mathcal{X}}$ of size $(D_0, \cdots, D_N)$, the TCL will produce a compact core tensor $\tilde{\mathcal{G}}$ of smaller size $(D_0, R_1, \cdots, R_N)$ defined as:

$$\tilde{\mathcal{G}} = \tilde{\mathcal{X}} \times_1 \mathbf{V}^{(1)} \times_2 \mathbf{V}^{(2)} \times \cdots \times_N \mathbf{V}^{(N)} \tag{5}$$

with $\mathbf{V}^{(k)} \in \mathbb{R}^{R_k, I_k}, k \in (1, \cdots, N)$. Note that the projections start at the second mode since the first mode corresponds to the batch-size.

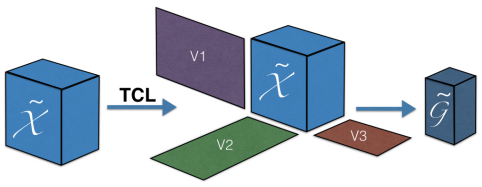


Figure 1: A representation of the Tensor Contraction Layer (TCL) on a tensor of order 3. The input tensor $\tilde{\mathcal{X}}$ is contracted into a low rank core $\tilde{\mathcal{G}}$.

The projection factors $\left(\mathbf{V}^{(k)}\right)_{k \in [1, \cdots N]}$ are learned end-to-end with the rest of the network by gradient backpropagation. In the rest of this paper, we denote *size–*$(R_1, \cdots, R_N)$ *TCL*, or *TCL–*$(R_1, \cdots, R_N)$ a TCL that produces a compact core of dimension $(R_1, \cdots, R_N)$.

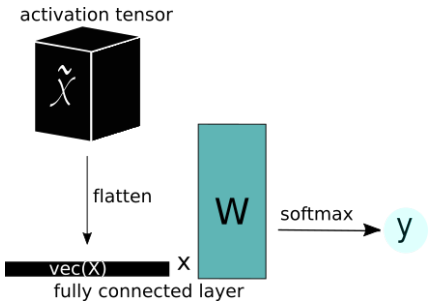


Figure 2: In a traditional deep neural network architecture, the input tensor $\tilde{\mathcal{X}}$ is flattened and passed on to a fully connected layer, where it is multiplied by a weight matrix $\mathbf{W}$.

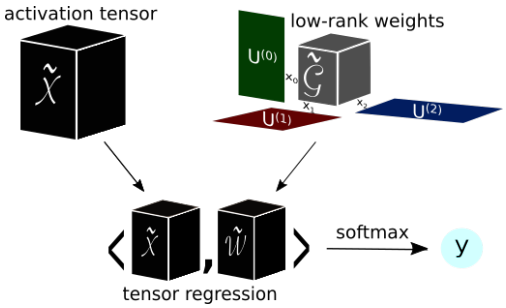


Figure 3: We propose to replace the flattening and fully connected layers by a tensor regression layer. In other words, we express directly the output as the result of the product between the activation tensor and low-rank regression weights $\tilde{\mathcal{W}}$.

Gradient back-propagation In the case of the TCL, we simply need to take the gradients with respect to the factors $\mathbf{V}^{(k)}$ for each $k \in 0, \cdots, N$ of the tensor contraction. Specifically, we compute

$$\frac{\partial \tilde{\mathcal{G}}}{\partial \mathbf{V}^{(k)}} = \frac{\partial \tilde{\mathcal{X}} \times_1 \mathbf{V}^{(1)} \times_2 \mathbf{V}^{(2)} \times \cdots \times_N \mathbf{V}^{(N)}}{\partial \mathbf{V}^{(k)}} \tag{6}$$

By rewriting the previous equality in terms of unfolded tensors, we get an equivalent rewriting where we have isolated the considered factor:

$$\frac{\partial \tilde{\mathcal{G}}_{[k]}}{\partial \mathbf{V}^{(k)}} = \frac{\partial \mathbf{V}^{(k)} \mathbf{X}_{[k]} \left(Id \otimes \mathbf{V}^{(1)} \otimes \cdots \mathbf{V}^{(k-1)} \otimes \mathbf{V}^{(k+1)} \otimes \cdots \otimes \mathbf{V}^{(N)} \right)^T}{\partial \mathbf{V}^{(k)}} \tag{7}$$

Model complexity Considering an activation tensor $\tilde{\mathcal{X}}$ of size $(D_0, D_1, \cdots, D_N)$, a size–$(R_1, \cdots, R_N)$ Tensor Contraction Layer will have a total of $\sum_{k=1}^{N} D_k \times R_k$ parameters.

3.2 Low-Rank Tensor Regression

Typically, the activation tensor resulting from the core of a deep convolutional network (e.g. a series of convolution, ReLu and Pooling) is passed through an average pooling layer that averages out the spatial components, indicating a hight degree of redundancy. The result is then flattened before being passed to a fully connected layer that produces the desired number of outputs. We propose to leverage the structure in the activation tensor and formulate the output as lying in a low rank subspace that models jointly the input and the output as well as their multi-modal structure. We do this by means of a low rank tensor regression, where we enforce a low multilinear rank of the regression weight tensor.

Tensor regression as a layer Specifically, let us note $\tilde{\mathcal{X}} \in \mathbb{R}^{S, I_0 \times I_1 \times \cdots \times I_N}$ the input activation tensor corresponding to S samples $\left(\tilde{\mathcal{X}}_1, \cdots, \tilde{\mathcal{X}}_S\right)$ and $\mathbf{Y} \in \mathbb{R}^{S,O}$ the O corresponding labels for each sample. The problem is that of estimating the regression weight tensor $\tilde{\mathcal{W}} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_N \times O}$ under some fixed low rank $(R_0, \cdots, R_N, R_{N+1})$, such that, $\mathbf{Y}_{ij} = \langle \tilde{\mathcal{X}}_i, \tilde{\mathcal{W}}_j \rangle + b$, i.e.

$$\begin{aligned} Y &= \tilde{\mathcal{X}}_{[0]} \times \tilde{\mathcal{W}}_{[-1]} + \mathbf{b} \\ \text{subject to } \tilde{\mathcal{W}} &= [\![\tilde{\mathcal{G}}; \mathbf{U}^{(0)}, \cdots, \mathbf{U}^{(N)}, \mathbf{U}^{(N+1)}]\!] \end{aligned} \tag{8}$$

with $\tilde{\mathcal{G}} \in \mathbb{R}^{R_0 \times \cdots \times R_N \times R_{N+1}}$, $\mathbf{U}^{(k)} \in \mathbb{R}^{I_k \times R_k}$ for each k in $[0 \,..\, N]$ and $\mathbf{U}^{(N+1)} \in \mathbb{R}^{O \times R_{N+1}}$.

This problem has previously been studied as a standalone problem where the input is directly mapped to the output. This approach has several drawbacks: it implies pre-processing the data to extract

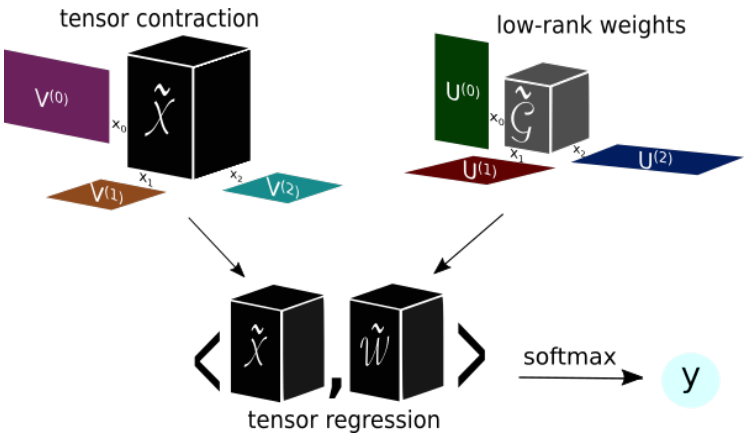


Figure 4: Alternatively, we propose to first reduce the dimensionality of the activation tensor by applying tensor contraction before performing tensor regression.

(hand-crafted) features. The problem is typically solved analytically by formulating the problem as a least squares one. This involves heavy computation and memory usage which renders it prohibitive in practice for large dataset sizes.

In this work, we depart from the standard approach and decide to leverage the multi-modal information in the activation tensor as well as the structure in of the problem. We do so by introducing a neural network layer that replaces the traditional flattening + fully-connected layers and applies tensor regression directly to its input while enforcing low rank constraints on the weights of the regression. We coin our layer *Tensor Regression Layer* (*TRL*). Intuitively, the advantage in doing so is many fold: we are able to leverage the multi-modal structure in the data and express the solution as lying on a low rank manifold encompassing both the data and the associated outputs.

Gradient backpropagation The gradients of the regression weights and the core with respect to each factor can be obtained by writing:

$$\frac{\partial \tilde{\mathcal{W}}}{\partial \mathbf{U}^{(k)}} = \frac{\partial \tilde{\mathcal{G}} \times_0 \mathbf{U}^{(0)} \times_1 \mathbf{U}^{(1)} \times \cdots \times_{N+1} \mathbf{U}^{(N+1)}}{\partial \mathbf{U}^{(k)}} \tag{9}$$

Using the unfolded expression of the regression weights, we obtain the equivalent formulation:

$$\frac{\partial \tilde{\mathcal{W}}_{[k]}}{\partial \mathbf{U}^{(k)}} = \frac{\partial \mathbf{U}^{(k)} \mathbf{G}_{[k]} \left(\mathbf{U}^{(0)} \otimes \cdots \mathbf{U}^{(k-1)} \otimes \mathbf{U}^{(k+1)} \otimes \cdots \otimes \mathbf{U}^{(N+1)}\right)^T}{\partial \mathbf{U}^{(k)}} \tag{10}$$

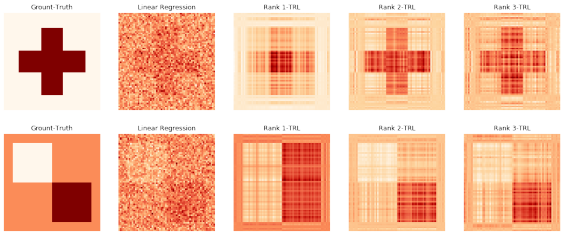
Similarly, we can obtain the gradient with respect to the core by considering the vectorized expressions:

$$\frac{\partial \text{vec}(\tilde{\mathcal{W}})}{\partial \text{vec}(\tilde{\mathcal{G}})} = \frac{\partial \left(\mathbf{U}^{(0)} \otimes \cdots \otimes \mathbf{U}^{(N+1)}\right) vec(\mathbf{G})}{\partial \text{vec}(\tilde{\mathcal{G}})} \tag{11}$$

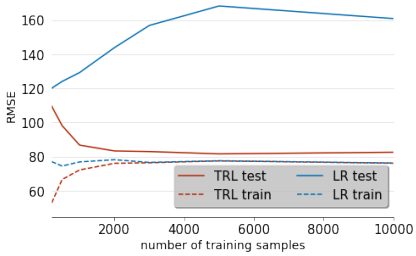
Model analysis We consider as input an activation tensor $\tilde{\mathcal{X}} \in \mathbb{R}^{S, I_0 \times I_1 \times \cdots \times I_N}$, and a rank-$(R_0, R_1, \cdots, R_N, R_{N+1})$ tensor regression layer, where, typically, $R_k \leq I_k$. Let's assume the output is n-dimensional. A fully connected layer taking $\tilde{\mathcal{X}}$ as input will have $n_{\text{FC}} = n \times \prod_{k=0}^{N} I_k$ parameters.

By comparison, the tensor regression layer will have a number of parameters n_{TRL}, with:

$$n_{\text{TRL}} = \prod_{k=0}^{N+1} R_k + \sum_{k=0}^{N} R_k \times I_k + R_{N+1} \times n \tag{12}$$



(a) Empirical comparison of the TRL with a Fully Connected layer regression. We plot the weight matrix of both the TRL and a Fully Connected layer. Thanks to its low-rank weights, the TRL is able to better capture the structure in the weights and is more robust to noise.



(b) Evolution of the RMSE as a function of the training set size for both the TRL and Fully connected based linear regression (*LR*)

Table 1: Results obtained on ImageNet by adding a TCL to a VGG-19 architecture. We reduce the number of hidden units proportionally to the reduction in size of the activation tensor following the tensor contraction. Doing so allows more than 65% space savings over all three fully-connected layers (i.e. 99.8% space saving over the fully-connected layer replaced by the TCL) with no corresponding decrease in performance (comparing to the standard VGG network as a baseline).

Method		Accuracy		Space Savings
TCL–size	Hidden Units	Top-1 (%)	Top-5 (%)	(%)
baseline	4096	68.7	88	0
(512, 7, 7)	4096	**69.4**	**88.3**	-0.21
(384, 5, 5)	3072	68.3	87.8	**65.87**

4 Experiments

We empirically demonstrate the effectiveness of preserving the tensor structure through tensor contraction and tensor regression by integrating it into a state-of-the-art architectures and demonstrating similar performance on the popular ImageNet dataset. In particular, we empirically verify the effectiveness of the TCL on VGG-19 [19] and conduct thorough experiment with the tensor regression on ResNet-50 and ResNet-101 [20].

4.1 Experimental setting

Synthetic data To illustrate the effectiveness of the low-rank tensor regression, we apply it to synthetic data $y = \text{vec}(\tilde{\mathcal{X}}) \times \mathbf{W}$ where each sample $\tilde{\mathcal{X}} \in \mathbb{R}^{(64)}$ follows a Gaussian distribution $\mathcal{N}(0, 3)$. $\mathbf{W}$ is a fixed structure matrix and the labels are generated as $y = \text{vec}(\tilde{\mathcal{X}}) \times \mathbf{W}$. We then train the data on $\tilde{\mathcal{X}} + \tilde{\mathcal{E}}$ where $\tilde{\mathcal{E}}$ is added Gaussian noise sampled from $\mathcal{N}(0, 3)$. We compare i) a TCL chained with squared loss and ii) a Fully Connected Layer with a squared loss. In Figure 5a, we show the trained weight of both a linear regression based on a fully-connected layer and a TRL with various ranks, both obtained in the same setting. As can be observed in Figure **??**, the TRL is easier to train on small datasets and less prone to over-fitting, due to the low rank structure of its regression weights, as opposed to typical Fully Connected based Linear Regression.

ImageNet Dataset We ran our experiments on the widely used ImageNet-1K dataset, using state-of-the-art network architectures. The ILSVRC dataset (ImageNet), is composed of $1,2$ million images for training and $50,000$ for validation, all labeled for 1,000 classes. We evaluate the classification error on the validation set as is the common practice, on single 224×224 single center crop from the raw input images. For training, we adopt the same data augmentation procedure as in the original Residual Networks (ResNets) paper [20].

Training the TRL When experimenting with the tensor regression layer, we did not retrain the whole network each time but started from a pre-trained ResNet. We experimented with two settings: i) First, we replaced the last average pooling, flattening and fully connected layer by either a TRL or a combination of TCL + TRL and trained these from scratch while keeping the rest of the network fixed. ii) Second, we investigate replacing the pooling and fully connected layers with a TRL that jointly learns the spatial pooling as part of the tensor regression. In that setting, further explore initializing the TRL by performing a Tucker decomposition on the weights of the fully connected layer.

Implementation details We implemented all models using the MXNet library [21] and ran all experiments training with data parallelism across multiple GPUs on Amazon Web Services, with 4 NVIDIA k80 GPUs. We report results on the validation set in term of Top-1 accuracy and Top-5 accuracy across all 1000 classes.

When training the layers from scratch, we found it useful to add a batch normalization layer [22] before and after the TCL/TRL to avoid vanishing or exploding gradients, and to make the layers more robust to changes in the initialization of the factors. In addition we constrain the weights of the tensor regression by applying ℓ_2 normalization [23] to the factors of the Tucker decomposition.

4.2 Results

Table 2: Results obtained with ResNet-50 on ImageNet. First row corresponds to the standard ResNet. Rows 2 and 3 presents the results obtained by replacing the last average pooling, flattening and fully connected layers with a Tensor Regression Layer. In the last row, we have also added a Tensor Contraction Layer.

Method			Accuracy	
Architecture	TCL–size	TRL rank	Top-1 (%)	Top-5 (%)
Resnet-50	baseline with spatial pooling		74.58	92.06
	-	(1000, 2048, 7, 7)	73.6	91.3
	-	(500, 1024, 3, 3)	72.16	90.44
	(1024, 3, 3)	(1000, 1024, 3, 3)	73.43	91.3
Resnet-101	baseline with spatial pooling		77.1	93.4
	-	(1000, 2048, 7, 7)	76.45	92.9
	-	(500, 1024, 3, 3)	76.7	92.9
	(1024, 3, 3)	(1000, 1024, 3, 3)	76.56	93

Impact of the tensor contraction layer We first investigate the effectiveness of the TCL using a VGG-19 network architecture [19]. It is particularly adapted for this test since of its $138,357,544$ parameters, $119,545,856$, more than 80%, are contained in the fully-connected layers. Therefore, by adding a TCL to contract the activation tensor prior to the fully connected layers we can achieve large space saving. We define the space saving of a model M with n_M total parameters in its fully connected layers with respect to a reference model R with n_R total parameters its fully connected layers is defined as $1 - \frac{n_M}{n_R}$ (bias excluded).

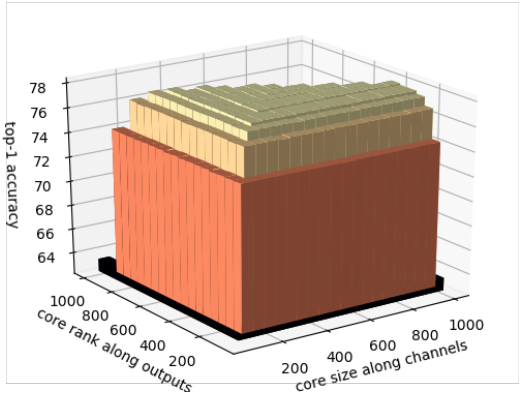
Table 1 presents the accuracy obtained by the different combinations of TCL in term of top-1 and top-5 accuracy as well as space saving. By adding a TCL that preserves the size of its input we are able to obtain slightly higher performance with little impact on the space saving (0.21% of space loss) while by decreasing the size of the TCL we got more than 65% space saving with almost no performance deterioration.

Overcomplete TRL We first tested the tensor regression layer with a ResNet-50 and a ResNet-101 architectures on ImageNet. In particular we removed the average pooling layer to preserve the spatial dimension of the tensor.

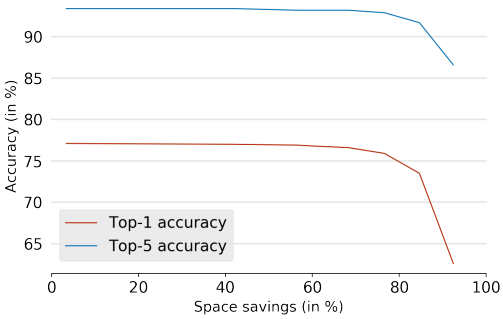
The full activation tensor is directly passed on to a tensor regression layer which produces the outputs on which we apply softmax to get the final predictions. This results in more parameters as the spatial dimensions are preserved. To reduce the computational burden but preserve the multi-dimensional information, we alternatively insert a Tensor Contraction Layer before the TRL. In Table 2, we present

Table 3: Results obtained with ResNet-101 on ImageNet by learning the spatial pooling as part of the TRL.

	Performance (%)		
TRL rank	Top-1	Top-5	Space savings
baseline	**77.1**	**93.4**	0
(200, 1, 1, 200)	**77.1**	**93.2**	68.2
(150, 1, 1, 150)	76	92.9	76.6
(100, 1, 1, 100)	74.6	91.7	84.6
(50 , 1, 1, 50)	73.6	91	**92.4**



(a) Accuracy as a function of the core size



(b) Accuracy as a function of space savings

Figure 6: 6a shows the evolution of the Top-1 accuracy (in %) as we vary the size of the core along the number of outputs and number of channels (the TRL does spatial pooling along the spatial dimensions, i.e. the core has rank 1 along these dimensions). In 3, we present the space savings associated with some of these configurations.

results obtained in this setting on ImageNet for various configurations of the network architecture. In each case, we report the size of the TCL (i.e. the dimension of the contracted tensor) and the rank of the TRL (i.e. the dimension of the core of the regression weights).

Joint spatial pooling and low-rank regression Alternatively, we can learn the spatial pooling as part of the tensor regression. In this case, we remove the average pooling layer and feed the tensor of size (batch size, number of channels, height, width) to the TRL, while imposing a rank of 1 on the spatial dimensions of the core tensor of the regression. Effectively, this setting simultaneously learns weights for the multi-linear spatial pooling as well as the regression.

In practice, to initialize the weights of the TRL in this setting, we consider the weight of fully connected layer from a pre-trained model as a tensor of size (batch size, number of channels, 1, 1, number of classes) and apply a partial tucker decomposition to it by keeping the first dimension (batch-size) untouched. The core and factors of the decomposition then give us the initialization of the tensor regression layer. The projection vectors over the spatial dimension are the initialize to $\frac{1}{\text{height}}$ and $\frac{1}{\text{width}}$, respectively. The Tucker decomposition was performed using TensorLy [24]. In this setting, we show that we can decrease drastically the number of parameters with little impact on performance. In particular, Figure **??** shows the evolution of the Top-1 and Top-5 accuracy as we decrease the size of the core tensor of the TRL as well as the space savings.

5 Conclusions

We introduced a tensor regression layer that can replace fully-connected layers in neural networks. Unlike fully-connected layers, tensor regression layers do not require flattening the input tensor, which

looses information, and can leverage the multi-dimensional dependencies in the data. Additionally, demonstrated that by imposing a low-rank constraint on the weights of the regression, we can effectively learn a low-rank manifold on which both the data and the labels lie. The result is a compact network, capable of learning over-complete representations without loss in accuracy. Going forward, we plan to apply the tensor regression layer to more network architectures. We also plan to leverage recent work [25] on extending BLAS primitives to avoid transpositions needed to compute the tensor contractions, which will further speed up the computations.

References

[1] A. Anandkumar, R. Ge, D. J. Hsu, S. M. Kakade, and M. Telgarsky, "Tensor decompositions for learning latent variable models.," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 2773–2832, 2014.

[2] A. Karatzoglou, X. Amatriain, L. Baltrunas, and N. Oliver, "Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering," in *Proceedings of the fourth ACM conference on Recommender systems*, pp. 79–86, ACM, 2010.

[3] N. Cohen, O. Sharir, and A. Shashua, "On the expressive power of deep learning: A tensor analysis," *CoRR*, vol. abs/1509.05009, 2015.

[4] B. D. Haeffele and R. Vidal, "Global optimality in tensor factorization, deep learning, and beyond," *CoRR*, vol. abs/1506.07540, 2015.

[5] H. Sedghi and A. Anandkumar, "Training input-output recurrent neural networks through spectral methods," *CoRR*, vol. abs/1603.00954, 2016.

[6] M. Janzamin, H. Sedghi, and A. Anandkumar, "Generalization bounds for neural networks through tensor factorization," *CoRR*, vol. abs/1506.08473, 2015.

[7] M. Janzamin, H. Sedghi, and A. Anandkumar, "Beating the perils of non-convexity: Guaranteed training of neural networks using tensor methods," *CoRR*, 2015.

[8] Y. Yang and T. M. Hospedales, "Deep multi-task representation learning: A tensor factorisation approach," *CoRR*, vol. abs/1605.06391, 2016.

[9] Y. Chen, X. Jin, B. Kang, J. Feng, and S. Yan, "Sharing residual units through collective tensor factorization in deep neural networks," 2017.

[10] V. Lebedev, Y. Ganin, M. Rakhuba, I. V. Oseledets, and V. S. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *CoRR*, vol. abs/1412.6553, 2014.

[11] Y. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *CoRR*, vol. abs/1511.06530, 2015.

[12] A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov, "Tensorizing neural networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems*, NIPS'15, pp. 442–450, 2015.

[13] W. Guo, I. Kotsia, and I. Patras, "Tensor learning for regression," *IEEE Transactions on Image Processing*, vol. 21, pp. 816–827, Feb 2012.

[14] G. Rabusseau and H. Kadri, "Low-rank regression with tensor responses," in *NIPS* (D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, eds.), pp. 1867–1875, 2016.

[15] H. Zhou, L. Li, and H. Zhu, "Tensor regression with applications in neuroimaging data analysis," *Journal of the American Statistical Association*, vol. 108, no. 502, pp. 540–552, 2013.

[16] Q. R. Yu and Y. Liu, "Learning from multiway data: Simple and efficient tensor regression," *CoRR*, vol. abs/1607.02535, 2016.

[17] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM REVIEW*, vol. 51, no. 3, pp. 455–500, 2009.

[18] J. Kossaifi, A. Khanna, Z. Lipton, T. Furlanello, and A. Anandkumar, "Tensor contraction layers for parsimonious deep nets," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, 2015.

[21] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *CoRR*, vol. abs/1512.01274, 2015.

[22] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[23] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *NIPS* (D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, eds.), pp. 901–909, 2016.

[24] J. Kossaifi, Y. Panagakis, and M. Pantic, "Tensorly: Tensor learning in python," *ArXiv e-print*, 2016.

[25] Y. Shi, U. N. Niranjan, A. Anandkumar, and C. Cecka, "Tensor contractions with extended blas kernels on cpu and gpu," in *2016 IEEE 23rd International Conference on High Performance Computing (HiPC)*, pp. 193–202, Dec 2016.