Matrix Decomposition and Approximation in Neural Network: Theory and Applications

Shuchang Zhou [1]

July 30, 2015

[1]State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, 100190.

Contents

List of Figures

List of Tables

List of Algorithms

Preface

In this paper we study theory and applications of matrix decomposition and approximation techniques to machine learning tasks based on Artificial Neural Network, an important tool in the branch of machine learning for modeling non-linear functions. There are two scenarios in which matrix techniques are relevant: first, the data being processed by neural network may be organized as matrices to reveal the underlying structure; second, the coefficients in each layer of neural network can be organized into matrices themselves. In particular, we show that matrix techniques may be used to perform following operations of neural networks: first, a neural network may be approximated with another neural network, possibly with some pre-specified properties; second, the input data may be augmented with their normalized forms before being passed to a neural network to improve the accuracy of prediction; third, we may enforce the output of neural network to be on some low-dimensional manifold by imposing structural constraint. The aim of this paper is to advance the state-of-art in model approximation, data normalization and prediction regularization, in tasks related to neural networks.

The main contributions of this paper are: first, a new type of layer that can be used as replacement for fully-connected layer based on Kronecker Tensor Product that reduces amount of computation and number of parameters; second, a new type of output layer that may be used to impose structural constraints on the predictions of neural networks which reduces the number of parameters and improves prediction accuracy; third, a new interpretation to unify several matrix norms and matrix decompositions in terms of optimization over group orbits, leading to a unified framework for deriving relationship between some well-known matrix norms and matrix decompositions, and investigating new kinds of data normalization. The new normalization may be applied to augmenting input to neural networks for improving prediction accuracy. We apply aforementioned techniques to several benchmarks, including MNIST, SVHN and a Chinese character recognition data set. Empirical results show that these techniques can help build neural networks that require less amount of computation, have less number of parameters and are more robust against noise.

Chapter 1

Introduction

The increasing speed of generation of sensor data has called for automatic processing and analyzing. Machine learning can help the processing of data with the supervised learning scenario: given some training data labeled with the ground truth, a non-linear function can be constructed to approximate the relationship between the input data and the output labels. In the task of Object Recognition in image for example, the input data are the images, possibly taken by a camera on a hand-held mobile phone, and the desired outputs are the locations and category of objects visible in these images. The difficulty of this task lies in several aspects: first the object being recognized can be of any pose and may not be rigid, have complex texture, and may be partially occluded; second, the background in the image may be cluttered by other irrelevant objects; and finally the imperfection in the optical parts of camera and post-processing of image may degrade the image quality by introducing blurriness, pepper-salt noise, and blobs of artifacts. When handling these kinds of noise and distortions that may be present in the input data, Artificial Neural Network has been found to be an important tool in machine learning due to its capability of automatic modeling of uncertainties. The simplest neural network may be the following linear regression model:

$$\inf_{\mathbf{M}} \|\mathbf{Y} - \mathbf{X}\mathbf{M}\|, \tag{1.1}$$

where $\mathbf{X}$ is the input data, $\mathbf{Y}$ is the ground truth labels, and $\mathbf{M}$ is the parameters of the neural network model. Based on this, modern neural networks introduce layer-wise structure and non-linearity to improve the ability of modeling more complex relations between input data and labels. An example two-layer neural network is:

$$\inf_{\mathbf{M}_1,\mathbf{M}_2} g(\mathbf{Y},\ h_2(h_1(\mathbf{X}\mathbf{M}_1)\mathbf{M}_2)), \tag{1.2}$$

where $\mathbf{M}_1$ and $\mathbf{M}_2$ are the parameters of two layers of the neural network, h_1 and h_2 are activation functions, and g measures the discrepancy between the prediction of neural network from the labels $\mathbf{Y}$. In general, a neural network may be described by its layout, coefficients and activiation functions, which we sometimes abbreviated as $f(\cdot;\theta)$.

A common scenario of using neural network consists of three steps:

1. Collecting of training data $\mathbf{X}$ and their labels $\mathbf{Y}$.

2. Train a neural network that can transform the input data to the labels, within some

tolerance of errors. I.e., find θ and f such that $f(\mathbf{X};\theta) \approx \mathbf{Y}$ by optimization below:

$$\inf_{\theta} g(\mathbf{Y},\ f(\mathbf{X};\theta)). \tag{1.3}$$

3. Deploy the neural network model in applications. That is given some new data $\mathbf{X}^{'}$, output $f(\mathbf{X}^{'};\theta)$.

The success of applying a neural network model depends on many factors in the previous steps, which we will discuss below one by one.

First, there is a trade-off between accuracy of predictions of neural network and its usability in practical applications. In general, using a neural network with more layers, and layers with more parameters will improve the "modeling capability" of the neural network, leading to smaller value in 1.3. However, more layers and more parameters in general calls for larger amount of computation and increases the storage requirement, which limits its use in practice. Moreover, in scenarios like doing Speech Recognition from a mobile phone that may not be connected to network, the device capacity of computation and storage may only be a small fraction of that of cloud servers in data centers. Model approximation, which aims to approximate a given neural network by a neural network with less computation and storage requirement, is therefore an important and useful technique to help adapt neural network models to the practical limits.

Second, there often exists some irrelevant degrees of freedom in input data that add extra complications to the machine learning task. Take Optical Character Recognition on natural images for example, a letter "R", even when rotated by dozens of degrees, still counts as a "R" to most of the inspectors. However, a neural network that takes the raw pixel values of the image as input, would find the rotated "R" very different form those of a up-right "R", and need be "taught" this fact by providing sufficient training data. Data normalization, which can eliminate some degrees of freedom from the data, is therefore important in reducing the complexity of input data and consequently improve prediction quality of a neural network.

Third, there are scenarios in which a neural network is used to give predictions that have structures. An example task is image denoising, in which a neural network, like Autoencoder[106, 132], is presented an image possbily corrupted by noise, and trained to produce an image with noise suppressed as output. However, as such a network will have many output nodes, the quality of prediction is limited by the amount of available training data to suppress overfitting and cost of computation. To remedy this, we note that natural images, in contrast to random values, tends to exhibit some regularities, with low-rankness being one of extensively studied property [19, 16, 15, 89]. Though low-rankness may be enforced on the output as an extra regularization term in objective function of the neural network, it is not easy to parallelize the costly computation of the regularization term. It is therefore of interest to investigate methods to efficiently enforce some structural constraints on the outputs of the neural networks.

1.1 ▪ Contributions

In this paper, we study matrix decomposition and approximation techniques that can be applied to tackle the problems presented in the above section.

1.1.1 ▪ Kronecker Fully-Connected layers as replacement for Fully-Connected layers

Our first contribution is a new type of layer that can be used as replacement for the Fully-Connected layer in a neural network, which we tentatively call Kronecker Fully-Connected layer(KFC). In particular, assume there is a FC layer as below in a neural network:

$$\mathbf{L}_a = h(\mathbf{L}_{a-1}\mathbf{M}_a + \mathbf{b}_a), \tag{1.4}$$

where $\mathbf{L}_i$ is the output of the i-th layer of the neural network, $\mathbf{M}_a \in \mathbb{F}^{m\times n}$ is often referred to as "weight term" and $\mathbf{b}_a$ as "bias term" of the a-th layer. In the simplest case, we can use the following KFC layer to replace it:

$$\mathbf{L}_a = h(\mathbf{L}_{a-1}(\mathbf{M}_{a1} \otimes \mathbf{M}_{a2}) + \mathbf{b}_a), \tag{1.5}$$

where $\mathbf{M}_{a1} \in \mathbb{F}^{m_1\times n_1}$, $\mathbf{M}_{a2} \in \mathbb{F}^{m_2\times n_2}$, and $m_1 m_2 = m$, $n_1 n_2 = n$.

There are several benefits in replacing FC layer with KFC layer. First, we can reduce the number of parameters in weight term from mn to $m_1 n_1 + m_2 n_2$. Second, the computation complexity of $\mathbf{L}_{a-1}(\mathbf{M}_{a1} \otimes \mathbf{M}_{a2})$ is $O(kmn(\frac{1}{n_2} + \frac{1}{m_1}))$, in comparison to $O(kmn)$ of FC when cubic complexity is assumed for matrix computation. In its most general form, a KFC layer is like:

$$\mathbf{L}_a = h(\mathbf{L}_{a-1}(\sum_{c=1}^{C}\sum_{d=1}^{D}\sum_{j=1}^{J}\sum_{i=1}^{k}\mathbf{U}^a(\mathbf{A}_{ij} \otimes \mathbf{B}_{ij})\mathbf{U}^b) + \mathbf{b}_a), \tag{1.6}$$

where C, D, J, K ranges over different amount of shifting of the matrix, different shapes of $\mathbf{A}$ and $\mathbf{B}$, number of components in Kronecker Tensor Product, respectively.

Experiments on MNIST, SVHN and a Chinese Character Recognition dataset verifies the efficacy of this approach. In particular, given a model trained on SVHN dataset, we can construct a new KFC model with 73% reduction in total number of parameters, while the error only rises mildly. In contrast, using low-rank method based on row/column vectors can only achieve 35% reduction in total number of parameters given similar quality degradation allowance.

1.1.2 ▪ Group Orbit Optimization Framework and Data Normalization for Neural Network

As the second contribution, we propose a framework that reinterprets matrix decompositions and matrix norms based on optimizations over group orbit as below:

$$\inf_{\mathbf{G}\in\mathfrak{G}} \phi(\mathbf{GM}), \tag{1.7}$$

where $\mathfrak{G}$ is some group and $\mathbf{M}$ is the input data organized into a matrix.

As an illustrative example, we prove that nuclear norm of a matrix, i.e. the sum of all singular values, is nothing but minimum of L_1-norm over orbit of Kronecker product unitary group as below:

$$\|\mathbf{M}\|_1 \geq \inf_{\mathbf{U},\mathbf{V}\in\mathfrak{U}} \|\mathbf{UMV}^*\|_1 = \|\mathbf{M}\|_* = \sum_i \sigma_i(\mathbf{M}). \tag{1.8}$$

Moreover, for Schatten p-metric, we have:

$$\|\mathbf{M}\|_p \geq \inf_{\mathbf{U},\mathbf{V}\in\mathfrak{U}} \|\mathbf{UMV}^*\|_p = \|\mathbf{M}\|_{*p} \quad \text{when } 0 < p < 2, \tag{1.9}$$

and

$$\|\mathbf{M}\|_p \leq \sup_{\mathbf{U},\mathbf{V}\in\mathfrak{U}} \|\mathbf{UMV}^*\|_p = \|\mathbf{M}\|_{*p} \quad \text{when } p > 2. \tag{1.10}$$

For data normalization, we propose a point-cloud normalization example based on optimization over the orbit of the Special Linear Group as below:

$$\inf_{\mathbf{G}\in\mathfrak{SL}(n)} \|\mathbf{MG}\|_\infty. \tag{1.11}$$

With the help of matrix functions constructed from the Group Orbit Optimization(GOO) framework, we are able to increase accuracy of predictions for some tasks in Optical Character Recognition. I.e., given a neural network task as in 1.3, we transform it to the following optimization:

$$\inf_{\theta} g(\mathbf{Y},\ f(\mathbf{X}, \hat{\mathbf{X}}; \theta)), \tag{1.12}$$

where $\hat{\mathbf{X}} = \arg\min h(\mathbf{X})$ is optimum of function h.

Experiments on MNIST and SVHN demonstrate that this approach can improve the accuracy of predictions of neural network.

1.1.3 ▪ Multilinear Map layer: Enforcing Structural Constraint on Prediction

As the third contribution, by proposing a new kind of output layer to explicitly encode the low-rankness of the output based on a hybrid of Kronecker and dot product, we are able to increase the quality of denoising and reduce amount of computation at the same time.

Assume each output instance of a neural network is an image represented by a tensor of order 3:

$$\mathcal{T} \in \mathbb{F}^{C \times H \times W}, \tag{1.13}$$

where C, H and W are number of channels, height and width of the image respectively.

We approximate $\mathcal{T}$ by the following multilinear map between $\mathcal{A} \in \mathbb{F}^{C_1 \times H_1 \times W_1}$ and $\mathcal{B} \in \mathbb{F}^{C_1 \times C_2 \times H_2 \times W_2}$:

$$T_{c,h,w} = T_{c,(h_1,h_2),(w_1,w_2)} \approx \sum_{k} \sum_{c_1} A_{k,c_1,h_2,w_2} B_{k,c,c_1,h_1,w_1}. \tag{1.14}$$

Hence we would refer to layers constructed following 8.31 as HKD layers. The general name of MLM layers will refer to all possible kinds of layers that can be constructed from some kind of multilinear map. For example, we also explore using Kronecker Tensor Product in MLM layers:

$$\mathcal{T} \approx \sum_{i=1}^{K} \mathcal{A}_i \otimes \mathcal{B}_i. \tag{1.15}$$

The alternative practice of enforcing low-rankness on the prediction would be introducing some regularizer on $\mathcal{T}$ like the following tensor unfold norm:

$$r(\mathcal{T}) = \sum_{c} \|\mathbf{T}_c\|_*, \tag{1.16}$$

where $\mathbf{T}_c$ are sub-matrices of $\mathcal{T}$. However, in this formulation, the neural network is still required to produce $\mathcal{T}$ as output, resulting in CHW number of output nodes. In comparison, HKD layer would only require $C_1H_2W_2 + CC_1H_1W_1 = CHW(\frac{C_1}{CH_1W_1} + \frac{C_1}{H_2W_2})$ number of output nodes, which could be significantly smaller when H, W are relatively large.

Empirical results also demonstrate the effectiveness of MLM layer in reducing amount of computation, number of paramters and improving the quality of prediction on MNIST and SVHN datasets.

1.2 ▪ Overview of Thesis

The chapters are roughly grouped into a few parts: Chapter 2, Chapter 4 gives background materials and a short introduction to neural network. Chapter 3 will cover the Multilinear algebra

tools used in the rest of paper; in Chapter 5, we apply the tools developed in previous chapters to several neural network tasks. Chapter 6 presents our framework of GOO for reinterpreting matrix decompositions and norms; Chapter 7 uses GOO to augment the input data to improve accuracy of predictions of neural networks; Chapter 8 explores using multilinear map to enforce structural constraint on the predictions of the neural network. Finally in Chapter 9 we conclude our work and discuss the future work.

Chapter 2
Notation

2.1 ▪ Matrix operation notation

In this paper, we let $\mathbf{I}_r$ denote the $r \times r$ identity matrix. Given an $n \times m$ matrix $\mathbf{X} = [x_{ij}]$, we denote $|\mathbf{X}| = [|x_{ij}|]$ and $\text{vec}(\mathbf{X}) = [x_{11}, \ldots, x_{n1}, x_{12}, \ldots, x_{nm}]^\top$. The ℓ_p-norm of $\mathbf{X}$ is defined by

$$\|\mathbf{X}\|_p \stackrel{\text{def}}{=\!=} (\sum_{ij} |x_{ij}|^p)^{\frac{1}{p}}$$

for $p \geq 0$. Note that we abuse the notation a little bit as $\|\mathbf{X}\|_p$ is not a norm when $p < 1$. For clarity, we will refer to norm function as convex norm function . When $p = 2$, it is also called the Frobenius norm and usually denoted by $\|\mathbf{X}\|_F$. When applied to vector $\mathbf{x}$, $\|\mathbf{x}\|_2$ is the ℓ_2-norm and it is shortened as $\|\mathbf{x}\|$. The dual norm of the p-norm where $p \geq 1$ is equivalent to the q-norm, where $\frac{1}{p} + \frac{1}{q} = 1$. We let $\|\mathbf{X}\|_{*p}$ denote the Schatten p-metric; that is, it is the ℓ_p norm of the vector of the singular values of $\mathbf{X}$.

Assume that $\mathbb{F}$ is some number field. Let $\mathbf{X}^{\text{c}}$ be the complex conjugate of $\mathbf{X}$, and $\mathbf{X}^*$ be the complex conjugate transpose of $\mathbf{X}$. Let $\text{dg}(\mathbf{M})$ be a vector consisting of the diagonal entries of $\mathbf{M}$, and $\text{diag}(\mathbf{v})$ be a matrix with $\mathbf{v}$ as its diagonals.

Given two matrices $\mathbf{A}$ and $\mathbf{B}$, $\mathbf{A} \odot \mathbf{B}$ is their Hadamard product and $\mathbf{A} \otimes \mathbf{B}$ is the Kronecker product. Similarly, $\mathbf{x} \otimes \mathbf{y}$ is the Kronecker product of vectors $\mathbf{x}$ and $\mathbf{y}$. For groups $\mathfrak{G}_1$ and $\mathfrak{G}_2$, we denote group $\{\mathbf{G}_1 \otimes \mathbf{G}_2 : \mathbf{G}_1 \in \mathfrak{G}_1, \mathbf{G}_2 \in \mathfrak{G}_2\}$ as $\mathfrak{G}_1 \otimes \mathfrak{G}_2$. The Kronecker sum for two square matrices $\mathbf{A} \in \mathbb{F}^{m \times m}, \mathbf{B} \in \mathbb{F}^{n \times n}$ is defined as

$$\mathbf{A} \oplus \mathbf{B} = \mathbf{A} \otimes \mathbf{I}_n + \mathbf{I}_m \otimes \mathbf{B}.$$

Definition 2.1. *A matrix $\mathbf{A} \in \mathbb{F}^{m \times n}$ is said to be pseudo-diagonal if there exist permutation matrices $\mathbf{P}$ and $\mathbf{Q}$ such that $\mathbf{PAQ}^\top$ is diagonal.*

Remark 2.2. *Note that a diagonal matrix is also pseudo-diagonal.*

Lemma 2.3. *Given a pseudo-diagonal matrix $\mathbf{A}$, we have that*

(i) *$\mathbf{A}^*\mathbf{A}$, $\mathbf{AA}^*$, $\mathbf{A}^\top\mathbf{A}$ and $\mathbf{AA}^\top$ are diagonal.*

(ii) *There exists a row permutation matrix $\mathbf{P}$ such that $\mathbf{PA}$ is diagonal.*

(iii) *There exists a row permutation matrix $\mathbf{P}$ such that $\mathbf{AP}^\top$ is diagonal.*

We let Poly $(\mathbf{M})$ be the polyhedral formed by points with coordinates being rows of $\mathbf{M}$, and $\mu(\text{Poly}\,(\mathbf{M}))$ be the Lebesgue measure of Poly $(\mathbf{M})$. We let Rasterize(Poly $(\mathbf{M})$) be a matrix $\mathbf{Z}$ where z_{ij} is the image pixel value at coordinate (i, j) of image rasterized from polyhedral Poly $(\mathbf{M})$ with unit grid.

2.2 ▪ Tensor operation notation

The notation of tensor operations used in this paper mostly follows that of [72]. Given an order-k tensor $\mathcal{X} \in \mathbb{F}^{n_1 \times n_2 \times \ldots \times n_k}$ and k matrices $\{\mathbf{U}_i\}_{i=1}^k$ where $\mathbf{U}_i \in \mathbb{F}^{m_i \times n_i}$, we define $\times_k$ to be the inner product over the k-th mode. That is, if $\mathcal{Y} = \mathcal{X} \times_a \mathbf{U}_a \in \mathbb{F}^{n_1 \times n_2 \times \ldots \times n_{a-1} \times m_a \times n_{a+1} \times \ldots \times n_k}$, then

$$y_{i_1 \cdots i_{a-1} j i_{a+1} \cdots i_k} = \sum_{i_a=1}^{n_a} x_{i_1 i_2 \cdots i_k} u_{j i_a}.$$

For shorthand, we denote

$$\prod_i \mathcal{X}\mathbf{U}_i \stackrel{\text{def}}{=\!=} \mathcal{X} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 \cdots \times_k \mathbf{U}_k.$$

Here $\mathcal{Y} = \prod_i \mathcal{X}\mathbf{U}_i$ when $\forall i, m_i = n_i$ is also known as the Tucker decomposition in the literature [125]. With this notation, the SVD of a real matrix $\mathbf{M} = \mathbf{U}_1 \mathbf{\Sigma} \mathbf{U}_2^\top$ can be written as

$$\mathbf{M} = \mathbf{\Sigma} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 = \prod_{i=1}^{2} \mathbf{\Sigma}\mathbf{U}_i.$$

Using the vectorization operation for tensor, we have

$$\text{vec}\,(\prod_i \mathcal{X}\mathbf{U}_i) = [\mathbf{U}_n \otimes \mathbf{U}_{n-1} \otimes \cdots \otimes \mathbf{U}_1]\,\text{vec}\,(\mathcal{X}) \stackrel{\text{def}}{=\!=} \otimes_i^{\downarrow} \mathbf{U}_i\,\text{vec}\,(\mathcal{X}),$$

where we denote $\otimes_i^{\downarrow} \mathbf{U}_i$ as shorthand for $\mathbf{U}_n \otimes \mathbf{U}_{n-1} \otimes \cdots \otimes \mathbf{U}_1$.

We let $\text{index}_{n_1, n_2, \ldots, n_k}(I)$ be a map from a sequence of indices $I = i_1, i_2, \ldots, i_k$ to an integer such that

$$[\text{vec}\,\mathcal{X}]_{\text{index}_{n_1, n_2, \ldots, n_k}(i_1, i_2, \ldots, i_k)} = \mathcal{X}_{i_1, i_2, \ldots, i_k}. \tag{2.1}$$

We note that $\text{index}_N^{-1}(n)$ is well-defined.

The unfold operation maps a tensor to a tensor of lower order and is defined by

$$\text{fold}_J^{-1} : \mathbb{F}^{n_1 \times n_2 \times \ldots \times n_k} \mapsto \mathbb{F}^{m_1 \times m_2 \times \ldots \times m_l},$$

where J is an index set grouping of the indices $I = \{1, 2, \ldots, k\}$ into sets $J = \{J_1, J_2, \ldots, J_l\}$, $m_o = \prod_{t \in J_o} t$, and satisfies:

$$\text{vec}[\text{fold}_J^{-1}(\mathcal{A})] = \text{vec}(\mathcal{A}).$$

When unfolding a single index, i.e., $J = \{\{j\}, I - \{j\}\}$, we also denote fold_J^{-1} as fold_j^{-1}.

The ℓ_p-norm of tensor $\mathcal{A}$ is defined as

$$\|\mathcal{A}\|_p \stackrel{\text{def}}{=\!=} \|\,\text{fold}_i^{-1}\,\mathcal{A}\|_p$$

for an arbitrary mode i. For tensors $\mathcal{A}, \mathcal{B} \in \mathbb{F}^{n_1 \times n_2 \times \ldots \times n_k}$, $\langle \mathcal{A}, \mathcal{B} \rangle$ is their Frobenius inner product defined as:

$$\langle \mathcal{A}, \mathcal{B} \rangle \stackrel{\text{def}}{=\!=} \langle \text{vec}(\mathcal{A}), \text{vec}(\mathcal{B}) \rangle.$$

Finally, given $f : \mathbb{F} \to \mathbb{F}$ and $\mathcal{T} \in \mathbb{F}^{n_1 \times n_2 \times \ldots \times n_k}$, $f(\mathcal{T})$ is defined as a tensor-valued function with f applied to each entry of $\mathcal{T}$. Therefore, $f(\mathcal{T}) \in \mathbb{F}^{n_1 \times n_2 \times \ldots \times n_k}$. When $f(x) = |x|$, we denote $f(\mathcal{T})$ as $|\mathcal{T}|$.

2.3 ▪ Group notation

$\mathfrak{O}$ is the orthogonal group over real field $\mathbb{R}$. $\mathfrak{SO}$ is the special orthogonal group over $\mathbb{R}$. $\mathfrak{U}$ is the unitary group over complex field. We let $\mathfrak{UUT}(n)$ denote the upper-unit-triangular group and $\mathfrak{LUT}(n)$ denote the lower-unit-triangular group, both of which have all entries along the diagonals being 1. $\mathfrak{H}$ is the group formed by (calibrated) homography transform below:

$$\mathbf{H}_{2w} = \mathbf{R}_{2w}(\mathbf{I}_3 + \mathbf{p}_2\frac{\mathbf{n}^\top}{d}),$$

where $\mathbf{R}_{2w} \in \mathfrak{SO}$ is attitude of the camera; $\mathbf{p}_2$ is position of the camera, and $\mathbf{n}^\top\mathbf{x} = d$ is equation of the object plane.

Chapter 3
Multilinear Map

In this section, we cover the basic linear algebra tools used in the rest of this paper, in particular several multilinear map. An introduction to Kronecker product of matrices is available in [129, 130, 92, 93]. Kronecker tensor product, which is a generalization of Kronecker product to general tensors, has been proposed in [103, 104]. Convolution as used in a neural network is defined in [83].

3.1 ▪ Kronecker Product of Matrix

Kronecker product is defined as follows for matrices:

Definition 3.1. *If* $\mathbf{A}$ *is an* $m \times n$ *matrix and* $\mathbf{B}$ *is a* $p \times q$ *matrix, then the Kronecker product* $\mathbf{A} \otimes \mathbf{B}$ *is the* $mp \times nq$ *block matrix:*

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix}, \tag{3.1}$$

or more explicitly as:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & \cdots & a_{11}b_{1q} & \cdots & \cdots & a_{1n}b_{11} & a_{1n}b_{12} & \cdots & a_{1n}b_{1q} \\ a_{11}b_{21} & a_{11}b_{22} & \cdots & a_{11}b_{2q} & \cdots & \cdots & a_{1n}b_{21} & a_{1n}b_{22} & \cdots & a_{1n}b_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ a_{11}b_{p1} & a_{11}b_{p2} & \cdots & a_{11}b_{pq} & \cdots & \cdots & a_{1n}b_{p1} & a_{1n}b_{p2} & \cdots & a_{1n}b_{pq} \\ \vdots & \vdots & & \vdots & \ddots & & \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots & & \ddots & \vdots & \vdots & & \vdots \\ a_{m1}b_{11} & a_{m1}b_{12} & \cdots & a_{m1}b_{1q} & \cdots & \cdots & a_{mn}b_{11} & a_{mn}b_{12} & \cdots & a_{mn}b_{1q} \\ a_{m1}b_{21} & a_{m1}b_{22} & \cdots & a_{m1}b_{2q} & \cdots & \cdots & a_{mn}b_{21} & a_{mn}b_{22} & \cdots & a_{mn}b_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{p1} & a_{m1}b_{p2} & \cdots & a_{m1}b_{pq} & \cdots & \cdots & a_{mn}b_{p1} & a_{mn}b_{p2} & \cdots & a_{mn}b_{pq} \end{bmatrix}. \tag{3.2}$$

Remark 3.2. *An example of Kronecker product is:*

$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} \otimes \begin{bmatrix}0 & 5\\ 6 & 7\end{bmatrix} = \begin{bmatrix}1\cdot 0 & 1\cdot 5 & 2\cdot 0 & 2\cdot 5\\ 1\cdot 6 & 1\cdot 7 & 2\cdot 6 & 2\cdot 7\\ 3\cdot 0 & 3\cdot 5 & 4\cdot 0 & 4\cdot 5\\ 3\cdot 6 & 3\cdot 7 & 4\cdot 6 & 4\cdot 7\end{bmatrix} = \begin{bmatrix}0 & 5 & 0 & 10\\ 6 & 7 & 12 & 14\\ 0 & 15 & 0 & 20\\ 18 & 21 & 24 & 28\end{bmatrix} \tag{3.3}$$

We note the following properties of Kronecker product:

$$\mathbf{A}\otimes(\mathbf{B}+\mathbf{C}) = \mathbf{A}\otimes\mathbf{B} + \mathbf{A}\otimes\mathbf{C}, \tag{3.4}$$
$$(\mathbf{A}+\mathbf{B})\otimes\mathbf{C} = \mathbf{A}\otimes\mathbf{C} + \mathbf{B}\otimes\mathbf{C}, \tag{3.5}$$
$$(k\mathbf{A})\otimes\mathbf{B} = \mathbf{A}\otimes(k\mathbf{B}) = k(\mathbf{A}\otimes\mathbf{B}), \tag{3.6}$$
$$(\mathbf{A}\otimes\mathbf{B})\otimes\mathbf{C} = \mathbf{A}\otimes(\mathbf{B}\otimes\mathbf{C}). \tag{3.7}$$

Kronecker product is related to outer product by the $\mathscr{R}$ operator defined in [129, 128] which shuffles indices.

$$\mathscr{R}(\mathbf{A}\otimes\mathbf{B}) = \operatorname{vec}\mathbf{A}(\operatorname{vec}\mathbf{B})^\top. \tag{3.8}$$

3.2 ▪ KPSVD: Kronecker Product Singular Value Decomposition

KPSVD is introduced in [129] as follows and has various applications[124, 148]:

$$\mathbf{A} \approx \mathbf{A}_r = \sum_{i=1}^{r} \sigma_i \mathbf{U}_i \otimes \mathbf{V}_i. \tag{3.9}$$

KPSVD bears strong connection with SVD, in fact it can be turned into following SVD using $\mathscr{R}$ operator.

$$\mathscr{R}(\mathbf{A}) \approx \mathscr{R}(\mathbf{A}_r) = \sum_{i=r}^{r} \sigma_i \operatorname{vec}\mathbf{U}_i \otimes (\operatorname{vec}\mathbf{V}_i)^\top. \tag{3.10}$$

However, the shuffling may be important in practice. In contrast to SVD which reflects the low rank structure of rows and columns of the matrix, KPSVD reflects the low rank structure of

blocks of $\mathbf{A}$. This can be seen from the following example:

$$\mathbf{A} = \mathbf{B} \otimes \mathbf{C} \tag{3.11}$$

$$= \left\| \left(\begin{array}{cc|cc} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ \hline a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \\ \hline a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} \\ a_{6,1} & a_{6,2} & a_{6,3} & a_{6,4} \end{array} \right) - \left(\begin{array}{cc} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \\ b_{3,1} & b_{3,2} \end{array} \right) \otimes \left(\begin{array}{cc} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{array} \right) \right\|_F \tag{3.12}$$

$$= \left\| \left(\begin{array}{cc} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ \hline a_{1,3} & a_{1,4} \\ a_{2,3} & a_{2,4} \\ \hline a_{3,1} & a_{3,2} \\ a_{4,1} & a_{4,2} \\ \hline a_{3,3} & a_{3,4} \\ a_{4,3} & a_{4,4} \\ \hline a_{5,1} & a_{5,2} \\ a_{6,1} & a_{6,2} \\ \hline a_{5,3} & a_{5,4} \\ a_{6,3} & a_{6,4} \end{array} \right) - \left(\begin{array}{c} b_{1,1} \\ b_{2,1} \\ b_{3,1} \\ b_{1,2} \\ b_{2,2} \\ b_{3,2} \end{array} \right) \left(\begin{array}{cc} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{array} \right) \right\|_F \tag{3.13}$$

$$= \left\| \left(\begin{array}{cc|cc} a_{1,1} & a_{2,1} & a_{1,2} & a_{2,2} \\ \hline a_{3,1} & a_{4,1} & a_{3,2} & a_{4,2} \\ \hline a_{5,1} & a_{6,1} & a_{5,2} & a_{6,2} \\ \hline a_{1,3} & a_{2,3} & a_{1,4} & a_{2,4} \\ \hline a_{3,3} & a_{4,3} & a_{3,4} & a_{4,4} \\ \hline a_{5,3} & a_{6,3} & a_{5,4} & a_{6,4} \end{array} \right) - \left(\begin{array}{c} b_{1,1} \\ b_{2,1} \\ b_{3,1} \\ b_{1,2} \\ b_{2,2} \\ b_{3,2} \end{array} \right) \left(\begin{array}{cccc} a_{1,1} & a_{2,1} & a_{1,2} & a_{2,2} \end{array} \right) \right\|_F \tag{3.14}$$

Hence $\mathbf{C}$ captures the low rank components of local patches of $\mathbf{A}$. As an example, rank of KPSVD of $\mathbf{M}$ may be different from rank of matrix $\mathbf{M}$.

Example 3.3. For

$$\mathbf{A} = \left(\begin{array}{cccc} 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \end{array} \right), \tag{3.15}$$

we have $\operatorname{rank} \mathbf{A} = 2$, but as

$$\mathbf{A} = \left(\begin{array}{cc} 1 & 2 \\ 3 & 4 \end{array} \right) \otimes \left(\begin{array}{cc} 1 & 1 \end{array} \right), \tag{3.16}$$

$\mathbf{A}$ is of rank one in KPSVD. ∎

3.3 ▪ Kronecker Tensor Product

WE note that Kronecker product $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$ is closely related to Tensor product $T_{ijkl} = A_{ij} B_{kl}$, with the difference being that Kronecker product ensures the result is still of the same order as $\mathbf{A}$ and $\mathbf{B}$. As this property turns out to be of importance in practice, we would like to generalize Kronecker product to tensors.

Definition 3.4. *Kronecker tensor product[103, 104] is defined for two tensors of the same order k. I.e., for two tensors:*

$$\mathcal{A} \in \mathbb{F}^{m_1 \times m_2, \cdots, \times m_k}, \tag{3.17}$$

and

$$\mathcal{B} \in \mathbb{F}^{n_1 \times n_2, \cdots, \times n_k}, \tag{3.18}$$

we define Kronecker product of tensor as

$$(\mathcal{A} \otimes \mathcal{B})_{i_1, i_2, \cdots, i_k} \coloneqq A_{\lfloor i_1/m_1 \rfloor, \lfloor i_2/m_2 \rfloor, \cdots, \lfloor i_k/m_k \rfloor} B_{i_1 \bmod m_1, i_2 \bmod m_2, \cdots, i_k \bmod m_k} \tag{3.19}$$

where

$$\mathcal{A} \otimes \mathcal{B} \in \mathbb{F}^{m_1 n_1 \times m_2 n_2, \cdots, \times m_k n_k}. \tag{3.20}$$

Remark 3.5. *It can be seen that Kronecker product of tensor of order two is the same as the Kronecker product of matrices.*

Approximation of a tensor $\mathcal{T} \in \mathbb{F}^{\mathbf{n}_1 \times \mathbf{n}_2 \times \cdots \times n_r}$ by KTP can then be done by:

$$\mathcal{T} = \sum_{i=1}^{r} \mathcal{A}_i \otimes \mathcal{B}_i \approx \sum_{i=1}^{r'} \mathcal{A}_i \otimes \mathcal{B}_i, \tag{3.21}$$

where $r' \leq r$.

In practice, KTP approximation may also be applied to the shifted versions of the tensor [103] as follows:

$$\mathscr{S}(\mathcal{T}) = \sum_{i=1}^{r} \mathcal{A}_i \otimes \mathcal{B}_i, \tag{3.22}$$

where shift operator $\mathscr{S}(\mathcal{T})_{i_1, i_2, \cdots, i_r} = \mathcal{T}_{i_1 - s_1, i_2 - s_2, \cdots, i_r - s_r}$. Indices shifted outside the tensor may be set to zero or are replicated etc., depending on the border condition.

3.4 ▪ Pseudo Inverse of Kronecker Tensor Product

We can then generalize Nearest Kronecker Product problem to tensor as either

$$\min_{\mathcal{X}} \|\mathcal{A} - \mathcal{X} \otimes \mathcal{B}\|_F \tag{3.23}$$

or

$$\min_{\mathcal{X}} \|\mathcal{A} - \mathcal{B} \otimes \mathcal{X}\|_F \tag{3.24}$$

For simplicity of notation, we define the pseudo-inverse operator of Kronecker product:

Definition 3.6. *We define right and left pseudo-inverse of Kronecker tensor product as:*

$$\mathcal{A} \oslash \mathcal{B} \coloneqq \arg\min_{\mathcal{X}} \|\mathcal{A} - \mathcal{X} \otimes \mathcal{B}\|_F, \tag{3.25}$$

and similarly

$$\mathcal{B} \bigcirc\!\!\!\!\backslash\ \mathcal{A} \coloneqq \arg\min_{\mathcal{X}} \|\mathcal{A} - \mathcal{B} \otimes \mathcal{X}\|_F \tag{3.26}$$

By applying the shuffle operator $\mathscr{R}$ for Kronecker product of tensor we can obtain:

$$\min_{\mathcal{X}} \|\mathcal{A} - \mathcal{X} \otimes \mathcal{B}\|_F = \min_{\mathcal{X}} \|\mathscr{R}(\mathcal{A}) - \operatorname{vec} \mathcal{X} \otimes \operatorname{vec} \mathcal{B}\|_F = \min_{\mathcal{X}} \|\mathscr{R}(\mathcal{A}) - \operatorname{vec} \mathcal{X} (\operatorname{vec} \mathcal{B})^\top\|_F \tag{3.27}$$

3.5 ▪ Multi-channel 2D Convolution

Circular discrete convolution of two vectors f and g is defined as:

$$(f * g)[n] = \sum_{m=0}^{N-1} f[m] g[n-m] \tag{3.28}$$

$$, \tag{3.29}$$

where f and g are assumed to be defined over integer values in $[1, N]$.

Given a tensor $\mathcal{T} \in \mathbb{F}^{C \times H_1 \times W_1}$ and weights tensor of a convolution layer $\mathcal{W} \in \mathbb{F}^{K \times C \times H_2 \times W_2}$, the 2D convolution is defined as:

$$(\mathcal{T} \star \mathcal{W})_{k,h,w} = \sum_{c=1}^{C} \sum_{h_2=1}^{H_2} \sum_{w_2=1}^{W_2} T_{c,h-h_2,w-w_2} W_{k,c,h_2,w_2}, \tag{3.30}$$

where indices out of range of tensor is either assumed to be zero or repeated depending on the border condition.

Chapter 4

Artificial Neural Network

Early use of Artificial Neural Network (hereafter referred to simply as neural network) can be traced back to as early as 1960s [109]. Throughout its history, many computation models [83, 75, 132, 48, 85, 116, 42, 97] has been proposed as neural networks, with vastly different structure, training method and dynamics. Below we give a brief introduction to structure of those neural networks that may be used in this thesis.

4.1 ▪ Structure of Neural Networks

The Supervised Learning task in machine learning aims to find an approximate function given input $\mathbf{X} \in \mathbb{F}^{n \times k}$ and output $\mathbf{Y} \in \mathbb{F}^{n \times m}$, where n is number of data instances, k is number of feature in a data instance, and m is number of feature in an output instance. The problem can then be formulated as finding an solution to the following equation:

$$\mathbf{Y} = f(\mathbf{X}). \tag{4.1}$$

In cases when an exact solution does not exist or is infeasible to find, the following functional optimization is solved in lieu of 4.1, where g is a measure of discrepancy between Y and $f(\mathbf{X})$ and is often called objective function or cost function:

$$\inf_f g(\mathbf{Y}, f(\mathbf{X})). \tag{4.2}$$

First we look at the simple problem of approximating the relationship between X and Y by a linear function. When we require that k and m be finite numbers and limit f to be a linear function, then the problem 4.1 can be formulated as a matrix approximation problem as below, where $\mathbf{M} \in \mathbb{F}^{k \times m}$:

$$\mathbf{Y} = \mathbf{X}\mathbf{M}. \tag{4.3}$$

When we choose g to be ℓ_2-norm in 4.2, the problem of 4.3 becomes linear regression:

$$\inf_{\mathbf{M}} \|\mathbf{Y} - \mathbf{X}\mathbf{M}\|, \tag{4.4}$$

which has an closed form solution:

$$\hat{\mathbf{M}} = \mathbf{X}^{+}\mathbf{Y}. \tag{4.5}$$

This is also the simplest form of a neural network. Simple as it is, it may not be able to model the nonlinear relationship between $\mathbf{X}$ and $\mathbf{Y}$. Next we move one step further and introduce a nonlinear function h so that 4.1 becomes a nonlinear matrix approximation:

$$\inf_{\mathbf{M}} g(\mathbf{Y},\ h(\mathbf{XM})). \tag{4.6}$$

The model 4.8 is also known as Generalized Linear Model[95], which includes linear regression, logistic regression, Poisson regression as special cases. The function h corresponds to "activiation function" in nerual networ terminology. In fact 4.8 is exactly an example of 1-layer feed-forward model in neural network terminology.

4.2 ▪ Layers of Neural Networks

We may introduce "layers" into 4.8 by having multiple optimization variables. As a minimal example, when we have two layers, 4.8 becomes below, where $\mathbf{M}_1 \in \mathbb{F}^{k\times z_1}$ and $\mathbf{M}_2 \in \mathbb{F}^{z_1\times m}$:

$$\inf_{\mathbf{M}_1,\mathbf{M}_2} g(\mathbf{Y},\ h(\mathbf{XM}_1\mathbf{M}_2)). \tag{4.7}$$

In neural network terminology, $\mathbf{M}_1$ and $\mathbf{M}_2$ in 4.9 are called first layer and second layer, respectively. However, the associativity of matrix multiplication means that 4.9 is different from 4.8 only when $z_1 < \operatorname{rank}(\hat{\mathbf{M}})$, in which case 4.9 is equivalent to the rank-constrained variant of generalized linear regression model:

$$\inf_{\mathbf{M}} g(\mathbf{Y},\ h(\mathbf{XM})) \text{ s.t. } \operatorname{rank}(\mathbf{M}) \le z_1. \tag{4.8}$$

However, if we apply a nonlinear function at each layer, we will get the general form of feed-forward neural network, which include the above formualations as speciall cases. We give a two-layer example as below:

$$\inf_{\mathbf{M}_1,\mathbf{M}_2} g(\mathbf{Y},\ h_2(h_1(\mathbf{XM}_1)\mathbf{M}_2)). \tag{4.9}$$

4.2.1 ▪ Fully-connected Layer and Activation Functions

A fully-connected layer without bias term is simply:

$$\mathbf{L}_a = h(\mathbf{L}_{a-1}\mathbf{M}_a) \tag{4.10}$$

In 4.9, the tuple $(h_1, \mathbf{M}_1)$ described the first layer, and $(h_2, \mathbf{M}_2)$ described the second layer. Due to this modular structure, neural network can have arbitrary number of layers. Let a neural network has $\{\mathbf{L}_i\}_{i=1}^k$ as output of layers, then the training of a neural network can be formulated as solving following optimization problem:

$$\inf_{\mathbf{M}_1,\mathbf{M}_2,\cdots,\mathbf{M}_k} g(\mathbf{Y},\mathbf{L}_k) \text{ s.t. } \mathbf{L}_i = h_i(\mathbf{L}_{i-1}\mathbf{M}_i),\ \mathbf{L}_0 = \mathbf{X},\ 1 \le i \le k. \tag{4.11}$$

Note in above formulation, we intentionally do not include the "bias term". In fact, given an optimization with "bias term" like

$$\inf_{\mathbf{M}_1,\mathbf{M}_2,\cdots,\mathbf{M}_k,\mathbf{b}_1,\mathbf{b}_2,\cdots,\mathbf{b}_k} g(\mathbf{Y},\mathbf{L}_k) \text{ s.t. } \mathbf{L}_i = h_i(\mathbf{L}_{i-1}\mathbf{M}_i + \mathbf{b}_i),\ \mathbf{L}_0 = \mathbf{X},\ 1 \le i \le k, \tag{4.12}$$

we can transform it to an equivalent form without "bias term" as

$$\inf_{\mathbf{M}_1,\mathbf{M}_2,\cdots,\mathbf{M}_k} g(\mathbf{Y},\mathbf{L}_k) \text{ s.t. } \mathbf{L}_i = a(h_i(\mathbf{L}_{i-1}\mathbf{M}_i)),\ \mathbf{L}_0 = a(\mathbf{X}),\ 1 \le i \le k, \tag{4.13}$$

where $a(\mathbf{X})$ adds an extra constant 1 along each row of $\mathbf{X}$. As 4.13 is essentially the same as 4.11 with the only difference being the activation function h_i, we will use 4.11 in the rest of this paper.

Often an elementwise function is used as the activation function. Some examples are:

1. Sign function: $h(x) = \operatorname{sgn}(x)$
2. "Sigmoid" function: $h(x) = \frac{1}{1+e^{-x}}$. The function is like Sign function but is everywhere differentiable.
3. Hyperbolic tangent: $h(x) = \tanh(x)$
4. Absolute hyperbolic tangent: $h(x) = |\tanh(x)|$
5. "ReLU"[98] function: $h(x) = \max(0, x)$
6. Capped "ReLU" function: $h(x) = \min(\max(0, x), 1)$
7. Abs function: $h(x) = |x|$
8. Capped absolute value function: $h(x) = \min(|x|, 1)$
9. Bounded identity function: $h(x) = \max(\min(x, 1), -1)$

We also find a few other functions not well-covered in literature to be usable as activation functions in some neural network configuration, which we list below:

1. Square function: $h(x) = x^2$
2. Sine function: $h(x) = \sin(x)$ [139]
3. Absolute Sine function: $h(x) = |\sin(x)|$
4. Arc tangent: $h(x) = \arctan(x)$
5. Arc sine: $h(x) = \arcsin(x)$
6. Arc hyperbolic sine: $h(x) = \operatorname{arcsinh}(x)$

An example of non-elementwise function is the "Softmax" function: $h(x_i) = \frac{1}{\sum_j e^{x_j}} e^{x_i}$. Several factors make this function special:

1. $\sum_i h(x_i) = 1$ and $\forall i,\ h(x_i) \ge 0$, hence $h(x_i)$ may be interpreted as a probability mass function.
2. In numerical implementation, it is in general desirable to have $h(x_i) > 0$, allowing us to use functions like $\log(h(x_i))$ without the need to worry about NaN values that may result from taking logarithm of zero.

4.2.2 ▪ Constrained Layers: Convolutional Layer and Locally-connected Layer

Besides the fully-connected layer described above, there are cases when putting constraints on $\mathbf{M}_i$ is desirable. In the so-called Convolutional layer[83, 87, 122], $\mathbf{M}_i$ is required to be a circulant matrix, with different variants corresponding to different handling of boundary conditions. The circulant structure of the matrix is used to represent the approximate "spatial-invariance" of neural network on the spatial dimensions of the input data.

For a circular circulant matrix $\mathbf{M}$, the following relationship holds:

$$\mathbf{M}\mathbf{x} = \mathbf{m} \star \mathbf{x}, \tag{4.14}$$

where $\mathbf{m}$ is the first column of $\mathbf{M}$ and $\star$ is the convolution operator. The above relationship can also be generated to 2D-convolution, by noting that convolution is a linear operator, there exists a matrix $\mathbf{M}$ such that given $\mathbf{m}$, we have:

$$\mathbf{M}\,\mathrm{vec}\,\mathbf{X} = \mathbf{m} \star \mathrm{vec}\,\mathbf{X}. \tag{4.15}$$

Hence if layer a is known to be a convolution layer in 4.11, like

$$\mathbf{L}_a = h_a(\mathbf{L}_{a-1} \star \mathbf{m}_a)\ , \tag{4.16}$$

we can reformulate it as a constrained form as

$$\mathbf{L}_a = h_a(\mathbf{L}_{a-1}\mathbf{M}_a)\ \ \mathsf{s.t.}\ \mathcal{A}(\mathbf{M}) = 0, \tag{4.17}$$

where $\mathcal{A}$ is a linear constraint. Altogether, we generalize 4.11 to the following constrained form of neural network as

$$\inf_{\mathbf{M}_1,\mathbf{M}_2,\cdots,\mathbf{M}_k} g(\mathbf{Y}, \mathbf{L}_k)\ \mathsf{s.t.}\ \mathbf{L}_i = h_i(\mathbf{L}_{i-1}\mathbf{M}_i),\ \mathbf{L}_0 = \mathbf{X},\ \mathcal{A}_i(\mathbf{M}_i) = 0,\ 1 \le i \le k. \tag{4.18}$$

A locally-connected layer[127] has the same non-zero support as a convolutional layer, but does not require the coefficient matrix to correpsond to a convolution. Hence the weight matrix $\mathbf{M}$ may not be circulant and a locally-connected layer often have more number of parameters than a convolutional layer of comparable shape.

4.2.3 ▪ Rank-4 Tensor Representation

In Convolutional Neural Network, a variant of neural network, the input data are often fixed-size patches extracted from images, and organized into a tensor as:

$$\mathcal{T} \in \mathbb{F}^{n\times c\times h\times w}, \tag{4.19}$$

where n is number of data instances, c corresponds to "channel" of the input image, h and w corresponds to the height and width of each patch in image respectively.

Given a rank-4 tensor, application of 2D convolutional filter produces another rank-4 tensor $\mathcal{T}'$:

$$\mathcal{T}'_{i,:,:,:} = \mathcal{T}_{i,:,:,:} \star \mathcal{W} + \mathcal{B}_{i,:,:,:}. \tag{4.20}$$

Matrix Ring Interpretation

A rank-4 tensor may be interpreted as a matrix over the ring of matrix. In this sense, $\mathcal{T}_{i,b,c,d}$ becomes a matrix $\tilde{\mathcal{T}}_{i,b}$ with each entry being a matrix. Formula 4.20 becomes a matrix product where the scalar product is replace by convolution:

$$\tilde{\mathcal{T}}'_{i,j} = \tilde{\mathcal{T}}_{i,b} \star_M \tilde{\mathcal{W}}_{b,j} + \tilde{\mathcal{B}}_{i,j}, \tag{4.21}$$

where $\star_M$ indicates the 2D matrix convolution.

4.2.4 ▪ Pooling Layer and Maxout Layer

Another interesting kind layer is the so-called Pooling layer, which enforces following equations:

$$(\mathbf{Z}_i)_{x,y} = h_i(\{(\mathbf{Z}_{i-1})_{a,b}\}_{(a,\,b)\in\mathcal{N}(x,\,y)}), \tag{4.22}$$

where $\operatorname{vec}\mathbf{Z}_i = \mathbf{L}_i$, $\mathcal{N}(x,\,y)$ is some pre-defined neighborhood of $(x,\,y)$.

Though 4.22 is superficially similar to 4.18, in general, there does not exist a linear constraint to reformulate the former into the latter when h_i is a elementwise function. As an example, when h_i is maximum over a discrete set, 4.22 is equivalent to matrix multiplication in Max-plus algebra[14], which is known to be not representable as 4.18 when h_i is a elementwise function.

For the case of mean pooling, i.e. pooling layer with h_i producing mean of value in the set, one can use the Pseudo Inverse of Kronecker Tensor Product defined in 3.26 to describe the operation of a $m \times n$ pooling layer:

$$\mathbf{L}_a = \mathbf{L}_{a-1} \oslash \mathbf{1}^{m\times n}. \tag{4.23}$$

The so-called "Maxout" layer[45, 96] is closely related to pooling layers, with the only difference being that the pooling operation is done across the channels of a rank-4 tensor $\mathcal{T}_{n,c,h,w}$. For example, an L-maxout layer can be described as:

$$\mathbf{L}_a = \mathcal{T}'_{n,k,h,w} = \max_{l=1}^{L} \mathcal{T}_{n,kl,h,w} = \mathbf{L}_{a-1}. \tag{4.24}$$

It can be noted that pooling layers and maxout layers don't have parameters. This property can be utilized to achieve adaptivity to change in dimensions of input data, like in Spatial Pyramid Pooling[54].

4.2.5 ▪ Time Recurrent Layer

In some scenarios, the given data is a sequence $\{\mathbf{x}_i\}$ with another sequence $\mathbf{y}_i$ being the corresponding output. Though we can stack $\{\mathbf{x}_i\}$ as rows into a matrix $\mathbf{X}$ and similarly $\{\mathbf{y}_i\}$ into $\mathbf{Y}$, and then solve the problem as with 4.18, the time-order between $\mathbf{x}_i$ is not exploited if elementwise functions are used as activation functions. In contrast, the so-called "time-recurrent layer" uses following formulation:

$$\inf_{\mathbf{M},\,\mathbf{N}} g(\mathbf{L},\,\mathbf{Y}) \text{ s.t. } \mathbf{l}_i = h_i(\mathbf{l}_{i-1}\mathbf{M} + \mathbf{x}_i\mathbf{N}),\ 1 \le i \le n. \tag{4.25}$$

A neural network with time recurrent layer is often called RNN[12] in literature.

Comparing 4.26 with 4.17, we see these differences:

1. Recurrent layer uses the same $\mathbf{M}$ and $\mathbf{N}$ across the sequence, effectively assuming the dynamics to be time-invariant.
2. Output of each layer is composed to compare with $\mathbf{Y}$, which means that the order of time is preserved.

Note 4.26 assumes one direction recurrence relationship between layers. A generalization of this formulation is the bidirectional model:

$$\inf_{\mathbf{M},\,\mathbf{N}} g(\mathbf{L},\,\mathbf{Y}) \text{ s.t. } \mathbf{l}_i^{\rightarrow} = h_i^{\rightarrow}(\mathbf{l}_{i-1}^{\rightarrow}\mathbf{M}^{\rightarrow},\ +\mathbf{x}_i\mathbf{N}^{\rightarrow}) \tag{4.26}$$

$$\mathbf{l}_i^{\leftarrow} = h_i^{\leftarrow}(\mathbf{l}_{i+1}^{\leftarrow}\mathbf{M}^{\leftarrow},\ +\mathbf{x}_i\mathbf{N}^{\leftarrow}) \tag{4.27}$$

$$\mathbf{l}_i = \mathbf{W}^{\rightarrow}\mathbf{l}_i^{\rightarrow} + \mathbf{W}^{\leftarrow}\mathbf{l}_i^{\leftarrow},\ 1 \le i \le n. \tag{4.28}$$

Time recurrent layer has application in OCR[49], speech recognition[48, 37], sequence generation[46], attention model [50] etc.

4.3 ▪ Gradient descent algorithm

Given an optimization problem as

$$\inf_{\mathbf{x}} f(\mathbf{x}), \tag{4.29}$$

we may use the following differential equation to solve it:

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = -\frac{\partial f}{\partial \mathbf{x}} \tag{4.30}$$

The rationale behind is this approach is that f always decrease with this dynamics because:

$$\frac{\mathrm{d}f(\mathbf{x})}{\mathrm{d}t} = \frac{\partial f}{\partial \mathbf{x}}\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = -(\frac{\partial f}{\partial \mathbf{x}})^2 \leq 0. \tag{4.31}$$

Based on the above observation, in general, we can use following ordinary differential equation to solve 4.29:

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = -g(\mathbf{x})\,\mathrm{sgn}(\frac{\partial f}{\partial \mathbf{x}}), \tag{4.32}$$

where $g(\mathbf{x}) \geq 0$ and $g(\mathbf{x}) \not\equiv 0$. This is because in this case we have:

$$\frac{\mathrm{d}f(\mathbf{x})}{\mathrm{d}t} = \frac{\partial f}{\partial \mathbf{x}}\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = -(\frac{\partial f}{\partial \mathbf{x}}\,\mathrm{sgn}(\frac{\partial f}{\partial \mathbf{x}}))g(\mathbf{x}) \leq 0, \tag{4.33}$$

Hence the differential equation is guaranteed to converge if $f(\mathbf{x})$ is lower bounded.

4.3.1 ▪ First Order Algorithm

By discretizing 4.30 we can derive the following algorithm with time step $\eta > 0$:

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \eta\frac{\partial f}{\partial \mathbf{x}}. \tag{4.34}$$

In neural network terminology, η is referred to as "learning rate" and the determination and dynamic changing of value of η is considered an important factor in improving the convergence rate and reducing final residue cost function value.

4.3.2 ▪ Second Order Algorithm

One may compare the update rule with 4.34 with the update rule in Newton's method:

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \mathbf{H}^{-1}\frac{\partial f}{\partial \mathbf{x}}, \tag{4.35}$$

where

$$\mathbf{H}_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}. \tag{4.36}$$

It can be seen that the first order method is a special case of the second order algorithm when $\mathbf{H}^{-1} \approx \eta\mathbf{I}$. Compared to the first order algorithm, the second order algorithm would require computation or approximation of $\mathbf{H}$, which incurs significant computation.

4.3.3 ▪ Learning Rate

Adjusting of η in 4.34, i.e. learning rate, is considered important in neural network training. Adadelta[141], ADAM[69] are example schemes for adjusting the learning rate and transforming the gradient.

4.3.4 ▪ Automatic Differentiation Framework

Automatic Differentiation aims to automatic construct the form of gradient of a function given the explicit construction of the function. If we replace every number x with the number $x + x'\varepsilon$, where x' is a real number, but ε is an abstract number with the property $\varepsilon^2 = 0$, we can encode the chain rule as below:

$$(x + x'\varepsilon) + (y + y'\varepsilon) = x + y + (x' + y')\varepsilon \tag{4.37}$$

$$(x + x'\varepsilon) \cdot (y + y'\varepsilon) = xy + xy'\varepsilon + yx'\varepsilon + x'y'\varepsilon^2 = xy + (xy' + yx')\varepsilon \tag{4.38}$$

By mapping $x + x'\varepsilon$ to a tuple $\langle x,\, x' \rangle$, we can construct more rules like below:

$$\langle u, u' \rangle + \langle v, v' \rangle = \langle u + v, u' + v' \rangle \tag{4.39}$$

$$\langle u, u' \rangle - \langle v, v' \rangle = \langle u - v, u' - v' \rangle \tag{4.40}$$

$$\langle u, u' \rangle \langle v, v' \rangle = \langle uv, u'v + uv' \rangle \tag{4.41}$$

$$\langle u, u' \rangle / \langle v, v' \rangle = \left\langle \frac{u}{v}, \frac{u'v - uv'}{v^2} \right\rangle \quad (v \neq 0) \tag{4.42}$$

$$f(\langle u, u' \rangle) = \langle f(u), u' f'(u) \rangle, \text{ where } f' \text{ is the gradient of } f \tag{4.43}$$

This automatic differentiation has been built into Open Source software like Theano[7, 8].

4.4 ▪ Cost function and Encoding of Label

The cost function is also known as objective function. It encodes the degrees of freedom allowed in comparing ground truth label with predicted label. It plays a similar role to the likelihood function in statistics.

4.4.1 ▪ Cost function

Choice of function g in 4.18 is assumed to be important in construction of neural network model. Several examples of cost functions are:

1. ℓ_2 distance is used when feature in $\mathbf{Y}$ is a continuous variable.

2. One problem related to ℓ_2 distance is that it is not robust w.r.t. outliers. In this case, we may use $f(\mathbf{x})_i = \log(\epsilon + |x_i|)$, which can suppress the influence of large discrepancies in matching.

3. Entropy-related function is used when each row of $\mathbf{Y}$ is a one-hot representation of a "class". A prominent example is cross entropy: $-\sum_i y_i \log l_i$. Note frequently $-\sum_i y_i \log l_i$ is chosen in favor of $-\sum_i l_i \log y_i$ when l_i is result of a "Softmax" layer, as in this case, $\log y_i$ is in general finite while y_i may be zero and would cause numerical instability when $-\sum_i l_i \log y_i$ is used as cost function.

4. Connectionist Temporal Classification[47] cost is used when both the prediction and the ground truth are labels but not aligned.

5. Triplet loss[114] that induces clustering.

4.4.2 ▪ Encoding of Label

The ground truth label type is related to the nature of the neural network task and may differ significantly across tasks. Below we list a few possible label types and discuss their encoding.

Classification Label

In the classification task, neural network is to predict one out of n possible classes for each input data instance. The one-hot encoding of the output class number is often used, which when given the class number k, will encode into a vector $\mathbf{v}$:

$$\mathbf{v_k} = \mathbf{1}, \text{ and } \mathbf{v_i} = \mathbf{0}, \forall \mathbf{i} \neq \mathbf{k}. \tag{4.44}$$

It can be seen this is exactly the state vector representation of the probability the output class. Due to this probability interpretation, we can use cross-entropy as distance function between the output of softmax layer of a neural network and the one-hot encoding of the ground truth label.

Real-value Label

When the output labels are real-value numbers like coordinate values, the state vector representation will no longer be discrete. In this case, simple ℓ_2 distance between the output of the neural network and the ground truth labels can be used.

4.5 ▪ Model Regularization

4.5.1 ▪ Regularization Function

In 4.4 and 4.5, when $\mathbf{X}$ is ill-conditioned, the computation of $\mathbf{M}$ is not stable. Model Regularization aims to improve the stability of computation by introducing extra cost function terms that depends on assumed regularity of $\mathbf{M}$. Given optimization problem 4.29, examples of regularization terms are:

1. ℓ_2 regularization: $\inf_{\mathbf{x}} f(\mathbf{M}, \mathbf{x}) + \gamma \|\mathbf{M}\|_F^2$. This regularization penalizes the presence of elements of large magnitude in $\mathbf{M}$, as ℓ_2 norm is often dominated by the larger elements.

2. ℓ_1 regularization: $\inf_{\mathbf{x}} f(\mathbf{M}, \mathbf{x}) + \gamma \|\mathbf{M}\|_1$. Compare to the ℓ_2 norm which has a diminishing gradient for values closer to zero, ℓ_1 has a constant gradient towards zero, and can induce more zeros in $\mathbf{M}$. As zero is the singular point for ℓ_1 norm, the following more numerically stable version may be used:

$$\|\mathbf{x}\|_{1,\varepsilon} = \sum_i \sqrt{\varepsilon + x_i^2}. \tag{4.45}$$

3. $f(\mathbf{M}) = \sum_{ij} \log(\epsilon + |M_{ij}|)$ can be used as an even-stronger inducer of sparsity, though it is not convex.

4. nuclear norm regularization: $\inf_{\mathbf{x}} f(\mathbf{M}, \mathbf{x}) + \gamma\|\mathbf{M}\|_*$. As nuclear norm is a convex proxy of the rank function, minimizing the nuclear norm $\|\mathbf{M}\|_*$ can indirectly reduce the rank $\text{rank}(\mathbf{M})$.

A few regularizers related to images are listed below. It can be seen that there are virtually no limits on which functions can be used as cost function terms so that regularization can be used to model the distribution of $\mathbf{M}$. We can construct more regularization functions within the GOO framework introduced in Chapter 6.

1D Total Variation Regularizer

For data represented as a vector $\mathbf{y} \in \mathbb{F}^m$, the total variation is defined as:

$$\text{TV}(\mathbf{y}) = \sum_{i=1}^{m-1} |y_i - y_{i-1}|. \tag{4.46}$$

2D Total Variation Regularizer

For data represented as a vector $\mathbf{M} \in \mathbb{F}^{m\times n}$, the isotropic [110] total variation is defined as:

$$\text{TV}_{ISO}(\mathbf{y}) = \sum_{i,j} \sqrt{|y_{i+1,j} - y_{i,j}|^2 + |y_{i,j+1} - y_{i,j}|^2}. \tag{4.47}$$

In contrast, the anisotropic total variation is defined as:

$$\text{TV}_{aniso}(y) = \sum_{i,j} \sqrt{|y_{i+1,j} - y_{i,j}|^2} + \sqrt{|y_{i,j+1} - y_{i,j}|^2} = \sum_{i,j} |y_{i+1,j} - y_{i,j}| + |y_{i,j+1} - y_{i,j}|. \tag{4.48}$$

Connectivity Regularizer

When the data is a boolean matrix $\mathbf{M}$ and one would like to penalize against the number of connected components in the matrix, one could define the nonlinear connectivity regularizer as:

$$R(\mathbf{M}) = \sum_{i,j} R_{i,j} = \sum_{i,j} \min_{(i',j')\in\mathcal{N}(i,j),(i',j')\neq(i,j)} |M_{i,j} - M_{i',j'}|. \tag{4.49}$$

It can be noted that $R_{i,j} = 0 \iff M_{i,j}$ is equal to one of neighbors of (i, j). Hence this regularizer will penalize the case when there are too many connected components in $\mathbf{M}$.

p-Fourier Regularizer and other Unitary Regularizer

The 1D p-Fourier regularizer is defined as ℓ_p norm of the Fourier transformed weights as

$$\|\mathbf{Fx}\|_p, \tag{4.50}$$

where $\mathbf{F}$ is the Fourier matrix.

Similarly, for a 2D matrix $\mathbf{X}$, the 2D p-Fourier regularizer is defined as

$$\|\mathbf{FXF}^\top\|_p. \tag{4.51}$$

Alternatively, we can define a p-Walsh regularizer as

$$\|\mathbf{WXW}^\top\|_p, \tag{4.52}$$

where $\mathbf{W}$ is the Hadamard matrix associated with the Walsh transform.

4.6 ▪ Using Label Regularization in Neural Network

In a neural network like

$$\inf_{\theta} d(\mathbf{Y}, f(\mathbf{X};\theta)), \tag{4.53}$$

If $f(\mathbf{X};\theta)$, the prediction of the network, is known to conform to some prior distribution, it is possible to introduce an extra cost term to regularize the output as

$$\inf_{\theta} d(\mathbf{Y}, \mathbf{Z}) + \lambda r(\mathbf{Z}) \text{ s.t. } \mathbf{Z} = f(\mathbf{X};\theta), \tag{4.54}$$

Hence we can arbitrarily choose the label regularization term depending on the property of data like below:

1. For low rank labels, we can use Schatten p-metric with nuclear norm being its special case.
2. For smooth labels, we can use Lapalcian-like operator, like a graph Laplacian operator.
3. For labels that exhibit tensor structure, we can use the tensor nuclear norm as in formula 8.1.
4. For labels having few frequency components, we can use the sparse Fourier regularizer.

4.7 ▪ Other Regularization Technique

4.7.1 ▪ Regularization by Random Masks

"Dropout"[58] and "Dropconnect"[134] techniques in neural networks aim at improving the generalizability of the converged neural network.

In "Dropout", for output of some intermediate layer $\mathbf{y}$, a random 0-1 mask Ω is imposed so that $\Omega \circ \mathbf{y}$ is used as input to the next layer. When the model has converged, the parameter of that layer are multiplied by $\mathrm{E}[\Omega]$ to avoid doing random sampling at test time.

In "Dropconnect", the random mask is imposed on the weights instead of activations, i.e., for a fully-connected layer $h(\mathbf{xW})$, the "dropconnect" form would be $h(\mathbf{x}(\Omega \circ \mathbf{W}))$. In fact, "dropout" regularization can be seen as a special case of "dropconnect" for activation functions satisfying $h(0) = 0$, as in that case we have:

$$\omega \circ h(\mathbf{xW}) = h(\omega \circ (\mathbf{xW})) = h(\mathbf{xW}\,\mathrm{diagg}(\omega)). \tag{4.55}$$

For inference, "dropconnect" technique approximates distribution of Ω with a Normal Distribution with matching mean and variance. When each element of Ω is sampled from a binomial distribution $\mathcal{B}(p)$, the normal distribution with matching moments would be:

$$\mathcal{N}(p\mathbf{xW}, p(1-p)(\mathbf{x} \circ \mathbf{x})(\mathbf{W} \circ \mathbf{W})). \tag{4.56}$$

4.7.2 ▪ Stochastic Pooling

Stochastic Pooling[142] introduces underterminstic behavor to the network, in the hope that the converged neural network will be more robust to noise and have better generalizability.

4.8 ▪ Neural Network Transformations

There exist several functionals that transform a neural network to another one. Below we will give some examples.

4.8.1 ▪ Transforming a Fully-connected Layer to a Convolutional Layer

We reproduce the transfer function of a fully-connected layer for each data instance as below:

$$\mathbf{y} = h(\mathbf{x} \cdot \mathbf{w} + b). \tag{4.57}$$

In contrast, a convolutional layers is as:

$$\mathbf{y} = h(\mathbf{x} \star \mathbf{w} + b), \tag{4.58}$$

Noting that when we use zero-padded circular convolution as $\star$ and $\mathbf{x}$ and $\mathbf{w}$ have the same length, we have

$$(\mathbf{x} \star \mathbf{w})_n = \mathbf{x} \cdot \tilde{\mathbf{w}}, \tag{4.59}$$

where $\tilde{\mathbf{w}}_{n-i+1} = \mathbf{w}_i$, $\mathbf{w} \in \mathbb{F}^n$.

Based on this we can transform a fully-connected layer to a convolutional layer so as to allow x of arbitrary length as input while being consistent with the definition of fully-connected layer.

4.8.2 ▪ Transfer Learning by Combining Layers from different Neural Networks

Due to the layer structure of the neural network, it is possible to take the output of some intermediate layer and use those values as non-linearly transformed features for the input[56, 57, 36]. I.e., suppose a neural network function $f(x)$ can be factorized into $f(x) = f_2(f_1(x))$, we may use $f_1(x)$ as the transformed feature in another neural network task in an optimization below:

$$\inf_{\theta} d(\mathbf{y}, h(f_1(\mathbf{x}); \theta)). \tag{4.60}$$

4.8.3 ▪ Collapsing the Output Class Hierarchy

Given a neural network $\hat{y} = f(\mathbf{x})$, when we want to adjust the number of output classes, in general we would have to re-train a neural network, probably with the Transfer Learning technique outlined above. However, in case the new output classes are subsets of the original classes, we can add a non-linear layer that non-linear projects the outputs of $f(\mathbf{x})$ to the new output space. I.e., if the original output is $\hat{y} \in \mathbb{F}^n$, and the desired output is $\hat{y}^{'} \in \mathbb{F}^{n^{'}}$, we can use the following transformation:

$$\hat{y}_i^{'} = \max_j(\hat{y}_j) \text{ s.t. } c_i \sim c_j \tag{4.61}$$

Note when it is desirable to have the output be probability-like, a normalization layer like a softmax layer can be added.

Chapter 5

Model Approximation with Kronecker Product

5.1 ▪ Abstract

In this paper we propose and study a technique to reduce the number of parameters and computation time in fully-connected layers of neural network using Kronecker product, at a mild cost of the prediction quality. The technique proceeds by replacing Fully-Connected layers with so-called Kronecker Fully-Connected layers, where the weight matrices of the FC layers are approximated by linear combinations of multiple Kronecker product of smaller matrices. In particular, given a model trained on SVHN dataset, we can construct a new KFC model with 73% reduction in total number of parameters, while the error only rises mildly. In contrast, using low-rank method based on row/column vectors can only achieve 35% reduction in total number of parameters given similar quality degradation allowance. If we only compare the KFC layer with its counterpart fully-connected layer, the reduction in number of parameters is more than 99%. The amount of computation is also reduced as we replace matrix product of a large matrix with matrix products of a few smaller matrices. Further experiments on MNIST, SVHN and some Chinese Character recognition models also demonstrate effectiveness of our technique.

5.2 ▪ Introduction

Model approximation aims at reducing the number of parameters and amount of computation of neural network models, while keeping the quality of prediction results mostly the same.[1] Model approximation is important for real world application of neural network to satisfy the time and storage constraints of the applications.

In general, given a neural network $\tilde{f}(\cdot;\tilde{\theta})$, we want to construct another neural network $f(\cdot;\theta)$ within some pre-specified resource constraint, and minimize the differences between the outputs of two functions on the possible inputs. An example setup is to directly minimize the differences between the output of the two functions:

$$\inf_{\theta} \sum_{i} d(f(x_i;\theta), \tilde{f}(x_i;\tilde{\theta})), \tag{5.1}$$

where d is some distance function and x_i runs over all input data.

The formulation 5.1 does not give any constraints between the structure of f and $\tilde{f}$, meaning that any model can be used to approximate another model. In practice, a structural similar

[1] In some circumstances, as less number of model parameters reduce the effect of overfitting, model approximation sometimes leads to more accurate predictions.

model is often used to approximate another model. In this case, model approximation may be approached in a modular fashion w.r.t. to each layer.

5.2.1 ▪ Low Rank Model Approximation

Low rank approximation in linear regression dates back to [2]. In [112, 86, 137, 147, 28], low rank approximation of fully-connected layer is used; and [66, 107, 82, 81, 27] also considered low rank approximation of convolution layer. [146] considered approximation of multiple layers with nonlinear activations.

We first outline the low rank approximation method below. The fully-connected layer widely used in neural network construction may be formulated as:

$$\mathbf{L}_a = h(\mathbf{L}_{a-1}\mathbf{M}_a + \mathbf{b}_a), \tag{5.2}$$

where $\mathbf{L}_i$ is the output of the i-th layer of the neural network, $\mathbf{M}_a$ is often referred to as "weight term" and $\mathbf{b}_a$ as "bias term" of the a-th layer.

As the coefficients of the weight term in the fully-connected layers are organized into matrices, it is possible to perform low-rank approximation of these matrices to achieve an approximation of the layer, and consequently the whole model. In particular, given Singular Value Decomposition of a matrix $\mathbf{M} = \mathbf{U}\mathbf{D}\mathbf{V}^*$, where $\mathbf{U}, \mathbf{V}$ are unitary matrices and $\mathbf{D}$ is a diagonal matrix with the diagonal made up of singular values of $\mathbf{M}$, a rank-k approximation of $\mathbf{M} \in \mathbb{F}^{m\times n}$ is:

$$\mathbf{M} \approx \mathbf{M}_k = \tilde{\mathbf{U}}\tilde{\mathbf{D}}\tilde{\mathbf{V}}^* \text{ , where } \tilde{\mathbf{U}} \in \mathbb{F}^{m\times k}, \tilde{\mathbf{D}} \in \mathbb{F}^{k\times k}, \tilde{\mathbf{V}} \in \mathbb{F}^{n\times k}, \tag{5.3}$$

where $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ are the first k-columns of the $\mathbf{U}$ and $\mathbf{V}$ respectively, and $\tilde{\mathbf{D}}$ is a diagonal matrix made up of the largest k entries of $\mathbf{D}$.

In this case approximation by SVD is optimal in the sense that the following holds [59]:

$$\mathbf{M}_r = \inf_{\mathbf{X}} \|\mathbf{X} - \mathbf{M}\|_F \text{ s.t. } \operatorname{rank}(\mathbf{X}) \leq r, \tag{5.4}$$

and

$$\mathbf{M}_r = \inf_{\mathbf{X}} \|\mathbf{X} - \mathbf{M}\|_2 \text{ s.t. } \operatorname{rank}(\mathbf{X}) \leq r. \tag{5.5}$$

The approximate fully connected layer induced by SVD is:

$$\mathbf{Z} = \mathbf{L}_{a-1}\tilde{\mathbf{U}} \tag{5.6}$$

$$\mathbf{L}_a = h(\mathbf{Z}\mathbf{D}\mathbf{V}^* + \mathbf{b}_a), \tag{5.7}$$

In the modular representation of neural network, this means that the original fully connected layer is now replaced by two consequent fully-connected layers.

However, the above post-processing approach only ensures getting an optimal approximation of $\mathbf{M}$ under the rank constraint, while there is still no guarantee that such an approximation is optimal w.r.t. the input data. I.e., the optimum of the following may well be different from the rank-r approximation $\mathbf{M}_r$ w.r.t. some given input $\mathbf{X}$:

$$\inf_{\mathbf{Z}} \|\mathbf{Z}\mathbf{X} - \mathbf{M}\mathbf{X}\|_F \text{ s.t. } \operatorname{rank}(\mathbf{Z}) \leq r. \tag{5.8}$$

Hence it is often necessary for the resulting low-rank model $\tilde{\mathcal{M}}$ to be trained for a few more epochs on the input, which is also known as the "fine-tuning" process.

Alternatively, we note that the rank constraint can be enforced by the following structural requirement for $\mathbf{X} \in \mathbb{F}^{m\times n}$:

$$\operatorname{rank}(\mathbf{X}) \leq r \Leftrightarrow \exists \mathbf{A} \in \mathbb{F}^{m\times r}, \mathbf{B} \in \mathbb{F}^{r\times n} \text{ s.t. } \mathbf{X} = \mathbf{AB}. \tag{5.9}$$

In light of this, if we want to impose a rank constraint on a fully-connected layer $L(\mathbf{M}, g)$ in a neural network where $\mathbf{M} \in \mathbb{F}^{m\times n}$, we can replace that layer with two consecutive layers $L_1(\mathbf{B}, g_1)$ and $L_2(\mathbf{A}, g_2)$, where $g_1(x) = x$, $g_2 = g$, and $\mathbf{M} = \mathbf{AB}$ where $\mathbf{A} \in \mathbb{F}^{m\times r}$, $\mathbf{B} \in \mathbb{F}^{r\times n}$, and then train the structurally constrained neural network on the training data.

As a third method, a regularization term inducing low rank matrices may be imposed on the weight matrices. In this case, the training of a k-layer model is modified to be:

$$\inf_{\theta} \sum_{i} f(x_i; \theta) + \sum_{j=1}^{k} r_j(\theta), \tag{5.10}$$

where r is the regularization term. For the weight term of the FC layers, conceptually we may use the matrix rank function as the regularization term. However, as the rank function is only well-defined for infinite-precision numbers, nuclear norm may be used as its convex proxy [66, 105].

5.3 ▪ Model Approximation by Kronecker Product

Next we propose to use Kronecker product of matrices of particular shapes for model approximation in Section 5.3.1. We also outline the relationship between the Kronecker product approximation and low-rank approximation in Section 5.3.2.

Below we measure the reduction in amount of computation by number of floating point operations. In particular, we will assume the computation complexity of two matrices of dimensions $M \times K$ and $K \times N$ to be $O(MKN)$, as many neural network implementations [7, 8, 67, 24] have not used algorithms of lower computation complexity for the typical inputs of the neural networks. Our analysis is mostly immune to the "hidden constant" problem in computation complexity analysis as the underlying computations of the transformed model may also be carried out by matrix products.

5.3.1 ▪ Weight Matrix Approximation by Kronecker Product

We next discuss how to use Kronecker product to approximate weight matrices of FC layers, leading to construction of a new kind of layer which we call Kronecker Fully-Connected layer. The idea originates from the observation that for a matrix $\mathbf{M} \in \mathbb{F}^{m\times n}$ where the dimensions are not prime [2], we have approximations like:

$$\mathbf{M} = \mathbf{M}_1 \otimes \mathbf{M}_2, \tag{5.11}$$

where $m = m_1 m_2$, $n = n_1 n_2$, $\mathbf{M}_1 \in \mathbb{F}^{m_1\times n_1}$, $\mathbf{M}_2 \in \mathbb{F}^{m_2\times n_2}$.

By property of Kronecker product, we have the following identity:

$$(\mathbf{M}_1 \otimes \mathbf{M}_2) \operatorname{vec} \mathbf{X} = \operatorname{vec}(\mathbf{M}_2 \mathbf{X} \mathbf{M}_1^\top), \tag{5.12}$$

Hence any matrix product with $\mathbf{M}_1 \otimes \mathbf{M}_2$ may be replaced with two matrix products with two smaller matrices.

[2] In case any of m and n is prime, it is possible to add some extra dummy feature or output class to make the dimensions dividable.

When KFC layers is fed with a batch of input, the above identity will become:

$$(\mathbf{M}_1 \otimes \mathbf{M}_2)[\operatorname{vec}\mathbf{X}_1, \operatorname{vec}\mathbf{X}_2, \cdots, \operatorname{vec}\mathbf{X}_n] \tag{5.13}$$
$$= [\operatorname{vec}(\mathbf{M}_2\mathbf{X}_1\mathbf{M}_1^\top), \operatorname{vec}(\mathbf{M}_2\mathbf{X}_2\mathbf{M}_1^\top), \cdots, \operatorname{vec}(\mathbf{M}_2\mathbf{X}_n\mathbf{M}_1^\top)] \tag{5.14}$$
$$= \mathcal{X} \times_1 \mathbf{M}_2^\top \times_2 \mathbf{M}_1^\top, \tag{5.15}$$

where $\times_k$ is the tensor-matrix product over mode k[72], which can be implemented as a matrix product following a linear time unfolding operation of tensor.

Any factors of m and n may be selected as m_1 and n_1 in the above formulation. However, in a Convolutional Neural Network, the input to a FC layer may be a tensor of order 4, which has some natural shape constraints that we will try to leverage in 5.3.1. Otherwise, when the input is a matrix, we do not have natural choices of m_1 and n_1. We will explore heuristics to pick m_1 and n_1 in 5.3.1. Nevertheless, we can use multiple components with different (m_1, n_1) to remedy the arbitrariness of the selection.

Kronecker product approximation for fully-connected layer with 4D tensor input

In a convolutional layer processing images, the input data $\mathbf{L}_{a-1}$ may be a tensor of order 4 as $\mathcal{T}_{nchw}$ where $n = 1, 2, \cdots, N$ runs over N different instances of data, $c = 1, 2, \cdots, C$ runs over C channels of the given images, $h = 1, 2, \cdots, H$ runs over H rows of the images, and $w = 1, 2, \cdots, W$ runs over W columns of the images. $\mathcal{T}$ is often reshaped into a matrix before being fed into a fully connected layer as $\mathbf{D}_{nj}$, where $n = 1, 2, \cdots, N$ runs over the N different instances of data and $j = 1, 2, \cdots, CHW$ runs over the combined dimension of channel, height, and width of images. The weights of the fully-connected layer would then be a matrix $\mathbf{M}_{jk}$ where $j = 1, 2, \cdots, CHW$ and $k = 1, 2, \cdots, K$ runs over output number of channels. I.e., the layer may be written as:

$$\mathbf{D} = \operatorname{Reshape}(\mathbf{L}_{a-1}) \tag{5.16}$$
$$\mathbf{L}_a = h(\mathbf{D}\mathbf{M} + \mathbf{b}). \tag{5.17}$$

Though the reshaping transformation from $\mathcal{T}$ to $\mathbf{D}$ does not incur any loss in pixel values of data, we note that the dimension information of the tensor of order 4 is lost in the matrix representation. As a consequence, $\mathbf{M}$ has $CHWK$ number of parameters.

Due to the shape of $\mathbf{M}$, we may propose a few kinds of structural constraint on $\mathbf{M}$ by requiring $\mathbf{M}$ to be Kronecker product of matrices of particular shapes.

Formulation I

In this formulation, we require $\mathbf{M} = \mathbf{M}_1 \otimes \mathbf{M}_2 \otimes \mathbf{M}_3$, where $\mathbf{M}_1 \in \mathbb{F}^{C\times K_1}, \mathbf{M}_2 \in \mathbb{F}^{H\times K_2}, \mathbf{M}_3 \in \mathbb{F}^{W\times K_3}$, and $K = K_1K_2K_3$. The number of parameters is reduced to $CK_1 + HK_2 + WK_3$. The underlying assumption for this model is that the transformation is invariant across rows and columns of the images.

Formulation II

In this formulation, we require $\mathbf{M} = \mathbf{M}_1 \otimes \mathbf{M}_2$, where $\mathbf{M}_1 \in \mathbb{F}^{C\times K_1}, \mathbf{M}_2 \in \mathbb{F}^{HW\times K_2}$, and $K = K_1K_2$. The number of parameters is reduced to $CK_1 + HWK_2$. The underlying assumption for this model is that the channel transformation should be decoupled from the spatial transformation.

Formulation III

In this formulation, we require $\mathbf{M} = \mathbf{M}_1 \otimes \mathbf{M}_2$, where $\mathbf{M}_1 \in \mathbb{F}^{CH \times K_1}, \mathbf{M}_2 \in \mathbb{F}^{W \times K_2}$, and $K = K_1 K_2$. The number of parameters is reduced to $CHK_1 + WK_2$. The underlying assumption for this model is that the transformation w.r.t. columns may be decoupled.

Formulation IV

In this formulation, we require $\mathbf{M} = \mathbf{M}_1 \otimes \mathbf{M}_2$, where $\mathbf{M}_1 \in \mathbb{F}^{CW \times K_1}, \mathbf{M}_2 \in \mathbb{F}^{H \times K_2}$, and $K = K_1 K_2$. The number of parameters is reduced to $CWK_1 + HK_2$. The underlying assumption for this model is that the transformation w.r.t. rows may be decoupled.

Combined Formulations

Note that the above four formulations may be linearly combined to produce more possible kinds of formulations. It would be a design choice with respect to trade off between the number of parameters, amount of computation and the particular formulation to select.

Kronecker product approximation for matrix input

For fully-connected layer whose input are matrices, there does not exist natural dimensions to adopt for the shape of smaller weight matrices in KFC. Through experiments, we find it possible to arbitrarily pick a decomposition of input matrix dimensions to enforce the Kronecker product structural constraint. We will refer to this formulation as KFCM.

Concretely, when input to a fully-connected layer is $\mathbf{X} \in \mathbb{F}^{N \times C}$ and the weight matrix of the layer is $\mathbf{W} \in \mathbb{F}^{C \times K}$, we can construct approximation of $\mathbf{W}$ as:

$$\tilde{\mathbf{W}} = \mathbf{W}_1 \otimes \mathbf{W}_2 \approx \mathbf{W}, \tag{5.18}$$

where $C = C_1 C_2$, $K = K_1 K_2$, $\mathbf{W}_1 \in \mathbb{F}^{C_1 \times K_1}$ and $\mathbf{W}_2 \in \mathbb{F}^{C_2 \times K_2}$.

The computation complexity will be reduced from $O(NCK)$ to $O(NCK(\frac{1}{K_2} + \frac{1}{C_1})) = O(NC_2C_1K_1 + NC_2K_1K_2)$, while the number of parameters will be reduced from CK to $C_1K_1 + C_2K_2$.

Through experiments, we have found it sensible to pick $C_1 \approx \sqrt{C}$ and $K_1 \approx \sqrt{K}$.

As the choice of C_1 and K_1 above is arbitrary, we may use linear combination of Kronecker products if matrices of different shapes for approximation.

$$\tilde{\mathbf{W}} = \sum_{j=1}^{J} \mathbf{W}_{1j} \otimes \mathbf{W}_{2j} \approx \mathbf{W}, \tag{5.19}$$

where $\mathbf{W}_{1j} \in \mathbb{F}^{C_{1j} \times K_{1j}}$ and $\mathbf{W}_{2j} \in \mathbb{F}^{C_{2j} \times K_{2j}}$.

5.3.2 ▪ Relationship between Kronecker Product Constraint and Low Rank Constraint

It turns out that factorization by Kronecker product is closely related to the low rank approximation method. In fact, approximating a matrix $\mathbf{M}$ with Kronecker product $\mathbf{M}_1 \otimes \mathbf{M}_2$ of two matrices may be casted into a Nearest Kronecker product Problem:

$$\inf_{\mathbf{M}_1, \mathbf{M}_2} \|\mathbf{M} - \mathbf{M}_1 \otimes \mathbf{M}_2\|_F. \tag{5.20}$$

An equivalence relation in the above problem is given in [130, 129] as:

$$\arg \inf_{\mathbf{M}_1,\mathbf{M}_2} \|\mathbf{M} - \mathbf{M}_1 \otimes \mathbf{M}_2\|_F = \arg \inf_{\mathbf{M}_1,\mathbf{M}_2} \|\mathcal{R}(\mathbf{M}) - \operatorname{vec} \mathbf{M}_1 (\operatorname{vec} \mathbf{M}_2)^\top\|_F, \tag{5.21}$$

where $\mathcal{R}(\mathbf{M})$ is a matrix formed by a fixed reordering of entries $\mathbf{M}$.

Note the right-hand side of formula 5.21 is a rank-1 approximation of matrix $\mathcal{R}(\mathbf{M})$, hence has a closed form solution. However, the above approximation is only optimal w.r.t. the parameters of the weight matrices, but not w.r.t. the prediction quality over input data.

Similarly, though there are iterative algorithms for rank-1 approximation of tensor [40, 79], the optimality of the approximation is lost once input data distribution is taken into consideration.

Hence in practice, we only use the Kronecker Product constraint to construct KFC layers and optimize the values of the weights through the training process on the input data.

5.3.3 ▪ Extension to Sum of Kronecker Product

Just as low-rank approximation may be extended beyond rank-1 to arbitrary number of ranks, one could extend the Kronecker Product approximation to Sum of Kronecker Product approximation. Concretely, one not the following decomposition of $\mathbf{M}$:

$$\mathbf{M} = \sum_{i=1}^{\operatorname{rank}(\mathcal{R}(\mathbf{M}))} \mathbf{A}_i \otimes \mathbf{B}_i. \tag{5.22}$$

Hence it is possible to find k-approximations:

$$\mathbf{M} \approx \sum_{i=1}^{k} \mathbf{A}_i \otimes \mathbf{B}_i. \tag{5.23}$$

We can then generalize Formulation I-IV in 5.3.1 to the case of sum of Kronecker Product.

We may further combine the multiple shape formulation of 5.19 to get the general form of KFC layer:

$$\mathbf{M} \approx \sum_{j=1}^{J} \sum_{i=1}^{k} \mathbf{A}_{ij} \otimes \mathbf{B}_{ij}. \tag{5.24}$$

where $\mathbf{A}_{ij} \in \mathbb{F}^{C_{1j} \times K_{1j}}$ and $\mathbf{B}_{ij} \in \mathbb{F}^{C_{2j} \times K_{2j}}$.

5.4 ▪ Empirical Evaluation of Kronecker product method

We next empirically study the properties and efficacy of the Kronecker product method and compare it with some other common low rank model approximation methods.

To make a fair comparison, for each dataset, we train a covolutional neural network with a fully-connected layer as a baseline. Then we replace the fully-connected layer with different layers according to different methods and train the new network until quality metrics stabilizes. We then compare KFC method with low-rank method and the baseline model in terms of number of parameters and prediction quality. We do the experiments based on implementation of KFC layers in Theano[8, 7] framework.

As the running time may depend on particular implementation details of the KFC and the Theano work, we do not report running time below. However, there is no noticeable slow down in our experiments and the complexity analysis suggests that there should be significant reduction in amount of computation.

5.4.1 ▪ MNIST

The MNIST dataset[84] consists of 28×28 grey scale images of handwritten digits. There are 60000 training images and 10000 test images. We select the last 10000 training images as validation set.

Our baseline model has 8 layers and the first 6 layers consist of four convolutional layers and two pooling layers. The 7th layer is the fully-connected layer and the 8th is the softmax output. The input of the fully-connected layer is of size $32 \times 3 \times 3$, where 32 is the number of channel and 3 is the side length of image patches(the mini-batch size is omitted). The output of the fully-connected layer is of size 256, so the baseline's weight matrix is of size 288×256.

CNN training is done with Adam[69] with weight decay of 0.0001. Dropout[58] of 0.5 is used on the fully-connected layer and the KFC layer. $y = |\tanh x|$ is used as activation function. Initial learning rate is $1e-4$ for Adam.

Test results are listed in Table 5.1. The number of layer parameters means the number of parameters of the fully-connected layer or its counterpart layer(s). The number of model parameters is the number of the parameters of the whole model. The test error is the min-validation model's test error and is averaged by 5 models.

In the Cut-N methods, we use N output neurons instead of 256 in fully-connected layer. In the SVD-N methods[137], we apply singular value decomposition on baseline's weight matrix, reconstruct it to rank N and fine-tune the restructured model. In the LowRank-N methods[112], we replace the fully-connected layer with two fully-connected layer where the first fully-connected layer output size is N and the second fully-connected layer output size is 256. In the KFC-II method, we replace the fully-connected layer with KFC layer using formulation II with $K_1 = 64$ and $K_2 = 4$. In the KFC-III and KFC-IV method, we replace the fully-connected layer with the KFC layer using formulation III and IV with $K_1 = 128$ and $K_2 = 2$. In the KFC-Combined method, we replace the fully-connected layer with KFC layer and linear combined the formulation II, III and IV($K_1 = 64, K_2 = 4$ in formulation II, $K_1 = 128, K_2 = 2$ in formulation III and IV).

Table 5.1. *Comparison of using Low-Rank method and using KFC layers on MNIST dataset*

Methods	# of Layer Params(%Reduction)	# of Model Params(%Reduction)	Test Error
Baseline	74.0K	99.5K	0.51%
Cut-128	37.0K(50.0%)	61.2K(37.5%)	0.55%
Cut-96	27.8K(62.5%)	51.7K(48.1%)	0.58%
LowRank-96	52.6K(29.0%)	78.1K(21.5%)	0.57%
SVD-96	27.8K(62.5%)	51.7K(48.1%)	0.54%
KFC-II	2.1K(97.2%)	27.7K(72.2%)	0.84%
KFC-III	12.4K(83.2%)	37.9K(38.1%)	0.79%
KFC-IV	12.4K(83.2%)	37.9K(38.1%)	0.65%
KFC-Combined	27.0K(63.51%)	52.5K(47.2%)	0.57%

We also extend the KFC to rank k approximation. Figure 5.1 shows that extend the Kronecker product approximation to sum of kronecker product approximation can increase the model prediction quality, which give us a method to trade off between the model size and prediction

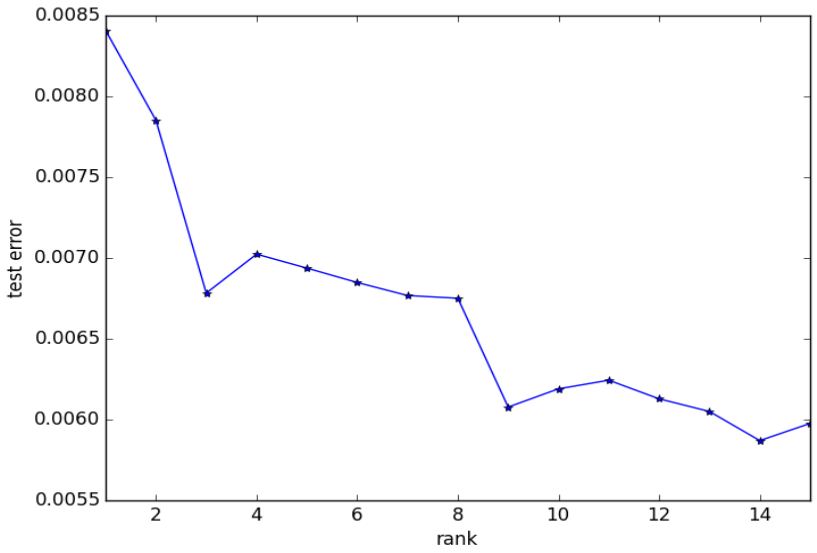


Figure 5.1. *Test error on MNIST with KFC layer using different ranks*

quality.

5.4.2 ▪ Street View House Numbers

The SVHN dataset[100] is a real-world digit recognition dataset consisting of photos of house numbers in Google Street View images. The dataset comes in two formats and we consider the second format: 32-by-32 colored images centered around a single character. There are 73257 digits for training, 26032 digits for testing, and 531131 less difficult samples which can be used as extra training data. To build a validation set, we randomly select 400 images per class from training set and 200 images per class from extra training set as [115, 45] did.

Here we use a similar but larger neural network as used in MNIST to be the baseline. The input of the fully-connected layer is of size $256 \times 5 \times 5$. The baseline's fully-connected layer has 256 output neurons. Other implementation details are not changed. Test results are listed in Table 5.3.

We can see the FKC methods can reduce the number of parameter significantly and it is impossible for SVD or other low rank method to have such a great reduction. The combined formulation or sum extension of KFC method gives more than 50% parameter reduction with negligible accuracy loss, which is better than other low rank methods.

5.4.3 ▪ Chinese Character Recognition

We also evaluate application of KFC to a Chinese character recognition model. Our experiments are done on a private dataset for the moment and may extend to other established Chinese character recognition datasets like HCL2000[143] and CASIA-HWDB[88].

For this task we also use a convolutional neural network. The distinguishing feature of the neural network is that following the convolution and pooling layers, it has two FC layers, one

Table 5.2. *Comparison of using Low-Rank method and using KFC layers on SVHN dataset*

Methods	# of Layer Params(%Reduction)	# of Model Params(%Reduction)	Test Error
Baseline	1.64M	2.20M	2.57%
Cut-128	0.82M(50.0%)	1.38M(37.3%)	2.79%
Cut-64	0.41M(25.0%)	0.97M(55.9%)	3.19%
SVD-128	0.82M(50.0%)	1.38M(37.3%)	3.67%
SVD-64	0.43M(73.7%)	0.97M(55.9%)	2.85%
LowRank-128	0.85M(48.0%)	1.42M(35.7%)	2.59%
LowRank-64	0.43M(74.0%)	0.99M(55.1%)	3.35%
KFC-II	0.41M(25.0%)	0.58M(73.7%)	3.13%
KFC-III	0.16M(90.0%)	0.72M(67.0%)	2.93%
KFC-IV	0.16M(90.0%)	0.72M(67.0%)	2.96%
KFC-Combined	0.34M(79.3%)	0.91M(58.6%)	2.60%
KFC-II-Rank10	0.17M(89.9%)	0.73M(66.9%)	2.86%

with 1536 hidden size, and the other with more than 6000 hidden size.

The two FC layers happen to be different type. The 1st FC layer accepts tensor as input and the 2nd FC layer accepts matrix as input. We apply KFC-I formulation to 1st FC and KFCM to 2nd FC.

Table 5.3. *Effect of using KFC layers on a Chinese recognition dataset*

Methods	%Reduction of 1st FC Layer Params	%Reduction of 2nd FC Layer Params	%Reduction of Total Params	Test Error
Baseline	0%	0%	0%	10.6%
KFC	99.3%	0%	36.0%	11.6%
KFC and KFCM (rank=1)	98.7%	99.9%	94.5%	21.8%
KFC and KFCM (rank=10)	93.3%	99.1%	91.8%	13.0%

It can be seen KFC can significantly reduce the number of parameters. However, in case of "KFC and KFCM (rank=1)", this also leads to serious degradation of prediction quality. However, by increasing the rank from 1 to 10, we are able to recover most of the lost prediction quality. Nevertheless, the rank-10 model is still very small compared to the baseline model.

5.5 ▪ Related Work

In this section we discuss related work not covered in previous sections. An alternative method for reducing the number of parameters and speedup the computation of a neural network is to use low precision arithmetic. It has been found that neural networks with weights being floating point numbers are to some extent not sensitive to the number of significant digits in weights. It is therefore possible to apply quantization of the floating point weights for model approximation [51, 43]. The transformed model will require significantly less storage space. Moreover, in architecture where lower precision floating point computation is possible, the quantized neural network may be directly used and exploits the architectural support for speeding up computation. We note that such techniques are orthogonal to the KFC technique and may be combined.

Another family of methods for reducing the number of parameters in a neural network has been presented in [22, 21], where random hash bins are created to enforce sharing of parameters in a convolutional neural network. This family of methods also exploits regularities in weights of fully-connected layer. However, KFC layer also enables significant reduction in amount of computation besides reduction of storage requirement.

5.6 ▪ Conclusion and Future Work

In this paper, we propose and study methods for approximating the weight matrices of fully-connected layers with sums of Kronecker product of smaller matrices, resulting in a new type of layer which we call Kronecker Fully-Connected layer. We consider both the cases when input to the fully-connected layer is a tensor of order 4 and when the input is a matrix. We have found that using the KFC layer can significantly reduce the number of parameters and amount of computation in experiments on MNIST, SVHN and Chinese character recognition.

As future work, we note that when weight parameters of a convolutional layer is a tensor of order 4 as $\mathcal{T} \in \mathbb{F}^{K \times C \times H \times W}$, it can be represented as a collection of $H \times W$ matrices $\mathbf{T}_{hw}$. We can then approximate each matrix by Kronecker products as $\mathbf{T}_{hw} = \mathbf{A}_{hw} \otimes \mathbf{B}_{hw}$ following KFCM formulation, and apply the other techniques outlined in this paper. It is also noted that the Kronecker product technique may also be applied to other neural network architectures like Recurrent Neural Network, for example approximating transition matrices with linear combination of Kronecker products.

Chapter 6

Group Orbit Optimization

6.1 ▪ Abstract

In this paper we propose and study an optimization problem over a matrix group orbit that we call *Group Orbit Optimization* (GOO). We prove that GOO can be used to induce matrix decomposition techniques such as singular value decomposition (SVD), LU decomposition, QR decomposition, Schur decomposition, Cholesky decomposition, etc. This gives rise to a unified framework for data normalization by matrix decomposition and allows us to bridge several data regularization penalties. In particular we are able to formulate nuclear norm as infimum of L1 norm over a Kronecker product unitary group orbit. Moreover, we generalize GOO for tensor decomposition. As a concrete application of GOO, we devise a new matrix decomposition method with respect to a special linear group to recover from geometric distortions like shearing, rotation and squeezing. Empirical results demonstrate the merits of our methods.

6.2 ▪ Introduction

In machine learning and data mining we often meet scenarios in which data normalization arises, for example as preprocessing or as regularization. As preprocessing, data normalization can be used to eliminate some redundant degree of freedom from a given data. A prominent example of such data normalization is principal component analysis (PCA). When the given matrix is interpreted as a point cloud with each row being coordinates of points, PCA removes the degree of freedom in translation and rotation of the point cloud with the help of singular value decomposition (SVD) on the matrix.

As another example, data normalization is also implicitly carried out by regularization penalty functions by ignoring some degrees of freedom from data. Let us take the matrix completion problem [16, 19] as an example. Given a partially observed matrix $\Omega \circ \mathbf{M}$ where Ω is an indication set of observed elements, we would like to solve the following optimization problem for recovery of original matrix $\mathbf{M}$:

$$\inf_{\mathbf{X}} \ \|\mathbf{X}\|_* \ \text{s.t.}\ \Omega \circ \mathbf{X} = \Omega \circ \mathbf{M}. \tag{6.1}$$

As the nuclear norm is invariant under unitary transform $\|\mathbf{UXV}\|_* = \|\mathbf{X}\|_*$, the choice of particular orthonormal basis is not penalized by the nuclear norm $\|\mathbf{X}\|_*$.

The specific degrees of freedom to normalize depends on data and purpose of using the data. For example, planar objects like digits, characters or iconic symbols, often look distorted in

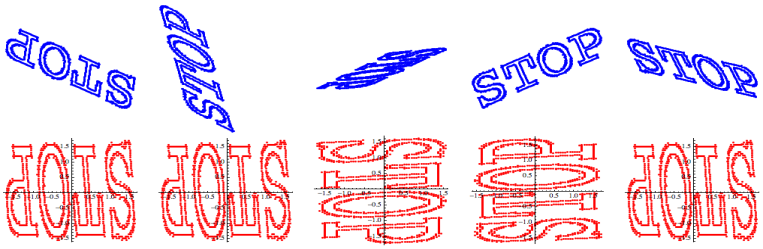


Figure 6.1. *Normalization by optimization over orbit generated by special linear group* $\mathfrak{SL}(2)$ *for 2D point clouds. The first row contains point clouds before normalization; the second row consists of corresponding point clouds after normalization for each entry in the first row. It can be observed that point clouds in the second row are approximately the same, modulo four orientations (rotated clockwise by angle of 0,* $\frac{\pi}{2}$*,* π*,* $\frac{3\pi}{2}$*).*

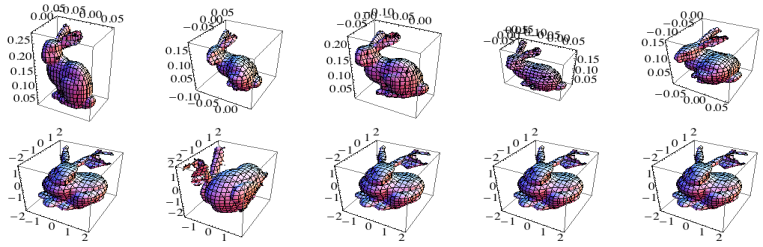


Figure 6.2. *Normalization by optimization over orbit generated by special linear group* $\mathfrak{SL}(3)$ *for 3D point clouds. The first row contains point clouds before normalization. In particular, the "rabbits" are of different shapes and sizes. The second row consists of corresponding point clouds after normalization for each entry in the first row. It can be observed that point clouds in the second row are approximately the same, modulo different orientations of the same shape.*

photos because the camera sensor plane may not be parallel to the plane carrying the objects. Therefore in this case, the degrees of freedom we would like to eliminate from data are homography transforms [53], which can be approximated as combination of translation, rotation, shearing and squeezing when the planar objects are sufficient far away relative to their size. However, PCA is not applicable to eliminate these degrees of freedom, because the normalized form found with PCA is not invariant under shearing and squeezing. In general, based on the property of data, we would need new data normalization methods that can uncover invariant structures depending on the degrees of freedom we would like to remove or ignore.

In this paper we study the case when degrees of freedom to be removed or ignored have a group structure $\mathfrak{G}$. Under such a condition, a data matrix $\mathbf{X}$ can be mapped to its quotient set $\mathbf{X}/\sim$ by the equivalence relation $\sim$ defined as

$$x_1 \sim x_2 \iff \exists g \in \mathfrak{G}, x_1 = g x_2.$$

We call the elements of quotient set $\hat{\mathbf{X}} \in \mathbf{X}/\sim$ canonical forms of data, as they are invariant with respect to (w.r.t.) group actions $g \in \mathfrak{G}$. A prior work on using the quotient set is the shape space method [33], which works in the quotient space of unitary group. In this paper we generalize over the shape space method by not limiting us to the unitary group.

Here and later, without special notes, we restrict ourselves to the case when $\mathfrak{G}$ is a matrix

group and when the group acts by simple matrix product. The quotient mapping $\mathbf{X} \to \hat{\mathbf{X}}$ can then be represented in the form of matrix decomposition:

$$\mathbf{X} = \mathbf{G}\hat{\mathbf{X}},\ \mathbf{G} \in \mathfrak{G}.$$

Rather than constructing separate algorithms for different $\mathfrak{G}$, we use an optimization process to induce corresponding matrix decomposition techniques. In particular, given a data matrix $\mathbf{X}$, we consider a group orbit optimization (GOO) problem as follows:

$$\inf_{\mathbf{G}\in\mathfrak{G}} \phi(\mathbf{G}^{-1}\mathbf{X}), \tag{6.2}$$

where $\phi : \mathbb{F}^{n_1\times n_2} \to \mathbb{R}$ is a cost function and $\mathbb{F}$ is some number field. The induced matrix decomposition of $\mathbf{X}$ can be then derived from $\hat{\mathbf{X}} = \hat{\mathbf{G}}^{-1}\mathbf{X}$. In Section 6.3 we present several special classes of cost functions, which are used to construct new formulations for several matrix decompositions including SVD, Schur, LU, Cholesky and QR (see Section 6.4). The unified framework of GOO allows us to use a template algorithm to induce these matrix decompositions (see Section 6.7). Consequently, corresponding degrees of freedom can be eliminated with the help of these matrix decompositions.

Another benefit of the GOO framework is the unification of several data regularization penalties, which we list below and will prove in Section 6.4.

- Nuclear norm as ℓ_1 norm over Kronecker product unitary group orbit:

$$\|\mathbf{X}\|_* = \inf_{\mathbf{U},\mathbf{V}\in\mathfrak{U}} \|\mathbf{U}\mathbf{X}\mathbf{V}\|_1.$$

- Schatten p-norm as ℓ_p norm over unitary group orbit: confer Corollary 6.15.
- Weighted nuclear norm: $\sum_{i=1}^n \mathbf{w}_i\sigma_i$ where σ_i are singular values of $\mathbf{X}$ and $\{\mathbf{w}_i\}_{i=1}^n$ is ascending [91]. As GOO formulation we have

$$\sum_{i=1}^n \mathbf{w}_i\sigma_i = \inf_{\mathbf{U},\mathbf{V}\in\mathfrak{U}} \|\mathbf{W}\mathbf{U}\mathbf{X}\mathbf{V}\|_1,$$

  where $\mathbf{W}$ is a diagonal matrix with $\mathbf{W}_{ii} = \mathbf{w}_i$.

- Sparse frequency domain component penalty: $\|\mathbf{F}\mathbf{X}\mathbf{F}^\top\|_1$ for $\mathbf{X} \in \mathbb{F}^{m\times n}$, where $\mathbf{F}$ is the Fourier matrix. Note as $\mathbf{F} \in \mathfrak{U}$, we have $\|\mathbf{F}\mathbf{X}\mathbf{F}^\top\|_1 \geq \|\mathbf{X}\|_*$.
- Sparse Hadamard wavelet penalty: $\|\mathbf{H}\mathbf{X}\mathbf{H}^\top\|_1$ where $\mathbf{H}$ is Hadamard wavelet coefficient matrix. Note as $\mathbf{H} \in \mathfrak{U}$, we have $\|\mathbf{H}\mathbf{X}\mathbf{H}^\top\|_1 \geq \|\mathbf{X}\|_*$.
- Hypercube inducing penalty: $\inf_{\mathbf{G}\in\mathfrak{G}} \|\mathbf{X}\mathbf{G}\|$, where $\mathfrak{G}$ depends on the degree of freedom in data. $\mathfrak{G}$ can be the unitary group or the special linear group. Confer Section 7.

As a concrete application of GOO, in Section 7 we illustrate how to use GOO to normalize point cloud data over a special linear group to recover form geometric distortions. Experiment results for two-dimensional and three-dimensional point cloud are given in Figure 6.1 and Figure 6.2. It can be observed that the effect of rotation, shearing and squeezing in data has been mostly eliminated in the normalized point clouds. The detail of this normalization is explained in Section 7.

The GOO formulation also allows us to construct generalizations of some matrix decompositions to tensor. Real world data have tensor structure when some value depends on multiple

factors. For example, in an electronic-commerce site, user preferences in different brands form a matrix. As such preferences change over time, the time-dependent preferences form a 3rd order tensor. As in the matrix case, tensor decomposition techniques [72, 74, 118] aim to eliminate degrees of freedom in data while respecting the tensor structure of data. In Section 6.5, we use GOO to induce tensor decompositions that can be used for normalizing tensor. In the unified framework of GOO, the GOO inducing tensor decomposition when applied to a 2nd order tensor, is exactly the same as the GOO inducing matrix decomposition, when the same group and cost function is used for both GOO problems.

The remainder of paper is organized as follows. Section 2 gives notation used in this paper. Section 6.3 defines several properties for describing the cost function used in defining GOO to induce matrix decompositions. Section 6.4 studies GOO formulations that can induce SVD, Schur, LU, Cholesky, QR, etc. Section 7 demonstrates how to normalize point cloud data distorted by rotation, shearing and squeezing with GOO over the special linear group. Section 6.7 presents numerical algorithms and examples of point cloud normalization decomposition. We discuss some related work in Section 6.8. Finally, we conclude the work in Section 9.

6.3 ▪ Preliminaries

In this paper we would like to show that matrix decompositions techniques can be induced from formulations of the group orbit optimization. As we have seen in formula (6.2), a GOO problem includes two key ingredients: a cost function ϕ and a group structure $\mathfrak{G}$. Thus, we present preliminaries, including *sparsifying function* and a *unit matrix group*. The sparsifying functions will be used to define cost functions for some matrix decompositions in Table 6.1 that have diagonal matrices in decomposed formulations.

6.3.1 ▪ Sparsifying functions

For two functions f and g, we here and later denote their composition as $f \circ g$ s.t. $f \circ g(x) \stackrel{\text{def}}{=\!=} f(g(x))$. We first list without proof several utility lemmas used for characterizing sparsifying functions.

Lemma 6.1 (Subadditive properties). *If $f(\sqrt{x})$ is subadditive, then*

(1) $\sum_{i=1}^{n} f(|x_i|) \geq f(\|\mathbf{x}\|)$ *where* $\mathbf{x} \in \mathbb{F}^n$.

(2) $f(0) \geq 0$.

Lemma 6.2. *If $f(e^x)$ is convex for any $x \in \mathbb{F}$, then when $\forall i, x_i \neq 0$, we have:*

$$\sum_{i=1}^{n} f(|x_i|) \geq n f\big((\prod_{i=1}^{n} |x_i|)^{\frac{1}{n}}\big) \quad .$$

Lemma 6.3. *If f is strictly concave and $f(0) \geq 0$, then $f(tx) \geq tf(x)$ where $0 \leq t \leq 1$, with equality only when $t = 0, 1$ or $x = 0$.*

Lemma 6.4. *Assume $f(x) = f(|x|)$. Then f is concave and $f(0) \geq 0$ iff f is concave and subadditive.*

Now we are ready to define the sparsifying function.

Definition 6.5 (sparsifying function). *A function f is sparsifying if*

(a) *f is symmetric about the origin; i.e., $f(x) = f(|x|)$;*

(b) *$f(\sum_i |x_i|) = \sum_i f(|x_i|) \Longrightarrow$ there is at most one i with $x_i \neq 0$.*

The following theorem gives a sufficient condition for function f to be sparsifying.

Theorem 6.6 (sufficient condition for sparsifying). *If $f(x) = f(|x|)$ and f is strictly concave and subadditive, then f is sparsifying.*

Proof. Because $f(x) = f(|x|)$, w.l.o.g. we assume $x \geq 0$. By Lemma 6.4, f is strictly concave and $f(0) \geq 0$. When $\sum_i x_i = 0$, there is no i with $x_i = 0$. Otherwise, it follows from Lemma 6.3 that

$$\sum_i f(x_i) = \sum_i f\big(\frac{x_i}{\sum_j x_j} \sum_j x_j\big) \geq \sum_i \frac{x_i}{\sum_j x_j} f(\sum_j x_j) = f(\sum_j x_j).$$

Also by Lemma 6.3, the equality holds iff $\frac{x_i}{\sum_j x_j} = 0$ or 1. Because $\sum_i \frac{x_i}{\sum_j x_j} = 1$, there is only one i with $x_i \neq 0$. In both cases, there is at most one i with $x_i \neq 0$. □

Corollary 6.7. *Conical combination of sparsifying functions. In particular, if f and g are sparsifying, then so is $\alpha f + \beta g$ where α and β are two nonnegative constants.*

Proof. As strict concavity is preserved by conical combination, we only need prove subadditivity is preserved by conical combination, which holds because:

$$\begin{aligned}(\alpha f + \beta g)(x + y) &= \alpha f(x + y) + \beta g(x + y) \\ &\leq \alpha f(x) + \alpha f(y) + \beta g(x) + \beta g(y) \\ &= (\alpha f + \beta g)(x) + (\alpha f + \beta g)(y).\end{aligned}$$

□

It can be directly checked that the following functions are sparsifying.

Example 6.8. Following functions are sparsifying:

(1) Power function: $f(x) = |x|^p$ for $0 < p < 1$;

(2) Capped power function: $f(x) = \min(|x|^p, 1)$ for $0 < p < 1$;

(3) $f(x) = -|x|^p$ for $p > 1$;

(4) $f(x) = \log(1 + |x|)$;

(5) Shannon Entropy: $f(x) = -|x| \log |x|$ when $0 \leq x$;

(6) Squared entropy: $f(x) = -x^2 \log x^2$ when $0 \leq x$;

(7) $f(x) = a - (a + |x|^p)^{\frac{1}{p}}$ for $p > 1$ and $a \geq 0$;

(8) $f(x) = -a + (a + |x|^p)^{\frac{1}{p}}$ for $p < 1$ and $a \geq 0$;

∎

Remark 6.9. *We note that* $\log|x|$ *is not subadditive because* $f(0) = -\infty < 0$. *Although* $f(x) = |x|^p$ *for* $p < 0$ *is subadditive,* $|x|^p$ *is not concave. Thus, these two functions are not sparsifying.*

Finally, in Table 6.1 we list matrix decompositions of $\mathbf{X}$ used in this paper. When referring to the Cholesky decomposition, $\mathbf{X}$ should be positive definite.

Table 6.1. *Matrix decompositions*

Name	Decomposition	Constraint
real SVD	$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$	$\mathbf{U}, \mathbf{V} \in \mathfrak{O}(n)$, $\mathbf{D}$ is diagonal
complex SVD	$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^*$	$\mathbf{U}, \mathbf{V} \in \mathfrak{U}(n)$, $\mathbf{D}$ is diagonal
QR	$\mathbf{X} = \mathbf{Q}\mathbf{D}\mathbf{R}$	$\mathbf{Q} \in \mathfrak{U}(n), \mathbf{R} \in \mathfrak{UUT}(n)$, $\mathbf{D}$ is diagonal
LU	$\mathbf{X} = \mathbf{L}\mathbf{D}\mathbf{U}$	$\mathbf{L} \in \mathfrak{LUT}(n), \mathbf{U} \in \mathfrak{UUT}(n)$, $\mathbf{D}$ is diagonal
Cholesky	$\mathbf{X} = \mathbf{L}\mathbf{D}\mathbf{L}^\top$	$\mathbf{L} \in \mathfrak{LUT}(n)$, $\mathbf{D}$ is diagonal
Schur	$\mathbf{X} = \mathbf{Q}\mathbf{U}\mathbf{Q}^*$	$\mathbf{Q} \in \mathfrak{U}(n)$, $\mathbf{U}$ is upper triangular

6.4 ▪ Group Orbit Optimization

6.4.1 ▪ Matrix Decomposition Induced from Group Orbit Optimization

GOO formulation

We now illustrate how matrix decomposition can be induced from GOO. Given two groups $\mathfrak{G}1, \mathfrak{G}2$ and a data matrix $\mathbf{M}$, we consider the following optimization problem

$$\inf_{\mathbf{G}_1 \in \mathfrak{G}_1, \mathbf{G}_2 \in \mathfrak{G}_2} \phi(\mathbf{G}_2 \mathbf{M} \mathbf{G}_1^\top). \tag{6.3}$$

Assume that $\hat{\mathbf{G}}_1$ and $\hat{\mathbf{G}}_2$ are minimizers of the above GOO and $\mathbf{D} = \hat{\mathbf{G}}_2 \mathbf{M} \hat{\mathbf{G}}_1^\top$, then we refer to

$$\mathbf{M} = \hat{\mathbf{G}}_2^{-1} \mathbf{D} \hat{\mathbf{G}}_1^{-\top},$$

as a matrix decomposition of $\mathbf{M}$ which is induced from Formula (6.3).

When $\phi = \varphi \circ \mathrm{vec}$, an equivalent formulation of Formula (6.3) is:

$$\inf_{\mathbf{G}_1 \in \mathfrak{G}_1, \mathbf{G}_2 \in \mathfrak{G}_2} \phi(\mathbf{G}_2 \mathbf{M} \mathbf{G}_1^\top) \equiv \inf_{\mathbf{G} \in \mathfrak{G}} \varphi(\mathbf{G}\,\mathrm{vec}(\mathbf{M})),$$

where $\mathbf{G} = \mathbf{G}_1 \otimes \mathbf{G}_2 \in \mathfrak{G}$ and $\mathfrak{G} \overset{\text{def}}{=\!=} \mathfrak{G}_1 \otimes \mathfrak{G}_2$.

GOO over a unit group

For a general matrix group $\mathfrak{G}$, $\mathbf{G} \in \mathfrak{G}$ implies that $|\det(\mathbf{G})| > 0$. However, group structure may not be sufficient to induce non-trivial matrix decomposition, as with some groups and cost

functions the infimum will be trivially zero. For example, with general linear group $\mathfrak{GL}$ and for any matrix $\mathbf{M}$, we have

$$\inf_{\mathbf{G}\in\mathfrak{GL}} \|\mathbf{GM}\|_p = 0,$$

because $s\mathbf{I} \in \mathfrak{GL}$ and

$$\lim_{s\to 0} \inf_{s\in\mathbb{R}} \|s\mathbf{IM}\|_p = \lim_{s\to 0} s\|\mathbf{M}\|_p = 0.$$

Nevertheless, if we require $\mathfrak{G}$ to be a unit group, we have $|\det(\mathbf{G})| = 1$. Consequently, we can prevent the infimum from vanishing trivially for any ℓ_p-norm. Thus, we mainly consider the case where $\mathfrak{G}$ is a unit group in this paper.

The following theorem shows that many matrix decompositions can be induced from the group orbit optimization.

Theorem 6.10. *SVD, LU, QR, Schur and Cholesky decompositions of matrix $\mathbf{M} \in \mathbb{F}^{m\times n}$ can be induced from GOO of the form*

$$\inf_{\mathbf{G}_1\in\mathfrak{G}_1,\mathbf{G}_2\in\mathfrak{G}_2} \phi(\mathbf{G}_2\mathbf{M}\mathbf{G}_1^\top),$$

by using the corresponding unit group $\mathbf{G}$ and cost function ϕ, which are given in Table 6.2.

Clearly, the matrix groups in Table 6.2 are unit groups. We will prove the rest of theorem in Section 6.4.2 and Section 6.4.3.

The cost function for SVD and QR can be $\phi(\mathbf{X}) = \|\mathbf{X}\|_p$, $1 \le p < 2$. And the cost function for LU, Schur and Cholesky can be $\phi(\mathbf{X}) = \sum_{ij} \|x_{ij}\mathbb{I}_{\{i<j\}}\|_p$, $1 \le p < 2$.

Table 6.2. *Matrix decompositions induced from optimizations*

Decomposition	Unit group $\mathfrak{G} = \mathfrak{G}_1 \otimes \mathfrak{G}_2$	Objective function $\phi(\mathbf{X})$
real SVD: $\mathbf{UDV}^\top$	$\{\mathbf{V}\otimes\mathbf{U} : \mathbf{U},\mathbf{V}\in\mathfrak{O}(n)\}$	$\sum_{ij} f(\lvert x_{ij}\rvert)$ where $f(\sqrt{x})$ is strictly concave, $f(0)\ge 0$
complex SVD: $\mathbf{UDV}^*$	$\{\mathbf{V}^{\mathrm{c}}\otimes\mathbf{U} : \mathbf{U},\mathbf{V}\in\mathfrak{U}(n)\}$	$\sum_{ij} f(\lvert x_{ij}\rvert)$ where $f(\sqrt{x})$ is strictly concave, $f(0)\ge 0$
QR: $\mathbf{QDR}$	$\{\mathbf{R}^\top\otimes\mathbf{Q} : \mathbf{Q}\in\mathfrak{U}(n), \mathbf{R}\in\mathfrak{UUT}(n)\}$	$\sum_{ij} f(\lvert x_{ij}\rvert)$ where $f(\sqrt{x})$ is strictly concave and increasing, $f(0)\ge 0$
LU: $\mathbf{LDU}$	$\{\mathbf{I}\otimes\mathbf{L} : \mathbf{L}\in\mathfrak{LUT}(n)\}$	$\sum_{ij} f(\lvert x_{ij}\mathbb{I}_{\{i<j\}}\rvert)$ where $f(x)\not\equiv 0$, $f(x)=0\Rightarrow x=0$, $f(x)\ge 0$
Cholesky: $\mathbf{LDL}^\top$	$\{\mathbf{I}\otimes\mathbf{L} : \mathbf{L}\in\mathfrak{LUT}(n)\}$	$\sum_{ij} f(\lvert x_{ij}\mathbb{I}_{\{i<j\}}\rvert)$ where $f(x)\not\equiv 0$, $f(x)=0\Rightarrow x=0$, $f(x)\ge 0$
Schur: $\mathbf{QUQ}^*$	$\{\mathbf{Q}^{\mathrm{c}}\otimes\mathbf{Q} : \mathbf{Q}\in\mathfrak{U}(n)\}$	$\sum_{ij} f(\lvert x_{ij}\mathbb{I}_{\{i<j\}}\rvert)$ where $f(x)\not\equiv 0$, $f(x)=0\Rightarrow x=0$, $f(x)\ge 0$

The formulation of QR decomposition exploits the fact that $\mathbf{M} = \mathbf{QR}$ is equivalent to $\mathbf{M} = \mathbf{Q}(\mathbf{D}\tilde{\mathbf{R}})$ where $\mathbf{Q}\in\mathfrak{U}(n)$, $\mathbf{R}$ is upper-triangular, $\tilde{\mathbf{R}}\in\mathfrak{UUT}(n)$, and $\mathbf{D}$ is diagonal.

However, there are matrix decompositions whose formulation cannot be expressed as GOO in the same way as Table 6.2. For example, Polar decomposition $\mathbf{M} = \mathbf{ULDL}^*$ where $\mathbf{U} \in \mathfrak{U}(n)$ and $\mathbf{L} \in \mathfrak{LUT}(n)$, though derivable from SVD, cannot be induced from a GOO formulation of diagonalization. This is because $\mathbf{L}^c \otimes \mathbf{UL}$ does not form a group as it is not closed under multiplication. For another example, consider a formulation of decomposition $\mathbf{M} = \mathbf{LDL}^{-\top}$ where $\mathbf{L} \in \mathfrak{LUT}(n)$ and $\mathbf{D}$ is diagonal. As we stated earlier, $\mathbf{L} \otimes \mathbf{L}^{-1}$ is not a group in general, so $\mathbf{S} = \mathbf{LDL}^{-\top}$ cannot be induced from a GOO formulation of diagonalization.

For matrix decomposition of the form $\mathbf{M} = \mathbf{ADB}^\top$, where $\mathbf{A} \in \mathbb{F}^{m\times r}$ and $\mathbf{B} \in \mathbb{F}^{n\times r}$ with $r \leq \min(m, n)$. In this case, we can zero-pad $\mathbf{D}$ to $\tilde{\mathbf{D}} \in \mathbb{F}^{m\times n}$, and extend $\mathbf{A}$ and $\mathbf{B}$ to $\tilde{\mathbf{A}} \in \mathbb{F}^{m\times m}$ and $\tilde{\mathbf{B}} \in \mathbb{F}^{n\times n}$ which are square matrices. Accordingly, we formulate a decomposition $\mathbf{M} = \tilde{\mathbf{A}}\tilde{\mathbf{D}}\tilde{\mathbf{B}}^\top$ which may be induced from GOO.

We next prove a lemma that characterizes the optimum.

Lemma 6.11. *If $\phi(\mathbf{GD}) \geq \phi(\mathbf{D})$ for any $\mathbf{G} \in \mathfrak{G}$ and there exists $\mathbf{A} \in \mathfrak{G}$ s.t. $\mathbf{M} = \mathbf{AD}$, then*

$$\inf_{\mathbf{G}\in\mathfrak{G}} \phi(\mathbf{GM}) = \phi(\mathbf{D}).$$

Proof. We note that $\inf_{\mathbf{G}\in\mathfrak{G}} \phi(\mathbf{GM}) = \inf_{\mathbf{G}\in\mathfrak{G}} \phi(\mathbf{GAD})$. By the group structure, the coset $\{\mathbf{GA} : \mathbf{G} \in \mathfrak{G}\} = \mathfrak{G}$. Hence we have

$$\inf_{\mathbf{G}\in\mathfrak{G}} \phi(\mathbf{GM}) = \inf_{\mathbf{G}\in\mathfrak{G}} \phi(\mathbf{GAD}) = \inf_{\mathbf{G}\in\mathfrak{G}} \phi(\mathbf{GD}).$$

Using the condition $\forall \mathbf{G} \in \mathfrak{G}, \phi(\mathbf{GD}) \geq \phi(\mathbf{D})$, we have

$$\inf_{\mathbf{G}\in\mathfrak{G}} \phi(\mathbf{GD}) \geq \inf_{\mathbf{G}\in\mathfrak{G}} \phi(\mathbf{D}) = \phi(\mathbf{D}).$$

On the other hand, as $\mathbf{I} \in \mathfrak{G}$ we have $\phi(\mathbf{D}) = \phi(\mathbf{ID}) \geq \inf_{\mathbf{G}\in\mathfrak{G}} \phi(\mathbf{GD})$. Hence

$$\phi(\mathbf{D}) = \inf_{\mathbf{G}\in\mathfrak{G}} \phi(\mathbf{GD}) = \inf_{\mathbf{G}\in\mathfrak{G}} \phi(\mathbf{GM}).$$

□

By virtue of Lemma 6.11, if we want to prove that matrix decomposition $\operatorname{vec}(\mathbf{M}) = \tilde{\mathbf{G}} \operatorname{vec}(\mathbf{D})$ is induced by a GOO w.r.t. ϕ and $\mathfrak{G}$, we only need prove that there exists a $\tilde{\mathbf{G}} \in \mathfrak{G}$ s.t. $\operatorname{vec}(\mathbf{M}) = \tilde{\mathbf{G}} \operatorname{vec}(\mathbf{D})$, and $\phi(\mathbf{G} \operatorname{vec}(\mathbf{D})) \geq \phi(\operatorname{vec}(\mathbf{D}))\ \forall \mathbf{G} \in \mathfrak{G}$. The equality condition will determine the uniqueness of the optimum of the optimization problem.

6.4.2 ▪ Matrix Diagonalization as GOO

Next we demonstrate how matrix diagonalization can be induced from GOO with proper choice of cost function and unit group.

Singular Value Decomposition

First we discuss SVD of a complex matrix and of a real matrix.

Lemma 6.12 (Cost function and group for SVD). *Let $\mathbf{D} = [d_{ij}]$ be pseudo-diagonal, and $\mathbf{U}, \mathbf{V} \in \mathfrak{U}(n)$. Given a function f such that $f(x) = f(|x|)$ and $f(\sqrt{|x|})$ is strictly concave and subadditive, and $\phi(\mathbf{X}) = \sum_{ij} f(x_{ij})$ we have*

$$\phi(\mathbf{UDV}^*) \geq \phi(\mathbf{D}),$$

with equality iff there exists a row permutation matrix $\mathbf{P}$ *such that* $|\mathbf{UDV}^*| = |\mathbf{PD}|$.

Furthermore, if $\mathbf{U}, \mathbf{V} \in \mathfrak{O}(n)$*, we have*

$$\phi(\mathbf{UDV}^\top) \geq \phi(\mathbf{D}), \tag{6.4}$$

with equality iff there exists a row permutation matrix $\mathbf{P}$ *such that* $|\mathbf{UDV}^\top| = |\mathbf{PD}|$.

Proof. First we prove the inequality. We write $g(x) = f(\sqrt{x})$ and $\mathbf{A} = \mathbf{UDV}^*$. We let $g(\mathbf{X}) = [g(x_{ij})]$ be a matrix-valued function of $\mathbf{X} = [x_{ij}]$. As g is concave and subadditive, by Lemma 6.1 for a vector $\mathbf{v} = (\mathbf{v_1}, \ldots, \mathbf{v_n})^\top$, we have $\sum_{i=1}^n f(v_i) \geq f(\|\|\mathbf{v}) = \mathbf{g}(\mathbf{v}^*\mathbf{v})$. Applying this to each column of $\mathbf{A}$, we have

$$\phi(\mathbf{A}) \geq \mathrm{tr}[g(\mathbf{A}^*\mathbf{A})] = \mathrm{tr}[g(\mathbf{VD}^*\mathbf{DV}^*)]. \tag{6.5}$$

Alternatively, we can also apply the inequality to each row of $\mathbf{A}$ and have

$$\phi(\mathbf{A}) \geq \mathrm{tr}[g(\mathbf{AA}^*)] = \mathrm{tr}[g(\mathbf{UDD}^*\mathbf{U}^*)]. \tag{6.6}$$

As $\mathbf{D}$ is pseudo-diagonal, $\mathbf{D}^*\mathbf{D}$ is diagonal. Because g is concave and $\mathbf{VV}^* = \mathbf{I}$, we can apply Jensen's inequality, obtaining

$$\mathrm{tr}(g(\mathbf{VD}^*\mathbf{DV}^*)) \geq \mathrm{tr}(\mathbf{V}(g(\mathbf{D}^*\mathbf{D}))\mathbf{V}^*).$$

Hence altogether we have:

$$\phi(\mathbf{A}) \geq \mathrm{tr}(g(\mathbf{VD}^*\mathbf{DV}^*)) \geq \mathrm{tr}(\mathbf{V}(g(\mathbf{D}^*\mathbf{D}))\mathbf{V}^*) = \mathrm{tr}(g(\mathbf{D}^*\mathbf{D})) = \sum_{ij} f(d_{ij}) = \phi(\mathbf{D}).$$

Next we check the equality condition. By Theorem 6.6, g is sparsifying. For the equality condition in inequality (6.5) to hold, $\mathbf{A}$ can have at most one nonzero in each column. By the symmetry between (6.5) and (6.6), and noting $\phi(\mathbf{A}^\top) = \phi(\mathbf{A})$ and $\phi(\mathbf{D}) = \phi(\mathbf{D}^\top)$, $\mathbf{A}$ can also have at most one nonzero in each row for $\phi(\mathbf{A}) = \phi(\mathbf{D})$ to hold. Hence when the equality holds, $\mathbf{A}$ is pseudo-diagonal. Then there exists a permutation matrix $\mathbf{P}$ such that $\mathbf{Z} = \mathbf{P}^{-1}|\mathbf{A}| = \mathbf{P}^{-1}\mathbf{QA}$ is a diagonal matrix with elements on diagonal in descending order and are all non-negative, where $\mathbf{Q}$ is a diagonal matrix s.t. $|\mathbf{Q}| = \mathbf{I}$. By the uniqueness of singular values of a matrix, we have $\mathbf{Z} = |\mathbf{D}|$. Hence equality in the inequality (6.4) holds when $|\mathbf{A}| = \mathbf{P}|\mathbf{D}| = |\mathbf{PD}|$.

The proof for $\mathbf{U}, \mathbf{V} \in \mathfrak{O}(n)$ is similar. □

Note that $\mathbf{D}$, modulo sign and permutation, is the global minimizer for a large class of functions f.

After applying Lemma 6.11, we have the following theorem.

Theorem 6.13 (SVD induced from optimization). *We are given a function* f *such that* $f(x) = f(|x|)$ *and* $f(\sqrt{|x|})$ *is strictly concave and subadditive, and* $\phi(\mathbf{X}) = \sum_{ij} f(x_{ij})$*. Let* $\hat{\mathbf{U}}$ *and* $\hat{\mathbf{V}}$ *be an optimal solution of the following optimization:*

$$\inf_{\mathbf{U}\in\mathfrak{U}(n),\mathbf{V}\in\mathfrak{U}(n)} \phi(\mathbf{U}^*\mathbf{MV}).$$

Then if SVD of $\mathbf{M}$ *is* $\mathbf{M} = \mathbf{USV}^*$*, there exist a permutation matrix* $\mathbf{P}$ *and a diagonal matrix* $\mathbf{Z}$ *such that* $\mathbf{M} = \hat{\mathbf{U}}\mathbf{PZ}\hat{\mathbf{V}}^*$ *and* $|\mathbf{Z}| = \mathbf{S}$.

Corollary 6.14. *With ϕ as in Theorem 6.13, eignedecomposition of a Hermitian matrix $\mathbf{M}$ can be induced from*

$$\inf_{\mathbf{U}\in\mathfrak{U}(n)} \phi(\mathbf{U}^*\mathbf{M}\mathbf{U}).$$

Similarly, eignedecomposition of a real symmetric matrix $\mathbf{M}$ can be induced from

$$\inf_{\mathbf{U}\in\mathfrak{O}(n)} \phi(\mathbf{U}^\top\mathbf{M}\mathbf{U}).$$

From the above optimization, we can derive several inequalities.

Corollary 6.15 (The Schatten p-norm and ℓ_p-norm inequality). *The ℓ_p-norm of matrix $\mathbf{A}$ is larger (smaller) than the Schatten p-norm of $\mathbf{A}$ when $0 \le p < 2$ (> 2).*

In particular, we have

$$\|\mathbf{M}\|_p \ge \inf_{\mathbf{U},\mathbf{V}\in\mathfrak{U}} \|\mathbf{U}\mathbf{M}\mathbf{V}^*\|_p = \|\mathbf{M}\|_{*p} \quad \textit{when } 0 < p < 2,$$

and

$$\|\mathbf{M}\|_p \le \sup_{\mathbf{U},\mathbf{V}\in\mathfrak{U}} \|\mathbf{U}\mathbf{M}\mathbf{V}^*\|_p = \|\mathbf{M}\|_{*p} \quad \textit{when } p > 2.$$

Proof. $f(x) = |x|^p$ satisfies $f(0) \ge 0$, and $f(\sqrt{|x|})$ is strictly concave when $0 \le p < 2$. On the other hand, $f(x) = -|x|^p$ satisfies $f(0) \ge 0$ and $f(\sqrt{|x|})$ is strictly concave when $p > 2$. By Theorem 6.13 we have

$$\inf_{\mathbf{U},\mathbf{V}\in\mathfrak{U}} \|\mathbf{U}\mathbf{M}\mathbf{V}^*\|_p = \|\mathbf{M}\|_{*p} \quad \text{when } 0 < p < 2,$$

and

$$\inf_{\mathbf{U},\mathbf{V}\in\mathfrak{U}} -\|\mathbf{U}\mathbf{M}\mathbf{V}^*\|_p = -\|\mathbf{M}\|_{*p} \quad \text{when } p > 2.$$

□

Corollary 6.16 (Duality gap). *Given SVD of $\mathbf{M}$ as $\mathbf{M} = \mathbf{U}\mathbf{D}\mathbf{V}^*$, we have*

$$\|\mathbf{M}\|_p \ge \|\mathbf{D}\|_p \ge \|\mathbf{D}\|_q \ge \|\mathbf{M}\|_q,$$

where $0 < p < 2 < q$.

Proof. Due to the non-increasing property of the ℓ_p-norm w.r.t. p, we have $\|\mathbf{D}\|_p \ge \|\mathbf{D}\|_q$. Also by Theorem 6.13 we have $\|\mathbf{D}\|_p = \|\mathbf{M}\|_{*p}$ and $\|\mathbf{D}\|_q = \|\mathbf{M}\|_{*q}$. Applying Corollary 6.15 completes the proof. □

Corollary 6.17. *The von Neumann entropy of density matrix $\mathbf{M}$ is smaller than the sum of the Shannon entropies of rows (columns) of $\mathbf{M}$.*

Proof. By noting that $f(x) = -x\log x$ is strictly concave and $f(0) \ge 0$ when $x \ge 0$, and that the von Neumann entropy is entropy of diagonal matrix in SVD of $\mathbf{M}$, the inequality holds. □

QR Decomposition

To derive GOO for QR decomposition, we first note that QR decomposition of a matrix $\mathbf{M}$ can be rewritten as $\mathbf{M} = \mathbf{QDR}$, where $\mathbf{Q} \in \mathfrak{U}(n)$, $\mathbf{D}$ is diagonal, and $\mathbf{R} \in \mathfrak{UUT}(m)$.

Lemma 6.18 (Cost function and group for QR). *Let f satisfy that $f(x) = f(|x|)$, $f(\sqrt{x})$ is concave and increasing; $f(0) \geq 0$. Let $\phi(\mathbf{X}) = \sum_{ij} f(x_{ij})$. If $\mathbf{Q} \in \mathfrak{U}(n)$, $\mathbf{R} \in \mathfrak{UUT}(n)$, and $\mathbf{D}$ is diagonal, then we have*

$$\phi(\mathbf{QDR}) \geq \phi(\mathbf{D}),$$

with equality when $|\mathbf{QDR}| = |\mathbf{PD}|$, where $\mathbf{P}$ is a row permutation matrix.

Proof. Let $g(x) = f(\sqrt{|x|})$ and $\mathbf{A} = \mathbf{QDR}$, and let $g(\mathbf{X}) = [g(x_{ij})]$. First we prove the inequality. As $g(x)$ is sparsifying and increasing, we have

$$\phi(\mathbf{A}) \geq \operatorname{tr}[g(\mathbf{A}^*\mathbf{A})] = \operatorname{tr}[g(\mathbf{R}^*\mathbf{D}^*\mathbf{DR})] \geq \operatorname{tr}[g(\mathbf{D}^*\mathbf{D})] = \phi(\mathbf{D}).$$

Next we check the equality condition. For the equality to hold in inequality $\operatorname{tr}[g(\mathbf{R}^*\mathbf{D}^*\mathbf{DR})] \geq \operatorname{tr}[g(\mathbf{D}^*\mathbf{D})]$, as g is increasing, $\mathbf{R}$ needs to be diagonal. Hence, $\mathbf{R} = \mathbf{I}$. Now for the equality to hold in $\phi(\mathbf{A}) \geq \operatorname{tr}[g(\mathbf{A}^*\mathbf{A})]$, $\mathbf{A}$ needs to be pseudo-diagonal. Thus, the equality holds only when $|\mathbf{A}| = \mathbf{P}|\mathbf{D}| = |\mathbf{PD}|$ where $\mathbf{P}$ is a row permutation matrix. □

Similarly, we derive the optimization inducing QR decomposition.

Theorem 6.19 (QR induced from optimization). *Assume that the conditions are satisfied in Lemma 6.18. Let $(\hat{\mathbf{Q}}, \hat{\mathbf{R}})$ be optimal solution of optimization as follows*

$$\inf_{\mathbf{Q} \in \mathfrak{U}(n), \mathbf{R} \in \mathfrak{UUT}(n)} \phi(\mathbf{Q}^{-1}\mathbf{MR}^{-1}).$$

Then $\mathbf{M} = \hat{\mathbf{Q}}\mathbf{Z}$ is the QR decomposition of $\mathbf{M}$, where $\mathbf{Z} = \hat{\mathbf{Q}}^{-1}\mathbf{M}$.

6.4.3 ▪ Matrix Triangularization as GOO

Next we demonstrate how matrix triangularization can be induced from GOO with proper choice of cost function and unit group. In fact, we can prove that any triangularization can be induced from optimization w.r.t. a masked norm.

Lemma 6.20. *A matrix decomposition $\mathbf{M} = \mathbf{G}_2\mathbf{U}\mathbf{G}_1^\top, \mathbf{G}_1 \in \mathfrak{G}_1, \mathbf{G}_2 \in \mathfrak{G}_2$, where $\mathbf{U}$ is upper triangular and $\mathfrak{G}_1 \otimes \mathfrak{G}_2$ is a unit group, can be induced from the following optimization:*

$$\inf_{\mathbf{G}_1 \in \mathfrak{G}_1, \mathbf{G}_2 \in \mathfrak{G}_2} \phi(\mathbf{G}_2\mathbf{M}\mathbf{G}_1^\top). \tag{6.7}$$

Here $\phi(\mathbf{X}) = \sum_{ij} f(|x_{ij}\mathbb{I}_{\{i<j\}}|)$ where $f(x) = 0 \Rightarrow x = 0$, $f(x) \not\equiv 0$, $f(x) \geq 0$.

Proof. We trivially have

$$\forall \mathbf{G},\ \phi(\mathbf{G}_2\mathbf{U}\mathbf{G}_1^\top) \geq 0 = \phi(\mathbf{U}).$$

By Lemma 6.11 and $\phi(\mathbf{X}) = 0 \iff \mathbf{X} = \mathbf{0}$, the decomposition can be induced from optimization. □

For example, the Schur decomposition can be induced from Formula (6.7) with $f(x) = |x|$. We will give a numerical example in Section 6.7.

6.4.4 ▪ Algorithm

We reproduce the optimization inducing decomposition for matrix as below:

$$\inf_{\mathbf{A},\mathbf{B}} \phi(\mathbf{A}^{-1}\mathbf{D}\mathbf{B}^{-\top}) \text{ s.t. } \mathbf{B}\otimes\mathbf{A}\in\mathfrak{G},$$

where $\mathfrak{G}$ is a unit group. Now when $\mathbf{B}\otimes\mathbf{A}$ is a Lie group, we can turn the above optimization to an unconstrained optimization:

$$\inf_{\mathbf{X}} \phi(\exp(\mathbf{X})\operatorname{vec}(\mathbf{D})),$$

which can be solved by any unconstrained optimization method. In particular we can perform heuristic search on $\mathbf{X}$. Alternatively, when f is smooth, we can compute gradient of $\sum f(\exp(\mathbf{X})\operatorname{vec}(\mathbf{D}))$ and use gradient flow algorithm. In fact, there is a class of algorithm on doing gradient descent on the Riemann manifold $\exp(\mathbf{X})$. The virtue of this approach is that we can start with a feasible solution and keep all intermediate solutions feasible.

On the other hand, we can penalize the constraint, and using a specific cost function to get:

$$\inf_{\mathbf{X},\mathbf{A},\mathbf{B}} \|\mathbf{X}\|_p + \frac{\mu}{2}\|\mathbf{D}-\mathbf{A}\mathbf{X}\mathbf{B}^{\top}\|_F^2, \quad 1\le p<2,$$

Then as $\mu\to\infty$, the optimization will find the infimum of original optimization. Note as $1\le p<2$ is convex, convergence of the algorithm is guaranteed as the optimization is sequential convex.

When $\mathbf{A}$ and $\mathbf{B}$ are unitary, there exist closed form update for $\mathbf{A}$ and $\mathbf{B}$ known as solution to orthogonal Procrustes problem. In fact, Sparse Unitary Decomposition is special in several aspect: 1. Frobenius norm is constant on the Lie orbit. 2. optimum w.r.t. ℓ_p-norm is Schatten p-norm. In contrast, optimum w.r.t. the ℓ_p-norm on other Lie orbit is not norm, as triangle inequality is not satisfied. 3. p-norm has dual norm. In contrast, $f(\mathbf{x})=-\|\mathbf{x}\|_q$ cannot be used as cost function for other canonical GOO when $q>2$.

6.5 ▪ Group Orbit Optimization on Tensor Data

The GOO problem on tensor $\mathcal{T}$ is defined as

$$\inf_{\mathbf{G}_i\in\mathfrak{G}_i} \phi(\prod_i \mathcal{T}\mathbf{G}_i).$$

When there exists function φ s.t. $\phi(\mathcal{T})=\varphi(\operatorname{vec}(\mathcal{T}))$, we get a form that bears resemblance to the matrix version:

$$\inf_{\mathbf{G}_i\in\mathfrak{G}_i} \varphi(\operatorname{vec}(\prod_i \mathcal{T}\mathbf{G}_i)) = \inf_{\mathbf{G}_i\in\mathfrak{G}_i} \varphi(\otimes_i^{\downarrow}\mathbf{G}_i \operatorname{vec}(\mathcal{T})).$$

Similar to the matrix case in Section 6.4.1, we now illustrate how the Tucker decomposition can be induced from an optimization formulation. Given a group $\mathfrak{G}=\otimes_i^{\downarrow}\mathfrak{G}_i$ and $\mathbf{G}=\otimes_i^{\downarrow}\mathbf{G}_i$ and a tensor $\mathcal{T}$, we define the following optimization problem

$$\inf_{\mathbf{G}\in\mathfrak{G}} \varphi(\mathbf{G}\operatorname{vec}(\mathcal{T})) = \inf_{\mathbf{G}_i\in\mathfrak{G}_i} \varphi((\otimes_i^{\downarrow}\mathbf{G}_i)\operatorname{vec}(\mathcal{T})) = \inf_{\mathbf{G}_i\in\mathfrak{G}_i} \varphi(\operatorname{vec}(\prod_i \mathcal{T}\mathbf{G}_i))).$$

If we assume that $\hat{\mathbf{G}}=\otimes_i^{\downarrow}\hat{\mathbf{G}}_i=\arg\inf_{\mathbf{G}\in\mathfrak{G}} \varphi(\mathbf{G}\operatorname{vec}(\mathcal{T}))$ and $\mathcal{Z}=\prod_i \mathcal{T}\hat{\mathbf{G}}_i$, then $\mathcal{M}=\prod_i \mathcal{Z}\hat{\mathbf{G}}_i^{-1}$ can be regarded as a tensor decomposition induced from the optimization problem.

In this section, we particularly generalize the results in Sections 6.3 and 6.4 to tensors. In Lemma 6.21, we prove that we can use the subgroup relation to induce a partial order of the infima of GOO. We also show that GOO w.r.t. the special linear group finds the "sparsest" Tucker-like decomposition of a tensor, and prove that GOO on tensor $\mathcal{T}$ is "denser" than GOO on any matrix unfolded from $\mathcal{T}$.

6.5.1 ▪ Subgroup Hierarchy

First we observe the following partial order of infima of GOO induced from a subgroup relation.

Lemma 6.21 (Infima partial order from subgroup relation). *If $\mathfrak{G}_1$ is a subgroup of $\mathfrak{G}_2$, then for any $\phi : \mathbb{F}^{n_1 \times n_2 \times \cdots n_k} \to \mathbb{R}$:*

$$\inf_{\mathbf{G} \in \mathfrak{G}_1} \phi(\mathbf{GM}) \geq \inf_{\mathbf{G} \in \mathfrak{G}_2} \phi(\mathbf{GM}).$$

Proof. As $\mathfrak{G}_1$ is a subgroup of $\mathfrak{G}_2$, the set of optimization variables of the left-hand side is a subset of those of the right-hand side. Hence, the inequality holds. □

Corollary 6.22. *For a matrix $\mathbf{M}$, we can construct an upper bound of the Schatten p-norm via*

$$\inf_{\mathbf{G} \in \mathfrak{U}} \|\mathbf{GM}\|_p \geq \|\mathbf{M}\|_{*p}.$$

Proof.

$$\inf_{\mathbf{G} \in \mathfrak{U}} \|\mathbf{GM}\|_p = \inf_{\mathbf{G} \in \mathfrak{U}} \|(\mathbf{I} \otimes \mathbf{G}) \operatorname{vec}(\mathbf{M})\|_p \geq \inf_{\mathbf{G}_1, \mathbf{G}_2 \in \mathfrak{U}} \|(\mathbf{G}_2 \otimes \mathbf{G}_1) \operatorname{vec}(\mathbf{M})\|_p = \|\mathbf{M}\|_{*p}.$$

□

Lemma 6.23 (GOO w.r.t. special linear group). *The infimum of GOO w.r.t. the special linear group is the smallest among all GOO w.r.t. a unit matrix group $\mathfrak{G}$ and the same ϕ for a tensor, that is,*

$$\inf_{\otimes_i^{\downarrow} \mathbf{G}_i \in \mathfrak{G}} \phi(\prod_i \mathcal{T} \mathbf{G}_i) \geq \inf_{\mathbf{G}_i \in \mathfrak{SL}(n_i)} \phi(\prod_i \mathcal{T} \mathbf{G}_i). \tag{6.8}$$

Proof. First we note for $\mathbf{A} \in \mathbb{F}^{m \times m}$, $\mathbf{B} \in \mathbb{F}^{n \times n}$,

$$\det(\mathbf{A} \otimes \mathbf{B})^{\frac{1}{mn}} = \det(\mathbf{A})^{\frac{1}{m}} \det(\mathbf{A})^{\frac{1}{n}}.$$

Let $\mathbf{Z}_i \in \mathfrak{GL}(n_i)$ and $\det(\otimes_i^{\downarrow} \mathbf{Z}_i) = 1$. We can find $\mathbf{N}_i \in \mathfrak{SL}(n_i)$ such that

$$\mathbf{N}_i = (\det(\mathbf{Z}_i))^{-\frac{1}{n_i}} \mathbf{Z}_i \ .$$

Now as $\det(\otimes_i^{\downarrow} \mathbf{Z}_i) = \det(\otimes_i^{\downarrow} \mathbf{N}_i) = 1$, we have $\otimes_i^{\downarrow} \mathbf{N}_i = \otimes_i^{\downarrow} \mathbf{Z}_i$. By Lemma 6.21, we have

$$\inf_{\otimes_i^{\downarrow} \mathbf{G}_i \in \mathfrak{G}} \phi(\prod_i \mathcal{T} \mathbf{G}_i) \geq \inf_{\mathbf{Z}_i \in \mathfrak{GL}(n_i), \det(\otimes_i^{\downarrow} \mathbf{Z}_i) = 1} \phi(\prod_i \mathcal{T} \mathbf{Z}_i) = \inf_{\mathbf{G}_i \in \mathfrak{SL}(n_i)} \phi(\prod_i \mathcal{T} \mathbf{G}_i).$$

□

Next we show that GOO gives a unified framework for matrix decomposition and tensor decomposition. We can rewrite decomposition in Table 6.2 in tensor notation as

$$\mathbf{M} = \prod_i \mathbf{D}\mathbf{G}_i,$$

which is induced w.r.t. a cost function ϕ by optimization

$$\inf_{\mathbf{G}\in\mathfrak{G}_i} \phi(\prod_i \mathbf{M}\mathbf{G}_i).$$

Now if there exists φ s.t. $\phi = \varphi \circ \mathrm{vec}$, we can generalize ϕ to tensor as

$$\tilde{\phi}(\mathcal{T}) =< \mathbf{1},\, f(\mathcal{T}) > ,$$

where $\mathbf{1}$ is a tensor with all entries being 1 and of the same dimension as $\mathcal{T}$.

This inspires us to define a tensor version of the above optimization and decomposition as below:

$$\mathcal{T} = \prod_i \mathcal{Z}\mathbf{G}_i,$$

and

$$\inf_{\mathbf{G}\in\mathfrak{G}_i} \tilde{\phi}(\prod_i \mathcal{T}\mathbf{G}_i).$$

In particular, if a matrix decomposition can be induced by entry-wise cost function $\phi(\mathbf{M}) = \sum_{ij} f(m_{ij})$ w.r.t. some unit group, we can consistently generalize the matrix decompositions to tensors using cost function $\tilde{\phi}(\mathcal{T}) =< \mathbf{1}, f(\mathcal{T}) >$. In this case, there is an inequality relation that follows from the Lemma 6.21.

Lemma 6.24 (Lifting lemma). *We are given a tensor $\mathcal{T}$ and its arbitrary unfolding $\mathrm{fold}_I^{-1}(\mathcal{T})$ w.r.t. an index set grouping I. Let $\{m_i\}$ and $\{n_j\}$ be the sizes of the square matrices before and after grouping. Then we have*

$$\inf_{\mathbf{M}_i\in\mathfrak{G}(m_i)} \phi(\prod_i \mathcal{T}\mathbf{M}_i) \geq \inf_{\mathbf{N}_j\in\mathfrak{G}(n_j)} \phi(\prod_j \mathrm{fold}_I^{-1}(\mathcal{T})\mathbf{N}_j).$$

Proof. We note $\mathfrak{G}(a) \otimes \mathfrak{G}(b)$ is a subgroup of $\mathfrak{G}(a+b)$. Hence

$$\begin{aligned}
\inf_{\mathbf{M}_i\in\mathfrak{G}(m_i)} \phi(\prod_i \mathcal{T}\mathbf{M}_i) &= \inf_{\mathbf{M}_i\in\mathfrak{G}(m_i)} \phi((\otimes_i^{\downarrow}\mathbf{M}_i)\,\mathrm{vec}(\mathcal{T})) \\
&\geq \inf_{\mathbf{N}_j\in\mathfrak{G}(n_j)} \phi((\otimes_j^{\downarrow}\mathbf{N}_j)\,\mathrm{vec}(\mathcal{T})) \\
&= \inf_{\mathbf{N}_j\in\mathfrak{G}(n_j)} \phi(\prod_j \mathrm{fold}_I^{-1}(\mathcal{T})\mathbf{N}_j).
\end{aligned}$$

□

6.5.2 ▪ An Upper Bound for Some Tensor Norms

In the literature, there are multiple generalizations of the Schatten p-norm to tensors. For example, the tensor unfolding trace norm [90, 123] is defined as a weighted sum of the trace norm of single index unfoldings of the tensor; namely,

$$\sum_{i=1}^{k} \alpha_i \| \operatorname{fold}_i^{-1} \mathcal{T} \|_* \,, \tag{6.9}$$

where $\alpha_i \geq 0$ and $\sum_i \alpha_i = 1$.

Another generalization as given in [117] is defined by

$$(\sum_{i=1}^{k} \frac{1}{k} \| \operatorname{fold}_i^{-1} \mathcal{T} \|_{*q}^p)^{\frac{1}{p}} \,. \tag{6.10}$$

These tensor norms are interesting as they correspond to the Schatten norm of matrix. We next study use of the Lemma 6.24 to construct an upper bound for the two norms.

Formula 6.9 and Formula 6.10 try to capture the tensor structure by considering all single-index unfoldings of the tensor. However, there are many unfoldings that are not single index. In general, for a kth-order tensor, there are $2^{k-1} - 2$ possible unfoldings, as in the following example.

Example 6.25. A 3rd order tensor has 3 unfoldings w.r.t. the following index set grouping: $\{\{1\},\{2,3\}\}, \{\{2\},\{1,3\}\}, \{\{3\},\{1,2\}\}$. Here $\{\{1\},\{2,3\}\}$ means putting index 1 of the tensor in the first dimension of the unfolded matrix, and index 2 and 3 of the tensor in the second dimension of the unfolded matrix. Additionally, a 4th th-order tensor has 6 unfoldings. ∎

It turns out the following GOO that respects the tensor structure produces an upper bound for the Schatten p-norm of the matrices unfolded from a tensor.

Lemma 6.26 (Infimum of GOO w.r.t. the unitary group). *For $0 \leq p < 2$, and for any index set grouping J, we have:*

$$\inf_{\mathbf{G}_i \in \mathfrak{U}_i} \| \prod_i \mathcal{T} \mathbf{G}_i \|_p \geq \inf_{\mathbf{G}_j \in \mathfrak{U}_j} \| \prod_j (\operatorname{fold}_J^{-1} \mathcal{T}) \mathbf{G}_j \|_p \,.$$

Similarly for $p > 2$, we have:

$$\sup_{\mathbf{G}_i \in \mathfrak{U}_i} \| \prod_i \mathcal{T} \mathbf{G}_i \|_p \leq \sup_{\mathbf{G}_j \in \mathfrak{U}_j} \| \prod_j (\operatorname{fold}_J^{-1} \mathcal{T}) \mathbf{G}_j \|_p \,.$$

Proof. The inequalities is obtained by applying Lemma 6.24 to GOO w.r.t. unitary group and $f(x) = \|x\|_p$ when $0 \leq p < 2$ and $f(x) = -\|x\|_p$ when $p > 2$. □

Corollary 6.27. *For $0 \leq p < 2$, we have*

$$\inf_{\mathbf{G}_i \in \mathfrak{U}_i} \| \prod_i \mathcal{T} \mathbf{G}_i \|_p \geq \sum_{i=1}^{k} \alpha_i \| \operatorname{fold}_i^{-1} \mathcal{T} \|_* \,,$$

where $\alpha_i \geq 0$ *and* $\sum_i \alpha_i = 1$. *We also have:*

$$\inf_{\mathbf{G}_i \in \mathfrak{U}_i} \| \prod_i \mathcal{T} \mathbf{G}_i \|_p \geq (\sum_{i=1}^{k} \frac{1}{k} \| \operatorname{fold}_i^{-1} \mathcal{T} \|_{*p}^q)^{\frac{1}{q}}.$$

Proof. By Lemma 6.26, we have

$$\inf_{\mathbf{G}_i \in \mathfrak{U}_i} \| \prod_i \mathcal{T} \mathbf{G}_i \|_p \geq \max_{1 \leq i \leq k} \inf_{\mathbf{G}_j \in \mathfrak{U}_j} \| \prod_j (\operatorname{fold}_i^{-1} \mathcal{T}) \mathbf{G}_j \|_p = \max_{1 \leq i \leq k} \| \operatorname{fold}_i^{-1} \mathcal{T} \|_{*p} .$$

We prove the inequalities as the following relations hold:

$$\max_{1 \leq i \leq k} \| \operatorname{fold}_i^{-1} \mathcal{T} \|_{*p} \geq \sum_{i=1}^{k} \alpha_i \| \operatorname{fold}_i^{-1} \mathcal{T} \|_* ,$$

and

$$\max_{1 \leq i \leq k} \| \operatorname{fold}_i^{-1} \mathcal{T} \|_{*p} = (\sum_{i=1}^{k} \frac{1}{k} \max_{1 \leq i \leq k} \| \operatorname{fold}_i^{-1} \mathcal{T} \|_{*p}^q)^{\frac{1}{q}} \geq (\sum_{i=1}^{k} \frac{1}{k} \| \operatorname{fold}_i^{-1} \mathcal{T} \|_{*p}^q)^{\frac{1}{q}} .$$

□

Due to the above inequality, an optimization that tries to minimize one of tensor norms defined as in Formula 6.9 and Formula 6.10 can have the tensor norm replaced by $\inf_{\mathbf{G}_i \in \mathfrak{U}_i} \| \prod_i \mathcal{T} \mathbf{G}_i \|_p$, as minimizing upper bound of a function f can always minimize f.

6.5.3 ▪ Sparse Structure in Tensor

The tensor rank is defined as the minimum number of non-zero rank-1 tensors required to sum up to $\mathcal{T}$, which is a generalization of the matrix rank. In the Tucker decomposition $\mathcal{T} = \prod_i \mathcal{X} \mathbf{U}_i$, one can define the Tucker rank w.r.t. the different constraints on the $\mathbf{U}_i$. For example, for a real tensor, when the $\mathbf{U}_i$ are required to be orthogonal, the number of nonzeros in $\mathcal{X}$ is defined as a tensor strong orthogonal rank of $\mathcal{T}$ [71]. Trivially, the tensor rank is a lower bound of all the Tucker ranks.

Tensor decomposition has a large body of the literature [52] [144] [79] [78] [80] [25] [63], the interested reader may refer to [72] and the references therein.

We find that the strong orthogonal rank of tensor $\mathcal{T}$ is exactly the infimum of the following GOO problem

$$\operatorname{rank}_{so} \mathcal{T} = \inf_{\mathbf{G}_i \in \mathfrak{U}_i} \| \prod_i \mathcal{T} \mathbf{G}_i \|_0.$$

Corollary 6.28. *We have*

$$\inf_{\det(\otimes_i^{\downarrow} \mathbf{G}_i) = 1} \| \prod_i \mathcal{T} \mathbf{G}_i \|_0 \geq \inf_{\mathbf{G}_i \in \mathfrak{S}\mathfrak{L}(n_i)} \| \prod_i \mathcal{T} \mathbf{G}_i \|_0 \geq \operatorname{rank}(\mathcal{T}).$$

Proof. The first inequality directly follows from Lemma 6.23 by choosing $\phi = \| \cdot \|_0$. For the second inequality, as the problem $\inf_{\mathbf{G}_i \in \mathfrak{G}_i} \| \prod_i \mathcal{T} \mathbf{G}_i \|_0$ induces a tensor decomposition into

$\|\prod_i \mathcal{T}\hat{\mathbf{G}}_i\|_0$ number of rank-1 tensors, by definition of the tensor rank we have the following inequality:

$$\inf_{\mathbf{G}_i \in \mathfrak{G}_i} \|\prod_i \mathcal{T}\mathbf{G}_i\|_0 \geq \operatorname{rank}(\mathcal{T}).$$

□

When $\phi = \|\cdot\|_0$, Lemma 6.24 provides a link between the rank of the tensor $\mathcal{T}$ and the ranks of the matrices or the vectors unfolded from $\mathcal{T}$. For example, when $\mathfrak{G} = \mathfrak{U}$ and $\mathcal{T} \neq \mathbf{0}$, we have

$$\inf_{\mathbf{G} \in \mathfrak{U}} \|\mathbf{G}\operatorname{vec}(\mathcal{T})\|_0 = 1.$$

6.6 ▪ Application: Point Cloud Normalization

Data normalization seeks to eliminate some arbitrary degrees of freedom in data. For example, when we are concerned with shape of an object, its attitude and position in space will become irrelevant. Given a point cloud, which is a sequence of coordinates of points, we demonstrate a method to obtain a representation of point clouds that does not depend on its attitude and position in this section. We craft the method as a special case of GOO with some particular choice of group and cost function.

In this section we use group $\mathbf{P}^\top \otimes \mathbf{I}_m, \mathbf{P} \in \mathfrak{SL}(n)$ for normalization of point cloud data. We assume that the distortions to the point cloud data are of a few categories of degrees of freedom: including mirroring, rotation, shearing and squeezing, which we seek to eliminate using the special linear group orbit.

Lemma 6.29. *The special linear group can represent any combination of mirroring, rotation, shearing and squeezing operations for point cloud data.*

Proof. Every special linear matrix $\mathbf{M}$ can be QR decomposed as $\mathbf{M} = \mathbf{QR}$, and $\mathbf{R}$ can be decomposed into $\mathbf{R} = \mathbf{DU}$ where $\mathbf{D}$ is diagonal and $\mathbf{U} \in \mathfrak{UUT}$. Accordingly, we have a decomposition $\mathbf{M} = \mathbf{QDU}$, where $\mathbf{U}$ models the shearing operation and $\mathbf{Q}$ models the rotation. As $\det(\mathbf{M}) = 1$, $|\det \mathbf{D}| = |\frac{\det \mathbf{M}}{\det \mathbf{Q} \det \mathbf{U}}| = 1$. Hence, the diagonal matrix $\mathbf{D}$ is the squeezing operation (optionally with the mirror operation). Hence the action of $\mathbf{G} \in \mathfrak{SL}$ applied to $\mathbf{M}$ is equivalent to the sequential application of rotation, squeezing, mirroring, and shearing. Now since the special linear group is a group, arbitrary composition of these operations can still be represented as some $\mathbf{G}' \in \mathfrak{SL}$. □

We show that for some point clouds, the normalized forms are exactly the axis-aligned hypercubes.

Lemma 6.30. *Given a matrix* $\mathbf{M} \in \mathbb{R}^{n \times d}$*, if* $\operatorname{Poly}(\mathbf{M})$ *is an axis-aligned hypercube and* $\det \mathbf{G} = 1$*, then*

$$\|\mathbf{MG}\|_\infty \geq \|\mathbf{M}\|_\infty.$$

Proof. First note that as $\det(\mathbf{G}) = 1$, given a Lebesgue measure μ, we have:

$$\mu(\operatorname{Poly}(\mathbf{MG})) = \mu(\operatorname{Poly}(\mathbf{M})).$$

We can construct a bounding box $\operatorname{Poly}(\mathbf{Z})$ for $\operatorname{Poly}(\mathbf{MG})$ with center at the origin and edge length $2\|\mathbf{MG}\|_\infty$. Note that $\operatorname{Poly}(\mathbf{Z})$ is also a hypercube and $\|\mathbf{Z}\|_\infty = \|\mathbf{MG}\|_\infty$. We thus have

$$\mu(\operatorname{Poly}(\mathbf{Z})) \geq \mu(\operatorname{Poly}(\mathbf{MG})) = \mu(\operatorname{Poly}(\mathbf{M})).$$

Because for any axis-aligned hypercube $\mathrm{Poly}(\mathbf{Y})$ we have $\mu(\mathrm{Poly}\,\mathbf{Y}) = 2^d\|\mathbf{Y}\|_\infty^d$. Hence, the following holds:

$$2^d\|\mathbf{MG}\|_\infty^d = 2^d\|\mathbf{Z}\|_\infty^d \geq 2^d\|\mathbf{M}\|_\infty^d.$$

□

By Lemma 6.11, we can prove the following corollary.

Corollary 6.31. *Let* $\mathbf{M} \in \mathbb{R}^{n\times k}$ *be a matrix such that* $\mathrm{Poly}(\mathbf{M})$ *can be transformed by* $\mathbf{G} \in \mathfrak{SL}(k)$ *into a hypercube. Then the following optimization problem attains its optimum when* $\mathrm{Poly}(\mathbf{M}\hat{\mathbf{G}})$ *is a hypercube:*

$$\inf_{\mathbf{G}\in\mathfrak{SL}(n)} \|\mathbf{MG}\|_\infty.$$

The optimization problem in Theorem 6.31 is not convex. For example, in $\mathbb{R}^2$, a square can be rotated by 90 degrees, 180 degrees, and 270 degrees while still being a square. Nevertheless, the degree of freedom associated with rotation, squeezing and shearing described by the special linear group is reduced to only one of four configurations. The three other optimal $\hat{\mathbf{G}}$ can be enumerated when one optimal $\mathbf{G}$ is known.

Optimality of $\inf_{\mathbf{G}\in\mathfrak{SL}(2)} \|\mathbf{MG}\|_\infty$ depends on whether $\mathrm{Poly}(\mathbf{MG})$ is a parallelogram. In practice, we find the above method works well in normalizing point clouds that arise in shape recognition. In Section 6.7 we will present several concrete examples.

6.7 ▪ Numerical Algorithm and Examples

6.7.1 ▪ Algorithm

Most of optimizations involved in this paper are the following constrained optimization problem of form:

$$\inf_{\mathbf{G}\in\mathfrak{G}} \varphi(\mathbf{G}\,\mathrm{vec}(\mathbf{M})), \tag{6.11}$$

where $\mathfrak{G}$ is a unit group. When $\mathfrak{G}$ is a compact Lie group, alternatively we can turn the above optimization to another constrained optimization:

$$\inf_{\mathbf{Z}\in\mathfrak{g}} \varphi(\exp(\mathbf{Z})\,\mathrm{vec}(\mathbf{M})),$$

where $\exp(\mathbf{G})$ is matrix exponential of matrix $\mathbf{G}$ and $\mathfrak{g}$ is the Lie algebra associated with Lie group $\mathfrak{G}$. This formulation may have constraints that are easier to encode in numeric software. For example, considering

$$\inf_{\mathbf{G}_1\in\mathfrak{SO}(m),\mathbf{G}_2\in\mathfrak{SO}(n)} \varphi(\mathrm{vec}(\mathbf{G}_2\mathbf{M}\mathbf{G}_1^\top)),$$

we have $\mathfrak{G} = \mathfrak{SO}(m)\otimes\mathfrak{SO}(n)$, $\mathfrak{g} = \{\mathbf{Z}_2\oplus\mathbf{Z}_1 : \mathbf{Z}_1 \in \mathbb{F}^{n\times n}, \mathbf{Z}_1+\mathbf{Z}_1^\top = 0, \mathbf{Z}_2 \in \mathbb{F}^{m\times m}, \mathbf{Z}_2+\mathbf{Z}_2^\top = 0\}$. Hence, we can turn the optimization into a constrained optimization over $\mathfrak{g}$ as

$$\inf_{\mathbf{Z}\in\mathfrak{g}} \varphi(\exp(\mathbf{Z})\,\mathrm{vec}(\mathbf{M})).$$

Moreover, in this particular case, we can turn the above optimization into an unconstrained optimization:

$$\inf_{\mathbf{G}_1\in\mathbb{F}^{n\times n},\mathbf{G}_2\in\mathbb{F}^{m\times m}} \varphi(\mathrm{vec}(\exp(\mathbf{G}_2-\mathbf{G}_2^\top)\mathbf{M}\exp(\mathbf{G}_1-\mathbf{G}_1^\top)^\top)).$$

In Table 6.3 we list a few more cases when the constrained optimization of Formula 6.11 can be turned into an unconstrained optimization.

Table 6.3. *Encoding of constraints for compact Lie groups*

Lie group	Lie algebra	Encoding of Constraint
$\mathfrak{SO}$	$\{\mathbf{Z} : \mathbf{Z} + \mathbf{Z}^\top = 0\}$	$\mathbf{Z} = \mathbf{X} - \mathbf{X}^\top$
$\mathfrak{LUT}$	$\{\mathbf{Z} : \operatorname{dg} \mathbf{Z} = \mathbf{0}, \mathbf{Z} \in \mathfrak{LUT}\}$	$\mathbf{Z} = \mathbf{X} \odot [\mathbb{I}_{i>j}]$
$\mathfrak{UUT}$	$\{\mathbf{Z} : \operatorname{dg} \mathbf{Z} = \mathbf{0}, \mathbf{Z} \in \mathfrak{UUT}\}$	$\mathbf{Z} = \mathbf{X} \odot [\mathbb{I}_{i<j}]$
$\mathfrak{SL}$	$\{\mathbf{Z} : \operatorname{tr} \mathbf{Z} = 0\}$	$\mathbf{Z} = \mathbf{X} - (\operatorname{tr} \mathbf{X})\mathbf{v}\mathbf{v}^\top$, where $\mathbf{v} = [\mathbf{1}, \mathbf{0}, \mathbf{0}, \ldots, \mathbf{0}]^\top$
$\mathfrak{G}_1 \otimes \mathfrak{G}_2$	$\mathfrak{g}_1 \oplus \mathfrak{g}_2$	

The exponential mapping used for optimization over compact Lie groups is related to other optimization on manifold methods [126, 34, 1]. When $\mathfrak{G}$ is a compact Lie group, we are able to build following simple template gradient descent algorithm.

Algorithm 6.1.

$$\frac{\partial}{\partial t}\mathbf{Z} = -\frac{\partial}{\partial \mathbf{Z}}\varphi(\exp(\mathbf{Z})\operatorname{vec}(\mathbf{M})).$$

The gradient $\frac{\partial}{\partial \mathbf{Z}}\varphi(\exp(\mathbf{Z})\operatorname{vec}(\mathbf{M}))$ can be computed in an automatic differentiation framework like Theano [7, 8], which in turn uses a result in [68].

In this section all numerical optimizations are solved by Nelder-Meld heuristic global optimization algorithm [99] implemented in Mathematica$^{\text{TM}}$ 9.0.0, unless noted.

6.7.2 ▪ GOO Inducing Matrix Decomposition

We empirically illustrate several examples of GOO inducing matrix decomposition. Due to the large amount of computation required by Nelder-Meld algorithm, here we only give a few examples involving small matrices.

Example 6.32 (Compute SVD of a 3×3 real matrix). Given a matrix $\mathbf{M}$:

$$\mathbf{M} \approx \begin{bmatrix} 0.17658 & 0.517888 & 0.448587 \\ 0.214066 & 0.718154 & 0.849892 \\ 0.796042 & 0.197801 & 0.233489 \end{bmatrix},$$

the SVD of $\mathbf{M} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^\top$ is given as

$$\mathbf{U} \approx \begin{bmatrix} -0.483076 & -0.175226 & -0.857865 \\ -0.768129 & -0.385453 & 0.511276 \\ -0.420256 & 0.905937 & 0.0516068 \end{bmatrix},$$

$$\mathbf{\Lambda} \approx \begin{bmatrix} 1.43557 & 0. & 0. \\ 0. & 0.66535 & 0. \\ 0. & 0. & 0.0910448 \end{bmatrix},$$

$$\mathbf{V} \approx \begin{bmatrix} -0.406999 & 0.913369 & -0.0104708 \\ -0.616441 & -0.28311 & -0.734745 \\ -0.674057 & -0.292586 & 0.678263 \end{bmatrix}.$$

We now use the optimization problem:

$$\inf_{\mathbf{U},\mathbf{V}\in\mathfrak{SO}(3)} \|\mathbf{U}^\top \mathbf{M}\mathbf{V}\|_1$$

to find the SVD of $\mathbf{M}$. A numerical solution, produced by the heuristic global

We use the optimization

$$\inf_{\mathbf{L}\in\mathfrak{LUT}(3)} \|\Delta \odot (\mathbf{L}^{-1}\mathbf{M})\|_1$$

to find the LU of $\mathbf{M}$, where $\Delta_{ij} = \mathbb{I}_{i>j}$. A numerical solution is given as

$$\hat{\mathbf{L}}\mathbf{\Lambda} \approx \begin{bmatrix} 0.843023 & 0. & 0. \\ 0.477613 & 0.771339 & 0. \\ 0.530248 & 0.824024 & 0.133735 \end{bmatrix}.$$

Here $\mathbf{\Lambda}$ is a diagonal matrix with square root of diagonals of $\hat{\mathbf{L}}^{-1}\mathbf{M}$ as its diagonals.

∎

Example 6.33 (Compute Schur decomposition of a 3×3 matrix). We use the same $\mathbf{M}$ as in Example 6.32. The Schur decomposition of $\mathbf{M} = \mathbf{Q}\mathbf{U}\mathbf{Q}^{-1}$ is given by

$$\Re\mathbf{Q} \approx \begin{bmatrix} -4.4809 \times 10^{-1} & 2.13496 \times 10^{-2} & -3.11518 \times 10^{-1} \\ -6.81147 \times 10^{-1} & -5.00762 \times 10^{-1} & 1.85125 \times 10^{-3} \\ -4.25133 \times 10^{-1} & 7.79816 \times 10^{-1} & 3.25373 \times 10^{-1} \end{bmatrix},$$

$$\Im\mathbf{Q} \approx \begin{bmatrix} -1.91558 \times 10^{-1} & 2.76329 \times 10^{-1} & -7.67245 \times 10^{-1} \\ -2.91189 \times 10^{-1} & -2.4227 \times 10^{-2} & 4.47096 \times 10^{-1} \\ -1.81744 \times 10^{-1} & -2.52434 \times 10^{-1} & 9.23392 \times 10^{-2} \end{bmatrix},$$

$$\Re\mathbf{U} \approx \begin{bmatrix} 1.38943 & -2.5768 \times 10^{-1} & -1.15903 \times 10^{-1} \\ 0. & -1.30605 \times 10^{-1} & -7.89372 \times 10^{-2} \\ 0. & 0. & -1.30605 \times 10^{-1} \end{bmatrix},$$

$$\Im\mathbf{U} \approx \begin{bmatrix} -1.38778 \times 10^{-16} & 2.11604 \times 10^{-1} & 3.88987 \times 10^{-2} \\ 0. & 2.13379 \times 10^{-1} & -5.69018 \times 10^{-1} \\ 0. & 0. & -2.13379 \times 10^{-1} \end{bmatrix},$$

We can use the optimization

$$\inf_{\mathbf{Q}\in\mathfrak{U}(3)} \|\Delta \odot (\mathbf{Q}^{-1}\mathbf{M}\mathbf{Q})\|_1$$

to find the Schur decomposition of $\mathbf{M}$, where $\Delta_{ij} = \mathbb{I}_{i>j}$. A numerical solution is given as

$$\Re\hat{\mathbf{Q}} \approx \begin{bmatrix} -0.159122 & 0.456745 & -0.924103 \\ -0.697698 & 0.596808 & 0.46622 \\ 0.777452 & 0.665151 & 0.227028 \end{bmatrix},$$

$$\Im\hat{\mathbf{Q}} \approx \begin{bmatrix} -0.288006 & -0.0686255 & 0.0156732 \\ 0.161731 & 0.0500731 & 0.177932 \\ 0.0861939 & 0.00219548 & -0.301602 \end{bmatrix},$$

$$\Re\hat{\mathbf{U}} \approx \begin{bmatrix} \mathbf{-1.30605 \times 10^{-1}} & -3.71633 \times 10^{-1} & -7.07291 \times 10^{-1} \\ -2.09852 \times 10^{-9} & \mathbf{1.38943} & -1.09975 \times 10^{-1} \\ -1.39859 \times 10^{-8} & 9.97677 \times 10^{-9} & \mathbf{-1.30604 \times 10^{-1}} \end{bmatrix},$$

$$\Im\hat{\mathbf{U}} \approx \begin{bmatrix} \mathbf{-2.13379 \times 10^{-1}} & -1.95892 \times 10^{-2} & 2.66849 \times 10^{-2} \\ 2.82526 \times 10^{-9} & \mathbf{2.10889 \times 10^{-9}} & -1.05424 \times 10^{-1} \\ 9.09511 \times 10^{-9} & 8.28696 \times 10^{-9} & \mathbf{2.13379 \times 10^{-1}} \end{bmatrix}.$$

We note that $\hat{\mathbf{U}}$ is permuted approximation of $\mathbf{U}$ modulo sign.

■

6.7.3 ▪ GOO Inducing Tensor Decomposition

We empirically illustrate several examples of GOO inducing tensor decomposition. Due to the large amount of computation required by the Nelder-Meld algorithm, here we only give examples involving small-size tensors.

Example 6.34 (Non-uniqueness of strong-orthogonal decomposition). Consider tensor $\mathcal{A}$ as in Example 3.3 of [71], which we reproduce below:

$$\mathcal{A} = \sigma_1 \mathbf{a} \otimes \mathbf{b} \otimes \mathbf{b} + \sigma_2 \mathbf{b} \otimes \mathbf{b} \otimes \mathbf{b} + \sigma_3 \mathbf{a} \otimes \mathbf{a} \otimes \mathbf{a}, \tag{6.12}$$

where $\sigma_1 > \sigma_2 > \sigma_3, \mathbf{a} \perp \mathbf{b}, \|\mathbf{a}\| = \|\mathbf{b}\| = 1$.

We note that Formula (6.12) is already a strong orthogonal decomposition of $\mathcal{A}$. Nevertheless, an alternative strong orthogonal decomposition is given therein as

$$\mathcal{A} = \hat{\sigma}_1 \hat{\mathbf{a}} \otimes \mathbf{b} \otimes \mathbf{b} + \hat{\sigma}_2 \hat{\mathbf{a}} \otimes \mathbf{a} \otimes \mathbf{a} + \hat{\sigma}_3 \hat{\mathbf{b}} \otimes \mathbf{a} \otimes \mathbf{a}, \tag{6.13}$$

where

$$\hat{\sigma}_1 = \sqrt{\sigma_1^2 + \sigma_2^2}, \qquad \hat{\sigma}_2 = \frac{\sigma_1 \sigma_3}{\hat{\sigma}_2}, \qquad \hat{\sigma}_3 = \frac{\sigma_2 \sigma_3}{\hat{\sigma}_1},$$

$$\hat{\mathbf{a}} = \frac{\sigma_1 \mathbf{a} + \sigma_2 \mathbf{b}}{\hat{\sigma}_1}, \text{ and } \hat{\mathbf{b}} = \frac{\sigma_2 \mathbf{a} - \sigma_1 \mathbf{b}}{\hat{\sigma}_1}.$$

Without loss of generality, we let $\sigma_1 = 3, \sigma_2 = 2, \sigma_3 = 1, \mathbf{a} = [1, 0]^\top$, and $\mathbf{b} = [0, 1]^\top$. Then

$$\text{vec}\,(\mathcal{A}) = [1, 0, 0, 0, 0, 0, 3, 2]^\top,$$

$$\hat{\sigma}_1 \approx 3.60555, \qquad \hat{\sigma}_2 \approx 0.832050, \qquad \hat{\sigma}_3 \approx 0.5547002.$$

$$\inf_{\mathbf{G}_1 \in \mathfrak{SL}(2), \mathbf{G}_2 \in \mathfrak{SL}(2), \mathbf{G}_3 \in \mathfrak{SL}(2), \mathbf{G}_4 \in \mathfrak{SL}(2)} \|\mathcal{A} \times_1 \mathbf{G}_1 \times_2 \mathbf{G}_2 \times_3 \mathbf{G}_3 \times_4 \mathbf{G}_4\|_1$$

is

$$\operatorname{vec}(\mathcal{T} \times_1 \hat{\mathbf{G}}_1 \times_2 \hat{\mathbf{G}}_2 \times_3 \hat{\mathbf{G}}_3 \times_4 \hat{\mathbf{G}}_4) \approx \begin{bmatrix} \mathbf{1}. \\ 5.85196 \times 10^{-9} \\ 2.40735 \times 10^{-10} \\ \mathbf{-1}. \\ -2.39741 \times 10^{-9} \\ -1.40295 \times 10^{-17} \\ -5.77138 \times 10^{-19} \\ 2.3974 \times 10^{-9} \\ 4.74159 \times 10^{-9} \\ 2.77475 \times 10^{-17} \\ 1.14146 \times 10^{-18} \\ -4.74158 \times 10^{-9} \\ \mathbf{9.99999 \times 10^{-1}} \\ 5.85194 \times 10^{-9} \\ 2.40734 \times 10^{-10} \\ \mathbf{-9.99998 \times 10^{-1}} \end{bmatrix}.$$

Hence there are four significant nonzero values even under GOO w.r.t. the special linear group. ∎

6.7.4 ▪ Normalization of point cloud w.r.t. special linear group

Here we apply the GOO defined in Section 7 to a publicly available set of 2D point cloud data http://vision.lems.brown.edu/content/available-software-and-databases/Datasets-Shapehere.

As the optimization variable only consists of a small matrix $\mathbf{M} \in \mathbb{F}^{2\times 2}$, we are able to deal with large point clouds consisting of more than thousands of points. The detailed steps are as follows:

Algorithm 6.2.
Point cloud normalization

(1) Normalize the point cloud corresponding to $\mathbf{M}$ w.r.t. the special linear group as

$$\hat{\mathbf{M}} = \operatorname*{arginf}_{\mathbf{G}\in\mathfrak{GL}(n)} \|\mathbf{MG}\|_\infty.$$

(2) (Optional) Let $\hat{\mathbf{M}}_x$ and $\hat{\mathbf{M}}_y$ be two columns of $\hat{\mathbf{M}}$. We can use a simple criterion to select one from four possible forms of normalized point clouds: $[\hat{\mathbf{M}}_x, \hat{\mathbf{M}}_y]$, $[-\hat{\mathbf{M}}_y, \hat{\mathbf{M}}_x]$, $[-\hat{\mathbf{M}}_x, -\hat{\mathbf{M}}_y]$, and $[\hat{\mathbf{M}}_y, -\hat{\mathbf{M}}_x]$ to further eliminate ambiguity. An example is to pick the matrix $\hat{\hat{\mathbf{M}}}$ that minimizes $\phi(\mathbf{X}) = \|g(\mathbf{X})\|_F$ where $g(x) = \max(0, x)$. $\hat{\hat{\mathbf{M}}}$ is called a canonical form of $\mathbf{M}$ in this section.

We find that Step 2 in Algorithm 6.7.4 is useful in eliminating the ambiguity in orientation in some circumstances. However, even if Step 2 fails or is skipped, one can still use $\hat{\mathbf{M}}$ as "canonical" form and enumerate the few number of possible orientations. The result of Algorithm 6.7.4 without Step 2 has been shown in Figure 6.1 and Figure 6.2.

As an example of running time, normalization of a point cloud with 2867 points takes 49 milliseconds with 54 function evaluations and 42 gradient evaluations on a laptop with Intel(R) Core(TM) i7-4870HQ using single core.

In Figure 6.3 we perform a side-by-side comparison of results of several normalization techniques. The point clouds in the row marked with "Distorted" are produced by applying random shearing, mirroring, squeezing and rotation to the same point cloud. The point clouds in the row marked with "PCA" are results of applying PCA to the matrices corresponding to the distorted point clouds in the "Distorted" row. It can be seen that PCA can remove the degree of freedom corresponding to rotation in the input data, but fails to remove effect of squeezing and shearing. The row marked with "GOO_SO" is produced by using GOO with orthogonal group:

$$\inf_{\mathbf{G}\in\mathfrak{O}(n)} \|\mathbf{MG}\|_\infty.$$

We can see that effect of rotation is removed but effects of squeezing and shearing remain. The row marked with "GOO_SL" is the canonical forms of matrices corresponding to point clouds derived by Algorithm 6.7.4. We can see that the normalized point clouds are approximately the same, and effects of rotation, squeezing and shearing are almost completely eliminated.

In Figure 6.4 we study the impact of number of points on the canonical form found by the GOO. We can see that though the number of points in the canonical form vary between 180 and 260, the canonical form is nearly the same, module different orientations. In this case although Step 2 in Algorithm 6.7.4 cannot completely eliminate the ambiguity of four possible orientations of point clouds, we can simply remove this ambiguity by enumerating all four possible orientations when doing comparison.

6.8 ▪ Related Work

In this section we discuss the related work not yet mentioned in the previous sections. An early example of GOO is a so-called quadratic assignment problem (QAP) [111] where the following optimization problem is studied: $\inf_{\mathbf{X}\in\Pi_n} \operatorname{tr}(\mathbf{WXDX}^\top)$, where Π_n is the permutation matrix group. Due to the combinatorial nature of Π_n, QAP is NP-hard. In contrast, our work mainly considers non-combinatorial matrix groups.

In [145], a non-linear GOO is used to find texture invariant to rotation for 2D point cloud $\mathbf{M} \in \mathbb{F}^{n\times 2}$: $\inf_{\mathbf{G}\in\mathfrak{O}} \|\operatorname{Rasterize}(\operatorname{Poly}(\mathbf{MG}))\|_*$. As $\mathfrak{O}$ is a unit group, the optimization is well defined and the induced matrix decomposition is found to be useful as a rotation-invariant representation for texture. Also [145] considered finding a homography-invariant representation of texture for 2D point cloud $\mathbf{M} \in \mathbb{F}^{n\times 2}$:

$$\inf_{\mathbf{G}\in\mathfrak{H},\ \mu(\operatorname{Poly}(\lambda\mathbf{MG}))=\text{const}} \|\operatorname{Rasterize}[\operatorname{Poly}(\lambda\mathbf{MG})]\|_*.$$

Note that here a coefficient λ is added to ensure μ measure of the point cloud be preserved w.r.t. the action of $\mathbf{G}$.

In [61] the following formulation is used to get the Ky-Fan k-norm [59] of a matrix $\mathbf{M} \in \mathbb{F}^{m\times n}$ when $m \geq k$ and $n \geq k$:

$$\sup_{\mathbf{G}_1\in\mathbb{F}^{m\times k},\mathbf{G}_1^\top\mathbf{G}_1=\mathbf{I}_k,\mathbf{G}_2\in\mathbb{F}^{n\times k},\mathbf{G}_2^\top\mathbf{G}_2=\mathbf{I}_k} \operatorname{tr}(\mathbf{G}_1^\top\mathbf{MG}_2).$$

Note that the above optimization is not a GOO when $k^2 \neq mn$ as in this case $\mathbf{G}_2^\top \otimes \mathbf{G}_1^\top \in \mathbb{F}^{k^2\times mn}$ does not form a group.

In [135, 23, 55], an isospectral flow equation is used to induce QR, LU, SVD and Cholesky decomposition. In the framework of GOO, isospectral flow equations are gradient algorithms with respect to some specific cost functions. For example, for a self-adjoint matrix $\mathbf{H}$ and a diagonal matrix $\mathbf{N}$,

$$\frac{\mathrm{d}\mathbf{H}}{\mathrm{d}t} = [\mathbf{H}, [\mathbf{H}, \mathbf{N}]]$$

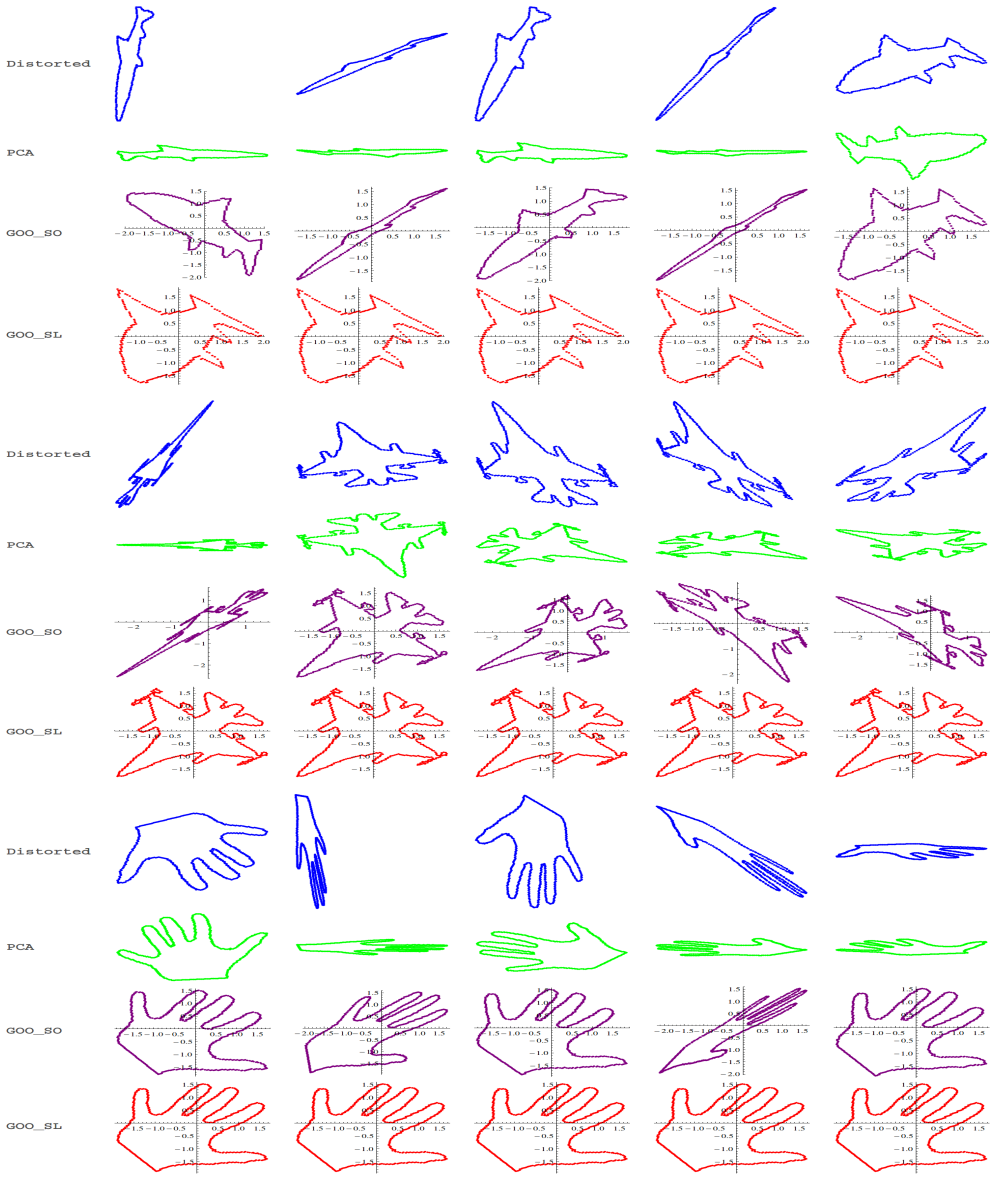


Figure 6.3. *This figures show the results of normalizing distorted point clouds by different methods. The rows marked with "Distorted" consist of distorted point clouds used as input to various normalization methods. The rows marked with "PCA" contain results of normalization by principal component analysis. It can be seen that the effects of rotational distortion have been partially eliminated, but results of shearing and squeezing remain. The rows marked with "GOO_SO" contain results of normalization using special orthogonal group in GOO. It can be seen that the effects of rotational distortion have been partially eliminated, but results of shearing and squeezing remain. The rows marked with "GOO_SL" contain results of normalization using Algorithm 6.7.4, where it can be seen that the algorithm can produce approximately the same point clouds after eliminating distortions like shearing, squeezing and rotation.*

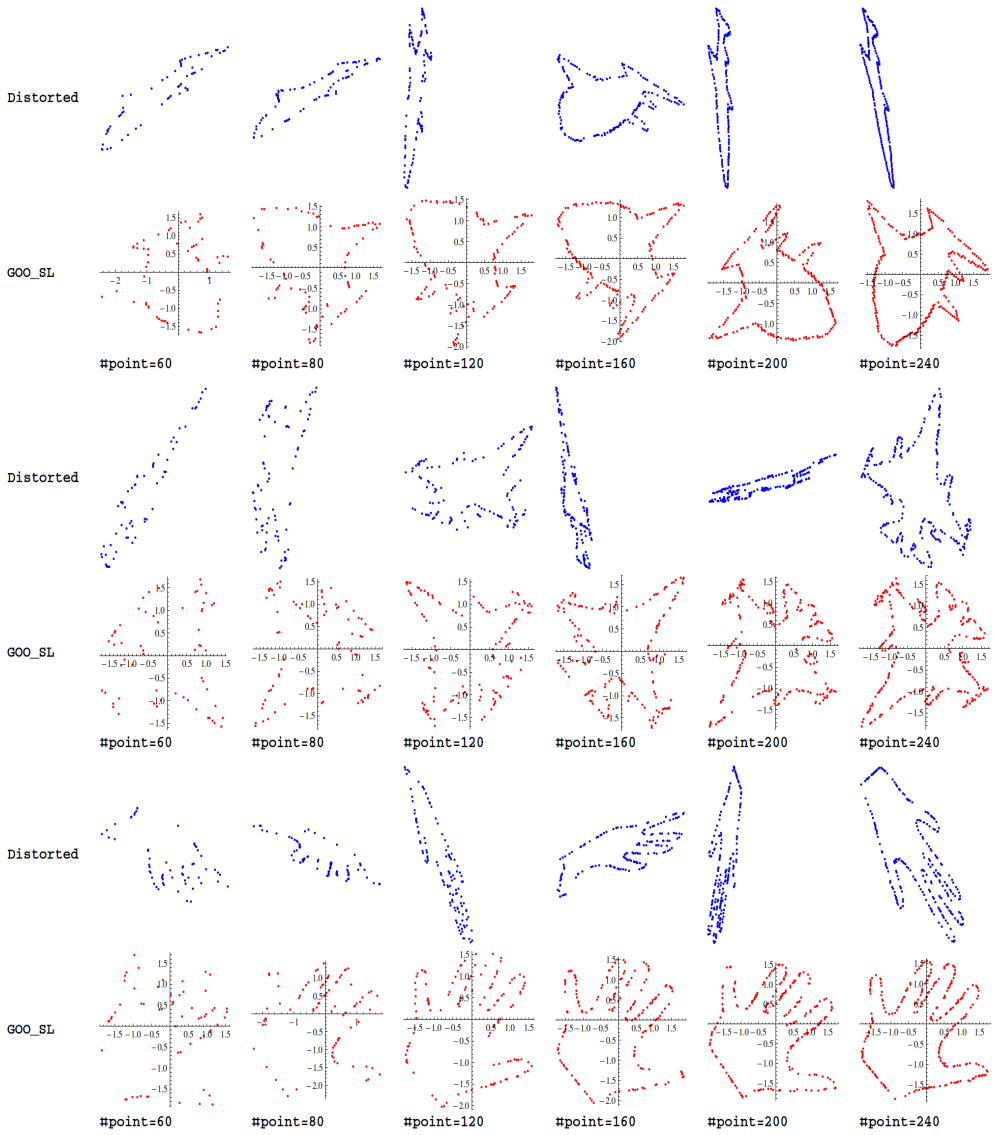


Figure 6.4. *The point clouds in rows marked with "Distorted" are the results of applying distortion generated by random special linear matrix to original point clouds. The point clouds in rows marked with "GOO_SL" are after normalization by special linear group. From left to right, the sparser point clouds are generated by sampling from the rightmost densest point cloud respectively. It can be seen that for all three examples, though density varies, the shape of the normalized point clouds remains stable, modulo four possible orientations.*

solves the following GOO problem

$$\underset{\mathbf{Q}\in\mathfrak{SO}(n)}{\operatorname{argmin}} \|\mathbf{N} - \mathbf{Q}^\top \mathbf{H}(0)\mathbf{Q}\|.$$

6.9 ▪ Conclusion

In this paper, we have shown that SVD/QR/LU/Cholesky/Schur decomposition can be reformulated under the GOO framework as in Theorem 6.10. Consequently, several data regularization functions are found to be expressible as the ℓ_p-norm over orbits of specific groups. As an application, we have applied GOO to point cloud data to demonstrate the use of data normalization in shape matching when objects are represented as point clouds.

Our work has demonstrated that the unified framework of GOO for data normalization is both of theoretical interests in providing a new perspective on matrix decompositions and data regularization functions, and of practical interests in modeling and elimination of distortions present in real world data. Moreover, we have used GOO to induce tensor decomposition. The unified framework of GOO for matrix decomposition and tensor decomposition allows us to bridge them. In particular, we have presented Lemma 6.24, which relates the infimum of the tensor-based GOO with the infimum of GOO of the matrix unfolded to the tensor.

As future work, it is worth pointing out that our work can be extended to Kronecker tensor decompositions, which We would like to address this extension in future.

Chapter 7

Neural Network Input Augmentation by Data Normalization

7.1 ▪ Abstract

In this paper we propose and study a technique to utilize normalized form of data to improve the prediction accuracy of neural networks. As data normalization can help reduce some degrees of freedom of the input data, the prediction accuracy of a neural network may be improved by using normalized rather than raw form of the data. To be safe against normalization errors which may propagate to be prediction errors, we also investigate a method to feed the neural network with the original data alongside the normalized forms resulting from different choices of normalization. Experiment results show that normalized form derived from special-linear transformations exhibit best accuracy of prediction on a dataset constructed from MNIST by introducing variation in poses. In particular, we can reduce the classification error from 0.0798 to 0.0617 by using special linear transformation alone, outperforming other variations.

7.2 ▪ Introduction

Artificial Neural networks, like other models in machine learning in general, depends heavily on kinds of normalization to suppress the variation in input data. The whitening transformation for example, is used to suppress the variation in value range of the input data, like the different lighting and aperture conditions of cameras reflected in data of photos. As another example, the variation in viewpoint leads to geometric distortion to the data, including rotation, translation and parallax effect. Though neural network is known to be able to cope with these variations to some extent, given sufficient training data and proper model, it still relies on normalization to improve the prediction accuracy, reduce training time and achieve generalizability[11, 75, 44, 120, 62, 121].

There are several ways to integrate data normalization into a neural network. First, it is possible to use data normalization as pre-processing. For example, [65] uses whitening transformation before doing text detection on the images.

Second, when the normalization need be done in intermediate layers of the neural network, to enable the applicability of back propagation algorithm, the normalization operation need be differentiable. For example, [64] carefully constructed the so-called Spatial Transform Network to enable differentiable geometric affine transformation of images.

In this paper, we focus on the first method of using normalization as preprocessing. We also explore the third method, which is feeding the neural network the normalized form alongside the original form. As the unavoidable errors in normalization process may introduce extra variations

in the normalized forms, we may remedy for this error by providing the original form, which in information theoretic viewpoint will ensure no loss of information. However, experiment results on a dataset transformed from MNIST suggests that use special linear normalization alone is more effective than providing the original form alongside the normalized form.

7.3 ▪ Data Normalization as Input Augmentation

However, using result of normalization as input to a neural network has the potential risk that incorrect normalization will have negative impact on the overall accuracy. We propose instead to use data normalization as a method for augmenting the input.

Assume each input instance of a neural network is an image represented by a tensor $\mathcal{T}$ and assume through some normalization induced from optimization, we obtain:

$$\hat{\mathbf{G}} = \inf_{\mathbf{G}\in\mathfrak{G}} g(\mathbf{G}\circ\mathcal{T}) \tag{7.1}$$

$$\hat{\mathcal{T}} = \hat{\mathbf{G}}\circ\mathcal{T}, \tag{7.2}$$

where g and $\mathfrak{G}$ determine what kind of degree of freedom to eliminate from $\mathcal{T}$.

Then we may be able to also use $\hat{\mathcal{T}}$ as input to the neural network so that instead of solving:

$$\inf_{\theta} f(\mathcal{T};\theta), \tag{7.3}$$

we may solve either the following optimization which only use the normalized form:

$$\inf_{\theta} f(\hat{\mathcal{T}};\theta), \tag{7.4}$$

7.3.1 ▪ Original alongside normalized forms

Alternatively we may use the normalized form alongside the original:

$$\inf_{\theta} f(\mathcal{T},\hat{\mathcal{T}};\theta). \tag{7.5}$$

As $\hat{\mathcal{T}}$ is computed independent of the neural network as a preprocessing step, the training of neural work is virtually unaffected except that the input may be augmented. We show the diagram of this method in Figure 7.1.

In case when $\mathcal{T}$ is an image represented by a tensor of order 3:

$$\mathcal{T}\in\mathbb{F}^{C\times H\times W}, \tag{7.6}$$

where C, H and W are number of channels, height and width of the image respectively. We may construct an augmented input:

$$\tilde{\mathcal{T}}\in\mathbb{F}^{2C\times H\times W}, \tag{7.7}$$

where $\tilde{\mathcal{T}}$ is produced from concatenation of $\mathcal{T}$ and $\hat{\mathcal{T}}$ along the C dimension.

7.3.2 ▪ Selecting principal components from multiple forms

As providing original forms alongside the normalized forms will increase the amount of input data, amount of computation will also be increased. One possible method is to use Singular Value Decomposition to select principal components from the input data. This can be done on the matrix $\text{fold}_2^{-1}\tilde{\mathcal{T}}$, which is unfolding of $\tilde{\mathcal{T}}$ alongside the 2nd dimension to a matrix.

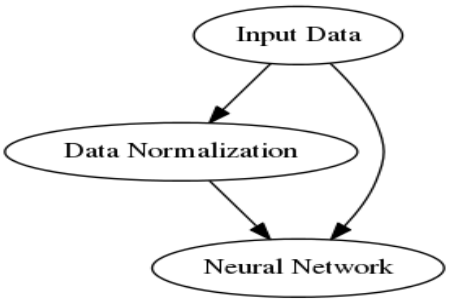


Figure 7.1. *Diagram of the structure of neural network augmented with normalized form of data.*

7.4 ▪ Empirical Evaluation of Data Normalization as Input Augmentation

We next empirically study the properties and efficacy of the method of using Data Normalization as Input Augmentation.

To make a fair comparison, for each dataset, we train a convolutional neural network with a fully-connected layer as a baseline. Then we use different variations of data as input to the same neural network. We do the experiments based on implementation of convolutional neural network in Theano[8, 7] framework.

As at the input level, the normalization of data may be done as a pre-processing step, the running times of neural networks are approximately the same, which we omit form reporting here. It would be a design choice to balance the computational complexity of the normalization process with the accuracy of predictions.

7.4.1 ▪ Spatial Normalizations by Group Orbit Optimization

We use the Group Orbit Optimization [150] to construct the spatial normalizations to be applied to the input of the network, which is done by:

$$\inf_{\mathbf{G}\in\mathfrak{G}(n)} \|\tilde{\mathbf{M}}\mathbf{G}\|_{\infty}\cdot, \tag{7.8}$$

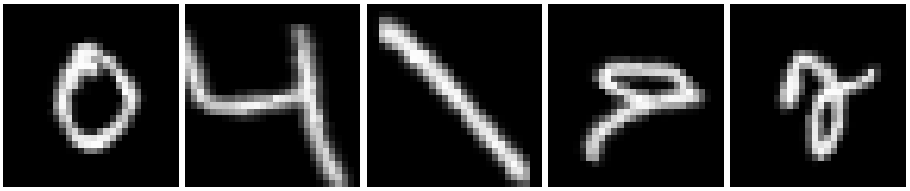
where $\mathbf{M}$ is the input data, $\tilde{\mathbf{M}}$ is $\mathbf{M}$ with mean value of rows subtracted, and $\mathfrak{G}$ is the group which corresponds to the kind degree of freedom one would like to ignore. For example, if $\mathfrak{G}$ is the $\mathfrak{SO}_2$, the special orthogonal group in two dimension, the optimized result $\tilde{\mathbf{M}}\hat{\mathbf{G}}$ will be invariant to rotational distortions of $\tilde{\mathbf{M}}$; similarly, if $\mathfrak{G}$ is $\mathfrak{SL}_2$, $\tilde{\mathbf{M}}\hat{\mathbf{G}}$ will be invariant to distortions representable as a special linear matrix, including rotation, shearing, squeezing and mirroring.

7.4.2 ▪ Transformed MNIST

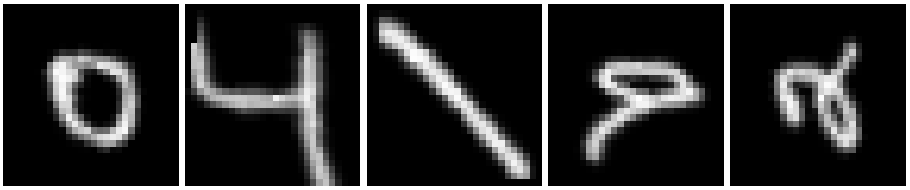
We first construct a dataset by applying affine transformations on the MNIST dataset[84], which consists of 28×28 grey scale images of handwritten digits. There are 57344 training images and 57344 test images. Samples of our dataset and their normalized forms are shown in Figure 7.2:

To demonstrate the effectiveness of data normalization, we intentionally select a neural network that is quite simple, in the hope that the normalization process will help reduce complexity of the problem to improve the prediction accuracy.

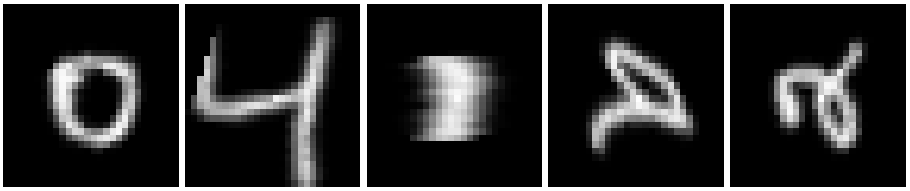
Our baseline model has one hidden layer with 512 hidden units, followed by a 10 unit softmax layer. Capped ReLU $f(x) = \min(1, \max(0, x))$, which is a variant of ReLU[98], is used as nonlinear activation function.



(a) Transformed MNIST by introducing affine transform distortions



(b) Normalized forms by rotation transforms



(c) Normalized forms by special linear transforms

Figure 7.2. *This figures visually compares the results of KPSVD and SVD approximation given the same number of parameters. For this example, it can be seen that KPSVD with right matrix shape 16x20 is considerably better than SVD in approximation.*

CNN training is done with Adam[69] with weight decay of 0.0001. Dropout[58] of 0.5 is used on all layers. Initial learning rate is $1e-4$ for Adam. Mini-batch size of 128 is used.

Test results are listed in Table 7.1. The "Baseline" row corresponds to the case when no transformation is done on the input. The "Rotation" row means that the original data has been optimized by optimization 7.8 with $\mathfrak{G} = \mathfrak{SO}_2$, which is equivalent to Principal Component Analysis. The "Special linear" row corresponds to optimization 7.8 with $\mathfrak{G} = \mathfrak{SL}_2$. "Original + Rotation" means both the normalized form w.r.t. rotations and the original data are used as input the neural network, and similarly for "Original + Special linear" and "Original + Rotation + Special linear".

It can be seen that "Special linear" achieves the smallest test error, even though its training error is larger than "Rotation" and "Baseline". It can also be seen that combinations of normalized forms of different optimizations do not necessarily increase accuracy of the prediction.

We also try using three duplicated copies of the original data as input, "Original duplicated three times" row corresponds to the case when the input data is duplicated three times, so that a tensor $\mathcal{T} \in \mathbb{F}^{C\times H\times W}$ will become $\mathcal{T} \in \mathbb{F}^{3C\times H\times W}$. The purpose of this experiment is to evaluate the influence of change of number of channels. By the result, it seems that although increasing the number of channels of input data will increase the number of parameters in the neural network, it will not necessarily increase the accuracy of prediction.

"First principal component of Original + Rotation + Special linear" row corresponds to the case when only first two components of $\text{fold}^{-1}\,\mathcal{T} \in \mathbb{F}^{3C\times HW}$ is used as input where $\mathcal{T} \in \mathbb{F}^{3C\times H\times W}$ is constructed as "Original + Rotation + Special linear". For this example, it turns out that

using SVD to extract high energy component may significantly degrade prediction accuracy.

Table 7.1. *Comparison of using different input augmentation methods on the transformed MNIST dataset.*

Methods	Error on Training set	Error on Test set
Baseline	0.00322	0.0798
Rotation	0.0153	0.0729
Special linear	0.0231	**0.0617**
Original + Rotation	0.0004	0.0761
Original + Special linear	0.0004	0.0761
Original + Rotation + Special linear	0.	0.0628
Original duplicated three times	0.	0.0790
First principal component of Original + Rotation + Special linear	0.0034	0.0832

7.5 ▪ Related Work

In this section we discuss related work not covered in previous sections.

A neural network or other learning algorithms can be used to achieve invariance against pose[151, 9, 136, 70, 3, 5], etc.

7.6 ▪ Conclusion

In this paper, we propose and study methods of using Group Orbit Optimization for Data Normalization. It is shown that given some particular degree of freedom one would like to eliminate from the input data, one may construct corresponding Group Orbit Optimization to derive a normalized form. We further explored using such normalized form for improving the accuracy of prediction of neural networks. Instead of using the normalized form directly as input to neural networks, which may potentially causes degradation when the normalization is incorrect, we augment the original input data with the normalized data to still allow for preservation of information. We also note that when the normalization operation is differentiable, it is possible to insert the normalization between any two layers in the neural network.

As future work, we note that there are many kinds of normalization that remain to be explored, including but not limited to non-rigid geometric transformation, thinning of strokes to skeleton etc.

Chapter 8

Multilinear Map Layer: Prediction Regularization by Structural Constraint

8.1 ▪ Abstract

In this paper we propose and study a technique to impose structural constraints on the output of a neural network, which can reduce amount of computation and number of parameters besides improving prediction accuracy when the output is known to approximately conform to the low-rankness prior. The technique proceeds by replacing the output layer of neural network with the so-called MLM layers, which forces the output to be the result of some Multilinear Map, like a hybrid-Kronecker-dot product or Kronecker Tensor Product. In particular, given an "autoencoder" model trained on SVHN dataset, we can construct a new model with MLM layer achieving 62% reduction in total number of parameters and reduction of ℓ_2 reconstruction error from 0.088 to 0.004. Further experiments on other autoencoder model variants trained on SVHN datasets also demonstrate the efficacy of MLM layers.

8.2 ▪ Introduction

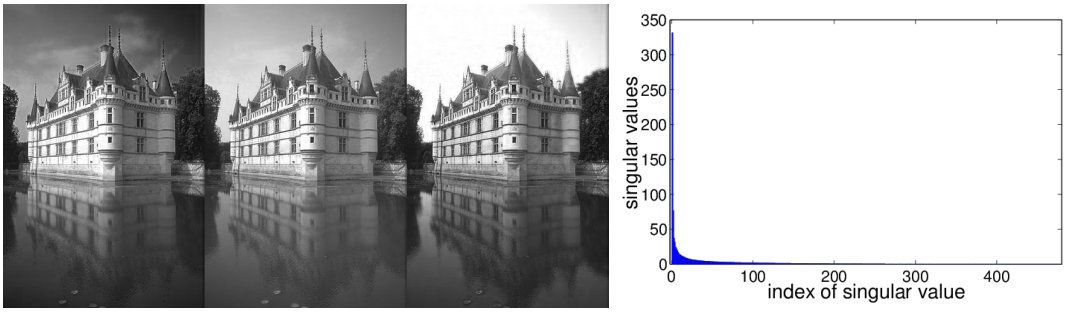
To the human eyes, images made up of random values are typically easily distinguishable from those images of real world. In terms of Bayesian statistics, a prior distribution can be constructed to describe the likelihood of an image being a natural image. An early example of such "image prior" is related to the frequency spectrum of an image, which assumes that lower-frequency components are generally more important than high-frequency components, to the extent that one can discard some high-frequency components when reducing the storage size of an image, like in JPEG standard of lossy compression of images [133].

Another family of image prior is related to the so called sparsity pattern[18, 17, 20], which refers to the phenomena that real world data can often be constructed from a handful of exemplars modulo some negligible noise. In particular, for data represented as vector $\mathbf{v} \in \mathbb{F}^{\mathbf{m}}$ that exhibits sparsity, we can construct the following approximation:

$$\mathbf{v} \approx \mathbf{V}\mathbf{x}, \tag{8.1}$$

where $\mathbf{V} \in \mathbb{F}^{m \times n}$ is referred to as dictionary for $\mathbf{v}$ and $\mathbf{x}$ is the weight vector combining rows of the dictionary to reconstruct $\mathbf{v}$. In this formulation, sparsity is reflected by the phenomena that number of non-zeros in $\mathbf{x}$ is often much less than its dimension, namely:

$$\|\mathbf{x}\|_0 \ll m. \tag{8.2}$$



(a) The unfolding of an RGB sample image. (b) The singular values

Figure 8.1. *An illustration of the singular values of the unfolding of an RBG sample image.*

The sparse representation $\mathbf{x}$ may be derived in the framework of dictionary learning[102, 94] by the following optimization:

$$\min_{\mathbf{x}} \|\mathbf{M} - \sum_i x_i \mathbf{D}_i\|_F + \lambda f(\mathbf{x}), \tag{8.3}$$

where $\mathbf{D}_i$ is a component in dictionary, f is used to induce sparsity in $\mathbf{x}$, with ℓ_1 being a possible choice.

8.2.1 ▪ Low-Rankness as Sparse Structure in Matrices

When data is represented as a matrix $\mathbf{M}$, the formulation of 8.3 is related to the rank of $\mathbf{M}$ by Theorem 6.13 in the following sense:

$$\operatorname{rank}(\mathbf{M}) = \min_{\mathbf{x}} \|\mathbf{x}\|_0 \tag{8.4}$$

$$\text{s.t. } \mathbf{M} = \sum_i x_i \mathbf{D}_i, \tag{8.5}$$

$$\mathbf{D}_i = \mathbf{u}_i \mathbf{v}_\mathbf{i}^\top,\ \mathbf{u}_\mathbf{i}^\top \mathbf{u}_\mathbf{i} = \mathbf{1},\ \mathbf{v}_\mathbf{i}^\top \mathbf{v}_\mathbf{i} = \mathbf{1}. \tag{8.6}$$

Hence a low rank matrix $\mathbf{M}$ always has a sparse representation w.r.t. some rank-1 orthogonal bases. This unified view allows us to generalize the sparsity pattern to images by requiring the underlying matrix to be low-rank.

When an image has multiple channels, it may be represented as a tensor $\mathcal{T} \in \mathbb{F}^{C \times H \times W}$ of order 3. Nevertheless, the matrix structure can still be recovered by unfolding the tensor along some dimension.

In Figure 8.1, it is shown that the unfolding of the RGB image tensor along the width dimension can be well approximated by a low-rank matrix as energy is concentrated in the first few components.

In particular, given Singular Value Decomposition of a matrix $\mathbf{M} = \mathbf{U}\mathbf{D}\mathbf{V}^*$, where $\mathbf{U}, \mathbf{V}$ are unitary matrices and $\mathbf{D}$ is a diagonal matrix with the diagonal made up of singular values of $\mathbf{M}$, a rank-k approximation of $\mathbf{M} \in \mathbb{F}^{m \times n}$ is:

$$\mathbf{M} \approx \mathbf{M}_k = \tilde{\mathbf{U}}\tilde{\mathbf{D}}\tilde{\mathbf{V}}^*,\ \tilde{\mathbf{U}} \in \mathbb{F}^{m \times k},\ \tilde{\mathbf{D}} \in \mathbb{F}^{k \times k},\ \tilde{\mathbf{V}} \in \mathbb{F}^{n \times k}, \tag{8.7}$$

where $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ are the first k-columns of the $\mathbf{U}$ and $\mathbf{V}$ respectively, and $\tilde{\mathbf{D}}$ is a diagonal matrix made up of the largest k entries of $\mathbf{D}$.

In this case approximation by SVD is optimal in the sense that the following holds [59]:

$$\mathbf{M}_r = \inf_{\mathbf{X}} \|\mathbf{X} - \mathbf{M}\|_F \text{ s.t. } \operatorname{rank}(\mathbf{X}) \leq r, \tag{8.8}$$

and

$$\mathbf{M}_r = \inf_{\mathbf{X}} \|\mathbf{X} - \mathbf{M}\|_2 \text{ s.t. } \operatorname{rank}(\mathbf{X}) \leq r. \tag{8.9}$$

An important variation of the SVD-based low rank approximation is to also model the ℓ_1 noise in the data:

$$\|\mathbf{M}\|_{RPCA} = \inf_{\mathbf{S}} \|\mathbf{M} - \mathbf{S}\|_* + \lambda \|\mathbf{S}\|_1. \tag{8.10}$$

This norm which we tentatively call "RPCA-norm" [15], is well-defined as it is infimal convolution [108] of two norms.

8.2.2 ▪ Low Rank Structure for Tensor

Above, we have used the low-rank structure of the unfolded matrix of a rank-3 tensor corresponding to RGB values of an image to reflect the low rank structure of the tensor. However, there are multiple ways to construct of matrix $\mathbf{D} \in \mathbb{R}^{m \times 3n}$, e.g., by stacking $\mathbf{R}$, $\mathbf{G}$ and $\mathbf{B}$ together horizontally. An emergent question is if we construct a matrix $\mathbf{D}' \in \mathbb{R}^{3m \times n}$ by stacking $\mathbf{R}$, $\mathbf{G}$ and $\mathbf{B}$ together vertically. Moreover, it would be interesting to know if there is a method that can exploit both forms of constructions. It turns out that we can deal with the above variation by enumerating all possible unfoldings. For example, the nuclear norm of a tensor may be defined as:

Definition 8.1. *Given a tensor $\mathcal{E}$, let $\mathbf{E}_{(i)} = \operatorname{fold}_i^{-1}(\mathcal{E})$ be unfolding of $\mathcal{E}$ to matrices. The nuclear norm of $\mathcal{E}$ is defined w.r.t. some weights β_i (satisfying $\beta_i \geq 0$) as a weighted sum of the matrix nuclear norms of unfolded matrices $\mathbf{E}_{(i)}$ as:*

$$\|\mathcal{E}\|_* = \sum_i \beta_i \|\mathbf{E}_{(i)}\|_*. \tag{8.11}$$

Consequently, minimizing the tensor nuclear norm will minimize the matrix nuclear norm of all unfoldings of tensor. Further, by adjusting the weights used in the definition of the tensor nuclear norm, we may selectively minimize the nuclear norm of some unfoldings of the tensor.

8.2.3 ▪ Stronger Sparsity for Tensor by Kronecker Product Low Rank

When an image is taken as matrix, the basis for low rank factorizations are outer products of row and column vectors. However, if data exhibits Principle of Locality, then only adjacent rows and columns can be meaningfully related, meaning the rank in this decomposition will not be too low. In contrast, local patches may be used as basis for low rank structure [35, 138, 113, 31, 140]. Further, patches may be grouped before assumed to be of low-rank [13, 60, 76].

A simple method to exploit the low-rank structure of the image patches is based on Kronecker Product SVD of matrix $\mathbf{M} \in \mathbb{F}^{m \times n}$:

$$\mathbf{M} = \sum_{i=1}^{\operatorname{rank}(\mathscr{R}(\mathbf{M}))} \sigma_i \mathbf{U}_i \otimes \mathbf{V}_i, \tag{8.12}$$

(a) Original image selected from BSD500 dataset[4]

(b) SVD approximations with rank 1, 2, 5, 10, 20 respectively from left to right

(c) KPSVD approximations with right matrix having shape 16x20 and with rank 1, 2, 5, 10, 20 respectively from left to right

Figure 8.2. *This figures visually compares the results of KPSVD and SVD approximation given the same number of parameters. For this example, it can be seen that KPSVD with right matrix shape 16x20 is considerably better than SVD in approximation.*

where $\mathscr{R}(\mathbf{A})$ is the operator defined in [129, 128] which shuffles indices.

$$\mathscr{R}(\mathbf{A} \otimes \mathbf{B}) = \operatorname{vec} \mathbf{A} (\operatorname{vec} \mathbf{B})^\top. \tag{8.13}$$

Note that outer product is a special case of Kronecker product when $\mathbf{A} \in \mathbb{F}^{m \times 1}$ and $\mathbf{B} \in \mathbb{F}^{1 \times n}$, hence we have SVD as a special case of KPSVD. The choice of shapes of $\mathbf{A}$ and $\mathbf{B}$, however, affects the extent to which the underlying sparsity assumption is valid. Below we give an empirical comparison of a KPSVD with SVD. The image is of width 480 and height 320, and we approximate the image with KPSVD and SVD respectively for different ranks. To make the results comparable, we let $\mathbf{B} \in \mathbb{F}^{16x20}$ to make the number of parameters in two approach equal.

We may extend Kronecker product to tensors as Kronecker Tensor Product as:

Definition 8.2. *Kronecker tensor product[103, 104] is defined for two tensors of the same order k. I.e., for two tensors:*

$$\mathcal{A} \in \mathbb{F}^{m_1 \times m_2, \cdots, \times m_k}, \tag{8.14}$$

and

$$\mathcal{B} \in \mathbb{F}^{n_1 \times n_2, \cdots, \times n_k}, \tag{8.15}$$

we define Kronecker product of tensor as

$$(\mathcal{A} \otimes \mathcal{B})_{i_1, i_2, \cdots, i_k} \coloneqq A_{\lfloor i_1/m_1 \rfloor, \lfloor i_2/m_2 \rfloor, \cdots, \lfloor i_k/m_k \rfloor} B_{i_1 \bmod m_1, i_2 \bmod m_2, \cdots, i_k \bmod m_k} \tag{8.16}$$

where

$$\mathcal{A} \otimes \mathcal{B} \in \mathbb{F}^{m_1 n_1 \times m_2 n_2, \cdots, \times m_k n_k}. \tag{8.17}$$

With the help of Kronecker Tensor Product, [103, 104] is able to extend KPSVD to tensors as:

$$\mathcal{T} = \sum_{i=1}^{\text{rank}(\mathscr{R}(\mathcal{T}))} \sigma_i \mathcal{U}_i \otimes \mathcal{V}_i, \tag{8.18}$$

where $\mathscr{R}(\mathcal{T})$ is a matrix.

8.3 ▪ Prediction Regularization by Structural Constraint

In a neural network like

$$\inf_{\theta} d(\mathbf{Y}, f(\mathbf{X}; \theta)), \tag{8.19}$$

If $f(\mathbf{X}; \theta)$, the prediction of the network, is known to be like an image, it is desirable to use the image priors to help improve the prediction quality. An example of such a neural network is the so-called "Autoencoder" [131, 26, 132, 101], which when presented with a possibly corrupted image, will output a reconstructed image that possibly have the noise suppressed.

One method to exploit the prior information is to introduce an extra cost term to regularize the output as

$$\inf_{\theta} d(\mathbf{Y}, \mathbf{Z}) + \lambda r(\mathbf{Z}) \text{ s.t. } \mathbf{Z} = f(\mathbf{X}; \theta), \tag{8.20}$$

The regularization technique is well studied in the matrix and tensor completion literature [90, 123, 41, 119, 73, 6]. For example, nuclear norm $\|\mathbf{Z}\|_*$, which is sum of singular values, or logarithm of determinant $\log(\det(\epsilon \mathbf{I} + \mathbf{Z}\mathbf{Z}^\top))$ [39], may be used to induce low-rank structure if $\mathbf{Z}$ is a matrix. It is even possible to use RPCA-norm to better handle the possible sparse non-low-rank components in $\mathbf{Z}$ by letting $r(\mathbf{Z}) = \|\mathbf{Z}\|_{RPCA}$.

However, using extra regularization terms also involve a few subtleties:

1. The training of neural network incurs extra cost of computing the regularizer terms, which slows down training. This impact is exacerbated if the regularization terms cannot be efficiently computed in batch, like when using the nuclear norm or RPCA-norm regularizers together with the popular mini-batch based Stochastic Gradient Descent training algorithm [10].

2. The value of λ is application-specific and may only be found through grid search on validation set. For example, when r reflects low-rankness of the prediction, larger value of λ may induce result of lower-rank, but may cause degradation of the reconstruction quality.

We take an alternative method by directly restricting the parameter space of the output. Assume in the original neural network, the output is given by a Fully-Connected (FC) layer as:

$$\mathbf{L}_a = h(\mathbf{L}_{a-1}\mathbf{M}_a + \mathbf{b}_a). \tag{8.21}$$

It can be seen that the output $\mathbf{L}_a \in \mathbb{F}^{m\times n}$ has mn number of free parameters. However, noting that the product of two matrices $\mathbf{A} \in \mathbb{F}^{m\times r}$ and $\mathbf{B} \in \mathbb{F}^{r\times n}$ will have the property

$$\operatorname{rank}(\mathbf{AB}) \leq r, \tag{8.22}$$

when $m \geq r$ and $n \geq r$.

Hence we may enforce that $\operatorname{rank}(\mathbf{L}_a) \leq r$ by the following construct for example:

$$\mathbf{L}_a = h(\mathbf{L}_{a-1}\mathbf{M}_a + \mathbf{b}_a)h(\mathbf{L}_{a-1}\mathbf{N}_a + \mathbf{c}_a), \tag{8.23}$$

when we have:

$$\mathbf{L}_{a-1}\mathbf{M}_a + \mathbf{b}_a \in \mathbb{F}^{m\times r} \tag{8.24}$$

$$\mathbf{L}_{a-1}\mathbf{N}_a + \mathbf{c}_a \in \mathbb{F}^{r\times n}. \tag{8.25}$$

In a Convolutional Neural Network where intermediate data are represented as tensors, we may enforce the low-rank image prior similarly. In fact, by proposing a new kind of output layer to explicitly encode the low-rankness of the output based on some kinds of Multilinear Map, like a hybrid of Kronecker and dot product, or Kronecker tensor product, we are able to increase the quality of denoising and reduce amount of computation at the same time. We outline the two formulations below.

Assume each output instance of a neural network is an image represented by a tensor of order 3:

$$\mathcal{T} \in \mathbb{F}^{C\times H\times W}, \tag{8.26}$$

where C, H and W are number of channels, height and width of the image respectively.

8.3.1 ▪ KTP layer

In KTP layers, we approximate $\mathcal{T}$ by Kronecker Tensor Product of two tensors.

$$\mathcal{T} \approx \mathcal{A} \otimes \mathcal{B}. \tag{8.27}$$

As by applying the shuffle operator defined in [129, 128], 8.28 is equivalent to:

$$\mathscr{R}(\mathcal{T}) \approx \operatorname{vec}\mathcal{A} \operatorname{vec}\mathcal{B}^\top, \tag{8.28}$$

hence we are effectively doing rank-1 approximation of the matrix $\mathscr{R}(\mathcal{T})$. A natural extension would then be to increase the number of components in approximation as:

$$\mathcal{T} \approx \sum_{i=1}^{K} \mathcal{A}_i \otimes \mathcal{B}_i. \tag{8.29}$$

We may further combine the multiple shape formulation of 8.29 to get the general form of KTP layer:

$$\mathcal{T} \approx \sum_{j=1}^{J}\sum_{i=1}^{K} \mathcal{A}_{ij} \otimes \mathcal{B}_{ij}. \tag{8.30}$$

where $\{\mathcal{A}_{ij}\}_{i=1}^{K}$ and $\{\mathcal{B}_{i,j}\}_{i=1}^{K}$ are of the same shape respectively.

8.3.2 ▪ HKD layer

In HKD layers, we approximate $\mathcal{T}$ by the following multilinear map between $\mathcal{A} \in \mathbb{F}^{C_1 \times H_1 \times W_1}$ and $\mathcal{B} \in \mathbb{F}^{C_1 \times C_2 \times H_2 \times W_2}$, which is a hybrid of Kronecker product and dot product:

$$T_{c,h,w} \approx \tilde{T}_{c,h,w} = \tilde{T}_{c,h_1+H_1*h_2,w_1+W_1*w_2} = \sum_{c_1} A_{c_1,h_2,w_2} B_{c,c_1,h_1,w_1}, \tag{8.31}$$

where $h = h_1 h_2$, $w = w_1 w_2$.

The rationale behind this construction is that the Kronecker product along the spatial dimension of H and W may capture the spatial regularity of the output, which enforces low-rankness; while the dot product along C would allow combination of information from multiple channels of $\mathcal{A}$ and $\mathcal{B}$.

In the framework of low-rank approximation, the formulation 8.31 is by no means unique. One could for example improve the precision of approximation by introducing multiple components as:

$$T_{c,h,w} = T_{c,h_1+H_1*h_2,w_1+W_1*w_2} \approx \sum_{k} \sum_{c_1} A_{k,c_1,h_2,w_2} B_{k,c,c_1,h_1,w_1}. \tag{8.32}$$

Hereafter we would refer to layers constructed following 8.31 as HKD layers. The general name of MLM layers will refer to all possible kinds of layers that can be constructed from other kinds of multilinear map.

8.3.3 ▪ General MLM layer

A Multilinear map is a function of several variables that is linear separately in each variable as:

$$f \colon V_1 \times \cdots \times V_n \to W, \tag{8.33}$$

where $V_1, \ldots, V_n$ and W are vector spaces with the following property: for each i, if all of the variables but v_i are held constant, then $f(v_1, \ldots, v_n)$ is a linear function of v_i[77]. It is easy to verify that Kronecker product, convolution, dot product are all special cases of multilinear map.

Figure 8.3 gives a schematic view of general structure of the MLM layer. The left factor and right factor are produced from the same input, which are later combined by the multilinear map to produce the output. Depending on the type of multilinear map used, the MLM layer will become HKD layer or KTP layer. We note that it is also possible to introduce more factors than two into an MLM layer.

8.4 ▪ Empirical Evaluation of Multilinear Map Layer

We next empirically study the properties and efficacy of the Multilinear Map layers and compare it with the case when no structural constraint is imposed on the output.

To make a fair comparison, we first train a covolutional autoencoder with the output layer being a fully-connected layer as a baseline. Then we replace the fully-connected layer with different kinds of MLM layers and train the new network until quality metrics stabilizes. For example, Figure 8.4 gives a subjective comparison of HKD layer with the original model on SVHN dataset. We then compare MLM layer method with the baseline model in terms of number of parameters and prediction quality. We do the experiments based on implementation of MLM layers in Theano[8, 7] framework.

Table 8.4 shows the performance of MLM layers on training an autoencoder for SVHN digit reconstruction. The network first transforms the 40×40 input image to a bottleneck feature

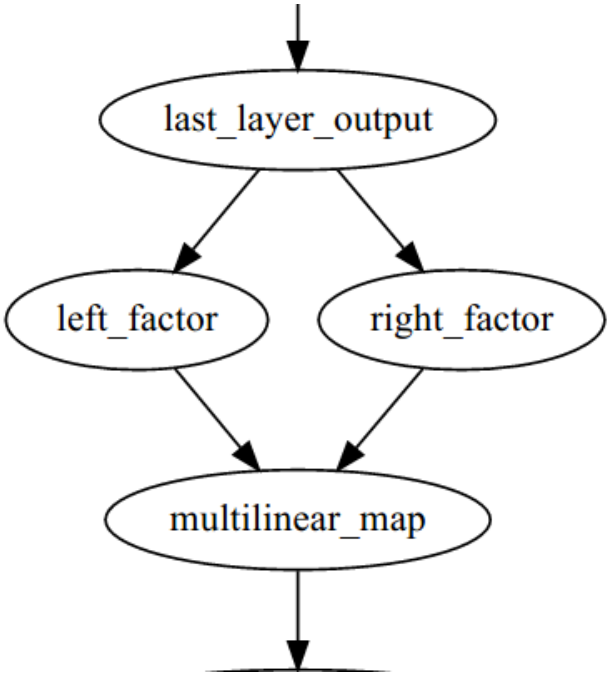
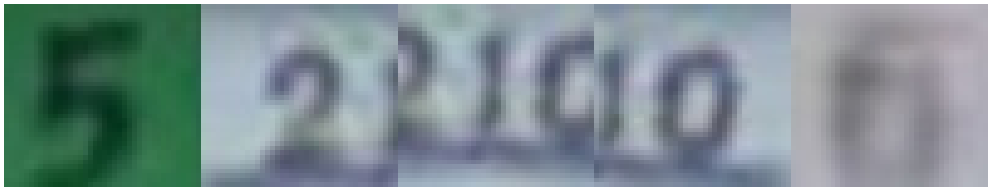


Figure 8.3. *Diagram of the structure of a MLM layer. the output of the preceding layer is fed into two nodes where left factor tensor and right factor tensor are computed. Then a multilinear map is applied on the left and right factors to construct the output.*

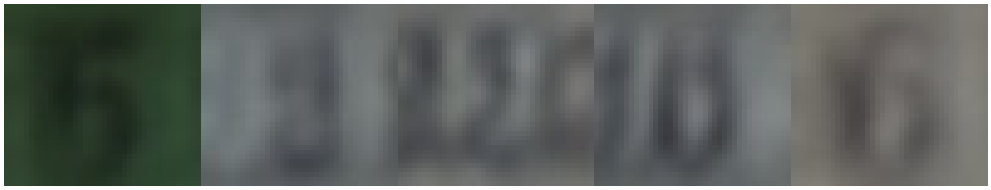
Table 8.1. *Evaluation of MLM layers on SVHN digit reconstruction*

model	bottleneck #hidden unit	total #params	layer #params	L2 error
conv + FC	512	13.33M	5764800	3.4e-2
conv + HKD	512	4.97M	46788	5.2e-4
conv + HKD multiple components	512	5.05M	118642	3.6e-4
conv + FC	64	13.40M	5764800	1.3e-1
conv+ HKD	64	5.04M	46788	1.8e-3
conv + KTP	64	5.04M	46618	3.2e-3
conv	16	5.09M	0	4.0e-2
conv + FC	16	13.36M	5764800	8.8e-2
conv + HKD	16	5.00M	46788	3.8e-3
conv + KTP	16	5.00M	46618	5.8e-3

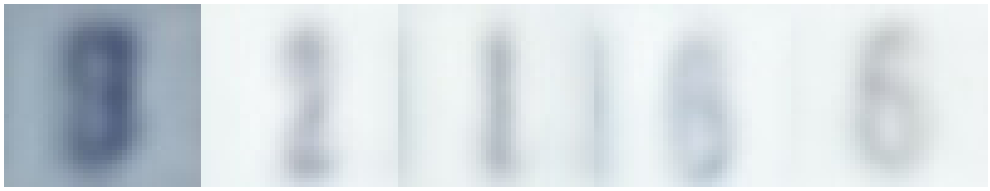
vector through a traditional ConvNet consisting of 4 convolutional layers, 3 max pooling layers and 1 fully connected layer. Then, the feature is transformed again to reconstruct the image through 4 convolutional layers, 3 un-pooling layers and the last fully-connected layer or its alternatives as output layer. The un-pooling operation is implemented with the same approach used in [32], by simply setting the pooled value at the top-left corner of the pooled region, and leaving other as zero. The fourth column of the table is the number of parameters in fully-connected layer or its alternative MLM layer. By varying the length of bottleneck feature and



(a) Original image patches selected from SVHN dataset



(b) Results of an Autoencoder A1



(c) Results of an Autoencoder A2 with less neurons in the bottleneck layer than A1

(d) Results of an Autoencoder with as many neurons in the bottleneck layer as A2 but uses HKD as output layer

Figure 8.4. *This figures show the results of passing five cropped patched from SVHN dataset as input images through Autoencoders with different output layers. The first row contains the original images. The second row contains the output of an Autoencoder "A1". The third row contains the output of an Autoencoder "A2", which has smaller number of hidden units in the bottleneck layer. The fourth row contains the output of an Autoencoder "A3" constructed from "A2" by replacing the output FC layer with a HKD layer. It can be seen that "A3", though with smaller number of hidden units in bottleneck layer, visually performs better than "A1" and "A2" in reconstruting the input images.*

using different alternatives for FC layer, we can observe that both HKD and KTP layer are good alternatives for FC layer as output layer, and they also both significantly reduce the number of parameters. We also tested the case with convolutional layer as output layer, and the result still shows the efficacy of MLM layer.

As the running time may depend on particular implementation details of the KFC and the Theano framework, we do not report running time. However, the complexity analysis suggests that there should be significant reduction in amount of computation.

8.5 ▪ Related Work

In this section we discuss related work not covered in previous sections.

Low rank approximation has been a standard tool in restricting the space of the parameters. Its application in linear regression dates back to [2]. In [112, 86, 137, 147, 28], low rank approximation of fully-connected layer is used; and [66, 107, 82, 81, 27] also considered low rank approximation of convolution layer. [146] considered approximation of multiple layers with nonlinear activations. To our best knowledge, these methods only consider applying low-rank approximation to weights of the neural network, but not to the output of the neural network.

As structure is a general term, there are also other types of structure that exist in the desired prediction. Neural network with structured prediction also exist for tasks other than autoencoder, like edge detection[29], image segmentation[149, 38], super resolution[30], image generation[32]. The structure in these problem may also exhibit low-rank structure exploitable by MLM layers.

8.6 ▪ Conclusion and Future Work

In this paper, we propose and study methods for incorporating the low-rank image prior to the predictions of the neural network. Instead of using regularization terms in the objective function of the neural network, we directly encode the low-rank constraints as structural constraints by requiring the output of the network to be the result of some kinds of multilinear map. We consider a few variants of multilinear map, including a hybrid-Kronecker-dot product and Kronecker tensor product. We have found that using the MLM layer can significantly reduce the number of parameters and amount of computation for autoencoders on SVHN.

As future work, we note that when using ℓ_1 norm as objective together with the structural constraint, we could effectively use the norm defined in Robust Principal Value Analysis as our objective, which would be able to handle the sparse noise that may otherwise degrade the low-rankness property of the predictions. In addition, it would be interesting to investigate applying the structural constraints outlined in this paper to the output of intermediate layers of neural networks.

Chapter 9

Conclusions and Future Work

This thesis aims at improving the state-of-art in applying matrix techniques to several problems in neural network tasks: model approximation, data normalization and prediction regularization.

For model approximation, we have introduced KFC layers as replacement for Fully-Connected layers, achieving both reduction of amount of computation and number of parameters at the same time within a mild degradation of prediction quality. We also study the influence of number components in Kronecker product, shape of factor matrices to the efficacy of KFC layers. For data normalization, we have presented the frame of Group Orbit Optimization in which we are able to give a unified view of several matrix decompositions and matrix norms. Based on the framework, we can construct matrix functions that we apply to augment input data to neural networks in order to improve the prediction accuracy of the latter.

Lastly, we apply low-rank regularization to the prediction of neural network. Instead of adding a regularization term in the objective function, we enforce the low-rank constraint by explicitly requiring the output to be result to be a hybrid of Kronecker product and dot product. Compared to the objective regularization approach, our method reduces amount of computation and number of parameters, besides reducing reconstruction error.

In the future, we would like to investigate the use of the matrix approximation and decomposition techniques to more components of neural network. For example, it would be of interest to investigate if we can use Kronecker product to approximate the convolution layers and the transition matrices in Recurrent Neural Networks like GRU and LSTM. The data normalization approach to input augmenting may be extended to handle input variation beyond geometric distortions. Lastly, for prediction regularization, more forms of structural constraints on the predictions of neural networks may be explored.

Acknowledgments

I would like to thank my supervisor Professor Xiaobing Feng for his continuous guidance and support in my research work and PhD study. I would also like to thank my co-author and mentor Professor Zhihua Zhang and Professor Edward Chang for leading me into the exciting area of machine learning and inspiring me on how to do research work. The other co-authors I would to thank include Shusen Wang, Yifan Pi, Fangtao Li, Haoruo Peng, Xiance Si, Zhengyu Wang, Yang Gao and Decheng Dai, Jianan Wu, Yuxin Wu, Cong Yao, Xinyu Zhou for all the helps and discussions in writing the papers. Over the years of my PhD time, I have worked at Chinese Academy of Sciences, Google Inc. and Megvii Inc. and I would like to thank all colleagues that I have worked with. Most of all, I would like to thank my family, my kids and my wife Lin for their encouragement, support, understanding and love.

Bibliography

[1] P-A Absil, Robert Mahony, and Rodolphe Sepulchre, *Optimization algorithms on matrix manifolds*, Princeton University Press, 2009.

[2] Theodore Wilbur Anderson, *Estimating linear restrictions on regression coefficients for multivariate normal distributions*, The Annals of Mathematical Statistics, (1951), pp. 327–351.

[3] Shervin Rahimzadeh Arashloo and Josef Kittler, *Energy normalization for pose-invariant face recognition based on mrf model image matching*, Pattern Analysis and Machine Intelligence, IEEE Transactions on, 33 (2011), pp. 1274–1280.

[4] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik, *Contour detection and hierarchical image segmentation*, IEEE Trans. Pattern Anal. Mach. Intell., 33 (2011), pp. 898–916.

[5] Akshay Asthana, Tim K Marks, Michael J Jones, Kinh H Tieu, and M Rohith, *Fully automatic pose-invariant face recognition via 3d pose normalization*, in Computer Vision (ICCV), 2011 IEEE International Conference on, IEEE, 2011, pp. 937–944.

[6] Francis Bach, Rodolphe Jenatton, Julien Mairal, and Guillaume Obozinski, *Optimization with sparsity-inducing penalties*, Found. Trends Mach. Learn., 4 (2012), pp. 1–106.

[7] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio, *Theano: new features and speed improvements.* Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[8] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio, *Theano: a CPU and GPU math expression compiler*, in Proceedings of the Python for Scientific Computing Conference (SciPy), June 2010. Oral Presentation.

[9] Volker Blanz and Thomas Vetter, *Face recognition based on fitting a 3d morphable model*, Pattern Analysis and Machine Intelligence, IEEE Transactions on, 25 (2003), pp. 1063–1074.

[10] Léon Bottou, *Large-scale machine learning with stochastic gradient descent*, in Proceedings of COMPSTAT'2010, Springer, 2010, pp. 177–186.

[11] Nuala Brady and David J Field, *Local contrast in natural images: normalisation and coding efficiency*, PERCEPTION-LONDON-, 29 (2000), pp. 1041–1056.

[12] Thomas M Breuel, Adnan Ul-Hasan, Mayce Ali Al-Azawi, and Faisal Shafait, *High-performance ocr for printed english and fraktur using lstm networks*, in Document Analysis and Recognition (ICDAR), 2013 12th International Conference on, IEEE, 2013, pp. 683–687.

[13] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel, *Image denoising by non-local averaging.*, in ICASSP (2), 2005, pp. 25–28.

[14] Peter Butkovic, *Max-linear systems: theory and algorithms*, Springer New York, 2010.

[15] Emmanuel Candès, Xiaodong Li, Yi Ma, and John Wright, *Robust principal component analysis*, Journal of the ACM, 58 (2011).

[16] E.J. Candès and B. Recht, *Exact matrix completion via convex optimization*, Foundations of Computational Mathematics, 9 (2009), pp. 717–772.

[17] Emmanuel J Candès, Justin Romberg, and Terence Tao, *Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information*, Information Theory, IEEE Transactions on, 52 (2006), pp. 489–509.

[18] Emmanuel J Candes, Justin K Romberg, and Terence Tao, *Stable signal recovery from incomplete and inaccurate measurements*, Communications on pure and applied mathematics, 59 (2006), pp. 1207–1223.

[19] Emmanuel J Candès and Terence Tao, *The power of convex relaxation: Near-optimal matrix completion*, Information Theory, IEEE Transactions on, 56 (2010), pp. 2053–2080.

[20] Emmanuel J Candes, Michael B Wakin, and Stephen P Boyd, *Enhancing sparsity by reweighted âĎŞ 1 minimization*, Journal of Fourier analysis and applications, 14 (2008), pp. 877–905.

[21] Wenlin Chen, James T Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen, *Compressing convolutional neural networks*, arXiv preprint arXiv:1506.04449, (2015).

[22] ——, *Compressing neural networks with the hashing trick*, arXiv preprint arXiv:1504.04788, (2015).

[23] Moody T Chu and Larry K Norris, *Isospectral flows and abstract matrix factorizations*, SIAM journal on numerical analysis, 25 (1988), pp. 1383–1391.

[24] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet, *Torch7: A matlab-like environment for machine learning*, in BigLearn, NIPS Workshop, no. EPFL-CONF-192376, 2011.

[25] Lieven De Lathauwer, *Decompositions of a higher-order tensor in block terms—part ii: Definitions and uniqueness*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 1033–1066.

[26] Li Deng, Michael L Seltzer, Dong Yu, Alex Acero, Abdel-rahman Mohamed, and Geoffrey E Hinton, *Binary coding of speech spectrograms using a deep auto-encoder.*, in Interspeech, Citeseer, 2010, pp. 1692–1695.

[27] Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al., *Predicting parameters in deep learning*, in Advances in Neural Information Processing Systems, 2013, pp. 2148–2156.

[28] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus, *Exploiting linear structure within convolutional networks for efficient evaluation*, in Advances in Neural Information Processing Systems, 2014, pp. 1269–1277.

[29] Piotr Dollár and C Lawrence Zitnick, *Structured forests for fast edge detection*, in Computer Vision (ICCV), 2013 IEEE International Conference on, IEEE, 2013, pp. 1841–1848.

[30] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang, *Image super-resolution using deep convolutional networks*, arXiv preprint arXiv:1501.00092, (2014).

[31] Weisheng Dong, Guangming Shi, Xin Li, Yi Ma, and Feng Huang, *Compressive sensing via nonlocal low-rank regularization*, Image Processing, IEEE Transactions on, 23 (2014), pp. 3618–3632.

[32] Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox, *Learning to generate chairs with convolutional neural networks*, arXiv preprint arXiv:1411.5928, (2014).

[33] Ian L Dryden and Kanti V Mardia, *Statistical shape analysis*, vol. 4, John Wiley & Sons New York, 1998.

[34] Alan Edelman, Tomás A Arias, and Steven T Smith, *The geometry of algorithms with orthogonality constraints*, SIAM journal on Matrix Analysis and Applications, 20 (1998), pp. 303–353.

[35] Michael Elad and Michal Aharon, *Image denoising via sparse and redundant representations over learned dictionaries*, Image Processing, IEEE Transactions on, 15 (2006), pp. 3736–3745.

[36] Haoqiang Fan, Zhimin Cao, Yuning Jiang, Qi Yin, and Chinchilla Doudou, *Learning deep face representation*, arXiv preprint arXiv:1403.2802, (2014).

[37] Yuchen Fan, Yao Qian, Fenglong Xie, and Frank K Soong, *Tts synthesis with bidirectional lstm based recurrent neural networks*, in Proc. Interspeech, 2014.

[38] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun, *Learning hierarchical features for scene labeling*, Pattern Analysis and Machine Intelligence, IEEE Transactions on, 35 (2013), pp. 1915–1929.

[39] Maryam Fazel, Haitham Hindi, and Stephen P Boyd, *Log-det heuristic for matrix rank minimization with applications to hankel and euclidean distance matrices*, in American Control Conference, 2003. Proceedings of the 2003, vol. 3, IEEE, 2003, pp. 2156–2162.

[40] Shmuel Friedland, Volker Mehrmann, Renato Pajarola, and SK Suter, *On best rank one approximation of tensors*, Numerical Linear Algebra with Applications, 20 (2013), pp. 942–955.

[41] Silvia Gandy, Benjamin Recht, and Isao Yamada, *Tensor completion and low-n-rank tensor recovery via convex optimization*, Inverse Problems, 27 (2011), p. 025010.

[42] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, in Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, IEEE, 2014, pp. 580–587.

[43] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev, *Compressing deep convolutional networks using vector quantization*, arXiv preprint arXiv:1412.6115, (2014).

[44] Ian J Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet, *Multi-digit number recognition from street view imagery using deep convolutional neural networks*, arXiv preprint arXiv:1312.6082, (2013).

[45] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio, *Maxout networks*, arXiv preprint arXiv:1302.4389, (2013).

[46] Alex Graves, *Generating sequences with recurrent neural networks*, arXiv preprint arXiv:1308.0850, (2013).

[47] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber, *Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks*, in Proceedings of the 23rd international conference on Machine learning, ACM, 2006, pp. 369–376.

[48] Alex Graves, A-R Mohamed, and Geoffrey Hinton, *Speech recognition with deep recurrent neural networks*, in Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on, IEEE, 2013, pp. 6645–6649.

[49] Alex Graves and Jürgen Schmidhuber, *Offline handwriting recognition with multidimensional recurrent neural networks*, in Advances in Neural Information Processing Systems, 2009, pp. 545–552.

[50] Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra, *Draw: A recurrent neural network for image generation*, arXiv preprint arXiv:1502.04623, (2015).

[51] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan, *Deep learning with limited numerical precision*, arXiv preprint arXiv:1502.02551, (2015).

[52] Richard A Harshman, *Foundations of the parafac procedure: models and conditions for an" explanatory" multimodal factor analysis*, (1970).

[53] Richard Hartley and Andrew Zisserman, *Multiple view geometry in computer vision*, Cambridge university press, 2003.

[54] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Spatial pyramid pooling in deep convolutional networks for visual recognition*, arXiv preprint arXiv:1406.4729, (2014).

[55] Uwe Helmke, John B Moore, and Würzburg Germany, *Optimization and dynamical systems*, Citeseer, 1994.

[56] G. Hinton and S. Roweis, *Stochastic neighbor embedding*, in Advances in Neural Information Processing Systems 15.

[57] Geoffrey E Hinton and Ruslan R Salakhutdinov, *Reducing the dimensionality of data with neural networks*, Science, 313 (2006), pp. 504–507.

[58] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov, *Improving neural networks by preventing co-adaptation of feature detectors*, arXiv preprint arXiv:1207.0580, (2012).

[59] Roger A Horn and Charles R Johnson, *Topics in matrix analysis*, Cambridge university press, 1991.

[60] Haijuan Hu, Jacques Froment, and Quansheng Liu, *Patch-based low-rank minimization for image denoising*, arXiv preprint arXiv:1506.08353, (2015).

[61] Yao Hu, Debing Zhang, Jieping Ye, Xuelong Li, and Xiaofei He, *Fast and accurate matrix completion via truncated nuclear norm regularization*, Pattern Analysis and Machine Intelligence, IEEE Transactions on, 35 (2013), pp. 2117–2130.

[62] Sergey Ioffe and Christian Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, arXiv preprint arXiv:1502.03167, (2015).

[63] Mariya Ishteva, Lieven De Lathauwer, P Absil, and Sabine Van Huffel, *Dimensionality reduction for higher-order tensors: algorithms and applications*, International Journal of Pure and Applied Mathematics, 42 (2008), p. 337.

[64] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu, *Spatial transformer networks*, arXiv preprint arXiv:1506.02025, (2015).

[65] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman, *Deep features for text spotting*, in Computer Vision–ECCV 2014, Springer, 2014, pp. 512–528.

[66] ———, *Speeding up convolutional neural networks with low rank expansions*, arXiv preprint arXiv:1405.3866, (2014).

[67] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell, *Caffe: Convolutional architecture for fast feature embedding*, in Proceedings of the ACM International Conference on Multimedia, ACM, 2014, pp. 675–678.

[68] JD Kalbfleisch and Jerald F Lawless, *The analysis of panel data under a markov assumption*, Journal of the American Statistical Association, 80 (1985), pp. 863–871.

[69] Diederik Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980, (2014).

[70] Jan Kodovskỳ and Jessica Fridrich, *Steganalysis in high dimensions: Fusing classifiers built on random subspaces*, in IS&T/SPIE Electronic Imaging, International Society for Optics and Photonics, 2011, pp. 78800L–78800L.

[71] Tamara G. Kolda, *Orthogonal tensor decompositions*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 243–255.

[72] Tamara G. Kolda and Brett W. Bader, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500.

[73] Daniel Kressner, Michael Steinlechner, and Bart Vandereycken, *Low-rank tensor completion by riemannian optimization*, BIT Numerical Mathematics, (2013), pp. 1–22.

[74] Akshay Krishnamurthy and Aarti Singh, *Low-rank matrix and tensor completion via adaptive sampling*, in Advances in Neural Information Processing Systems, 2013, pp. 836–844.

[75] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, *Imagenet classification with deep convolutional neural networks*, in Advances in neural information processing systems, 2012, pp. 1097–1105.

[76] Quanming Yao James T Kwok, *Colorization by patch-based local low-rank matrix completion*, (2015).

[77] Serge Lang, *Algebra revised third edition*, GRADUATE TEXTS IN MATHEMATICS-NEW YORK-, (2002).

[78] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle, *A multilinear singular value decomposition*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1253–1278.

[79] ———, *On the best rank-1 and rank-(r1,r2,. . .,rn) approximation of higher-order tensors*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1324–1342.

[80] ———, *Computation of the canonical decomposition by means of a simultaneous generalized schur decomposition*, SIAM J. Matrix Anal. Appl., 26 (2005), pp. 295–327.

[81] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky, *Speeding-up convolutional neural networks using fine-tuned cp-decomposition*, arXiv preprint arXiv:1412.6553, (2014).

[82] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan V. Oseledets, and Victor S. Lempitsky, *Speeding-up convolutional neural networks using fine-tuned cp-decomposition*, CoRR, abs/1412.6553 (2014).

[83] Yann LeCun and Yoshua Bengio, *Convolutional networks for images, speech, and time series*, The handbook of brain theory and neural networks, 3361 (1995), p. 310.

[84] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[85] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu, *Deeply-supervised nets*, arXiv preprint arXiv:1409.5185, (2014).

[86] Hank Liao, Erik McDermott, and Andrew Senior, *Large scale deep neural network acoustic modeling with semi-supervised training data for youtube video transcription*, in Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on, IEEE, 2013, pp. 368–373.

[87] Min Lin, Qiang Chen, and Shuicheng Yan, *Network in network*, arXiv preprint arXiv:1312.4400, (2013).

[88] Cheng-Lin Liu, Fei Yin, Da-Han Wang, and Qiu-Feng Wang, *Online and offline handwritten chinese character recognition: benchmarking on new databases*, Pattern Recognition, 46 (2013), pp. 155–162.

[89] Guangcan Liu, Zhouchen Lin, Shuicheng Yan, Ju Sun, Yong Yu, and Yi Ma, *Robust recovery of subspace structures by low-rank representation*, Pattern Analysis and Machine Intelligence, IEEE Transactions on, 35 (2013), pp. 171–184.

[90] Ji Liu, Przemyslaw Musialski, Peter Wonka, and Jieping Ye, *Tensor completion for estimating missing values in visual data*, IEEE Trans. Pattern Anal. Mach. Intell., 35 (2013), pp. 208–220.

[91] Canyi Lu, Changbo Zhu, Chunyan Xu, Shuicheng Yan, and Zhouchen Lin, *Generalized singular value thresholding*, arXiv preprint arXiv:1412.2231, (2014).

[92] Helmut Lutkepohl, *Handbook of matrices.*, Computational Statistics and Data Analysis, 2 (1997), p. 243.

[93] Jan R Magnus and Heinz Neudecker, *Matrix differential calculus with applications in statistics and econometrics*, (1995).

[94] Julien Mairal, Jean Ponce, Guillermo Sapiro, Andrew Zisserman, and Francis R Bach, *Supervised dictionary learning*, in Advances in neural information processing systems, 2009, pp. 1033–1040.

[95] Peter McCullagh, John A Nelder, and P McCullagh, *Generalized linear models*, vol. 2, Chapman and Hall London, 1989.

[96] Yajie Miao, Florian Metze, and Shourabh Rawat, *Deep maxout networks for low-resource speech recognition*, in Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on, IEEE, 2013, pp. 398–403.

[97] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller, *Playing atari with deep reinforcement learning*, arXiv preprint arXiv:1312.5602, (2013).

[98] Vinod Nair and Geoffrey E Hinton, *Rectified linear units improve restricted boltzmann machines*, in Proceedings of the 27th International Conference on Machine Learning (ICML-10), 2010, pp. 807–814.

[99] John A Nelder and Roger Mead, *A simplex method for function minimization*, The computer journal, 7 (1965), pp. 308–313.

[100] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng, *Reading digits in natural images with unsupervised feature learning*, in NIPS workshop on deep learning and unsupervised feature learning, vol. 2011, Granada, Spain, 2011, p. 5.

[101] Andrew Ng, *Sparse autoencoder*, CS294A Lecture notes, 72 (2011).

[102] Bruno A Olshausen and David J Field, *Sparse coding with an overcomplete basis set: A strategy employed by v1?*, Vision research, 37 (1997), pp. 3311–3325.

[103] Anh Huy Phan, Andrzej Cichocki, Petr Tichavskỳ, Gheorghe Luta, and Austin J Brockmeier, *Tensor completion through multiple kronecker product decomposition.*, in ICASSP, 2013, pp. 3233–3237.

[104] Anh Huy Phan, Andrzej Cichocki, Petr Tichavskỳ, Danilo P Mandic, and Kiyotoshi Matsuoka, *On revealing replicating structures in multiway data: A novel tensor decomposition approach*, in Latent Variable Analysis and Signal Separation, Springer, 2012, pp. 297–305.

[105] Benjamin Recht, Maryam Fazel, and Pablo A. Parrilo, *Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization*, SIAM Rev., 52 (2010), pp. 471–501.

[106] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio, *Contractive auto-encoders: Explicit invariance during feature extraction*, in Proceedings of the 28th International Conference on Machine Learning (ICML-11), 2011, pp. 833–840.

[107] Roberto Rigamonti, Amos Sironi, Vincent Lepetit, and Pascal Fua, *Learning separable filters*, in Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on, IEEE, 2013, pp. 2754–2761.

[108] Ralph Tyrell Rockafellar, *Convex analysis*, Princeton university press, 2015.

[109] Frank Rosenblatt, *Principles of neurodynamics*, (1962).

[110] Leonid I Rudin, Stanley Osher, and Emad Fatemi, *Nonlinear total variation based noise removal algorithms*, Physica D: Nonlinear Phenomena, 60 (1992), pp. 259–268.

[111] Sartaj Sahni and Teofilo Gonzalez, *P-complete approximation problems*, Journal of the ACM (JACM), 23 (1976), pp. 555–565.

[112] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran, *Low-rank matrix factorization for deep neural network training with high-dimensional output targets*, in Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on, IEEE, 2013, pp. 6655–6659.

[113] Hayden Schaeffer and Stanley Osher, *A low patch-rank interpretation of texture*, SIAM Journal on Imaging Sciences, 6 (2013), pp. 226–262.

[114] Florian Schroff, Dmitry Kalenichenko, and James Philbin, *Facenet: A unified embedding for face recognition and clustering*, arXiv preprint arXiv:1503.03832, (2015).

[115] Pierre Sermanet, Sandhya Chintala, and Yann LeCun, *Convolutional neural networks applied to house numbers digit classification*, in Pattern Recognition (ICPR), 2012 21st International Conference on, IEEE, 2012, pp. 3288–3291.

[116] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun, *Overfeat: Integrated recognition, localization and detection using convolutional networks*, arXiv preprint arXiv:1312.6229, (2013).

[117] Marco Signoretto, Lieven De Lathauwer, and Johan AK Suykens, *Nuclear norms for tensors and their use for convex multilinear estimation*, Submitted to Linear Algebra and Its Applications, 43 (2010).

[118] Marco Signoretto, Dinh Quoc Tran, Lieven De Lathauwer, and Johan A. K. Suykens, *Learning with tensors: a framework based on convex optimization and spectral regularization*, Machine Learning, 94 (2014), pp. 303–351.

[119] Marco Signoretto, Raf Van de Plas, Bart De Moor, and Johan AK Suykens, *Tensor versus matrix completion: a comparison with application to spectral data*, Signal Processing Letters, IEEE, 18 (2011), pp. 403–406.

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

[121] Jinli Suo, Song-Chun Zhu, Shiguang Shan, and Xilin Chen, *A compositional and dynamic model for face aging*, Pattern Analysis and Machine Intelligence, IEEE Transactions on, 32 (2010), pp. 385–401.

[122] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich, *Going deeper with convolutions*, arXiv preprint arXiv:1409.4842, (2014).

[123] Ryota Tomioka, Kohei Hayashi, and Hisashi Kashima, *On the extension of trace norm to tensors*, in NIPS Workshop on Tensors, Kernels, and Machine Learning, 2010, p. 7.

[124] Theodoros Tsiligkaridis and Alfred O Hero, *Covariance estimation in high dimensions via kronecker product expansions*, Signal processing, ieee transactions on, 61 (2013), pp. 5347–5360.

[125] Ledyard R Tucker, *Some mathematical notes on three-mode factor analysis*, Psychometrika, 31 (1966), pp. 279–311.

[126] Constantin Udriste, *Convex functions and optimization methods on Riemannian manifolds*, vol. 297, Springer, 1994.

[127] Rafael Uetz and Sven Behnke, *Locally-connected hierarchical neural networks for gpu-accelerated object recognition*, in NIPS 2009 Workshop on Large-Scale Machine Learning: Parallelism and Massive Datasets, 2009.

[128] CF Van Loan, NP Pitsianis, MS Moonen, and GH Golub, *Linear algebra for large scale and real time applications*, 1993.

[129] Charles F Van Loan, *The ubiquitous kronecker product*, Journal of computational and applied mathematics, 123 (2000), pp. 85–100.

[130] Charles F Van Loan and Nikos Pitsianis, *Approximation with Kronecker products*, Springer, 1993.

[131] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol, *Extracting and composing robust features with denoising autoencoders*, in Proceedings of the 25th international conference on Machine learning, ACM, 2008, pp. 1096–1103.

[132] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol, *Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion*, The Journal of Machine Learning Research, 11 (2010), pp. 3371–3408.

[133] Gregory K Wallace, *The jpeg still picture compression standard*, Communications of the ACM, 34 (1991), pp. 30–44.

[134] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus, *Regularization of neural networks using dropconnect*, in Proceedings of the 30th International Conference on Machine Learning (ICML-13), 2013, pp. 1058–1066.

[135] David S Watkins and Ludwig Elsner, *Self-similar flows*, Linear algebra and its applications, 110 (1988), pp. 213–242.

[136] Chenyu Wu, Ce Liu, Heung-Yueng Shum, Ying-Qing Xy, and Zhengyou Zhang, *Automatic eyeglasses removal from face images*, Pattern Analysis and Machine Intelligence, IEEE Transactions on, 26 (2004), pp. 322–336.

[137] Jian Xue, Jinyu Li, and Yifan Gong, *Restructuring of deep neural network acoustic models with singular value decomposition.*, in INTERSPEECH, 2013, pp. 2365–2369.

[138] Jianchao Yang, John Wright, Thomas Huang, and Yi Ma, *Image super-resolution as sparse representation of raw image patches*, in Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, IEEE, 2008, pp. 1–8.

[139] Zhou Yisu, Unpublished personal communications, (2014).

[140] Huisu Yoon, Kyung Sang Kim, Dongkyu Kim, Yoram Bresler, and Jong Chul Ye, *Motion adaptive patch-based low-rank approach for compressed sensing cardiac cine mri*, Medical Imaging, IEEE Transactions on, 33 (2014), pp. 2069–2085.

[141] Matthew D Zeiler, *Adadelta: an adaptive learning rate method*, arXiv preprint arXiv:1212.5701, (2012).

[142] Matthew D Zeiler and Rob Fergus, *Stochastic pooling for regularization of deep convolutional neural networks*, arXiv preprint arXiv:1301.3557, (2013).

[143] Honggang Zhang, Jun Guo, Guang Chen, and Chunguang Li, *Hcl2000-a large-scale handwritten chinese character database for handwritten character recognition*, in Document Analysis and Recognition, 2009. ICDAR'09. 10th International Conference on, IEEE, 2009, pp. 286–290.

[144] Tong Zhang and Gene H. Golub, *Rank-one approximation to high order tensors*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 534–550.

[145] Xin Zhang, Zhouchen Lin, Fuchun Sun, and Yi Ma, *Rectification of optical characters as transform invariant low-rank textures*, in Document Analysis and Recognition (ICDAR), 2013 12th International Conference on, IEEE, 2013, pp. 393–397.

[146] Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun, *Efficient and accurate approximations of nonlinear convolutional networks*, arXiv preprint arXiv:1411.4229, (2014).

[147] Yu Zhang, Ekapol Chuangsuwanich, and James Glass, *Extracting deep neural network bottleneck features using low-rank matrix factorization*, in Proc. ICASSP, 2014.

[148] Hui Zhao, Jiuqiang Han, Naiyan Wang, Congfu Xu, and Zhihua Zhang, *A fast spectral relaxation approach to matrix completion via kronecker products*, in Twenty-Fifth AAAI Conference on Artificial Intelligence, 2011.

[149] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip Torr, *Conditional random fields as recurrent neural networks*, arXiv preprint arXiv:1502.03240, (2015).

[150] Shuchang Zhou, Zhihua Zhang, and Xiaobing Feng, *Group orbit optimization: A unified approach to data normalization*, arXiv preprint arXiv:1410.0868, (2014).

[151] Zhenyao Zhu, Ping Luo, Xiaogang Wang, and Xiaoou Tang, *Recover canonical-view faces in the wild with deep neural networks*, arXiv preprint arXiv:1404.3543, (2014).