Physarum Powered Differentiable Linear Programming Layers and Applications

Zihang Meng, [1] **Sathya N. Ravi,** [2] **Vikas Singh** [1]
[1] University of Wisconsin-Madison
[2] University of Illinois at Chicago
zihangm@cs.wisc.edu, sathya@uic.edu, vsingh@biostat.wisc.edu

Abstract

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 dynamics 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 its applicability for our use cases. 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 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 a LP, within a larger network.

1 Introduction

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, also known as a Linear Program (LP). 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, Recht, and Romberg 2013) surface reconstruction (Grady 2008), graphical models (Ravikumar and Lafferty 2006), scene/view understanding (Mauro et al. 2014), and numerous 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, Frosst, and Hinton 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, Chintala, and Bottou 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. 2020), 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.

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 in Thm. 2 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 demonstrate how these properties provide a practical and easily usable module for solving LPs.

1.1 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, Xu, and Kolter 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, Xu, and Kolter 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, Cremers, and Nießner 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. 2019) 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 our paper.

2 Why Physarum Dynamics?

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

$$\min_{x\in\mathbb{R}^n} \quad c^T x \quad \text{s.t.} \quad Ax = b, x \geq 0 \tag{1}$$

where $A \in \mathbb{R}^{m\times n}, c \in \mathbb{R}^n_{>0}, b \in \mathbb{R}^m$. In (1), c is called the *cost vector* (we explain how to deal with nonpositive c in Section 3), and the intersection of the linear equalities $Ax = b$, and inequalities $x \geq 0$ 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 (1).

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

Remark 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).*

2.2 Interior Point Algorithms: Trading Exactness for Efficiency

Asking for exact solutions of LP (1) may be a stringent requirement. An approximate solution of LP (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: (i) *Good:* IPM based algorithms can efficiently solve LP (1) in theory (Lee and Sidford 2014; Gondzio 2012); (ii) *Bad:* IPMs *need* to be started from a feasible point although there are special infeasible start IPMs (Roos 2006); and (iii) *Bad:* In practice, IPMs are faster than Simplex Method *only* when m, and n are large, e.g., millions (Cui et al. 2019).

Remark 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).*

2.3 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, Hiroyasu, and Ágota 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, Kobayashi, and Nakagaki 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 (1) that we consider here

Assumption 1 (Feasibility)**.** *The feasible set* $P := \{x : Ax = b, x \geq 0\}$ *of* (1) *is nonempty.*

For the applications considered in this paper, 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 $x \in \mathbb{R}^n$ with $x > 0$ and let $W \in \mathbb{R}^{n\times n}$ be the diagonal matrix with entries $\frac{x_i}{c_i}, i = 1, 2, ..., n$. Let $L = AWA^T$ and $p \in \mathbb{R}^m$ is the solution to the linear system $Lp = b$. Let $q = WA^Tp$. The PD for a LP (e.g., in (1)) given by (A, b, c) is defined as,

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

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

$$\dot{x} = W(A^TL^{-1}b - c). \tag{3}$$

Theorem 1 and 2 in (Straszak and Vishnoi 2015) guarantee that (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 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 (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 5, in the two applications that we consider in this paper, 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 (3) is *not* invertible. Next, we discuss how to deal with noninvertibility of L using our proposed algorithm called $\gamma-$AuxPD (in Algorithm 1).

3 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 (1):

Assumption 2 (Bounded)**.** *The feasible set* $P \subseteq [0, M]^n$*, i.e.,* $x \in P \implies x_i \leq M \; \forall \; i \in [n]$.

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:

Algorithm 1: $\gamma-$AuxPD Layer

Input: LP problem parameters A, b, c, initial point x_0, Max iteration number K, step size h, accuracy level ϵ, approximate diameter γ_P
Set $x_s \leftarrow x_0$ if x_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: $W \leftarrow \text{diag}(x_s/c)$
  Compute: $L \leftarrow AWA^T$
  Compute: $p \leftarrow L^{-1}b$ using iterative solvers
  Set: $q \leftarrow WA^T p$
  Update: $x_s \leftarrow (1-h)x_s + hq$
  Project onto $\mathbb{R}_{\geq\epsilon}$: $x_s \leftarrow \max(x_s, \epsilon)$
end
Return: x_s

Observation 1 ((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 $\hat{x} \in P$ such that $c^T\hat{x} - c^T x^* \leq \epsilon$ where x^* is the optimal solution to the LP* (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 x_2 be an extreme point that achieves the second lowest cost to LP* (1)*. Now it is easy to see that if $\gamma < \frac{\delta}{n \cdot M}$ where $\delta = c^T x_2 - c^T x^*$, then x^* 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 1 that solves a slightly perturbed version of the given LP.

Remark 5. *Note that $\gamma-$perturbation argument does not work for any P and c since LP* (1) *may be unbounded or have* no *extreme points.*

Observation 1 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 1 implies that $\gamma-$AuxPD algorithm is also efficient. Fortunately, for applications that satisfy the bounded assumption 2, our Theorem 2 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 $x_i = -y_i$ whenever $c_i < 0$, or in other words, by flipping the coordinate axis 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 $\mathbb{R}_{\geq\epsilon}$ after each iteration: this corresponds to a simple clamping operation.

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

In order to illustrate the potential of the $\gamma-$AuxPD layer (Alg. 1), 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 §5.

4.1 Bipartite Matching using Physarum Dynamics

Given two finite non-intersecting sets I, J such that $|I| = m$, $|J| = n, n \ll m$, and a cost function $C : I \times J \to \mathbb{R}$, solving a minimum cost bipartite matching problem corresponds to finding a map $f : I \to J$ such that total cost $\sum_i C(i, f(i))$ 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 (1) as,

$$\min_{(X, s_m) \geq 0} \ \text{tr}(CX^T) + \gamma \mathbf{1}_m^T s_m$$
$$\text{s.t. } X\mathbf{1}_m = 1_n,\ X^T 1_n + s_m = 1_m \qquad (4)$$

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

Remark 6. *Note that in LP* (4)*, the slack variables s_m impose the m inequalities $X^T 1_n \leq 1_m$.*

The following theorem shows that the convergence rate of $\gamma-$AuxPD applied to the bipartite matching in (4) only has a dependence which is **logarithmic** in n.

Theorem 2. *Assume we set $0 < \gamma \leq \gamma_u$ such that $1/\gamma_u = \Theta(\sqrt{m})$. Then, our $\gamma-$AuxPD (Algorithm 1) converges to an optimal solution to* (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 2, 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 C, P but *not* on any combinatorial function of P like extreme points/vertices). See Section A for details. □

Verifying Theorem 2. We construct random matching problems of size $n = 5, m = 50$ (used later in §5.1) with batch size 32, where we randomly set elements of 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 Dykstra's algorithm (used by (Zeng et al. 2019) in §5.1) (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

	$\gamma-$**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 1: Results on solving random matching problems.

(both used in §5.1). 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 1. Our $\gamma-$AuxPD algorithm achieves faster convergence and better quality solutions.

4.2 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 (1). Below, $\widetilde{K}^{[i,j]}$ stands for $K(x_i, x_j)(\alpha_{1j} - \alpha_{2j})$:

$$\begin{aligned}
&\min_{\alpha_1,\alpha_2,s,b_1,b_2,\xi} \quad \sum_{i=1}^{n} s_i + C\sum_{i=1}^{n}(\xi_i + 2z_i) \\
&\text{s.t.} \quad y_i\left(\sum_{j=1}^{n} y_j\widetilde{K}^{[i,j]} + (b_1 - b_2)\right) + \xi_i - Mz_i - l_i = 1, \\
&\sum_{j=1}^{n} y_j\widetilde{K}^{[i,j]} - s_i + p_i = 0, \quad \sum_{j=1}^{n} y_j\widetilde{K}^{[i,j]} + s_i - q_i = 0, \\
&z_i + r_i = 1, \quad \alpha_1, \alpha_2, s, b_1, b_2, \xi, z_i, l_i, p_i, q_i, r_i, \geq 0 \\
&\forall i = 1, 2, \cdots, n.
\end{aligned} \tag{5}$$

Like Thm. 2, we can show a convergence result for ℓ_1-SVM (5) (see Section B).

Verifying convergence of $\gamma-$AuxPD for ℓ_1-SVM (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 §5.2 that when used in training, $\gamma-$AuxPD achieves better performance and faster training time than CVXPY-SCS.

5 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.

5.1 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. 1.

	$\mathcal{J}_m$	$\mathcal{J}_r$	$\mathcal{J}_d$	$\mathcal{F}_m$	$\mathcal{F}_r$	$\mathcal{F}_d$
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 2: Results on Youtube-VOS train-val split. Subscripts m, r, d stand for mean, recall, and decay respectively.

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) (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 $X^T\mathbf{1}_{\boldsymbol{n}} \leq \mathbf{1}_m$ instead of $X^T\mathbf{1}_{\boldsymbol{n}} = \mathbf{1}_m$.

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. 1 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 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 2. Our solver works well and since the workflow is near identical to (Zeng 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.

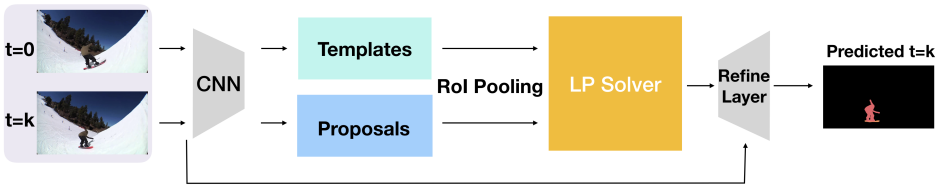


Figure 1: Architecture of DMM (Zeng et al. 2019): The yellow box is where the linear program is solved. In this application the linear program is a bipartite matching problem.

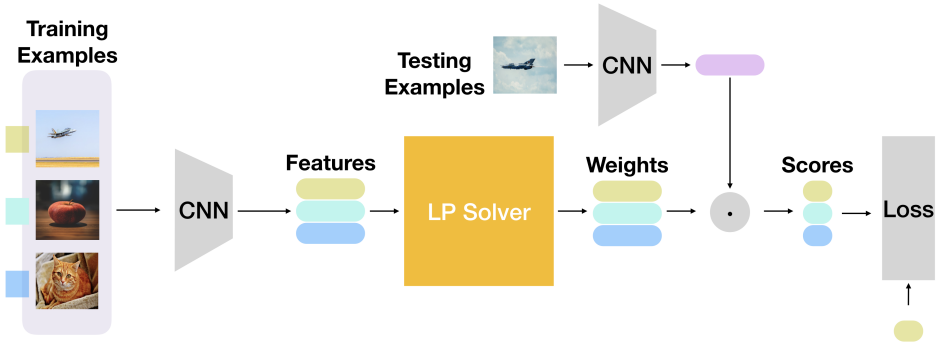


Figure 2: Architecture of Meta-learning (Lee et al. 2019): The yellow box is where the linear program is solved. In this application, the linear program is a linear SVM.

5.2 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. 2.

Problem Formulation. Given a training set $D^{train} = \{(\boldsymbol{x}_t, y_t)\}_{t=1}^{T}$, in this problem, the goal of the base learner $\mathcal{A}$ is to estimate parameters θ of the predictor $y = f(\boldsymbol{x}; \theta)$ so that it generalizes well to the unseen test set $D^{test} = \{(\boldsymbol{x}_t, y_t)\}_{t=1}^{Q}$. 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 (5). This is a 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 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 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.

LP Solver	CIFAR-FS 5-way 1-shot	CIFAR-FS 5-way 5-shot	FC100 5-way 1-shot	FC100 5-way 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 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. Performance of more baseline methods is in appendix Table 6.

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: 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 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.

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 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 5 that the quality of LP solution has a clear influence on the overall performance of the training (few-shot learning in this example).

6 Discussion

6.1 Other Potential Applications

Linear programming appears frequently in machine learning/vision, and $\gamma-$AuxPD can be potentially applied fairly directly. We cover a few recent examples which are interesting since they are not often solved as a LP.

Differentiable Calibration. Confidence calibration is important for many applications, e.g., self-driving cars (Bojarski et al. 2016) and medical diagnosis (Liang et al. 2020). However, it is known that SVMs and deep neural networks give a poor estimate of the confidence to their outputs. In general, calibration is used only as a post-procedure (Guo et al. 2017). Observe that some calibration methods can be written or relaxed in the form of a LP. For example, Isotonic regression (Guo et al. 2017), fits a piecewise non-decreasing function to transform uncalibrated outputs. By using a ℓ_1 loss, Isotonic regression can be written as a linear program. Therefore $\gamma-$AuxPD layer can solve it differentiably within an end to end network during training, which may be a desirable and lead to better calibration.

Differentiable Calculation of Wasserstein Distance (WD). WD is widely used in generative adversarial models (Arjovsky, Chintala, and Bottou 2017) as well as the analysis of shapes/point clouds (Trillos 2017). An entropy regularized LP formulation of WD can be solved using the Sinkhorn algorithm. Results in (Amari et al. 2019) suggest that Sinkhorn may be suboptimal since the limit of the sequence generated by the Sinkhorn algorithm may not coincide with the minimizer of the unregularized WD. Interestingly, we can apply Thm 2 (or Thm. 1 in (Straszak and Vishnoi 2015)) to conclude that PD (i) is asymptotically *exact*; and (ii) matches the convergence rate of the Sinkhorn algorithm. For training deep networks, this means that we can obtain unbiased gradients using $\gamma-$AuxPD layers which may lead to faster training.

Differentiable Hierarchical Clustering. Hierarchical clustering algorithms are often used in segmentation based vision tasks, see (Arbelaez et al. 2010). It is well known that an approximate hierarchical clustering can be computed by first rounding the optimal solution of a LP relaxation, see (Charikar and Chatziafratis 2017). Observe that the LP formulation of the sparsest cut problem has more constraints than decision variables owing to the ultrametric requirement of the decision variables. Hence, $\gamma-$AuxPD may be employed to approximately solve the Hierarchical Clustering problem, thus enabling us to differentiate through clustering based objective functions in end-to-end deep learning frameworks. Until recently, the EM-style clustering was a bottleneck

6.2 Implicit Differentiation of PD

In this section we show how to get implicit differentiation of c. Then A and b follow similarly. Let the updating direction

$P(x) = W(A^T L^{-1} b - c)$, where W is the diagonal matrix with entries $\frac{x_i}{c_i}$, denoted as $\text{diag}(x \oslash c)$, and $L = AWA^T$, where $\oslash$ is element-wise division. So we can rewrite P at the optimal solution x^* as

$$P(x^*) = \text{diag}(x^* \oslash c)(A^T(A \cdot \text{diag}(x^* \oslash c)A^T)^{-1} b - c) \quad (6)$$

As a joint function of c, x^*, we differentiate both the sides of (6) with respect to c as,

$$\frac{\partial P}{\partial c} + \frac{\partial P}{\partial x^*}\frac{\partial x^*}{\partial c} = 0 \implies \frac{\partial x^*}{\partial c} = -\left(\frac{\partial P}{\partial x^*}\right)^{-1}\frac{\partial P}{\partial c}. \quad (7)$$

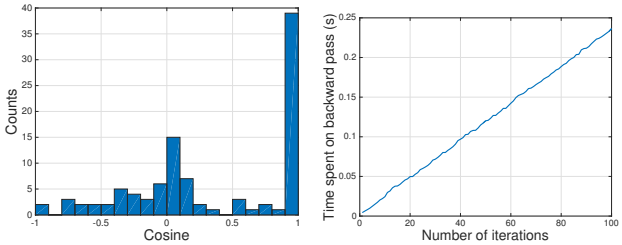
Denote $t_0 = x \oslash c$, $T_1 = (A\text{diag}(t_0)A^T)^{-1}$ and $t_2 = c \odot c$. $\frac{\partial P}{\partial x^*}$ and $\frac{\partial P}{\partial c}$ can be computed analytically as follows:

$$\frac{\partial P}{\partial x^*} = \text{diag}((A^T T_1 b - c) \oslash c) - \text{diag}(t_0)A^T T_1 A \text{diag}(b^T (A\text{diag}(t_0)A^T)^{-1} A \oslash c^T)$$

and

$$\frac{\partial P}{\partial c} = \text{diag}(t_0)A^T T_1 A \text{diag}(x^{*T} \odot (b^T (A\text{diag}(t_0)A^T)^{-1} A) \oslash t_2^T) - \text{diag}(t_0) - \text{diag}(x \odot (A^T T_1 b - c) \oslash t_2)$$

For computational purposes, without loss of generality, we can assume that the norm of the gradient to be some fixed value, say one. This is because, for training networks with PD layer using first order methods, scale or magnitude of the gradients can simply absorbed in the learning rate (interpreted as a hyperparameter) and tuned using statistical techniques such as cross validation. Hence, in order to evaluate the quality of the implicit gradient calculated from the above equations, we ignore the scale and use similarity based measures. To this end, we used our bipartite matching problem as a testbed and compared explicit gradient (calculated by unrolling the update rules) and implicit gradient (using the formula above). A high cosine value between explicit gradient and implicit gradient indicates that the two gradients are mostly in the same direction. After running 100 matching problems with different random cost matrices, we find that in all cases (as shown in Fig. 3a), the $P(x)$ becomes very small (with norm less than 0.01) which means that the quality of the final solution from our solver is good. But we find that in a fair number of cases, the cosine values are less than 0.99. We suspect that this is due to the inverse operation in (7) – note that both the terms in (7) are matrices, so after the forward pass, we have to solve n linear systems in order to compute the gradient. Indeed, this can be tricky in practice – the local geometry around the optimal solution x^* may not be ideal (for example, Hessian at x^* with a high condition number) which can then introduce floating point errors that can affect overall gradient computation significantly. Fortunately, we find that our algorithm converges in less than 10 iterations in our experiments, so it is extremely convenient to do unrolled gradient computation which tends to perform better with the overall training of the network. The above discussion also provides reasons why in Table 3, our solver performs slightly better than CVXPY and Optnet: both of which are based on implicit gradients.



(a) Histogram of the cosine values between implicit and explicit gradient on 100 random constructed matching problems.

(b) Time cost of explicit differentiation. The time cost increases linearly with the number of iterations.

Figure 3: Comparison between explicit gradient and implicit gradient.

Finally, we note that implicit differentiation may have potential benefits in certain scenarios. One benefit of implicit differentiation is that the time spent on the backward pass (gradient backpropagation) is not related to the number of iterations that our Physarum solver uses in the forward pass. From this perspective, when is implicit differentiation preferable compared with explicit differentiation (unrolling)? Consider the bipartite matching problem ($m = 10, n = 50$) as a LP example, we plot the time spent on explicit differentiation as a function of the number of iterations that our Physarum solver uses in forward pass time in Fig. 3b. Leaving aside the numerical issues discussed above, implicit differentiation costs $0.028s$, which is roughly equal to the backward pass time of explicit differentiation for $10-15$ iterations. This means that when the iterations needed in the forward pass is larger than $10 - 15$ iterations, implicit differentiation may be preferable in terms of time for the backward pass, in addition to potential memory savings an unrolled scheme would need for a large number of iterations.

7 Conclusions

This paper 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. 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

LP Solver	CIFAR-FS 5-way 1-shot	CIFAR-FS 5-way 5-shot	FC100 5-way 1-shot	FC100 5-way 5-shot
MAML (Finn, Abbeel, and Levine 2017)	58.9 ± 1.9	71.5 ± 1.0	–	–
Prototypical Networks (Snell, Swersky, and Zemel 2017)	55.5 ± 0.7	72.0 ± 0.6	35.3 ± 0.6	48.6 ± 0.6
Relation Networks (Sung et al. 2018)	55.0 ± 1.0	69.3 ± 0.8	–	–
R2D2 (Bertinetto et al. 2018)	65.3 ± 0.2	79.4 ± 0.1	–	–
TADAM (Oreshkin, López, and Lacoste 2018)	–	–	40.1 ± 0.4	56.1 ± 0.4
ProtoNets(with backbone in (Lee et al. 2019)) (Snell, Swersky, and Zemel 2017)	72.2 ± 0.7	83.5 ± 0.5	37.5 ± 0.6	52.5 ± 0.6
MetaOptNet-RR (Lee et al. 2019)	72.6 ± 0.7	84.3 ± 0.5	40.5 ± 0.6	55.3 ± 0.6
MetaOptNet-SVM (Lee et al. 2019)	72.0 ± 0.7	84.2 ± 0.5	41.1 ± 0.6	55.5 ± 0.6
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 6: More baseline results on CIFAR-FS and FC100. We achieve comparable performance using ℓ_1-SVM with (Lee et al. 2019) which uses ℓ_2-SVM and surpasses previous baseline methods. The choice between ℓ_1 and ℓ_2 often depends on the specific application considered, and ℓ_1 is often faster to solve than ℓ_2. Using the same ℓ_1-SVM, our solver achieves better performance than CVXPY-SCS and Optnet while being faster in terms of training time.

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. Our code is available at https://github.com/zihangm/Physarum-Differentiable-LP-Layer and integration with CVXPY is ongoing, which will complement functionality offered by tools like OptNet and CVXPY-SCS.

Acknowledgements

We would like to thank one of the anonymous AAAI 2021 reviewers who apart from suggestions also provided an alternative implementation that improved the performance of CVXPY-SCS in our experiments. This helped strengthen our evaluations. We thank Damian Straszak and Nisheeth Vishnoi for helpful clarifications regarding the convergence of continuous time physarum dynamics, and Yingxin Jia for interfacing our solver with a feature matching problem studied in computer vision (https://github.com/HeatherJiaZG/SuperGlue-pytorch). This research was supported in part by UW CPCP AI117924, NSF CCF #1918211, NIH R01 AG062336 and R01 AG059312, NSF CAREER award RI#1252725 and American Family Insurance. Sathya Ravi was also supported by UIC-ICR start-up funds.

Appendix

A Proof of Theorem 2

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 $\Theta(\sqrt{m})$. □

B 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

$$\|x_i\|_2 \leq 1. \tag{8}$$

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,

$$D \leq \left(K\left(x_i, x_i\right)\right)^n. \tag{9}$$

For a linear kernel, we have that, $D = \|x_i\|^n \leq 1$ (by assumption (8)). 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.

More baseline results on the meta-learning experiments are shown in Table 6.

References

Agrawal, A.; Amos, B.; Barratt, S.; Boyd, S.; Diamond, S.; and Kolter, Z. 2019. Differentiable Convex Optimization Layers. *arXiv preprint arXiv:1910.12430* .

Ahmed, A.; Recht, B.; and Romberg, J. 2013. Blind deconvolution using convex programming. *IEEE Transactions on Information Theory* 60(3): 1711–1732.

Amari, S.-i.; Karakida, R.; Oizumi, M.; and Cuturi, M. 2019. Information geometry for regularized optimal transport and barycenters of patterns. *Neural computation* 31(5): 827–848.

Amos, B.; and Kolter, J. Z. 2017. Optnet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th ICML-Volume 70*, 136–145. JMLR. org.

Amos, B.; Xu, L.; and Kolter, J. Z. 2017. Input convex neural networks. In *Proceedings of the 34th ICML-Volume 70*, 146–155. JMLR. org.

Arbelaez, P.; Maire, M.; Fowlkes, C.; and Malik, J. 2010. Contour detection and hierarchical image segmentation. *IEEE TPAMI* 33(5): 898–916.

Arjovsky, M.; Chintala, S.; and Bottou, L. 2017. Wasserstein gan. *arXiv preprint arXiv:1701.07875* .

Arsham, H. 1997. Initialization of the Simplex Algorithm: An Artificial-Free Approach. *SIAM Review* .

Barvinok, A. 2013. A bound for the number of vertices of a polytope with applications. *Combinatorica* .

Belanger, D.; and McCallum, A. 2016. Structured prediction energy networks. In *ICML*, 983–992.

Berthet, Q.; Blondel, M.; Teboul, O.; Cuturi, M.; Vert, J.-P.; and Bach, F. 2020. Learning with differentiable perturbed optimizers. *arXiv preprint arXiv:2002.08676* .

Bertinetto, L.; Henriques, J. F.; Torr, P. H.; and Vedaldi, A. 2018. Meta-learning with differentiable closed-form solvers. *arXiv preprint arXiv:1805.08136* .

Bojarski, M.; Del Testa, D.; Dworakowski, D.; Firner, B.; Flepp, B.; Goyal, P.; Jackel, L. D.; Monfort, M.; Muller, U.; Zhang, J.; et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* .

Bousquet, O.; Gelly, S.; Tolstikhin, I.; Simon-Gabriel, C.-J.; and Schoelkopf, B. 2017. From optimal transport to generative modeling: the VEGAN cookbook. *arXiv preprint arXiv:1705.07642* .

Charikar, M.; and Chatziafratis, V. 2017. Approximate hierarchical clustering via sparsest cut and spreading metrics. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, 841–854. SIAM.

Chen, T. Q.; Rubanova, Y.; Bettencourt, J.; and Duvenaud, D. K. 2018. Neural ordinary differential equations. In *NeurIPS*, 6571–6583.

Crammer, K.; and Singer, Y. 2001. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of machine learning research* 2(Dec): 265–292.

Cui, Y.; Morikuni, K.; Tsuchiya, T.; and Hayami, K. 2019. Implementation of interior-point methods for LP based on Krylov subspace iterative solvers with inner-iteration preconditioning. *Computational Optimization and Applications* doi:10.1007/s10589-019-00103-y. URL https://doi.org/10.1007/s10589-019-00103-y.

Dave, A.; Tokmakov, P.; Schmid, C.; and Ramanan, D. 2019. Learning to Track Any Object. *arXiv preprint arXiv:1910.11844* .

de Roos, F.; and Hennig, P. 2017. Krylov Subspace Recycling for Fast Iterative Least-Squares in Machine Learning. *arXiv preprint arXiv:1706.00241* .

Ferber, A.; Wilder, B.; Dilkina, B.; and Tambe, M. 2020. MIPaaL: Mixed Integer Program as a Layer. In *AAAI*, 1504–1511.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th ICML-Volume 70*, 1126–1135. JMLR. org.

Frerix, T.; Cremers, D.; and Nießner, M. 2019. Linear Inequality Constraints for Neural Network Activations. *arXiv preprint arXiv:1902.01785* .

Gondzio, J. 2012. Interior point methods 25 years later. *European Journal of Operational Research* .

Goodfellow, I.; Mirza, M.; Courville, A.; and Bengio, Y. 2013. Multi-prediction deep Boltzmann machines. In *NeurIPS*, 548–556.

Grady, L. 2008. Minimal surfaces extend shortest path segmentation methods to 3D. *IEEE TPAMI* 32(2): 321–334.

Guo, C.; Pleiss, G.; Sun, Y.; and Weinberger, K. Q. 2017. On calibration of modern neural networks. In *Proceedings of the 34th ICML-Volume 70*, 1321–1330. JMLR. org.

Hess, E. J.; and Brooks, J. P. 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.

Johannson, A.; and Zou, J. 2012. A slime mold solver for linear programming problems. In *Conference on Computability in Europe*, 344–354. Springer.

John, E.; and Yıldırım, E. A. 2008. Implementation of warm-start strategies in interior-point methods for linear programming in fixed dimension. *Computational Optimization and Applications* doi:10.1007/s10589-007-9096-y. URL https://doi.org/10.1007/s10589-007-9096-y.

Lee, K.; Maji, S.; Ravichandran, A.; and Soatto, S. 2019. Meta-learning with differentiable convex optimization. In *CVPR*, 10657–10665.

Lee, Y. T.; and Sidford, A. 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.

Liang, G.; Zhang, Y.; Wang, X.; and Jacobs, N. 2020. Improved trainable calibration method for neural networks on medical imaging classification. *arXiv preprint arXiv:2009.04057* .

Liu, C.; Arnon, T.; Lazarus, C.; Barrett, C.; and Kochenderfer, M. J. 2019. Algorithms for Verifying Deep Neural Networks. *arXiv preprint arXiv:1903.06758* .

Mauro, M.; Riemenschneider, H.; Signoroni, A.; Leonardi, R.; and Van Gool, L. 2014. An integer linear programming model for view selection on overlapping camera clusters. In *2014 2nd International Conference on 3D Vision*, volume 1, 464–471. IEEE.

Mena, G.; Belanger, D.; Linderman, S.; and Snoek, J. 2018. Learning latent permutations with gumbel-sinkhorn networks. *arXiv preprint arXiv:1802.08665* .

Mensch, A.; and Blondel, M. 2018. Differentiable dynamic programming for structured prediction and attention. *arXiv preprint arXiv:1802.03676* .

Metz, L.; Poole, B.; Pfau, D.; and Sohl-Dickstein, J. 2016. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163* .

Nixon, J.; Dusenberry, M.; Zhang, L.; Jerfel, G.; and Tran, D. 2019. Measuring calibration in deep learning. *arXiv preprint arXiv:1904.01685* .

O'Donoghue, B.; Chu, E.; Parikh, N.; and Boyd, S. 2016. Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding. *Journal of Optimization Theory and Applications* 169(3): 1042–1068. URL http://stanford.edu/~boyd/papers/scs.html.

O'Donoghue, B.; Chu, E.; Parikh, N.; and Boyd, S. 2019. SCS: Splitting Conic Solver, version 2.1.2. https://github.com/cvxgrp/scs.

Oreshkin, B.; López, P. R.; and Lacoste, A. 2018. Tadam: Task dependent adaptive metric for improved few-shot learning. In *NeurIPS*, 721–731.

Ravi, S. N.; Dinh, T.; Lokhande, V. S.; and Singh, V. 2019. Explicitly imposing constraints in deep networks via conditional gradients gives improved generalization and faster convergence. In *AAAI*, volume 33, 4772–4779.

Ravi, S. N.; Venkatesh, A.; Fung, G. M.; and Singh, V. 2020. Optimizing Nondecomposable Data Dependent Regularizers via Lagrangian Reparameterization Offers Significant Performance and Efficiency Gains. *AAAI* 34: 5487–5494. URL https://ojs.aaai.org/index.php/AAAI/article/view/5999.

Ravikumar, P.; and Lafferty, J. 2006. Quadratic programming relaxations for metric labeling and markov random field map estimation. In *Proceedings of the 23rd ICML*, 737–744. ACM.

Robinson, S. M. 2018. A short proof of the sticky face lemma. *Mathematical Programming* 168(1-2): 5–9.

Roos, C. 2006. A full-Newton step O (n) infeasible interior-point algorithm for linear optimization. *SIAM Journal on Optimization* 16(4): 1110–1136.

Sabour, S.; Frosst, N.; and Hinton, G. E. 2017. Dynamic routing between capsules. In *NeurIPS*, 3856–3866.

Salimans, T.; Zhang, H.; Radford, A.; and Metaxas, D. 2018. Improving GANs using optimal transport. *arXiv preprint arXiv:1803.05573* .

Sanjabi, M.; Ba, J.; Razaviyayn, M.; and Lee, J. D. 2018. On the convergence and robustness of training GANs with regularized optimal transport. In *NeurIPS*, 7091–7101.

Sattigeri, P.; Hoffman, S. C.; Chenthamarakshan, V.; and Varshney, K. R. 2018. Fairness gan. *arXiv preprint arXiv:1805.09910* .

Schmidt, U.; and Roth, S. 2014. Shrinkage fields for effective image restoration. In *CVPR*, 2774–2781.

Snell, J.; Swersky, K.; and Zemel, R. 2017. Prototypical networks for few-shot learning. In *NeurIPS*, 4077–4087.

Straszak, D.; and Vishnoi, N. K. 2015. On a natural dynamics for linear programming. *arXiv preprint arXiv:1511.07020* .

Sung, F.; Yang, Y.; Zhang, L.; Xiang, T.; Torr, P. H.; and Hospedales, T. M. 2018. Learning to compare: Relation network for few-shot learning. In *CVPR*, 1199–1208.

Tavakoli, A.; and Pourmohammad, A. 2012. Image denoising based on compressed sensing. *International Journal of Computer Theory and Engineering* 4(2): 266.

Tero, A.; Kobayashi, R.; and Nakagaki, T. 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): 57–62. doi:10.4153/CMB-1961-010-9.

Toshiyuki, N.; Hiroyasu, Y.; and Ágota, T. 2000. Maze-solving by an amoeboid organism. *Nature* 407: 470.

Trillos, N. G. 2017. Gromov-Hausdorff limit of Wasserstein spaces on point clouds. *arXiv preprint arXiv:1702.03464* .

Tsuda, K.; and Rätsch, G. 2004. Image reconstruction by linear programming. In *NeurIPS*, 57–64.

Vlastelica, M.; Paulus, A.; Musil, V.; Martius, G.; and Rolínek, M. 2019. Differentiation of blackbox combinatorial solvers. *arXiv preprint arXiv:1912.02175* .

Wright, S. J. 1997. *Primal-dual interior-point methods*, volume 54. Siam.

Yang, H. H.; and Amari, S.-i. 1998. The efficiency and the robustness of natural gradient descent learning rule. In *NeurIPS*.

Zeng, X.; Liao, R.; Gu, L.; Xiong, Y.; Fidler, S.; and Urtasun, R. 2019. DMM-Net: Differentiable Mask-Matching Network for Video Object Segmentation. In *ICCV*, 3929–3938.

Zhang, H.; and Sra, S. 2016. First-order methods for geodesically convex optimization. In *Conference on Learning Theory*.