Utilizing dynamical systems as layers to help build deep learning models

by

Zihang Meng

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2022

Date of final oral examination: 08/17/2022

The dissertation is approved by the following members of the Oral Committee:

Sharon Yixuan Li, Assistant Professor, Computer Sciences
Mohit Gupta, Assistant Professor, Computer Sciences
Yingyu Liang, Assistant Professor, Computer Sciences
Yin Li, Assistant Professor, Biostatistics and Medical Informatics
Vikas Singh (Adviser), Professor, Biostatistics and Medical Informatics

Acknowledgments

First and foremost, I would like to express my sincere gratitude to Vikas for his continuous support during my Ph.D study. His enthusiasm for research and his wealth of knowledge inspired me and helped shape my views as an independent researcher. I thank the rest of my thesis committee: Sharon, Mohit, Yingyu and Yin, for their insightful comments. I also thank all my collaborators. It was a great pleasure to work with them. Finally, many thanks to my friends and family for their precious support, without which it would not be possible to finish my Ph.D.

Contents

List of Figures

List of Tables

Abstract

Deep learning models have achieved great success in a wide range of areas over the past decade, like image processing, natural language processing, audio recognition and robot control. When building a deep learning model for a specific task, one of the main challenges is to choose the appropriate type of layers that suits the task of interest. After building blocks (layers) are chosen, a deep model can be formed by simply stacking them in an appropriate way. As a result, a majority of milestones in deep learning can be attributed to the introduction of novel layers. To enrich the family of deep learning layers, the dynamical system is a very useful subject to study since it covers a large family of powerful procedures/algorithms. In this thesis, our goal is to identify suitable dynamical systems and develop machine learning algorithms to utilize them as layers within deep neural networks to solve computer vision problems while addressing application-specific challenges. We show the effectiveness of our ideas on various problems where our proposed layers can be integrated into deep models to solve the problems accurately and efficiently.

Chapter 1

Introduction

Deep learning models have achieved great success in a wide range of areas over the past decade, like image classification He et al. (2016); Russakovsky et al. (2015); Szegedy et al. (2016) and segmentation Chen et al. (2017); Kirillov et al. (2019); Milletari et al. (2016), video processing Kahou et al. (2016); Caelles et al. (2017); Feichtenhofer et al. (2017), audio recognition Purwins et al. (2019); Noda et al. (2015), multimodal learning Ramachandram and Taylor (2017); Meng et al. (2021e,d), robot control Lillicrap et al. (2015); Arulkumaran et al. (2017); Duan et al. (2016), etc. This success comes from several factors including large-scale datasets, specialized hardwares to support the running of deep models, and the backbone networks which can be modified to work for different applications. In this dissertation, we study the building blocks of the backbone networks and explore what ideas can be used to motivate/design better backbones with better performance and the ability to model more data structures.

To understand or explain the operational mechanics of deep learning models properly, it is helpful to first understand an object known as the computational graph which plays a crucial role. Fig. 1.1 shows a simple example of a computational graph $y = wx + b$ where the output equals the product of the input and some scalar, which is then added to another scalar.

If we make this example a little more complicated, by doing this kind of operation twice, we get Fig. 1.2. It is easy to notice that the subgraph in the green circle has a similar structure to the one in the blue circle. We can keep adding more and more operations to the graph until the model fits our need. Then, we could denote this operation in a subgraph as a *layer* and the whole computational graph contains

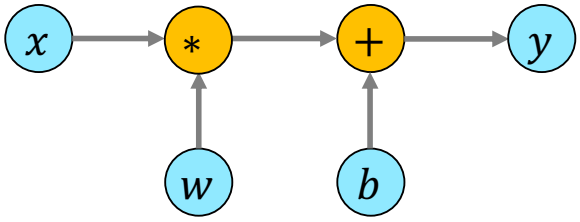


Figure 1.1: A simple example of the computational graph.

two (or more) layers. In the context of deep learning, generally speaking, the subgraph (the layer) can contain any type of operations. For example, the subgraph can contain a full sequence of steps based on a certain algorithm for solving some task. To instantiate this computational graph in deep learning pipeline, we need to define the forward pass of this algorithm clearly, as well as how to compute the backward gradients (or say the backward pass). Briefly, a forward pass refers to the execution of a sequence of pre-defined operations, which include the calculation and storage of intermediate variables in the order from the beginning (input) layer to the last (output) layer. The backward pass refers to the process of computing the gradients from the final loss function back to all trainable parameters in the network by the chain rule.

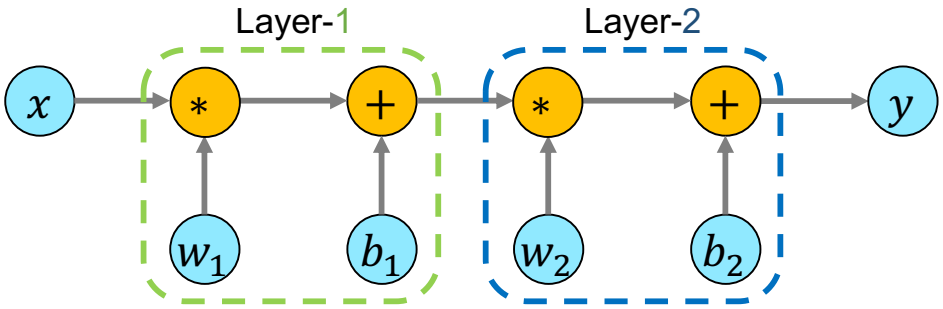


Figure 1.2: A simple example of the computational graph with two "layers".

Besides expressing the operations preceisely, the computational graph also offers other advantages. For example, when the data or model becomes too large to compute on a single machine, distributed computing can be directly implemented according to the computational graph. The computational graph is also helpful when we want to solve relational problems where the relationship between variables can usually be expressed in terms of a graph.

If we look at the computational graph of a deep learning model, the graph is often quite deep Goodfellow et al. (2016), which shows how the model captures more complex concepts on top of simpler ones. This hierarchy of layers is the main characteristic of deep learning models and the key reason why deep models can derive representations for many tasks without the need for human specialists to specify all the knowledge. Naturally, when building a deep learning model for a specific task, one of the main challenges is to choose appropriate type of *layers* that suit the task of interest (Fig. 1.3). After the building blocks (the layers) are chosen, a deep model can be formed by simply stacking them in an appropriate way (in most cases just sequential stacking). As a result, a majority of milestones in deep learning can be attributed to the introduction of novel *layers*. In the following, we briefly review some examples showing how the introductions of novel layers keep enlarging the capacities of deep learning models, which contributes to their success in a wide range of areas.

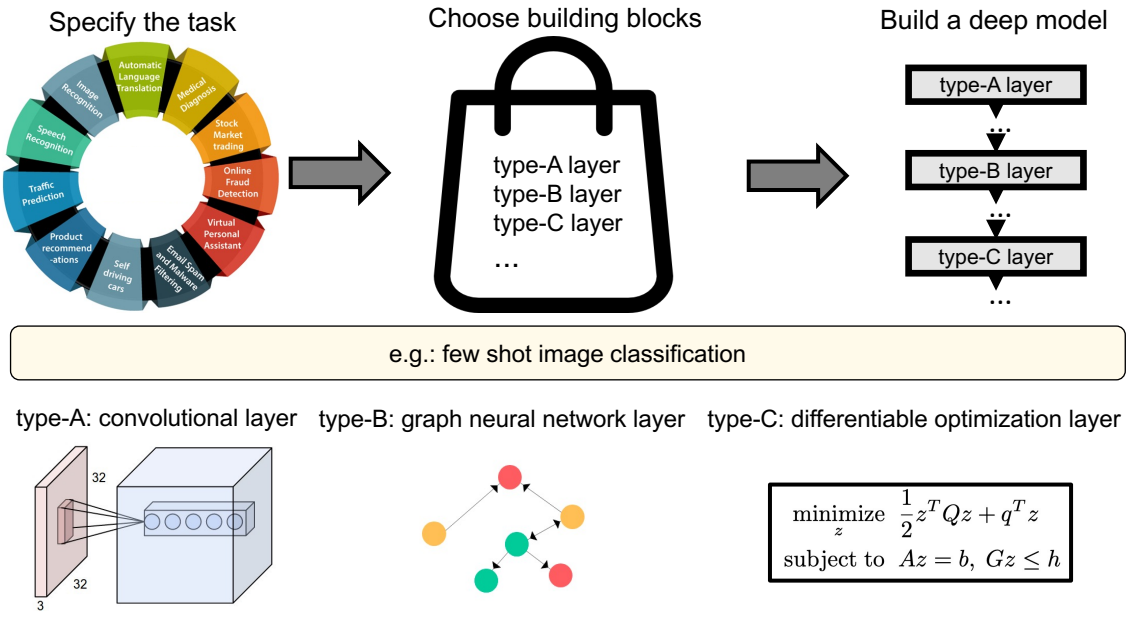


Figure 1.3: General steps to build a deep model. The key step is to choose appropriate layers as building blocks to build the needed model for the task of interest.

1. *Fully-Connected layers*: Arguably, the simplest layer is the *Fully-Connected* (FC) layers. The Multi-Layer perceptron model Rumelhart et al. (1985) is formed by stacking multiple FC layers in a sequential way with non-linear activation

functions. The Multi-Layer perceptron model can distinguish data that is not linear separable. Each FC layer consists of a matrix multiplication between the layer input and the weight matrix, and adding a bias term. If a FC layer is an intermediate layer in the model we are considering, usually a non-linear activation function is added after each FC layer.

2. *Convolutional layers*: Although a Multi-Layer perceptron formed by FC layers can be trained to distinguish complex data and theoretically can approximate any function according to the universal approximation theorem Hornik et al. (1989), in practice, their performance is limited, especially for structured data like 2-D images. Convolutional layers, due to the useful properties of convolution, are more suitable to learn from 2-D images. Convolutional layers were used together with backpropagation by LeCun et al. (1998) to identify the number in an image of the hand-written digit and became the foundation of modern convolutional neural networks. Convolutional layers are suitable for images, but still have limitations in terms of the modeling ability. In the past decade, many novel convolutional layers were proposed to address the limitations. Examples include dilated convolutional layers Yu and Koltun (2015), residual layers He et al. (2016), deformable convolutional layers Dai et al. (2017), etc.

   Besides modifying convolutional layers, researchers have also drawn inspiration from other fields to propose novel type of layers, which allow deep learning models to perform better and work well for more various data structures.

3. *Message passing layer for better segmentation*: While convolutional neural networks have become the dominant approach in image segmentation, the results around object boundaries are relatively poor. Chen et al. (2017) proposes that adding a conditional random field (solved by message passing updates) can refine the results produced by CNNs to get better segmentation results, especially on object boundaries due to the nature of probabilistic graphical models. This kind of layer has also been utilized by others for 3D segmentation Kamnitsas et al. (2017).

4. *Quadratic optimization layer for solving quadratic optimization in the latent space*: Even modern convolutional layers often cannot encode constraints and capture complex dependencies between hidden states, such as capturing an optimization problem like linear programming or quadratic programming. Amos and

Kolter (2017) proposes a layer which can integrate quadratic optimization problem as individual layers in end-to-end trainable deep networks. This opens the door to more possibilities like differentiable end-to-end planning and control, differentiable integration of SVMs into deep networks. Besides building differentiable optimization layers from scratch, building them on top of some black-box optimizer has also been explored Berthet et al. (2020); Vlastelica et al. (2019).

5. *Stochastic layers*: Most layers used in deep models are deterministic which means that the latent states are determinictic. Fraccaro et al. (2016) proposed a stochastic sequential layer to form the recurrent neural network which models the uncertainty in the latent space explicitly and improves the performance in sequence modeling by a large margin. Liu et al. (2019c) shows that adding stochasticity can help stabilize the neural ODE networks.

From the above examples, we can see that these novel layers play an important role in expanding the ability and scope of deep learning models and enabling deep models to work for various problems. In this thesis, this need for novel type of layers in deep learning is our main motivation. Specifically, we draw inspiration from dynamical systems and propose novel layers to enrich deep models with more/better modeling abilities, including learning better representations of graph structured data, solving linear programming informed by the latent variables, computing the canonical correlation between variables, and modeling the stochastic flow of features, as we will describe in more details in the remainder of this thesis.

1.1 Novel deep learning layers inspired by dynamical systems

As we discussed above, the notion of a *layer* can refer to any procedure/algorithm (with a clear forward and backward pass). If we take the position that the family of deep learning layers can be much richer, the dynamical system is a very useful starting point to study since it covers a large family of powerful procedures/algorithms under a unified framework. Besides, a dynamical system often contains operations that are sequentially conducted, thus a potentially clear forward/backward pass.

Dynamical systems have been studied for many decades and there have been some well-established theories about their properties Strogatz (2018); Michel et al. (2001); Hirsch (1988) like the steady state, robustness to noise, convergence, etc. Deep learning models, on the other hand, have demonstrated great potential in recent years while some of their properties remain unknown and the capacity is sometimes limited by the chosen layers. When used properly, dynamical systems and deep learning models can potentially help the development of each other. For example, we can use a DNN to solve the partial differential equation at much faster speed compared to traditional numerical solvers Lu et al. (2021); We can also use the established theories from dynamical systems to help analyze the properties of DNNs Thorpe and van Gennip (2018), or use dynamical systems as layers to help build DNNs Scarselli et al. (2008).

To understand more closely how dynamical systems can help deep learning, first note that dynamical systems already have a close relationship with deep learning models. In fact, some DNNs themselves can be viewed as a dynamical system, for example, the construction of Resnet He et al. (2016) can be viewed as a dynamical system Weinan et al. (2019). In the simplest form, the feed-forward propagation in a T-layer residual network can be represented by the difference equations:

$$x_{t+1} = x_t + f(x_t, \theta_t), \quad t = 0, ..., T-1 \tag{1.1}$$

where x_0 is the input which can take various forms depending on the specific application, like an image, time-series data, etc. x_T is the final output of the neural network. Usually, the final output will be compared with some target label corresponding to this input via some loss function. This expression of Resnet can be thought of as a discretization of the ordinary differential equation (ODE):

$$\dot{x}_t = f(x_t, \theta_t) \tag{1.2}$$

with the initial condition equal to the network input x_0.

This dynamical system view of DNNs enables researchers to use the well developed theory from the dynamical system community to help analyze the properties of DNNs. Weinan et al. (2019) introduced a mathematical formulation of the population risk minimization problem in deep learning as a mean-field optimal control problem and established some quantitative relationships between population and

empirical learning problems. Thorpe and van Gennip (2018) showed that the deep layer limit coincides with a parameter estimation problem for a nonlinear ODE and showed the convergence of the estimation problem in a variational sense. Haber and Ruthotto (2017) propose new forward propagation techniques inspired by systems of ODE that overcome the critical issues of exploding or vanishing gradients and lead to well-posed learning problems for arbitrarily deep networks.

Besides utilizing the theory from dynamical systems to understand existing DNN models, we can also utilize dynamical systems to help build new DNN models which is a central thread of this thesis.

In this thesis, we devote our main efforts in this direction. Specifically, we focus on utilizing dynamical systems as layers in the constructed DNN pipeline, to bring new abilities to DNN models which cannot be accomplished with commonly used layers like the fully connected layer, the convolutional layer, etc.

Let us consider an example that demonstrates how dynamical systems can help build DNN models.

Example 1. Scarselli et al. (2008) unroll a dynamical system to construct the graph neural network model which can solve a variety of problems on graphs. Specifically, Scarselli et al. (2008) consider the set $\mathcal{D}$ of pairs of a graph and a node as the domain, *i.e.*, $\mathcal{D} = \mathcal{G} \times \mathcal{N}$ where $\mathcal{G}$ is a set of the graphs and $\mathcal{N}$ is a subset of their nodes. In the model, they attach a state $x_n \in \mathbb{R}^s$ to each node with index n that is based on the information contained in the neighborhood of node n. The state x_n contains a representation of the concept denoted by n and can be used to produce an output o_n, *i.e.*, a decision about the concept. In this graph model, the nodes iteratively pass information to their neighbor nodes and update the node state according to the received message. Let $\mathbf{x}, \mathbf{o}, \mathbf{l}$ be the vectors constructed by stacking all the states, all the outputs, all the labels. The iterative updating scheme is,

$$\mathbf{x}(t+1) = f_\theta(\mathbf{x}(t), \mathbf{l}) \tag{1.3}$$

where f_θ is the transition function and θ refers to the parameters.

Scarselli et al. (2008) demonstrates that this dynamical system which updates the graph node states converge exponentially fast to the final solution, which enables unrolling the dynamical system and treating each updating operation as one

parameterized layer, and the whole module can be used as a DNN that is suitable for modeling various problems on graphs. The constructed DNN model, *i.e.*, the graph neural network model, inherits the good property of the dynamical system to converge quickly when updating the node states in the network, which is not achievable using standard fully connected layers or convolutional layers. In this example, each unrolled step is basically a *layer* (corresponding to a computational graph of a set of steps) and all layers together form the whole graph neural network model.

Knowing that dynamical systems can help build DNN models and that DNNs are formed by different layers, how do we instantiate the dynamical systems as layers?

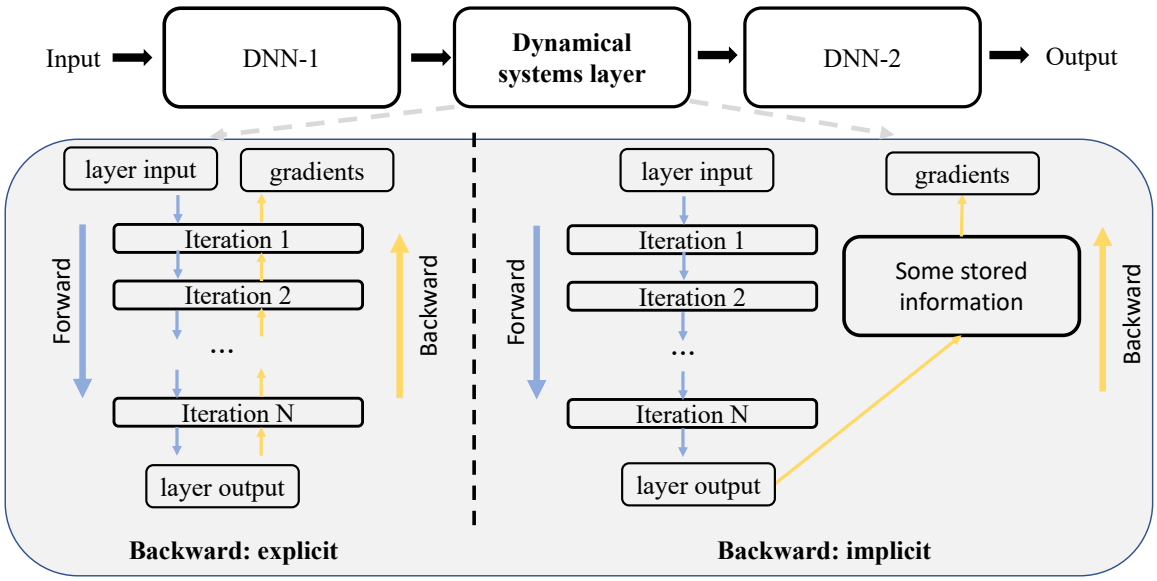


Figure 1.4: Two types of method to differentiate through a dynamical system layer.

After we find a suitable dynamical system to use for the problem of interest, how to instantiate it as a layer in deep neural networks is not trivial. There are several challenges here: (i) The dynamical systems need to be differentiable. (ii) The runtime of the overall model needs to be considered. (iii) The memory cost of the overall model needs to be taken into account. Let us discuss (i) a little more. The methods to differentiate through a dynamical system can be divided into two types: one can choose to either explicitly store all the intermediate computational graphs and unroll all the operations to compute backward gradients (Fig. 1.4 left), or figure out some way to implicitly compute the backward gradients without saving all the intermediate computational graph on GPU (Fig. 1.4 left). The first approach usually has a larger

memory cost while the correct backward gradient is guaranteed by construction and it is even possible to parameterize the unrolled steps. The second approach usually has a small memory cost because it does not store all the intermediate steps, while its time cost and the quality of backward gradients vary according to the specific chosen methods.

Example 2. Chen et al. (2018) propose to instantiate an ODE using neural networks and consider optimizing a scalar-valued loss function $L()$ whose input is the result of an ODE solver:

$$L(x(t_1)) = L(\text{ODESolve}(x(t_0), f, t_0, t_1, \theta)) \tag{1.4}$$

where f is the state update function parameterized by θ.

To optimize L, we need backward gradients with respect to θ. One problem is that usually an ODE may take many steps to compute the forward pass, thus if we need to store all the computational graphs for computing the backward gradients, the memory cost may be huge. Thus, Chen et al. (2018) treats the ODE solver as a black-box and computes the gradients using the adjoint sensitivity method Pontryagin (1987), which computes the gradients by solving a second, augmented ODE backward in time. Let $a(t) = \frac{\partial L}{\partial x(t)}$, the dynamics of this quantity called the adjoint is given by another ODE:

$$\frac{da(t)}{dt} = -a(t)^T \frac{\partial f(x(t), t, \theta)}{dx} \tag{1.5}$$

such that we do not need to store the computational graph during the forward pass. That being said, the memory cost issue can also be solved by identifying a suitable dynamical system that converges well within a small number of iterations. As an example, in Meng et al. (2020) we utilize physarum dynamics to construct a linear programming solver and unrolling can be used due to a small number of needed iterations.

Overall, the goal of this thesis is: *identify suitable dynamical systems and develop machine learning algorithms to utilize them as layers within deep neural networks to solve computer vision problems while addressing application-specific challenges.*

1.2 Contribution and Scope of the Thesis

In this thesis, we explore how dynamical systems can help deep learning and how to utilize them within a deep neural network on various applications in computer vision. We show the overall scope of the thesis along two axes as shown in Fig. 1.5, where the vertical axis represents the type of forward pass of the layer formed using the dynamical system, and the horizontal axis represents the type of backward pass. For example, if we construct a layer motivated by a dynamical system with stochastic terms, and use the unrolling to compute backward gradients, then it should fall on the lower right quadrant of Fig. 1.5. In this section, we will briefly review five problems studied in this thesis.

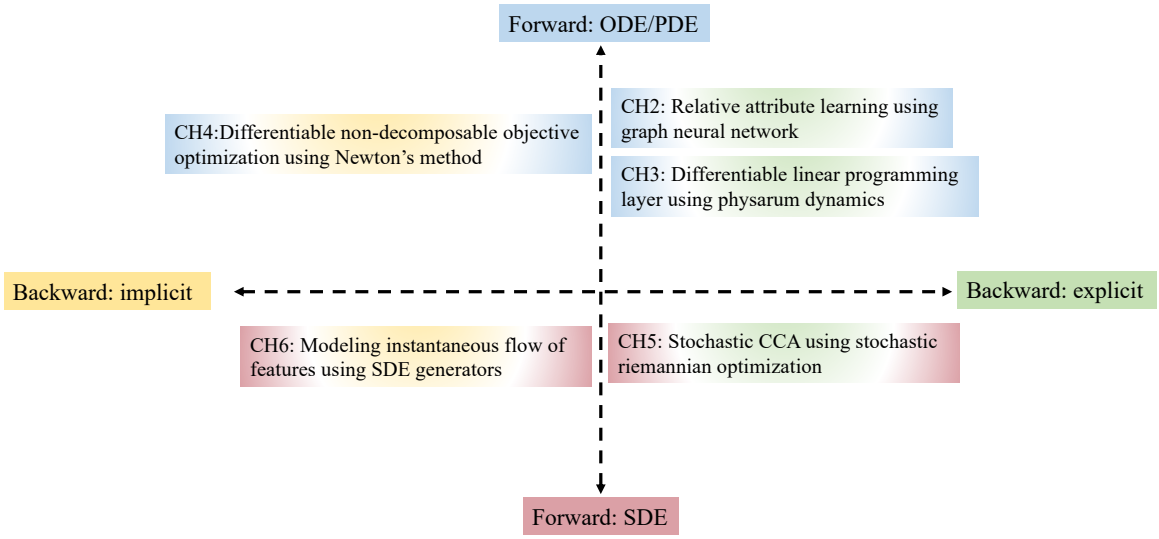


Figure 1.5: The overall scope of the thesis. The layer formed by the dynamical system can be divided into four types according to its forward and backward pass.

Graph Neural Networks for Relative Attribute Learning

A sizable body of work on relative attributes provides evidence that relating pairs of images along a continuum of strength pertaining to a visual attribute yields improvements in a variety of vision tasks. We will show how instantiating a dynamical system (message passing) on the graph nodes as layers (graph neural networks) can yield a solution to various problems that broadly fall under relative attribute learning. Our main idea is the observation that relative attribute learning naturally

benefits from exploiting the graph of dependencies among the different relative attributes of images, especially when only partial ordering is provided at training time. We propose to use message passing to perform end-to-end learning of the image representations, their relationships as well as the interplay between different attributes. We will show by experiments that this simple framework is effective in achieving competitive accuracy with specialized methods for both relative attribute learning and binary attribute prediction, while relaxing the requirements on the training data and/or the number of parameters, or both.

Physarum Powered Differentiable Linear Programming Layers

Consider a learning algorithm, which involves an internal call to an optimization routine such as a generalized eigenvalue problem, a cone programming problem or even sorting. Integrating such a method as a layer(s) within a trainable deep neural network (DNN) in an efficient and numerically stable way is not straightforward – for instance, only recently, strategies have emerged for eigendecomposition and differentiable sorting. We propose an efficient and differentiable solver for general linear programming problems which can be used in a plug-and-play manner within DNNs as a layer. Our development is inspired by a fascinating but not widely used link between the dynamical system of slime mold (physarum) and optimization schemes such as steepest descent. We describe our development and show the use of our solver in a video segmentation task and meta-learning for few-shot learning. We review the existing results and provide a technical analysis describing their applicability for our use cases. We will show that our solver performs comparably with a customized projected gradient descent method on the first task and outperforms the differentiable CVXPY-SCS solver on the second task. Experiments will show that our solver converges quickly without the need for a feasible initial point. Our proposal is easy to implement and can easily serve as layers whenever a learning procedure needs a fast approximate solution to an LP, within a larger network.

Differentiable Optimization of Generalized Nondecomposable Functions using Linear Programs

We propose a framework that makes it feasible to directly train deep neural networks with respect to popular families of task-specific non-decomposable perfor-

mance measures such as AUC, multi-class AUC, F-measure and others. A feature of the optimization model that emerges from these tasks is that it involves solving Linear Programs (LP) during training where representations learned by upstream layers characterize the constraints or the feasible set. The constraint matrix is not only large but the constraints are also modified at each iteration. The large number of constraints brings additional challenges compared to the case of our physarum solver. We will show how adopting a set of ingenious ideas proposed by Mangasarian for 1-norm SVMs – which advocates for solving LPs with a generalized Newton method, whose updating process can be viewed as a dynamical system – provides a simple and effective solution that can be run on the GPU. In particular, this strategy needs little unrolling, which makes it more efficient during the backward pass. Further, even when the constraint matrix is too large to fit on the GPU memory (say large minibatch settings), we will show that running the Newton method in a lower-dimensional space yields accurate gradients for training, by utilizing a statistical concept called *sufficient* dimension reduction. While a number of specialized algorithms have been proposed for the models that we describe here, our module turns out to be applicable without any specific adjustments or relaxations. We will describe each use case, study its properties and demonstrate the efficacy of the approach over alternatives which use surrogate lower bounds and often, specialized optimization schemes. We will show that frequently, we can achieve superior computational behavior and performance improvements on common datasets used in the literature.

An Online Riemannian PCA for Stochastic CCA

We present an efficient stochastic algorithm (RSG+) for canonical correlation analysis (CCA) using a reparametrization of the projection matrices. We will show how this reparametrization (into structured matrices), simple in hindsight, directly presents an opportunity to repurpose/adjust mature techniques for numerical optimization on Riemannian manifolds. Our developments nicely complement existing methods for this problem which either require $O(d^3)$ time complexity per iteration with $O(\frac{1}{\sqrt{t}})$ convergence rate (where d is the dimensionality) or only extract the top 1 component with $O(\frac{1}{t})$ convergence rate. In contrast, our algorithm offers a strict improvement for this classical problem: it achieves $O(d^2k)$ runtime complexity per iteration for extracting the top k canonical components with $O(\frac{1}{t})$ convergence rate. While this chapter primarily focuses on the formulation and technical analysis of its properties,

we will show that the dynamics of streaming CCA updates can be integrated into DNNs which scales better than the existing DeepCCA method, and our experiments will show that the empirical behavior on common datasets is quite promising. We will also explore a potential application in training fair models where the label of the protected attribute is missing or otherwise unavailable.

Modeling Instantaneous flow of features via SDE Generators

We will study how stochastic differential equation (SDE, a kind of dynamical system with stochastic terms) based ideas can inspire new modifications to existing algorithms for a set of problems in computer vision. Loosely speaking, our formulation is related to both explicit and implicit strategies for data augmentation and group equivariance, but is derived from new results in the SDE literature on estimating infinitesimal generators of a class of stochastic processes. If and when there is a nominal agreement between the needs of an application/task and the inherent properties and behavior of the types of processes (which we can efficiently handle), we obtain a very simple and efficient plug-in layer that can be incorporated within any existing network architecture, with minimal modification and only a few additional parameters. We will show experiments on a number of vision tasks including few-shot learning, point cloud transformers and deep variational segmentation obtaining efficiency or performance improvements.

1.3 Outline

From Chapter 2–6 we describe the five problems we study and the proposed solutions in detail. We demonstrate how to utilize a graph neural network (message passing dynamics instantiated as layers) to model relative attribute learning in Chapter 2. Then, we identify a useful dynamical system called physarum dynamics and utilize it to construct a differentiable linear programming layer used in deep neural networks in Chapter 3. Next, we propose a framework for solving non-decomposable objectives using linear programming and construct a differentiable linear programming solver for this type of problem in Chapter 4, utilizing the dynamics of a kind of Newton's method. The solvers we propose in Chapter 3 and 4 are special instances of dynamical systems and used during the forward pass of a neural network, then in Chapter 5, we describe how we model the dynamics that happens across the whole

training process of a neural network, by designing an online PCA based algorithm for solving stochastic CCA. Finally, we show a new scheme of using SDEs to model the possible dynamics happening in the feature space in Chapter 6. We draw the conclusions and discuss the future work in Chapter 7.

Chapter 2

Background

In this chapter, we briefly describe the basics of dynamical systems and algorithms (message passing, physarum dynamics, a Newton's method for solving linear programming, canonical correlation analysis, and stochastic differential equation generators) which facilitate the discussion of our works on constructing novel layers in later chapters.

2.1 Notations

Before going into introducing the background, here we introduce some standard notations that we follow consistently throughout the thesis.

- We use $\mathbb{R}^n$ to represent n-dimensional vector space over the reals.
- Vectors are denoted in lower case like x, y, z and matrices are denoted in upper case such as C, X, Y, T.
- We use calligraphic capital letters to represent sets like $\mathcal{I}, \mathcal{J}$.
- X_{ij} denotes the element at the i-th row and j-th column of the matrix X.
- Inner product between $x, y \in \mathbb{R}^n$ is denoted by $x^\top y$.
- x_i denotes the vector x indexed by i.
- trace(T) denotes the trace of a matrix T, trace(T)=$\sum_i T_{ii}$.
- $A \otimes B$: Kronecker product of matrices A and B.

- I_r: Identity matrix of size r and $\mathbb{1}$ is the indicator function.
- $\mathbf{B}_{k,-}$ (and $\mathbf{B}_{|,k'}$) gives the k-th row (and k'-th) column of $\mathbf{B}$.

2.2 Message Passing

Message passing generally refers to an iterative scheme on a graph which passes messages between nodes and update the nodes. In the context of probabilistic graphical models Jordan (1999), the sum-product message passing is also known as belief propagation Yedidia et al. (2003), which refers to a message-passing algorithm for performing inference on graphical models, such as Bayesian networks Murphy et al. (2013) and Markov random fields Freeman and Pasztor (2000).

Graphical models. Probabilistic graphical models can describe joint distributions in a structured way, and the structure often allows us to compute certain probabilistic estimates efficiently even when the whole distribution is too complex to write explicitly. These models encode a given distribution into a graph where the representations (possibly high-dimensional) of nodes and edges are computed from the given distribution. Specifically, in a graphical representation, the nodes correspond to the variables in some domain, and the edges correspond to direct probabilistic interactions between them. For example, the graphical representation in Fig. 2.1 Koller and Friedman (2009) illustrates one possible graph structure for the relationship between being affected by Flu and several related factors. In this graph, we can see that there is no direct interaction between Muscle, Pain and Season, but both interact directly with Flu. The graph is a compact representation of a set of independencies that hold in the distribution; these properties take the form X is independent of Y given Z, denoted $(X \perp Y|Z)$, for some subsets of variables X, Y, Z. For example, Fig. 2.1 suggests that

$$P(\text{Congestion}|\text{Flu, Hayfever, Season}) = P(\text{Congestion}|\text{FLu, Hayfever}) \tag{2.1}$$

If we want to know the probabilistic distribution of the person having congestion, and we know whether he has the flu and whether he has hayfever, the season does not provide additional information. Note that this statement does not mean that Season is independent of Congestion; It only means that all of the information we may obtain from the season on the probability of having congestion we already know by

knowing whether the person has the flu and hayfever. The middle part of Fig. 2.1 shows the set of independence assumptions associated with the graph. We refer readers to Koller and Friedman (2009) for more detailed introduction of the probabilistic graphical models.

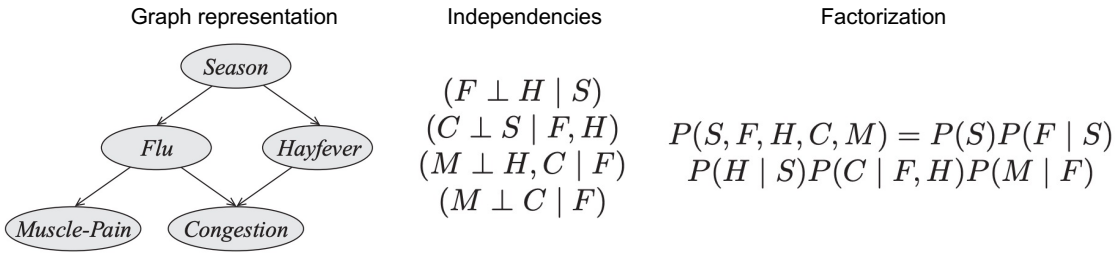


Figure 2.1: An example of the probabilistic graphical model.

Representation and inference on graphical models. The graphical language can encode many distributions that we often need in practice which have the property that variables tend to interact (have edges) directly only with a few others.

This graphical framework has many advantages. First, we can write down the distribution in a tractable form using the graphical models even when the explicit representation of the joint distribution is very large. Another important property of this type of representation is that it provide transparency: a specialist can look at the graph and understand the reasoning behind a solution computed from this model.

Second, the same structure also allows us to do inference, which is to answer queries using the distribution (represented by the graph). For example, there are algorithms to compute the posterior probability of some variables conditioned on some other variables. In Fig. 2.1, we may observe that the season is spring and the patient has muscle pain, and we would like to know how likely he/she is to have the flu. This query can be written as P(Flu=true|Season=spring, Muscle Pain=true). The inference algorithms (such as message passing) can work directly on the graph structure and are generally several orders of magnitude faster than manipulating the joint distribution explicitly.

Message passing. The message passing scheme, as a way to performing inference on the graphical model, proceeds as follows. Let node i represents the variable x_i, and $\phi_{ij}(x_i, x_j)$ denote some compatibility function (depending on specific use case) that measures some score between x_i and x_j. Usually there are three steps to compute the message from node j to node i,

1. Multiply together all messages coming in to node j, except for the message from node i back to node j.

2. Multiply by the compatibility function $\phi_{ij}(x_i, x_j)$.

3. Marginalize over the variable x_j.

The three steps above can be summarized in this equation to compute the message from node j to node i:

$$\text{message}_{ji}(x_i) = \sum_{x_j} \phi_{ij}(x_i, x_j) \prod_{k \in \eta(j) \setminus i} \text{message}_{kj}(x_j) \tag{2.2}$$

where $\eta(j)\setminus i$ refers to the neighbors of node j except for node i.

After computing the messages, we can use the messages to update the node. The marginal probability at a node is the product of all incoming messages at that node

$$o_i(x_i) = \prod_{j \in n(i)} \text{message}_{ji}(x_i) \tag{2.3}$$

In Chapter 3, we will explore the deep learning model built by parameterizing this message passing scheme for efficient modeling of relative attributes.

2.3 Linear Programming and Physarum Dynamics

Linear programming

Since linear programming was first proposed by Dantzig (1951) as a way for specialists to set certain objectives and solve for a detailed schedule to meet the goals, it has been widely adopted in many areas Dantzig and Thapa (1997). It has many nonlinear and integer extensions such as integer programming Conforti et al. (2014); Wolsey (2020), nonlinear programming Bertsekas (1997); Kuhn and Tucker (2014), stochastic programming Prékopa (2013), combinatorial optimization Schrijver (2005); Murphy et al. (2013), etc. These together form the mathematical programming field. Linear programming can be thought of as a special case of this more general mathematical programming: Linear programming seeks to minimize or maximize a linear objective function subject to linear equality and inequality constraints Dantzig and Thapa (1997). The size of linear programming problems varies from small to large.

In Dantzig and Thapa (1997), the number of constraints less than 1000 is considered small, between 1000 and 2000 is considered medium and greater than 2000 is considered large. In the context of machine learning, these numbers are an order of magnitude (or more) bigger since the modern size datasets are much larger.

Many problems in real life can be cast as linear programming. Here we can take the simple Knapsack problem as an example: Suppose we have n items, each having a weight w_i and a profit c_i. How should we choose a subset of the items such that the total weight is less than W and the total profit is maximized? To answer this question, we can solve the following linear programming:

$$\max c^T z \quad \text{s.t. } w^T z \leqslant W \tag{2.4}$$

where z is the vector indicating the choice of each item. After introducing the basic background of linear programming, let us look at an interesting scheme to solve it.

Physarum dynamics

Physarum refers to a single celled organism ALDRICH and DANIEL (1982). It has been found to be very interesting by both biologists and computer scientists because of its ability to solve complex optimization problems. Nakagaki et al. (2000) conducted an experiment showing that the slime mold could solve the shortest path problem on a maze. The solution process can be captured using dynamical systems, leading to physarum dynamics. Straszak and Vishnoi (2015) proved that this highly nonlinear scheme can solve the linear programming problem with convergence guarantees.

Let us consider a linear program in the standard form

$$\min c^T z \quad \text{s.t. } Az = b,\ z \geqslant 0 \tag{2.5}$$

where $A \in Z^{m\times n}$, $z \in Z^n_{>0}$ and $b \in Z^m$ and which has a feasible solution. The physarum dynamics for solving linear programming proceeds as follows:

Consider any vector $z \in R^n$ with $z > 0$ and let W be the diagonal matrix with entries z_i/c_i. Let $O = AWA^T$ and $p \in R^n$ is the solution to the linear system $Op = b$. Denote $q = WA^T p$. The physarum dynamics for the linear program given by (A, b, c) then is

$$\dot{z} = q - z \tag{2.6}$$

This can be rewritten as

$$\dot{z} = W(A^T L^{-1} b - c) \tag{2.7}$$

This dynamical system has an initial condition of the form $z(0) = s$ for some $s > 0$. Note that it is not required that s is feasible and Straszak and Vishnoi (2015) shows that the procedure can work with any initial condition

Theorem 2.1 (Existence of Solution Straszak and Vishnoi (2015))**.** *For any initial condition $s \in R^n_{>0}$, the physarum dynamics $\dot{z} = q - z$ has a unique solution $z : [0, \infty) \to R^n_{>0}$ with $z(0) = s$.*

After we know that the solution exists, Straszak and Vishnoi (2015) further shows that no matter where one starts the physarum dynamics from, as long as it is in the positive orthant, one converges to an optimal solution of the linear program.

Theorem 2.2 (Convergence to an Optimal Solution Straszak and Vishnoi (2015))**.** *For any initial condition $s \in R^n_{>0}$, consider the solution $z : [0, \infty] \to R^n_{>0}$ to the physarum dynamics with $z(0) = s$. Denote by z^* an optimal solution of the considered linear program. Then:*

1. $|c^T z(t) - c^T z^*| = O(e^{-\alpha t})$ *for some positive α, which only depends upon A, b, c and s,*

2. *the limit $z^\infty = \lim_{t\to\infty} z(t)$ exists, is a feasible point and $c^T z^\infty = c^T z^*$.*

Note that this theorem also proves that the limits of trajectories of the physarum exist even when the optimal solution is not unique. With the convergence results of the continuous case in hand, let us look at the discretization case which is often needed in practice.

Theorem 2.3 (Convergence Time of Discrete Physarum Dynamics Straszak and Vishnoi (2015))**.** *Consider the discretization of the physarum dynamics, i.e., $z(k+1) = (1 - h)z(k) + hq(k)$. Suppose we initialize the physarum algorithm with $z(0) = s$, i.e., $As = b$ and $M^{-1} \leqslant s_i \leqslant M$ for every $i = 1, ..., n$ and some $M \leqslant 1$. Assume additionally that $c^T s \leqslant M \cdot opt$. Choose any $\epsilon > 0$ and let $h = \frac{1}{6} \cdot \epsilon \cdot C_s^{-2} \cdot D^{-2}$. Then after $k = O(\frac{\ln M}{\epsilon^2 h^2})$ steps $z(k)$ is a feasible solution with: $opt \leqslant c^T z(k) \leqslant (1 + \epsilon) \cdot opt$.*

This theorem gives us the convergence guarantee of the discrete physarum dynamics. In chapter 4, we will utilize a slightly modified version of this discrete

physarum dynamics to construct a differentiable layer for solving linear programming.

2.4 Newton Method for Solving Linear Programming

Newton method

The Newton method, also known as the Newton Raphson algorithm, is an iterative procedure that can be used to find the roots of a differentiable function f. When f is twice differentiable, one common use case is to use Newton method to find the roots of the first order derivative f', i.e., find x where $f(x) = 0$, which are the critical points of f. Thus Newton method can used for nonlinear function minimization. Specifically, given a twice differentiable function $f : R \to R$, we want to minimize f:

$$\min_{x \in R} f(x) \tag{2.8}$$

The update rule of the Newton method can be computed by looking at the second order Taylor expansion:

$$f(x_k + t) \approx f(x_k) + f'(x_k)(t - x_k) + \frac{1}{2} f''(x_k)(t - x_k)^2 \tag{2.9}$$

The next iterate x_{k+1} is computed by minimizing this expansion as a (quadratic) function of t. Thus we can set the derivative of t to be zero

$$0 = \frac{d}{dt}(f(x_k) + f'(x_k)(t - x_k) + \frac{1}{2} f''(x_k)(t - x_k)^2) = f'(x_k) + f''(x_k)t \tag{2.10}$$

then the minimum is achieved at $t = -\frac{f'(x_k)}{f''(x_k)}$ Thus the updating rule is

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)} \tag{2.11}$$

When we work with a vector $x \in R^n$, this update rule becomes

$$x_{k+1} = x_k - \nabla^2 f^{-1} \nabla f \tag{2.12}$$

A Newton Method for Solving Linear Programming

In real life applications, it is quite common to observe the case where the linear program has a very large number of constraints and a moderate number of variables. Standard LP solvers like the physarum dynamics mentioned above may fail on these cases. Mangasarian (2004) proposes a fast Newton method to tackle this problem. Before introducing the algorithm, we first briefly describe the observation made by Mangasarian (2004) that motivates their proposed algorithm.

Consider the primal linear program

$$\min_{x \in R^n} c^T x, \quad \text{s.t. } Ax \leqslant b \tag{2.13}$$

where $c \in R^n$, $A \in R^{m \times n}$, $b \in R^m$, and its dual

$$\max_{u \in R^m} -b^T u = -\min_{u \in R^m} b^T u, \quad \text{s.t. } A^T u + c = 0,\ u \geqslant 0 \tag{2.14}$$

The parametric exterior penalty formulation of the primal linear program for a fixed positive penalty value ϵ can be written as a unconstrained minimization problem

$$\min_{x \in R^n} f(x) \tag{2.15}$$

where f is the penalty function

$$f(x) = \min_{x \in R^n} \epsilon c^T x + (1/2)\|(Ax - b)_+\|^2 \tag{2.16}$$

Mangasarian (2004) shows that the unique least 2-norm solution to the dual linear program is given by

$$v = (1/\epsilon)(Ay - b)_+ \tag{2.17}$$

where y is a solution of the primal penalty problem $\min_{y \in R^n} f(y) = \epsilon c^T x + (1/2)\|(Ax - b)_+\|^2$ (we refer readers to Mangasarian (2004) for the proof). Thus we could use Newton method to first solve v and then get our final solution to the primal linear program.

The algorithm proposed by Mangasarian (2004) proceeds as follows: Set the parameter value ϵ, δ and tolerance tol (typically 10^{-3}, 10^{-4}, and 10^{-12} respectively). Start with any $y^0 \in R^n$. For $i = 0, 1, ...$, do the following steps.

1. Compute $y^{i+1} = y_i - \lambda_i(\partial^2 f(y_i) + \delta I)^{-1} \nabla f(y^i) = y^i + \lambda_i d^i$ (let $d := -(\partial^2 f(y_i) +$

$\delta I)^{-1}\nabla f(y^i)$), where the Armijo stepsize Nocedal and Wright (1999) $\lambda_i = max\{1, 1/2, 1/4, ...\}$ is such that

$$f(y^i) - f(y^i + \lambda_i d^i) \geqslant -(\lambda_i/4)\nabla f(y^i)^T d^i, \tag{2.18}$$

and d^i is the modified Newton direction, $d^i = -(\partial^2 f(y^i) + \delta I)^{-1}\nabla f(y^i)$.

2. Stop if $\|y^i - y^{i+1} \leqslant tol\|$. Else, set $i = i + 1$ and go to step 1.

3. Define the least 2-norm dual solution v as $v = (1/\epsilon)(Ay^{i+1} - b)_+$ and a solution z of the primal linear program by $A_j z = b_j$, $j \in S = \{j | v_j > 0, j = 1, ..., m\}$.

Mangasarian (2004) also shows a convergence result for this algorithm which proves the exactness of the solution z provided that the submatrix A_v selected by v from A has linearly independent columns.

In chapter 5, we utilize a modified version of this algorithm to construct a differentiable layer for solving linear programs with large number of constraints, and explore its use in optimizing non-decomposable objectives.

2.5 Canonical Correlation Analysis

Canonical correlation analysis (CCA), first proposed by Hotelling (1992), is a way to infer relationships between two sets of variables. In multivariate statistical analysis, the data comprises multiple variables measured for a set of individuals. In the case of CCA, the variables of an observation can be partitioned into two sets that can be seen as the two views of the data.

CCA can be defined as the problem of finding two sets of basis vectors, one for x and one for y, such that the correlations between the projections of the variables onto these basis vectors are mutually maximized Borga (2001). We can first look at the case of solving top-1 CCA for simplicity, meaning that we only solve for the pair of basis vectors corresponding to the largest canonical correlation value: consider the linear combinations $x' = x^T w_x$ and $y' = y^T w_y$ of the two variables respectively,

where $x, y, w_x, w_y \in R^n$. The function to be maximized is

$$\rho = \frac{E[x'y']}{\sqrt{E[x'^2]E[y'^2]}} \tag{2.19}$$

$$= \frac{E[w_x^T x y^T w_y]}{\sqrt{E[w_x^T x x^T w_x]E[w_y^T y y^T w_y]}} \tag{2.20}$$

$$= \frac{w_x^T C_{xy} w_y}{\sqrt{w_x^T C_{xx} w_x w_y^T C_{yy} w_y}} \tag{2.21}$$

The maximum of ρ with respect to w_x and w_y is the maximum canonical correlation. The subsequent canonical correlations are uncorrelated for different solutions, i.e.,

$$\begin{cases} E[x'_i x'_j] = E[w_{x'i}^T x x^T w_{x'j}] = w_{x'i}^T C_{xx} w_{x'j} = 0 \\ E[y'_i y'_j] = E[w_{y'i}^T y y^T w_{y'j}] = w_{y'i}^T C_{yy} w_{y'j} = 0 \quad \text{for } i \neq j. \\ E[x'_i y'_j] = E[w_{x'i}^T x y^T w_{y'j}] = w_{x'i}^T C_{xy} w_{y'j} = 0 \end{cases} \tag{2.22}$$

The projections onto w_x and w_y, i.e., x' and y' are called canonical variates.

In practice, we often utilize the following CCA formulation corresponding to the case where we have a finite number of samples and want to extract top-k canonical correlations (k depending on the use case). Let $X \in R^{N \times d_x}$ and $Y \in R^{N \times d_y}$ be N samples respectively drawn from the pair of random variables X and Y, with unknown joint probability distribution. The goal is to find the projection matrices $U \in R^{d_x \times k}$ and $V \in R^{d_y \times k}$, with $k \leqslant \min d_x, d_y$, such that the correlation is maximized:

$$\max_{U,V} F = \text{trace}(U^T C_{XY} V) \tag{2.23}$$

$$\text{s.t. } U^T C_X U = I_k, \; V^T C_Y V = I_k \tag{2.24}$$

Here, $C_X = \frac{1}{N} X^T X$ and $C_Y = \frac{1}{N} Y^T Y$ are the sample covariance matrices, and $C_{XY} = \frac{1}{N} X^T Y$ denotes the sample cross-covariance.

To compute the solution of CCA, the most common way is to utilize the closed-form solution shown by Golub and Zha (1995) utilizing the whitened covariance. Let us define the whitened covariance $T = C_X^{-1/2} C_{XY} C_Y^{-1/2}$ and Φ_k (and Ψ_k) contains the top-k left (and right) singular vectors of T. Golub and Zha (1995) shows that the optimum of the CCA optimization is achieved at $U^* = C_X^{-1/2} \Phi_k$, $V^* = C_Y^{-1/2} \Psi_k$.

This procedure is simple but is only feasible when data matrices are small. In

Chapter 6, we propose a stochastic approach based on differential geometry to compute the solution of CCA which allows large datasets for modern applications. Thus next we give a brief review of relevant differential geometry concepts since certain differential geometry components will play a role in our algorithm.

Differential geometry concepts

Here we only include a condensed description – needed for our algorithm and analysis in Chapter 6 – and refer the interested reader to Boothby (1986) for a comprehensive and rigorous treatment of the topic.

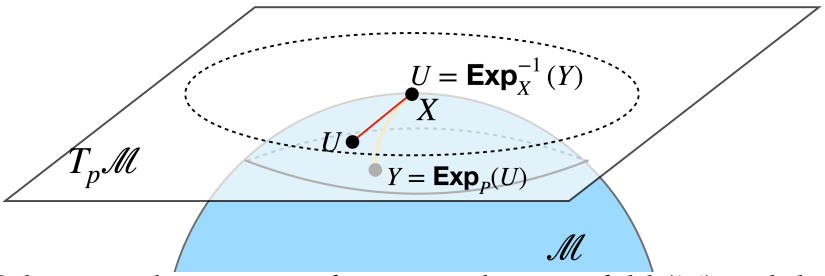


Figure 2.2: Schematic description of an exemplar manifold ($\mathcal{M}$) and the visual illustration of Exp and Exp^{-1} map.

Riemannian Manifold: A Riemannian manifold, $\mathcal{M}$, (of dimension m) is defined as a (smooth) topological space which is locally diffeomorphic to the Euclidean space R^m. Additionally, $\mathcal{M}$ is equipped with a Riemannian metric g which can be defined as

$$g_X : T_X\mathcal{M} \times T_X\mathcal{M} \to R,$$

where $T_X\mathcal{M}$ is the tangent space at X of $\mathcal{M}$, see Fig. 2.2.

If $X \in \mathcal{M}$, the Riemannian Exponential map at X, denoted by $\text{Exp}_X : T_X\mathcal{M} \to \mathcal{M}$ is defined as $\gamma(1)$ where $\gamma : [0,1] \to \mathcal{M}$. We can find γ by solving the following differential equation:

$$\gamma(0) = X, (\forall t_0 \in [0,1]) \frac{d\gamma}{dt}\Big|_{t=t_0} = U.$$

In general Exp_X is not invertible but the inverse

$$\text{Exp}_X^{-1} : \mathcal{U} \subset \mathcal{M} \to T_X\mathcal{M}$$

is defined only if $\mathcal{U} = \mathcal{B}_r(X)$, where r is called the *injectivity radius* Boothby (1986) of $\mathcal{M}$. This concept will be useful to define the mechanics of gradient descent on the manifold.

In our reformulation used in chapter 6, we made use of the following manifolds, specifically, when decomposing U and V into a product of several matrices.

(a) $St(p, n)$: the **Stiefel** manifold consists of $n \times p$ column orthonormal matrices

(b) $Gr(p, n)$: the **Grassman** manifold consists of p-dimensional subspaces in R^n

(c) $SO(n)$, the manifold/group consists of $n \times n$ **special orthogonal matrices**, i.e., space of orthogonal matrices with determinant 1.

Differential Geometry of $SO(n)$: $SO(n)$ is a compact Riemannian manifold, hence by the Hopf-Rinow theorem, it is also a geodesically complete manifold Helgason (2001). Its geometry is well understood – we recall a few relevant concepts here and note that Helgason (2001) includes a more comprehensive treatment.

$SO(n)$ has a Lie group structure and the corresponding Lie algebra, $\mathfrak{so}(n)$, is defined as,

$$\mathfrak{so}(n) = \{W \in R^{n\times n} | W^T = -W\}.$$

In other words, $\mathfrak{so}(n)$ (the set of Left invariant vector fields with associated Lie bracket) is the set of $n \times n$ anti-symmetric matrices. The Lie bracket, $[,]$, operator on $\mathfrak{so}(n)$ is defined as the commutator, i.e.,

$$\text{for } U, V \in \mathfrak{so}(n), \quad [U, V] = UV - VU.$$

Now, we can define a Riemannian metric on $SO(n)$ as follows:

$$\langle U, V\rangle_X = \text{trace}\left(U^T V\right), \quad \text{where}$$

$$U, V \in T_X(SO(n)), X \in SO(n).$$

It can be shown that this is a bi-invariant Riemannian metric. Under this bi-invariant metric, now we define the Riemannian exponential and inverse exponential map as follows. Let, $X, Y \in SO(n)$, $U \in T_X(SO(n))$. Then,

$$Exp_X^{-1}(Y) = X \log(X^T Y)$$

$$\mathrm{Exp}_X(U) = X \exp(X^T U),$$

where, exp, log are the matrix exponential and logarithm respectively.

Differential Geometry of the Stiefel manifold: The set of all full column rank $(n \times p)$ dimensional real matrices form a Stiefel manifold, $\mathrm{St}(p, n)$, where $n \geqslant p$.

A compact Stiefel manifold is the set of all column orthonormal real matrices. When $p < n$, $\mathrm{St}(p, n)$ can be identified with

$$\mathrm{SO}(n)/\mathrm{SO}(n-p).$$

Note that, when we consider the quotient space, $\mathrm{SO}(n)/\mathrm{SO}(n-p)$, we assume that $\mathrm{SO}(n-p) \simeq \iota(\mathrm{SO}(n-p))$ is a subgroup of $\mathrm{SO}(n)$, where,

$$\iota : \mathrm{SO}(n-p) \to \mathrm{SO}(n)$$

defined by

$$X \mapsto \begin{bmatrix} I_p & 0 \\ 0 & X \end{bmatrix}$$

is an isomorphism from $\mathrm{SO}(n-p)$ to $\iota(\mathrm{SO}(n-p))$.

Differential Geometry of the Grassmannian Gr(p, n): The Grassmann manifold (or the Grassmannian) is defined as the set of all p-dimensional linear subspaces in R^n and is denoted by $\mathrm{Gr}(p, n)$, where $p \in \mathbf{Z}^+$, $n \in \mathbf{Z}^+$, $n \geqslant p$. Grassmannian is a symmetric space and can be identified with the quotient space

$$\mathrm{SO}(n)/S\left(\mathrm{O}(p) \times \mathrm{O}(n-p)\right),$$

where $S\left(\mathrm{O}(p) \times \mathrm{O}(n-p)\right)$ is the set of all $n \times n$ matrices whose top left $p \times p$ and bottom right $n-p \times n-p$ submatrices are orthogonal and all other entries are 0, and overall the determinant is 1.

A point $\mathcal{X} \in \mathrm{Gr}(p, n)$ can be specified by a basis, X. We say that $\mathcal{X} = \mathrm{Col}(X)$ if X is a basis of $\mathcal{X}$, where $\mathrm{Col}(.)$ is the column span operator. It is easy to see that the general linear group $\mathrm{GL}(p)$ acts isometrically, freely and properly on $\mathrm{St}(p, n)$. Moreover, $\mathrm{Gr}(p, n)$ can be identified with the quotient space $\mathrm{St}(p, n)/\mathrm{GL}(p)$. Hence, the projection map

$$\Pi : \mathrm{St}(p, n) \to \mathrm{Gr}(p, n)$$

is a *Riemannian submersion*, where $\Pi(X) \triangleq \text{Col}(X)$. Moreover, the triplet $(\text{St}(p,n), \Pi, \text{Gr}(p,n))$ is a fiber bundle.

Horizontal and Vertical Space: At every point $X \in \text{St}(p,n)$, we can define the *vertical space*, $\mathcal{V}_X \subset T_X\text{St}(p,n)$ to be $\text{Ker}(\Pi_{*X})$. Further, given g^{St}, we define the *horizontal space*, $\mathcal{H}_X$ to be the g^{St}-orthogonal complement of $\mathcal{V}_X$.

Horizontal lift: Using the theory of principal bundles, for every vector field $\widetilde{U}$ on $\text{Gr}(p,n)$, we define the *horizontal lift* of $\widetilde{U}$ to be the unique vector field U on $\text{St}(p,n)$ for which $U_X \in \mathcal{H}_X$ and $\Pi_{*X}U_X = \widetilde{U}_{\Pi(X)}$, for all $X \in \text{St}(p,n)$.

Metric on Gr: As, Π is a Riemannian submersion, the isomorphism $\Pi_{*X}|_{\mathcal{H}_X} : \mathcal{H}_X \to T_{\Pi(X)}\text{Gr}(p,n)$ is an isometry from $(\mathcal{H}_X, g^{St}_X)$ to $(T_{\Pi(X)}\text{Gr}(p,n), g^{Gr}_{\Pi(X)})$. So, $g^{Gr}_{\Pi(X)}$ is defined as:

$$\begin{aligned} g^{Gr}_{\Pi(X)}(\widetilde{U}_{\Pi(X)}, \widetilde{V}_{\Pi(X)}) &= g^{St}_X(U_X, V_X) \\ &= \text{trace}((X^TX)^{-1}U_X^TV_X) \end{aligned} \tag{2.25}$$

where, $\widetilde{U}, \widetilde{V} \in T_{\Pi(X)}\text{Gr}(p,n)$ and $\Pi_{*X}U_X = \widetilde{U}_{\Pi(X)}$, $\Pi_{*X}V_X = \widetilde{V}_{\Pi(X)}$, $U_X \in \mathcal{H}_X$ and $V_X \in \mathcal{H}_X$.

We covered the exponential map and the Riemannian metric above, and their explicit formulation for manifolds listed above is provided for easy reference in Table 2.1.

	$g_X(U,V)$	$\text{Exp}_X(U)$	$\text{Exp}_X^{-1}(Y)$
$\text{St}(p,n)$ Kaneko et al. (2012)	$\text{trace}\left(U^TV\right)$	$\widetilde{U}\widetilde{V}^T$, $\widetilde{U}\widetilde{S}\widetilde{V}^T = \text{svd}(X+U)$	$(Y-X) - X(Y-X)^TX$
$\text{Gr}(p,n)$ Absil et al. (2004)	$\text{trace}\left(\Pi_*^{-1}(U)^T\,\Pi_*^{-1}(V)\right)$	$\widehat{U}\widehat{V}^T$, $\widehat{U}\widehat{S}\widehat{V}^T = \text{svd}(\bar{X}+U)$	$\bar{Y}\left(\bar{X}^T\bar{Y}\right)^{-1} - \bar{X}$, $X = \Pi(\bar{X}), Y = \Pi(\bar{Y})$
$\text{SO}(n)$ Subbarao and Meer (2009)	$\text{trace}\left(X^TUX^TV\right)$	$X\text{expm}\left(X^TU\right)$	$X\text{logm}\left(X^TY\right)$

Table 2.1: Explicit forms for some operations we need. $\Pi(X)$ returns X's column space; Π_* is Π's differential.

2.6 Stochastic Differential Equation

Differential equations

A differential equation is an equation consisting of a function and its derivatives. Usually, the function is unknown and the notion of a solution in the context of the differential equation refers to a function that satisfies this equation and is differentiable. For a concrete example, $x' - x = 0$ is a differential equation. Here x is the "unknown function" with respect to the time variable t. One solution can be obtained by observing that $(e^t)' - e^t = e^t - e^t = 0$. Thus we know that $x(t) = e^t$ is a solution of this differential equation.

Types of differential equations. Differential equations can be classified into many types depending on their own characteristics. One commonly used way to categorize a differential equation is by whether it contains partial derivatives. If a differential equation does not involve partial derivatives, then it is called an *ordinary differential equation*, otherwise, it is called an *partial differential equation*. Another way to distinguish differential equations is to use order. The order of a differential equation refers to the order of the highest derivative in the equation. For example, $x'' - 4x' + 2x = 0$ is a second order differential equation while $x' - x = 0$ is first order. There are also other ways to classify a differential equation, like whether it contains nonlinear functions, or whether it contains stochastic terms, etc.

The use cases of differential equations exist in a much broader scope than mathematics. Differential equations act as an important tool for most areas of science and engineering. A differential equation may describe the movement of some microorganism, the motion of a car, the development of economics in an area, or the spread of news.

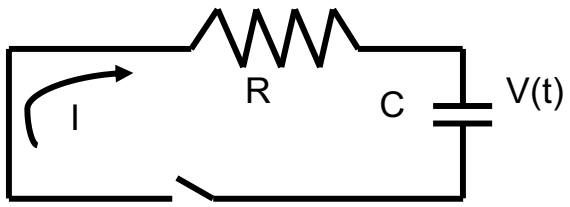


Figure 2.3: An RC circuit example for illustrating the use of differential equations.

For example, Fig. 2.3 shows an RC circuit with resistance R and capacity C without considering external current or voltage source Ahmad and Ambrosetti (2015).

Here we show how to analyze the current in this circuit utilizing the differential equation as a tool. Let us denote the capacity voltage as $x(t)$ and the current in the circuit as $I(t)$. We then have

$$R \cdot I(t) + x(t) = 0 \tag{2.26}$$

In addition, according to the constitutive law of capacitor we have

$$I(t) = C \cdot \frac{dx(t)}{dt} \tag{2.27}$$

Inserting the formulation of $I(t)$ into the previous equation, we obtain the following differential equation

$$RC \cdot x'(t) + x(t) = 0 \tag{2.28}$$

This is equivalent to

$$x'(t) + \frac{x(t)}{RC} = 0 \tag{2.29}$$

The solution to this differential equation can be given by

$$x(t) = x_0 e^{-t/RC} \tag{2.30}$$

This indicates that the voltage $x(t) = V(t)$ decays exponentially to 0 as $t \to +\infty$, which matches our real-life experience. We can also find the intensity of current $I(t)$

$$I(t) = Cx'(t) = -C \cdot \frac{x_0}{RC} e^{-t/RC} = -\frac{x_0}{R} e^{-t/RC} \tag{2.31}$$

After the brief introduction of differential equations above, next we discuss a special type of differential equation which involves stochastic terms and will be utilized in our Chapter 7.

Stochastic differential equation

The stochastic differential equation (SDE) is a kind of differential equation where one or more of the terms is a stochastic process. Accordingly the solution of the equation will be a random function. The importance of stochastic differential equations has become clear for many problems in physics, chemistry, engineering Van Kampen (1976) and recently machine learning Liu et al. (2019c); Fraccaro et al. (2016). In principle every differential equation that purports to describe a physical system should be

replaced with a stochastic one, in order to take into account the inevitable perturbations due to interactions with the surroundings Van Kampen (1976), which inspires us to bring the stochasticity to general deep learning models.

One frequently used formulation is the Îto's stochastic differential equation:

$$d\underline{X}_t = \underline{b}(\underline{X}_t)dt + \sigma(\underline{X}_t)d\underline{W}_t \tag{2.32}$$

where $\underline{X}_t$ is the variable, $\underline{W}_t$ is a (multidimensional) Brownian motion with some covariance, $\underline{b}$ represents the drift term and σ represents the diffusion function. To compute $\underline{X}_T$ at some time point T, we need to perform the integration from $t = 0$ to $t = T$. In many applications we may be interested in some function f evaluated at $\underline{X}_T$. Note that the $f(\underline{X}_T)$ is still a stochastic function (suppose f is deterministic) and we may need the expectation $E[f(\underline{X}_T)]$. Then the computation becomes

$$E[f(\underline{X}_T)] = E[f(\underline{X}_0 + \int_{t=0}^{t=T} \underline{b}(\underline{X}_t)dt + \int_{t=0}^{t=T} \sigma(\underline{X}_t)d\underline{W}_t)] \tag{2.33}$$

Infinitesimal generator

This above computation may be very costly when f is computationally heavy. One useful concept here is the *infinitesimal generator* which we will describe in more detail in chapter 7. Given a stochastic process $\underline{X}_t$, the infinitesimal generator $\mathcal{L}$ of a function f is defined as

$$\mathcal{L}f(\underline{X}) = \lim_{t\to 0} \frac{E[f(\underline{X}_t)] - f(\underline{X}_0)}{t} \tag{2.34}$$

In Chapter 7 we explore bringing stochasticity into deep models by instantiating the infinitesimal generator of an SDE there.

Chapter 3

Efficient Relative Attribute Learning using Graph Neural Networks

Graph structured data appear naturally in many real-life applications which standard convolutional layers have difficulty modeling directly and the visual attribute learning studied in this chapter provides one specific example. In this chapter, we show how the graph structure emerges from the data and how a dynamical system (message passing perspective) can help build an efficient model for this problem. The work presented in this chapter appeared as a conference paper at ECCV 2018 Meng et al. (2018).

3.1 Introduction

Visual attributes Farhadi et al. (2009) correspond to mid-level semantic and even non-semantic concepts or properties of the image or objects contained in the image that are interpretable by humans. For instance, an image can be "natural", "smiling" or "furry" depending on the properties of the key entities contained in it. The ability to associate such attributes with images has enabled systems to perform better in traditional categorization tasks, and even go beyond basic level naming Parikh and Grauman (2011). The insight in this line of work is to first select features that can predict attributes for the object class of interest – the subsequent classifier must then leverage only those "relevant" features since material properties or shape may be differentially important for different categories. The concept of "relative attributes" takes this idea further Parikh and Grauman (2011) by arguing that the strength of

an attribute in an image is best judged in the context of its strength with respect to all other images in the training data rather than as a binary concept. For example, while it is difficult to characterize how "man-made" an image is, one could set up a comparison where humans compare the images in terms of *this* attribute. This strategy of describing images in relative terms works well in challenging cases Jamieson et al. (2015) – for instance, calculating how "open" an image is versus another.

Since the early works on relative attributes Souri et al. (2016); Singh and Lee (2016); Xiao and Jae Lee (2015), several papers have proposed more task-specific models for ranking based on specialized features. But given the success of convolutional neural networks (CNN) architectures, most recent proposals utilize CNNs for feature learning in the context of learning the overall ranking. For instance, given a set of annotated image-pairs with respect to one/more attributes, the network learns weights that are maximally consistent with the attribute-specific ranking of the images. Related ideas have also explored designing image-part specific detectors, that are aligned to an attribute. For instance, what is the spatial support for an attribute such as "smiling". Clearly, this will involve localizing the visual concept to a part of the image, say the mouth or lips region. In Singh and Lee (2016), the authors transitively connect the visual chains across the attribute continuum and make the case that feature extraction and ranking should not be performed separately.

The starting point of our work is the observation that the space of attributes which induce a ranking over the images share a great deal of correlational structure. For instance, the attribute "furry" may be associated with the attribute "four-legged"

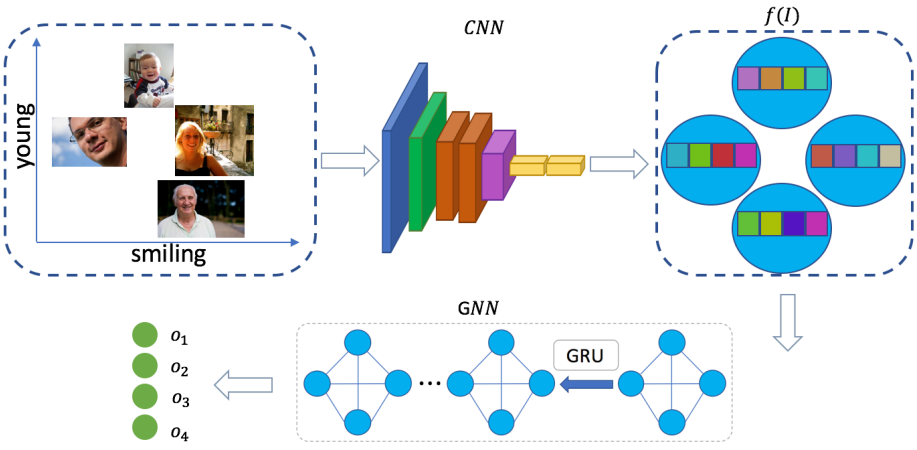


Figure 3.1: Overview of our framework for relative attribute learning.

and the attribute "congested" may have some information to provide to the attribute "man-made". This induces a natural graph of attributes and images, where the input data provides either pair-wise relationships between images for one/more attributes or a partial (or full) ranking of the images for the attribute. We do not assume that the annotation is exhaustive – many edges (or relationships) between the images may, in fact, be unavailable. Extending recent work on graph neural networks (GNNs) which extends the notion of convolution and other basic deep learning operations to non-Euclidean grids by parameterizing the message passing dynamics on graph nodes as layers (Gori et al. (2005); Scarselli et al. (2009); Li et al. (2015); Gilmer et al. (2017); Bronstein et al. (2017)), we show how these ideas yield a natural model for learning on this graph involving image↔attribute and image↔image edges. Not only are the image features (relevant for each attribute) extracted automatically but we also concurrently learn the similarity function that is most consistent with the given pair-wise annotations as well as the latent relationships between the attributes (similar to multi-task learning). This machinery is simple, yet performs competitively with more specialized proposals on several different problems.

Our contributions. (1) we formulate and solve relative attribute learning via message passing dynamics on a graph, where the convolutional layers, ranking as well as imputation of unseen relationships is performed concurrently. (2) our framework yields results similar to the best reported for each task (at the time this research was undertaken) with minimal change, often providing sizable reduction in the number of parameters to be estimated or with far less stringent requirements on the training data annotations. We note that GNNs were independently used in a classification task in a paper made available on arXiv Garcia and Bruna (2017).

3.2 Related Work

Visual attributes. Visual attributes are semantic properties in images which can be understood by humans and are shared among all images of similar categories (e.g., all images of human faces share the attribute "smiling", whose strength can vary from weak to strong as we will show with examples shortly. Most existing works in visual attributes focus on binary attribute prediction (BAP) where each attribute is predicted from a given image and cast as a binary classification problem. "Relative attributes" were proposed in Parikh and Grauman (2011) and have been ex-

plored in a number of settings Singh and Lee (2016); Souri et al. (2016); Xiao and Jae Lee (2015). Several current techniques use deep neural networks to learn relative attributes (e.g.,Souri et al. (2016)), and also borrow ideas from attention mechanism research (e.g.,Singh and Lee (2016)) to help the networks focus only on the most informative areas in the images. Most of these works deal with a pair of images at a time. Our work shows that dealing with groups of images on a fully connected graph instead of just pairwise comparisons improves performance.

Multi-task learning. Multi-task learning is intended to achieve knowledge sharing by learning several correlated tasks at the same time. This technique has recently been used in binary attribute prediction. Learning several correlated attributes together can improve performance, and this has been demonstrated by some recent works Abdulnabi et al. (2015); Han et al. (2017); Wang et al. (2017). Abdulnabi et al. Abdulnabi et al. (2015) propose a multi-task CNN framework which improves accuracy compared with learning one attribute at a time. Wang et al. Wang et al. (2017) designed a simpler deep multi-task network for prediction of face attributes. In contrast to most strategies related to multi-task learning, our multi-task formulation learns attributes simultaneously and is shown to benefit relative attribute learning.

Graph neural networks (GNN). Graph neural networks were proposed by Gori et al. (2005); Scarselli et al. (2009), where the authors describe GNN as a parameterized message passing scheme which can be trained. Later, Li et al. (2015) proposed using gated recurrent units (GRUs) within GNNs, which much improves the representation capacity of the network and makes it suitable for graph structured data. Gilmer et al. (2017) generalized the GNN using message passing neural network and demonstrated state-of-the-art results on molecular prediction benchmarks. More recently, concurrent to and independent of our work, Garcia and Bruna (2017) applied GNNs for classification and achieved good results on several different datasets.

3.3 Approach

Our approach is based on the observation that in a relative attribute learning task, different images are correlated and the attributes may or may not be correlated. The learning procedure can benefit from exploring the similarity among multiple images on a graph, where each node represents an image and the edges are formed based

on the relationship between the to-be-learned representations of the nodes. Furthermore, such a graphical structure can benefit multi-task learning where we can add different *types* of nodes to the graph for representing different attributes that are being learned. In this way, we explicitly learn the properties of certain attributes, the interplay between the attributes when necessary, the representations of the images and their relationships on the graph in a way that best informs the task at hand.

We first explain how the input images are mapped into the graph representation, and give the details of our network architecture for relative attribute learning in the context of *one attribute*. Then, we show how the construction can be used to perform multi-task attribute learning with minimal modifications. Finally, we also show how our model can be used for a binary attribute prediction (BAP) task efficiently. The overview of our framework is shown in Fig. 3.2.

Network Architecture

Let X be the dataset of input samples (images) with X_i representing the i-th image, and for a certain attribute (e.g., smile), we assume that a set of pairwise relation-

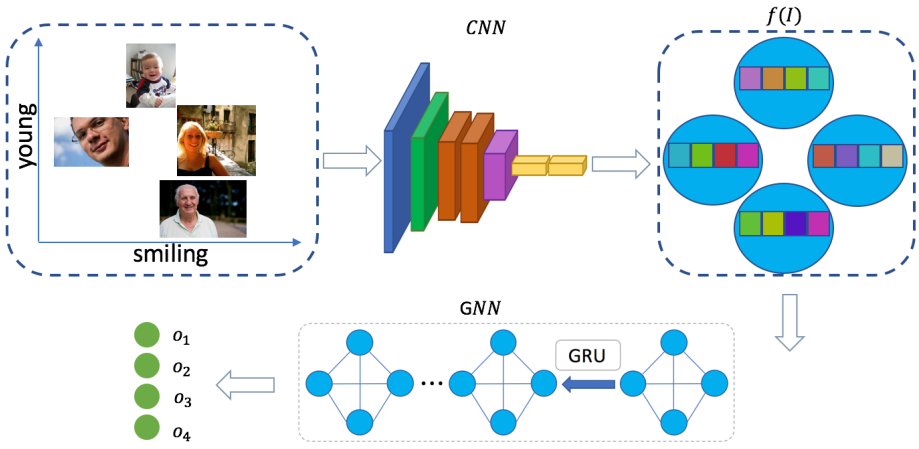


Figure 3.2: Overview of our framework for RAL and BAP tasks. Since many natural attributes of images are interrelated, discovering their common latent representations would be beneficial to the attribute learning tasks. This can be efficiently achieved by mapping these images to a graphical structure. Every image has a corresponding node on the graph and a corresponding output node. The initial features $f(\cdot)$ for the nodes are generated using a CNN on the images and the edge features and following updates are performed using GNNs (details in Fig. 3.3). The weights in the entire framework including those in the CNN and GNN are trained end-to-end.

ship labels $\mathcal{P} = \{\phi(X_i, X_j)\}_{i,j=1;i\neq j}^{n}$, where $\phi(X_i, X_j)$ indicates the relative strength of the attribute t between the two images X_i and X_j. This relationship may be logical (e.g.,"stronger than" or "weaker than"). With this data, a generalized GNN is trained where both the node features (representations of the images) and edge weights are learned. The core architecture of our GNN is shown in Fig. 3.3.

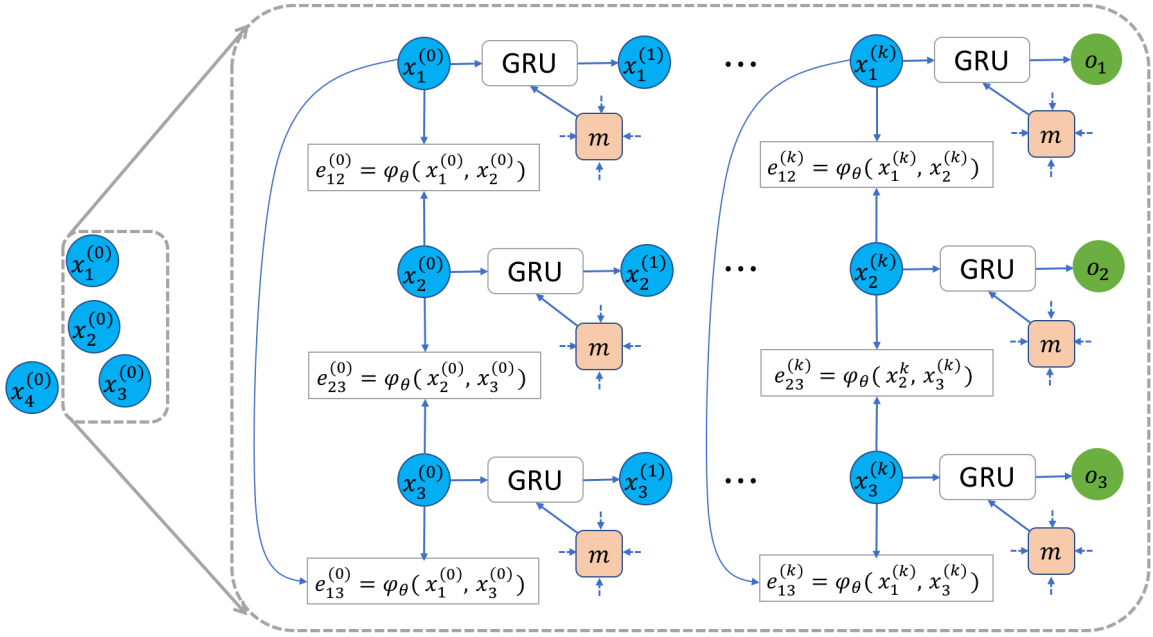


Figure 3.3: The architectural details of our GNN which remains the same for both RAL and BAP. The edges on the graph are learned from adjacent nodes using a parameterized function (φ_ϑ, see (3.3)), which is shared among all edges. The "m" in this figure refers to the message for a node passed from its connected nodes and edges, which is defined in (3.4). Then, a GRU cell takes as input a node and its corresponding message, and outputs the updated node. The parameters in GRU are also shared across all nodes.

Assume that we operate on groups (or mini-batches) of a certain size (which is allowed to vary) sampled with or without replacement from the underlying training dataset. The relationships among all the images in each mini-batch in the training set are represented using a fully-connected graph $\underline{G} = (\underline{V}, \underline{E})$, where each node $\underline{v}_i$ in $\underline{V}$ corresponds to an image X_i in the mini-batch. Each time, the network takes in a group of images and passes them through a convolutional neural network. This may also be thought of as a set of convolutional networks that share weights. The representations derived from this network yield the initial representations of the node

features as

$$x_i^{(0)} = f(X_i), \tag{3.1}$$

where $f(\cdot)$ refers to a CNN which operates on the images. Here, X_i is the input image and $x_i^{(0)}$ is the initial node feature for the image at time $k = 0$. Next, the network learns edge features as,

$$e_{i,j}^{(k)} = \varphi_\vartheta \left(x_i^{(k)}, x_j^{(k)} \right), \tag{3.2}$$

where φ is a symmetric function parameterized with a single layer Neural Network:

$$\varphi_\vartheta(x_i^{(k)}, x_j^{(k)}) = \mathfrak{N}_\vartheta^{\text{edges}} \left(\|x_i^{(k)} - x_j^{(k)}\|_1 \right). \tag{3.3}$$

We assume that φ_ϑ is a metric which is learned by a non-linear combination of the absolute difference between the learned features of the two nodes (or any other simple function involving the node features). This ensures that the symmetric property $\varphi_\vartheta(a, b) = \varphi_\vartheta(b, a)$ is satisfied by design.

Our goal now is to update the belief at each node based on the beliefs at the other nodes in the graph as well as its own state at the previous time point. To accomplish this, we use a message function $M(\cdot)$ to aggregate information from all neighbors of each node. In particular, for each node $x_i^{(k)}$, the message is defined as below,

$$\tilde{m}_i^{(k)} = \sum_{j, j \neq i} M \left(x_j^{(k)}, e_{i,j}^{(k)} \right). \tag{3.4}$$

Here, $M(\cdot)$ is parameterized using a single layer neural network whose details are presented later in section 3.3. We now need to define a mechanism that utilizes the messages received from the node's neighbors and its previous state to update its state. To do so, we use an updating layer $G(\cdot)$ which takes as input a signal $x^{(k)}$ (the current state of the node) and produces $x^{(k+1)}$. This is accomplished using a Gated Recurrent Unit (GRU) as the updating function.

$$x_i^{(k+1)} = G \left(x_i^{(k)}, \tilde{m}_i^{(k)} \right). \tag{3.5}$$

With this setup in hand, we simply use a readout function $o_i = R(x_i)$ to get the output from each node and finally define our loss function based on these outputs

from all relevant nodes on the graph as

$$\text{Loss} = \mathcal{R}\left(\{o_i\}_{i=1}^{n}\right), \tag{3.6}$$

where n is the number of graph nodes. Note that $\mathcal{R}(\cdot)$ can also be parameterized with a simple (or more complicated) neural network depending on the needs of the application. The specific form of $\mathcal{R}(\cdot)$ depends on the specific task, which will be discussed in the following Sections 3.3–3.3.

Learning Relative Attributes, One at a Time

The Relative Attribute Learning (RAL) task seeks to learn a network that, given input images, outputs pairwise labels according to the relative strength of certain attributes between each pair of images. In this section, we consider training a network for one attribute at a time.

Recall that our network is designed to better explore the correlated information among different images. So unlike other approaches in RAL (Souri et al. (2016); Singh and Lee (2016)) which take two images at a time as an input, we sample a group of images from the training set as input at every draw. The size of the group need not be fixed and can vary for learning different attributes in a single dataset or different datasets, because our network has the benefit of weight sharing on the graphical structure of the samples. We use the five convolutional layers and the first two fully-connected layers in `AlexNet` Krizhevsky et al. (2012) (`conv1` through `fc7`) although other architectures can be substituted in. The dimension of the output feature vector of the node is fixed to be 4096.

Messages. We impose a fully-connected graphical structure on the images in each group. After mapping these images on the graph, we perform message passing, which is effective in information propagation among the nodes. We adopt the strategy to *learn* edge features from the current node hidden representation formulated by Gilmer et al. (2017), as suggested in (3.2). The parameters of the edge learning function φ_ϑ are shared among all nodes on the graph. Then for every node $x_i^{(k)}$ on the graph, a message signal will be extracted from all the in-coming nodes through the edges, see (3.4). Here, we specify the message function $M(\cdot)$ as,

$$M(x_j^{(k)}, e_{i,j}^{(k)}) = \text{ReLU}\left(W\left(x_j^{(k)} \| e_{i,j}^{(k)}\right) + b\right), \tag{3.7}$$

where $\|$ denotes the concatenation operator of two vectors, W and b are the weight matrix and the bias respectively, and $\mathrm{ReLU}(\cdot)$ is the rectified linear unit (ReLU) function. We would also like to note that the parameters (W and b) of the message function $M(\cdot)$ are also shared by all nodes and edges in our graph, thus providing an explicit control on the number of parameters.

Updating. Let us now discuss the updating function for nodes. At each iteration, each GRU takes the previous state of the node and an incoming message as input, and produces a new hidden state as the output (see Fig. 3.3). Let $x_i^{(k-1)}$ be the node's hidden representation at the previous time step, $m_i^{(k)}$ be the message received via (3.4), and $x_i^{(k)}$ be the updated node. With these notations, the basic operations of GRU are simply given as,

$$\begin{aligned}
\underline{z}_i^k &= \sigma\left(W^z \tilde{m}_i^{(k)} + U^z x_i^{(k-1)}\right), \\
\underline{r}_i^k &= \sigma\left(W^r \tilde{m}_i^{(k)} + U^r x_i^{(k-1)}\right), \\
\tilde{x}_i^{(k)} &= \tanh\left(W \tilde{m}_i^{(k)} + U\left(\underline{r}_i^k \odot x_i^{(k-1)}\right)\right), \\
x_i^{(k)} &= (1 - \underline{z}_i^k) \odot x_i^{(k-1)} + \underline{z}_i^k \odot \tilde{x}_i^{(k)},
\end{aligned} \tag{3.8}$$

where $\underline{z}$ and $\underline{r}$ are the intermediate variables in the GRU cells, W, Z are weight matrices of size h (the hidden dimension) by h, $\sigma(x) = 1/(1+e^{-x})$ is the sigmoid function and $\odot$ is element-wise multiplication.

Each node in our graph maintains its internal state in the corresponding GRU, and all nodes share the same weights of the GRU, which makes our model efficient while can also seamlessly deal with differently sized groups as input. In this work, we use one time step of GRU updating. During testing time, any number of images are allowed, and the network will output a pairwise label for every two images based on the obtained value of output nodes on the graph. After constructing our graph using (3.1)–(3.6), the loss defined on the output of graph takes the form

$$\mathrm{RALLoss} = \sum_{i,j,i\neq j} -l_{ij}\log(P_{ij}) - (1-l_{ij})\log(1-P_{ij}), \text{where} \tag{3.9}$$

$$l_{ij} = \begin{cases} 1 & \text{if } I_i \succ I_j, \\ 0 & \text{if } I_i \prec I_j, \\ 0.5 & \text{otherwise,} \end{cases}$$

and $P_{i,j} = o_i - o_j$ (outputs of nodes i and j). This formulation has a nice property that it is robust to noise as described in Burges et al. (2005), and symmetric by construction so that we can easily utilize training data where some pairs of images appear with "equal" label for one/more attributes.

Learning Relative Attributes, All at Once

In this section, we show that our graphical structure can be efficiently applied to learn multiple relative attributes all at the same time i.e., perform multi-task attribute learning. We consider two aspects of multi-task learning, (1) the performance of RAL can be improved by utilizing several attributes which have common latent representations. Although this has been demonstrated in binary attribute prediction (BAP) setting, we present experimental results showing that RAL can benefit from multi-task learning. (2) The second aspect is the efficiency of the construction. While multi-task learning can improve the performance when attributes are correlated, in the previous methods Abdulnabi et al. (2015); Wang et al. (2017), the number of parameters of the network grows *much faster* as a function of the number of attributes learned together, which increases the cost of training a multi-task network. As an example, if the number of parameters trained in RAL one at a time is $O(K^2)$ then our version only increases the number to $O(K^2 + nK)$, where n is the number of different relative attributes learned simultaneously. This is much smaller than $O(nK^2)$ which may be needed within other multi-task approaches Abdulnabi et al. (2015); Wang et al. (2017).

We note that a näive way to adapt our network (Fig. 3.3) to the multi-task setting proceeds as follows. We simply change the dimension of the output o_i from 1 to m where m is the number of attributes. But the only change this induces is in the size of the weight matrix in the readout function. We find that in this case, the graphical structure may slightly lose its expressive capacity. To address this issue, unlike section 3.3, which treats all nodes in the graph in the same way, here, we define two different *types* of nodes x_i, $i = 1, 2, \cdots, n$, and r_i, $i = n+1, n+2, \cdots, n+m$, where n is the number of input images in each group (to be consistent, we choose $n = 5$ throughout our experiments) and m equals the number of attributes the network is learning at the same time. Here, x_i has the same meaning as in section 3.3, which corresponds to one image and each r_j corresponds to a certain attribute. It is important to note that while the representation at x_i is learned by the convolutional network,

the attribute node r_j is randomly initialized at the beginning of the training phase and keeps getting updated in a global manner, similar to the other parameters in the GNN.

This scheme allows us to explicitly learn a hidden representation for each attribute in a way that the latent variables of the graphical model are influencing all attribute nodes — this is similar to multi-task learning where we expect that learning related tasks can benefit each other when carried out concurrently. The feature extraction process using the convolutional network and the GNN procedure remain identical as in section 3.4. The only change needed is to redefine how we use the readout function $R(\cdot)$ to get the output. Here, $o_{i,j} = R(\|x_i - r_j\|_1)$, where for $o_{i,j}$, i gives the index of nodes for the images (from 1 to n) and j gives the index of different attributes (from 1 to m). The loss function is then defined as the sum of the loss for each single attribute (see (3.9)) as,

$$\text{RALLoss}_{\text{multi}} = \sum_{i=1}^{m} \text{RALLoss}_i. \tag{3.10}$$

Binary Attribute Prediction

In this section, we present details of how our graphical model can also be used to predict binary attributes with comparable accuracy as the multi-task CNN model Abdulnabi et al. (2015), but using much fewer number of parameters.

Binary attribute prediction (BAP) task seeks to predict whether an image has a certain attribute (e.g., whether a person is wearing a necktie), which can be thought of as a binary classification task. As suggested in papers for multi-task learning Abdulnabi et al. (2015); Wang et al. (2017), simultaneously learning several attributes which are correlated can improve the performance of BAP. In this setting, the labels no longer provide pairwise information. So, it is not simple to easily extend other RAL methods and adapt them for BAP. For example, the construction using Siamese network Singh and Lee (2016) cannot be easily modified for BAP since the subnetworks are no longer linked – this is because it is the pairwise annotations that link the networks. But our network can still benefit from a fully-connected graph structure on the training samples because despite the unavailability of pairwise annotations, the images themselves are still related. So, we can use the *same* basic architecture. The framework before loss layer remains the same as the network in section 3.3. The

loss function for BAP is simply defined as

$$\text{BAPLoss}_i = -l_i \log(P_i) - (1 - l_i) \log(1 - P_i), \tag{3.11}$$

where l_i is the binary label of image X_i, and $P_i = o_i$. The total loss is defined as,

$$\text{BAPLoss}_{\text{multi}} = \sum_{i=1}^{m} \text{BAPLoss}_i. \tag{3.12}$$

3.4 Experimental Results

In this section, we analyze the performance of our model on several different settings described in section 3.3. First, we present some key **implementation details.** Our network takes in a group of images and outputs the pairwise relationships for this group (in a relative attribute task) or a binary label for each image (in a attribute prediction task). We split the train/test set randomly. Then, we randomly split the train/test set into groups (we choose 5 images per group, but the number can vary) and use this as the input to our network. We report the pairwise accuracy measured on the groups of images. In a preprocessing step, we subtract the mean of training set and crop images to size 227×227.

For training, we initialize the `conv1` to `fc7` layers using `AlexNet` pre-trained on the ILSVRC 2012 Krizhevsky et al. (2012) dataset and randomly initialize other parts using the Xavier initializer Glorot and Bengio (2010). We use mini-batches of size 10 and Adam optimizer Kingma and Ba (2014) with $\beta_1 = 0.9$, $\beta_2 = 0.999$. The learning rate of relative attribute learning task is 0.0001, and for attribute prediction task, we set the learning rate to 0.00001.

Relative Attribute Learning, One at a Time

In this experiment, we evaluate the network described in section 3.3. The goal is to compare pairs or sets of images according to the strength of a given attribute. We used the OSR scene dataset Oliva and Torralba (2001) and a subset of the Public Figure Face Dataset (PubFig) Parikh and Grauman (2011). The OSR scene dataset consists of 2,688 images with outdoor scene attributes (`natural`, `open`, `perspective`, `large-objects`, `diagonal-plane` and `close-depth`). The subset of the PubFig contains nearly 800 images from 8 random identities. We split the train/test set ran-

domly and then split the train/test set into groups and use this as the input of network. We report the results in terms of pairwise accuracy on the groups of images.

Our scheme makes it possible to make use of the information in the group of images as a whole, which is more informative than just a pair of images (common to Siamese Networks construction). For a fair comparison with other methods, we measure the performance of our model by computing the pairwise accuracy for all pairs in each group.

We choose two methods for baseline comparisons. The first one is the work of Souri et al. Souri et al. (2016), which trains a deep convolutional network to learn relative attributes for pairs of images. The second one is the DeepPermNet Anoop and Gould, which learns relative attributes by learning permutations. Note that this method needs fully ranked sequences of images as input, which is a more stringent requirement compared to our network and the work of Souri et al. Souri et al. (2016), which only needs pairwise labels during training. The accuracy results are shown in Tables 3.1–3.2. Qualitative results are shown in Fig.3.4.

Table 3.1: Relative attribute learning accuracy evaluated on **OSR** dataset. On average, we outperform all previous methods. The penultimate row presents the results of our network in section 3.4 and the last row presents the results of our multi-task network in section 3.4, which learns all of the six attributes at once.

Method	`natural`	`open`	`diagonal-plane`	`close-depth`	**Mean**
Souri et al.Souri et al. (2016)	99.4	97.44	98.43	97.65	97.77
Cruz et al.(AlexNet)Anoop and Gould	97.21	96.65	94.53	96.09	96.62
Cruz et al.(VGG)Anoop and Gould	96.87	99.79	97.99	96.87	98.48
Ours	99.56	99.19	**99.63**	97.98	98.96
Ours(multi-task)	**99.89**	99.42	99.46	**98.93**	**99.20**

Table 3.2: Relative attribute learning accuracy evaluated on the **PubFig** dataset. Our results outperform the work of Souri et al. Souri et al. (2016), which is the state-of-art for the traditional setting where only pairwise labels are used. Our results are also competitive and get slightly better results than those in Cruz et al. Anoop and Gould, which uses ranked input data. The last row shows the results of our network with multi-task loss function, which learns all of the 11 attributes at once.

Method	`lips`	`eyebrows`	`chubby`	`male`	`eyes`	`nose`	`face`	`smiling`	**Mean**
Souri et al.Souri et al. (2016)	93.62	94.53	92.32	95.59	93.19	94.24	94.76	95.36	94.52
Cruz et al.Anoop and Gould	99.55	97.21	97.66	99.44	96.54	96.21	99.11	97.88	98.14
Ours	98.28	97.11	98.67	98.05	**98.62**	**99.24**	97.32	**99.26**	98.51
Ours(multi-task)	**99.67**	**99.33**	**99.00**	98.33	97.32	98.46	99.00	97.51	**98.55**

Compared to the work of Souri et al. Souri et al. (2016), we outperform that method by a margin of 4% on the Public Figure Face Dataset, and by 1% on the OSR

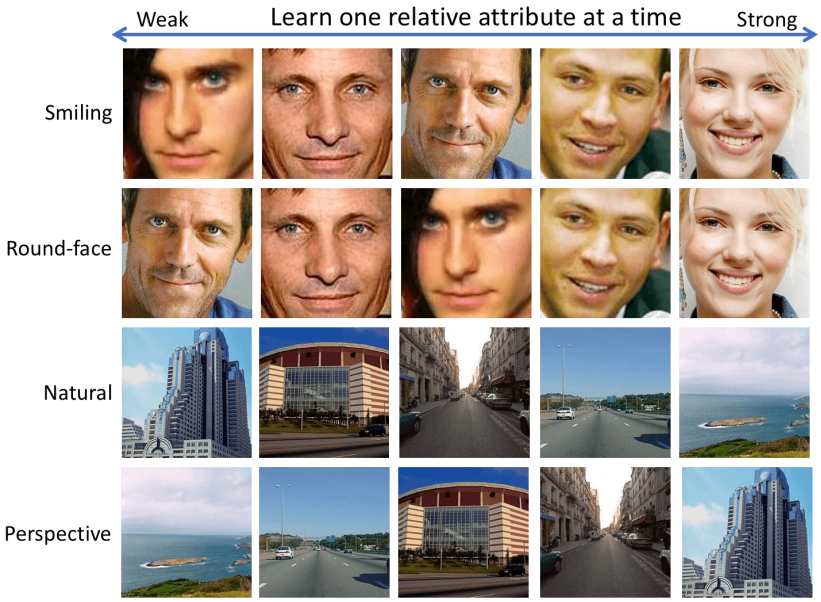


Figure 3.4: Qualitative results on RAL (**one at a time**) from our network. We randomly choose five different images from four different attributes from the PubFig and OSR datasets and show the results by ordering them for those attributes. The images are ranked by the corresponding output value of our network. The first two rows are from the PubFig dataset and the last two rows are images from the OSR dataset.

scene dataset. Since the accuracy on the OSR dataset is already high, a 1% improvement is meaningful. Compared with the DeepPermNet Anoop and Gould algorithm, we outperform that algorithm on both datasets on average. Note that DeepPermNet requires *ranked sequences* of data with the same length as training data, which may not be possible in some applications. Also note that both Souri et al. Souri et al. (2016) and DeepPermNet Anoop and Gould use `VGG CNN` model in their experiments, while we choose the simpler `Alexnet` Krizhevsky et al. (2012) in all experiments, which has far fewer parameters. As a result, our model can be trained faster than the baseline models.

Relative Attribute Learning, All at Once

In this experiment, we evaluate our multi-task network described in section 3.3. We learn all the attributes in each dataset and report the prediction accuracy results for each of the attributes on two different datasets in Table 3.1 and 3.2. Qualitative results are shown in Fig. 3.5.

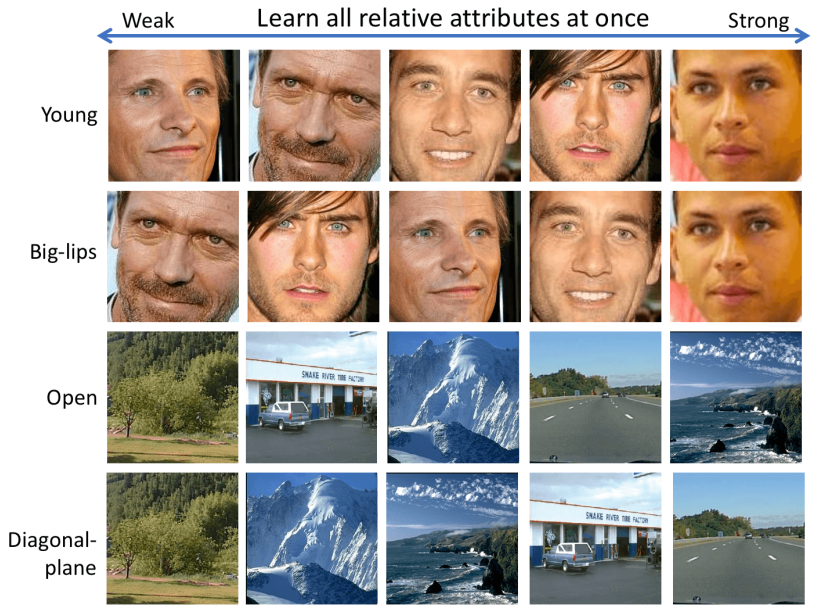


Figure 3.5: Qualitative results on RAL (**all at once**) using our network. The images are arranged again by ordering them according to the output of our network as Fig. 3.4 but these are learnt from our multi-task loss function (Eq. (3.10)). We can see that the images are quite nicely ordered even without learning the order explicitly as is done in DeepPermNet. We also note that the performance on almost all the images and the attributes is consistent and any randomly chosen subset gives us good quality results.

As the data presented show, our multi-task model slightly outperforms our single attribute learning model (section 3.3) and this indicates that some of the attributes are interrelated thus helping the learning process when we learn them all at once. Note that in our framework, with every additional attribute to learn, the increase in the number of parameters of the network is equal to the dimension of the two vectors, one in the readout function and one in the attribute node (in our work, the dimension of these two vectors is 4096×1). The reader may contrast this with most multi-task learning networks, such as Wang et al. (2017); Han et al. (2017); Abdulnabi et al. (2015), many of which use an additional CNN or several more fully connected layers for each additional attribute, which contribute to more parameters compared to our model.

Table 3.3: Attribute prediction accuracy on the Clothing Dataset Chen et al. (2012). Similar to Abdulnabi et al. (2015) we partition the 23 binary attributes into 4 groups (shown in Table 3.4). We achieve comparable results as those from MG-CNN Abdulnabi et al. (2015), but with significantly fewer parameters (see section 3.3) and faster training speed.

Method	Colors	Patterns	Cloth-parts	Appearance	Total
M-CNNAbdulnabi et al. (2015)	91.72	94.26	87.96	91.51	91.70
MG-CNNAbdulnabi et al. (2015)	93.12	95.37	88.65	91.93	92.82
Ours	91.64	96.81	89.25	89.53	92.39

Binary Attribute Prediction

Here, we evaluate our network for attribute prediction task described in section 3.3. The multi-task CNN model Abdulnabi et al. (2015) is a natural choice for the baseline. This model proposes to pre-train a convolutional neural network on each attribute to get the feature vectors, and then performs multi-task learning for multiple attributes. That model has a large number of parameters and a rich representation capacity. Similar to Abdulnabi et al. (2015), we also evaluate our model on Clothing Attributes Dataset Chen et al. (2012). It contains 1,856 images and 26 attributes. The ground truth is provided at the image-level, and each image is annotated for every attribute. For comparison, we ignore the multi-class value attributes as in Abdulnabi et al. (2015) and use this information in the same way to divide the 23 binary attributes into groups. We then use our multi-task network to train each group of attributes together. We report our results in Table 3.3 and the group information is provided in Table 3.4. M-CNN is the multi-task framework without group information in Abdulnabi et al. (2015) and MG-CNN is their multi-task framework with group encoding. The performance of our model is comparable to the results presented in MG-CNN framework, but is far more efficient both in the number of parameters and convergence time. For the number of parameters, Abdulnabi et al. (2015) needs one CNN for each attribute, while we only add 4096×1 parameters twice. In terms of training time, MG-CNN Abdulnabi et al. (2015) takes 1.5 days for training on the Clothing dataset with two NVIDIA TK40 16GB GPU, while our training takes less than 4 hours for all 4 groups of attributes on two NVIDIA Geforce GTX 1080Ti 12GB GPU.

Table 3.4: Grouping information used in Clothing Dataset Chen et al. (2012).

Group	Attributes
Colors	black, blue, brown, cyan, gray, green, many, red, purple, white,yellow
Patterns	floral, graphics, plaid, solid, stripe, spot
Cloth-parts	necktie, scarf, placket, collar
Appearance	skin-exposure, gender

Limitations

For our network to get a sizable performance benefit, we want that the graph formed by each random sample, i.e., group or mini-batch of n (e.g., $n = 5$) images should be "connected" or at least a subgraph with more than 2 nodes is connected. This allows learning from more than one image pair at a time to be meaningful – which is the main strength of our proposal. But if most pair labels do not have any node overlap, then the graph formed by a group or mini-batch of images will not have a connected component of size larger than two. We refer the reader to Frieze and Karoński (2015) (Chapter 4) to see the technical aspects of connectivity. The UT-Zappos50K dataset Yu and Grauman (2014, 2017) manifests this behavior (and is not ideal for our model to deliver performance gains). Under this condition, our model actually performs similar to (although not exactly the same) a Siamese network used in the literature. The results in Table 3.5 indeed support this intuition: our performance is only slightly better than Souri et al. (2016), rather than stronger improvements we see elsewhere.

Table 3.5: Relative attribute learning evaluated on UT-Zappos50K-lexicon dataset. It contains 50025 images of shoes with annotations on 4000 ordered pairs for each of 10 fine-grained attributes. The method in Anoop and Gould does not directly work on this dataset because of its "ordered sequence" requirement on the input data.

Method	comfort	casual	simple	sporty	colorful	durable	supportive	Mean
Souri et al.[3]	88.93	89.20	88.27	91.33	91.67	89.27	91.00	89.31
Ours	88.80	**89.82**	**90.13**	**92.60**	**91.87**	**90.07**	**92.73**	**90.07**

3.5 Summary

We presented a simple framework that can perform both relative attribute learning and attribute prediction. To exploit the underlying relationships between latent representations of a variety of attributes among a collection of images in a dataset, we

proposed a simple framework based on natural instantiation of graph neural network, where we unroll the message passing dynamics and use it to construct the layers. This formulation of a graph neural network can effectively encode the correlational information among multiple images and the multiple attributes as demonstrated in our experiments on three different datasets. Our framework can be used to learn the relative attributes either one at a time or all at once with only a modest increase in the number of parameters compared to other multi-task based methods. Because our framework learns mainly from pairs of images and does not require a full ranking it concurrently is less stringent on the annotation requirements of the training dataset. To the best of our knowledge, this proposal is among the first to explore the efficacy of multi-task GNN formulations for relative attribute learning. Our experiments also demonstrate the effectiveness of this architecture in achieving or surpassing the state-of-the-art results even for binary attributes prediction, where each attribute is predicted in a binary classification setup.

Chapter 4

Physarum Powered Differentiable Linear Programming Layers and Applications

In the previous chapter, the unrolled message passing dynamics are utilized as layers to model the graph-structured data. In this chapter, we attempt to model a more challenging setup–Linear Programming (LP). Many problems in machine learning can be expressed as, or otherwise involve as a sub-routine, the minimization of a linear function constrained by a set of linear equality and inequality constraints which defines a linear program. However, mechanisms that can solve it in a differentiable way are beginning to be proposed (at the time that this work was carried out). In this chapter, we tackle this problem by utilizing a special kind of dynamical system and instantiating it as layers. The work presented in this chapter was published as a conference paper at AAAI 2021 Meng et al. (2021b).

4.1 Introduction

LPs can be solved efficiently even when the problem sizes are large, and industrial strength solvers are readily available. Over the last twenty years, direct applications of LPs in machine learning and computer vision include image reconstruction Tsuda and Rätsch (2004), denoising Tavakoli and Pourmohammad (2012), deconvolution Ahmed et al. (2013) surface reconstruction Grady (2008), graphical models Ravikumar and Lafferty (2006), scene/view understanding Mauro et al. (2014), and numer-

ous others. While the use of specialized solvers based on combinatorial optimization rather than the direct use of a simplex or interior point method has been more common in large scale settings (e.g., in vision), there are also numerous instances where LP duality inspired schemes (such as primal-dual methods) have led to competitive and/or more general solution schemes.

Are LPs needed in modern learning problems? Within the last decade, deep neural networks have come to dominate many AI problems. So, an LP (or other well-studied numerical algorithms/methods) will rarely provide an *end-to-end* model for a practical problem. Nonetheless, similar to how various linear algebra routines such as eigendecomposition still play a key role as a sub-routine in modern learning tasks, *LP type models are still prevalent* in numerous pipelines in machine learning. For instance, consider a representation learner defined by taking our favorite off-the-shelf architecture where the representations are used to setup the cost for a "matching" problem (commonly written as a LP). Then, once a matching problem is solved, we route that output to pass through downstream layers and finally the loss is evaluated. Alternatively, consider the case where we must reason about (or group) a set of low-level primitives, via solving an assignment problem, to define a higher order semantic construct as is often the case in capsule networks Sabour et al. (2017). Or, our architecture involves estimating the Optimal transport distance Salimans et al. (2018); Bousquet et al. (2017); Sanjabi et al. (2018) where the cost matrix depends on the outputs of previous layers in a network. Such a module (rather, its approximations) lie at the heart of many popular methods for training generative adversarial networks (GANs) Arjovsky et al. (2017). Separately, confidence calibration is becoming an important issue in deep learning Guo et al. (2017); Nixon et al. (2019); several forms of calibration involve solutions to LPs. One approach for dealing with such a "in the loop" algorithmic procedure Amos and Kolter (2017) is to treat it as a general two-level optimization. When the feasible set of the LP is a box/simplex or can be represented using ratio type functions Ravi et al. (2020a), it is possible to unroll the optimization with some careful modifications of existing subroutines such as projections. This is not as straightforward in general where one must also concurrently perform projections on to the feasible set. An ideal solution would be a LP module that could be used anywhere in our architecture: one which takes its inputs from the previous layers and feeds into subsequent layers in the network.

Ours contributions: Backpropagation through LP. The key difficulty in solving LPs within a deep network is efficiently minimizing a loss $\ell(\cdot)$ which depends on a parameter derived from the solution of a LP – we must backpropagate *through* the LP solver to update the network weights. This problem is, of course, not unique to LPs but has been recently encountered in inserting various optimization modules as layers in a neural network, e.g., reverse mode differentiation through an ODE solver Chen et al. (2018), differentiable sorting Mena et al. (2018) and formulating quadratic Amos and Kolter (2017) or cone programs as neural network layers Agrawal et al. (2019). Our inspiration is a beautiful link Straszak and Vishnoi (2015); Johannson and Zou (2012) between dynamics of a slime mold (physarum polycephalum) and mathematical optimization that has not received attention in deep learning. Exploiting the ideas in Straszak and Vishnoi (2015); Johannson and Zou (2012) with certain adjustments leads to a "LP module/layers" called $\gamma-$AuxPD that can be incorporated within various architectures. Specifically, our main result together with the results in Straszak and Vishnoi (2015); Johannson and Zou (2012) shows that $\gamma-$AuxPD can solve a much larger class of LPs. Some immediate advantages of $\gamma-$AuxPD include **(a)** simple plug-and-play differentiable LP layers; **(b)** converges fast; **(c)** does not need a feasible solution as an initialization **(d)** very easy to integrate or implement. We will demonstrate how these properties provide a practical and easily usable module for solving LPs.

4.2 Related Works

The challenge in solving an optimization module *within* a deep network often boils down to the specific steps and the end-goal of that module itself. In some cases (unconstrained minimization of simple functions), the update steps can be analytically calculated Dave et al. (2019); Schmidt and Roth (2014). For more general unconstrained objectives, we must perform unrolled gradient descent during training Amos et al. (2017); Metz et al. (2016); Goodfellow et al. (2013). When the optimization involves certain constraints, one must extend the frameworks to use iterative schemes incorporating projection operators, that repeatedly project the solution into a subspace of feasible solutions Zeng et al. (2019). Since such operators are difficult to differentiate in general, it is hard to incorporate them directly outside of special cases. To this end, Amos et al. (2017) dealt with constraints by incorporating them in the

Lagrangian and using the KKT conditions. For combinatorial problems with linear objectives, Vlastelica et al. (2019) implemented an efficient backward pass through blackbox implementations of combinatorial solvers and Berthet et al. (2020) recently reported success with end-to-end differentiable learning with blackbox optimization modules. In other cases, when there is no associated objective function, some authors have reported some success with using reparameterizations for homogeneous constraints Frerix et al. (2019), adapting Krylov subspace methods de Roos and Hennig (2017), conditional gradient schemes Ravi et al. (2019) and so on.

Our goal here is to incorporate an LP as a module within the network, and is related in principle to some other works that incorporate optimization routines of different forms within a deep model which we briefly review here. In Belanger and McCallum (2016), the authors proposed a novel structured prediction network by solving an energy minimization problem within the network whereas Mensch and Blondel (2018) utilized differentiable dynamic programming for structured prediction and attention. To stabilize the training of Generative Adversarial Networks (GANs), Metz et al. (2016) defined the generator objective with respect to an unrolled optimization of the discriminator. Recently, it has been shown that incorporating concepts such as fairness Sattigeri et al. (2018) and verification Liu et al. (2019a) within deep networks also requires solving an optimization model internally. Closely related to our work is OptNet Amos and Kolter (2017), which showed how to design a network architecture that integrates constrained Quadratic Programming (QP) as a differentiable layers. While the method is not directly designed to work for linear programs (quadratic term needs to be positive definite), in experiments, one may add a suitable quadratic term as a regularization. More recently, Agrawal et al. (2019) introduces a package for differentiable constrained convex programming. Specifically, it utilizes a solver called SCS implemented in CVXPY package O'Donoghue et al. (2016, 2019), which we denote as CVXPY-SCS in this chapter.

4.3 Why Physarum Dynamics?

Consider a Linear Program (LP) in the standard form given by,

$$\min_{z \in \mathbb{R}^n} \quad c^T z \quad \text{s.t.} \quad Az = b, z \geqslant 0 \tag{4.1}$$

where $A \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^n_{>0}$, $b \in \mathbb{R}^m$. In (4.1), c is called the *cost vector* (we explain how to deal with nonpositive c in Section 4.4), and the intersection of the linear equalities $Ax = b$, and inequalities $x \geqslant 0$ is called the *feasible set* denoted by P. Now, we briefly discuss two main families of algorithms that are often used to solve LPs of the form (4.1).

Simplex Algorithms: The Workhorse

Recall that by the Minkowski-Weyl theorem, the feasible set P can be decomposed into a finite set of extreme points and rays. A family of algorithms called *Simplex* exploits this decomposition of P to solve LPs. Intuitively, the Simplex method is based on the principle that if there exists a solution to a LP, then there is at least one vertex (or an extreme point) of P that is optimal. In fact, Simplex algorithms can be seen as **First Order** methods with a careful choice of update direction so as to move along the edges of P. There are three key properties of simplex algorithms to solve LP (4.1):

1. *Good:* We can obtain *exact* solutions in finite number of iterations;
2. *Bad:* The worst case complexity is exponential in m (or n); and
3. *Highly undesirable:* The update directions are computed by forming the *basis* matrix making the algorithm *combinatorial/nondifferentiable* in nature.

Remark 4.1. *It may not be possible to use a differentiable update rule since it would require an enumeration of vertices of* P *– exponential in dimensions* n *Barvinok (2013).*

Interior Point Algorithms: Trading Exactness for Efficiency

Asking for exact solutions of LP (4.1) may be a stringent requirement. An approximate solution of LP (4.1) can be computed using a different family of methods called *Interior Point Method* (IPM) in $O(\sqrt{\max(m,n)})$ Wright (1997). Intuitively, while the iterates of a simplex method proceed along the edges of P, an IPM passes through the *interior* of this polyhedron. In particular, IPMs are **second order** algorithms since they directly solve the system of nonlinear equations derived from KKT conditions by applying variants of Newton's method Wright (1997). As with Simplex methods, we point out to three key properties of IPM:

1. *Good:* IPM based algorithms can efficiently solve LP (4.1) in theory Lee and Sidford (2014); Gondzio (2012);

2. *Bad:* IPMs *need* to be started from a feasible point although there are special infeasible start IPMs Roos (2006);

3. *Bad:* In practice, IPMs are faster than Simplex Method *only* when m, and n are large, e.g., millions Cui et al. (2019).

Remark 4.2. *Even if we can find a feasible point efficiently, it is not easy to* warm start *IPM methods due to the high sensitivity of the central path equation John and Yıldırım (2008). In contrast, first order methods like Simplex can be easily warm started Arsham (1997).*

Physarum Dynamics: Best of Both Worlds?

The term *Physarum Dynamics (PD)* refers to the movement of a slime mold called *Physarum polycephalum*, and is studied in mathematical biology for its inherent computational nature and properties that closely mirror mathematical optimization. For example, in an interesting result, Toshiyuki et al. (2000) showed that the slime mold can solve a shortest path problem on a maze. Further, the temporal evolution of Physarum has been used to learn robust network design Tero et al. (2007); Johannson and Zou (2012), by connecting it to a broad class of dynamical systems for basic computational problems such as shortest paths and LPs. In Straszak and Vishnoi (2015), the authors studied the convergence properties of PD for LPs, and showed that these steps surprisingly mimic a steepest-descent type algorithm on a certain Riemannian manifold. While these interesting links have not been explored in AI/deep learning, we find that the simplicity of these dynamics and its mathematical behavior provide an excellent approach towards our key goal.

We make the following mild assumption about LPs (4.1) that we consider here

Assumption 1 (Feasibility)**.** *The feasible set* $B := \{z : Az = b, z \geqslant 0\}$ *of (4.1) is nonempty.*

For the applications considered in this chapter, Assumption 1 is always satisfied. We now describe the PD for solving LPs and illustrate the similarities and differences between PD and other methods.

Consider any vector $z \in \mathbb{R}^n$ with $z > 0$ and let $W \in \mathbb{R}^{n\times n}$ be the diagonal matrix with entries $\frac{z_i}{c_i}, i = 1, 2, ..., n$. Let $O = AWA^T$ and $p \in \mathbb{R}^m$ is the solution to the linear

system $Op = b$. Let $q = WA^T p$. The PD for a LP (e.g., in (4.1)) given by (A, b, c) is defined as,

$$\frac{dz_i(t)}{dt} = q_i(t) - z_i(t), \quad i = 1, 2, \ldots, n. \tag{4.2}$$

Equivalently, using the definition of q we can write the *continuous* time PD compactly as,

$$\dot{z} = W(A^T L^{-1} b - c). \tag{4.3}$$

Theorem 1 and 2 in Straszak and Vishnoi (2015) guarantee that (4.3) converges to an $\epsilon-$approximate solution efficiently with no extra conditions and its discretization converges as long as the positive step size is small enough.

Remark 4.3 (PD vs IPM)**.** *Similar to IPM, PD requires us to compute a full linear system solve at each iteration. However, note that the matrix* L *associated with linear system in PD is completely different from the KKT matrix that is used in IPM. Moreover, it turns out that unlike most IPM, PD can be started with an* **infeasible starting point**. *Note that PD only requires the initial point to satisfy* $As = b$ *which corresponds to solving ordinary least squares which can be easily done using any iterative method like Gradient Descent.*

Remark 4.4 (PD vs Simplex)**.** *Similar to Simplex, PD corresponds to a gradient, and therefore is a* first order *method. The crucial difference between the two methods, is that the metric used in PD is* **geodesic** *whereas Simplex uses the Euclidean metric. Intuitively, using the geodesic metric of* P *instead of the Euclidean metric can vastly improve the convergence speed since the performance of first order methods is dependent on the choice of coordinate system Yang and Amari (1998); Zhang and Sra (2016).*

When is PD efficient? As we will see shortly in Section 4.6, in the two applications that we consider in this chapter, the sub-determinant of A is *provably* small – constant or at most quadratic in m, n. In fact, when A is a node incidence matrix, PD computes the shortest path, and is known to converge extremely fast. In order to be able to use PD for a wider range of problems, we propose a simple modification described below. Note that since many of the vision problems require auxiliary/slack variables in their LP (re)formulation, the convergence results in Straszak and Vishnoi (2015) do *not* directly apply since L in (4.3) is *not* invertible. Next, we discuss how to

deal with noninvertibility of L using our proposed algorithm called $\gamma-$AuxPD (in Algorithm 3).

4.4 Dealing with Auxiliary Variables using $\gamma-$AuxPD

In the above description, we assume that $c \in \mathbb{R}^n_{>0}$. We now address the case where $c_i = 0$ under the following assumption on the feasible set P of LP (4.1):

Assumption 2 (Bounded). *The feasible set* $B \subseteq [0, \bar{K}]^n$, *i.e.,* $z \in B \implies z_i \leqslant \bar{K}\ \forall\, i \in [n]$.

Intuitively, if P is bounded, we may expect that the optimal solution set to be invariant under a sufficiently small perturbation of the cost vector along *any* direction. The following observation from Johannson and Zou (2012) shows that this is indeed possible as long as P is finitely generated:

Observation 4.5 (Johannson and Zou (2012))**.** *Let* $\epsilon > 0$ *be the given desired level of accuracy, and say* $c_i = 0$ *for some* $i \in [n]$. *Recall that our goal is to find a point* $\hat{z} \in P$ *such that* $c^T\hat{z} - c^T z^* \leqslant \epsilon$ *where* z^* *is the optimal solution to the LP (4.1). Consider the* $\gamma-$*perturbed LP given by* $\{A, b, \hat{c}\}$, *where* $\hat{c}_i = c_i$ *if* $c_i > 0$ *and* $\hat{c} = \gamma$ *if* $c_i = 0$. *Let* z_2 *be an extreme point that achieves the second lowest cost to LP (4.1). Now it is easy to see that if* $\gamma < \frac{\delta}{n \cdot M}$ *where* $\delta = c^T z_2 - c^T z^*$, *then* z^* *is an approximate solution of* $\{A, b, \hat{c}\}$. *Hence, it suffices to solve the* $\gamma-$*perturbed LP.*

With these modifications, we present our discretized $\gamma-$AuxPD algorithm 3 that solves a slightly perturbed version of the given LP.

Remark 4.6. *Note that* $\gamma-$*perturbation argument does not work for any* P *and* c *since LP (4.1) may be unbounded or have* no *extreme points.*

Observation 4.5 can be readily used for computational purposes by performing a binary search over γ if we can obtain a finite upper bound γ_u. Furthermore, if γ_u is a polynomial function of the input parameters m, n of LP, then Observation 4.5 implies that $\gamma-$AuxPD algorithm is also efficient. Fortunately, for applications that satisfy the bounded assumption 2, our Theorem 4.8 shows that a *tight* upper bound γ_u on γ_P can be provided in terms of M (diameter of P).

Implementation. Under Assumption 2, negative costs can be handled by replacing $z_i = -y_i$ whenever $c_i < 0$, or in other words, by flipping the coordinate axis

Algorithm 1: $\gamma-$AuxPD Layer

Input: LP problem parameters A, b, c, initial point z_0, Max iteration number K, step size h, accuracy level ϵ, approximate diameter γ_P
Set $z_s \leftarrow z_0$ if z_0 is provided else rand$([n], (0, 1))$
Perturb cost $c \leftarrow c + \gamma_P \mathbf{1_0}$ where $\mathbf{1_0}$ is the binary vector with unit entry on the indices i with $c_i = 0$
for $i = 1$ *to* K **do**
  Set: $W \leftarrow \text{diag}(z_s/c)$
  Compute: $O \leftarrow AWA^T$
  Compute: $p \leftarrow O^{-1}b$ using iterative solvers
  Set: $q \leftarrow WA^Tp$
  Update: $z_s \leftarrow (1-h)z_s + hq$
  Project onto $\mathbb{R}_{\geqslant \epsilon}$: $z_s \leftarrow \max(z_s, \epsilon)$
end
Return: z_s

of coordinates with negative costs, which has been noticed in Johannson and Zou (2012). Since we use an iterative linear system solver to compute q, we project x on to $\mathbb{R}_{\geqslant \epsilon}$ after each iteration: this corresponds to a simple clamping operation.

4.5 Analysis of Some Testbeds for $\gamma-$AuxPD: Bipartite Matching and SVMs

In order to illustrate the potential of the $\gamma-$AuxPD layer (Alg. 3), we consider two classes of LPs common in a number of applications and show that they can be solved using $\gamma-$AuxPD. These two classes of LPs are chosen because they link nicely to interesting problems involving deep neural networks which we study in §4.6.

	$\gamma-$**AuxPD**			**PGD-Dykstra**		
Iter. #	10	50	100	10	50	100
Proj. #	NA	NA	NA	5	10	50
Objective	0.100	0.098	0.099	0.137	0.121	0.120
Time (s)	0.016	0.040	0.071	0.016	0.146	0.498

Table 4.1: Results on solving random matching problems.

Bipartite Matching using Physarum Dynamics

Given two finite non-intersecting sets $\mathcal{I}$, $\mathcal{J}$ such that $|\mathcal{I}| = m$, $|\mathcal{J}| = n, n \ll m$, and a cost function $\mathcal{C} : \mathcal{I} \times \mathcal{J} \to \mathbb{R}$, solving a minimum cost bipartite matching problem corresponds to finding a map $f : \mathcal{I} \to \mathcal{J}$ such that total cost $\sum_i \mathcal{C}(i, f(i))$ is minimized. If we represent f using an assignment matrix $X \in \mathbb{R}^{n\times m}$, then a LP relaxation of the matching problem can be written in standard form (4.1) as,

$$\begin{aligned} \min_{(X,s_m)\geqslant 0} \quad & \operatorname{tr}(\mathcal{C}X^T) + \gamma \mathbf{1}_m^T s_m \\ \text{s.t.} \quad & X\mathbf{1}_m = 1_n,\ X^T 1_n + s_m = 1_m \end{aligned} \tag{4.4}$$

where $\mathcal{C} \in \mathbb{R}^{n\times m}$ is the cost matrix, $\mathbf{1}_d$ is the all-one vector in d dimension, and $s_m \in \mathbb{R}^m$ is the slack variable.

Remark 4.7. *Note that in LP (4.4), the slack variables s_m impose the m inequalities $X^T 1_n \leqslant 1_m$.*

The following theorem shows that the convergence rate of $\gamma-$AuxPD applied to the bipartite matching in (4.4) only has a dependence which is **logarithmic** in n.

Theorem 4.8. *Assume we set $0 < \gamma \leqslant \gamma_u$ such that $1/\gamma_u = \Theta(\sqrt{m})$. Then, our $\gamma-$AuxPD (Algorithm 3) converges to an optimal solution to (4.4) in $\tilde{O}\left(\frac{m}{\epsilon^2}\right)$ iterations where $\tilde{\mathcal{O}}$ hides the logarithmic factors in m and n.*

Proof. (Sketch) To prove Theorem 4.8, we use a result from convex analysis called the *sticky face lemma* to show that for all small perturbations of c, the optimal solution set remains invariant. We can then simply estimate γ_u to be the largest acceptable perturbation (which may depend on $\mathcal{C}, B$ but *not* on any combinatorial function of B like extreme points/vertices). □

Verifying Theorem 4.8. We construct random matching problems of size $n = 5, m = 50$ (used later in §4.6) with batch size 32, where we randomly set elements of $\mathcal{C}$ to be values in $[0, 1]$. We compare our method with CVXPY-SCS and a projected gradient descent algorithm in which the projection exploits the Dykstras algorithm (used by Zeng et al. (2019) in §4.6) (we denote it as PGD-Dykstra).

Evaluation Details. We run 100 random instances of matching problems for both our $\gamma-$AuxPD algorithm and PGD-Dykstra with different number of iterations.

We report the objective value computed using the solution given by our $\gamma-$AuxPD solver/PGD-Dykstra/CVXPY-SCS. Our step size is 1 and learning rate of PGD-Dykstra is set to 0.1 (both used in §4.6). For CVXPY-SCS, the number of iterations is determined by the solver itself for each problem and it gets 0.112 objective with mean time 0.195 (s). The results of $\gamma-$AuxPD and PGD-Dykstra are reported in Table 4.1. Our $\gamma-$AuxPD algorithm achieves faster convergence and better quality solutions.

ℓ_1-normalized Linear SVM using $\gamma-$AuxPD

In the next testbed for $\gamma-$AuxPD, we solve a ℓ_1-normalized linear SVM Hess and Brooks (2015) in the standard form of LP (4.1). Below, $\widetilde{K}^{[i,j]}$ stands for $K(x_i, x_j)(\alpha_{1j} - \alpha_{2j})$:

$$\begin{aligned}
&\min_{\alpha_1,\alpha_2,s,b_1,b_2,\xi} \quad \sum_{i=1}^{n} s_i + \underline{C} \sum_{i=1}^{n} (\xi_i + 2z_i) \\
&\text{s.t.} \quad y_i \left(\sum_{j=1}^{n} y_j \widetilde{K}^{[i,j]} + (b_1 - b_2) \right) + \xi_i - M z_i - l_i = 1, \\
&\sum_{j=1}^{n} y_j \widetilde{K}^{[i,j]} - s_i + p_i = 0, \quad \sum_{j=1}^{n} y_j \widetilde{K}^{[i,j]} + s_i - q_i = 0, \\
&z_i + r_i = 1, \quad \alpha_1, \alpha_2, s, b_1, b_2, \xi, z_i, l_i, p_i, q_i, r_i, \geqslant 0 \\
&\forall i = 1, 2, \cdots, n.
\end{aligned} \tag{4.5}$$

Like Thm. 4.8, we can show a convergence result for ℓ_1-SVM (4.5) (see appendix).

Verifying convergence of $\gamma-$AuxPD for ℓ_1-SVM (4.5). We compare our method with the recent CVXPY-SCS solver Agrawal et al. (2019) which can also solve LPs in a differentiable way. We constructed some simple examples to check whether CVXPY-SCS and our $\gamma-$AuxPD solver works for SVMs (e.g., binary classification where training samples of different class come from Gaussian distribution with different mean). Both $\gamma-$AuxPD and CVXPY-SCS give correct classification results. We will further show in §4.6 that when used in training, $\gamma-$AuxPD achieves better performance and faster training time than CVXPY-SCS.

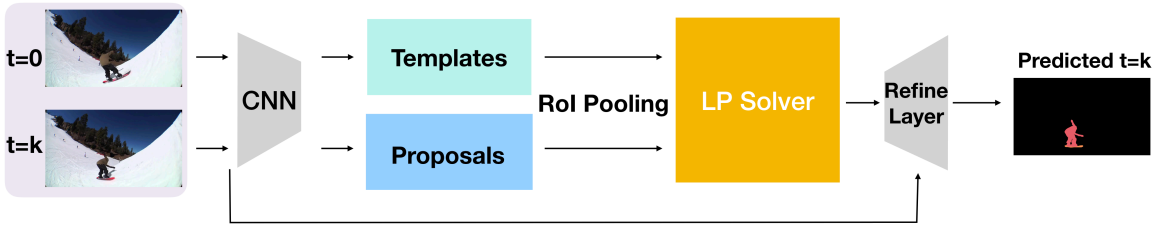


Figure 4.1: Architecture of DMM Zeng et al. (2019): The yellow box is where the linear program is solved. In this application the linear program is a bipartite matching problem.

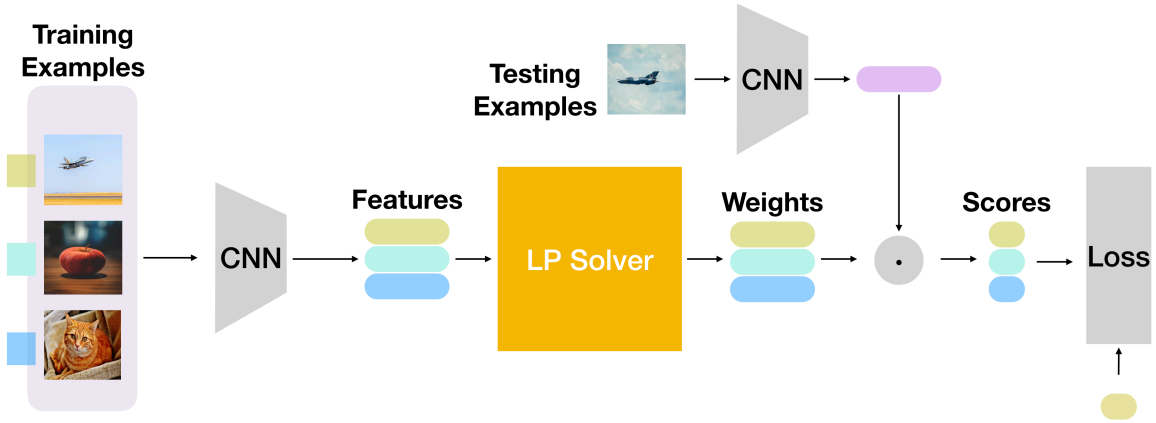


Figure 4.2: Architecture of Meta-learning Lee et al. (2019): The yellow box is where the linear program is solved. In this application, the linear program is a linear SVM.

4.6 Differentiable LPs in Computer Vision

We now demonstrate the versatility of our $\gamma-$AuxPD layer in particular scenarios in computer vision. Our goal here is to show that while the proposed procedure is easy, it can indeed be used in a plug and play manner in fairly different settings, where the current alternative is either to design, implement and debug a specialized sub-routine Zeng et al. (2019) or to utilize more general-purpose schemes when a simpler one would suffice (solving a QP instead of a LP) as in Lee et al. (2019). We try to keep the changes/modifications to the original pipeline where our LP solver is deployed as minimal as possible, so ideally, we should expect that there are no major fluctuations in the overall accuracy profile.

Differentiable Mask-Matching in Videos

We review the key task from Zeng et al. (2019) to introduce the differentiable mask-matching network for video object segmentation, and how/why it involves a LP solution. The overall architecture is in Fig. 4.1.

Problem Formulation. Given a video with T frames as well as the mask templates in the first frame, the goal is to obtain a segmentation of the same set of instances in all of remaining frames. Zeng et al. (2019) shows that differentiable matching between the templates and the bounding boxes proposed by the detector achieves superior performance over previous methods.

LP instance. The goal is to use the cost matrix and solve a matching problem. Recall that minimum-cost bipartite matching can be formulated as a integer linear program (ILP) and can be relaxed to a LP, given by the formulation in standard form stated in (4.4) (identical to the ILP and LP in Zeng et al. (2019)). The number of proposals m is much larger than the number of templates n and so one would ask that $X^T\mathbf{1}_n \leqslant \mathbf{1}_m$ instead of $X^T\mathbf{1}_n = \mathbf{1}_m$.

Solver. In Zeng et al. (2019), the authors use a specialized projected gradient descent algorithm with a cyclic constraint projection method (known as *Dykstra's algorithm*) to solve the LP. The constraints in this LP are simple enough that calculating the projections is not complicated although the convergence rate is **not known**. We can directly replace their solver with $\gamma-$AuxPD in Alg. 3 to solve the problem, also in a differentiable way. Once the solution is obtained, Zeng et al. (2019) uses a mask refinement module which we also use to ensure consistency between the pipelines.

Experiments on Youtube-VOS.

Parameter settings. The projection gradient descent solver in Zeng et al. (2019) has three parameters to tune: number of gradient steps, number of projections, and learning rate. We use $N_{grad} = 40, N_{proj} = 5, lr = 0.1$ as in their paper to reproduce their results. For $\gamma-$AuxPD layer, the choice is simple: step size $h = 1$ and $K = 10$ iterations work well for both two experiments and the other tests we performed. From Table 4.1 we can see that the PGD-Dykstra solver from Zeng et al. (2019) is faster and more tailormade for this application than CVXPY-SCS thus we only compare with the PGD-Dykstra solver for this application.

How do different solvers compare on Youtube-VOS? Our final results are shown in Table 4.2. Our solver works well and since the workflow is near identical to Zeng

	$\mathcal{J}_m$	$\mathcal{J}_r$	$\mathcal{J}_d$	$\mathcal{F}_m$	$\mathcal{F}_r$	$\mathcal{F}_d$
DMM-Net Zeng et al. (2019)	63.4	72.7	9.3	77.3	84.9	10.5
$\gamma-$AuxPD layer	63.4	72.2	9.2	77.3	85.3	10.4

Table 4.2: Results on Youtube-VOS train-val split. Subscripts m, r, d stand for mean, recall, and decay respectively.

et al. (2019), we achieve comparable results with Zeng et al. (2019) while achieving small benefits in inference time. We notice that although our solver performs better for a simulated matching problems; since the matching problem here is small and the cost matrix learned by the feature extractor is already good (so easy to solve), the runtime behavior is similar. Nonetheless, it shows that the general-purpose solver can be directly plugged in and offers performance which is as good as a *specialized solution* in Zeng et al. (2019) that exploits the properties of the particular constraint set.

Meta-learning for Few-shot Learning

We briefly review the key task from Lee et al. (2019) to introduce the few-shot learning task using a meta-learning approach, and how it involves getting a solution to a LP. Due to limited space, we refer readers to Lee et al. (2019) for more details of the meta-learning for few-shot learning task. The overall architecture is in Fig. 4.2.

Problem Formulation. Given a training set $D^{train} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$, in this problem, the goal of the base learner $\mathcal{A}$ is to estimate parameters θ of the predictor $y = f(\mathbf{x}; \theta)$ so that it generalizes well to the unseen test set $D^{test} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{Q}$. The meta learner seeks to learn an embedding model ϕ that minimizes the generalization error across tasks given a base learner $\mathcal{A}$.

LP instance. There are several requirements for the base learners. First, the evaluation needs to be very efficient since a base learner needs to be solved in every iteration within the *meta-learning* procedure. Second, we need to be able to estimate and backpropagate the gradient from the solution of the base learner back to the embedding model f_ϕ, which means that the solver for the base learner needs to be differentiable. In Lee et al. (2019), the authors use a multi-class linear support vector machine (SVM) with an ℓ_2 norm on the weights Crammer and Singer (2001). Instead, to instantiate an LP, we use a ℓ_1 normalized SVM proposed by Hess and Brooks (2015). The optimization model for this SVM in a standard form is shown in (4.5). This is a

	CIFAR-FS 5-way		FC100 5-way	
LP Solver	1-shot	5-shot	1-shot	5-shot
MetaOptNet-CVXPY-SCS	70.2 ± 0.7	83.6 ± 0.5	38.1 ± 0.6	51.7 ± 0.6
MetaOptNet-Optnet (with regularization)	69.9 ± 0.7	83.9 ± 0.5	37.3 ± 0.5	52.2 ± 0.5
MetaOptNet-$\gamma-$AuxPD (Ours)	71.4 ± 0.7	84.3 ± 0.5	38.2 ± 0.5	54.2 ± 0.5

Table 4.3: Results on CIFAR-FS and FC100. In K-way, N-shot few shot learning, K is the number of classes and N is the number of training examples per class.

binary SVM model, on top of which we run $\binom{k}{2}$ pairwise SVMs to obtain the solution where k is the number of classes in the task.

Solver. In Lee et al. (2019), the authors use OptNet. Note that the number of parameters is only related to the number of training examples and the number of classes, which is often much smaller than the dimensionality of the features for few-shot learning. Since feature selection seems more appropriate here, we may directly replace OptNet with our $\gamma-$AuxPD layer to solve the ℓ_1-SVM efficiently. Our baseline method is CVXPY-SCS Agrawal et al. (2019). The implementation of Optnet Amos and Kolter (2017) does not directly support solving LPs since it requires a positive definite quadratic term. Still, to test its ability of solving LPs, we add a diagonal matrix with a small value (0.1, since diagonal value smaller than 0.1 leads to numerical errors in our experiment) as the quadratic term (can be thought of as a regularization term).

Experiments on CIFAR-FS and FC100.

Datasets. We follow the code from Lee et al. (2019) to conduct the experiments on **CIFAR-FS** and **FC100**. Other training details and dataset information are in the supplement.

How do different solvers compare on CIFAR-FS and FC100? The results on CIFAR-FS and FC100 are shown in Table 4.3. Using the ℓ_1 normalized SVM, our solver achieves better performance than CVXPY-SCS Agrawal et al. (2019) and Optnet (with a small quadratic term as regularization) on both datasets and both the 1-shot and 5-shot setting. Expectedly, since the pipeline is very similar to Lee et al. (2019), we achieve a similar performance as reported there, although their results were obtained through a different solver. This suggests that our simpler solver works at least as well, and no other modifications were needed. Importantly, during the

batch size	8	32	128
CVXPY-SCS	32.3	122.7	455.2
Optnet	42.4	88.1	243.7
$\gamma-$AuxPD (Ours)	24.0	25.1	25.8

Table 4.4: Time (ms) spent on solving a batch of LP problems. The time reported here for CVXPY-SCS does not include that spent on constructing the canonicalization mapping.

Variance of noise	0	0.01	0.03	0.05	0.1
Test accuracy	71.4	70.1	69.1	68.2	61.91

Table 4.5: Experiment on CIFAR-FS 5-way 1-shot setting where zero mean random Gaussian noise is added to the solution of $\gamma-$AuxPD solver.

training phase, our solver achieves $4\times$ **improvement in runtime** compared with CVXPY-SCS (baseline which can also solve the ℓ_1-SVM). Lee et al. (2019) also reported the performance of solving ℓ_2 normalized SVM. The choice of ℓ_1 versus ℓ_2 often depends on specific application settings.

We also compare the time spent on solving a batch of LP problems with $n = 92, m = 40, p = 122$ (same size used in the experiment), where n is number of variables, m is number of equality constraints and p is the number of inequality constraints in the original problem form. Table 4.4 shows that our implementation is efficient for batch processing on GPU, which is crucial for many modern AI applications. We also performed a GPU memory consumption comparison with a batch size of 32: our solver needs 913MB GPU memory, CVXPY-SCS needs 813MB and Optnet needs 935MB which are mostly comparable.

How does LP solver influence the global convergence of the task? To understand how the quality of LP solver influences the global convergence of the learning task (i.e., where the LP is being used), we conduct a simple experiment. This addresses the question of whether a good LP solver is really needed? Here, we add a random Gaussian noise with zero mean and small variance to the solution of LP solver (to emulate results from a worse solver) and observe the convergence and final accuracy in the context of the task. We can see in Table 4.5 that the quality of LP solution has a clear influence on the overall performance of the training (few-shot learning in this example).

4.7 Summary

This chapter describes how Physarum dynamics based ideas Straszak and Vishnoi (2015); Johannson and Zou (2012) can be used to obtain a differentiable LP solver that can be easily integrated within various deep neural networks if the task involves obtaining a solution to a LP. The dynamics is unrolled for a certain number of steps until it converges and this unrolled dynamics is used as the layer. Outside of the tasks shown in our experiments, there are many other use cases including differentiable isotonic regression for calibration, differentiable calculation of Wasserstein Distance, differentiable tracking, and so on. The algorithm, $\gamma-$AuxPD, converges quickly without requiring a feasible solution as an initialization, and is easy to implement/integrate. Experiments demonstrate that when we preserve existing pipelines for video object segmentation and separately for meta-learning for few-shot learning, with substituting in our simple $\gamma-$AuxPD layer, we obtain comparable performance as more specialized schemes. As briefly discussed earlier, recent results that utilize implicit differentiation to solve combinatorial problems Vlastelica et al. (2019) or allow using blackbox solvers for an optimization problem during DNN training Berthet et al. (2020); Ferber et al. (2020), are indeed promising developments because any state of the art solver can be utilized. However, current LP solvers are often implemented to be CPU-intensive and suffer from overhead compared with solvers that are entirely implemented on the GPU. This is beneficial for DNN training. The LPs considered in this chapter have comparable numbers of variables and constraints, and the proposed solver may become inefficient when the number of constraints is much larger than that of the variables. In the next chapter, we will propose a solver suitable for this setting.

Chapter 5

Differentiable Optimization of Generalized Nondecomposable Functions using Linear Programs

In the last chapter, we unrolled physarum dynamics to construct a differentiable LP solver. In this chapter, the goal is similar–to propose an LP solver, but here we focus specifically on optimizing nondecomposable functions, where the number of constraints is usually larger than the number of variables. This brings additional challenges and we show how to use a Newton method to tackle this problem. Besides, the solver we propose in this chapter computes backward gradients via the fixed point which saves the time cost during the process of computing the backward gradients compared with the physarum solver in the previous chapter. The work in this chapter was published as a conference paper at NeurIPS 2021.

5.1 Introduction

Commonly used losses such as cross-entropy used in deep neural network (DNN) models can be expressed as a sum over the per-sample losses incurred by the current estimate of the model. This allows the direct use of mature optimization routines, and is sufficient for a majority of use cases. But in various applications ranging from ranking/retrieval systems to class imbalanced learning, the most suitable losses for the task do not admit a "decompose over samples" form. Examples include Area under the ROC curve (AUC), multi-class variants of AUC, F-score, Precision at a fixed

recall (P@R) and others. Optimizing such measures in a scalable manner can pose challenges even in the shallow setting. For AUC maximization, we now know that convex surrogate losses can be used in a linear model Liu et al. (2018); Natole et al. (2018) in the so-called ERM framework. These ideas have been incorporated within a deep neural network and solved using SGD type method in Liu et al. (2019b). These kinds of computational results on stochastic and online data models have also been explored in Ataman et al. (2006); Cortes and Mohri (2004); Gao et al. (2013). There are also results for measures other than the AUC: Nan et al. (2012); Dembczynski et al. (2011) gives exact algorithms for optimizing F-score and Eban et al. (2017); Ravi et al. (2020b) proposes scalable methods for non-decomposable objectives which utilizes Lagrange multipliers to construct the proxy objective. The authors in Mohapatra et al. (2018) propose to use a structured hinge-loss upper bound to optimize average precision and NDCG. Recently, Fathony and Kolter (2020) presented a general formulation for such nondecomposable measures using adversarial prediction framework, and showed that it is indeed possible to incorporate such measures into differentiable pipelines using ADMM based techniques.

Our work described here utilizes the simple observation that a number of these non-decomposable objectives can be expressed in the form of an integer program that can be relaxed to a linear program (LP). Our approach is based on the premise that tackling the LP form of the non-decomposable objective as a module within the DNN, one which permits forward and reverse mode differentiation and can utilize in-built support for specialized GPU hardware in modern libraries such as PyTorch, is desirable. First, as long as a suitable LP formulation for an objective is available, the module may be directly used. Second, based on which scheme is used to solve the LP, one may be able to provide guarantees for the non-decomposable objective based on simple calculations (e.g., number of constraints, primal-dual gap). The current tools, however, do not entirely address all these requirements, as we briefly describe next. A characteristic of the LPs that arise from the losses mentioned above is that the constraints (including the mini-batch of samples themselves) are modified at each iteration – as a function of the updates to the representations of the data in the upstream layers.

In principle, of course, backpropagating through a convex optimization model (and in particular, LPs) is not an unsolved problem (e.g., the physarum solver we proposed in the last chapter). For LPs, we can take derivatives of the optimal value

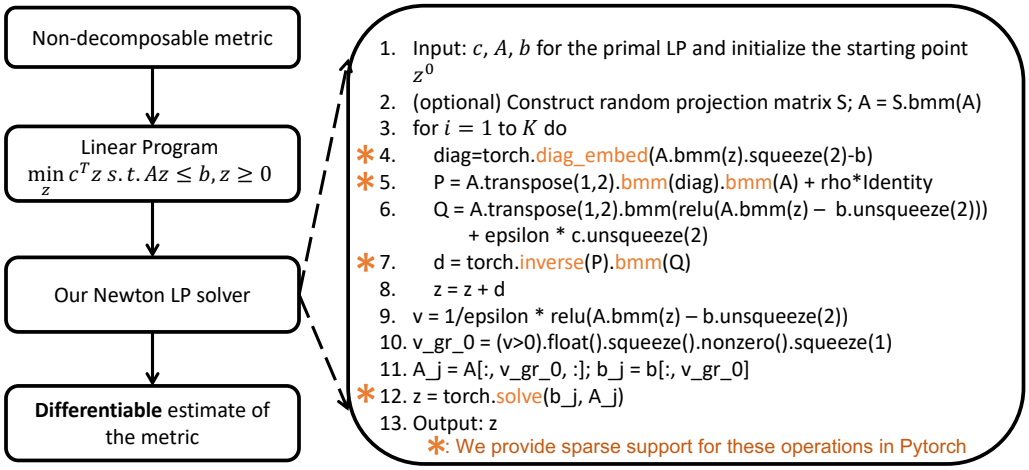


Figure 5.1: Overview of how to compute differentiable non-decomposable metric and code example of our LP solver. We provide sparse support for operations which are not supported in official Pytorch library (colored operations in the figure).

(or the optimal solution) of the model with respect to the LP parameters, and this can be accomplished by calling a powerful external solver. Often, this would involve the overhead of running the solver on the CPU. Solvers within CVXPY, are effective but due to their general-purpose nature, rely on interior point methods. OptNet Amos and Kolter (2017) is quite efficient but designed for quadratic programs (QP): the theoretical results and its efficiency depends on factorizing a matrix in the quadratic term in the objective (which is zero/non-invertible for LPs). The primal-dual properties and implicit differentiation for QPs do not easily translate to LPs due to the polyhedral geometry in LPs. The ideas in Meng et al. (2020) are only applicable when the number of constraints is approximately equal to the number of variables – an invalid assumption for the models that we study here.

Our contributions. We provide LP formulations of widely used nondecomposable terms that satisfy the requirements for our solver. We show that the dynamics formed by the modified Newton's algorithm in Mangasarian (2004) can be used for deep neural network (DNN) training in an end-to-end manner without requiring external solvers where support for GPUs remains limited. Specifically, by exploiting self-concordance of the objective, we show that the algorithm can converge globally *without* line search strategies. We then analyze the gradient properties of our approach, and some modifications to improve stability during backpropagation. Our experiments will show that this scheme based on Mangasarian's parametric exterior penalty formulation of the primal LP is a computationally effective and scalable strat-

egy to solve LPs with a large number of constraints. On the practical side, we provide two ways to deal with the scaling issue when the constraint matrix is large. On the one hand, we show that sufficient dimension reduction can be used in our solver to solve the problem in a lower dimension space. On the other hand, when the matrix is sparse, with our new sparse implementation, we show that we can train Resnet-18 with $10\times$ mini-batch size (memory savings) on a 2080TI Nvidia GPU, see Fig. 5.1.

5.2 Nondecomposable Functions and corresponding LP models

We first present a standard LP form and then reparameterize several generalized nondecomposable objectives in this way, summarized in a table in the appendix. We start with the binary AUC, extend it to multi-class AUC, and then later, discuss a ratio objective, F-score. The appendix also includes a discussion of other objectives that are amenable to our method.

Notations. Recall that we use I_r to represent the identity matrix of size r; $\mathbf{B}_{k,\cdot}$ (and $\mathbf{B}_{|,k'}$) gives the k-th row (and k'-th) column of $\mathbf{B}$. We also use $X \in \mathbb{R}^{n\times \dim_X}$ to represent the features fed to a classifider and $Y \in \{0,1\}$ to represent the binary label. $f(x_i)$ denotes the score function for the classifier where $x_i \in X$.

We consider a general linear program (LP) that can be given by

$$\min_z c^T z \text{ subject to } Az \leqslant b,\ z \geqslant 0 \tag{5.1}$$

Maximizing AUC

The Area under the ROC Curve (AUC) calculates the probability that a classifier $f(\cdot)$ will rank a randomly chosen positive sample *higher* than a randomly chosen negative sample. AUC varies between 0 and 1, where 1 represents all positives being ranked above the negatives. AUC may be estimated using the Wilcoxon-Mann-Whitney (WMW) Statistic Hanley and McNeil (1982), as

Definition 5.1 (AUC). *Let n be the number of samples. Let X_+ (and X_- resp.) be the set of positive (and negative resp.) samples such that $|X_+| + |X_-| = n$ where $|\cdot|$ is the cardinality of the set. Then, AUC is given as $\left(\sum_{i=1}^{|X_+|}\sum_{i=1}^{|X_-|}\mathbb{1}_{f(x_i)>f(x_j)}\right)/\left(|X_+|\,|X_-|\right)$ for $x_i : i \in \{1,\cdots,n\}$.*

Here, we follow Ataman et al. (2006) to calculate the AUC based on the WMW statistic as follows.

$$\min_{z_{ij}} \sum_{i=1}^{|X_+|} \sum_{j=1}^{|X_-|} z_{ij} \quad \text{s.t.} \quad f(x_i) - f(x_j) \geqslant \epsilon - z_{ij} \quad \text{where } x_i \in X_+, x_j \in X_-; \quad z_{ij} \geqslant 0, \tag{5.2}$$

where ϵ is a given constant. Problem (5.2) computes AUC indirectly by minimizing the number of pairs (one each from the positive and negative classes) where the positive sample is not ranked higher than the negative sample: so, the number of zero entries in z equals the number of pairs where this condition is not true. For a given z, we have that,

$$\text{AUC} = (n - \|z\|_0) \,/\, (|X_+|\,|X_-|) = \left(n - \sum_i \sum_j \epsilon^{-1}\text{relu}(0, -z_{ij} + \epsilon)\right) \Big/ (|X_+|\,|X_-|)\,.$$

If z_{ij} is 0, then $\epsilon^{-1}\text{relu}(0, -z_{ij} + \epsilon)$ equals 1. Otherwise $z_{ij} > 0$, it follows from the first constraint in (5.2), that $z_{ij} \geqslant \epsilon$, so $\epsilon^{-1}\text{relu}(0, -z_{ij} + \epsilon)$ equals 0. Observe that in (5.2), the number of constraints is $|X||X_-|$, which is $O\left(n^2\right)$.

Maximizing Multi-class AUC

An extension of AUC to the multi-class setting, AUC_μ, is defined in Kleiman and Page (2019). The AUC_μ objective optimizes an indicator matrix calculated on the orientation function, $O_{i,j}$ defined as,

Definition 5.2 (Orientation Function; Kleiman and Page (2019))**.** *Assume we have* K *classes* $\{y_1, \cdots, y_K\}$*. Let* $\mathbf{f}(x_i, -) \in \mathbb{R}^K$ *(extension of* $f(.)$ *to the multi-class setting) indicate the model's prediction on* x_i *for each of the* K *classes (class-specific probability). Let* x_i^* *provide the index of* x_i*'s true class label. Let* $\mathbf{P} \in \mathbb{R}^{k \times k}$ *be a partition matrix where* $\mathbf{P}_{k,k'}$ *is the cost of classifying a sample as class* k *when it should be* k'*. Define,*

$$\mathbf{v}_{kk'} = \mathbf{P}_{k,-} - \mathbf{P}_{k',-} \text{ and } \widetilde{\mathbf{v}} = \mathbf{v}_{x_i^* x_j^*} \in \mathbb{R}^K; \quad O_{i,j} = (\widetilde{\mathbf{v}}_{x_i^*} - \widetilde{\mathbf{v}}_{x_j^*})(\langle \widetilde{\mathbf{v}}, \mathbf{f}(x_i, -)\rangle - \langle \widetilde{\mathbf{v}}, \mathbf{f}(x_j, -)\rangle).$$

The LP formulation of AUC_μ is fully characterized by $\mathbf{P}$. We can set $\mathbf{P}(k, k) = 0\,\forall k$ and 1 for all other entries. We can also define a $\mathbf{P}$ with arbitrary entries or formulate AUC in a one-vs-all setting (denoted as AUC^{ova}). Here, for presentation, use the

simplified 0/1 partition matrix $\mathbf{P}$. Let

$$\widetilde{\mathbf{f}}(i,j) = \mathbf{f}(x_i, x_i^*) - \mathbf{f}(x_j, x_i^*) + \mathbf{f}(x_j, x_j^*) - \mathbf{f}(x_i, x_i^*) \tag{5.3}$$

Then the LP formulation is given by,

$$\text{AUC}_\mu^{\text{bin}} : \min_{z_{ij}} \sum_{i=1}^{n} \sum_{j=1:x_i^*<x_j^*}^{n} z_{ij} \quad \text{s.t.} \quad \widetilde{\mathbf{d}}_{ij}\widetilde{\mathbf{f}}(i,j) \geqslant \epsilon - z_{ij}, \quad \forall i,j : x_i^* < x_j^* \quad z_{ij} \geqslant 0 \tag{5.4}$$

where $\widetilde{\mathbf{d}}_{ij} = \widetilde{\mathbf{v}}_{x_i^*} - \widetilde{\mathbf{v}}_{x_j^*}$. Problem 5.4 can be seen as an extension of our binary AUC model, where z_{ij} is the ranking between a pair of points defined for multiple classes.

Maximizing F-score

The F-score (or F-measure) is a representative of objectives expressed as ratios of some combination of True positives (TP), False positives (FP), True negatives (TN) and False negatives (FN). We use the result in Dembczynski et al. (2011) to express the F-score in the ratio form and further relax it into the LP form. The detailed LP formulation of F-score and other ratio functions is in the appendix.

5.3 Backpropagation via Fast Exterior Penalty Optimization

Unlike traditional feedforward networks, where the output of each layer is a relatively simple (albeit non-linear) function of the previous layer, a LP layer must solve a constrained problem, therefore implementing scalable/efficient backpropagation schemes that minimizes overhead requires more care and is an active topic of research. This problem is, as discussed in the previous chapter, not unique to LPs and manifests in differentiable sorting Mena et al. (2018) and formulating quadratic or cone programs Amos et al. (2017). One may unroll gradient descent steps Amos et al. (2017); Goodfellow et al. (2013); Metz et al. (2016) or use projections Zeng et al. (2019). Recently Agrawal et al. (2019) introduced a package for differentiable constrained convex programming, which includes LPs as a special case. For LPs, Meng et al. (2020) presents an unrolled scheme and Blondel et al. (2020) shows that we can

differentiate through LP formulations of sorting/ranking *exactly* by using smooth approximations of projection steps. Berthet et al. (2020) describes an interesting approach where one computes approximate gradients through ranking/shortest path problems by stochastic perturbation techniques.

Remark 5.3. *Some previous works Zeng et al. (2019) have considered LPs where the constraints are deterministic (for a fixed input dimension), i.e., do not depend on the data* X*, which is different from the LPs in Sections 5.2–5.2.*

Note that perturbation techniques in Berthet et al. (2020) are applicable to our LPs as well. The Fenchel Young losses in Berthet et al. (2020) is attractive because there is no need to compute the Jacobian. Implementation-wise, one could think of the backward pass as a function given the input and output of the forward pass. But the gradient expressions of the losses involves an expectation and hence may require multiple calls to a LP solver in order to approximate the expectation. Parallelization and warm starts were shown to alleviate this dependency to some extent by sampling in parallel.

Rationale for our approach. Consider a LP with a large m number of constraints in fixed dimensions n ($n \ll m$). This assumption holds in all formulations in Section 5.2. This is because we assume that the architecture is fixed whereas minibatch size depends on the complexity of the task (stable gradient or when noise in gradient is high). Hence, solving such LPs using off-the-shelf solvers as in Berthet et al. (2020), in the general case, could slow down training. The strategy in Agrawal et al. (2019) does offer benefits over Amos and Kolter (2017) for sparse QPs. Our strategy is to run Mangasarian's Newton's method on an exterior penalty function of the LP. There are two advantages: **(i)** during **forward** pass, quadratic local convergence of Newton's method indicates that unrolling the method may be reasonable; and further **(ii)** based on the relationship between dual and primal variables, and the exactness of the exterior penalty we can show that **backward** pass is independent of m. We will discuss both results and some modifications to deal with discontinuous Hessian (and its inverse) that is required for Newton's method. A similar approach is adopted in Amos and Kolter (2017) where Primal-Dual Interior Point methods with implicit differentiation is used for differentiation purposes. But the exterior penalty in (5.5) satisfies a nice property: primal and dual solutions are related by a *closed form* expression which can be exploited for efficient backpropagation.

Forward Pass using Newton's Algorithm on a Sufficiently Reduced Space

Fast automatic (forward or reverse mode) differentiation requires performing the *forward* pass efficiently. In our setting, we seek to solve and backpropagate through an LP. We will focus on reverse mode differentiation since it is the most suitable for DNN training.

Given a primal LP, for a fixed accuracy $\varepsilon > 0$, Mangasarian (2004) solves an unconstrained problem,

$$\min_{\bar{y}} g(\bar{y}) := \frac{1}{2} \|\sigma(A\bar{y} - b)\|^2 + \varepsilon c^T \bar{y}, \tag{5.5}$$

where $\sigma(\cdot) = \max(\cdot, 0)$ represents the elementwise relu function. A modified Newton's method can be used to solve (5.5) that performs the following iterations:

$$\bar{y} = \bar{y} + \lambda \bar{d} \tag{5.6}$$

where

$$\bar{d} = \tilde{H}(\bar{y})^{-1} \nabla g(\bar{y}) := (\nabla^2 g(\bar{y}) + \rho I)^{-1} \nabla g(\bar{y}) \tag{5.7}$$

In large scale settings of A, b, such Newton methods are known to perform empirically better than gradient descent Mangasarian (2006); Keerthi et al. (2007). We will evaluate if this holds for our purposes shortly.

Why is Newton's method applicable for minibatches? In general, the convergence of Newton's method depends strongly on initialization (even for convex problems), i.e., we can only provide local convergence results. However, this is not the case for our problems since in our examples, either the level sets are bounded from below; or the feasible set is compact, as noted in Mangasarian (2004). There are two reasons why the above result, by itself, is insufficient for our purposes: **(i)** it assumes that we can perform line search to satisfy Armijo condition; **(ii)** even with line search, the result does not provide a *rate* of convergence. In DNN training, such line search based convergence results can be very expensive. The key difficulty is handling the discontinuity in the Hessian. As a remedy, we use self concordance of (5.5) to guarantee global convergence of (5.6) iterations for the exterior penalty formulation in (5.5). To do so, we first show a result (proof in appendix) that characterizes the discrepancy between the actual Hessian of (5.5) and the modified one in (5.6) when the

(feature) matrix A is randomly distributed.

Lemma 5.4. *Assume that A is a random matrix, and fix arbitrary $\bar{y} \in \mathbb{R}^n$. Then with probability one, g in (5.5) can be approximated by a quadratic function (given by $\tilde{H}, \nabla g(\bar{y})$) over a sufficiently small neighborhood of $\bar{y}$.*

Intuitively, Lemma 5.4 states that with probability one, each y has a neighborhood in which the Hessian is constant. In addition, the modified Hessian is nonsingular at all points (in particular the optimal $\bar{y}^*$), and so we can then show the following global convergence result.

Theorem 5.5. *Assume that the primal LP has a unique optimal solution, and that the level set $\{z : Az \leqslant b, c^T z \leqslant \alpha\}$ is bounded for all α (for dual feasibility). Then short step (no line search) Newton's method converges globally at a linear rate with local quadratic convergence.*

Proof. First, note that the objective function is piecewise quadratic since it is a sum of piecewise quadratic functions defined by coordinatewise relu function σ. In particular, g is self concordant since its third derivative is zero almost everywhere. Now setting $\rho < \varepsilon$, we see that an approximate solution of the problem with the modified Hessian is also an approximate solution to equation 3. Moreover, since the possible values of $\tilde{H}$ is finite, the local norm (also known as Newton's decrement) $\nabla g(\bar{y})^T \tilde{H}(\bar{y})^{-1} \nabla g(\bar{y})$ is finite. Hence, we can choose $\rho > 0$ so that there is a descent direction d, that is, there exists a step size $\lambda > 0$ such that $\lambda \nabla g(x)^T d < 0$. Finally, we use Theorem 4.1.12 in Nesterov (2013) to claim the desired result. □

The assumptions in Theorem 5.5 are standard: 1. uniqueness can easily be satisfied by randomly perturbing the cost vector; 2. in most of our formulations, we explicitly have bound constraints on the decision variables, hence level sets are bounded.

Remark 5.6. *Convergence in Thm. 5.5 is guaranteed under standard constraint qualification assumptions. Linear Independent Constraint Qualification (LICQ) is satisfied for AUC, and Multi-class AUC formulations in §5.2. But the F-score formulation does not satisfy LICQ, hence we need safeguarding principles in the initial iterations (until iterates get close to the optimal solution).*

Our analysis of the Newton's method for LP so far indicate that we may be able to use a constant step size λ (avoid linesearch) to obtain fast convergence provided

that we are able to choose ρ sufficiently small. For our purposes, we assume λ to be a hyperparameter and can be tuned by cross validation.

Sketching d using Sufficient Dimension Reduction (SDR). For training the backbone network, we have to compute d in (5.6) using A (and b) which could be very large within each training iteration. But each minibatch corresponds to an LP instantiation in (5.5), so it may suffice to compute an approximate d in each Newton's iteration. To approximate d, we use sufficient dimension reduction in which a lower dimensional LP obtained by projecting A is solved Zhang et al. (2020); Kim et al. (2020). That is, instead of the inverse of $\tilde{H}$, we will compute a lower dimensional sketch of $\tilde{H}$ by using SA (and Sb) instead of A (and b) for some sampling matrix S and solve problem (5.5). During each iteration, the metrics are computed on the current minibatch as is done in Fathony and Kolter (2020). Thus (without SDR) the memory cost is directly proportional to minibatch size viz., number of samples and feature dimensions. *The advantage of using SDR (as opposed to naive sketching) is that the lower dimensional space can also be chosen using data driven cross validation or other techniques, making it ideal for training purposes.* In essence, the size of minibatch does not matter – as long as the minibatch can be sufficiently reduced, our solver is directly applicable, especially for low resource, memory constrained settings. For a fixed (low) dimension parameter, our sketched $\underline{d}$ is,

$$\underline{d} = \underline{H}^{-1}\underline{g} \text{ where } \underline{g} = \frac{\partial}{\partial y}(\|\text{relu}(A\bar{y} - b)\|_2^2) \approx (SA)^T S \cdot \text{relu}(\text{sign}(A\bar{y} - b)), \tag{5.8}$$

$$\underline{H} \approx (\text{diag}(S \cdot \text{relu}(\text{sign}(A\bar{y} - b)) \odot S \cdot \text{relu}(\text{sign}(A\bar{y} - b))) \cdot SA)^T \cdot SA. \tag{5.9}$$

It is easy to see that the approximate update can be seen as equivalent to Iterative Hessian Sketch algorithm which has geometric convergence rate, see Pilanci and Wainwright (2016).

Backward Pass using Optimal Dual Variables Aided by Unrolling

The advantage of optimizing the exterior penalty in (5.5) is that given an iterate y_t, accuracy ε, we can get the optimal dual solution ν_t by simple thresholding, i.e., $\nu_t = 1/\varepsilon(A^T\sigma(A\bar{y} - b))$. By complementarity slackness, nonzero coordinates of ν_t specify the set of active constraints in $Az \leqslant b$. So, given an approximate y_t such that $\nabla g(\bar{y}_t)\tilde{H}(\bar{y}_t)^{-1}\nabla g(\bar{y}_t) \leqslant \varepsilon$, to get the primal solution z^*, we solve the "active" linear

system given by $\tilde{A}\tilde{b}$, where $\tilde{A}$ denotes the active rows of A and the corresponding subvector $\tilde{b}$. So, backpropagation through the layer reduces to computing derivatives of $\tilde{A}^{-1}\tilde{b}$ (simple via automatic differentiation). The appendix gives a systematic way of choosing ε.

How to backpropagate through unrolled iterations? We assume that the chain rule is applicable up to this LP layer and is $\frac{\partial L}{\partial z}\frac{\partial z}{\partial A}$ (for one of the parameters A), and note that it is possible to find $\frac{\partial L}{\partial z}$ (either directly or using a chain rule). Therefore we focus our attention on $\frac{\partial z}{\partial A}$, which involves the LP layer. Indeed, when we unroll the dynamics formed by optimization steps, each iteration in (5.6) is equivalent to a "sublayer". So, in order to backpropagate we have to show the partial derivatives of each operation or step wrt to the LP parameters A, b, and c.

Our goal is to calculate $\frac{\partial \bar{d}}{\partial A}$ where $\bar{d} = Q^{-1}q$, $Q = \tilde{H}$ and $w = \nabla g(\bar{y})$. We can use the product rule to arrive at: $\partial \bar{d} = -\left(w^T Q^{-1} \otimes Q^{-1}\right)\partial Q + Q^{-1}\partial w$. To see this, note that we have used the chain rule to differentiate through the inverse in the first term. The second term is easy to compute similar to the computation of Hessian. For each of these terms we eventually have to compute $\frac{\partial Q}{\partial \text{var}}$ or $\frac{\partial w}{\partial \text{var}}$ where $\text{var} \in \{c, A, b\}$ which can also be done by another application of chain rule. Please see appendix for empirical verification of unrolled gradient and the one provided by $\tilde{A}^{-1}\tilde{b}$.

Before proceeding, we should note an issue that comes up when differentiating each step of the unrolled algorithm because the Hessian is piecewise linear (constant) as a function of the input to that particular layer. Here, some possible numerical approximations are needed, as we describe below.

Remark 5.7. *Note that the diagonal matrix term in $\frac{\partial Q}{\partial A}$ is nondifferentiable due to the presence of the step function. However, the step function is a piecewise constant function, and hence has zero derivative almost surely, that is, in any bounded set $\underline{S}$, $z \in \underline{S}$, if a ball (of radius $r > 0$,) $B_r(z) \subseteq \underline{S}$, then the Lebesgue measure of the set of nondifferentiable points on $\underline{S}$ is zero. Please see appendix for a formal justification where we show this by approximating the step function using a sequence of logistic functions with increasing slope parameter at the origin.*

Therefore, in this setting, Remark 5.7 provides a way to compute an approximate sub-gradient when using Newtons method based LP layers. The function is a piecewise quadratic function and differentiable everywhere, and the inverse of the Hessian acts as a preconditioner.

Summary. Our forward pass involved three steps: **(1)** finite steps of Newton's method using which we **(2)** computed the dual variable by a thresholding operation, and **(3)** finally, to get the primal solution, these dual variables are first used to identify the active constraints followed by solving a linear system. To backpropagate through these three steps, we must differentiate through each layer of our procedure including $\tilde{A}^{-1}\tilde{b}$: this is independent of whether we use unrolling or Danskin's theorem Danskin (1966). For instance, using Danskin's theorem here would involve differentiating through the fixed point of the Newton's iterations similar to (regularized) gradient descent iterations in iMAML Rajeswaran et al. (2019).

5.4 Experiments

In this section, we conduct experiments on commonly used benchmarks to show that our framework can be used to optimize multiple different objectives within deep neural networks and lead to performance gain. We start with binary AUC optimization, and then extend to multi class AUC optimization and F-score optimization. We also show that nonnegative matrix factorization can be optimized in linear programming form in our framework.

Optimizing Binary AUC

We follow the current state-of-the-art work on AUC optimization Liu et al. (2019b) to conduct experiments on optimizing AUC score directly with deep neural networks. The baseline algorithms we compare with for binary AUC are cross-entropy loss and two algorithms (PPD-SG and PPD-AdaGrad) from Liu et al. (2019b).

Datasets: Cat&Dog, CIFAR10, CIFAR100, and STL10 (See supplement for introduction of these datasets). We follow Liu et al. (2019b) to use 19k/1k, 45k/5k, 45k/5k, 4k/1k training/validation split on Cat&Dog, CIFAR10, CIFAR100, STL10 respectively.

Experimental setting. We construct imbalanced binary classification task by using half of the classes as positive class and another half as negative class, and dropping samples from negative class by a certain ratio, which is reflected by the positive ratio (the ratio of the majority class to the minority class) in Table 5.1. During the experiment, We use the same random seed, learning rate and total number of iterations in all of our experiments including multi class AUC and F-score experiments. (See supplement for details of model architecture, learning rate, etc.)

Table 5.1: Binary AUC optimization results on four benchmark datasets.

AUC(%)	Cat&Dog				CIFAR10			
Positive Ratio	91%	83%	71%	50%	91%	83%	71%	50%
Cross-Entropy	67.6	74.6	85.1	87.4	65.2	73.3	78.1	**83.7**
PPD-SG	**79.1**	**81.5**	85.5	87.1	69.8	73.9	**79.1**	82.6
PPD-AdaGrad	77.3	80.6	83.7	86.3	69.7	74.1	78.4	83.1
Ours	78.6	81.3	**85.6**	**87.8**	**72.5**	**74.4**	78.3	82.7
AUC(%)	CIFAR100				STL10			
Positive Ratio	91%	83%	71%	50%	91%	83%	71%	50%
Cross-Entropy	57.8	58.4	62.2	66.3	63.5	67.1	72.7	80.8
PPD-SG	56.5	58.9	61.6	65.2	**70.7**	71.6	75.1	77.4
PPD-AdaGrad	56.2	59.0	62.6	67.6	68.5	**72.4**	76.7	78.5
Ours	**58.2**	**60.5**	**64.5**	**69.0**	68.4	71.1	**76.7**	**81.6**

Results. The results are shown in Table 5.1. We can see that our method slightly outperforms Liu et al. (2019b) and outperforms cross-entropy loss by a large margin, especially on imbalanced datasets, where the AUC objective shows superiority over cross-entropy loss.

Influence of ϵ. We test the influence of ϵ using Cat&Dog as an example. Results (see supplement for the table) show that $\epsilon = 0.1$ gets slightly worse performance than $\epsilon = 0.01$, while $\epsilon = 0.001$ performs much worse. To choose ϵ, we follow the approach proposed by Mangasarian (2004). If for two successive values of $\epsilon_1 > \epsilon_2$, the value of the ϵ perturbed quadratic function is the same, then it is the least 2-norm solution of the dual. Therefore, we simply choose an ϵ that satisfies this property, which is chosen to be 0.01 in our experiments.

Optimizing Multiclass AUC

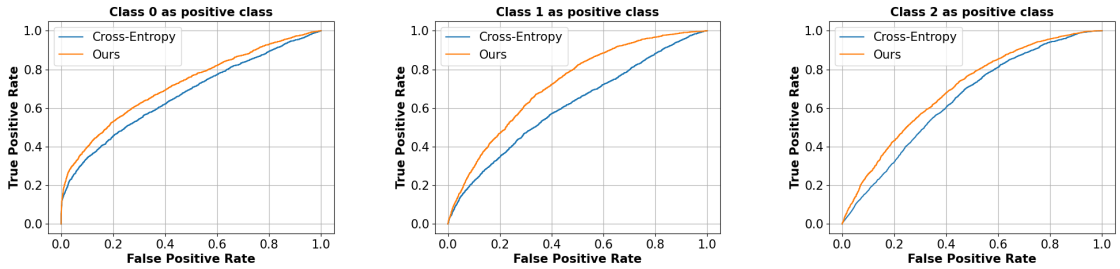
We further demonstrate our method for optimizing multiclass AUC. Similar to previous section on binary AUC, we construct imbalanced multiclass datasets by dividing datasets into 3 classes and drop samples from 2 of them and report the one-versus-all AUC (denoted as AUC^{ova}) and $\text{AUC}_{\mu}^{\text{bin}}$ score . For STL10, we group class $0-2, 3-5, 6-9$ into the three classes, and drop samples from the first two classes. For CIFR100, we group class $0-32, 33-65, 66-99$ into three classes, and also drop samples from the first two classes.

Results. Results are in Table 5.2. In addition to one-versus-all AUC metric, we also report the performance in terms of AUC_{μ} Kleiman and Page (2019) which is specifically designed for measuring multiclass AUC and preserves nice properties

Table 5.2: Multiclass AUC optimization results on STL10 and CIFAR100. Drop rate is the proportion used when dropping samples from two of three classes.

AUC^{ova}(%)	**CIFAR100**				**STL10**			
Drop rate	90%	80%	60%	0%	90%	80%	60%	0%
Cross-Entropy	54.3	**59.4**	62.7	63.5	66.9	68.0	74.8	81.0
Ours	**58.4**	59.2	**64.1**	**65.7**	**72.9**	**72.5**	**75.7**	**82.7**
AUC_{μ}^{bin}(%)	**CIFAR100**				**STL10**			
Drop rate	90%	80%	60%	0%	90%	80%	60%	0%
Cross-Entropy	55.1	60.6	65.0	64.0	68.9	69.6	75.8	82.2
Ours	**60.1**	**61.2**	**66.0**	**67.2**	**76.1**	**74.4**	**77.7**	**84.5**

of binary AUC such as being insensitive to class skew. We can see that our method outperforms cross-entropy loss on all four datasets and under all different skewed ratios.

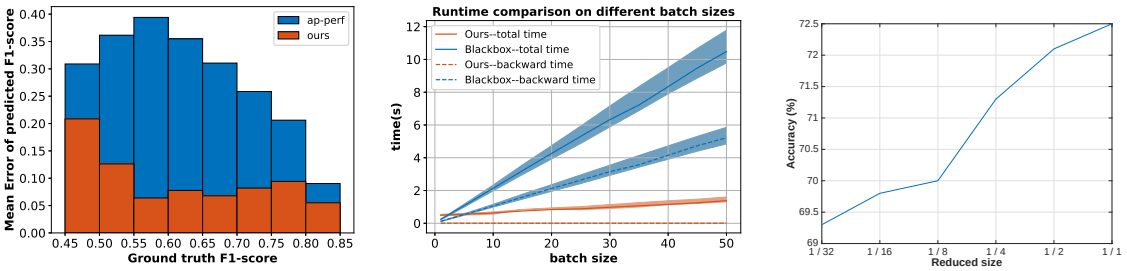


(a) Positive class: 0 (b) Positive class: 1 (c) Positive class: 2

Figure 5.2: ROC curve of multiclass AUC optimization on STL10 with 90% drop rate. We divide STL10 into 3 classes and use one as positive class and other two as negative class to plot the ROC.

Optimizing F1**-score**

We show that by directly optimizing F1-score, we can achieve a better performance than when using cross entropy loss. In addition to cross entropy loss, we perform evaluations with two other methods that can also directly optimize the F1-score. First, we replace our solver with CVXPY-SCS Agrawal et al. (2019), which is a differentiable general purpose linear programming solver; second, we perform comparisons with AP-Perf Fathony and Kolter (2020) which offers differentiable optimization of F1-score using an adversarial prediction framework. The datasets and our setup to group them into two classes remain the same as in binary AUC section. The results show that Our method yields consistent improvement over cross-entropy



(a) Evaluation of the quality of F1-score predicted by our solver and Ap-Perf.

(b) Runtime comparison of our solver and blackbox solver Pogančić et al. (2019)

(c) The relationship between reduced size and accuracy when using SDR on constraint matrix A.

Figure 5.3: Three numerical experiments to show the properties of our solver.

loss in terms of $\mathrm{F1-score}$. The accuracy of cross-entropy loss on Cat&Dog, CIFAR10, CIFAR100 and STL10 is 76.0%, 70.3%, 60.4%, 71.8%, respectively, while our method gives **77.8%**, **72.6%**, **63.4%**, **72.7%**. When we optimize F1-score directly, there exists a local optimal point where assigning all examples to the positive class leads to F1-score of 66.7%. Both CVXPY-SCS and AP-Perf fall into this local optimal on four datasets (except that CVXPY-SCS gets out of the local optimal point on Cat&Dog and gives 70.1% accuracy). Note that although our solver, CVXPY-SCS and AP-Perf are trying to solve the same objective, the backward gradient will be different due to different approximations used by the methods.

Table 5.3: Properties of different methods. c, A, b are the parameters describing a linear programming: $\min cx, \mathrm{s.t.} Ax \leqslant b$. x^* denotes the optimal solution. The extra cost for computing backward gradient for Berthet et al. (2020), depending on implementation, can either be N times time cost, or N times memory cost plus some time cost brought by increased batch size (see supplement for detailed explanation).

	Ours	CVXPY-SCS	Perturbed solver Berthet et al. (2020)
Gradients support	$\frac{dx^*}{dc}, \frac{dx^*}{dA}, \frac{dx^*}{db}$	$\frac{dx^*}{dc}, \frac{dx^*}{dA}, \frac{dx^*}{db}$	$\frac{dx^*}{dc}$
GPU support	✓	✗	depends on solver
Sparsity support	✓	✗	depends on solver
Extra cost for computing backward gradient (denote forward time cost as 1)	Nearly zero	around 1	N (# perturbations)

Comparing with blackbox solvers. Recently, several results Pogančić et al. (2019); Berthet et al. (2020) use blackbox solvers for solving combinatorial problems, which can also been used as a LP solver for problems where we need dx^*/dc (c is the cost vector in standard form LP). These methods can utilize existing LP solvers and compute backward gradient by calling the LP solvers on perturbed LP problems. To our

knowledge, the current state-of-the-art LP solvers are often CPU based, which means that one needs to transfer the LP parameters from GPU to CPU, then solve the LP on the CPU before getting the solution back. These CPU-based solvers will usually be slower than our GPU based solver, especially when the batch size is large. To evaluate this property, we randomly construct LP problems using AUC as an example and make the cost vector the parameter to optimize. In Fig. 5.3b, we compare the runtime of our solver and the blackbox solver in Pogančić et al. (2019) (with the LP solver from scipy package). We can see that our GPU based solver is not sensitive to the increase in the batch size, while the CPU based solver is, due to the lack of support for mini-batch operations. Our solver has a more clear advantage in backward pass because the matrix needed for computing gradient is already computed during forward pass thus the backward pass is nearly **free**, while the blackbox solver Pogančić et al. (2019) needs at least the same time as forward pass to compute backward gradient. We also compare our solver with Berthet et al. (2020) in Table 5.3.

Dealing with large scale LP. In practice we often encounter large scale LPs whose constraint matrix may be too large to fit into the GPU memory. We first show that we can utilize sufficient dimension reduction in Fig. 5.3c, which demonstrates a reasonable tradeoff between reduced size and accuracy. Then we show that when the constraint matrix is sparse, we can readily utilize sparse operations to save memory. We have incorporated functionality to use sparsity within our solver flaport (2020); rusty1s el al. (2020). Consider AUC maximization in which the constraint matrix is extremely sparse. With our sparse implementation, we are able to run SGD on Resnet-18 with minibatch size upto 200 on a 2080Ti GPU with just 1GB memory whereas the same problem can take upto 11GB using dense operations ($\approx 10\times$ memory savings) with some overhead.

Limitations. Scaling is still a key limitation for differentiable LP solvers. We present two ways to mitigate these problems but there is an associated cost. Our proposed SDR sacrifices some accuracy. Partly due to support within available libraries sparse operations can sometimes be slow when the size of the matrices is large which should improve with better sparsity support in existing libraries.

5.5 Summary

We demonstrated that various non-decomposable objectives can be optimized within deep neural networks in a differentiable way under the same general framework of LPs using a modified Newton's algorithm proposed by Mangasarian. A number of recent papers have studied the general problem of backpropagating through convex optimization modules, and this literature provides several effective approaches although scalability remains a topic of active research. Our work complements these results and shows that the operations needed can be implemented to utilize the capabilities of modern deep learning libraries. While our experimental results suggest that promising results on binary AUC, multi-class AUC and F-score optimization within DNNs is achievable, we believe that the module may have other applications where the number of constraints are large and data-dependent.

Chapter 6

An Online Riemannian PCA for Stochastic Canonical Correlation Analysis

In the previous chapters, the dynamical systems are instantiated and then integrated as layers and run independently for each input sample. In this chapter, we study the problem of streaming canonical correlation analysis, where the dynamics formed by the optimization steps happen across the entire training procedure when integrated into deep models. The key challenge here is that most existing methods for streaming CCA have high time and space complexity in each iteration, which may not fit the needs of modern size datasets. We demonstrate how to design a more efficient streaming CCA algorithm based on an observation of the link between PCA and CCA. The work covered in this chapter was published as a conference paper at NeurIPS 2021 Meng et al. (2021a).

6.1 Introduction

Canonical correlation analysis (CCA) is a classical method for evaluating correlations between two sets of variables. It is commonly used in unsupervised multi-view learning, where the multiple views of the data may correspond to image, text, audio and so on, Rupnik and Shawe-Taylor (2010); Chaudhuri et al. (2009); Luo et al. (2015), and has been applied to manifold-valued data also Kim et al. (2014). Classical formulations have also been extended to leverage advances in representation learning,

for example, Andrew et al. (2013) showed how the CCA can be interfaced with deep neural networks enabling modern use cases. Many results over the last few years have used CCA or its variants for problems including measuring representational similarity in deep neural networks Morcos et al. (2018) and speech recognition Couture et al. (2019).

The goal in CCA is to find linear combinations within two random variables $\mathbf{X}$ and $\mathbf{Y}$ which have maximum correlation with each other. Formally, the CCA problem is defined as follows. Let $X \in \mathbf{R}^{N \times d_x}$ and $Y \in \mathbf{R}^{N \times d_y}$ be N samples respectively drawn from pair of random variables $\mathbf{X}$ (d_x-variate random variable) and $\mathbf{Y}$ (d_y-variate random variable), with unknown joint probability distribution. The goal is to find the projection matrices $U \in \mathbf{R}^{d_x \times k}$ and $V \in \mathbf{R}^{d_y \times k}$, with $k \leqslant \min\{d_x, d_y\}$, such that the correlation is maximized:

$$\max_{U,V} \; F = \text{trace}\left(U^T C_{XY} V\right) \quad \text{s.t.} \qquad U^T C_X U = I_k, V^T C_Y V = I_k \tag{6.1}$$

Here, $C_X = \frac{1}{N} X^T X$ and $C_Y = \frac{1}{N} Y^T Y$ are the sample covariance matrices, and $C_{XY} = \frac{1}{N} X^T Y$ denotes the sample cross-covariance.

The objective in (6.1) is the expected cross-correlation in the projected space and the constraints specify that different canonical components should be decorrelated. Let us define the whitened covariance $T := C_X^{-1/2} C_{XY} C_Y^{-1/2}$ and Φ_k (and Ψ_k) contains the top-k left (and right) singular vectors of T. It is known Golub and Zha (1992) that the optimum of (6.1) is achieved at $U^* = C_X^{-1/2} \Phi_k$, $V^* = C_Y^{-1/2} \Psi_k$. We can compute U^*, V^* by applying a k-truncated SVD to T.

Runtime and memory considerations. The above procedure is simple but is only feasible when the data matrices are small. In modern applications, not only are the datasets large but also the dimension d (let $d = \max\{d_x, d_y\}$) of each sample can be large, especially if representations are learned using deep models. As a result, the resource needs of the algorithm can be high. This has motivated the study of stochastic optimization routines for solving CCA, and many efficient strategies have been proposed. For example, Ge et al. (2016); Wang et al. (2016) present Empirical Risk Minimization (ERM) models which optimize the empirical objective. More recently, Gao et al. (2019); Bhatia et al. (2018); Arora et al. (2017) describe proposals that optimize the population objective. To summarize the approaches, if we are satisfied with the top 1 component of CCA, effective schemes with $O(\frac{1}{t})$ convergence rate are available by utilizing either extensions of the Oja's rule Oja (1982) to the generalized

eigenvalue problem Bhatia et al. (2018) or the alternating SVRG algorithm Gao et al. (2019). Otherwise, a stochastic approach will use an explicit whitening operation which can cost d^3 operations for each iteration Arora et al. (2017) and the convergence rate for the stochastic scheme depends on its specific steps and calculations, e.g., $O(\frac{1}{\sqrt{t}})$ in Arora et al. (2017) (Thm 2.3, pp 5).

Observation. Most approaches either directly optimize (6.1) or instead a reparameterized or regularized form Ge et al. (2016); Allen-Zhu and Li (2016); Arora et al. (2017). Often, the search space for U and V corresponds to the entire $\mathbf{R}^{d\times k}$ (ignoring the constraints for the moment). But if the formulation could be cast in a form which involved approximately writing U and V as a product of structured matrices, we may be able to obtain specialized routines which are tailored to exploit those properties. Such a reformulation is not difficult to derive – where the matrices used to express U and V can be identified as objects that live in well studied geometric spaces. Then, utilizing the geometry of the space and borrowing relevant tools from differential geometry could lead to an efficient approximate scheme for top-k CCA which optimizes the population objective in a streaming fashion.

Contributions. (a) First, we re-parameterize the top-k CCA problem as an optimization problem on specific matrix manifolds, and show that it is equivalent to the original formulation in (6.1). (b) Informed by the geometry of the manifold, we derive stochastic gradient descent (SGD) algorithms for solving the re-parameterized problem with $O(d^2k)$ cost per iteration and provide convergence rate guarantees. (c) This analysis gives a direct mechanism to obtain an upper bound on the number of iterations needed to guarantee an ϵ error w.r.t. the population objective for the CCA problem. (d) The algorithm works in a streaming manner so it easily scales to large datasets and we do not need to assume access to the full dataset at the outset. (e) We present empirical evidence for the standard CCA model and the DeepCCA setting Andrew et al. (2013), describing advantages and limitations.

6.2 Stochastic CCA: Reformulation, Algorithm and Analysis

Let us review the objective for CCA as given in (6.1). We denote $X \in \mathbf{R}^{N\times d_x}$ as the matrix consisting of the samples $\{\mathbf{x}_i\}$ drawn from a zero mean random variable $\mathbf{X} \sim \mathcal{X}$ and $Y \in \mathbf{R}^{N\times d_y}$ denotes the matrix consisting of samples $\{\mathbf{y}_i\}$ drawn from a

zero mean random variable $\mathbf{Y} \sim \mathcal{Y}$. For simplicity, we assume that $d_x = d_y = d$ although the results hold for general d_x and d_y. Also recall that $C_X \in \mathbf{R}^{d_x \times d_x}$ (and C_Y resp.) is the covariance matrix of $\mathbf{X}$ (and $\mathbf{Y}$ resp.) and C_{XY} is the cross-covariance matrix between $\mathbf{X}$ and $\mathbf{Y}$. Let $U \in \mathbf{R}^{d \times k}$ ($V \in \mathbf{R}^{d \times k}$) be the matrix consisting of $\{\mathbf{u}_j\}$ ($\{\mathbf{v}_j\}$) where $(\{\mathbf{u}_j\}, \{\mathbf{v}_j\})$ are the canonical directions. The constraints in (6.1) are called *whitening constraints*.

Reformulation: In the CCA formulation, the matrices consisting of canonical correlation directions, i.e., U and V, are unconstrained, hence the search space is the entire $\mathbf{R}^{d \times k}$. Now, we reformulate the CCA objective by reparameterizing U and V. In order to do that, let us take a brief detour and recall the objective function of principal component analysis (PCA):

$$\widehat{U} = \arg\max_{U'} \quad \text{trace}(\widehat{R}) \qquad \text{subject to} \qquad \widehat{R} = U'^{T} C_X U'; \quad U'^{T} U' = I_k \tag{6.2}$$

Observe that by performing PCA and assigning $U = \widehat{U}\widehat{R}^{-1/2}$ in (6.1) (analogous for V using C_Y), we can satisfy the whitening constraint. Of course, writing $U = \widehat{U}\widehat{R}^{-1/2}$ does satisfy the whitening constraint, but such a U (and V) will not maximize trace $\left(U^T C_{XY} V\right)$, the objective of (6.1). Hence, additional work beyond the PCA solution is needed. Let us start from $\widehat{R}$ but relax the PCA solution by using an arbitrary $\widetilde{R}$ instead of diagonal $\widehat{R}$ (this will still satisfy the whitening constraint).

Write $U = \widetilde{U}\widetilde{R}$ with $\widetilde{U}^T\widetilde{U} = I_k$ and $\widetilde{R} \in \mathbf{R}^{k \times k}$. Thus we can approximate CCA objective (we will later check how good this approximation is) as

$$\max_{\substack{\widetilde{U},\widetilde{V} \in St(k,d) \\ R_u, R_v \in \mathbf{R}^{k \times k} \\ U = \widetilde{U}R_u;\ V = \widetilde{V}R_v}} \underbrace{\text{trace}\left(U^T C_{XY} V\right)}_{\widetilde{F}} + \underbrace{\text{trace}\left(\widetilde{U}^T C_X \widetilde{U}\right) + \text{trace}\left(\widetilde{V}^T C_Y \widetilde{V}\right)}_{\widetilde{F}_{pca}} \tag{6.3}$$

$$\text{s.t. } U^T C_X U = I_k;\ V^T C_Y V = I_k \tag{6.4}$$

Here, $St(k, d)$ denotes the manifold consisting of $d \times k$ (with $k \leqslant d$) column orthonormal matrices, i.e., $St(k, d) = \left\{X \in \mathbf{R}^{d \times k} | X^T X = I_k\right\}$. Observe that in (6.3), we approximate the optimal U and V as a linear combination of $\widetilde{U}$ and $\widetilde{V}$ respectively. *Thus, the aforementioned PCA solution can act as a feasible initial solution for (6.3).*

As the choice of R_u and R_v is arbitrary, we can further reparameterize these matrices by constraining them to be full rank (of rank k) and using the RQ decomposition Golub and Reinsch (1971) which gives us the following reformulation.

A Reformulation for CCA

$$\max_{\substack{\widetilde{U},\widetilde{V},S_u,S_v,Q_u,Q_v \\ U=\widetilde{U}S_uQ_u;\ V=\widetilde{V}S_vQ_v}} \quad \underbrace{\operatorname{trace}\left(U^T C_{XY} V\right)}_{\widetilde{F}} + \underbrace{\operatorname{trace}\left(\widetilde{U}^T C_X \widetilde{U}\right) + \operatorname{trace}\left(\widetilde{V}^T C_Y \widetilde{V}\right)}_{\widetilde{F}_{pca}} \tag{6.5a}$$

$$\begin{aligned} \text{subject to} \quad & U^T C_X U = I_k \\ & V^T C_Y V = I_k \\ & \widetilde{U}, \widetilde{V} \in St(k,d);\ Q_u, Q_v \in SO(k) \\ & S_u, S_v \text{ is upper triangular} \end{aligned} \tag{6.5b}$$

Here, $SO(k)$ is the space of $k \times k$ special orthogonal matrices, i.e.,

$$SO(k) = \left\{X \in \mathbf{R}^{k\times k} | X^T X = I_k; \det(X) = 1\right\} \tag{6.6}$$

Before evaluating how good the aforementioned approximation is, we first point out some useful properties of the reformulation (6.5): **(a)** in the reparametrization of U and V, all components are structured, hence, the search space becomes a subset of $\mathbf{R}^{k\times k}$ **(b)** we can essentially initialize with a PCA solution and then try to optimize (6.5) via some scheme.

Why (6.5) helps? First, we note that CCA seeks to maximize the total correlation under the constraint that different components are decorrelated. One difficulty in the optimization is to ensure decorrelation, which leads to a higher complexity in existing streaming CCA algorithms. On the contrary, in (6.5), we separate (6.1) into finding the PCs, $\widetilde{U}, \widetilde{V}$ (by adding the variance maximization terms) and finding the linear combination (S_uQ_u and S_vQ_v) of the principal directions. After optimizing for these variables, the whitening constraints are, up to a rescaling, automatically satisfied. Here, we can (almost) utilize an efficient off-the-shelf streaming PCA algorithm. We will defer describing the specific details of the individual steps until the next sub-section. First, we will show why substituting (6.1) with (6.5) is sensible under some assumptions.

Why the solution of the reformulation makes sense? We start by stating some mild assumptions needed for the analysis. **Assumptions: (a)** The random variables $\mathbf{X} \sim \mathcal{N}(\mathbf{0}, \Sigma_x)$ and $\mathbf{Y} \sim \mathcal{N}(\mathbf{0}, \Sigma_y)$ with covariance $\Sigma_x \preceq cI_d$ and covariance $\Sigma_y \preceq cI_d$ for some $c > 0$. **(b)** The samples X and Y drawn from $\mathcal{X}$ and $\mathcal{Y}$ respectively have zero mean. **(c)** For a given $k \leqslant d$, Σ_x, Σ_y have non-zero top-k eigen values.

We show how the presented solution, assuming access to an effective numerical procedure, approximates the CCA problem presented in (6.1). We formally state the result in the following theorem with a sketch of proof (appendix includes the full proof) by first stating the following proposition.

Definition 6.1. *A random variable* $\mathbf{X}$ *is called sub-Gaussian if the norm given by* $\|\mathbf{X}\|_\star := \inf\{d \geqslant 0 | \mathbf{E}_{\mathbf{X}}[\exp(^{trace(X^TX)}/d^2)] \leqslant 2\}$ *is finite. Let* $U \in \mathbf{R}^{d\times k}$*, then* $\mathbf{X}U$ *is sub-Gaussian Vershynin (2017).*

Proposition 1 (Reiß et al. (2020))**.** *Let* $\mathbf{X}$ *be a random variable which follows a sub-Gaussian distribution. Let* $\widehat{X}$ *be the approximation of* $X \in \mathbf{R}^{N\times d}$ *(samples drawn from* $\mathcal{X}$*) with the top-*k *principal vectors. Let* $\widetilde{C}_X$ *be the covariance of* $\widehat{X}$*. Also, assume that* λ_i *is the* i^{th} *eigen value of* C_X *for* $i = 1, \cdots, d-1$ *and* $\lambda_i \geqslant \lambda_{i+1}$ *for all* i*. Then, the PCA reconstruction error, denoted by* $\mathcal{E}_k = \mathbf{E}_{\mathbf{X}}\|X - \widehat{X}\|$ *(in the Frobenius norm sense) can be upper bounded as follows*

$$\mathcal{E}_k \leqslant \min\left(\sqrt{2k}\|\Delta\|_2, \frac{2\|\Delta\|_2^2}{\lambda_k - \lambda_{k+1}}\right), \quad \Delta = C_X - \widetilde{C}_X.$$

The aforementioned proposition suggests that the error between the data matrix X and the reconstructed data matrix $\widehat{X}$ using the top-k principal vectors is bounded.

Recall from (6.1) and (6.5) that the optimal value of the true and approximated CCA objective is denoted by F and $\widetilde{F}$ respectively. The following theorem states that we can bound the error, $E = \|F - \widetilde{F}\|$ (proof in the appendix). In other words, if we start from PCA solution and can successfully optimize (6.5) without leaving the feasible set, we will obtain a good solution.

Theorem 6.2. *Using the hypothesis and assumptions above, the approximation error* $E = \|F - \widetilde{F}\|$ *as a function of* N *is bounded and goes to zero as* $N \to \infty$ *while the whitening constraints in (6.5b) are satisfied.*

Sketch of the Proof. Let U^* and V^* be the true solution of CCA, i.e., of (6.1). Let $U = \widetilde{U}S_uQ_u, V = \widetilde{V}S_vQ_v$ be the solution of (6.5), with $\widetilde{U}, \widetilde{V}$ be the PCA solutions of X and Y respectively. Let $\widehat{X} = X\widetilde{U}\widetilde{U}^T$ and $\widehat{Y} = Y\widetilde{V}\widetilde{V}^T$ be the reconstruction of X and Y using principal vectors. Let $S_uQ_u = \widetilde{U}^TU^*$ and $S_vQ_v = \widetilde{V}^TV^*$. Then we can write $\widetilde{F} = \text{trace}\left(U^TC_{XY}V\right) = \text{trace}\left(\frac{1}{N}\left(\widehat{X}U^*\right)^T\widehat{Y}V^*\right)$. Similarly we can write $F = \text{trace}\left(\frac{1}{N}\left(XU^*\right)^TYV^*\right)$. As $\widehat{X}$ and $\widehat{Y}$ are the approximation of X and Y respectively using the principal vectors, we use Prop. 1 to bound the error $\|F - \widetilde{F}\|$. Now

observe that $\widehat{X}U$ can be rewritten into $X\widetilde{U}\widetilde{U}^T U$ (similar for $\widehat{Y}V$). Thus, as long as the solution $S_u Q_u$ and $S_v Q_v$ respectively well-approximate $\widetilde{U}^T U$ and $\widetilde{V}^T V$, $\widetilde{F}$ is a good approximation of F. □

Now, the only unresolved issue is an optimization scheme for (6.5a) that keeps the constraints in (6.5b) satisfied by leveraging the geometry of the structured solution space.

How to numerically optimize (6.5a) satisfying constraints in (6.5b)?

Overview. We now describe how to maximize the formulation in (6.5a)–(6.5b) with respect to $\widetilde{U}$, $\widetilde{V}$, Q_u, Q_v, S_u and S_v. We will first compute top-k principal vectors to get $\widetilde{U}$ and $\widetilde{V}$. Then, we will use a gradient update rule to solve for Q_u, Q_v, S_u and S_v to improve the objective. Since all these matrices are "structured", care must be taken to ensure that the matrices *remain on their respective manifolds* – which is where the geometry of the manifolds will offer desirable properties. We re-purpose a Riemannian stochastic gradient descent (RSGD) to achieve this task, so call our algorithm *RSG+*. Of course, more sophisticated Riemannian optimization techniques can be substituted in. For instance, different Riemannian optimization methods are available in Absil et al. (2007) and optimization schemes for many manifolds are offered in PyManOpt Boumal et al. (2014).

The algorithm block is in Algorithm 2. Recall that

$$\widetilde{F}_{pca} = \text{trace}\left(U^T C_X U\right) + \text{trace}\left(V^T C_Y V\right) \tag{6.7}$$

is the contribution from the principal directions which we used to ensure the "whitening constraint". Moreover, $\widetilde{F} = \text{trace}\left(U^T C_{XY} V\right)$ is the contribution from the canonical correlation directions (*note that we use the subscript 'cca' for making CCA objective explicit*). The algorithm consists of four main blocks denoted by different colors, namely **(a)** the Red block deals with gradient calculation of the objective function where we calculate the top-k principal vectors (denoted by $\widetilde{F}_{pca}$) with respect to $\widetilde{U}$, $\widetilde{V}$; **(b)** the Green block describes calculation of the gradient corresponding to the canonical directions (denoted by $\widetilde{F}$) with respect to $\widetilde{U}$, $\widetilde{V}$, S_u, S_v, Q_u and Q_v; **(c)** the Gray block combines the gradient computation from both $\widetilde{F}_{pca}$ and $\widetilde{F}$ with respect to unknowns $\widetilde{U}$, $\widetilde{V}$, S_u, S_v, Q_u and Q_v; and finally **(d)** the Blue block performs a batch update of the canonical directions $\widetilde{F}$ using Riemannian gradient updates.

Gradient calculations. The gradient update for $\widetilde{U}, \widetilde{V}$ is divided into two parts **(a)** The (Red block) gradient updates the "principal" directions (denoted by $\nabla_{\widetilde{U}}\widetilde{F}_{pca}$ and $\nabla_{\widetilde{V}}\widetilde{F}_{pca}$), which is specifically designed to satisfy the *whitening constraint*. This requires updating the principal subspaces, so, the gradient descent needs to proceed on the manifold of k-dimensional subspaces of $\mathbf{R}^d$, i.e., on the Grassmannian $Gr(k, d)$. **(b)** The (green block) gradient from the objective function in (6.5), is denoted by $\nabla_{\widetilde{U}}\widetilde{F}$ and $\nabla_{\widetilde{V}}\widetilde{F}$. In order to ensure that the Riemannian gradient update for $\widetilde{U}$ and $\widetilde{V}$ stays on the manifold $St(k, d)$, we need to make sure that the gradients, i.e., $\nabla_{\widetilde{U}}\widetilde{F}$ and $\nabla_{\widetilde{V}}\widetilde{F}$ lies in the tangent space of $St(k, d)$. To do so, we need to first calculate the Euclidean gradient and then project on to the tangent space of $St(k, d)$.

The gradient updates for Q_u, Q_v, S_u, S_v are given in the green block, denoted by $\nabla_{Q_u}\widetilde{F}$, $\nabla_{Q_v}\widetilde{F}$, $\nabla_{S_u}\widetilde{F}$ and $\nabla_{S_v}\widetilde{F}$. Note that unlike the previous step, this gradient only has components from canonical correlation calculation. As before, this step requires first computing the Euclidean gradient and then projecting on to the tangent space of the underlying Riemannian manifolds involved, i.e., $SO(k)$ and the space of upper triangular matrices.

Finally, we get the gradient to update the canonical directions by combining the gradients which is shown in the gray block. With these gradients we can perform a batch update as shown in the blue block. A schematic diagram is given in Fig. 6.1.

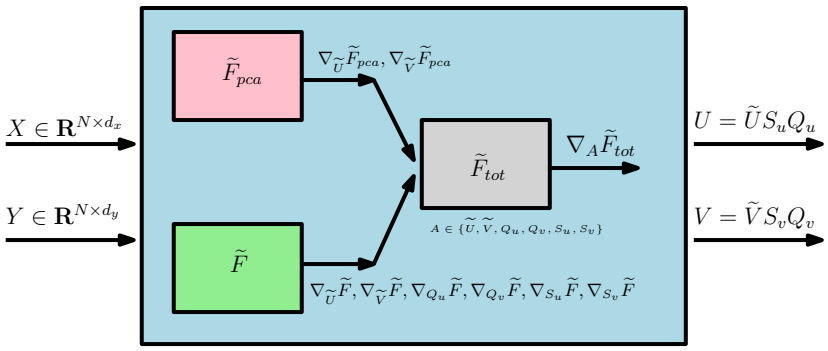


Figure 6.1: Schematic diagram of the proposed CCA algorithm, here $\widetilde{F}_{tot} = \widetilde{F} + \widetilde{F}_{pca}$, where $\widetilde{F}$ is the approximated objective value for CCA (as in (6.5))

Using results presented next in Propositions 2–3, this scheme can be shown (under some assumptions) to approximately optimize the CCA objective in (6.1).

We can now move to the convergence properties of the algorithm. We present two results stating the asymptotic proof of convergence for top-k principal vectors

and canonical directions in the algorithm.

Proposition 2 (Chakraborty et al. (2020))**.** *(Asymptotically) If the samples, X, are drawn from a Gaussian distribution, then the gradient update rule presented in Step C.2 in Algorithm 2 returns an orthonormal basis – the top-k principal vectors of the covariance matrix C_X.*

Proposition 3. *(Bonnabel (2013)) Consider a connected Riemannian manifold $\mathcal{M}$ with injectivity radius bounded from below by $I > 0$. Assume that the sequence of step sizes (γ_l) satisfy the condition (a) $\sum \gamma_l^2 < \infty$ (b) $\sum \gamma_l = \infty$. Suppose $\{A_l\}$ lie in a compact set $K \subset \mathcal{M}$. We also suppose that $\exists D > 0$ such that, $g_{A_l}\left(\nabla_{A_l}\widetilde{F}, \nabla_{A_l}\widetilde{F}\right) \leqslant D$. Then $\nabla_{A_l}\widetilde{F} \to 0$ and $l \to \infty$.*

Notice that in our problem, the injectivity radius bound in Proposition 3 is satisfied as "I" for $Gr(p,n)$, $St(p,n)$ or $SO(p)$ is $\pi/2\sqrt{2}, \pi/2\sqrt{2}, \pi/2$ respectively. So, in order to apply Proposition 3, we need to guarantee the step sizes satisfy the aforementioned condition. One example of the step sizes that satisfies the property is $\gamma_l = \frac{1}{l+1}$.

Convergence rate and complexity of the RSG+ algorithm

In this section, we describe the convergence rate and complexity of the algorithm proposed in Algorithm 2. Observe that the key component of Algorithm 2 is a Riemannian gradient update. Let A_t be the generic entity needed to be updated in the algorithm using the Riemannian gradient update $A_{t+1} = \mathrm{Exp}_{A_t}\left(-\gamma_t \nabla_{A_t}\widetilde{F}\right)$, where γ_t is the step size at time step t. Also assume $\{A_t\} \subset \mathcal{M}$ for a Riemannian manifold $\mathcal{M}$. The following proposition states that under certain assumptions, the Riemannian gradient update has a convergence rate of $O\left(\frac{1}{t}\right)$.

Proposition 4. *(Nemirovski et al. (2009); Bécigneul and Ganea (2018)) Let $\{A_t\}$ lie inside a geodesic ball of radius less than the minimum of the injectivity radius and the strong convexity radius of $\mathcal{M}$. Assume $\mathcal{M}$ to be a geodesically complete Riemannian manifold with sectional curvature lower bounded by $\kappa \leqslant 0$. Moreover, assume that the step size $\{\gamma_t\}$ diverges and the squared step size converges. Then, the Riemannian gradient descent update given by $A_{t+1} = Exp_{A_t}\left(-\gamma_t \nabla_{A_t}\widetilde{F}\right)$ with a bounded $\nabla_{A_t}\widetilde{F}$, i.e., $\|\nabla_{A_t}\widetilde{F}\| \leqslant C < \infty$ for some $C \geqslant 0$, converges in the rate of $O\left(\frac{1}{t}\right)$ with the number of iterates bounded by $\mathcal{O}(N + D/\epsilon^2)$, for some tolerance $\epsilon > 0$ and for the Lipschitz bound D of the objective function $\widetilde{F}$.*

Algorithm 2: Riemannian SGD based algorithm (RSG+) to compute canonical directions

Input: $X \in \mathbf{R}^{N \times d_x}$, $Y \in \mathbf{R}^{N \times d_y}$, $k > 0$

Output: $U \in \mathbf{R}^{d_x \times k}$, $V \in \mathbf{R}^{d_y \times k}$

Initialize $\widetilde{U}, \widetilde{V}, Q_u, Q_v, S_u, S_v$

Partition data X, Y into batches of size B. Let j^{th} batch be denoted by X_j and Y_j

for $j \in \left\{1, \cdots, \lfloor \frac{N}{B} \rfloor\right\}$ **do**

Gradient for top-k principal vectors: calculating $\nabla_{\widetilde{U}} \widetilde{F}_{pca}, \nabla_{\widetilde{V}} \widetilde{F}_{pca}$

1. Partition X_j (Y_j) into L ($L = \lfloor \frac{B}{k} \rfloor$) blocks of size $d_x \times k$ ($d_y \times k$);
2. Let the l^{th} block be denoted by Z_l^x (Z_l^y);
3. Orthogonalize each block and let the orthogonalized block be denoted by $\hat{Z}_l^x$ ($\hat{Z}_l^y$);
4. Let the subspace spanned by each $\hat{Z}_l^x$ (and $\hat{Z}_l^y$) be $\hat{\mathcal{Z}}_l^x \in \mathsf{Gr}(k, d_x)$ (and $\hat{\mathcal{Z}}_l^y \in \mathsf{Gr}(k, d_y)$);

$$\nabla_{\widetilde{U}} \widetilde{F}_{pca} = -\sum_l \mathsf{Exp}_{\widetilde{U}}^{-1}\left(\hat{\mathcal{Z}}_l^x\right) \quad \nabla_{\widetilde{V}} \widetilde{F}_{pca} = -\sum_l \mathsf{Exp}_{\widetilde{V}}^{-1}\left(\hat{\mathcal{Z}}_l^y\right) \tag{6.8}$$

Gradient from (6.5): calculating $\nabla_{\widetilde{U}} \widetilde{F}, \nabla_{\widetilde{V}} \widetilde{F}, \nabla_{Q_u} \widetilde{F}, \nabla_{Q_v} \widetilde{F}, \nabla_{S_u} \widetilde{F}, \nabla_{S_v} \widetilde{F}$

$$\nabla_{\widetilde{U}} \widetilde{F} = \frac{\partial \widetilde{F}}{\partial \widetilde{U}} - \widetilde{U} \frac{\partial \widetilde{F}}{\partial \widetilde{U}}^T \widetilde{U} \quad \nabla_{\widetilde{V}} \widetilde{F} = \frac{\partial \widetilde{F}}{\partial \widetilde{V}} - \widetilde{V} \frac{\partial \widetilde{F}}{\partial \widetilde{V}}^T \widetilde{V}$$
$$\nabla_{Q_u} \widetilde{F} = \frac{\partial \widetilde{F}}{\partial Q_u} - \frac{\partial \widetilde{F}}{\partial Q_u}^T \quad \nabla_{Q_v} \widetilde{F} = \frac{\partial \widetilde{F}}{\partial Q_v} - \frac{\partial \widetilde{F}}{\partial Q_v}^T$$
$$\nabla_{S_u} \widetilde{F} = \mathrm{Upper}\left(\frac{\partial \widetilde{F}}{\partial S_u}\right) \quad \nabla_{S_v} \widetilde{F} = \mathrm{Upper}\left(\frac{\partial \widetilde{F}}{\partial S_v}\right)$$

Here, Upper returns the upper triangular matrix of the input matrix and $\frac{\partial \widetilde{F}}{\partial \widetilde{U}}, \frac{\partial \widetilde{F}}{\partial \widetilde{V}}, \frac{\partial \widetilde{F}}{\partial Q_u}, \frac{\partial \widetilde{F}}{\partial Q_v}, \frac{\partial \widetilde{F}}{\partial S_u}, \frac{\partial \widetilde{F}}{\partial S_v}$ give the Euclidean gradients, which are provided in appendix.

Gradient to update canonical directions

$\nabla_{\widetilde{U}} \widetilde{F}_{tot} = \nabla_{\widetilde{U}} \widetilde{F}_{pca} + \nabla_{\widetilde{U}} \widetilde{F}$ $\quad$ $\nabla_{\widetilde{V}} \widetilde{F}_{tot} = \nabla_{\widetilde{V}} \widetilde{F}_{pca} + \nabla_{\widetilde{V}} \widetilde{F}$;
$\nabla_X \widetilde{F}_{tot} = \nabla_X \widetilde{F}$ where, X is a generic entity: $X \in \{Q_u, Q_v, S_u, S_v\}$;

Batch update of canonical directions

$A = \mathsf{Exp}_A\left(-\gamma_j \nabla_A \widetilde{F}_{tot}\right)$ where, A is a generic entity: $A \in \{\widetilde{U}, \widetilde{V}, Q_u, Q_v, S_u, S_v\}$;

end for

$U = \widetilde{U} Q_u S_u$ and $V = \widetilde{V} Q_v S_v$;

For this result to be applicable, we need the CCA objective function to be geodesically convex as a function of U and V (proof in the appendix). All Riemannian manifolds we used, i.e., $\mathsf{Gr}(k, d)$, $\mathsf{St}(k, d)$ and $\mathsf{SO}(k)$ are geodesically complete, and these manifolds have non-negative sectional curvatures, i.e., lower bounded by $\kappa = 0$. Moreover the minimum of convexity and injectivity radius for $\mathsf{Gr}(k, d)$, $\mathsf{St}(k, d)$ and $\mathsf{SO}(k)$ are $\pi/2\sqrt{2}$. Now, as long as the Riemannian updates lie inside the geodesic ball of radius less than $\pi/2\sqrt{2}$, the convergence rate for RGD applies in our setting.

Running time. To evaluate time complexity, we must look at the main compute-heavy steps needed. The basic modules are Exp and Exp^{-1} maps for $\mathsf{St}(k, d)$, $\mathsf{Gr}(k, d)$ and $\mathsf{SO}(k)$ manifolds (see Table 1 in appendix for a detailed specification of these maps). Observe that the complexity of these modules is influenced by the complexity of svd needed for the Exp map for the St and Gr manifolds. Our algorithm involves structured matrices of size $d \times k$ and $k \times k$, so any matrix operation should not exceed a cost of $O(\max(d^2k, k^3))$, since in general $d \gg k$. Specifically, the most expensive calculation is SVD of matrices of size $d \times k$, which is $O(d^2k)$, see Golub and Reinsch (1971). All other calculations are dominated by this term.

6.3 Experiments

We first evaluate RSG+ for extracting top-k canonical components on three benchmark datasets and show that it performs favorably compared with Arora et al. (2017). Then, we show that RSG+ also fits into feature learning in DeepCCA Andrew et al. (2013), and can scale to large feature dimensions where the non-stochastic method fails. Finally, we show that RSG+ can be used to improve fairness of deep neural networks without full access to labels of protected attributes during training.

CCA on Fixed Datasets

Datasets and baseline. We conduct experiments on three benchmark datasets (MNIST LeCun et al. (2010), Mediamill Snoek et al. (2006) and CIFAR-10 Krizhevsky (2009)) to evaluate the performance of RSG+ to extract top-k canonical components. To our knowledge, Arora et al. (2017) is the only previous work which stochastically optimizes the population objective in a streaming fashion and can extract top-k components, so we compare our RSG+ with the matrix stochastic gradient (MSG) method proposed in Arora et al. (2017) (note: there are two methods proposed in Arora et al.

(2017) and we choose MSG because it performs better in the experiments in Arora et al. (2017)). The details regarding the three datasets and how we process them are as follows:

MNIST. LeCun et al. (2010): MNIST contains grey-scale images of size 28×28. We use its full training set containing 60K images. Every image is split into left/right half, which are used as the two views. **Mediamill.** Snoek et al. (2006): Mediamill contains around 25.8K paired features of videos and corresponding commentary of dimension $120, 101$ respectively. **CIFAR-10.** Krizhevsky (2009): CIFAR-10 contains 60K 32×32 color images. Like MNIST, we split the images into left/right half and use them as two views.

Evaluation metric. We choose to use Proportion of Correlations Captured (PCC) which is widely used Ma et al. (2015); Ge et al. (2016), partly due to its efficiency, especially for relatively large datasets. Let $\hat{U} \in R^{d_x \times k}$, $\hat{V} \in R^{d_y \times k}$ denote the estimated subspaces returned by RSG+, and $U^* \in R^{d_x \times k}$, $V^* \in R^{d_y \times k}$ denote the true canonical subspaces (all for top-k). The PCC is defined as $\text{PCC} = \frac{\text{TCC}(X\hat{U}, Y\hat{V})}{\text{TCC}(XU^*, YV^*)}$, where TCC is the sum of canonical correlations between two matrices.

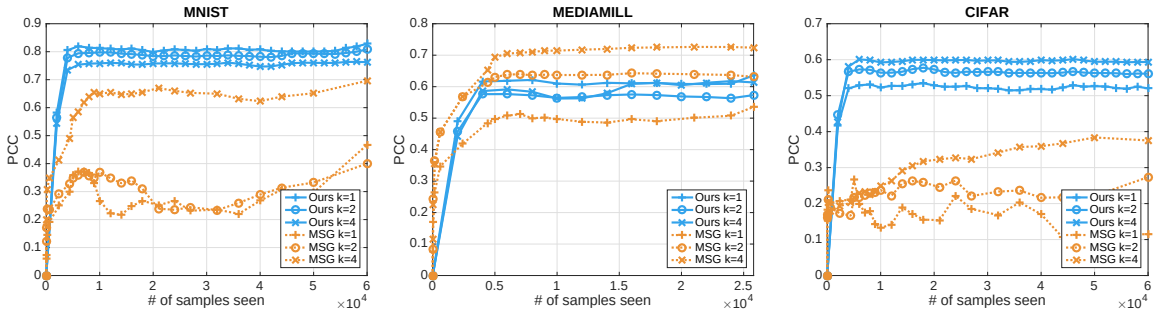
Performance. We run our algorithm with step sizes chosen from {1, 0.1, 0.01, 0.001, 0.0001, 0.00001}. The performance in terms of PCC as a function of the number of seen samples (shown in a streaming manner) are shown in Fig. 6.2, and our RSG+ achieves around $10\times$ runtime improvement over MSG (see Table C.1). Our RSG+ captures more correlation than MSG Arora et al. (2017) while being $5 - 10$ times faster. One case where our RSG+ underperforms Arora et al. (2017) is when the top-k eigenvalues are dominated by the top-l eigenvalues with $l < k$ (Fig. 6.2b): on Mediamill dataset, the top-4 eigenvalues of the covariance matrix in view 1 are: $8.61, 2.99, 1.15, 0.37$. The first eigenvalue is dominantly large compared to the rest and our RSG+ performs better for $k = 1$ and worse than Arora et al. (2017) for $k = 2, 4$. Runtime of RSG+ for different data dimensions (set $d_x = d_y = d$) and number of total samples (from a joint Gaussian distribution) is in the appendix.

CCA for Deep Feature Learning

Background and motivation. A deep neural network (DNN) extension of CCA was proposed by Andrew et al. (2013) and has become popular in multi-view representation learning tasks. The idea is to learn a deep neural network as the mapping from original data space to a latent space where the canonical correlations are maximized.

Table 6.1: Wall-clock runtime of one pass through the data of our RSG+ and MSG on MNIST, Mediamill and CIFAR (average of 5 runs).

	MNIST			Mediamill			CIFAR		
Time (s)	k = 1	k = 2	k = 4	k = 1	k = 2	k = 4	k = 1	k = 2	k = 4
RSG+ (Ours)	4.16	4.24	4.71	1.89	1.60	1.44	14.80	17.22	22.10
MSG	35.32	42.09	49.17	11.59	14.21	17.34	80.21	100.80	106.55



(a) on MNIST (b) on Mediamill (c) on CIFAR

Figure 6.2: Performance on three datasets in terms of PCC as a function of # of seen samples.

We refer the reader to Andrew et al. (2013) for details of the task. Since deep neural networks are usually trained using SGD on mini-batches, this requires obtaining an estimate of the CCA objective at every iteration in a streaming fashion, thus our RSG+ can be a natural fit. We conduct experiments on a noisy version of MNIST dataset to evaluate RSG+.

Dataset. We follow Wang et al. (2015a) to construct a noisy version of MNIST: View 1 is a randomly sampled image which is first rescaled to $[0, 1]$ and then rotated by a random angle from $[-\frac{\pi}{4}, \frac{\pi}{4}]$. View 2 is randomly sampled from the same class as view 1. Then we add independent uniform noise from $[0, 1]$ to each pixel. Finally the image is truncated into $[0, 1]$ to form view 2.

Table 6.2: Results of feature learning on MNIST. N/A means fails to yield a result on our hardware.

Accuracy(%)	d = 100	d = 500	d = 1000
DeepCCA	80.57	N/A	N/A
Ours	79.79	84.09	86.39

Implementation details. We use a simple 2-layer MLP with ReLU nonlinearity, where the hidden dimension in the middle is 512 and the output feature dimension is $d \in \{100, 500, 1000\}$. After the network is trained on the CCA objective, we use

a linear Support Vector Machine (SVM) to measure classification accuracy on output latent features. Andrew et al. (2013) uses the closed form CCA objective on the current batch directly, which costs $O(d^3)$ memory and time for every iteration.

Performance. Table 6.2 shows that we get similar performance when $d = 100$ and can scale to large latent dimensions $d = 1000$ while the batch method Andrew et al. (2013) encounters numerical difficulty on our GPU resources and the Pytorch Paszke et al. (2019) platform in performing an eigen-decomposition of a $d \times d$ matrix when $d = 500$, and becomes difficult if d is larger than 1000.

CCA for Fairness applications

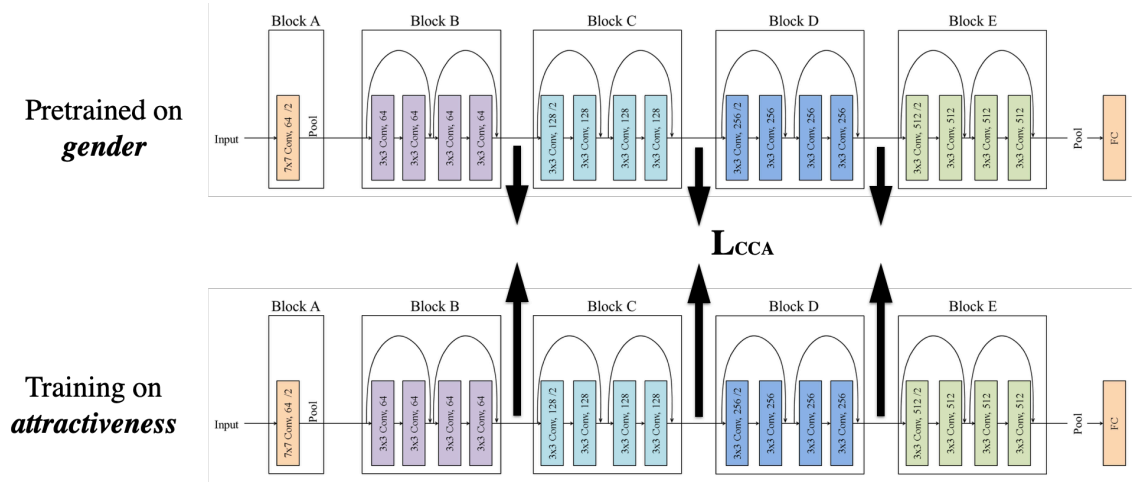


Figure 6.3: Training architecture for fairness experiment. The model above is the pretrained model and the model below is being trained. Use of CCA allows the two network architectures to be different.

Background and motivation. Fairness is becoming an important issue to consider in the design of learning algorithms. A common strategy to make an algorithm fair is to remove the influence of one/more protected attributes when training the models, see Lokhande et al. (2020). Most methods assume that the labels of protected attributes are known during training but this may not always be possible. CCA enables considering a slightly different setting, where we may not have per-sample protected attributes which may be sensitive or hard to obtain for third-parties Price and Cohen (2019). On the other hand, we assume that a model pre-trained to predict the protected attribute labels is provided. For example, if the protected attribute is gender, we only assume that a good classifier which is trained to predict gender from

the samples is available rather than sample-wise gender values themselves. We next demonstrate that fairness of the model, using standard measures, can be improved via constraints on correlation values from CCA.

Dataset. CelebA Wang et al. (2015b) consists of 200K celebrity face images from the internet. There are up to 40 labels, each of which is binary-valued. Here, we follow Lokhande et al. (2020) to focus on the *attactiveness* attribute (which we want to train a classifier to predict) and the *gender* is treated as "protected" since it may lead to an unfair classifier according to Lokhande et al. (2020).

Table 6.3: Fairness results on CelebA. We applied CCA on three different layers in Resnet-18 respectively. See appendix for positions of conv $0, 1, 2$. "Ours-conv[0,1]-conv[1,2]" means stacking features from different layers to form hypercolumn features Hariharan et al. (2015), which shows that our approach allows two networks to have different shape/size.

	Accuracy(%)	DEO(%)	DDP(%)
Unconstrained	76.3	22.3	4.8
Ours-conv0	76.5	17.4	**1.4**
Ours-conv1	**77.7**	**15.3**	3.2
Ours-conv2	75.9	22.0	2.8
Ours-conv[0,1]-conv[1,2]	76.0	22.1	3.9

Method. Our strategy is inspired by Morcos et al. (2018) which showed that canonical correlations can reveal the similarity in neural networks: when two networks (same architecture) are trained using different labels (schemes) for example, canonical correlations can indicate how similar their features are. Our observation is the following. Consider a classifier that is trained on gender (the protected attribute), and another classifier that is trained on *attractiveness*, if the features extracted by the latter model share a high similarity with the one trained to predict gender, then it is more likely that the latter model is influenced by features in the image pertinent to gender, which will lead to an unfairly biased trained model. We show that by imposing a loss on the canonical correlation between the network being trained (but we lack per-sample protected attribute information) and a well trained classifier pretrained on the protected attributes, we can obtain a more fair model. This may enable training fairer models in settings which would otherwise be difficult. The training architecture is shown in Fig. 6.3.

Implementation details. To simulate the case where we only have a pretrained network on protected attributes, we train a Resnet-18 He et al. (2016) on the *gender* attribute, and when we train the classifier to predict *attractiveness*, we add a loss using the canonical correlations between these two networks on intermediate layers: $\text{LOSS}_{\text{total}} = \text{LOSS}_{\text{cross-entropy}} + \text{LOSS}_{\text{CCA}}$ where the first term is the standard cross entropy term and the second term is the canonical correlation. See appendix for more details of training/evaluation.

Results. We choose two commonly used error metrics for fairness: difference in Equality of Opportunity Hardt et al. (2016a) (DEO), and difference in Demographic Parity Yao and Huang (2017) (DDP). We conduct experiments by applying the canonical correlation loss on three different layers in Resnet-18. In Table 6.3, we can see that applying canonical correlation loss generally improves the DEO and DDP metrics (lower is better) over the standard model (trained using cross entropy loss only). Specifically, applying the loss on early layers like conv0 and conv1 gets better performance than applying at a relatively late layer like conv2. Another promising aspect of our approach is that is can easily handle the case where the protected attribute is a continuous variable (as long as a well trained regression network on the protected attribute is given) while other methods like Lokhande et al. (2020); Zhang et al. (2018) need to first discretize the variable and then enforce constraints which can be much more involved.

Limitations. Our current implementation has difficulty to scale beyond $\mathrm{d} = 10^5$ data dimension and this may be desirable for large scale DNNs. Exploring sparsity may be one way to solve the problem and will be enabled by additional developments in modern toolboxes.

6.4 Related Work

Stochastic CCA: There has been much interest in designing scalable and provable algorithms for CCA: Ma et al. (2015) proposed the first stochastic algorithm for CCA, where local convergence is proven for the non-stochastic version. Wang et al. (2016) designed an algorithm which uses alternating SVRG combined with shift-and-invert pre-conditioning, with global convergence properties. These stochastic methods, and Ge et al. (2016) Allen-Zhu and Li (2016), which reduce the CCA problem to a generalized eigenvalue problem and solve it via an efficient power method, all belong to the class of methods that seeks to to solve the empirical CCA problem. It

can be seen as an ERM approximation of the original population objective, which requires solving numerically the empirical CCA objective on a fixed data set. These methods usually assume access to the full dataset at the outset, which may not be suitable for some practical applications where data is presented in a streaming manner. Recently, there appears to be an interest in considering the population CCA problem Arora et al. (2017) Gao et al. (2019). The main difficulty in the population setting is that we have limited knowledge about the objective unless we know the distribution of $\mathbf{X}$ and $\mathbf{Y}$. Arora et al. (2017) handles this problem by deriving an estimation of the gradient of the population objective whose error can be properly bounded so that applying proximal gradient to a convex relaxed objective will provably converge. Gao et al. (2019) provides a tightened analysis of the time complexity of the algorithm in Wang et al. (2016), and provides sample complexity for certain distributions. The problem we study is similar to the one in Arora et al. (2017); Gao et al. (2019): to optimize the population objective of CCA in a streaming fashion.

Riemannian Optimization: Riemannian optimization is a generalization of standard Euclidean optimization methods to smooth manifolds, which takes the following form: given $f : \mathcal{M} \to \mathbf{R}$, solve $\min_{x \in \mathcal{M}} f(x)$, where $\mathcal{M}$ is a Riemannian manifold. Advantages often include efficient numerical procedures for certain classes of constrained optimization problems. Applications include matrix and tensor factorization Ishteva et al. (2011), Tan et al. (2014), PCA Edelman et al. (1998), CCA Yger et al. (2012), and so on. We remark that Yger et al. (2012) also describes CCA formulation by rewriting it as a Riemannian optimization on the Stiefel manifold. In our work, we further explore the benefits of the Riemannian optimization toolkit, decomposing the linear space spanned by canonical vectors into products of several matrices which lie in several different Riemannian manifolds.

6.5 Summary

In this work, we presented a stochastic approach (RSG+) for the CCA model based on the observation that the solution of CCA can be decomposed into a product of matrices which lie on certain structured spaces. This affords specialized numerical schemes and makes the optimization more efficient. The optimization is based on Riemannian stochastic gradient descent and we provide a proof for its $O(\frac{1}{t})$ convergence rate with the number of iterates upper bounded, with $O(d^2k)$ time complexity per iteration. In experimental evaluations, we find that our RSG+ behaves favorably

relative to the baseline stochastic CCA method in capturing the correlation in the datasets. We also show the use of RSG+ in the DeepCCA setting showing feasibility when scaling to large dimensions as well as in an interesting use case in training fair models. In addition, in an ongoing collaboration with aging researchers, we are exploring the possibility of utilizing the DeepCCA setting to analyze the relationship between the human brain and the gut microbiome.

Chapter 7

Neural TMDlayer: Modeling Instantaneous flow of features via SDE Generators

In chapters 3–5, the dynamical systems we utilize do not involve stochastic terms. In chapter 6, the stochastic terms in the dynamics of streaming CCA are introduced due to choice of the SGD, not the network itself. In this chapter, we explicitly model the stochasticity taking place in the feature space of deep models utilizing ideas from the stochastic differential equations, and show that this stochasticity can help in various vision-related tasks. The work covered in this chapter was published as a conference paper at ICCV 2021 Meng et al. (2021c).

7.1 Introduction

Consider a deep neural network model with parameters W which we train using the following update rule,

$$W \leftarrow W - \eta \nabla_W \mathbb{E}_z \mathrm{R}\left(W, z\right) \tag{7.1}$$

where z is a random variable representing data and $\mathrm{R}(\cdot)$ represents the loss function. Now, consider a slightly general form of the same update formula,

$$W \leftarrow W - \eta \nabla_W \mathbb{E}_z \mathrm{R}\left(W, \mathrm{T}z\right). \tag{7.2}$$

The only change here is the introduction of T which can be assumed to be *some* data transformation matrix. Letting $T = I$, we see that Stochastic Gradient Descent (SGD) is a special case of (7.2) under the assumption that we will approximate the expectation in (7.2) with finite samples (or a mini-batch).

Let us unpack the data transformation notation a bit to check what it offers. If a set of transformations T are chosen beforehand, and applied to the data samples before training commences, Tz simply represents data samples derived via data augmentation (e.g., see the left part of Fig. 7.1). On the other hand, Tz may not necessarily be explicitly instantiated as above. For example, spherical CNN Esteves et al. (2017) shows that when point cloud type data are embedded in the sphere with spherical convolutional operators, then it is possible to learn representations of data that are equivariant to the transformation action of rotations with **no** explicit data augmentation procedure. In particular, these approaches *register* each data point on a standard *template* (like the sphere) on which efficient convolutions can be defined based on differential geometric constructions – in other words, utilizing the properties of the transformations T of interest *and* how they relate the data points, such a treatment enables the updates to implicitly take into account the loss on Tz. Conceptually, many results Esteves et al. (2017); Spezialetti et al. (2019); Qi et al. (2020) on equivariance show that by considering the *entire* orbit of each sample (a 3D point cloud) during training, for special types of T, it is possible to avoid explicit data augmentation.

We can take a more expanded view of the above idea. Repeated application of a transformation T on data point z produces a discrete sequence $\{z(t)\}_{t=0}^{\infty}$ where $z(0) = z, z(t) = T^{t-1}z$. In general, the transformation matrix at the t-th step, denoted by $T(t)$, need not even be generated from a fixed matrix. Indeed, in practice $T(t)$ is selected from a set of appropriate transformations such as rotation, blur and so on, with some ordering, which could even be stochastic. At a high level, approaches such as Esteves et al. (2017); Cohen et al. (2018) can be seen as a special case of (7.2). Making this argument precise needs adding an appropriate number of auxiliary variables and by averaging over all possible realizable T's – the specific steps are not particularly relevant since apart from helping set up the intuition we just described, algorithms for equivariance to specific group actions do not directly inform our development. For the sake of convenience, we will primarily focus on the continuous time system since under the same initial conditions, the trajectories of both (continuous and discrete) systems coincide at all integers t.

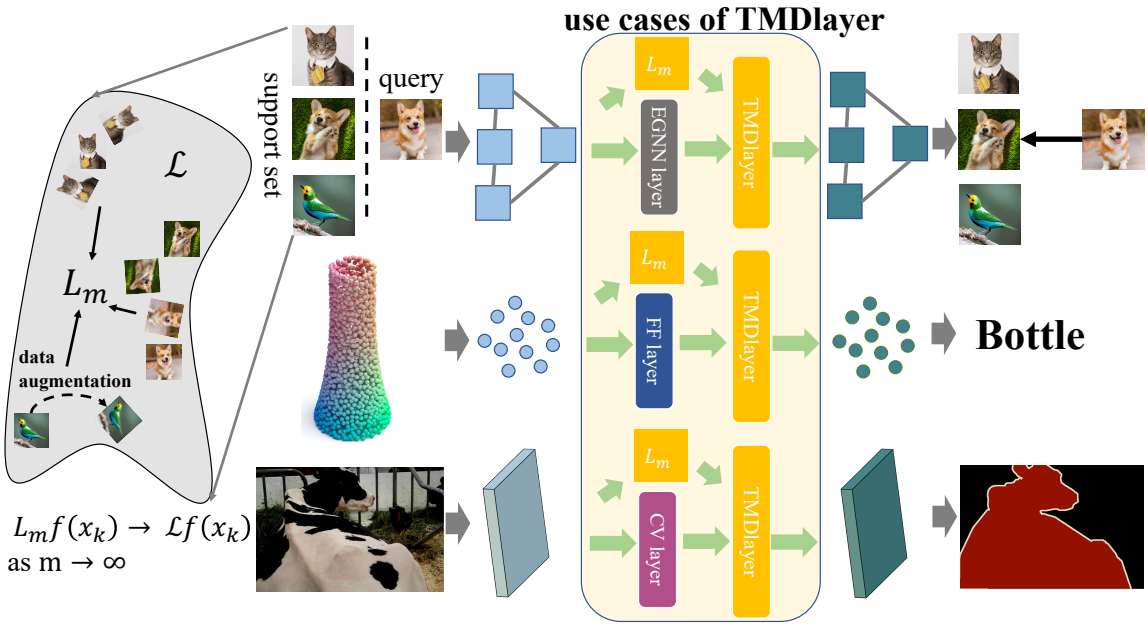


Figure 7.1: Approach overview of TMDlayer used in three applications (few-shot recognition, point cloud learning and segmentation respectively). "EGNN" refers to edge-labeling graph neural network Kim et al. (2019); "FF" refers to feed-forward layer and "CV" refers to our proposed deep chan-vese model. The manifold on the left illustrates the meaning of $\mathcal{L}$ and $\mathrm{L_m}$: $\mathcal{L}$ captures the structure of the manifold and $\mathrm{L_m}$ is an approximation of $\mathcal{L}$ constructed from samples.

What does $z(t)$ actually represent? There are two primary interpretations of $z(t)$: (i) it formalizes on-the-fly or instantaneous (smooth) data augmentation techniques which are often used to accelerate training by exploiting symmetries in the landscape of risk R, and (ii) a data dependent T can be designed for invariance type requirements which are useful for downstream applications (note: Cubuk et al. (2019) also proposes learning augmentation from data instead of hand designing it.). The starting point of this work is to exploit the view that the data sample provided to us is merely a *snapshot* of an underlying process which we will describe in more detail shortly. Nonetheless, the key hypothesis is that specifying this process to our deep neural network model will be beneficial and provide a fresh perspective on some strategies that are already in use in the literature.

Main ideas. The foregoing use of "process" to describe the data sample hints at the potential use of an ordinary differential equation (ODE). While ODE type constructions can be used to characterize simple processes, it will be insufficient to

model more complex processes that will better reflect practical considerations. The key challenge in directly instantiating the "$z(t)$" idea for SDEs is clearly infeasible since there are infinite possible trajectories for the same initial conditions. Our main insight is that recent results in the SDE literature show that (under some technical conditions), the dynamics $z(t)$ can be **completely** characterized by (functions of) the infinitesimal generator $\mathcal{L}$ of the process $z(t)$ which can be efficiently estimated using finite data. We exploit this result by proposing a simple modification to the estimation procedure that can be directly utilized within *any* backpropagation based training framework. Specifically, we exploit the result proposed by Banisch et al. (2020) where they call the generator Target Measure Diffusion map (TMDmap). This leads to our **TMDlayer** that can be conveniently dropped into a network, and be simply used as a plug-and-play module with just a few additional parameters. As a result, when utilized within standard vision pipelines, our layer provides a simple way to incorporate much richer domain information if available, or as a regularizer or augmentation scheme, or as a substitute to an existing layer, and we find is beneficial to the overall performance of the model.

Our contributions. There is a growing interest in the interface of physics-based formulations and deep learning models, both for algorithm design and analysis. Models such as a Neural ODE Chen et al. (2018) and Neural SDE Liu et al. (2019c) usually implement or parameterize the dynamical system as a stand-alone model, and show how gradients can be efficiently backpropagated through this module. In this chapter, we take a different angle: we propose a stochastic process inspired layer whose dynamics, in its most simplistic form, can be thought of as an augmentation technique that can work with any existing layer in deep neural networks. But different from standard explicit data augmentation (rotation, flipping) that happens in the input image space, our augmentation layer can be utilized in the feature space and is fully adaptive to the input. Further, it can be used in both training and test phases. Our proposed layer allows modeling the time-varying/stochastic property of the data/features, and controls them by a proper parameterization in a manner which is very parameter efficient. We will show that this stochasticity, while interesting at a mathematical level, can be easily exploited in various vision applications including point cloud transformers, object segmentation and few-shot recognition.

7.2 Related Work

Early work in vision has made extensive use of differential equations Chan and Vese (2001); Malladi et al. (1995); Rudin and Osher (1994); Caselles et al. (1997), especially for segmentation. In machine learning, differential equations are useful for manifold learning Belkin and Niyogi (2003) and semi-supervised learning Belkin et al. (2006); Melas-Kyriazi (2020) among others. Recently, a number of strategies combine differential equations with deep neural networks (DNNs) for solving vision problems. For example, Chen et al. (2017) utilizes a conditional random field after the CNN encoder to refine the semantic segmentation results whose update rules can be viewed as a differential equation and Marcos et al. (2018); Hatamizadeh et al. (2019) uses a CNN to extract visual features before feeding them to an active contour model which iteratively refines the contour according to the differential equation. Separately, the literature includes strategies for solving differential equations with DNNs Kharazmi et al. (2019); Michoski et al. (2020); Li et al. (2020). Over the last few years, a number of formulations including neural ODE Chen et al. (2018), neural SDE Liu et al. (2019c) and augmented neural ODE Dupont et al. (2019) have been proposed, motivated by the need to solve differential equation modules within DNNs. Note that Liu et al. (2019c) proposes to stabilize the neural ODE network with stochastic noise, which leads to a neural SDE, a setting quite different from the one studied here. Finally, we note that SDEs as a tool have also been used for stochastic analysis of DNNs Chaudhari and Soatto (2018).

7.3 Method

Background. Given a time invariant stochastic process $\underline{X}_t$, the (infinitesimal) generator $\mathcal{L}$ of a function f is defined as,

$$\mathcal{L}f(\underline{X}) := \lim_{t \to 0} \frac{\mathbb{E}\left[f(\underline{X}_t)\right] - f(\underline{X}_0)}{t}. \tag{7.3}$$

If the process $\underline{X}_t$ is deterministic, the expectation operator $\mathbb{E}$ becomes identity, and so the generator $\mathcal{L}$ simply measures the instantaneous rate of change in f with respect to $\underline{X}$. In addition, say that $\underline{X}_t$ can also be expressed as a Stochastic Differential Equation (SDE), i.e., $\underline{X}_t$ satisfies:

$$d\underline{X}_t = \underline{b}(\underline{X}_t)dt + \sigma(\underline{X}_t)d\underline{W}_t, \tag{7.4}$$

where $\underline{W}_t$ is a (multidimensional) Brownian motion with covariance C, and $\underline{b}$, σ represent drift and diffusion functions. We consider the case of general stochastic differential equations where b and σ depend not only on the present value of the process $\underline{X}_t$, but also on previous values of the process and possibly on present or previous values of other processes too. In this case the solution process, $\underline{X}$, is not a Markov process, and it is called an Itô process and not a diffusion process. Then, it turns out that $\mathcal{L}$ can be written in closed form (without limit) as,

$$\mathcal{L}f = \underline{b} \cdot \nabla f + \sigma C \sigma^T \cdot \nabla^2 f, \tag{7.5}$$

where $\mathcal{L}$ acts as an operator on functions f, see Kunita (1997) for more details. In this section, we will explain how to estimate and utilize $\mathcal{L}$ within popular deep learning frameworks.

Setup. Consider the setting in which $\underline{X}$ represents our input features, say image as a three dimensional array (corresponding to the three RGB channels) and f corresponds to a neural network with L layers. Let the data be given in the form of points $\mathcal{D}^{(m)} := x_1, x_2, ..., x_m \in \mathcal{R}^N$ with $N > 0$, which lie on a compact d-dimension differentiable submanifold $\mathcal{M} \in \mathcal{R}^N$ which is assumed to be unknown. Diffusion map Coifman and Lafon (2006) uncovers the geometric structure by utilizing the data $\mathcal{D}^{(m)}$ to construct an $m \times m$ matrix that approximates a differential operator.

Interpreting SDE. As discussed before, when $\mathcal{L}$ is used on the input space, it can model stochastic transformations to the input image which include commonly used hand designed rotation and clipping as special cases. When $\mathcal{L}$ is used on feature space (e.g., in an intermediate layer of a DNN), it can then model stochastic transformations of the features where it is hard to hand design augmentation methods. Besides, it allows us to parameterize and learn the underlying stochastic changes/SDE of the features.

Constructing L_m in DNN training. The definition in (7.3) while intuitive, is not immediately useful for computational purposes. Under some technical conditions such as smoothness of b, σ, f, and rank of C, Banisch et al. (2020) recently showed that for processes that satisfy (7.4), it is indeed possible to construct finite sample estimators L_m of $\mathcal{L}$. In Banisch et al. (2020) the approach is called Target Measure Diffusion (TMD) so we call our proposed layer, a TMDlayer.

To construct the differential operator, we first need to compute a kernel matrix $K \in R^{m \times m}$ from the data. For problems involving a graph or a set of points as input,

we can simply use the given data points (m would be the number of nodes in the graph, or the number of points in the set), while for problems with a single input (e.g., standard image classification problem), we may not have access to m data points directly. In this case, we can construct the kernel matrix by sampling a batch from the dataset and process them together because we can often assume that the whole dataset is sampled from some underlying distribution.

Algorithm 3: Our TMDlayer

Input: Function f, a batch of data samples $\{x_1, ..., x_m\}$, coefficient ϵ, parameterized time interval Δt
Construct distance matrix of batch data samples by $(K_\epsilon)_{ij} = \exp(-(4\epsilon)^{-1}\|x_i - x_j\|^2)$
Compute kernel density estimate: $q_\epsilon(x_i) = \sum_{j=1}^{m}(K_\epsilon)_{ij}$
Parameterize target distribution: $\pi^{1/2}(x_i) = \text{Linear}(x_i)$
Form the diagonal matrix $D_{\epsilon,\pi}$ with components $(D_{\epsilon,\pi})_{ii} = \pi^{1/2}(x_i)q_\epsilon^{-1}(x_i)$
Use $D_{\epsilon,\pi}$ to right-normalize K_ϵ: $K_{\epsilon,\pi} = K_\epsilon D_{\epsilon_\pi}$
Form the diagonal matrix $\tilde{D}_{\epsilon,\pi}$ with $(\tilde{D}_{\epsilon,\pi})_{ii} = \sum_{j=1}^{m}(K_{\epsilon,\pi})_{ij}$
Construct L_m by $L_m = \epsilon^{-1}(\tilde{D}_{\epsilon,\pi}^{-1}K_{\epsilon,\pi} - I)$
Return: $f(X) + \Delta t \cdot L_m f(X)$

After getting the set of data samples, we first project the data into a latent space $\mathbb{R}^h$ with suitable h using a learnable linear layer, before evaluating them with an appropriate kernel function such as,

$$k_\epsilon(x_1, x_2) = \exp(-(4\epsilon)^{-1}\|x_1 - x_2\|^2). \tag{7.6}$$

We then follow Banisch et al. (2020) to construct the differential operator $\mathcal{L}$ as follows: we compute the kernel density estimate $q_\epsilon(x_i) = \sum_{j=1}^{m}(K_\epsilon)_{ij}$. Then form the diagonal matrix $D_{\epsilon,\pi}$ with components $(D_{\epsilon,\pi})_{ii} = \pi^{(1/2)}(x_i)q_\epsilon^{-1}(x_i)$. Here, we allow the network to learn π by

$$\pi^{1/2}(x_i) = g(x_i) \tag{7.7}$$

where g can be a linear layer or a MLP depending on specific application. Next we use $D_{\epsilon,\pi}$ to right-normalize the kernel matrix $K_{\epsilon,\pi} = K_\epsilon D_{\epsilon,\pi}$ and use $\tilde{D}_{\epsilon,\pi}$ which is the diagonal matrix of row sums of $K_{\epsilon,\pi}$ to left-normalize $K_{\epsilon,\pi}$.

Then we can build the TMDmap operator

$$L_m = \epsilon^{-1}(\tilde{D}_{\epsilon,\pi}^{-1}K_{\epsilon,\pi} - I). \tag{7.8}$$

We will use (7.8) to form our TMDlayer as follows.

TMDlayer: A Transductive Correction via L_m

Observe that (7.4) is very general and can represent many computer vision problems where the density π could be defined using an problem specific energy function, and W_t is the source of noise. In other words, we aim to capture the underlying structure of the so-called image manifold Zhu et al. (2016) by using its corresponding differential operator (7.5). Intuitively, this means that if we are given with a network f_W with parameters W, then by Taylor's theorem, the infinitesimal generator estimate L_m can be used to approximate the change of f_W as follows:

$$\mathbb{E}_x f_W(x, \Delta t) \approx f_W(x, 0) + \Delta t \cdot L_m[f_W], \tag{7.9}$$

where $[f_W] \in \mathbb{R}^m$ such that the $i-$th coordinate $[f_W]_i = f_W(x_i)$, and Δt is interpreted as a hyperparameter in our use cases. Please see Algorithm 3 for more details.

Inference using L_m. In the ERM framework, typically, each test sample is used independently, and identically i.e., network (at optimal parameters) is used in a sequential manner for predictive purposes. Our framework allows us to further use relationships between the test samples for prediction. In particular, we can design custom choices of b, σ tailored for downstream applications. For example, in applications that require robustness to *small* and *structured* perturbations, it may be natural to consider low bias diffusion processes i.e., we can prescribe the magnitude using $\|b\|_p \leqslant \kappa$ almost everywhere for some small constant $\kappa > 0$ (akin to radius of perturbation) and structure using diffusion functions σ, C. Inference can then be performed using generators $\mathcal{L}$ derived using the corresponding process.

Layerwise $\mathcal{L}$ for improved estimation of L_m. While (7.9) allows us to use L_m for *any* network with no modifications, using it naively can be unsatisfactory in practice. For example, often we find that features from input layers might not be too informative for the task and may hinder training, especially in the early stages. We suggest a simple adjustment: instead of applying approximation in (7.9) on the entire network, we do it layerwise – could be every intermediate layer or several interested layers. It means that f can in principle be any layer (e.g., a layer in graph neural networks or a layer in Resnet), as shown in Fig. 7.1.

Justification. Recall that most feed-forward neural networks can be completely defined by a finite sequence of linear transformations followed by activation functions (along with intermediate normalization layers). One option is to estimate L_m by directly applying the Taylor series-like expansion in (7.9) on $f = f^l \circ f^{l-1} \circ \cdots f^1$ where l represents the number of layers. However, from (7.9) we can see that the variance of such an estimate of the value $L_m[f_W]$ will be high due to the well-known propagation of uncertainty phenomenon (across f_i's). To avoid this, we can estimate $L_m[f_W]$ in a sequential manner i.e., use $L_m[f_W^{i-1}]$ to estimate $L_m[f_W^i]$ $\forall\, i \in [l]$. We will show in section 7.4 that this parameterization can be useful in various applications.

Synopsis. We briefly list the benefits of our TMDlayer next which we will further demonstrate by experiments in section 7.4:

1. Our TMDlayer can parameterize and model the underlying stochastic transformations of features, providing a simple way to augment features at any layer.

2. The stochasticity/randomness in our TMDlayer is a stability inducing operation designed for robust predictive purposes Hardt et al. (2016b).

3. Our TMDlayer is parameter efficient. All we need is a projection linear layer and a linear layer parameterizing the density π and a scalar parameter Δt. In practice, we can work with small latent dimension (e.g., $h = 16$) when constructing L_m, thus the total number of parameters in TMDlayer is very small compared with the layer function f in most deep learning applications.

But the reader will recognize that a mild limitation of the SDE perspective in practice is that in principle it is possible that the dynamics eventually get stuck in a metastable state. This means that in this case, the estimate L_m will not be very informative in the forward pass, and so the gradient estimates might be biased. In such cases, it may be useful to add points by sampling on the orbit if needed. We will now describe four different vision settings in which our TMDlayer can be simply instantiated in a plug-and-play manner.

7.4 Applications

In this section, we show the use of our TMDlayer in different applications. As a start, we demonstrate the use of TMDlayer on a simple image classification task and

study its properties in both inductive and transductive settings. We then move to point cloud learning where the point cloud naturally forms a suitable object for the TMDlayer's use, and conduct experiments in an inductive setting. Next we explore the use of TMDlayer on segmentation problem, also in inductive setting. We propose a novel deep active contour model which can be viewed as a dynamical process within a neural network, and demonstrate the use of TMDlayer on top of this dynamic process. Finally we investigate few-shot learning which natively provides the graph needed for computing our L_m and allows transductive inference.

A Simple Sanity check on Resnet

We start with a simple example of image classification on CIFAR10 Krizhevsky et al. (2009) using Resnet He et al. (2016), to demonstrate the wide applicability of our TMDlayer and study its properties.

Role of TMDlayer: Use in Resnet

We choose Resnet-18 as the backbone network and treat each of its three residual blocks as f (see He et al. (2016) for details of a residual block),

$$f(x^{l-1}) = \text{Residual-block}(x^{l-1}) \tag{7.10}$$

and then use our TMDlayer on it,

$$x^l = f(x^{l-1}) + \Delta t \cdot L_m f(x^{l-1}) \tag{7.11}$$

where x^l is the feature at l-th layer and L_m is constructed from a mini-batch of samples.

Experimental results

During training, we first sample m data points in a batch as input so that we can construct L_m. During inference, we also group m test samples together as input, where m increases from 1 to 200. We can see from Table 7.1 that m does have an influence on the test accuracy with larger m performing better than small m, since L_m with large m may better capture the geometric structure of the data.

We also test whether our TMDlayer can lead to improvement of the robustness of the network, by adding random noise to input image and evaluating the test accuracy. Results are shown in Table 7.2. We can see that with TMDlayer, the network gains better resilience to the noise injected in the input. This can be parly attributed to the use of our parameterized Δt, which allows the network to control the stochastic process in TMDlayer adaptively to the input.

m	Inference w/ TMDlayer	Accuracy (%)
1	No	75.15
1	Yes	87.35
10	Yes	87.65
50	Yes	88.14
100	Yes	88.52
150	Yes	88.55
200	Yes	88.25

Table 7.1: Accuracy on test set of CIFAR10 by adding TMDlayer to Resnet-18. m is the batch size used to construct L_m during test/inference time. The accuracy of Resnet-18 alone is 88.27%.

σ	0.01	0.02	0.03	0.05	0.1
Resnet-18	87.54	83.90	75.85	53.87	17.27
Ours	87.79	84.37	77.96	56.18	19.18

Table 7.2: Accuracy on CIFAR10 when adding random noise (mean = 0, std = σ) to input. "Ours" means Resnet-18 plus TMDlayer.

Point cloud transformer

Point cloud learning is a important topic in 3D vision. The input is usually a point cloud represented by a set of points, each associated with its feature description. These points can be naturally thought of as samples from the underlying distribution which captures the structure of the object. It provides an ideal sandbox to study the effect of our TMDlayer. Very recently, Guo et al. (2020) proposed a transformer based model for point cloud learning which achieves state-of-the-art performance on point cloud learning, motivated partly by the broad use and popularity of transformers in machine learning and computer vision. Nonetheless, Transformer models are known to be very parameter costly and it is sensible to check to what extent our

TMDlayer operating on a simple linear layer can be competitive with the transformer layer proposed in Guo et al. (2020), while offering significant parameter efficiency.

Problem formulation

Given an input point cloud $\mathcal{P} \in R^{N \times d}$ with N points each with d-dimensional feature description, the classification task is to predict a class label for the whole point cloud.

Role of TMDlayer: to replace transformer layer

The point cloud transformer layer proposed by Guo et al. (2020) is designed as,

$$F_{out} = FF(F_{in} - F_{sa}) + F_{in} \tag{7.12}$$

where FF refers to their feed-forward layer (a combination of Linear, BatchNorm and ReLU layer), and F_{sa} is the output of self-attention module which takes F_{in} as input (we refer readers to Guo et al. (2020) for more details of their network design, also included in appendix).

The reason transformer layer is suitable for point cloud is that it can simultaneously captures the correlation among features of all points. Since our TMDlayer can be viewed as a diffusion operator which captures the structure of underlying manifold from the data, we test its ability by using TMDlayer on a single feed-forward layer to replace the transformer layer in (7.12).

$$F_{out} = FF(F_{in}) + \Delta t \cdot L_m FF(F_{in}) \tag{7.13}$$

Surprisingly, it turns out that this simple layer can perform comparably with the carefully designed transformer layer in (7.12) while being much more parameter efficient (L_m is constructed using points of the same point cloud).

Experimental results

Dataset. We follow Guo et al. (2020) to conduct point cloud classification experiment on ModelNet40 Wu et al. (2015). It contains 12311 CAD models in 40 object categories and has been widely used in point cloud shape classification benchmarking. We use the official splits for training and evaluation.

Network architecture and training details. We use the same network as Guo et al. (2020) except that we replace each point cloud transformer layer with a TMD-layer built on a single feed forward layer. We follow Guo et al. (2020) to use the same sampling strategy to uniformly sample each object to 1024 points and the same data augmentation strategy during training. The mini-batch size is 32 and we train 250 epochs using the SGD optimizer with momentum 0.9, initial learning rate 0.01 and a cosine annealing schedule. The hidden dimension is 256 for the whole network and 16 for the construction of L_m in our TMDlayer.

Results. We see from Table 7.3 that our approach achieves comparable performance with Guo et al. (2020). In terms of number of parameters, using hidden dimension 256 (used in this experiment) as an example, one self-attention layer contains 148k parameters; one linear layer contains 65.5k parameters; and the TMDlayer module only needs 4k parameters.

Method	Input	#Points	Accuracy(%)
PointNet Qi et al. (2017)	P	1k	89.2
A-SCN Xie et al. (2018)	P	1k	89.8
SO-Net Li et al. (2018)	P, N	2k	90.9
Kd-Net Klokov and Lempitsky (2017)	P	32k	91.8
PointNet++ Qi et al. (2017)	P	1k	90.7
PointNet++ Qi et al. (2017)	P, N	5k	91.9
PointGrid Le and Duan (2018)	P	1k	92.0
PCNN Atzmon et al. (2018)	P	1k	92.3
PointConv Wu et al. (2019)	P, N	1k	92.5
A-CNN Komarichev et al. (2019)	P, N	1k	92.6
DGCNN Wang et al. (2019)	P	1k	92.9
PCT Guo et al. (2020)	P	1k	**93.2**
Ours	P	1k	93.0

Table 7.3: Results of classification task on ModelNet40. Accuracy means overall accuracy. P = points, N = normals. Ours means replacing transformer layers in PCT with our TMDlayer.

Object segmentation

In this experiment we show that our TMDlayer, directly based on dynamical systems principles, can also be built on top of another dynamical system. We do so by demonstrating experiments on object segmentation.

Recall that active contour models are a family of effective segmentation models which evolve the contour iteratively until a final result is obtained. Among many models that exist in the literature, the Chan-vese Chan and Vese (2001) model is a widely used and efficient model which evolves the contour based on a variational functional. Here we propose to combine Chan-vese functional with a deep neural network by parameterizing the iterative evolving steps and build our TMDlayer on top of it, and show that it leads to better segmentation results. (See appendix for details of our proposed deep active contour model.)

Problem formulation

Let Ω be a bounded open subset of R^2, with $\partial\Omega$ its boundary. Let $I : \bar{\Omega} \to R$ be a given image, the task of object segmentation is to predict a dense map in $\bar{\Omega} \to 0/1$ where 1 indicates the object and 0 indicate the background. In our method, we parameterized the contour of object by a level set function $\phi : \Omega \to \mathbb{R}$ and evolve it within the DNN. We note that hybrid approaches using level sets segmentation schemes together with DNNs is not unique to our work, and has been independently investigated via specialized architectures in Marcos et al. (2018); Yuan et al. (2020).

Role of TMDlayer: in deep active contour model

Our proposed deep active contour model evolves the contour in the form of level set function within the network, and the updating function is,

$$\phi^l = \phi^{l-1} + \frac{\partial \phi}{\partial t}\Delta t' \tag{7.14}$$

where ϕ^{l-1} is the level set function at layer $l-1$ and $\frac{\partial \phi}{\partial t}$ is derived from our proposed deep variational functional. (See appendix for details of our proposed model, the variational functional, and the derivation of updating equation).

Denote the updating function in (7.14) as $\phi^l = f(\phi^{l-1})$. Together with our TMD-layer, the complete updating function of the whole layer is:

$$\phi^l = f(\phi^{l-1}) + \Delta t \cdot L_m f(\phi^{l-1}) \tag{7.15}$$

Remark 7.1. *Note that $\Delta t'$ in (7.14) and the Δt in (7.15) are for two different dynamical systems. The first one is for the updating function of deep active contour model and the second*

one is for the TMDlayer. L_m *in (7.15) is constructed using samples from the same mini-batch.*

Remark 7.2. *Note that our proposed segmentation model is different from Yuan et al. (2020) which uses the variational energy function directly as the final loss function, whereas we are parameterizing the updating steps within our network so that the final output will already satisfy low variational energy.*

Experimental results

Dataset. The Vaihingen buildings dataset consists of 168 building images extracted from the training set of ISPRS "2D semantic labeling contest" with a resolution of 9cm. We use only 100 images to train the model and the rest 68 serve as the test set.

Network Architecture and Experiment Setup. We use an encoder CNN with an architecture similar to Hariharan et al. (2015) and Marcos et al. (2018). The input is the original image. The network is trained with a learning rate of 10^{-4} for 300 epochs using a batch size of 10. We setup our baseline using the same CNN architecture to predict the segmentation mask without our Chan-Vese updating module. Previous works combining active contour model and deep learning Marcos et al. (2018); Ling et al. (2019) can only be used to provide segmentations of a single building based on manual initialization or another algorithm based initialization whereas our model can be used to segment multiple buildings in the image *without* any initialization. For this reason, the results cannot be meaningfully compared. See our appendix for more details about the setup.

Results and Discussion. We use the average Intersection over Union (IoU) to evaluate the performance on Vaihingen dataset: the baseline yields **68.9** while our model without TMDlayer achieves **73.5** and our complete model with TMDlayer achieves **74.6**, which is a significant improvement in terms of IoU. This shows that our TMDlayer can be built on top of another dynamical system and provide extra benefits.

Qualitative results of the baseline model and our model are shown in Fig. 7.2. We can see that our method tends to predict more precise shape and boundary, and also fixes some flaws compared with the baseline results.

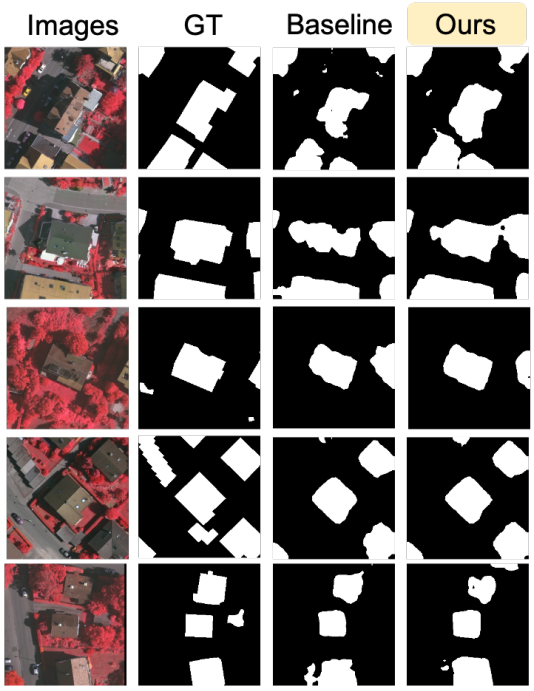


Figure 7.2: Qualitative results on Vaihingen dataset. Our model performs well despite the small sample size.

Few-shot learning

In N-way B-shot few-shot learning, the input is $N * B$ samples which naturally forms a fully connected graph and can be used to construct the differential operator L_m. Kim et al. (2019) proposed a graph neural network based approach named EGNN for few-shot learning and this model achieves state-of-the-art performance. We show that by adding TMDlayer, the performance of the model increases by a clear margin.

Problem formulation

The goal of few-shot learning classification is to learn a classifier given only a few training samples for every class. Each few-shot classification task $\mathcal{T}$ contains a support set S which is a set of labeled input-label pairs, and a query set Q which is an unlabeled set where the learned classifier is evaluated. If there are B labeled samples for each of N classes in the support set S, the problem is called N-way B-shot classification problem.

Role of TMDlayer: Use in graph neural network

Let $\mathcal{G}$ be the graph formed by samples from the task $\mathcal{T}$, with nodes denoted as $\mathcal{V} := \{\underline{v}_i\}_{i=1,\dots,|\mathcal{T}|}$. The node feature updating equation is designed as (we refer readers to Kim et al. (2019) or our appendix for more details about the network)

$$\underline{v}_i^l = \text{NodeUpdate}(\{\underline{v}_i^{l-1}\}, \{e_{ij}^{l-1}\}; \theta_v^l) \tag{7.16}$$

where $\underline{v}_i^l$ is the feature of node i at l-th layer, e_{ij} is the edge feature between node i and node j, and θ refers to the parameters in the updating function. We abstract (7.16) as $\underline{v}_i^l = f(\underline{v}_i^{l-1})$ and use our TMDlayer

$$\underline{v}_i^l = f(\underline{v}_i^{l-1}) + \Delta t \cdot L_m f(\underline{v}_i^{l-1}) \tag{7.17}$$

Remark 7.3. *In (7.17) the L_m is constructed using samples from the same episode, and f is a GNN module updating the node features using all node features and edge features.*

Experimental results

Dataset. We follow Kim et al. (2019) to conduct experiments on *mini*ImageNet, proposed by Vinyals et al. (2016) and derived from ILSVRC-12 dataset Russakovsky et al. (2015). The images are sampled from 100 different classes with 600 samples per class, all of size 84×84 pixels. We use the same splits as in Ravi and Larochelle (2017); Kim et al. (2019): 64, 16 and 20 classes for training, validation and testing respectively.

Network architecture and training details. We use the same graph neural network architecture and follow training strategy as Kim et al. (2019) by utilizing the code provided by the authors. We add our TMDlayer as shown in (7.17) to each node updating layer in the graph neural network, with a latent dimension 16 for constructing the L_m. We follow Kim et al. (2019) to conduct experiments for 5-way 5-shot learning, in both transductive and non-transductive settings, as well as for both supervised and semi-supervised settings. The network is trained with Adam optimizer with an initial learning rate 5×10^{-4} and weight decay of 10^{-6}. The learning rate is cut in half every $15,000$ episodes. For evaluation, each test episode was formed by randomly sampling 15 queries for each of 5 classes, and the performance is averaged over 600 randomly generated episodes from the test set. Note that the feature embedding module is a convolutional neural network which consists of four

blocks (following Kim et al. (2019)) and used in most few-shot learning models without any skip connections. Thus, Resnet-based models are excluded from the table for fair comparison. We refer reader to Kim et al. (2019) or the appendix for more training and evaluation details.

Results. The performance of supervised and semi-supervised 5-way 5-shot learning are reported in Table 7.4 and 7.5 respectively. We can see that our TMDlayer leads to consistent improvements with a clear margin in both supervised and semi-supervised settings, and also for both transductive and non-transductive settings.

Model	Trans.	Accuracy(%)
Matching Networks Vinyals et al. (2016)	No	55.30
Reptile Nichol et al. (2018)	No	62.74
Prototypical Net Snell et al. (2017)	No	65.77
GNN Garcia and Bruna (2017)	No	66.41
EGNN Kim et al. (2019)	No	66.85
Ours	No	**68.35**
MAML Finn et al. (2017)	BN	63.11
Reptile + BN Nichol et al. (2018)	BN	65.99
Relation Net Sung et al. (2018)	BN	67.07
MAML + Transduction Finn et al. (2017)	Yes	66.19
TNP Liu et al. (2019d)	Yes	69.43
TPN (Higher K) Liu et al. (2019d)	Yes	69.86
EGNN+Transduction Kim et al. (2019)	Yes	76.37
Ours+Transduction	Yes	**77.78**

Table 7.4: Results of 5-way 5-shot learning on *mini*ImageNet. Ours means EGNN plus our TMDlayer. BN means that the query batch statistics are used instead of global batch normalization parameters. All results are averaged over 600 test episodes.

	Labeled Ratio (5-way 5-shot)			
Training method	20%	40%	60%	100%
GNN-semi Garcia and Bruna (2017)	52.45	58.76	-	66.41
EGNN-semi Kim et al. (2019)	61.88	62.52	63.53	66.85
Ours	**63.14**	**64.32**	**64.83**	68.35
EGNN-semi(T) Kim et al. (2019)	63.62	64.32	66.37	76.37
Ours(T)	**64.84**	**66.43**	**68.62**	77.78

Table 7.5: Accuracy of semi-supervised few-shot classification. "Ours" means EGNN plus our TMDlayer.

7.5 Summary

We proposed a SDE based data augmentation framework that can be used for data and feature augmentation purposes in any layer. Our framework is beneficial where data generation can be described using stochastic processes, or more specifically diffusion operators. This is particularly useful in settings in which obtaining a deterministic model of image manifold or when learning density functions are impossible due to high sample complexity requirements. TMD layer does not require explicit knowledge of samples, especially during training making it computationally efficient. The "process" of which a data sample is a snapshot, whose characterization is enabled by our TMDlayer, also appears to have implications for robust learning. Indeed, if the parameters that define the process are explicitly optimized, we should be able to establish an analogy between the resultant model as a stochastic/simpler version of recent results for for certified radius maximization which often rely on Monte Carlo sampling. We believe that periodicity in SDEs for data augmentation is an important missing ingredient – for instance – this may help model seasonal patterns in disease progression studies for predictions, automatically. For this purpose, tools from Floquet theory will allow us to consider transformed versions of the process, potentially with simplified generators.

Chapter 8

Conclusions

In this thesis, we demonstrated how dynamical systems can motivate novel layers for deep neural networks which expand the ability of deep models. We proposed novel layers based on several different dynamical systems and applied them on various applications in computer vision and machine learning. Specifically,

1. In Chapter 3, we leveraged message passing algorithm as layers to model relative attributes efficiently.

2. In Chapter 4, we proposed a differentiable layer for solving linear programs based on physarum dynamics, which is time efficient and GPU friendly.

3. In Chapter 5, we utilized a Newton's method to construct a differentiable LP layer to optimize non-decomposable objectives where the number of constraints is larger than the number of variables.

4. In Chapter 6, we studied the problem of streaming canonical correlation analysis where the dynamics happen across the whole training stage when integrated into deep models, and proposed an algorithm with less complexity per iteration and fast convergence rate compared with previous approaches.

5. In Chapter 7, we explicitly modeled the stochasticity taking place in the feature space of deep models by proposing a layer based on ideas from the stochastic differential equations, and show that this stochasticity can help in various vision-related tasks.

8.1 Future Work

We now discuss some ongoing and future research directions.

Utilize relative attribute learning beyond vision

Due to the flexibility that the framework of relative attribute learning provides, we can use it to solve real world problems (possibly outside of computer vision) by casting them as relative attribute learning. As an example, in a follow-up work Kanchinadam et al. (2021) to our work in Chapter 3, we utilize our method proposed in Chapter 3 to solve a regression task on language data – to predict a customer's satisfaction score from his/her phone call. The system takes as an input speech-to-text transcriptions of calls and predicts call satisfaction reported by customers on post-call surveys (scale from 1 to 10). Because of its ordinal, subjective, and often highly-skewed nature, predicting survey scores is not a trivial task and presents several modeling challenges. We introduce a graph neural network (GNN) approach that takes into account the comparative nature of the problem by considering the relative scores among batches, instead of only pairs of calls when training. This approach produces more accurate predictions than previous approaches including standard regression and classification models that directly fit the survey scores with call data. Compared with directly performing regression, our model is more tolerant to the noise in the labels since it considers the relative strength among a group of data samples. Compared with the pairwise ranking model, the benefit of our method is that the GNN predicted rankings for a batch of examples is guaranteed to be coherent with respect to the transitive nature of a ranking. In addition, the GNN may utilize higher order features among the data samples. It can be a promising direction to check whether our proposed relative attribute learning can work on more tasks beyond computer vision like language data demonstrated in the follow-up work, or other data modalities like audio, gaze, etc.

More efficient differentiable combinatorial optimization layer

Our works in this thesis focus primarily on differentiable continuous optimization although the discrete optimization setting is also important since its frequently used in real world applications, for example, when some part of the hidden state becomes discrete Rolfe (2016), or when the problems involves certain selection over candi-

dates Abbas and Swoboda (2021). Current works on differentiable combinatorial optimization are mostly based on perturbations and treat the specific solver as a blackbox Berthet et al. (2020); Vlastelica et al. (2019). This approach has the flexibility of utilizing existing combinatorial solvers, but is not efficient enough in a number of scenarios since the computation increases linearly with the number of perturbations and the number of perturbations needed for a good solution is often large. Since this process treats the specific combinatorial solver as a blackbox, we cannot rely on the solver to speedup this process of adding perturbations. It would be useful to explore whether some specialized solver based on some type of dynamics can act as better candidates. For example, if we can define gradients for each step of the solver then we can use unrolling to differentiate through it. Or if we can define the gradients based on certain information stored in the forward pass of the solver, we can also possibly avoid using perturbations.

A unified framework for differentiable optimization

Although there have been quite a few works on differentiable optimization, their implementations and practical considerations are quite different. For example, the solver proposed in Zeng et al. (2019) is specialized for matching problems, while the solver proposed in Cuturi et al. (2019) is for sorting problems. Optnet Amos and Kolter (2017) is designed for continuous optimization and implemented in Pytorch, while Berthet et al. (2020) is for discrete optimization and implemented in Tensorflow. Our solver in Chapter 4 works for LP problems with a comparable number of constraints and variables, while our solver in Chapter 5 work for LPs when the number of constraints is much larger than the number of variables. When a practitioner has a specific use case in hand, it might be very hard to find a good differentiable solver ready to use in a plug-and-play manner. It would be beneficial if we could develop a unified framework for differentiable optimization, which allows the user to input the order (first/second/more), the type (continuous/discrete), the speed preference (fast/precise), and maybe the GPU memory, etc. In this way, the differentiable optimization really becomes accessible to researchers who have very limited or no experience with low-level optimization concepts. To achieve this, one of the challenges is that different solvers may perform differently in terms of speed/accuracy tradeoff, thus we need to have a comprehensive evaluation to determine which solver to use given a user's preference. Most existing (non-differentiable) solvers are

implemented on CPU and implementing them on GPU can encounter challenges (for methods which start from existing non-differentiable solvers). Of course, there are also engineering challenges to implement solvers for so many different use cases under a unified framework.

More applications of our DeepCCA algorithm

In Chapter 6, we conduct experiments on several simple image datasets like MNIST and CIFAR, although our method can be applied to many other scientifically meaningful datasets. For example, many tasks associated with biomedical and scientific datasets need to analyze the relationship between two groups of variables or multiple modalities. While the simple variants of this task can be handled via regression, a large scale regression with hundreds (or more) of response variables takes a form quite similar to CCA. DeepCCA is a promising tool for these applications: it can utilize the power of deep neural networks to learn from the data, and can find correlated subgroups from the two set of variables using a single model. In addition, with our algorithm proposed in Chapter 6, DeepCCA can scale to accept large inputs. In an ongoing collaboration with aging researchers, we are exploring the possibility of utilizing the DeepCCA to analyze the relationship between the human brain and the gut microbiome. Specifically, researchers first make quantitative measurements of the human brain and the gut microbiome respectively, which results in two set of variables with N samples per variable (each sample here could come from a different individual). We then utilize our DeepCCA algorithm on this set of measurements. As a preliminary result, we find relatively strong correlation between certain anatomical regions in human brain and certain microbiome groups, which appears to be meaningful. In the future, we can also explore utilizing partial CCA to control for a set of variables whose influence is known. In a recent work Zhen et al. (2022) we propose a reasonable alternative to partial CCA and have verified its effectiveness on several experiments including training independent deep models and doing informative comparisons between deep networks. It will be interesting to compare these approaches with an extension of our deepCCA where one or more variables could be controlled for.

A new family of graph convolutional networks based on our proposed TMDlayer

The graph convolutional network (GCN) has been widely used in various problems involving graph structured data. Since GCN was first proposed by Kipf and Welling (2016), there have been a family of follow-up works proposed. Examples include graph attention networks Veličković et al. (2017) which adds the attention mechanism for calculating weights over the neighborhood nodes, graph partition neural networks Liao et al. (2018) which partition the nodes to decrease the amount of resources needed for computing, etc. Overall, the graph convolutional network models a collection of data points with certain graph structures utilizing the normalized adjacency matrix. Similarly, our TMDlayer (proposed in Chapter 7) also accepts a batch of data as input, and we also consider the relationship between data points when we construct this L_m. In other words, when the input X is a batch of data with some graph structure, the TMDlayer may possibly act in a similar way as GCN while the mechanism is different. Our TMDlayer can be viewed as a graph neural network with a residual link (since our layer is defined based on the first order Talor expansion) where infinitesimal SDE generator L_m replaces the normalized adjacency matrix. This perspective may provide a new family of graph neural networks which considers the stochasticity of the input. We carried out some preliminary experiments on some standard GNN benchmarks (CORA citation dataset) where the TMDlayer outperforms the GCN layer. It remains to be seen whether this improvement can generalize to other datasets and tasks.

Appendix A

Physarum Powered Differentiable Linear Programming Layers and Applications: Appendix

Proof of Theorem 4.8

Proof. It is sufficient to show that $\gamma_u = \Theta(\sqrt{m+n})$. But showing such a constant exists is equivalent to showing that there is a neighborhood $\mathcal{N} = \mathcal{B}(c, r)$ around the cost vector or objective function c of radius $r > 0$ such that the optimal values of any two cost $c_1, c_2 \in \mathcal{N}$ coincide i.e., there exists $x^* \in P$ such that $c_1^T x^* = c_2^T x^*$. To see that this is sufficient for our purposes, note that we can add small but positive constant to all the coordinates in c that correspond to auxiliary/slack variables. Now, it is easy to see that Assumptions 1 and 2 guarantee that the optimal solution set is a *bounded* polyhedral multifunction. Hence, we can use the Sticky Face lemma Robinson (2018) to guarantee that such a nonzero r exists. To conclude, we observe from the proof of the Sticky Face lemma, that r can be upper bounded by $1/M$, where M corresponds to the the diameter of P which is $\Theta(\sqrt{m})$. □

Proof of Convergence of ℓ_1-SVM

Since the SVM formulation is always feasible, by the separating hyperplane theorem, there exists a $\kappa > 0$ such that the when we add cost of κ to each coordinate of $\alpha_1, \alpha_2, b_1, b_2, p, q, r$, then the (cost) perturbed linear program and the original LP ((6) in the main paper), have the same optimal solution. Then, it is easy to see that C_s of

this perturbed problem is quadratic in n, C and κ. By scaling the data points, we can assume that

$$\|x_i\|_2 \leqslant 1. \tag{A.1}$$

We now bound the magnitude of sub-determinant D of the perturbed SVM LP. First note that the slack variables are diagonal, hence, the contribution to the determinant will be at most 1. Hence, to bound D, we need to bound the determinant of the kernel matrix $K(X, X)$. Using Fischer's inequality Thompson (1961), we have that,

$$D \leqslant \left(K\left(x_i, x_i\right)\right)^n. \tag{A.2}$$

For a linear kernel, we have that, $D = \|x_i\|^n \leqslant 1$ (by assumption (A.1)). For a Gaussian kernel scale σ, we have that, $D = O(\sigma)$ with high probability. We can easily extend this to any bounded kernel K.

Appendix B

Differentiable Optimization of Generalized Nondecomposable Functions using Linear Programs: Appendix

LP formulation for multi-class AUC

One way to extend binary AUC to multi-class is by considering multiple one-versus-all pairs. This leads us to the following formulation:

$$\begin{aligned} \text{AUC}: \min_{z_{ij}} & \sum_{i=1}^{n} \sum_{j=1:x_i^* \neq x_j^*}^{n} z_{ij} \\ \text{s.t.} \quad & (\mathbf{f}(x_i, x_i^*) - \mathbf{f}(x_j, x_i^*)) \geqslant \epsilon - z_{ij} \forall i, j : x_i^* \neq x_j^*, \\ & z_{ij} \geqslant 0 \end{aligned} \tag{B.1}$$

In our multi-class AUC experiment, we use this one-versus-all AUC as training loss and report performance in both one-versus-all AUC and $\text{AUC}_\mu^{\text{bin}}$. In addition, we can also consider the setting of AUC_μ where $\mathbf{P}$ is set arbitarily. In this case, the exact terms in orientation function O proposed by Kleiman and Page (2019) can be

written as follows:

$$\begin{aligned} \text{AUC}^{\text{arbit}} : \min_{z_{ij}} & \sum_{i=1}^{n} \sum_{j=1:x_i^* < x_j^*}^{n} z_{ij} \\ \text{s.t.} \quad & \widetilde{\mathbf{d}}_{ij} \sum_{k=1}^{K} \widetilde{\mathbf{v}}(k)(\mathbf{f}(x_i, k) - \mathbf{f}(x_j, k)) \geqslant \epsilon - z_{ij} \\ & \forall i, j : x_i^* < x_j^*, \\ & z_{ij} \geqslant 0 \end{aligned} \tag{B.2}$$

Note that $\text{AUC}^{\textbf{arbit}}$ has the same number of constraints and variables as $\textbf{AUC}^{\textbf{bin}}$. Once the LPs are solved, the loss function is calculated the same way as binary AUC.

Formulating Ratio Objectives

In this section, we study a subset of non-decomposable metrics, which are typically expressed as ratios of some combination of True Positive(TP), False Positives(FP), True Negatives(TN) and False Negatives(FN). These can be expressed in a general form as $\frac{a_{11}TP + a_{12}}{a_{21}TP + a_{22}FP + a_{23}FN + a_{24}}$, where a_{pq} are constants/cofficients which if set to 0, means the term is absent and not equal to zero in other cases. This formulation can used to define Fscore, F_β, Jaccard, IOU and Precision at fixed recall. In the following section, we describe the formulation of Fscore as a representative of this approach, other metrics can be formulated similarly.

Given Y the groud truth, our goal is to compute $\hat{Y}$ both of length n, which aligns with Y based on the specific metric. We first show how to write TP, FP, TN and FN wrt to these vectors.

$$\begin{aligned} TP &= Y^T \times \hat{Y} & FP &= (1 - Y)^T \times \hat{Y} \\ TN &= (1 - Y)^T \times (1 - \hat{Y}) & FN &= (Y)^T \times (1 - \hat{Y}) \end{aligned}$$

Formulating F-score

The F-score or F-measure is routinely used as a performance metric for different types of prediction problems, including binary classification and, multi-label classification. Compared to measures like error rate in binary classification and Hamming loss, it enforces a better balance between performance on the minority and the majority

classes, and, therefore, it is more suitable in the case of imbalanced data Dembczynski et al. (2011). F-score is defined as follows:

$$\text{F-score}(Y, \hat{Y}) = \frac{2P(Y, \hat{Y}) \times R(Y, \hat{Y})}{P(Y, \hat{Y}) + R(Y, \hat{Y})} \tag{B.3}$$

where P is the measure of precision defined as

$$P(Y, \hat{Y}) = \frac{TP}{TP + FP}$$

and R stands for the measure of recall, given as

$$R(Y, \hat{Y}) = \frac{TP}{TP + FN}$$

Plugging this in Eq(1), and replacing the formulations for TP, FP and FN(from Eq B) we get

$$\begin{aligned} \text{F-score}(Y, \hat{Y}) &= \frac{2TP}{2TP + FP + FN} \\ &= \frac{2(Y^T \times \hat{Y})}{\sum_{i=1}^{n} y_i + \sum_{i=1}^{n} \hat{y}_i} \\ &= \frac{2(Y^T \times \hat{Y})}{\mathbf{1}^T Y + \mathbf{1}^T \hat{Y}} \end{aligned} \tag{B.4}$$

where y_i refers to the ith element of Y(same for $\hat{y}_i$). $\mathbf{1}$ represents an all one vector in R^n. Note that in training, since Y is generally provided, we can assume $\mathbf{1}^T Y = \beta$ which is constant (the number of examples in the positive class in the ground truth). We can also represent the the values of $2Y$ as a coefficient matrix c, then the optimization problem for finding F-score can be written as

$$\begin{aligned} \underset{\hat{Y}}{\text{maximize}} \quad & \frac{c^T \hat{Y}}{1^T \hat{Y} + b} \\ \text{subject to} \quad & \hat{Y}_i \in [0, 1], \; i = 1, \ldots, n. \end{aligned} \tag{B.5}$$

Maximizing Jaccard Coefficient and F_β

The Jaccard Coefficient and Dice Index lead to similar formulation as F-score. The Jaccard coefficient can be expressed as:

$$\begin{aligned} \text{Jacc}(Y, \hat{Y}) &= \frac{TP}{TP + FP + FN} \\ &= \frac{(Y^T \times \hat{Y})}{\sum_{i=1}^{n} y_i + \sum_{i=1}^{n} \hat{y}_i - \sum_{i=1}^{n} y_i \times \hat{y}_i} \\ &= \frac{(Y^T \times \hat{Y})}{\mathbf{1}^T Y + (1 - Y)^T \hat{Y}} \end{aligned} \tag{B.6}$$

This can be equivalently written as a linear factional program as shown in Model(B.3) where $c = Y$, $d = (1 - Y)$ and $b = \mathbf{1}^T Y$. The rest of the construction is similar to F-score.
Note that F_β which is defined as

$$F_\beta(Y, \hat{Y}) = (1 + \mathbf{fi}^2) \frac{P(Y, \hat{Y}) \times R(Y, \hat{Y})}{\mathbf{fi}^2 P(Y, \hat{Y}) + R(Y, \hat{Y})} \tag{B.7}$$

where β is a user specified parameter (balancing the importance of precision and recall) also permits a similar formulation. Here we simply set $c = (1 + \mathbf{fi}^2)Y$, $d = \mathbf{1}$ and $b = \mathbf{fi}^2 \mathbf{1}^T Y$.

Maximizing P@R

We begin by defining the maximum precision at fixed minimum recall problem as

$$P@R\alpha = \text{maximize} \quad P \quad \text{s.t.} R \geqslant \alpha \tag{B.8}$$

$$= \text{maximize} \quad \frac{(Y^T \hat{Y})}{\mathbf{1}^T \hat{Y}} \quad \text{s.t.} \quad Y^T \hat{Y} \geqslant \alpha \mathbf{1}^T Y \tag{B.9}$$

This is again a linear fractional objective with a linear constraint. So we can write it as an equivalent Linear program using the same transformation where $c = Y$, $d = \mathbf{1}$ and $b = 0$. R@P on the other hand directly leads to a linear program.

Optimizing Non-negative Matrix Factorization (NMF)

Nonnegative matrix factorization is different from the other objectives we presented in that it is primarily used in unsupervised learning and does not satify the criteria for a metric loss function. It can still be formulated in a generalized non-decomposable form because **(i)** cannot be written as a sum over individual samples, **(ii)** leads to a model where the constraints depend on learned features.

Note that purpose of discussing NMF in this context, is not to provide a general purpose solver for the problem, and instead to assess whether NMF layers can serve as a regularizer or a clustering module, e.g., learning more interpretable attributes, co-segmentation and a substitute for clustering, see Trigeorgis et al. (2014) and Collins et al. (2018). The following description in this section and the experimental validation in the following section is a proof of principle instantiation of this idea.

We briefly review from Arora et al. (2012) and Recht et al. (2012), how NMF is written as a LP.

We know from Arora et al. (2012) that a NMF decomposition $V = FW$ where F is $s \times s'$ and W is $s' \times w$ and V and W have a row sum of 1, is 'separable' if the rows of W are simplicial and there is a permutation matrix $R \in \mathbb{R}^{s \times s}$ such that $RF = \begin{bmatrix} I_r & M \end{bmatrix}^T$. The top r rows of F contains so-called anchor words. Recht et al. (2012) proposed a data-driven model where the most salient features in the data are used to express the remaining features, given as $V \sim CV$, where C is of size $s \times s$. Assuming V admits a rank-r separable factorization, then $V = R^T \begin{bmatrix} I_r & 0 \\ M & 0 \end{bmatrix} RV = CV$. To show that thsis factorization is possible, we need to first make F square, which is why it is zero-padded to make it a size $s \times s$ matrix. Let $\tilde{p}$ be any vector which is used as the coefficient in the objective in the following model. According to Recht et al. (2012), any value for the entries of $\tilde{p}$ should suffice as long as they are distinct. with distinct values. Then the LP formulation is as follows:

$$\begin{aligned} \min_C \tilde{p}^T \mathrm{diag}(C) \quad \text{s.t.} \quad & CV = V, \\ & \mathrm{tr}(C) = r,\ C_{jj} \leqslant 1\ \forall j,\ C_{ij} \leqslant C_{jj}\ \forall ij, \\ & C \geqslant 0 \end{aligned}$$

With C in hand, W is constructed by extracting rows of V for those indices k where

$C_{kk} = 1$. F is constructed by extracting rows of C which correspond to k where $C_{kk} = 1$.

Experimental results on Nonnegative Matrix Factorization

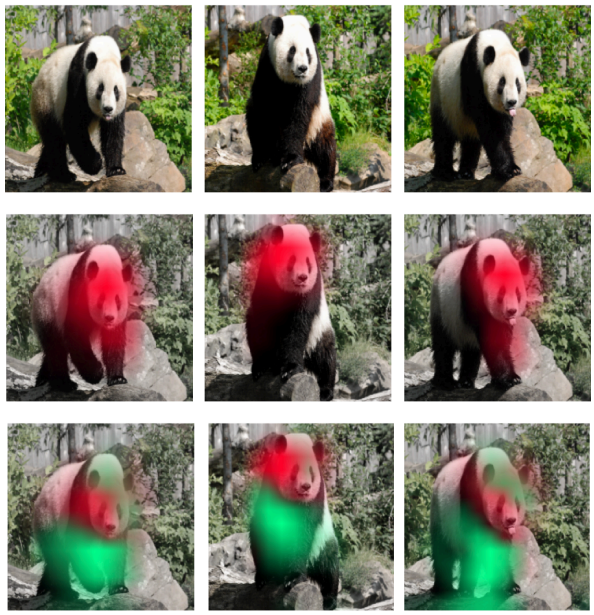


Figure B.1: NMF example. Three rows correspond to original images, $k = 1$ and $k = 2$ respectively.

We demonstrate applicability of our strategy to nonnegative matrix factorization (NMF) by performing a rank k factorization on Convolutional Neural Network (CNN) activations as an example, following Collins et al. (2018). Recall that the activation tensor of an image at some layer in CNN has the shape $V \in R^{c \times h \times w}$ where h, w are the spatial sizes and c is the number of channels. We can reshape it into $V \in \mathbb{R}^{c \times (h \cdot w)}$ and calculate a rank k NMF for V: $V = FW$. Each row W_j of the resultant $W \in \mathbb{R}^{k \times (h \cdot w)}$ can be reshaped into a heat map of dimension $h \times w$ which highlights regions in the image that correspond to the factor W_j. We show an example for $k = 1, 2$ in Fig. B.1. We can see that heatmap consistently captures a meaningful part/concept in the examples. Currently, our memory consumption increases quickly with c here since the constraint matrix in our LP formulation is of size $O(c^2) \times O(c^2)$. This makes our method only work for small c on a GPU with 11GB memory (here, we use $c = 20$). This scaling issue can be possibly solved by utilizing sparsity in the constraint matrix, but the sparse matrix operations are cur-

Objective	g	h	E	F	p	B	G	q
AUC	$\mathbf{1} \in \mathbb{Z}^{\lvert T\rvert\times\lvert N\rvert}$	-	$\mathbf{-1} \in \mathbb{Z}^{\lvert T\rvert\times\lvert N\rvert}$	-	$p_{ij} = (f(x_i) - f(x_j) - \epsilon)$	-	-	-
AUC_μ^{bin}	$\mathbf{1}^\dagger$,	-	$\mathbf{-1}^\dagger$	-	$p_{ij} = (\mathbf{f}^\ddagger_{ij} - \epsilon)$	-	-	-
F-score	$\mathbf{c}$	0	$\begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}^\S$	$\begin{bmatrix} -1 \\ \phi(f(x_i)) \\ -(1+\phi(f(x_i))) \end{bmatrix}^\S$	0	$\mathbf{d}$	b	b

Table B.1: Table showing the general LP coefficients for each model. †: length based on problem setting; ‡: $\mathbf{f}^\ddagger_{ij} = \widetilde{\mathbf{d}}_{ij}(f(x_i, y_{C(x_i)}) - f(x_j, y_{C(x_i)}) + f(x_j, y_{C(x_j)}) - f(x_i, y_{C(x_j)}))$; §: one block for each $i \in [1, ..n]$. We do not include NMF in this table, as its formulation as a general LP is more verbose including vectorization of matrices and kronecker product calculations.

rently not well supported on mainstream deep learning platforms like PyTorch and Tensorflow. Since our method provides backward gradients for the NMF operation, the heatmap generated here can, in fact, be used to construct a loss function during training in order to learn a interpretable models.

Verification of Unrolling gradient and the one provided by $\tilde{A}^{-1}\tilde{b}$

We use Fscore formulation as an example. For input sample x, the neural network predicts a score $f(x)$, and then the scores of a batch of samples will be used in solving the linear programming form of Fscore and be used to construct the loss function. We compute the gradient from the final loss function back to the predicted scores from the neural network and compare two approaches: one is that we use $z = \tilde{A}^{-1}\tilde{b}$ as the solution (the one we used in our experiment) where we can compute gradient by only one step, another one is that we directly use y_t resulting from the Newton iterations as the solution and compute gradients by unrolling those iterations. We then compute the cosine value between these two gradient vectors. By experiments on 100 randomly sampled batches, the average cosine value is 0.9991, which means the two gradients are highly consistent.

Table B.2: Ablation study of ϵ on Cat&Dog dataset.

Positive Ratio	91%	83%	71%	50%
Ours($\epsilon = 0.1$)	71.3	77.0	84.4	87.3
Ours($\epsilon = 0.01$)	78.6	81.3	85.6	87.8
Ours($\epsilon = 0.001$)	65.9	71.3	71.8	76.1

B.1 F-score

The F-score (or F-measure) is a representative of objectives expressed as ratios of some combination of True positives (TP), False positives (FP), True negatives (TN) and False negatives (FN). The general form of the ratio functions and formulations for other objectives is in the supplement. Specifically, F-score is defined as follows:

Definition B.1 (F-score). $\text{F-}score = \frac{2\times(Precision\times Recall)}{Precision+Recall} = \frac{2TP}{2TP+FP+FN} = 2(Y^\intercal \times \hat{Y})/(\mathbf{1}^\intercal Y + \mathbf{1}^\intercal \hat{Y})$.

The second equality in the definition is due to a simplification of the precision $\left(\frac{TP}{TP+FP}\right)$ and recall $\left(\frac{TP}{TP+FN}\right)$ expressions based on Dembczynski et al. (2011). The last part is obtained by replacing TP with $Y^\intercal \times \hat{Y}$, FP with $(1-Y)^\intercal \times \hat{Y}$ and FN with $(Y)^\intercal \times (1-\hat{Y})$ as functions of Y and $\hat{Y}$. This leads to the following integer fractional optimization model,

$$\text{F-score} = \max_{\hat{Y}} \frac{\mathbf{c}^\intercal \hat{Y}}{1^\intercal \hat{Y} + b} \quad \text{s.t.} \quad \hat{Y}_i \in [0,1],\ i = 1,\dots,n \text{ where } \mathbf{c} = 2Y \text{and } b = \sum_{i=1}^{n} Y_i. \tag{B.10}$$

To solve this, we first relax the constraint on $\hat{Y}$ and reformulate the model as the following LP, by introducing two variables $z \in R^n$ and $t \in R^1$ where $z = \frac{b\hat{Y}}{1^\intercal \hat{Y}+b}$, $t = \frac{b}{1^\intercal \hat{Y}+b}$ and $i \in \{1,\cdots,n\}$:

$$\max_{z,t} \frac{\mathbf{c}^\intercal z}{b} \quad \text{s.t} \quad \underbrace{1^\intercal z + bt = b}_{(a)};\ \underbrace{z_i \leqslant t}_{(b)};\ \underbrace{\phi(f(x_i))t \leqslant z_i \leqslant (1+\phi(f(x_i)))t}_{(c)};\ 1 \geqslant z_i, t \geqslant 0$$

The supplement simple algebraic adjustments to derive the three constraints (a)(b)(c). This model imposes $4n$ constraints for n samples. Since this is a maximization, a solution to the LP, $O_{\hat{Y}}$, is an upper bound on the integer objective opt^*, and serves as the loss.

B.2 How to choose ϵ

How to choose ϵ? If we can successfully retrieve the active constraints at the optimal solution, we do not need to store the intermediate iterates y_t at all during the forward pass (memory efficient). However, setting ϵ correctly can be tricky for arbitrary polyhedra since it depends on the geometric properties such as facets and vertices that may be difficult to enumerate. One possible way to get around this is to use a "burn-in" period in which we increase ϵ slightly in each iteration (of deep network training) and backpropagate through the unrolled Newton's iterations during this period. Once we see that the convergence profile has stabilized, we can fix ϵ at that value and start using the complementarity conditions and derive the active linear system $\tilde{A}^{-1}\tilde{b}$ as discussed above.

Appendix C

An Online Riemannian PCA for Stochastic Canonical Correlation Analysis: Appendix

Proof of Theorem 1

We first restate the assumptions from section 3.1:

Assumptions:

(a) The random variables $\mathbf{X} \sim \mathcal{N}(\mathbf{0}, \Sigma_x)$ and $\mathbf{Y} \sim \mathcal{N}(\mathbf{0}, \Sigma_y)$ with $\Sigma_x \preceq cI_d$ and $\Sigma_y \preceq cI_d$ for some $c > 0$.

(b) The samples X and Y drawn from $\mathcal{X}$ and $\mathcal{Y}$ respectively have zero mean.

(c) For a given $k \leqslant d$, Σ_x and Σ_y have non-zero top-k eigen values.

Recall that F and $\widetilde{F}$ are the optimal values of the true and approximated CCA objective in (1) and (4) respectively, we next restate Theorem 1 and give its proof:

Theorem C.1. *Under the assumptions and notations above, the approximation error* $E = \|F - \widetilde{F}\|$ *is bounded and goes to zero while the whitening constraints in (4b) are satisfied.*

Proof. Let U^*, V^* be the true solution of CCA. Let $U = \widetilde{U} S_u Q_u, V = \widetilde{V} S_v Q_v$ be the solution of (4) with $\widetilde{U}, \widetilde{V}$ be the PCA solutions of X and Y respectively with $S_u Q_u = \widetilde{U}^\top U^*$ and $S_v Q_v = \widetilde{V}^\top V^*$ (using RQ decomposition). Let $\widehat{X} = X\widetilde{U}\widetilde{U}^\top$ and $\widehat{Y} = Y\widetilde{V}\widetilde{V}^\top$ be the reconstruction of X and Y using principal vectors.

Then, we can write

$$\widetilde{F} = \text{trace}\left(U^T C_{XY} V\right) = \text{trace}\left(\frac{1}{N}\left(\widehat{X}U^*\right)^T \widehat{Y}V^*\right) \tag{C.1}$$

Similarly we can write $F = \text{trace}\left(\frac{1}{N}\left(XU^*\right)^T YV^*\right)$.

Using Def. 1, we know that $\widehat{X}$, $\widehat{Y}$ follow a sub-Gaussian distribution.

Consider the approximation error between the objective functions as $E = |F - \widetilde{F}|$. Due to von Neumann's trace inequality and CauchySchwarz inequality, we have

$$\begin{aligned}
E &= \frac{1}{N}|\text{trace}\left((U^*)^T\widehat{X}^T\widehat{Y}(V^*) - (U^*)^T X^T Y(V^*)\right)| \\
&\leqslant |\text{trace}\left((U^*)^T\left(\left(\widehat{X}-X\right)^T\left(\widehat{Y}-Y\right) - 2X^TY + X^T\widehat{Y} + \widehat{X}^TY\right)(V^*)\right)| \\
&\leqslant \sum_i \sigma_i(\widehat{X}_u - X_u)\sigma_i(\widehat{Y}_v - Y_v) + \sum_i \sigma_i(\widehat{X}_u - X_u)\sigma_i(Y_v) + \sum_i \sigma_i(\widehat{Y}_v - Y_v)\sigma_i(X_u) \\
&\leqslant \|\left(\widehat{X}_u - X_u\right)\|_F\|\left(\widehat{Y}_v - Y_v\right)\|_F + \|\left(\widehat{X}_u - X_u\right)\|_F\|Y_v\|_F + \left(\widehat{Y}_v - Y_v\right)\|_F\|X_u\|_F
\end{aligned} \tag{A.1}$$

Here $A_u = AU^*$ and $A_v = AV^*$ for any suitable A. where $\sigma_i(A)$ denote the i-th singular value of matrix A and $\|\bullet\|_F$ denotes the Frobenius norm.

Now, using Proposition 1, we get

$$\begin{aligned}
\|\left(\widehat{X}_u - X_u\right)\|_F &\leqslant \min\left(\sqrt{2k}\|\Delta_x\|_2, \frac{2\|\Delta_x\|_2^2}{\lambda_k^x - \lambda_{k+1}^x}\right) \\
\|\left(\widehat{Y}_v - Y_v\right)\|_F &\leqslant \min\left(\sqrt{2k}\|\Delta_y\|_2, \frac{2\|\Delta_y\|_2^2}{\lambda_k^y - \lambda_{k+1}^y}\right)
\end{aligned} \tag{A.2}$$

where

$$\Delta_x = C(X_u) - C(\widehat{X}_u), \quad \Delta_y = C(Y_v) - C(\widehat{Y}_v).$$

Here λ^xs and λ^ys are the eigen values of $C(X_u)$ and $C(Y_v)$ respectively. Now, assume that $C(X_u) = I_k$ and $C(Y_v) = I_k$ as X_u and Y_v are solutions of Eq. 1. Furthermore assume $\lambda_k^x - \lambda_{k+1}^x \geqslant \Lambda$ and $\lambda_k^y - \lambda_{k+1}^y \geqslant \Lambda$ for some $\Lambda > 0$. Then, we can rewrite Eq. (A.1) as

$$E \leqslant \min\left(\sqrt{2k}\|I_k - C(\widehat{X}_u)\|_2, \frac{2\|I_k - C(\widehat{X}_u)\|_2^2}{\Lambda}\right)\min\left(\sqrt{2k}\|I_k - C(\widehat{Y}_v)\|_2, \frac{2\|I_k - C(\widehat{Y}_v)\|_2^2}{\Lambda}\right) +$$
$$\min\left(\sqrt{2k}\|I_k - C(\widehat{X}_u)\|_2, \frac{2\|I_k - C(\widehat{X}_u)\|_2^2}{\Lambda}\right)\|Y_v\|_F +$$
$$\min\left(\sqrt{2k}\|I_k - C(\widehat{Y}_v)\|_2, \frac{2\|I_k - C(\widehat{Y}_v)\|_2^2}{\Lambda}\right)\|X_u\|_F$$

And as $C(\widehat{X}_u) \to I_k$ or $C(\widehat{Y}_v) \to I_k$, $E \to 0$. Observe that, the limiting conditions for $C(\widehat{X}_u)$ and $C(\widehat{Y}_v)$ can be satisfied by the "whitening" constraint. In other words, as $C(X_u) = I_k$ and $C(Y_v) = I_k$, $C(\widehat{X}_u)$ and $C(\widehat{Y}_v)$ converge to $C(X_u)$ and $C(Y_v)$, the approximation error goes to zero. □

RSG+ algorithm

Here we show our algorithm with more details about the gradients in every step in Alg.4.

Implementation details of CCA on fixed dataset

Implementation details. On all three benchmark datasets, we only passed the data once for both our RSG+ and MSG Arora et al. (2017) and we use the code from Arora et al. (2017) to produce MSG results. We conducted experiments on different dimensions of target space: $k = 1, 2, 4$. The choice of k is motivated by the fact that the spectrum of the datasets decays quickly. Since our RSG+ processes data in small blocks, we let data come in mini-batches (mini-batch size was set to 100).

Runtime of RSG+ and baseline methods

The runtime comparison of RSG+ and MSG is reported in Table C.1. Our algorithm is 5–10 times faster.

We also plot the runtime of our algorithm under different data dimension (set $d_x = d_y = d$) and number of total samples sampled from joint gaussian distribution in Fig. C.1.

Algorithm 4: Riemannian SGD based algorithm (RSG+) to compute canonical directions

$X \in \mathbf{R}^{N \times d_x}$, $Y \in \mathbf{R}^{N \times d_y}$, $k > 0$ $U \in \mathbf{R}^{d_x \times k}$, $V \in \mathbf{R}^{d_y \times k}$

Initialize $\widetilde{U}, \widetilde{V}, Q_u, Q_v, S_u, S_v$;

Partition data X, Y into batches of size B. Let j^{th} batch be denoted by X_j and Y_j ;

$j \in \{1, \cdots, \lfloor \frac{N}{B} \rfloor\}$

Gradient for top-k principal vectors: calculating $\nabla_{\widetilde{U}} \widetilde{F}_{pri}, \nabla_{\widetilde{V}} \widetilde{F}_{pri}$

1. Partition X_j (Y_j) into L ($L = \lfloor \frac{B}{k} \rfloor$) blocks of size $d_x \times k$ ($d_y \times k$);
2. Let the l^{th} block be denoted by Z_l^x (Z_l^y);
3. Orthogonalize each block and let the orthogonalized block be denoted by $\hat{Z}_l^x$ ($\hat{Z}_l^y$);
4. Let the subspace spanned by each $\hat{Z}_l^x$ (and $\hat{Z}_l^y$) be $\hat{\mathcal{Z}}_l^x \in \mathsf{Gr}(k, d_x)$ (and $\hat{\mathcal{Z}}_l^y \in \mathsf{Gr}(k, d_y)$);

$$\nabla_{\widetilde{U}} \widetilde{F}_{pri} = -\sum_l \mathsf{Exp}_{\widetilde{U}}^{-1}\left(\hat{\mathcal{Z}}_l^x\right) \quad \nabla_{\widetilde{V}} \widetilde{F}_{pri} = -\sum_l \mathsf{Exp}_{\widetilde{V}}^{-1}\left(\hat{\mathcal{Z}}_l^y\right) \tag{C.2}$$

Gradient from the original CCA objective: calculating $\nabla_{\widetilde{U}} \widetilde{F}_{can}, \nabla_{\widetilde{V}} \widetilde{F}_{can}, \nabla_{Q_u} \widetilde{F}_{can}, \nabla_{Q_v} \widetilde{F}_{can}, \nabla_{S_u} \widetilde{F}_{can}, \nabla_{S_v} \widetilde{F}_{can}$

$$\nabla_{\widetilde{U}} \widetilde{F}_{can} = \frac{\partial \widetilde{F}}{\partial \widetilde{U}} - \widetilde{U} \frac{\partial \widetilde{F}}{\partial \widetilde{U}}^T \widetilde{U} \qquad \nabla_{\widetilde{V}} \widetilde{F}_{can} = \frac{\partial \widetilde{F}}{\partial \widetilde{V}} - \widetilde{V} \frac{\partial \widetilde{F}}{\partial \widetilde{V}}^T \widetilde{V};$$

$$\nabla_{Q_u} \widetilde{F}_{can} = \frac{\partial \widetilde{F}}{\partial Q_u} - \frac{\partial \widetilde{F}}{\partial Q_u}^T \qquad \nabla_{Q_v} \widetilde{F}_{can} = \frac{\partial \widetilde{F}}{\partial Q_v} - \frac{\partial \widetilde{F}}{\partial Q_v}^T;$$

$$\nabla_{S_u} \widetilde{F}_{can} = \mathsf{Upper}\left(\frac{\partial \widetilde{F}}{\partial S_u}\right) \qquad \nabla_{S_v} \widetilde{F}_{can} = \mathsf{Upper}\left(\frac{\partial \widetilde{F}}{\partial S_v}\right);$$

Here, Upper returns the upper triangular matrix of the input matrix and $\frac{\partial \widetilde{F}}{\partial \widetilde{U}}, \frac{\partial \widetilde{F}}{\partial \widetilde{V}}, \frac{\partial \widetilde{F}}{\partial Q_u}, \frac{\partial \widetilde{F}}{\partial Q_v}, \frac{\partial \widetilde{F}}{\partial S_u}, \frac{\partial \widetilde{F}}{\partial S_v}$ give the Euclidean gradients. For completeness, the closed form expression of the gradients is,

$$\begin{aligned}
&\frac{\partial \widetilde{F}}{\partial \widetilde{U}} = -C_{XY} \widetilde{V} Q_v S_v Q_u^T S_u^T \qquad \frac{\partial \widetilde{F}}{\partial Q_u} = -\widetilde{U}^T C_{XY} \widetilde{V} Q_v S_v S_u^T \qquad \frac{\partial \widetilde{F}}{\partial S_u} = -Q_u^T \widetilde{U}^T C_{XY} \widetilde{V} Q_v S_v \\
&\frac{\partial \widetilde{F}}{\partial \widetilde{V}} = -C_{YX} \widetilde{U} Q_u S_u Q_v^T S_v^T \qquad \frac{\partial \widetilde{F}}{\partial Q_v} = -\widetilde{V}^T C_{YX} \widetilde{U} Q_u S_u S_v^T \qquad \frac{\partial \widetilde{F}}{\partial S_v} = -S_u^T Q_u^T \widetilde{U}^T C_{XY} \widetilde{V} Q_v
\end{aligned} \tag{C.3}$$

Gradient to update canonical directions

$\nabla_{\widetilde{U}} \widetilde{F} = \nabla_{\widetilde{U}} \widetilde{F}_{pri} + \nabla_{\widetilde{U}} \widetilde{F}_{can}$ $\qquad \nabla_{\widetilde{V}} \widetilde{F} = \nabla_{\widetilde{V}} \widetilde{F}_{pri} + \nabla_{\widetilde{V}} \widetilde{F}_{can}$;

$\nabla_X \widetilde{F} = \nabla_X \widetilde{F}_{can}$ where, X is a generic entity: $X \in \{Q_u, Q_v, S_u, S_v\}$;

Batch update of canonical directions

$A = \mathsf{Exp}_A\left(-\gamma_j \nabla_A \widetilde{F}\right)$ where, A is a generic entity: $A \in \{\widetilde{U}, \widetilde{V}, Q_u, Q_v, S_u, S_v\}$;

$U = \widetilde{U} Q_u S_u$ and $V = \widetilde{V} Q_v S_v$;

Table C.1: Wallclock runtime of one pass through the data of our RSG+ and MSG on MNIST, Mediamill and CIFAR (average of 5 runs).

	MNIST			Mediamill			CIFAR		
Time (s)	$k = 1$	$k = 2$	$k = 4$	$k = 1$	$k = 2$	$k = 4$	$k = 1$	$k = 2$	$k = 4$
RSG+ (Ours)	4.16	4.24	4.71	1.89	1.60	1.44	14.80	17.22	22.10
MSG	35.32	42.09	49.17	11.59	14.21	17.34	80.21	100.80	106.55

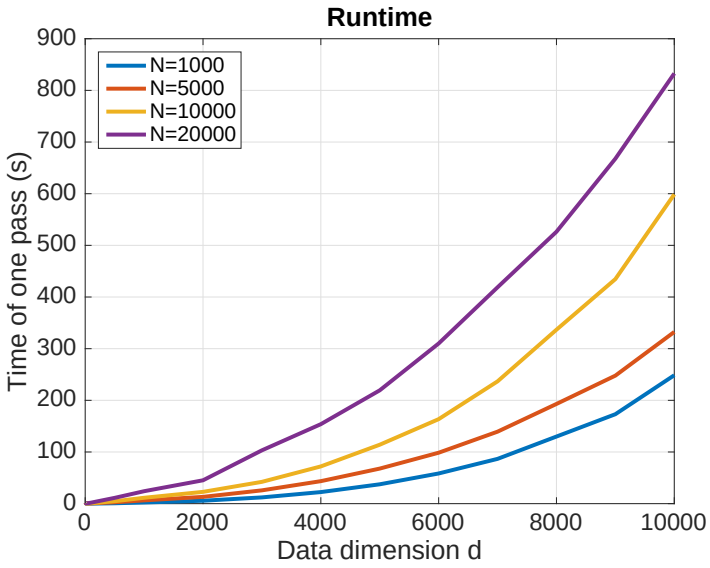


Figure C.1: Runtime of RSG+ under different data dimensions and size of datasets.

Error metrics for fairness

Equality of Opportunity (EO) Hardt et al. (2016a): A classifier h is said to satisfy EO if the prediction is independent of the protected attribute s (in our experiment s is a binary variable where $s = 1$ stands for *Male* and $s = 0$ stands for *Female*) for classification label $y \in \{0, 1\}$. We use the difference of false negative rate (conditioned on $y = 1$) across two groups identified by protected attribute s as the error metric, and we denote it as DEO.

Demographic Parity (DP) Yao and Huang (2017): A classifier h satisfies DP if the likelihodd of making a misclassification among the positive predictions of the classifier is independent of the protected attribute s. We denote the difference of demographic parity between two groups identified by the protected attribute as DDP.

Implementation details of fairness experiments

Implementation details. The network is trained for 20 epochs with learning rate 0.01 and batch size 256. We follow Donini et al. (2018) to use NVP (novel validation procedure) to evaluate our result: first we search for hyperparameters that achieves the highest classification score and then report the performance of the model which gets minimum fairness error metrics with accuracy within the highest 90% accuracies. When we apply our RSG+ on certain layers, we first use randomized projection to project the feature into 1k dimension, and then extract top-10 canonical components for training. Similar to our previous experiments on DeepCCA, the batch method does not scale to 1k dimension.

Resnet-18 architecture and position of Conv-0,1,2

The Resnet-18 contains a first convolutional layer followed by normalization, nonlinear activation, and max pooling. Then it has four residual blocks, followed by average polling and a fully connected layer. We denote the position after the first convolutional layer as conv0, the position after the first residual block as conv1 and the position after the second residual block as conv2. We choose early layers since late layers close to the final fully connected layer will have feature that is more directly relevant to the classification variable (*attractiveness* in this case).

Yger et al. 2012

We implemented the method from Yger et al. (2012) and conduct experiments on the three datasets above. The results are shown in Table C.2. We tune the step size between $[0.0001, 0.1]$ and $\beta = 0.99$ as used in their paper. On MNIST and MEDIAMILL, the method performs comparably with ours except $k = 4$ case on MNIST where it does not converge well. Since this algorithms also has a d^3 complexity, the runtime is $100\times$ more than ours on MNIST and $20\times$ more on Mediamill. On CIFAR10, we fail to find a suitable step size for convergence.

	MNIST			Mediamill		
Performance	$k = 1$	$k = 2$	$k = 4$	$k = 1$	$k = 2$	$k = 4$
PCC	0.93	0.81	0.53	0.55	0.61	0.51
Time (s)	575.88	536.46	540.91	41.89	28.66	28.76

Table C.2: Results of Yger et al. (2012) (on CIFAR-10, our implementation of Yger et al. (2012) faces convergence issues).

Appendix D

Neural TMDlayer: Modeling Instantaneous flow of features via SDE Generators: Appendix

D.1 Details regarding Point cloud Transformer layer proposed in Guo et al. (2020)

Here, we provide additional low-level details relevant to our point cloud learning experiments.

Let Q, K, V denote the query, key and value matrices respectively, generated by linear transformations of the input features $F_{in} \in R^{N \times d_e}$ as follows:

$$(Q, K, V) = F_{in} \cdot (W_q, W_k, W_v) \tag{D.1}$$

$$Q, K \in R^{N \times d_a}, \quad V \in R^{N \times d_e} \tag{D.2}$$

$$W_q, W_k \in R^{d_e \times d_a}, \quad W_v \in R^{d_e \times d_e} \tag{D.3}$$

where W_q, W_k and W_v are the shared learnable linear transformations, and d_a is the dimension of the query and key vectors. Note that d_a may not be equal to d_e.

Next, we calculate the attention weights:

$$\tilde{A} = (\tilde{\alpha})_{i,j} = Q \times K^T \tag{D.4}$$

The weights are then normalized to give $A = (\alpha)_{i,j}$:

$$\bar{\alpha}_{i,j} = \frac{\tilde{\alpha}_{i,j}}{\sqrt{(d_a)}} \tag{D.5}$$

$$\alpha_{i,j} = \text{softmax}(\bar{\alpha}_{i,j}) = \frac{\exp(\bar{\alpha}_{i,j})}{\sum_k \exp(\bar{\alpha}_{i,k})} \tag{D.6}$$

The self-attention output F_sa is computed as:

$$F_{sa} = A \cdot V \tag{D.7}$$

In Guo et al. (2020), they design an offset-attention which is used on top of F_{sa}. After computing F_{sa}, the final output of the transformer layer is specified as

$$F_{out} = \text{FF}(F_{in} - F_{sa}) + F_{in} \tag{D.8}$$

where FF is the feed-forward layer comprised of the linear layer, normalization layer and nonlinear layer.

D.2 Details regarding GNNs for few-shot learning proposed in Kim et al. (2019)

Here, we provide additional low-level details relevant to our few-shot learning experiments.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}; \mathcal{T})$ be the graph constructed with samples from the task $\mathcal{T}$, where $\mathcal{V} = \{V_i\}_{i=1,\ldots,|\mathcal{T}|}$ and $\mathcal{E} := \{E_{ij}\}_{i,j=1,\ldots,|\mathcal{T}|}$ denote the set of nodes and edges of the graph, respectively. Let v_i and e_{ij} be the node feature of V_i and the edge feature of E_{ij}, respectively. We take $|\mathcal{T}| = N \times K + T$ to be the total number of samples in the task $\mathcal{T}$. Each ground-truth edge-label y_{ij} is defined by the ground-truth node labels as $y_{ij} = 1$ if $y_i = y_j; 0$ otherwise.

In the proposed GNN in Kim et al. (2019), in the experiments, node feature update and edge feature update are done iteratively for L layers. The node features are initialized by the output of the convolutional embedding network and the edge features represent the strengths of the intra- and inter-class relations between two connected nodes. We treat the node feature update function as f and use our TMD-layer on top of it. In Kim et al. (2019), the proposed node feature update function

is:

$$v_i^l = g_v^l([\sum_j \tilde{e}_{ij1}^{l-1} v_j^{l-1} \| \sum_j \tilde{e}_{ij2}^{l-1} v^{l-1} v_j]; \theta_v^j) \tag{D.9}$$

where each edge feature $e_{ij} = \{e_{ijd}\}_{d=1}^2$ is a 2-dimensional vector $\tilde{e}_{ijd} = \frac{e_{ijd}}{\sum_k e_{ikd}}$, and g_d^l is the feature transformation network with the parameter set θ_v^l.

D.3 Details regarding deep active contour model

Here, we provide additional details relevant to our segmentation experiments with our proposed deep active contour model. In our segmentation experiments, we treat the entire update module introduced here as f and use our TMDlayer on top of it.

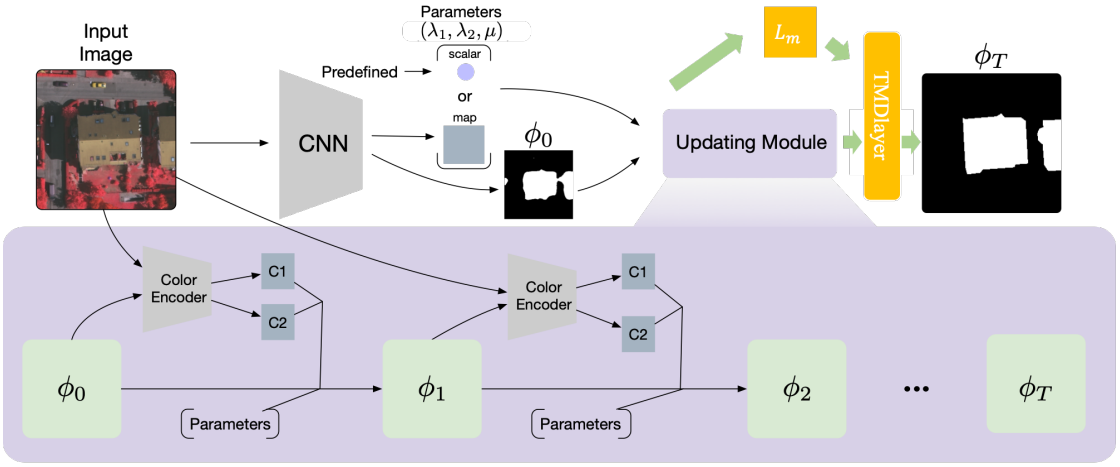


Figure D.1: **Architecture of our model:** The image is fed into the CNN encoder to produce initial level set function and needed parameters. The parameters could be chosen to be a learned map or simply a constant. The updating module will evolve the level set function using initial ϕ_0 and the parameters, producing the final ϕ_T. Within the updating module, in every step the color encoder learns C1, C2 from the image and the current ϕ_t. The images of ϕ_0 and ϕ_T are binarized for a clear view. Our TMDlayer is added to every layer in the updating module. Here we only show once for clear view.

Overview. Given an input image I, our model learns to perform weighted length minimization and learns feature-value sets in the image region, which are then used to perform T steps of a level set update from the initial configuration ϕ_0. Our framework also enables a semi-supervised segmentation once we add a modified variational energy term which our level set updates will explicitly optimize as a loss func-

tion (for the unlabeled images). The level set update takes place within the network, which seamlessly enables end-to-end training of our whole network.

Weighted Length Minimization

In active contour models, a length minimization term is used to yield curvature-based evolution that is responsible for a smooth curve representation of the desired object boundary. More precisely, one uses the arc length minimization Chan and Vese (2001):

$$\inf_{C} \int_0^1 |C'(s)| ds \tag{D.10}$$

However, minimizing the curve length alone will not suffice as a means to segment object boundaries. A weighted curve length is more meaningful in this context, especially if the weighting function is derived from image data and sensitive to edges in the image. Thus, Caselles et al. proposed geodesic active contour (GAC), which can be viewed as a weighted length minimization written as,

$$\inf_{C} \int_0^1 g(C(s)) ds \tag{D.11}$$

where $g(C(s)) := \frac{1}{1+|\nabla G_\sigma * I|^2}$, G_σ is a Gaussian with a variance parameter σ. Note that this choice of weighting function is hand-crafted like so many others that have been used in literature. Indeed, such hand-crafted weighting functions in general can be the right choice for a small class of images and tasks but is often suboptimal in general. To overcome this limitation, we replace the image gradient based weighting function $g(C(s))$ with a term parameterized by deep neural networks thereby allowing the weighting function to be data driven as follows:

$$\inf_{C} \int_0^1 \mu(C(s)) ds \quad \text{where} \quad \mu(\cdot) = \text{CNN}(I) \tag{D.12}$$

where $\mu(\cdot)$, which can be viewed as a 2-D function in the discrete case, is parameterized using a convolutional neural network. As we see in our experiments, this leads to better segmentation results.

In a level set framework, this weighted length minimization is usually performed

by solving the strong form given by the following PDE:

$$\left[\frac{\partial \phi}{\partial t}\right]_{\text{length}} = \text{div}\left(\mu \frac{\nabla \phi}{|\nabla \phi|}\right)|\nabla \phi| \tag{D.13}$$

Chan-Vese in Learned Feature (Latent) Space

We now develop a Chan-Vese based active contour energy term that is set in a latent space i.e., a feature space where the features are learned by a deep neural network (DNN). To make the description simple, we will derive the loss function for a single image I (or X in the main paper), and so in this section (x, y) represents 2D pixel coordinates in X. As usual, the total loss function is given by summing over the examples in the training set. Let us first recall the classical Chan-Vese active contour energy functional Chan and Vese (2001) given by,

$$F(c_1, c_2, C) = \lambda_1 \int_{in(C)} |I(x, y) - c_1|^2 \, dxdy + \tag{D.14}$$

$$\lambda_2 \int_{out(C)} |I(x, y) - c_2|^2 \, dxdy \tag{D.15}$$

In (D.14), c_1 is the color averaged over pixels inside the contour C and c_2 is the color averaged over the pixels outside.

$$c_1 = \frac{\int \mathbf{1}_{\phi>0} \cdot \phi dxdy}{\int \mathbf{1}_{\phi>0} dxdy}, \; c_2 = \frac{\int \mathbf{1}_{\phi<0} \cdot \phi dxdy}{\int \mathbf{1}_{\phi<0} dxdy} \tag{D.16}$$

This assumption has been shown to work well for simple images, but for most natural images that we may want to segment, this assumption may be too restrictive. In this paper, we replace the weighting factors $(I(x, y) - c_i)^2$ in the Chan-Vese energy with an energy in the learned feature space obtained using a deep network. The intuition here is that rather than perform variance minimization in the native space, we can perform this operation in a learned feature space which better captures the unknown homogeneity property of each region within the image. Note that the transformation from image space to the feature space can be achieved by simply modifying original scalar λ into a spatially variant map. Thus, the final en-

ergy functional of our active contour is given by,

$$\begin{aligned} F(c_1, c_2, C) = &\int_{in(C)} \lambda_1(x, y) \cdot |I(x, y) - c_1|^2 \, dxdy \\ &+ \int_{out(C)} \lambda_2(x, y) \cdot |I(x, y) - c_2|^2 \, dxdy \end{aligned} \tag{D.17}$$

Another observation one can make is that if we restrict $\lambda(\cdot)$ to learn a nonnegative function, it becomes similar to the localized kernel weighting function. Thus, the functional based on the image data denoted by $[F(\phi)]_{C_1}$ becomes:

$$\underbrace{\sum_{i=1,\dots,n} \int_{\Omega} \lambda_{1i}(x, y) \cdot (|I(x, y) - C_{1i}|^2) H_\epsilon(\phi) dxdy}_{[F(\phi)]_{C_1}} \tag{D.18}$$

where C_{1i} refers to the ith channel of C_1 (e.g., there are 3 channels in a color image) and we follow Chan and Vese (2001) to define a Heaviside function H_ϵ which acts as an differentiable approximation to the indicator function $\mathbf{1}$ defined as,

$$H_\epsilon(\cdot) := \frac{1}{2}\left(1 + \frac{2}{\pi} \arctan\left(\frac{\cdot}{\epsilon}\right)\right), \tag{D.19}$$

where ϵ is a specified constant (hyperparameter).

After deriving the corresponding Euler-Lagrange equation of (D.18) and parameterizing the descent direction with an artificial time parameter $t \geqslant 0$, we obtain,

$$\left[\frac{\partial \phi}{\partial t}\right]_{C1} = -\delta_\epsilon(\phi) \left[\sum_{i=1,\dots,n} \lambda_{1i}(x, y) \cdot (|I(x, y) - C_{1i}|^2)\right] \tag{D.20}$$

Similarly, we can have an energy and corresponding evolution equation for the region outside outside the active contour,

$$\left[\frac{\partial \phi}{\partial t}\right]_{C2} = \delta_\epsilon(\phi) \left[\sum_{i=1,\dots,n} \lambda_{2i}(x, y) \cdot (|I(x, y) - C_{2i}|^2)\right] \tag{D.21}$$

where we follow Chan and Vese (2001) to define $\delta_\epsilon = H'_\epsilon$.

Initial Level Set Function

Active contour models usually need manual initialization, which often influences the quality of the final result. This makes it impractical in large-scale applications. We propose to learn the initial level set function by using a segmentation neural network. That is,

$$\phi_0 = f(I) \tag{D.22}$$

where I is the given image and f denotes an encoder CNN. We see in our experiments that despite the "multi-stage" nature of this setup (CNN followed by level set updates), it can indeed be trained end-to-end.

The Full Evolution Model

The full ('total') model evolution is simply the sum of all the terms above:

$$\left[\frac{\partial \phi}{\partial t}\right]_{\text{total}} = \left[\frac{\partial \phi}{\partial t}\right]_{C1} + \left[\frac{\partial \phi}{\partial t}\right]_{C2} + \left[\frac{\partial \phi}{\partial t}\right]_{\text{length}} \tag{D.23}$$

Training Procedure

For training, we first learn an initial level set function ϕ_0 from an encoder CNN shown in (D.22). Then, we perform T update steps according to (D.23). The final update after ϕ_T steps will be used to calculate the loss based on the ground truth segmentation mask label M_{GT}

$$\text{Loss} = CE(H_\epsilon(\phi_T), M_{GT}) \tag{D.24}$$

where CE refers to the standard cross-entropy loss.

references

Abbas, Ahmed, and Paul Swoboda. 2021. Combinatorial optimization for panoptic segmentation: A fully differentiable approach. *Advances in Neural Information Processing Systems* 34:15635–15649.

Abdulnabi, Abrar H, Gang Wang, Jiwen Lu, and Kui Jia. 2015. Multi-task cnn model for attribute prediction. *IEEE Transactions on Multimedia* 17(11):1949–1959.

Absil, P-A, Robert Mahony, and Rodolphe Sepulchre. 2004. Riemannian geometry of grassmann manifolds with a view on algorithmic computation. *Acta Applicandae Mathematica* 80(2):199–220.

Absil, Pierre-Antoine, Robert E. Mahony, and Rodolphe Sepulchre. 2007. Optimization algorithms on matrix manifolds.

Agrawal, Akshay, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and Zico Kolter. 2019. Differentiable convex optimization layers. *arXiv preprint arXiv:1910.12430*.

Ahmad, Shair, and Antonio Ambrosetti. 2015. *A textbook on ordinary differential equations*, vol. 88. Springer.

Ahmed, Ali, Benjamin Recht, and Justin Romberg. 2013. Blind deconvolution using convex programming. *IEEE Transactions on Information Theory* 60(3):1711–1732.

rusty1s el al. 2020. pytorch-sparse. https://github.com/rusty1s/pytorch_sparse.

ALDRICH, HENRY C, and JOHN W DANIEL. 1982. Cell biology of physarum and didymium.

Allen-Zhu, Zeyuan, and Yuanzhi Li. 2016. Doubly accelerated methods for faster cca and generalized eigendecomposition. In *Icml*.

Amos, Brandon, and J Zico Kolter. 2017. Optnet: Differentiable optimization as a layer in neural networks. In *International conference on machine learning*, 136–145. PMLR.

Amos, Brandon, Lei Xu, and J Zico Kolter. 2017. Input convex neural networks. In *Proceedings of the 34th icml-volume 70*, 146–155. JMLR. org.

Andrew, Galen, Raman Arora, Jeff Bilmes, and Karen Livescu. 2013. Deep canonical correlation analysis. In *International conference on machine learning*, 1247–1255.

Anoop, Rodrigo Santa Cruz Basura Fernando, and Cherian Stephen Gould. Deeppermnet: Visual permutation learning. *learning* 33:25.

Arjovsky, Martin, Soumith Chintala, and Léon Bottou. 2017. Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

Arora, Raman, Teodor Vanislavov Marinov, Poorya Mianjy, and Nati Srebro. 2017. Stochastic approximation for canonical correlation analysis. In *Advances in neural information processing systems*, 4775–4784.

Arora, Sanjeev, Rong Ge, and Ankur Moitra. 2012. Learning topic models–going beyond svd. In *2012 ieee 53rd annual symposium on foundations of computer science*, 1–10. IEEE.

Arsham, H. 1997. Initialization of the simplex algorithm: An artificial-free approach. *SIAM Review*.

Arulkumaran, Kai, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine* 34(6):26–38.

Ataman, Kaan, W Nick Street, and Yi Zhang. 2006. Learning to rank by maximizing auc with linear programming. In *The 2006 ieee international joint conference on neural network proceedings*, 123–129. IEEE.

Atzmon, Matan, Haggai Maron, and Yaron Lipman. 2018. Point convolutional neural networks by extension operators. *arXiv preprint arXiv:1803.10091*.

Banisch, Ralf, Zofia Trstanova, Andreas Bittracher, Stefan Klus, and Péter Koltai. 2020. Diffusion maps tailored to arbitrary non-degenerate itô processes. *Applied and Computational Harmonic Analysis* 48(1):242–265.

Barvinok, Alexander. 2013. A bound for the number of vertices of a polytope with applications. *Combinatorica*.

Bécigneul, Gary, and Octavian-Eugen Ganea. 2018. Riemannian adaptive optimization methods. *arXiv preprint arXiv:1810.00760*.

Belanger, David, and Andrew McCallum. 2016. Structured prediction energy networks. In *Icml*, 983–992.

Belkin, Mikhail, and Partha Niyogi. 2003. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation* 15(6):1373–1396.

Belkin, Mikhail, Partha Niyogi, and Vikas Sindhwani. 2006. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research* 7(11).

Berthet, Quentin, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. 2020. Learning with differentiable pertubed optimizers. *Advances in neural information processing systems* 33:9508–9519.

Bertsekas, Dimitri P. 1997. Nonlinear programming. *Journal of the Operational Research Society* 48(3):334–334.

Bhatia, Kush, Aldo Pacchiano, Nicolas Flammarion, Peter L Bartlett, and Michael I Jordan. 2018. Gen-oja: Simple & efficient algorithm for streaming generalized eigenvector computation. In *Advances in neural information processing systems*, 7016–7025.

Blondel, Mathieu, Olivier Teboul, Quentin Berthet, and Josip Djolonga. 2020. Fast differentiable sorting and ranking. In *International conference on machine learning*.

Bonnabel, Silvere. 2013. Stochastic gradient descent on riemannian manifolds. *IEEE Transactions on Automatic Control* 58(9):2217–2229.

Boothby, William M. 1986. *An introduction to differentiable manifolds and riemannian geometry*. Academic press.

Borga, Magnus. 2001. Canonical correlation: a tutorial. *On line tutorial http://people. imt. liu. se/magnus/cca* 4(5).

Boumal, Nicolas, Bamdev Mishra, P-A Absil, and Rodolphe Sepulchre. 2014. Manopt, a matlab toolbox for optimization on manifolds. *The Journal of Machine Learning Research* 15(1):1455–1459.

Bousquet, Olivier, Sylvain Gelly, Ilya Tolstikhin, Carl-Johann Simon-Gabriel, and Bernhard Schoelkopf. 2017. From optimal transport to generative modeling: the vegan cookbook. *arXiv preprint arXiv:1705.07642*.

Bronstein, Michael M, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34(4):18–42.

Burges, Chris, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on machine learning*, 89–96. ACM.

Caelles, Sergi, Kevis-Kokitsi Maninis, Jordi Pont-Tuset, Laura Leal-Taixé, Daniel Cremers, and Luc Van Gool. 2017. One-shot video object segmentation. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 221–230.

Caselles, Vicent, Ron Kimmel, and Guillermo Sapiro. 1997. Geodesic active contours. *International journal of computer vision* 22(1):61–79.

Chakraborty, Rudrasis, Liu Yang, Soren Hauberg, and Baba Vemuri. 2020. Intrinsic grassmann averages for online linear, robust and nonlinear subspace learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Chan, Tony F, and Luminita A Vese. 2001. Active contours without edges. *IEEE Transactions on image processing* 10(2):266–277.

Chaudhari, Pratik, and Stefano Soatto. 2018. Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks. In *2018 information theory and applications workshop (ita)*, 1–10. IEEE.

Chaudhuri, Kamalika, Sham M Kakade, Karen Livescu, and Karthik Sridharan. 2009. Multi-view clustering via canonical correlation analysis. In *Proceedings of the 26th annual international conference on machine learning*, 129–136.

Chen, Huizhong, Andrew Gallagher, and Bernd Girod. 2012. Describing clothing by semantic attributes. In *European conference on computer vision*, 609–623. Springer.

Chen, Liang-Chieh, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. 2017. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence* 40(4):834–848.

Chen, Ricky TQ, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. 2018. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*.

Cohen, Taco S, Mario Geiger, Jonas Köhler, and Max Welling. 2018. Spherical cnns. *arXiv preprint arXiv:1801.10130*.

Coifman, Ronald R, and Stéphane Lafon. 2006. Diffusion maps. *Applied and computational harmonic analysis* 21(1):5–30.

Collins, Edo, Radhakrishna Achanta, and Sabine Susstrunk. 2018. Deep feature factorization for concept discovery. In *Proceedings of the european conference on computer vision (eccv)*, 336–352.

Conforti, Michele, Gérard Cornuéjols, Giacomo Zambelli, et al. 2014. *Integer programming*, vol. 271. Springer.

Cortes, Corinna, and Mehryar Mohri. 2004. Auc optimization vs. error rate minimization. In *Advances in neural information processing systems*, 313–320.

Couture, Heather D., Roland Kwitt, J. S. Marron, Melissa A. Troester, Charles M. Perou, and Marc Niethammer. 2019. Deep multi-view learning via task-optimal cca. *ArXiv* abs/1907.07739.

Crammer, Koby, and Yoram Singer. 2001. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of machine learning research* 2(Dec): 265–292.

Cubuk, Ekin D, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. 2019. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, 113–123.

Cui, Yiran, Keiichi Morikuni, Takashi Tsuchiya, and Ken Hayami. 2019. Implementation of interior-point methods for lp based on krylov subspace iterative solvers with inner-iteration preconditioning. *Computational Optimization and Applications*.

Cuturi, Marco, Olivier Teboul, and Jean-Philippe Vert. 2019. Differentiable ranking and sorting using optimal transport. *Advances in neural information processing systems* 32.

Dai, Jifeng, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. 2017. Deformable convolutional networks. In *Proceedings of the ieee international conference on computer vision*, 764–773.

Danskin, John M. 1966. The theory of max-min, with applications. *SIAM Journal on Applied Mathematics* 14(4):641–664.

Dantzig, GB. 1951. Gb,(1947).maximization of a linear function of variables subject to linear inequalities, 1939 also published pp. 339–347 in tc koopmans (ed.): Activity analysis of production and allocation.

Dantzig, George B, and Mukund N Thapa. 1997. The linear programming problem. *Linear Programming: 1: Introduction* 1–33.

Dave, Achal, Pavel Tokmakov, Cordelia Schmid, and Deva Ramanan. 2019. Learning to track any object. *arXiv preprint arXiv:1910.11844*.

Dembczynski, Krzysztof J., Willem Waegeman, Weiwei Cheng, and Eyke Hüllermeier. 2011. An exact algorithm for f-measure maximization. In *Advances in neural information processing systems 24*, ed. J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, 1404–1412. Curran Associates, Inc.

Donini, Michele, Luca Oneto, Shai Ben-David, John S Shawe-Taylor, and Massimiliano Pontil. 2018. Empirical risk minimization under fairness constraints. In *Advances in neural information processing systems*, 2791–2801.

Duan, Yan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. 2016. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, 1329–1338. PMLR.

Dupont, Emilien, Arnaud Doucet, and Yee Whye Teh. 2019. Augmented neural odes. *arXiv preprint arXiv:1904.01681*.

Eban, Elad, Mariano Schain, Alan Mackey, Ariel Gordon, Ryan Rifkin, and Gal Elidan. 2017. Scalable learning of non-decomposable objectives. In *Artificial intelligence and statistics*, 832–840.

Edelman, Alan, Tomás A. Arias, and Steven Thomas Smith. 1998. The geometry of algorithms with orthogonality constraints. *SIAM J. Matrix Anal. Appl.* 20:303–353.

Esteves, Carlos, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. 2017. Learning so(3) equivariant representations with spherical cnns. *CoRR*. 1711.06721.

Farhadi, Ali, Ian Endres, Derek Hoiem, and David Forsyth. 2009. Describing objects by their attributes. In *Computer vision and pattern recognition, 2009. cvpr 2009. ieee conference on*, 1778–1785. IEEE.

Fathony, Rizal, and Zico Kolter. 2020. Ap-perf: Incorporating generic performance metrics in differentiable learning. In *International conference on artificial intelligence and statistics*, 4130–4140. PMLR.

Feichtenhofer, Christoph, Axel Pinz, and Richard P Wildes. 2017. Spatiotemporal multiplier networks for video action recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 4768–4777.

Ferber, Aaron, Bryan Wilder, Bistra Dilkina, and Milind Tambe. 2020. Mipaal: Mixed integer program as a layer. In *Aaai*, 1504–1511.

Finn, Chelsea, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th icml-volume 70*, 1126–1135. JMLR. org.

flaport. 2020. torch-sparse-solve. https://github.com/flaport/torch_sparse_solve.

Fraccaro, Marco, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. 2016. Sequential neural models with stochastic layers. *Advances in neural information processing systems* 29.

Freeman, William T, and Egon C Pasztor. 2000. Markov networks for super-resolution. In *Proc. 34th annual conf. on information sciences and systems (ciss 2000)*.

Frerix, Thomas, Daniel Cremers, and Matthias Nießner. 2019. Linear inequality constraints for neural network activations. *arXiv preprint arXiv:1902.01785*.

Frieze, Alan, and Michał Karoński. 2015. *Introduction to random graphs*. Cambridge University Press.

Gao, Chao, Dan Garber, Nathan Srebro, Jialei Wang, and Weiran Wang. 2019. Stochastic canonical correlation analysis. *Journal of Machine Learning Research* 20(167):1–46.

Gao, Wei, Rong Jin, Shenghuo Zhu, and Zhi-Hua Zhou. 2013. One-pass auc optimization. In *International conference on machine learning*, 906–914.

Garcia, Victor, and Joan Bruna. 2017. Few-shot learning with graph neural networks. *arXiv preprint arXiv:1711.04043*.

Ge, Rong, Chi Jin, Praneeth Netrapalli, Aaron Sidford, et al. 2016. Efficient algorithms for large-scale generalized eigenvector computation and canonical correlation analysis. In *International conference on machine learning*, 2741–2750.

Gilmer, Justin, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*.

Glorot, Xavier, and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 249–256.

Golub, Gene H, and Christian Reinsch. 1971. Singular value decomposition and least squares solutions. In *Linear algebra*, 134–151. Springer.

Golub, Gene H., and Hongyuan Zha. 1992. The canonical correlations of matrix pairs and their numerical computation.

Golub, Gene H, and Hongyuan Zha. 1995. The canonical correlations of matrix pairs and their numerical computation. In *Linear algebra for signal processing*, 27–49. Springer.

Gondzio, Jacek. 2012. Interior point methods 25 years later. *European Journal of Operational Research*.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

Goodfellow, Ian, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. 2013. Multi-prediction deep boltzmann machines. In *Neurips*, 548–556.

Gori, Marco, Gabriele Monfardini, and Franco Scarselli. 2005. A new model for learning in graph domains. In *Neural networks, 2005. ijcnn'05. proceedings. 2005 ieee international joint conference on*, vol. 2, 729–734. IEEE.

Grady, Leo. 2008. Minimal surfaces extend shortest path segmentation methods to 3d. *IEEE TPAMI* 32(2):321–334.

Guo, Chuan, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On calibration of modern neural networks. In *Proceedings of the 34th icml-volume 70*, 1321–1330. JMLR. org.

Guo, Meng-Hao, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R Martin, and Shi-Min Hu. 2020. Pct: Point cloud transformer. *arXiv preprint arXiv:2012.09688*.

Haber, Eldad, and Lars Ruthotto. 2017. Stable architectures for deep neural networks. *Inverse problems* 34(1):014004.

Han, Hu, Anil K Jain, Shiguang Shan, and Xilin Chen. 2017. Heterogeneous face attribute estimation: A deep multi-task learning approach. *IEEE transactions on pattern analysis and machine intelligence*.

Hanley, James A, and Barbara J McNeil. 1982. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology* 143(1):29–36.

Hardt, Moritz, Eric Price, and Nati Srebro. 2016a. Equality of opportunity in supervised learning. In *Advances in neural information processing systems*, 3315–3323.

Hardt, Moritz, Ben Recht, and Yoram Singer. 2016b. Train faster, generalize better: Stability of stochastic gradient descent. In *International conference on machine learning*, 1225–1234. PMLR.

Hariharan, Bharath, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. 2015. Hypercolumns for object segmentation and fine-grained localization. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 447–456.

Hatamizadeh, Ali, Debleena Sengupta, and Demetri Terzopoulos. 2019. End-to-end deep convolutional active contours for image segmentation. *arXiv preprint arXiv:1909.13359*.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 770–778.

Helgason, Sigurdur. 2001. *Differential geometry and symmetric spaces*, vol. 341. American Mathematical Soc.

Hess, Eric J, and J Paul Brooks. 2015. The support vector machine and mixed integer linear programming: Ramp loss svm with l1-norm regularization. In *14th informs computing society conference*, 226–235.

Hirsch, Morris W. 1988. Stability and convergence in strongly monotone dynamical systems.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2(5):359–366.

Hotelling, Harold. 1992. Relations between two sets of variates. In *Breakthroughs in statistics*, 162–190. Springer.

Ishteva, Mariya, Pierre-Antoine Absil, Sabine Van Huffel, and Lieven De Lathauwer. 2011. Best low multilinear rank approximation of higher-order tensors, based on the riemannian trust-region scheme. *SIAM J. Matrix Anal. Appl.* 32:115–135.

Jamieson, Kevin G, Lalit Jain, Chris Fernandez, Nicholas J Glattard, and Rob Nowak. 2015. Next: A system for real-world development, evaluation, and application of active learning. In *Advances in neural information processing systems*, 2656–2664.

Johannson, Anders, and James Zou. 2012. A slime mold solver for linear programming problems. In *Conference on computability in europe*, 344–354. Springer.

John, Elizabeth, and E. Alper Yıldırım. 2008. Implementation of warm-start strategies inăinterior-point methods for linear programming inăfixed dimension. *Computational Optimization and Applications*.

Jordan, Michael Irwin. 1999. *Learning in graphical models*. MIT press.

Kahou, Samira Ebrahimi, Xavier Bouthillier, Pascal Lamblin, Caglar Gulcehre, Vincent Michalski, Kishore Konda, Sébastien Jean, Pierre Froumenty, Yann Dauphin, Nicolas Boulanger-Lewandowski, et al. 2016. Emonets: Multimodal deep learning approaches for emotion recognition in video. *Journal on Multimodal User Interfaces* 10(2):99–111.

Kamnitsas, Konstantinos, Christian Ledig, Virginia FJ Newcombe, Joanna P Simpson, Andrew D Kane, David K Menon, Daniel Rueckert, and Ben Glocker. 2017. Efficient multi-scale 3d cnn with fully connected crf for accurate brain lesion segmentation. *Medical image analysis* 36:61–78.

Kanchinadam, Teja, Zihang Meng, Joseph Bockhorst, Vikas Singh, and Glenn Fung. 2021. Graph neural networks to predict customer satisfaction following interactions with a corporate call center. *arXiv preprint arXiv:2102.00420*.

Kaneko, Tetsuya, Simone Fiori, and Toshihisa Tanaka. 2012. Empirical arithmetic averaging over the compact stiefel manifold. *IEEE Transactions on Signal Processing* 61(4):883–894.

Keerthi, S Sathiya, Vikas Sindhwani, and Olivier Chapelle. 2007. An efficient method for gradient-based adaptation of hyperparameters in svm models. In *Advances in neural information processing systems*, 673–680.

Kharazmi, Ehsan, Zhongqiang Zhang, and George Em Karniadakis. 2019. Variational physics-informed neural networks for solving partial differential equations. *arXiv preprint arXiv:1912.00873*.

Kim, Hyunwoo J, Nagesh Adluru, Barbara B Bendlin, Sterling C Johnson, Baba C Vemuri, and Vikas Singh. 2014. Canonical correlation analysis on riemannian manifolds and its applications. In *European conference on computer vision*, 251–267. Springer.

Kim, Jongmin, Taesup Kim, Sungwoong Kim, and Chang D Yoo. 2019. Edge-labeling graph neural network for few-shot learning. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, 11–20.

Kim, Kyongwon, Bing Li, Zhou Yu, Lexin Li, et al. 2020. On post dimension reduction statistical inference. *Annals of Statistics* 48(3):1567–1592.

Kingma, Diederik P, and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kipf, Thomas N, and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Kirillov, Alexander, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. 2019. Panoptic segmentation. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, 9404–9413.

Kleiman, Ross, and David Page. 2019. Aucμ: A performance metric for multi-class machine learning models. In *International conference on machine learning*, 3439–3447.

Klokov, Roman, and Victor Lempitsky. 2017. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *Proceedings of the ieee international conference on computer vision*, 863–872.

Koller, Daphne, and Nir Friedman. 2009. *Probabilistic graphical models: principles and techniques*. MIT press.

Komarichev, Artem, Zichun Zhong, and Jing Hua. 2019. A-cnn: Annularly convolutional neural networks on point clouds. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, 7421–7430.

Krizhevsky, Alex. 2009. Learning multiple layers of features from tiny images. Tech. Rep.

Krizhevsky, Alex, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Kuhn, Harold W, and Albert W Tucker. 2014. Nonlinear programming. In *Traces and emergence of nonlinear programming*, 247–258. Springer.

Kunita, Hiroshi. 1997. *Stochastic flows and stochastic differential equations*, vol. 24. Cambridge university press.

Le, Truc, and Ye Duan. 2018. Pointgrid: A deep network for 3d shape understanding. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 9204–9214.

LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

LeCun, Yann, Corinna Cortes, and CJ Burges. 2010. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2.

Lee, Kwonjoon, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. 2019. Meta-learning with differentiable convex optimization. In *Cvpr*, 10657–10665.

Lee, Yin Tat, and Aaron Sidford. 2014. Path finding methods for linear programming: Solving linear programs in o (vrank) iterations and faster algorithms for maximum flow. In *2014 ieee 55th annual symposium on foundations of computer science*. IEEE.

Li, Jiaxin, Ben M Chen, and Gim Hee Lee. 2018. So-net: Self-organizing network for point cloud analysis. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 9397–9406.

Li, Yujia, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*.

Li, Zongyi, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. 2020. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*.

Liao, Renjie, Marc Brockschmidt, Daniel Tarlow, Alexander L Gaunt, Raquel Urtasun, and Richard Zemel. 2018. Graph partition neural networks for semi-supervised classification. *arXiv preprint arXiv:1803.06272*.

Lillicrap, Timothy P, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Ling, Huan, Jun Gao, Amlan Kar, Wenzheng Chen, and Sanja Fidler. 2019. Fast interactive object annotation with curve-gcn. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, 5257–5266.

Liu, Changliu, Tomer Arnon, Christopher Lazarus, Clark Barrett, and Mykel J Kochenderfer. 2019a. Algorithms for verifying deep neural networks. *arXiv preprint arXiv:1903.06758*.

Liu, Mingrui, Zhuoning Yuan, Yiming Ying, and Tianbao Yang. 2019b. Stochastic auc maximization with deep neural networks. *ICLR*.

Liu, Mingrui, Xiaoxuan Zhang, Zaiyi Chen, Xiaoyu Wang, and Tianbao Yang. 2018. Fast stochastic auc maximization with $o(1/n)$-convergence rate. In *International conference on machine learning*, 3189–3197.

Liu, Xuanqing, Tesi Xiao, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh. 2019c. Neural sde: Stabilizing neural ode networks with stochastic noise. *arXiv preprint arXiv:1906.02355*.

Liu, Y, J Lee, M Park, S Kim, E Yang, SJ Hwang, and Y Yang. 2019d. Learning to propagate labels: Transductive propagation network for few-shot learning. In *7th international conference on learning representations, iclr 2019*.

Lokhande, Vishnu Suresh, Aditya Kumar Akash, Sathya N Ravi, and Vikas Singh. 2020. Fairalm: Augmented lagrangian method for training fair models with little regret. *arXiv preprint arXiv:2004.01355*.

Lu, Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. 2021. Deepxde: A deep learning library for solving differential equations. *SIAM Review* 63(1):208–228.

Luo, Yong, Dacheng Tao, Kotagiri Ramamohanarao, Chao Xu, and Yonggang Wen. 2015. Tensor canonical correlation analysis for multi-view dimension reduction. *IEEE transactions on Knowledge and Data Engineering* 27(11):3111–3124.

Ma, Zhuang, Yichao Lu, and Dean P. Foster. 2015. Finding linear structure in large datasets with scalable canonical correlation analysis. In *Icml*.

Malladi, Ravi, James A Sethian, and Baba C Vemuri. 1995. Shape modeling with front propagation: A level set approach. *IEEE transactions on pattern analysis and machine intelligence* 17(2):158–175.

Mangasarian, OL. 2004. A newton method for linear programming. *Journal of Optimization Theory and Applications* 121(1):1–18.

Mangasarian, Olvi L. 2006. Exact 1-norm support vector machines via unconstrained convex differentiable minimization. *Journal of Machine Learning Research* 7(Jul):1517–1530.

Marcos, Diego, Devis Tuia, Benjamin Kellenberger, Lisa Zhang, Min Bai, Renjie Liao, and Raquel Urtasun. 2018. Learning deep structured active contours end-to-end. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 8877–8885.

Mauro, Massimo, Hayko Riemenschneider, Alberto Signoroni, Riccardo Leonardi, and Luc Van Gool. 2014. An integer linear programming model for view selection on overlapping camera clusters. In *2014 2nd international conference on 3d vision*, vol. 1, 464–471. IEEE.

Melas-Kyriazi, Luke. 2020. The geometry of semi-supervised learning. Ph.D. thesis, Harvard University Cambridge, Massachusetts.

Mena, Gonzalo, David Belanger, Scott Linderman, and Jasper Snoek. 2018. Learning latent permutations with gumbel-sinkhorn networks. *arXiv preprint arXiv:1802.08665*.

Meng, Zihang, Nagesh Adluru, Hyunwoo J Kim, Glenn Fung, and Vikas Singh. 2018. Efficient relative attribute learning using graph neural networks. In *Proceedings of the european conference on computer vision (eccv)*, 552–567.

Meng, Zihang, Rudrasis Chakraborty, and Vikas Singh. 2021a. An online riemannian pca for stochastic canonical correlation analysis. *Advances in Neural Information Processing Systems* 34:14056–14068.

Meng, Zihang, Sathya N Ravi, and Vikas Singh. 2020. Physarum powered differentiable linear programming layers and applications. *arXiv preprint arXiv:2004.14539*.

———. 2021b. Physarum powered differentiable linear programming layers and applications. In *Proceedings of the aaai conference on artificial intelligence*, vol. 35, 8939–8949.

Meng, Zihang, Vikas Singh, and Sathya N Ravi. 2021c. Neural tmdlayer: Modeling instantaneous flow of features via sde generators. In *Proceedings of the ieee/cvf international conference on computer vision*, 11635–11644.

Meng, Zihang, David Yang, Xuefei Cao, Ashish Shah, and Ser-Nam Lim. 2021d. Object-centric unsupervised image captioning. *arXiv preprint arXiv:2112.00969*.

Meng, Zihang, Licheng Yu, Ning Zhang, Tamara L Berg, Babak Damavandi, Vikas Singh, and Amy Bearman. 2021e. Connecting what to say with where to look by modeling human attention traces. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, 12679–12688.

Mensch, Arthur, and Mathieu Blondel. 2018. Differentiable dynamic programming for structured prediction and attention. *arXiv preprint arXiv:1802.03676*.

Metz, Luke, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. 2016. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*.

Michel, Anthony, Kaining Wang, and Bo Hu. 2001. *Qualitative theory of dynamical systems*. CRC Press.

Michoski, Craig, Miloš Milosavljević, Todd Oliver, and David R Hatch. 2020. Solving differential equations using deep neural networks. *Neurocomputing* 399:193–212.

Milletari, Fausto, Nassir Navab, and Seyed-Ahmad Ahmadi. 2016. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 fourth international conference on 3d vision (3dv)*, 565–571. IEEE.

Mohapatra, Pritish, Michal Rolinek, CV Jawahar, Vladimir Kolmogorov, and M Pawan Kumar. 2018. Efficient optimization for rank-based loss functions. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 3693–3701.

Morcos, Ari, Maithra Raghu, and Samy Bengio. 2018. Insights on representational similarity in neural networks with canonical correlation. In *Advances in neural information processing systems*, 5727–5736.

Murphy, Kevin, Yair Weiss, and Michael I Jordan. 2013. Loopy belief propagation for approximate inference: An empirical study. *arXiv preprint arXiv:1301.6725*.

Nakagaki, Toshiyuki, Hiroyasu Yamada, and Ágota Tóth. 2000. Maze-solving by an amoeboid organism. *Nature* 407(6803):470–470.

Nan, Ye, Kian Ming Chai, Wee Sun Lee, and Hai Leong Chieu. 2012. Optimizing f-measure: A tale of two approaches. *arXiv preprint arXiv:1206.4625*.

Natole, Michael, Yiming Ying, and Siwei Lyu. 2018. Stochastic proximal algorithms for auc maximization. In *International conference on machine learning*, 3710–3719.

Nemirovski, Arkadi, Anatoli B. Juditsky, Guanghui Lan, and Alexander Shapiro. 2009. Robust stochastic approximation approach to stochastic programming. *SIAM J. Optimization* 19:1574–1609.

Nesterov, Yurii. 2013. *Introductory lectures on convex optimization: A basic course*, vol. 87. Springer Science & Business Media.

Nichol, Alex, Joshua Achiam, and John Schulman. 2018. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*.

Nixon, Jeremy, Mike Dusenberry, Linchuan Zhang, Ghassen Jerfel, and Dustin Tran. 2019. Measuring calibration in deep learning. *arXiv preprint arXiv:1904.01685*.

Nocedal, Jorge, and Stephen J Wright. 1999. *Numerical optimization*. Springer.

Noda, Kuniaki, Yuki Yamaguchi, Kazuhiro Nakadai, Hiroshi G Okuno, and Tetsuya Ogata. 2015. Audio-visual speech recognition using deep learning. *Applied Intelligence* 42(4):722–737.

O'Donoghue, B., E. Chu, N. Parikh, and S. Boyd. 2016. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications* 169(3):1042–1068.

———. 2019. SCS: Splitting conic solver, version 2.1.2. https://github.com/cvxgrp/scs.

Oja, Erkki. 1982. Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology* 15:267–273.

Oliva, Aude, and Antonio Torralba. 2001. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International journal of computer vision* 42(3):145–175.

Parikh, Devi, and Kristen Grauman. 2011. Relative attributes. In *Computer vision (iccv), 2011 ieee international conference on*, 503–510. IEEE.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, 8024–8035.

Pilanci, Mert, and Martin J. Wainwright. 2016. Iterative hessian sketch: Fast and accurate solution approximation for constrained least-squares. *Journal of Machine Learning Research* 17(53):1–38.

Pogančić, Marin Vlastelica, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. 2019. Differentiation of blackbox combinatorial solvers. In *International conference on learning representations*.

Pontryagin, Lev Semenovich. 1987. *Mathematical theory of optimal processes*. CRC press.

Prékopa, András. 2013. *Stochastic programming*, vol. 324. Springer Science & Business Media.

Price, W Nicholson, and I Glenn Cohen. 2019. Privacy in the age of medical big data. *Nature medicine* 25(1):37–43.

Purwins, Hendrik, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo-Yiin Chang, and Tara Sainath. 2019. Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing* 13(2):206–219.

Qi, Charles R, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 652–660.

Qi, Guo-Jun, Liheng Zhang, Feng Lin, and Xiao Wang. 2020. Learning generalized transformation equivariant representations via autoencoding transformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Rajeswaran, Aravind, Chelsea Finn, Sham M Kakade, and Sergey Levine. 2019. Meta-learning with implicit gradients. In *Advances in neural information processing systems*, 113–124.

Ramachandram, Dhanesh, and Graham W Taylor. 2017. Deep multimodal learning: A survey on recent advances and trends. *IEEE signal processing magazine* 34(6):96–108.

Ravi, Sachin, and Hugo Larochelle. 2017. Optimization as a model for few-shot learning. *ICLR 2017*.

Ravi, Sathya N, Tuan Dinh, Vishnu Suresh Lokhande, and Vikas Singh. 2019. Explicitly imposing constraints in deep networks via conditional gradients gives improved generalization and faster convergence. In *Aaai*, vol. 33, 4772–4779.

Ravi, Sathya N., Abhay Venkatesh, Glenn M. Fung, and Vikas Singh. 2020a. Optimizing nondecomposable data dependent regularizers via lagrangian reparameterization offers significant performance and efficiency gains. *AAAI* 34:5487–5494.

Ravi, Sathya N, Abhay Venkatesh, Glenn M Fung, and Vikas Singh. 2020b. Optimizing nondecomposable data dependent regularizers via lagrangian reparameterization offers significant performance and efficiency gains. In *Proceedings of the aaai conference on artificial intelligence*, vol. 34, 5487–5494.

Ravikumar, Pradeep, and John Lafferty. 2006. Quadratic programming relaxations for metric labeling and markov random field map estimation. In *Proceedings of the 23rd icml*, 737–744. ACM.

Recht, Ben, Christopher Re, Joel Tropp, and Victor Bittorf. 2012. Factoring nonnegative matrices with linear programs. In *Advances in neural information processing systems*, 1214–1222.

Reiß, Markus, Martin Wahl, et al. 2020. Nonasymptotic upper bounds for the reconstruction error of pca. *Annals of Statistics* 48(2):1098–1123.

Robinson, Stephen M. 2018. A short proof of the sticky face lemma. *Mathematical Programming* 168(1-2):5–9.

Rolfe, Jason Tyler. 2016. Discrete variational autoencoders. *arXiv preprint arXiv:1609.02200*.

Roos, Cornelis. 2006. A full-newton step o (n) infeasible interior-point algorithm for linear optimization. *SIAM Journal on Optimization* 16(4):1110–1136.

de Roos, Filip, and Philipp Hennig. 2017. Krylov subspace recycling for fast iterative least-squares in machine learning. *arXiv preprint arXiv:1706.00241*.

Rudin, Leonid I, and Stanley Osher. 1994. Total variation based image restoration with free local constraints. In *Proceedings of 1st international conference on image processing*, vol. 1, 31–35. IEEE.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams. 1985. Learning internal representations by error propagation. Tech. Rep., California Univ San Diego La Jolla Inst for Cognitive Science.

Rupnik, Jan, and John Shawe-Taylor. 2010. Multi-view canonical correlation analysis. In *Conference on data mining and data warehouses (sikdd 2010)*, 1–4.

Russakovsky, Olga, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115(3):211–252.

Sabour, Sara, Nicholas Frosst, and Geoffrey E Hinton. 2017. Dynamic routing between capsules. In *Neurips*, 3856–3866.

Salimans, Tim, Han Zhang, Alec Radford, and Dimitris Metaxas. 2018. Improving gans using optimal transport. *arXiv preprint arXiv:1803.05573*.

Sanjabi, Maziar, Jimmy Ba, Meisam Razaviyayn, and Jason D Lee. 2018. On the convergence and robustness of training gans with regularized optimal transport. In *Neurips*, 7091–7101.

Sattigeri, Prasanna, Samuel C Hoffman, Vijil Chenthamarakshan, and Kush R Varshney. 2018. Fairness gan. *arXiv preprint arXiv:1805.09910*.

Scarselli, Franco, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE transactions on neural networks* 20(1):61–80.

———. 2009. The graph neural network model. *IEEE Transactions on Neural Networks* 20(1):61–80.

Schmidt, Uwe, and Stefan Roth. 2014. Shrinkage fields for effective image restoration. In *Cvpr*, 2774–2781.

Schrijver, Alexander. 2005. On the history of combinatorial optimization (till 1960). *Handbooks in operations research and management science* 12:1–68.

Singh, Krishna Kumar, and Yong Jae Lee. 2016. End-to-end localization and ranking for relative attributes. In *European conference on computer vision*, 753–769. Springer.

Snell, Jake, Kevin Swersky, and Richard Zemel. 2017. Prototypical networks for few-shot learning. In *Neurips*, 4077–4087.

Snoek, Cees GM, Marcel Worring, Jan C Van Gemert, Jan-Mark Geusebroek, and Arnold WM Smeulders. 2006. The challenge problem for automated detection of 101 semantic concepts in multimedia. In *Proceedings of the 14th acm international conference on multimedia*, 421–430.

Souri, Yaser, Erfan Noury, and Ehsan Adeli. 2016. Deep relative attributes. In *Asian conference on computer vision*, 118–133. Springer.

Spezialetti, Riccardo, Samuele Salti, and Luigi Di Stefano. 2019. Learning an effective equivariant 3d descriptor without supervision. In *Proceedings of the ieee/cvf international conference on computer vision*, 6401–6410.

Straszak, Damian, and Nisheeth K Vishnoi. 2015. On a natural dynamics for linear programming. *arXiv preprint arXiv:1511.07020*.

Strogatz, Steven H. 2018. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC press.

Subbarao, Raghav, and Peter Meer. 2009. Nonlinear mean shift over riemannian manifolds. *International journal of computer vision* 84(1):1.

Sung, Flood, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. 2018. Learning to compare: Relation network for few-shot learning. In *Cvpr*, 1199–1208.

Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 2818–2826.

Tan, Mingkui, Ivor Wai-Hung Tsang, Li Wang, Bart Vandereycken, and Sinno Jialin Pan. 2014. Riemannian pursuit for big matrix recovery. In *Icml*.

Tavakoli, Amin, and Ali Pourmohammad. 2012. Image denoising based on compressed sensing. *International Journal of Computer Theory and Engineering* 4(2):266.

Tero, Atsushi, Ryo Kobayashi, and Toshiyuki Nakagaki. 2007. A mathematical model for adaptive transport network in path finding by true slime mold. *Journal of theoretical biology* 244(4):553–564.

Thompson, R. C. 1961. A determinantal inequality for positive definite matrices. *Canadian Mathematical Bulletin* 4(1):5762.

Thorpe, Matthew, and Yves van Gennip. 2018. Deep limits of residual neural networks. *arXiv preprint arXiv:1810.11741*.

Toshiyuki, NAKAGAKI, YAMADA Hiroyasu, and TÓTH Ágota. 2000. Maze-solving by an amoeboid organism. *Nature* 407:470.

Trigeorgis, George, Konstantinos Bousmalis, Stefanos Zafeiriou, and Bjoern Schuller. 2014. A deep semi-nmf model for learning hidden representations. In *International conference on machine learning*, 1692–1700.

Tsuda, Koji, and Gunnar Rätsch. 2004. Image reconstruction by linear programming. In *Neurips*, 57–64.

Van Kampen, Nicolaas G. 1976. Stochastic differential equations. *Physics reports* 24(3):171–228.

Veličković, Petar, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Vershynin, Roman. 2017. Four lectures on probabilistic methods for data science. *The Mathematics of Data, IAS/Park City Mathematics Series* 231–271.

Vinyals, Oriol, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. 2016. Matching networks for one shot learning. *arXiv preprint arXiv:1606.04080*.

Vlastelica, Marin, Anselm Paulus, Vít Musil, Georg Martius, and Michal Rolínek. 2019. Differentiation of blackbox combinatorial solvers. *arXiv preprint arXiv:1912.02175*.

Wang, Fang, Hu Han, Shiguang Shan, and Xilin Chen. 2017. Deep multi-task learning for joint prediction of heterogeneous face attributes. In *Automatic face & gesture recognition (fg 2017), 2017 12th ieee international conference on*, 173–179. IEEE.

Wang, Weiran, Raman Arora, Karen Livescu, and Jeff Bilmes. 2015a. On deep multi-view representation learning. In *International conference on machine learning*, 1083–1092.

Wang, Weiran, Raman Arora, Karen Livescu, and Nathan Srebro. 2015b. Stochastic optimization for deep cca via nonlinear orthogonal iterations. In *2015 53rd annual allerton conference on communication, control, and computing (allerton)*, 688–695. IEEE.

Wang, Weiran, Jialei Wang, Dan Garber, and Nati Srebro. 2016. Efficient globally convergent stochastic optimization for canonical correlation analysis. In *Advances in neural information processing systems*, 766–774.

Wang, Yue, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. 2019. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)* 38(5):1–12.

Weinan, E, Jiequn Han, and Qianxiao Li. 2019. A mean-field optimal control formulation of deep learning. *Research in the Mathematical Sciences* 6(1):1–41.

Wolsey, Laurence A. 2020. *Integer programming*. John Wiley & Sons.

Wright, Stephen J. 1997. *Primal-dual interior-point methods*, vol. 54. Siam.

Wu, Wenxuan, Zhongang Qi, and Li Fuxin. 2019. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, 9621–9630.

Wu, Zhirong, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 2015. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 1912–1920.

Xiao, Fanyi, and Yong Jae Lee. 2015. Discovering the spatial extent of relative attributes. In *Proceedings of the ieee international conference on computer vision*, 1458–1466.

Xie, Saining, Sainan Liu, Zeyu Chen, and Zhuowen Tu. 2018. Attentional shapecontextnet for point cloud recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition*, 4606–4615.

Yang, Howard Hua, and Shun-ichi Amari. 1998. The efficiency and the robustness of natural gradient descent learning rule. In *Neurips*.

Yao, Sirui, and Bert Huang. 2017. Beyond parity: Fairness objectives for collaborative filtering. In *Advances in neural information processing systems*, 2921–2930.

Yedidia, Jonathan S, William T Freeman, Yair Weiss, et al. 2003. Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium* 8(236-239):0018–9448.

Yger, Florian, Maxime Berar, Gilles Gasso, and Alain Rakotomamonjy. 2012. Adaptive canonical correlation analysis based on matrix manifolds. In *Icml*.

Yu, A., and K. Grauman. 2014. Fine-grained visual comparisons with local learning. In *Computer vision and pattern recognition (cvpr)*.

———. 2017. Semantic jitter: Dense supervision for visual comparisons via synthetic images. In *International conference on computer vision (iccv)*.

Yu, Fisher, and Vladlen Koltun. 2015. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*.

Yuan, Jialin, Chao Chen, and Li Fuxin. 2020. Deep variational instance segmentation. *NeurIPS 2020*.

Zeng, Xiaohui, Renjie Liao, Li Gu, Yuwen Xiong, Sanja Fidler, and Raquel Urtasun. 2019. Dmm-net: Differentiable mask-matching network for video object segmentation. In *Iccv*, 3929–3938.

Zhang, Brian Hu, Blake Lemoine, and Margaret Mitchell. 2018. Mitigating unwanted biases with adversarial learning. In *Proceedings of the 2018 aaai/acm conference on ai, ethics, and society*, 335–340.

Zhang, Hongyi, and Suvrit Sra. 2016. First-order methods for geodesically convex optimization. In *Conference on learning theory*.

Zhang, Xin, Qing Mai, and Hui Zou. 2020. The maximum separation subspace in sufficient dimension reduction with categorical response. *Journal of Machine Learning Research* 21(29):1–36.

Zhen, Xingjian, Zihang Meng, Rudrasis Chakraborty, and Vikas Singh. 2022. On the versatile uses of partial distance correlation in deep learning. *arXiv preprint arXiv:2207.09684*.

Zhu, Jun-Yan, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. 2016. Generative visual manipulation on the natural image manifold. In *European conference on computer vision*, 597–613. Springer.