Faster Predict-and-Optimize with Davis-Yin Splitting

Daniel McKenzie
Colorado School of Mines

Samy Wu Fung
Colorado School of Mines

Howard Heaton
Typal Academy

Abstract

In many applications, a combinatorial problem must be repeatedly solved with similar, but distinct parameters. Yet, the parameters w are not directly observed; only contextual data d that correlates with w is available. It is tempting to use a neural network to predict w given d, but training such a model requires reconciling the discrete nature of combinatorial optimization with the gradient-based frameworks used to train neural networks. When the problem in question is an Integer Linear Program (ILP), one approach to overcoming this issue is to consider a continuous relaxation of the combinatorial problem. While existing methods utilizing this approach have shown to be highly effective on small problems (10–100 variables), they do not scale well to large problems. In this work, we draw on ideas from modern convex optimization to design a network and training scheme which scales effortlessly to problems with thousands of variables.[1]

1 Introduction

Many high-stakes decision problems in healthcare [54], logistics and scheduling [31, 45], and transportation [51] can be viewed as a two step process. In the first step, one gathers as much as data as possible about the situation at hand. This data is used to assign a value (or cost) to the outcomes arising from each possible action. The second step is then to select the action yielding maximum value (alternatively, lowest cost). Mathematically, this can be framed as an optimization problem with a data-dependent cost function:

$$x^\star(d) \triangleq \operatorname*{argmin}_{x\in\mathcal{X}} f(x;d), \tag{1}$$

In this work, we focus on the case where $\mathcal{X} \subset \mathbb{R}^n$ is a finite constraint set and $f(x;d) = w(d)^\top x$ is a linear function. This class of problems is quite rich, containing the shortest path, traveling salesperson, and sequence alignment problems, to name a few. Given $f(\bullet;d)$, solving (1) may be straightforward (*e.g.* shortest path) or NP-hard (*e.g.* traveling salesperson problem [32]). However, our present interest is settings where the dependence of $f(\bullet;\bullet)$ on d is *unknown* and must be learned from data. We propose *learning a mapping* w_Θ to approximate the unknown objective: $w_\Theta(d) \approx w(d)$. The data d is observed and is called the *context*. As an illustrative running example, consider the shortest path prediction problem shown in Figure 1, which is studied in [10, 41].

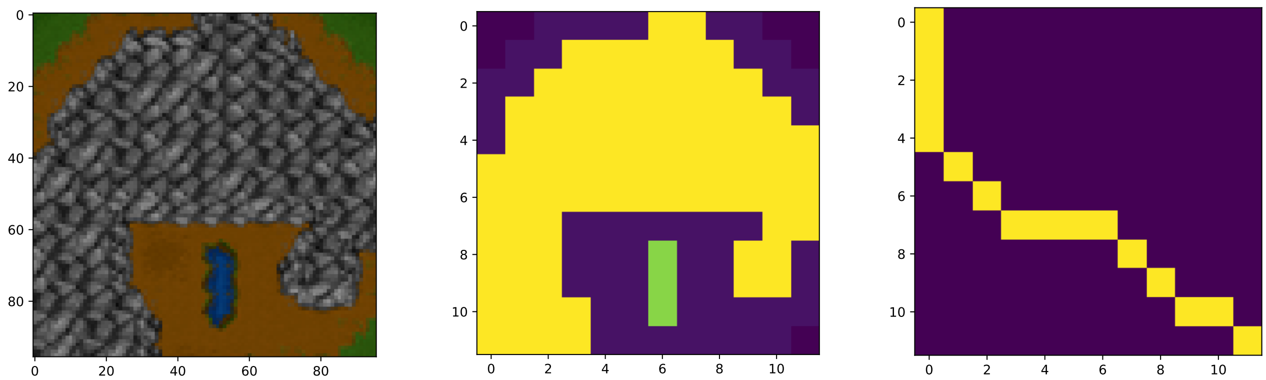
At first glance, it may appear gradient-based methods [15] are well-suited to tune the weights Θ. However, a key obstacle for such approaches is "differentiating through" the solution

$$x_\Theta(d) \triangleq \operatorname*{argmin}_{x\in\mathcal{X}} w_\Theta(d)^\top x \tag{2}$$

to obtain a gradient with which to update Θ. Specifically, the combinatorial nature of $\mathcal{X}$ may cause the solution $x_\Theta(d)$ to remain unchanged for many small perturbations to Θ; yet, for some perturbations $x_\Theta(d)$ may "jump" to a different point in $\mathcal{X}$. Hence, the gradient $\mathrm{d}x_\Theta/\mathrm{d}w_\Theta$ is always either zero or undefined [46]. To compute an informative gradient, we follow recent works (*e.g.* [52]) and relax (2) to a quadratic program over the convex hull of $\mathcal{X}$ by adding a small regularizer (see (12)).

[1] Code and additional documentation for this work are available online: fpo-dys.research.typal.academy

arXiv Preprint. July 25, 2023



(a) Map (b) Costs for Each Square (c) Shortest Path

Figure 1: The shortest path prediction problem [41]. The goal is to find the shortest path (from top-left to bottom-right) through a randomly generated terrain map from the Warcraft II tileset [28]. The contextual data d, shown in (a), is an image sub-divided into 8-by-8 squares, each representing a vertex in a 12-by-12 grid graph. The cost of traversing each square, *i.e.* $w(d)$, is shown in (b), with darker shading representing lower cost. The true shortest path is shown in (c).

Contribution Most prior works [10, 23, 35, 41, 52] focus on problems with fewer than one thousand variables. Drawing upon recent advances in convex optimization [44] and implicit neural networks [26, 30], we propose a method designed specifically for large-scale predict-and-optimize problems. Our approach is fast, easy to implement using our provided code, and, unlike several prior works (*e.g.* see [10, 41]), runs completely on GPU. Numerical examples herein demonstrate our approach, run using only standard computing resources, easily scales to problems with tens of thousands of variables. Theoretically, we verify our approach computes an informative gradient via a refined analysis of Jacobian-free Backpropagation (JFB) [26]. Along the way, we delineate two variants of the predict-and-optimize problem based upon the type of training data available, and argue that the distinction between these two variants ought to be treated with more care. In summary, we do the following.

- ▷ Building upon [29], we use Davis and Yin's three operator splitting technique [21] to propose `DYS-Net`.
- ▷ We provide, for the first time, theoretical guarantees for differentiating through the fixed point of a non-expansive, but not contractive, operator.
- ▷ We numerically show `DYS-Net` easily handles combinatorial problems with tens of thousands of variables.

2 The Predict-and-Optimize Paradigm

LP Reformulation In this work we focus on optimization problems of the form (1) where $f(x; d) = w(d)^\top x$ and $\mathcal{X}$ is the integer or binary points of a polytope, which without loss of generality we assume to be expressed in standard form [55]

$$\mathcal{X} = \mathcal{C} \cap \mathbb{Z}^n \text{ or } \mathcal{X} = \mathcal{C} \cap \{0,1\}^n \text{ where } \mathcal{C} = \{x \in \mathbb{R}^n : \ Ax = b \text{ and } x \geq 0\}\,. \tag{3}$$

In other words, (1) is an Integer Linear Program (ILP). We follow [23, 35, 52] and others in replacing the model (2) with its continuous relaxation, redefining

$$x_\Theta(d) \triangleq \operatorname*{argmin}_{x \in \mathcal{C}} w_\Theta(d)^\top x. \tag{4}$$

as a step towards making the computation of $\mathrm{d}x_\Theta / \mathrm{d}w_\Theta$ feasible, see [52] for further discussion. Henceforth, we focus exclusively on this LP reformulation.

Losses and Training Data We aim to tune weights Θ such that $x_\Theta(d) \approx x^\star(d)$. Prior works [24, 50] suggest gathering training data in the tuple form $(d, w(d))$ and then tuning weights to minimize the discrepancy[2]

[2] for example the least-square discrepancy $\|w(d) - w_\Theta(d)\|^2$

between $w(d)$ and $w_\Theta(d)$; this is referred to as the two-stage approach [47]. However, small discrepancies in the approximation $w_\Theta(d) \approx w(d)$ in areas crucial to the optimization problem (1) can yield wildly different minimizers, leading to poor generalization [9].

A better approach is to consider a loss more in line with the task at hand, for example the regret incurred by using $x_\Theta(d)$ in lieu of the true optimal solution $x^\star(d)$:

$$\text{(Regret)} = \mathcal{R}(\Theta; d, w) \triangleq w(d)^\top x_\Theta(d) - w(d)^\top x^\star(d). \tag{5}$$

As the second term is independent of weights Θ, it may be omitted from training with the regret loss

$$\text{(Regret Loss)} \equiv \mathcal{L}_R(\Theta) \triangleq \mathbb{E}_{d\sim\mathcal{D}}\left[\ell_R(\Theta; d)\right] \text{ where } \ell_R(\Theta; d) = w(d)^\top x_\Theta(d) \tag{6}$$

and $\mathcal{D}$ is the distribution of contextual data. This loss is also called the Smart Predict-then-Optimize (SPO) loss [23] or task loss [35]. Finding a model with low regret ensures the *cost* of the model output (*i.e.* $w^\top x(d)$) is close to the true optimal cost (*i.e.* $w^\top x^\star(d)$). In [23], a convex relaxation of regret is proposed, under the name SPO+

$$\text{(SPO+)} \equiv \mathcal{L}_{SPO^+}(\Theta) \triangleq \mathbb{E}_{d\sim\mathcal{D}}\left[\min_{x\in\mathcal{C}}\left\{(2w_\Theta(d) - w(d))^\top x\right\} + 2w_\Theta(d)^\top x^\star(d) - w(d)^\top x^\star(d)\right], \tag{7}$$

which is notable for the amenable form of its subgradient. In some settings [10, 30, 41] $w(d)$ is not accessible, and only training data of the form $(d, x^\star(d))$ is available. For this variant of the Predict-and-Optimize problem, an appropriate loss is one measuring the discrepancy between $x_\Theta(d)$ and $x^\star(d)$, for example

$$\text{(Argmin Loss)} \equiv \mathcal{L}_A(\Theta) \triangleq \mathbb{E}_{d\sim\mathcal{D}}\left[\ell_A(\Theta; d)\right], \quad \text{where } \ell_A(\Theta; d) = \|x^\star(d) - x_\Theta(d)\|^2. \tag{8}$$

A similar loss to $\mathcal{L}_A$ is used in [41], differing by usage of the Hamming distance between $x^\star(d)$ and $x(d)$. In principle we select the optimal weights by solving $\text{argmin}_\Theta\, \mathcal{L}(\Theta)$ where $\mathcal{L} = \mathcal{L}_R$ or $\mathcal{L} = \mathcal{L}_A$. In practice, the population risk is inaccessible, and so we minimize empirical risk instead [48]:

$$\underset{\Theta}{\text{argmin}}\ \frac{1}{N}\sum_{i=1}^{N} \ell\left(\Theta; d_i\right), \quad \text{where } \ell = \ell_R \text{ or } \ell = \ell_A. \tag{9}$$

Argmin Differentiation Omitting d from notation (for notational brevity), the gradient of regret is

$$\frac{\mathrm{d}}{\mathrm{d}\Theta}\left[\ell_R(\Theta)\right] = \frac{\mathrm{d}}{\mathrm{d}\Theta}\left[w^\top(x_\Theta - x^\star)\right] = w^\top \frac{\partial x_\Theta}{\partial w_\Theta}\frac{\mathrm{d}w_\Theta}{\mathrm{d}\Theta}, \tag{10}$$

and, for ℓ_2 error in model output,

$$\frac{\mathrm{d}}{\mathrm{d}\Theta}\left[\ell_A(\Theta)\right] = \frac{\mathrm{d}}{\mathrm{d}\Theta}\left[\|x_\Theta - x^\star\|^2\right] = (x_\Theta - x^\star)^\top \frac{\partial x_\Theta}{\partial w_\Theta}\frac{\mathrm{d}w_\Theta}{\mathrm{d}\Theta}. \tag{11}$$

As discussed in Section 1, $x^\star$ is piecewise constant as a function of w, and this remains true for the LP relaxation (4). Consequently, for all w_Θ, either $\partial x_\Theta/\partial\Theta = 0$ or $\partial x_\Theta/\partial\Theta$ is undefined—neither case yields an informative gradient. To remedy this, [35, 52] propose adding a small amount of regularization to the objective function in (4) to make the objective function strongly convex. This ensures x_Θ is a continuously differentiable function of w_Θ. Letting $f_\Theta(x; \gamma, d) \triangleq w_\Theta(d)^\top x + \gamma\|x\|_2^2$, we follow [52] by adding a small quadratic regularizer, modulated by $\gamma \geq 0$, to henceforth replace (4) by

$$x_\Theta(d) \triangleq \underset{x\in\mathcal{C}}{\text{argmin}}\ f_\Theta(x; \gamma, d). \tag{12}$$

A more principled regularizer (*e.g.* the logarithmic barrier function [35]) may be more effective, which we leave to future work. During training, we aim to solve (12) *and* simultaneously compute the derivative $\partial x_\Theta/\partial\Theta$; this problem is frequently referred to as argmin differentiation and received much attention lately [3, 4, 5, 6, 26].

3 Prior Work

The most common approach to computing $\partial x_\Theta/\partial\Theta$, proposed in [5] and used in [25, 35, 42, 52], starts with the KKT conditions for constrained optimality:

$$\frac{\partial f_\Theta}{\partial x}(x_\Theta) + A^\top\hat{\lambda} + \hat{\nu} = 0, \qquad Ax - b = 0, \qquad D(\hat{\nu})x_\Theta = 0, \tag{13}$$

where $\hat{\lambda}$ and $\hat{\nu} \geq 0$ are Lagrange multipliers associated to the optimal solution x_Θ [12] and $D(\hat{\nu})$ is a matrix with $\hat{\nu}$ along its diagonal. Differentiating these equations with respect to Θ and rearranging yields

$$\begin{bmatrix} \frac{\partial^2 f_\Theta}{\partial x^2} & A & I \\ A^\top & 0 & 0 \\ D(\hat{\nu}) & 0 & D(x_\Theta) \end{bmatrix} \begin{bmatrix} \frac{dx_\Theta}{d\Theta} \\ \frac{d\hat{\lambda}}{d\Theta} \\ \frac{d\hat{\nu}}{d\Theta} \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 f_\Theta}{\partial x \partial \Theta} \\ 0 \\ 0 \end{bmatrix}. \tag{14}$$

The matrix and right hand side vector in (14) are computable, thus enabling one to solve for $\frac{dx_\Theta}{d\Theta}$ (as well as $\frac{d\hat{\lambda}}{d\Theta}$ and $\frac{d\hat{\nu}}{d\Theta}$). The computational bottleneck in this approach is computing the Lagrange multipliers at optimality—*i.e.* $\hat{\lambda}$ and $\hat{\nu}$—in addition to x_Θ. If $x \in \mathbb{R}^n$ and $A \in \mathbb{R}^{m \times n}$ this can be done with a primal-dual interior point method at a cost of $\mathcal{O}\left(\max\{n, m\}^3\right)$ [5]. In principle it is possible to exploit sparsity in A or $\frac{\partial f_\Theta}{\partial x}(x_\Theta)$ to solve (14) faster, but in practice we observe the state-of-the-art implementation of this approach, `cvxpylayers` [3], struggles with problems containing more than 100 variables (see Section 5).

Another approach, proposed for deep equilibrium models in [6] and adapted to constrained optimization layers in [13, 17] is to re-formulate (12) as a fixed point problem:

$$\text{Find } x_\Theta \text{ such that } x_\Theta = P_\mathcal{C}\left(x_\Theta - \alpha \nabla_x f_\Theta(x_\Theta; d)\right). \tag{15}$$

Then apply the implicit function theorem to obtain an explicit formula for $\partial x_\Theta / \partial \Theta$. However, the cost of computing $P_\mathcal{C}$ can be prohibitive, see the discussion in Section 4.

Finally, many works use a perturbation-based approach to define a continuously differentiable proxy for the solution to the *unregularized* optimization problem (4), which we rewrite here as

$$g(w) = \min_{x \in \mathcal{C}} w^\top x, \tag{16}$$

omitting the dependence of w on d and Θ for notational clarity. For example, [41] define a piecewise-affine interpolant to $g(w)$. The gradients of $g_\lambda(w)$ are strikingly easy to compute, requiring just one additional solve of (16) with perturbed cost w'. We implement this approach as `BB-net` in Section 5. In [10], a stochastic perturbation is considered:

$$g_\varepsilon(w) = \mathbb{E}_Z\left[\min_{x \in \mathcal{C}} (w + \varepsilon Z)^\top x\right], \tag{17}$$

which is analogous to Nesterov-Spokoiny smoothing [36] in zeroth-order optimization. By Danskin's theorem [20], the gradients of $g_\varepsilon(w)$ are also easy to compute:

$$\nabla_w g_\varepsilon(w) = \mathbb{E}_Z\left[\operatorname*{argmin}_{x \in \mathcal{C}} (w + \varepsilon Z)^\top x\right] \approx \frac{1}{m} \sum_{i=1}^{m} \operatorname*{argmin}_{x \in \mathcal{C}} (w + \varepsilon Z_i)^\top x. \tag{18}$$

We implement this approach as `PertOpt-net` in Section 5. The advantage of such approaches is they easily wrap around existing combinatorial solvers (e.g. Dijkstra for the shortest path problem), as only repeated solves of (16) are required for computing gradients. The disadvantage is that such solvers are usually run on CPU. Thus, data needs to be shuttled between CPU and GPU when training. In addition, we observe the gradient approximations computed through such means are quite coarse, and so unsuitable for fine-grained tasks (see Section 5).

4 DYS-Net

We now introduce our proposed model, `DYS-net`. We use this term to refer to the model *and* the custom backpropagation procedure. Fixing an architecture for w_Θ, and an input d, `DYS-net` computes an approximation to $x_\Theta(d)$ in a way that is easy to backpropagate through:

$$\texttt{DYS-net}(d;\ \Theta) \approx x_\Theta \triangleq \operatorname*{argmin}_{x \in \mathcal{C}} f_\Theta(x; \gamma, d). \tag{19}$$

`DYS-net` may be trained using either the regret loss or the argmin loss.

The Forward Pass As we wish to compute x_Θ and $\partial x_\Theta/\partial\Theta$ for high dimensional settings (*i.e.* n where $x_\Theta \in \mathbb{R}^n$ and n is large), we eschew second-order methods (*e.g.* Newton's method) in favor of first-order methods such as projected gradient descent (PGD). With PGD, a sequence $\{x^k\}$ of estimates of x_Θ are computed so that

$$x_\Theta = \lim_{k\to\infty} x^k, \quad \text{where} \quad x^{k+1} = P_{\mathcal{C}}\left(x^k - \alpha\nabla_x f(x^k;\gamma,d)\right) \quad \text{for all } k\in\mathbb{N}, \tag{20}$$

where $P_{\mathcal{C}}$ is the orthogonal projection[3] onto $\mathcal{C}$. This approach works for simple sets $\mathcal{C}$ for which there exists an explicit form of $P_{\mathcal{C}}$, *e.g.* when $\mathcal{C}$ is the probability simplex [19, 22, 33]. However, for general polytopes $\mathcal{C}$ no such form exists, thereby requiring a second iterative procedure, *run at each iteration k*, to compute $P_{\mathcal{C}}(x^k)$. We adapt the architecture incorporating Davis-Yin splitting (DYS) [21] proposed in [30] to avoid computation of $P_{\mathcal{C}}$ in the forward pass. (Also, see [40, 53] where this technique is used for conventional optimization). To this end, we rewrite $\mathcal{C}$ as an intersection:

$$\mathcal{C} = \{x: \ Ax = b \text{ and } x\geq 0\} = \underbrace{\{x: \ Ax=b\}}_{\triangleq\mathcal{C}_1} \cap \underbrace{\{x: \ x\geq 0\}}_{\triangleq\mathcal{C}_2} = \mathcal{C}_1\cap\mathcal{C}_2. \tag{21}$$

While $P_{\mathcal{C}}$ is hard to compute, both $P_{\mathcal{C}_1}$ and $P_{\mathcal{C}_2}$ can be computed cheaply (once an SVD is computed for A). We verify this via the following lemma (included for completeness, as the two results are already known).

Lemma 1. *If $\mathcal{C}_1 \triangleq \{x: \ Ax=b\}, \mathcal{C}_2\triangleq\{x: \ x\geq 0\}$ and A is full-rank, then*

1. *$P_{\mathcal{C}_1}(z) = z - A^\dagger(Az-b)$, where $A^\dagger = V\Sigma^{-1}U^\top$ and $U\Sigma V^\top$ is the compact singular value decomposition of A such that U and V have orthonormal columns and Σ is invertible;*
2. *$P_{\mathcal{C}_2}(z) = \mathrm{ReLU}(z) \triangleq \max\{0,z\}$, where the max is applied element-wise.*

Further splitting of $\mathcal{C}_1$ can yield even simpler projections, see [30]. The following theorem formulates (12) as a fixed point problem involving only $P_{\mathcal{C}_1}$ and $P_{\mathcal{C}_2}$, not $P_{\mathcal{C}}$.

Theorem 2. *Let $\mathcal{C}_1, \mathcal{C}_2$ be as in (21), and suppose $f_\Theta(x;\gamma,d) = w_\Theta(d)^\top x + \frac{\gamma}{2}\|x\|_2^2$ for any neural network $w_\Theta(d)$. For all $\alpha>0$, define*

$$\begin{aligned} T_\Theta(z) &\triangleq z - P_{\mathcal{C}_2}(z) + P_{\mathcal{C}_1}\left(2\cdot P_{\mathcal{C}_2}(z) - z - \alpha\left[w_\Theta(d) + \gamma P_{\mathcal{C}_2}(z)\right]\right) & \text{(22a)}\\ &= z - P_{\mathcal{C}_2}(z) + P_{\mathcal{C}_1}\left((2-\alpha\gamma)\cdot P_{\mathcal{C}_2}(z) - z - \alpha w_\Theta(d)\right). & \text{(22b)}\end{aligned}$$

Then x_Θ solves (12) if and only if

$$x_\Theta = P_{\mathcal{C}_2}(z_\Theta), \quad \text{for some } \ z_\Theta \in \{z: z = T_\Theta(z)\}. \tag{23}$$

Proof. First note $\nabla_x f_\Theta(x_\Theta;\gamma,d) = w_\Theta(d) + \gamma x$, and so $\nabla_x f_\Theta(x_\Theta;\gamma,d)$ is γ-Lipschitz continuous. Furthermore, $\nabla_x f_\Theta$ is $1/\gamma$-cocoercive by the Baillon-Haddad theorem [7, 8]. Because $f_\Theta(x_\Theta;\gamma,d)$ is strongly convex, x_Θ is unique and is characterized by the first order optimality condition:

$$\nabla_x f_\Theta(x_\Theta;d)^\top(x - x_\Theta) \geq 0 \quad \text{for all } x\in\mathcal{C}. \tag{24}$$

The claim then follows from standard results on Davis-Yin splitting, see [30, Theorem 3.2] or [44]. ☐

The simplified expression for T_Θ given in (22b) will be useful later. The next result shows that the simple fixed point iteration method, applied with T_Θ, will converge for small enough α (see [43, Sec 2.2.1] for a proof).

Corollary 3. *With notation and assumptions as in Theorem 2, take $\alpha\in(0,2/\gamma)$, if the sequence $\{z^k\}$ is defined by $z^{k+1} = T_\Theta(z^k)$, then $x^k \triangleq P_{\mathcal{C}_2}(z^k)\to x_\Theta$ with rate $\mathcal{O}(1/k)$.*

The Backward Pass Upon attempting to differentiate both sides of the fixed-point condition (23):

$$\frac{\mathrm{d}z_\Theta}{\mathrm{d}\Theta} = \frac{\partial T_\Theta}{\partial\Theta} + \frac{\partial T_\Theta}{\partial z}\frac{\mathrm{d}z_\Theta}{\mathrm{d}\Theta} \quad\Longrightarrow\quad \mathcal{J}_\Theta(z_\Theta)\frac{\mathrm{d}z_\Theta}{\mathrm{d}\Theta} = \frac{\partial T_\Theta}{\partial\Theta}, \quad \text{where } \mathcal{J}_\Theta(z) = I - \frac{\partial T_\Theta}{\partial z}. \tag{25}$$

We notice two immediate problems: (i) T_Θ is not everywhere differentiable with respect to z, as $P_{\mathcal{C}_2}$ is not; (ii) if T_Θ were a contraction (*i.e.* Lipschitz in z with constant less than unity), then $\mathcal{J}_\Theta$ would be invertible. However,

[3] For a set $\mathcal{A}\subseteq\mathbb{R}^n$, the projection is defined by $P_{\mathcal{A}}(x)\triangleq \operatorname{argmin}_{z\in\mathcal{A}}\|z-x\|$.

this is not necessarily the case. Thus, it is not clear *a priori* that (25) can be solved for $\mathrm{d}z_\Theta/\mathrm{d}\Theta$. Our key result (Theorem 7 below) is to provide reasonable conditions under which $\mathcal{J}_\Theta(z_\Theta)$ is invertible.

Assuming these issues can be resolved, one may compute the gradient of the loss using the chain rule:

$$\frac{\mathrm{d}\ell}{\mathrm{d}\Theta} = \frac{\mathrm{d}\ell}{\mathrm{d}x}\frac{\mathrm{d}x_\Theta}{\mathrm{d}\Theta} = \frac{\mathrm{d}\ell}{\mathrm{d}x}\left(\frac{\mathrm{d}P_{\mathcal{C}_1}}{\mathrm{d}z}\frac{\mathrm{d}z_\Theta}{\mathrm{d}\Theta}\right) = \frac{\mathrm{d}\ell}{\mathrm{d}x}\frac{\mathrm{d}P_{\mathcal{C}_1}}{\mathrm{d}z}\mathcal{J}_\Theta^{-1}\frac{\partial T_\Theta}{\partial\Theta} \tag{26}$$

This approach requires solving a linear system with $\mathcal{J}_\Theta$ which becomes particularly expensive when n is large. Instead, we use the recently introduced *Jacobian-free Backpropagation* (JFB) in which the Jacobian $\mathcal{J}_\Theta$ is replaced with the identity matrix. This leads to an approximation of the true gradient $\mathrm{d}\ell/\mathrm{d}\Theta$ using

$$p_\Theta = \left[\frac{\partial\ell}{\partial x}\frac{\mathrm{d}P_{\mathcal{C}_1}}{\mathrm{d}z}\frac{\partial T_\Theta}{\partial\Theta}\right]_{(x,z)=(x_\Theta,z_\Theta)}. \tag{27}$$

We show (27) is a valid descent direction by resolving the two problems highlighted above. We begin by rigorously deriving a formula for $\partial T_\Theta/\partial z$. Recall the following generalization of the Jacobian to non-smooth operators due to Clarke [18].

Definition 4. *For any locally Lipshitz $F:\mathbb{R}^n\to\mathbb{R}^n$ let D_F denote the set upon which F is differentiable. The Clarke Jacobian of F is the set-valued function defined as*

$$\partial F(\bar{z}) = \begin{cases} \left.\frac{\mathrm{d}F}{\mathrm{d}z}\right|_{z=\bar{z}} & \text{if } \bar{z}\in D_F \\ \mathrm{Con}\left\{\lim_{z'\to\bar{z}:z'\in D_F} \left.\frac{\mathrm{d}F}{\mathrm{d}z}\right|_{z=z'}\right\} & \text{if } \bar{z}\notin D_F \end{cases} \tag{28}$$

Where Con{} *denotes the convex hull of a set.*

The Clarke Jacobian of $P_{\mathcal{C}_2}$ is easily computable, see Lemma 10. Define the (multi-valued) functions

$$c(\alpha) \triangleq \partial\max(0,\alpha) = \begin{cases} 1 & \text{if } \alpha>0 \\ 0 & \text{if } \alpha<0 \\ [0,1] & \text{if } \alpha=0 \end{cases} \quad \text{and} \quad \tilde{c}(\alpha) = \begin{cases} 1 & \text{if } \alpha>0 \\ 0 & \text{if } \alpha\le 0 \end{cases} \tag{29}$$

Then

$$\partial P_{\mathcal{C}_2}(\bar{z}) = \left[\frac{\mathrm{d}}{\mathrm{d}z}\mathrm{ReLU}(z)\right]_{z=\bar{z}} = \mathrm{diag}(c(\bar{z})), \tag{30}$$

where c is applied element-wise. If $z_i\neq 0$ for all i then $\partial P_{\mathcal{C}_2}$ is a singleton. If one or more $z_i=0$ then $\partial P_{\mathcal{C}_2}$ is multi-valued, so we choose the element of $\partial P_{\mathcal{C}_2}$ with 0 in the (i,i) position for every $z_i=0$. Abusing notation slightly, we write

$$\left.\frac{\mathrm{d}P_{\mathcal{C}_2}}{\mathrm{d}z}\right|_{z=\bar{z}} = \mathrm{diag}(\tilde{c}(\bar{z}))\in\partial P_{\mathcal{C}_2}(\bar{z})$$

This aligns with the default rule for assigning a sub-gradient to ReLU used in the popular machine learning libraries TensorFlow[1], PyTorch [39] and JAX [16], and has been observed to yield networks which are more stable to train than other choices [11].

Given the above convention, we can compute $\partial T_\Theta/\partial z$. Astonishingly, $\partial T_\Theta/\partial z$ may be expressed using only orthogonal projections to hyperplanes. Throughout, we let $e_i\in\mathbb{R}^n$ be the one-hot vector with 1 in the i-th position and zeros elsewhere, and $a_i^\top$ be the i-th row of A.

Theorem 5. *If $\mathcal{H}_1\triangleq\mathrm{Null}(A)$ with A full-rank, $\mathcal{H}_{2,z}\triangleq\mathrm{Span}\,(e_i:z_i>0)$ and $z_i\neq 0$ for all $i\in[n]$, then*

$$\left.\frac{\partial T_\Theta}{\partial z}\right|_{z=\hat{z}} = P_{\mathcal{H}_1^\perp}P_{\mathcal{H}_{2,\hat{z}}^\perp} + (1-\alpha\gamma)\cdot P_{\mathcal{H}_1}P_{\mathcal{H}_{2,\hat{z}}}, \quad \textit{for all } \hat{z}\in\mathbb{R}^n. \tag{31}$$

To show JFB is applicable, it suffices to verify $\|\partial T_\Theta/\partial z\|<1$ when evaluated at the fixed point z_Θ. The characterization in Theorem 5 enables us to show this inequality holds when x_Θ satisfies a commonly-used "niceness" condition, which we formalize as follows.

Definition 6 (LICQ condition, specialized to our case)**.** *Let x_Θ denote the solution to (12). Let $\mathcal{A}(x_\Theta)\subseteq\{1,\dots,n\}$ denote the set of active positivity constraints:*

$$\mathcal{A}(x_\Theta)\triangleq\{i:\ [x_\Theta]_i=0\}. \tag{32}$$

Algorithm 1 `DYS-Net` Training with JFB

1: **Input:** A and b defining $\mathcal{C}$, f_Θ
2: Initialize Θ^0 randomly
3: Compute SVD of A for $P_{\mathcal{C}_1}$ formula
4: **for** $m = 0, \ldots, M-1$ **do**
5:     Compute $x^K = P_{\mathcal{C}_1}(z^K) \approx x_{\Theta^m}$ using iteration $z^{k+1} = T_{\Theta^m}(z^k)$.
6:     Compute $p_\Theta(x^K) \approx p_\Theta(x_\Theta)$ using (27).
7:     $\Theta^{m+1} = \Theta^m - \eta p_{\Theta^m}(x^K)$.
8: **end for**

The point x_Θ satisfies the Linear Independence Constraint Qualification *(LICQ) condition if the vectors*

$$\{a_1, \ldots, a_m\} \cup \{e_i : i \in \mathcal{A}(x_\Theta)\} \tag{33}$$

are linearly independent.

Theorem 7. *If the LICQ condition holds at x_Θ, A is full-rank and $\alpha \in (0, 2/\gamma)$, then $\|\partial T_\Theta / \partial z\|_{z=z_\Theta} < 1$.*

The significance of Theorem 7 is outlined by the following theorem, which states use of JFB is justified with `DYS-Net` even though T_Θ is not (necessarily) a contraction.

Corollary 8. *If T_Θ is continuously differentiable with respect Θ at z_Θ, the assumptions in Theorem 7 hold and $(\partial T_\Theta/\partial\Theta)^\top(\partial T_\Theta/\partial\Theta)$ has condition number sufficiently small, then*

$$p_\Theta \triangleq \frac{\mathrm{d}}{\mathrm{d}\Theta}\left[\ell(T_\Theta(z; d))\right]_{z=z_\Theta} \tag{34}$$

is a descent direction for ℓ with respect to Θ.

Thus, using p_Θ instead of $d\ell/d\Theta$ guarantees a decrease in $\ell(\Theta)$. We summarize this training procedure as Algorithm 1. Theorem 7 also provides a sufficient condition for the application of numerous other gradient approximation techniques, for example replacing $\mathcal{J}_\Theta$ by (a truncation of) the Neumann series [27, 34]

$$\left(I - \frac{\partial T_\Theta}{\partial z}\right)^{-1} = I + \frac{\partial T_\Theta}{\partial z} + \frac{1}{2}\left(\frac{\partial T_\Theta}{\partial z}\right)^2 + \frac{1}{3!}\left(\frac{\partial T_\Theta}{\partial z}\right)^3 + \ldots. \tag{35}$$

5 Numerical Experiments

5.1 Knapsack Problem

In the (0–1, single) knapsack problem, we are presented with a container (*i.e.* a knapsack) of size c and I items, of sizes $s_1, \ldots, s_I$ and values $w_1(d), \ldots, w_I(d)$. The goal is to select the subset of maximum value that fits in the container, *i.e.* to solve:

$$x^\star = \operatorname*{argmax}_{x \in \mathcal{X}} w(d)^\top x \qquad \text{where } \mathcal{X} = \{x \in \{0,1\}^I : \ s^\top x \leq c\} \tag{36}$$

In the (0-1) k-knapsack problem we imagine the container having various notions of "size" (i.e. length, volume, weight limit) and hence a k-tuple of capacities $c \in \mathbb{R}^k$. Correspondingly, the items each have a k-tuple of sizes $s_1, \ldots, s_I \in \mathbb{R}^k$. We aim to select a subset of maximum value, amongst all subsets satisfying the k capacity constraints:

$$x^\star = \operatorname*{argmax}_{x \in \mathcal{X}} w(d)^\top x \text{ where } \mathcal{X} = \{x \in \{0,1\}^I : \ Sx \leq c\} \text{ and } S = \begin{bmatrix} s_1 & \cdots & s_k \end{bmatrix} \in \mathbb{R}^{k \times I} \tag{37}$$

In Appendix B we discuss how to transform $\mathcal{X}$ into the canonical form discussed in Section 2.

Data Generation We generate two parallel data sets using the benchmarking suite PyEOPO [47], $\mathcal{D}_w = \{(d_i, w_i \approx w(d_i)\}_{i=1}^N$ and $\mathcal{D}_x = \{(d_i, x_i^\star \approx x^\star(d_i))\}_{i=1}^N$. In both cases the d_i are sampled from a five-dimensional multivariate Gaussian distribution with mean 0 and covariance I, see Appendix C for further details. We vary I, the number of items, in increments of 5 from 20 to 60.

grid size	number of variables	network size
5-by-5	40	500
10-by-10	180	2040
20-by-20	760	8420
30-by-30	1740	19200
50-by-50	4900	53960
100-by-100	19800	217860

Table 1: Number of variables (*i.e.* number of edges) per grid size for the shortest path problem described in Section 5. Third column: number of parameters for all three models used: DYS-Net, `cvxpyayers` and PertOpt-Net. For `BB-Net`, we found a latent dimension that is 20-times larger than the aforementioned three to be more effective.

Models and Training We consider five approaches. All use the same neural network architecture $w_\Theta(d)$, thus they only differ in the way the $x^\star_\Theta$ and $\partial x_\Theta/\partial\Theta$ are computed. The four benchmarks we consider are: the Perturbed Optimization approach of [10] (`PertOpt-net`) as well as the variant proposed in [10] using the Fenchel-Young loss (`PertOpt-FY-net`); the Blackbox Backpropagation strategy of [41] (`BB-net`); and the SPO+ loss proposed by [23] (`SPO+-net`). All four are implemented using PyEPO. We train `PertOpt-net`, `BB-net`, and `DYS-net` on the $\mathcal{D}_x$ dataset using the argmin loss (8). We train the aforementioned approaches as well as `SPO+-net` and `PertOpt-FY-net` on the $\mathcal{D}_w$ dataset using the regret/ SPO loss[4] (6). Note that `SPO+-net` and `PertOpt-FY-net` are incompatible with data in the $(d_i, x_i^\star)$ format.

For w_Θ we use a three-layer fully connected neural network with leaky ReLU activation functions. We also add drop-out during training to the output layer—empirically, we find that without drop-out w_Θ tends to output a sparse approximation to w supported on a feasible set of items, and so does not generalize well. At test time, we solve (37) exactly, given $w_\Theta(d)$, using a `Gurobi`-based combinatorial solver included in PyEPO.

Training We use a validation set for model selection as we observe that, for all models, the best loss is seldom achieved at the final iteration. We train for a maximum of 25 epochs or 20 minutes, whichever comes first. We average over five trials per model and problem size (*i.e.* number of items).

Results The results are displayed in Figure 2. Given the $\mathcal{D}_w$ dataset, `SPO+-net` achieves the lowest (*i.e.* best) regret, and trains second-fastest. This corroborates the findings of [47]. However, given the $\mathcal{D}_x$ dataset, `DYS-net` appears to offer the best balance between low regret and rapid training.

5.2 Shortest Path Prediction

The shortest path between two vertices in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be found by:

$$x^\star = \underset{x\in\mathcal{X}}{\operatorname{argmin}}\, w(d)^\top x \text{ where } \mathcal{X} = \{x \in \{0,1\}^{|\mathcal{E}|} : Ex = b\} \tag{38}$$

where E is the vertex-edge adjacency matrix, b encodes the initial and terminal vertices, and $w(d) \in \mathbb{R}^{|\mathcal{E}|}$ is a vector encoding (d-dependent) edge lengths; see Appendix for further details. In this experiment we focus on the case where $\mathcal{G}$ is the $k \times k$ grid graph.

Data Generation We generate datasets $\mathcal{D} = \{(d, x^\star(d)\}$ for $k \in \{5, 10, 20, 30, 50, 100\}$ where d is sampled uniformly at random from $[0,1]^5$, the true edge weights are computed as $w(d) = Wd$ for fixed $W \in \mathbb{R}^{|\mathcal{E}|\times 5}$, and $x^\star(d)$ is computed given $w(d)$ using Dijkstra's algorithm. Further details are presented in Appendix C.

Models and Training We test four approaches: `PertOpt-net`, `BB-net`, an approach using `cvxpylayers` [2] to solve the (regularized) LP `CVX-net`, and the proposed `DYS-net`. We use the exact same neural network architecture for $w_\Theta(d)$ for `DYS-net`, `PertOpt-net`, and `Cvx-net`; a two layer fully connected neural network

[4]Except we use the SPO+ loss for `SPO+-net`

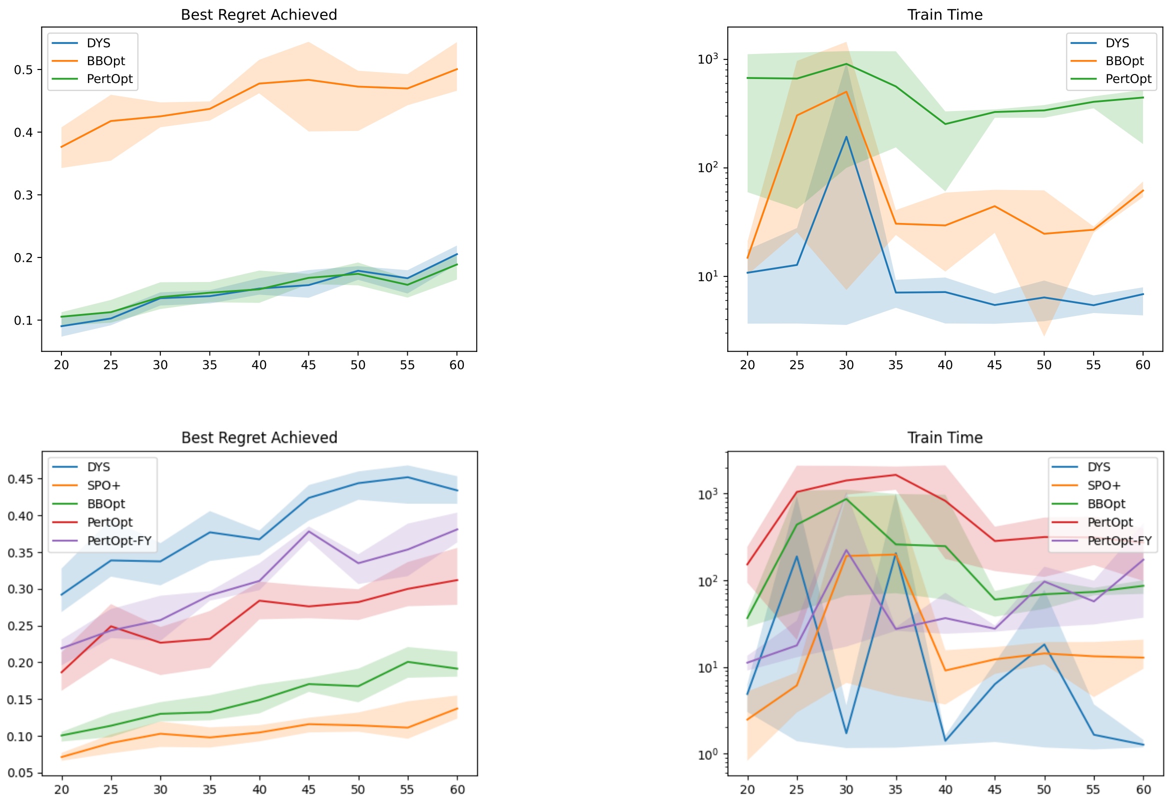


Figure 2: Results for the contextual knapsack problem. **Top Row:** Training with the $\mathcal{D}_x$ dataset. **Bottom Row:** Training with the $\mathcal{D}_w$ dataset. `DYS-net` trains (at least) one order of magnitude faster than benchmark approaches. Given the $\mathcal{D}_w$ dataset, `DYS-net` achieves a regret which compares poorly with other approaches. However, given the $\mathcal{D}_x$ dataset `DYS-net` performs well, achieving a regret only fractionally worse than that achieved by the best approach (`SPO+-net`) given the $\mathcal{D}_w$ dataset.

with leaky ReLU activation functions. For `BB-net` we use a larger network by making the latent dimension 20-times larger than that of the first three as we found this more effective. Network sizes can be seen in Table 1.

We tuned the hyperparameters for each architecture to the best of our ability on the smallest problem (5-by-5 grid graphs) and then used these hyperparameter values for all other graph sizes. We train all approaches for 100 epochs total on each problem using the argmin loss (8).

Results The results are displayed in Figure 3. While `CVX-net` and `PertOpt-net` achieve low regret for small grids, `DYS-net` model achieves a low regret *for all* grids. In addition to training faster, `DYS-net` can also be trained for much larger problems, *e.g.*, 100-by-100 grids, as shown in Figure 3. We found that `CVX-net` could not handle grids larger than 30-by-30, *i.e.* , problems with more than 1740 variables[5] (see Table 1). Importantly, `PertOpt-net` takes close to a week to train for the 100-by-100 problem, whereas

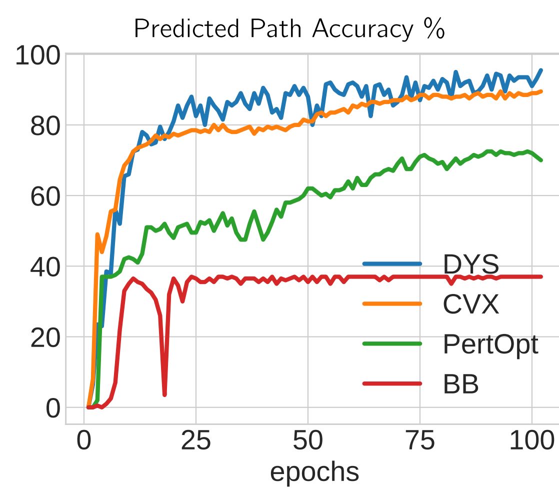


Figure 4: Accuracy (in percentage) of predicted paths on 5-by-5 grid during training.

[5]This is to be expected, as discussed in in [2, 5]

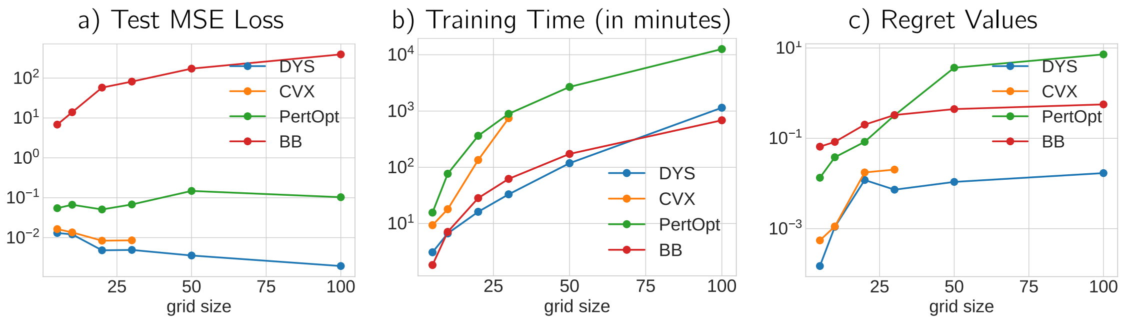


Figure 3: a) Test MSE loss (left), b) training time in minutes (middle), and c) regret values (right) vs. gridsize for three methods: DYS-Net, `cvxpylayers`[3], PertOpt [10], and Blackbox-Backprop (BB) [49]. The grid sizes are chosen to be 5-by-5, 10-by-10, 20-by-20, 30-by-30, 50-by-50, and 100-by-100. All three algorithms are shown up to gridsize 30-by-30, however, CVX is unable to load or run problems with gridsize over 30. Indeed, this is because the optimization variable x is too large. Dimensions of the variables can be found in Table 1. PertOpt [10] can be trained on the larger problems but takes a substantial amount to train.

`DYS-net` takes about a day (see right Figure 3b). On the other hand, the training speed of `BB-net` is comparable to that of `DYS-net`, but does not lead to competitive accuracy as shown in Figure 3a). Interpreting the outputs of `DYS-net` and `CVX-net` as (unnormalized) probabilities over the grid, one can use a greedy decoder to determine the most probable path from top-left to bottom-right. For small grids, *e.g.* 5-by-5, this most probable path coincides exactly with the true path for most d (see Fig. 4). For larger grids, we find there are often slight differences between the predicted and true paths. This is not surprising, as the number of possible paths grows exponentially with k.

6 Conclusions

This work presents a new method for Predict-and-Optimize capable of scaling to truly large problems. We call this approach `DYS-net`, as the core ingredient is Davis-Yin splitting. Theoretically, we show that the gradient approximation computed in the backward pass of `DYS-net` is indeed a descent direction, thus advancing the current understanding of Jacobian-free backpropagation[14, 26]. We have delineated two variants of the Predict-and-Optimize problem, distinguished by whether available data is of the form $(d, w(d))$ or $(d, x^\star(d))$, a distinction that appears to be lacking in the literature. For $(d, x^\star(d))$ data, our experiments show `DYS-Net` leads to comparable (if not lower) regret *with substantially lower training times*, as compared to state-of-the-art benchmarks. For $(d, w(d))$ data `DYS-net` performs poorly as compared to `SPO+-net`. This is not surprising as the regret/ SPO loss used in training is known to be challenging to work with [23]. Future work will focus on formulating a more performant loss function for this setting. Finally, as the dimensions of problems increase, this problem becomes more akin to using deep learning for optimal control problem [37, 38], where the aim is to find an optimal path that minimizes an energy functional. Future work may investigate these connections.

Acknowledgments

This work was partially funded by the National Science Foundation award DMS-2309810.

References

[1] Martin Abadi et al. "TensorFlow: a system for Large-Scale machine learning". In: *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 2016, pp. 265–283.

[2] A. Agrawal et al. "Differentiable Convex Optimization Layers". In: *Advances in Neural Information Processing Systems*. 2019.

[3] Akshay Agrawal et al. "Differentiable convex optimization layers". In: *Advances in neural information processing systems* 32 (2019).

[4] Akshay Agrawal et al. “Differentiating through a cone program”. In: *arXiv preprint arXiv:1904.09043* (2019).

[5] Brandon Amos and J Zico Kolter. “Optnet: Differentiable optimization as a layer in neural networks”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 136–145.

[6] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. “Deep equilibrium models”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[7] Jean-Bernard Baillon and Georges Haddad. “Quelques propriétés des opérateurs angle-bornés et n-cycliquement monotones”. In: *Israel Journal of Mathematics* 26 (1977), pp. 137–150.

[8] Heinz H Bauschke and Patrick L Combettes. “The baillon-haddad theorem revisited”. In: *arXiv preprint arXiv:0906.0807* (2009).

[9] Yoshua Bengio. “Using a financial training criterion rather than a prediction criterion”. In: *International journal of neural systems* 8.04 (1997), pp. 433–443.

[10] Quentin Berthet et al. “Learning with differentiable pertubed optimizers”. In: *Advances in neural information processing systems* 33 (2020), pp. 9508–9519.

[11] David Bertoin et al. “Numerical influence of ReLU’(0) on backpropagation”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 468–479.

[12] Dimitri P Bertsekas. “Nonlinear programming”. In: *Journal of the Operational Research Society* 48.3 (1997), pp. 334–334.

[13] Mathieu Blondel et al. “Efficient and modular implicit differentiation”. In: *arXiv preprint arXiv:2105.15183* (2021).

[14] Jérôme Bolte, Edouard Pauwels, and Samuel Vaiter. “One-step differentiation of iterative algorithms”. In: *arXiv preprint arXiv:2305.13768* (2023).

[15] Léon Bottou, Frank E Curtis, and Jorge Nocedal. “Optimization methods for large-scale machine learning”. In: *SIAM review* 60.2 (2018), pp. 223–311.

[16] James Bradbury et al. “JAX: composable transformations of Python+ NumPy programs”. In: (2018).

[17] Bingqing Chen et al. “Enforcing policy feasibility constraints through differentiable projection for energy optimization”. In: *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*. 2021, pp. 199–210.

[18] FH Clarke. “Optimization and Nonsmooth Analysis, Wiley-Interscience”. In: *New York* (1983).

[19] Laurent Condat. “Fast projection onto the simplex and the ℓ_1 ball”. In: *Mathematical Programming* 158.1 (2016), pp. 575–585.

[20] John M Danskin. “The theory of max-min, with applications”. In: *SIAM Journal on Applied Mathematics* 14.4 (1966), pp. 641–664.

[21] Damek Davis and Wotao Yin. “A three-operator splitting scheme and its optimization applications”. In: *Set-valued and variational analysis* 25.4 (2017), pp. 829–858.

[22] John Duchi et al. “Efficient projections onto the ℓ_1-ball for learning in high dimensions”. In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 272–279.

[23] Adam N Elmachtoub and Paul Grigas. “Smart “predict, then optimize””. In: *Management Science* 68.1 (2022), pp. 9–26.

[24] Fei Fang et al. “Deploying PAWS: Field Optimization of the Protection Assistant for Wildlife Security.” In: *AAAI*. Vol. 16. 2016, pp. 3966–3973.

[25] Aaron Ferber et al. “Mipaal: Mixed integer program as a layer”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 02. 2020, pp. 1504–1511.

[26] Samy Wu Fung et al. “JFB: Jacobian-free backpropagation for implicit networks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2022.

[27] Zhengyang Geng et al. “On training implicit models”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 24247–24260.

[28] Jean Guyomarch. “Warcraft ii open-source map editor, 2017”. In: *http://github. com/war2/war2edit* ().

[29] Howard Heaton and Samy Wu Fung. “Explainable AI via Learning to Optimize”. In: *Scientific Reports* (2023). URL: https://doi.org/10.1038/s41598-023-36249-3.

[30] Howard Heaton et al. "Learn to Predict Equilibria via Fixed Point Networks". In: *arXiv preprint arXiv:2106.00906* (2021).

[31] Imed Kacem, Hans Kellerer, and A Ridha Mahjoub. "Preface: New trends on combinatorial optimization for network and logistical applications". In: *Annals of Operations Research* 298.1 (2021), pp. 1–5.

[32] Richard M Karp. "Reducibility among combinatorial problems". In: *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[33] Qiuwei Li, Daniel McKenzie, and Wotao Yin. "From the simplex to the sphere: Faster constrained optimization using the Hadamard parametrization". In: *arXiv preprint arXiv:2112.05273* (2021).

[34] Renjie Liao et al. "Reviving and improving recurrent back-propagation". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 3082–3091.

[35] Jayanta Mandi and Tias Guns. "Interior Point Solving for LP-based prediction+ optimisation". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 7272–7282.

[36] Yurii Nesterov and Vladimir Spokoiny. "Random gradient-free minimization of convex functions". In: *Foundations of Computational Mathematics* 17 (2017), pp. 527–566.

[37] Derek Onken et al. "A neural network approach applied to multi-agent optimal control". In: *2021 European Control Conference (ECC)*. IEEE. 2021, pp. 1036–1041.

[38] Derek Onken et al. "A Neural Network Approach for High-Dimensional Optimal Control Applied to Multiagent Path Finding". In: *IEEE Transactions on Control Systems Technology* (2022).

[39] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019).

[40] Fabian Pedregosa and Gauthier Gidel. "Adaptive three operator splitting". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4085–4094.

[41] Marin Vlastelica Pogančić et al. "Differentiation of blackbox combinatorial solvers". In: *International Conference on Learning Representations*. 2019.

[42] Lars Ruthotto, Julianne Chung, and Matthias Chung. "Optimal experimental design for inverse problems with state constraints". In: *SIAM Journal on Scientific Computing* 40.4 (2018), B1080–B1100.

[43] Ernest Ryu and Wotao Yin. *Large-Scale Convex Optimization: Algorithm Designs via Monotone Operators*. Cambridge, England: Cambridge University Press, 2022.

[44] Ernest K Ryu and Wotao Yin. *Large-scale convex optimization via monotone operators*. 2021.

[45] Abdelkader Sbihi and Richard W Eglese. "Combinatorial optimization and green logistics". In: *Annals of Operations Research* 175.1 (2010), pp. 159–175.

[46] Peter J Stuckey et al. "Dynamic programming for predict+ optimise". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 02. 2020, pp. 1444–1451.

[47] Bo Tang and Elias B Khalil. "PyEPO: A PyTorch-based End-to-End Predict-then-Optimize Library for Linear and Integer Programming". In: *arXiv preprint arXiv:2206.14234* (2022).

[48] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999.

[49] Marin Vlastelica et al. "Differentiation of blackbox combinatorial solvers". In: *International Conference on Learning Representations*. 2019.

[50] Hao Wang et al. "COPE: Traffic engineering in dynamic networks". In: *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*. 2006, pp. 99–110.

[51] Qi Wang and Chunlei Tang. "Deep reinforcement learning for transportation network combinatorial optimization: A survey". In: *Knowledge-Based Systems* 233 (2021), p. 107526.

[52] Bryan Wilder, Bistra Dilkina, and Milind Tambe. "Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 1658–1665.

[53] Alp Yurtsever, Varun Mangalick, and Suvrit Sra. "Three operator splitting with a nonconvex loss function". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 12267–12277.

[54] Liwei Zhong and Guochun Tang. "Preface: Combinatorial optimization drives the future of Health Care". In: *Journal of Combinatorial Optimization* 42.4 (2021), pp. 675–676.

[55] Günter M Ziegler. *Lectures on polytopes*. Vol. 152. Springer Science & Business Media, 2012.

A Proofs

For the reader's convenience we restate each result given in the main text before proving it. We being with two auxiliary lemmas relating the Jacobian matrices to projections onto linear subspaces.

Lemma 9. *If $\mathcal{C}_1 \triangleq \{x : Ax = b\}$, for full-rank $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^n$ with $m < n$, and $\mathcal{H}_1 \triangleq \mathrm{Null}(A)$ then*

$$\frac{\partial P_{\mathcal{C}_1}}{\partial z} = P_{\mathcal{H}_1}, \quad \text{for all } z \in \mathbb{R}^n. \tag{39}$$

Proof. Let $A = U\Sigma V^\top$ denote the reduced SVD of A, and note that as $A \in \mathbb{R}^{m\times n}$ with $m < n$ we have $U \in \mathbb{R}^{m\times m}, \Sigma \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times m}$. Differentiating the formula for $P_{\mathcal{C}_1}$ given in Lemma 1 yields

$$\frac{\partial P_{\mathcal{C}_1}}{\partial z} = I - A^\dagger A, \tag{40}$$

where $A^\dagger \triangleq V\Sigma^{-1}U^\top$. Note

$$A^\dagger A = \left(V\Sigma^{-1}U^\top\right)\left(U\Sigma V^\top\right) = VV^\top, \tag{41}$$

which is the orthogonal projection onto $\mathrm{Range}(V) = \mathrm{Range}(A^\top)$. It follows that $I - A^\dagger A$ is the orthogonal projection on to $\mathrm{Range}(A^\top)^\perp = \mathrm{Null}(A)$. □

Lemma 10. *Define the multi-valued function*

$$c(\alpha) \triangleq \partial \max(0,\alpha) = \begin{cases} 1 & \text{if } \alpha > 0 \\ 0 & \text{if } \alpha < 0 \\ [0,1] & \text{if } \alpha = 0 \end{cases} \tag{42}$$

and, for $z \in \mathbb{R}^n$, define $\mathcal{H}_{2,z} \triangleq \mathrm{Span}(e_i : z_i > 0)$. Then

$$\partial P_{\mathcal{C}_2}(\bar{z}) = \left[\frac{\mathrm{d}}{\mathrm{d}z}\mathrm{ReLU}(z)\right]_{z=\bar{z}} = \mathrm{diag}(c(\bar{z})), \tag{43}$$

and adopting the convention for choosing an element of $\partial P_{\mathcal{C}_2}(\bar{z})$ stated in the main text:

$$\left.\frac{\mathrm{d}P_{\mathcal{C}_2}}{\mathrm{d}z}\right|_{z=\bar{z}} = \mathrm{diag}(\tilde{c}(\bar{z})) = P_{\mathcal{H}_{2,z}}. \tag{44}$$

Proof. First, suppose $z \in \mathbb{R}^n$ satisfies $z_i \neq 0$, for all $i \in [n]$, *i.e.* z is a smooth point of $P_{\mathcal{C}_2}$. Note

$$\frac{\mathrm{d}[\mathrm{ReLU}(z_i)]}{\mathrm{d}z} = 1 \text{ if } i = j \text{ and } z_i > 0 \qquad \text{and} \qquad \frac{\mathrm{d}[\mathrm{ReLU}(z_i)]}{\mathrm{d}z} = 0 \text{ if } i \neq j \text{ or } z_i < 0. \tag{45}$$

Thus, the Jacobian matrix is diagonal with a 1 in the (i,i)-th position whenever $z_i > 0$ and 0 otherwise, *i.e.* $\left.\frac{\mathrm{d}P_{\mathcal{C}_2}}{\mathrm{d}z}\right|_{z=\bar{z}} = \mathrm{diag}(c(\bar{z}))$. Now suppose $z_i = 0$ for one i. For all $\bar{z} \in \mathbb{R}^n$ with $z_i < 0$, the Jacobian $\left.\frac{\mathrm{d}P_{\mathcal{C}_2}}{\mathrm{d}z}\right|_{z=\bar{z}}$ is well-defined and has a 0 in the (i,i)-th position, while for $\bar{z} \in \mathbb{R}^n$ with $z_i > 0$, the Jacobian $\left.\frac{\mathrm{d}P_{\mathcal{C}_2}}{\mathrm{d}z}\right|_{z=\bar{z}}$ is well-defined and has a 1 in the (i,i)-th position. Taking the convex hull yields the interval $[0,1]$ in the (i,i)-th position, as claimed. The case where $z_i = 0$ for multiple i is similar.

Consequently, the product of $\left.\frac{\mathrm{d}P_{\mathcal{C}_2}}{\mathrm{d}z}\right|_{z=\bar{z}}$ and any vector $v \in \mathbb{R}^n$ equals v if and only if $v \in \mathrm{Span}(e_i : z_i > 0)$. This shows the linear operator is idempotent with fixed point set $\mathcal{H}_{2,z}$, *i.e.* it is the projection operator $P_{\mathcal{H}_{2,z}}$. □

Theorem 5. *If $\mathcal{H}_1 \triangleq \mathrm{Null}(A)$ with A full-rank, $\mathcal{H}_{2,z} \triangleq \mathrm{Span}\left(e_i : z_i > 0\right)$ and $z_i \neq 0$ for all $i \in [n]$, then*

$$\left.\frac{\partial T_\Theta}{\partial z}\right|_{z=\hat{z}} = P_{\mathcal{H}_1^\perp}P_{\mathcal{H}_{2,\hat{z}}^\perp} + (1-\alpha\gamma)\cdot P_{\mathcal{H}_1}P_{\mathcal{H}_{2,\hat{z}}}, \quad \text{for all } \hat{z} \in \mathbb{R}^n. \tag{46}$$

Proof. Differentiating the expression for T_Θ in (22b) with respect to z yields

$$\left.\frac{\partial T_\Theta}{\partial z}\right|_{z=\hat{z}} = I - \left.\frac{\mathrm{d}P_{\mathcal{C}_2}}{\mathrm{d}z}\right|_{z=\hat{z}} + \left.\frac{\mathrm{d}P_{\mathcal{C}_1}}{\mathrm{d}z}\right|_{z=y(\hat{z})}\left[(2-\alpha\gamma)\cdot\left.\frac{\mathrm{d}P_{\mathcal{C}_2}}{\mathrm{d}z}\right|_{z=\hat{z}} - I\right] \tag{47a}$$

$$= I - P_{\mathcal{H}_{2,\hat{z}}} + P_{\mathcal{H}_1}\left((2-\alpha\gamma)P_{\mathcal{H}_{2,\hat{z}}} - I\right), \quad \text{for all } \hat{z} \in \mathbb{R}^n, \tag{47b}$$

where, for notational brevity, we set $y(\hat{z}) \triangleq (2-\alpha\gamma)\cdot P_{\mathcal{C}_2}(\hat{z}) - \hat{z} - \alpha w_\ominus(d)$ in the first line and the second line follows from Lemmas 9 and 10. Repeatedly using the fact, for any subspace $\mathcal{H} \subset \mathbb{R}^n$, $P_{\mathcal{H}^\perp} = I - P_{\mathcal{H}}$, the derivative $\partial T_\ominus/\partial z$ can be further rewritten:

$$\left.\frac{\partial T_\ominus}{\partial z}\right|_{z=\hat{z}} = I - P_{\mathcal{H}_{2,\hat{z}}} + (2-\alpha\gamma)\cdot P_{\mathcal{H}_1}P_{\mathcal{H}_{2,\hat{z}}} - P_{\mathcal{H}_1} \tag{48a}$$

$$= P_{\mathcal{H}_{2,\hat{z}}^\perp} + (2-\alpha\gamma)\cdot P_{\mathcal{H}_1}\left(I - P_{\mathcal{H}_{2,\hat{z}}^\perp}\right) - P_{\mathcal{H}_1} \tag{48b}$$

$$= P_{\mathcal{H}_{2,\hat{z}}^\perp} + P_{\mathcal{H}_1} + (1-\alpha\gamma)\cdot P_{\mathcal{H}_1} - P_{\mathcal{H}_1}P_{\mathcal{H}_{2,\hat{z}}^\perp} - (1-\alpha\gamma)\cdot P_{\mathcal{H}_1}P_{\mathcal{H}_{2,\hat{z}}^\perp} - P_{\mathcal{H}_1} \tag{48c}$$

$$= (I - P_{\mathcal{H}_1})P_{\mathcal{H}_{2,\hat{z}}^\top} + (1-\alpha\gamma)\cdot P_{\mathcal{H}_1}(I - P_{\mathcal{H}_{2,\hat{z}}^\perp}) \tag{48d}$$

$$= P_{\mathcal{H}_1^\perp}P_{\mathcal{H}_{2,\hat{z}}^\perp} + (1-\alpha\gamma)\cdot P_{\mathcal{H}_1}P_{\mathcal{H}_{2,\hat{z}}}, \quad \text{for all } \hat{z} \in \mathbb{R}^n, \tag{48e}$$

completing the proof. □

We use the following lemma to prove Theorem 7.

Lemma 11. *If the LICQ condition holds at $x_\ominus$, then $\mathcal{H}_1^\perp \cap \mathcal{H}_{2,z_\ominus}^\perp = \{0\}$.*

Proof. We first rewrite $\mathcal{H}_1^\perp$ and $\mathcal{H}_{2,z_\ominus}^\perp$. The subspace $\mathcal{H}_{2,z_\ominus}^\perp$ is spanned by all non-positive coordinates of $z_\ominus$. By (23), $[x_\ominus]_i = \max\{0, [z_\ominus]_i\}$, and so $i \in \mathcal{A}(x_\ominus)$ if and only if $[z_\ominus]_i \le 0$. It follows that

$$\mathcal{H}_{2,z_\ominus}^\perp \triangleq \mathrm{Span}\{e_i : [z_\ominus]_i \le 0\} = \mathrm{Span}\{e_i : i \in \mathcal{A}(x_\ominus)\} = \mathrm{Span}\{e_{i_1}, \ldots, e_{i_\ell}\}, \tag{49}$$

where we enumerate $\mathcal{A}(x_\ominus)$ via $\mathcal{A}(x_\ominus) = \{i_1, \ldots, i_\ell\}$. On the other hand, $\mathcal{H}_1^\perp = \mathrm{Range}(A^\top) = \mathrm{Span}(a_1, \ldots, a_m)$ where $a_i^\top$ denotes the i-th row of A.

Let $v \in \mathcal{H}_1^\perp \cap \mathcal{H}_{2,z_\ominus}^\perp$ be given. Since $v \in \mathcal{H}_1^\perp$, there are scalars $\alpha_1, \ldots, \alpha_\ell$ such that $v = \alpha_1 e_{i_1} + \cdots + \alpha_\ell e_{i_\ell}$. Similarly, since $v \in \mathcal{H}_{2,z_\ominus}^\perp$, there are scalars $\beta_1, \ldots, \beta_m$ such that $v = \beta_1 a_1 + \cdots + \beta_m a_m$. Hence

$$0 = v - v = \left(\alpha_1 e_{i_1} + \ldots + \alpha_\ell e_{i_\ell}\right) - \left(\beta_1 a_1 + \ldots + \beta_m a_m\right). \tag{50}$$

By the LICQ condition, $\{e_{i_1}, \ldots, e_{i_\ell}\} \cup \{a_1, \ldots, a_m\}$ is a linearly independent set of vectors; hence $\alpha_1 = \ldots = \alpha_\ell = \beta_1 = \ldots = \beta_m = 0$ and, thus, $v = 0$. Since v was arbitrarily chosen in $\mathcal{H}_1^\perp \cap \mathcal{H}_{2,z_\ominus}^\perp$, the result follows. □

Theorem 7. *If the LICQ condition holds at $x_\ominus$, A is full-rank and $\alpha \in (0, 2/\gamma)$, then $\|\partial T_\ominus/\partial z\|_{z=z_\ominus} < 1$.*

Proof. By Lemma 11, the LICQ condition implies $\mathcal{H}_1^\perp \cap \mathcal{H}_{2,z_\ominus}^\perp = \{0\}$. This implies that either (i) the first principal angle τ between these two subspaces is nonzero, and so the cosine of this angle is less than unity, *i.e.*

$$1 > \cos(\tau) \triangleq \max_{u \in \mathcal{H}_1^\perp : \|u\|=1} \ \max_{v \in \mathcal{H}_{2,z}^\perp : \|v\|=1} \langle u, v\rangle, \tag{51}$$

or (ii) (at least) one of $\mathcal{H}_1^\perp, \mathcal{H}_{2,z_\ominus}^\perp$ is the trivial vector space $\{0\}$. In either case, let $w \in \mathbb{R}^n$ be given. By Theorem 5, in case (ii)

$$\left[\frac{\partial T_\ominus}{\partial z} w\right]_{z=z_\ominus} = P_{\mathcal{H}_1^\perp}P_{\mathcal{H}_{2,z_\ominus}^\perp} w + (1-\alpha\gamma)\cdot P_{\mathcal{H}_1}P_{\mathcal{H}_{2,z_\ominus}} w = (1-\alpha\gamma)\cdot P_{\mathcal{H}_1}P_{\mathcal{H}_{2,z_\ominus}} w \tag{52}$$

implying that

$$\left\|\frac{\partial T_\ominus}{\partial z} w\right\|_{z=z_\ominus} = (1-\alpha\gamma)\left\|P_{\mathcal{H}_1}P_{\mathcal{H}_{2,z_\ominus}} w\right\| \le (1-\alpha\gamma)\|w\|, \tag{53}$$

where the inequality follows as projection operators are firmly nonexpansive. In case (i), write $w = w_1 + w_2$, where $w_1 \in \mathcal{H}_{2,z_\ominus}$ and $w_2 \in \mathcal{H}_{2,z_\ominus}^\perp$. Appealing to Theorem 5 again

$$\left[\frac{\partial T_\ominus}{\partial z} w\right]_{z=z_\ominus} = P_{\mathcal{H}_1^\perp}P_{\mathcal{H}_{2,z_\ominus}^\perp} w + (1-\alpha\gamma)\cdot P_{\mathcal{H}_1}P_{\mathcal{H}_{2,z_\ominus}} w = P_{\mathcal{H}_1^\perp} w_2 + (1-\alpha\gamma)\cdot P_{\mathcal{H}_1} w_1. \tag{54}$$

Pythagoras' theorem may be applied to deduce, together with the fact $P_{\mathcal{H}_1^\perp} w_2$ and $P_{\mathcal{H}_1} w_1$ are orthogonal,

$$\left\|\frac{\partial T_\ominus}{\partial z} w\right\|_{z=z_\ominus}^2 = \left\|P_{\mathcal{H}_1^\perp} w_2\right\|^2 + (1-\alpha\gamma)^2\cdot\|P_{\mathcal{H}_1} w_1\|^2. \tag{55}$$

Since $w_2 \in \mathcal{H}_{2,z_\Theta}^\perp$, the angle condition (51) implies

$$\left\| P_{\mathcal{H}_1^\perp} w_2 \right\|^2 = \langle P_{\mathcal{H}_1^\perp} w_2, P_{\mathcal{H}_1^\perp} w_2 \rangle = \langle w_2, P_{\mathcal{H}_1^\perp} P_{\mathcal{H}_1^\perp} w_2 \rangle = \langle w_2, P_{\mathcal{H}_1^\perp} w_2 \rangle \leq \cos(\tau) \cdot \|w_2\|^2, \tag{56}$$

where the third equality holds since orthogonal linear projections are symmetric and idempotent. Because projections are non-expansive and $P_{\mathcal{H}_{2,z_\Theta}}$ is linear,

$$\left\| P_{\mathcal{H}_{2,z_\Theta}} w_1 \right\|^2 = \left\| P_{\mathcal{H}_{2,z_\Theta}} w_1 - P_{\mathcal{H}_{2,z_\Theta}} 0 \right\|^2 \leq \|w_1 - 0\|^2 = \|w_1\|^2. \tag{57}$$

Combining (55), (56) and (57) reveals

$$\begin{aligned} \left\| \frac{\partial T_\Theta}{\partial z} w \right\|^2_{z=z_\Theta} &\leq \cos(\tau) \cdot \|w_2\|^2 + (1-\alpha\gamma)^2 \|w_1\|^2 && \text{(58a)} \\ &\leq \max\{\cos(\tau), (1-\alpha\gamma)^2\} \cdot \left(\|w_1\|^2 + \|w_2\|^2\right) && \text{(58b)} \\ &= \max\{\cos(\tau), (1-\alpha\gamma)^2\} \cdot \|w\|^2, && \text{(58c)} \end{aligned}$$

noting the final equality holds since w_1 and w_2 are orthogonal. Because (58) holds for arbitrarily chosen $w \in \mathbb{R}^n$,

$$\left\| \frac{\partial T_\Theta}{\partial z} \right\|_{z=z_\Theta} \triangleq \sup \left\{ \left\| \frac{\partial T_\Theta}{\partial z} w \right\|_{z=z_\Theta} : \|w\| = 1 \right\} \leq \sqrt{\max\{\cos(\tau), (1-\alpha\gamma)^2\}} < 1, \tag{59}$$

where the final inequality holds by (51) and the fact $\alpha \in (0, 2/\gamma)$ implies $1 - \alpha\gamma \in (-1, 1)$, as desired. □

Corollary 8. *If T_Θ is continuously differentiable with respect Θ at z_Θ, the assumptions in Theorem 7 hold and $(\partial T_\Theta / \partial \Theta)^\top (\partial T_\Theta / \partial \Theta)$ has condition number sufficiently small, then*

$$p_\Theta \triangleq \frac{\mathrm{d}}{\mathrm{d}\Theta} \left[\ell(T_\Theta(z; d) \right]_{z=z_\Theta} \tag{60}$$

is a descent direction for ℓ with respect to Θ.

Proof. From the proof of Theorem 7 we see that T_Θ is contractive with constant $\Gamma = \sqrt{\max\{\cos(\tau), (1-\alpha\gamma)^2\}}$ and so the main theorem of [26], guaranteeing p_Θ is a descent direction, as long as the condition number of $(\partial T_\Theta / \partial \Theta)^\top (\partial T_\Theta / \partial \Theta)$ is less than $1/\Gamma$. □

Remark 12. *Similar guarantees, albeit with less restrictive assumptions on $\partial T_\Theta / \partial \Theta$, can be deduced from the results of the recent work [14].*

B Derivation for Canonical Form of Knapsack Problem

For completeness, we explain how to transform the k-knapsack problem into the canonical form (12), and how to derive the standardized representation of the constraint polytope $\mathcal{C}$. Recall that the k-knapsack problem, as originally stated, is

$$x^\star = \underset{x \in \mathcal{X}}{\operatorname{argmax}}\, w^\top x \text{ where } \mathcal{X} = \{x \in \{0,1\}^\ell : \ Sx \leq c\} \text{ and } S = \begin{bmatrix} s_1 & \cdots & s_\ell \end{bmatrix} \in \mathbb{R}^{k \times \ell} \tag{61}$$

We introduce slack variables $y_1, \ldots, y_k$ so that the inequality constraint $Sx \leq c$ becomes

$$\begin{aligned} -Sx + c \geq 0 &\implies -Sx + c = y \text{ and } y \geq 0 \\ &\implies \begin{bmatrix} S & I_k \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = c \end{aligned}$$

We relax the binary constraint $x_i \in \{0,1\}$ to $0 \leq x_i \leq 1$. We add additional slack variables $z_1, \ldots, z_\ell$ to account for the upper bound:

$$1 - x_i \geq 0 \implies 1 - x_i = z_i \text{ and } z_i \geq 0 \implies \begin{bmatrix} I_{\ell \times \ell} & 0_{\ell \times k} & I_{\ell \times \ell} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{1} \tag{62}$$

Putting this together, define

$$A = \begin{bmatrix} -S & -I_{k\times k} & 0_{k\times \ell} \\ I_{\ell\times\ell} & 0_{\ell\times k} & I_{\ell\times\ell} \end{bmatrix} \in \mathbb{R}^{(k+\ell)\times(2\ell+k)} \text{ and } b = \begin{bmatrix} -c \\ \mathbf{1}_\ell \end{bmatrix} \in \mathbb{R}^{k+\ell} \tag{63}$$

Finally, redefine $x = \begin{bmatrix} x & y & z \end{bmatrix}^\top$ and $w = \begin{bmatrix} -w & \mathbf{0}_k & \mathbf{0}_\ell \end{bmatrix}$ (where we're using $-w$ to switch the argmax to an argmin) and obtain:

$$x^\star = \underset{x\in\mathrm{Conv}(\mathcal{X})}{\mathrm{argmin}}\ w^\top(d)x + \gamma\|x\|_2^2 \text{ where } \mathrm{Conv}(\mathcal{X}) = \{x : Ax = b \text{ and } x \geq 0\} \tag{64}$$

C Experimental Details

C.1 Additional Data Details for Knapsack Problem

As mentioned, we use PyEPO [47] to generate the training data. Specifically, $d \in \mathbb{R}^5$ is sampled from the multivariate Gaussian distribution with mean 0 and covariance I. Then, $B \in \mathbb{R}^{n\times 5}$ is sampled where each $B_{ij} = +1$ with probability 0.5 and -1 with probability 0.5. The associated cost vector $w(d)$ is computed as

$$[w(d)]_i = \left[\frac{1}{3.5^{\mathrm{deg}}}\left(\frac{1}{\sqrt{5}}(Bd)_i + 3\right)^{\mathrm{deg}} + 1\right]\cdot \epsilon_{ij}$$

where $\mathrm{deg} = 4$ and ϵ_{ij} is sampled uniformly from the interval $[0.5, 1.5]$.

C.2 Additional Training Details for Knapsack Problem

For all models we use an initial learning rate of 10^{-3} and a scheduler that reduces the learning rate whenever the validation loss plateaus. We also used weight decay with a parameter of 5×10^{-4}. All networks were trained on a MacBook Pro with Apple M2 Max Chip and 32 GB of (combined) memory.

C.3 Additional Model Details for Shortest Path

Our implementation of `PertOpt-net` used a PyTorch implementation[6] of the original TensorFlow code[7] associated to the paper [10]. We train `PertOpt-net` using the argmin loss (see (11)), also referred to as MSE loss in the text. We do so for consistency with the other two models tested. We experimented with various hyperparameter settings for 5-by-5 grids and found setting the number of samples equal to 3, the temperature (*i.e.* ε) to 1 and using Gumbel noise to work best, so we used these values for all other experiments.

C.4 Additional Training Details for Shortest Path

To train DYS-net and `cvxpylayers`, we use an initial learning rate of 10^{-2} and use a scheduler that reduces whenever the loss plateaus - we found this to perform the best for these two models. For PertOpt-net, however, we found that using a fixed learning rate of 10^{-2} performed the best. For `BB-net`, we performed a logarithmic grid-search on the learning rate between $10{-}1$ to 10^{-4} and found that 10^{-3} performed best - we also attempted adaptive learning rate schemes such as reducing learning rates on plateau but did not obtain improved performance. All networks were trained using a AMD Threadripper Pro 3955WX: 16 cores, 3.90 GHz, 64 MB cache, PCIe 4.0 CPU and an NVIDIA RTX A6000 GPU.

D Additional Experimental Results

In Figure 5, we show the test loss and training time per epoch for all three architectures: `DYS-net`, `CVX-net`, and `PertOpt-net` for 10-by-10, 20-by-20, and 30-by-30 grids. In terms of MSE loss, `CVX-net` and `DYS-net` lead to comparable performance. In the second row of Figure 5, we observe the benefits of combining the three-operator splitting with JFB [26]; in particular, DYS-Net trains much faster. Figure 6 shows some randomly selected outputs for the three architectures once fully trained.

[6] See code at github.com/tuero/perturbations-differential-pytorch

[7] See code at github.com/google-research/google-research/tree/master/perturbations

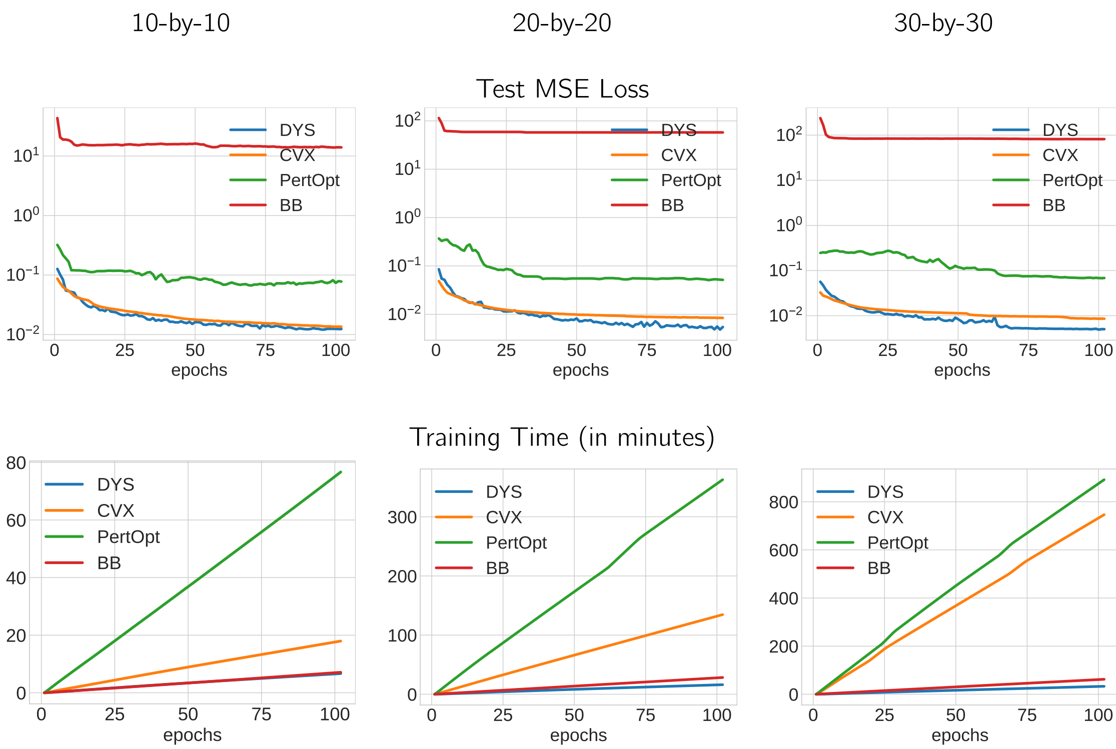


Figure 5: Comparison of of DYS-Net, `cvxpylayers` [2], PertOptNet [10], and Blackbox Backpropagation-net (BB-Net) [41] for three different grid sizes: 10×10 (first column), 20×20 (second column), and 30×30 (third column). The first row shows the MSE loss vs. epochs of the testing dataset. The second row shows the training time vs. epochs.

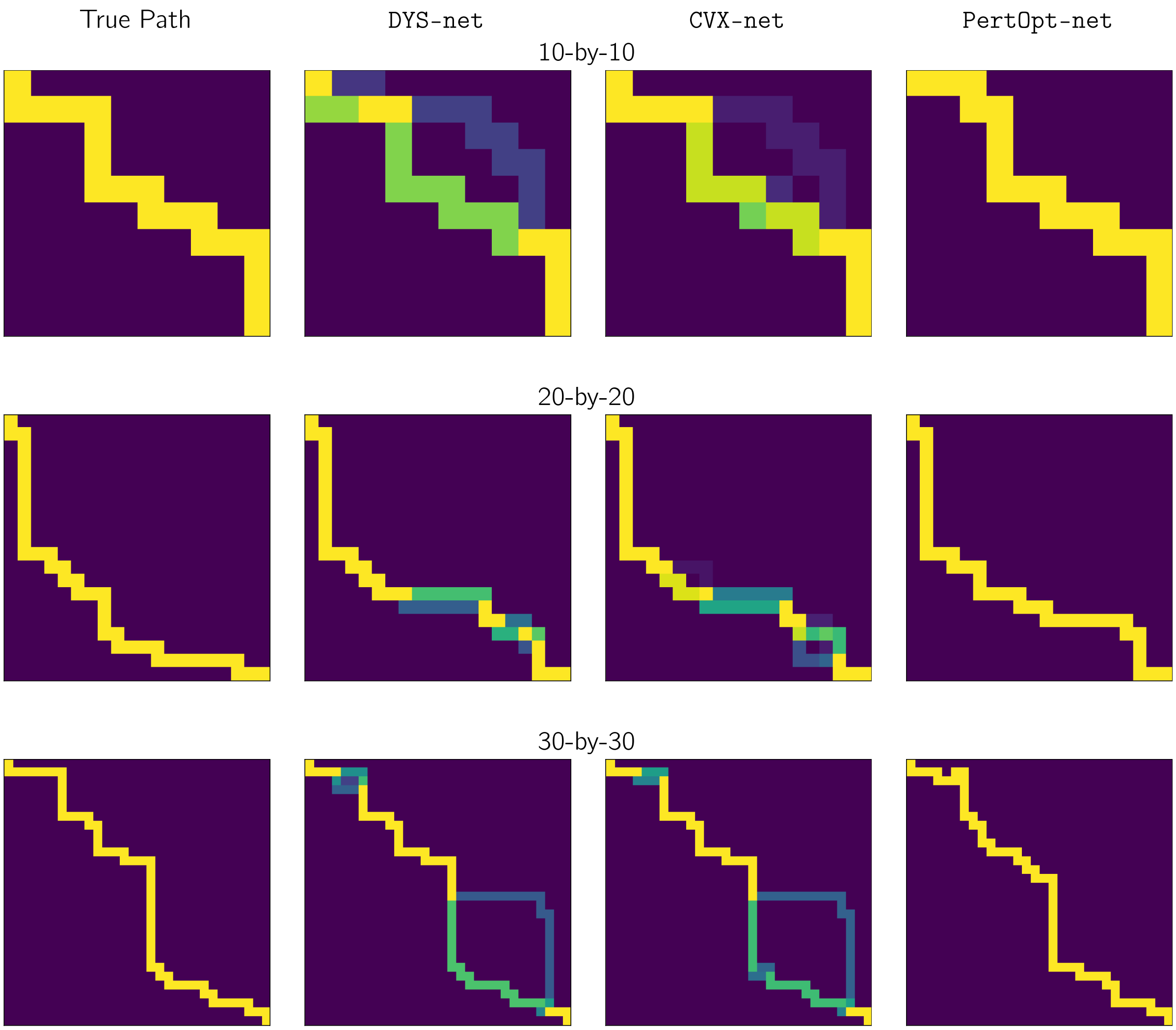


Figure 6: True paths (column 1), paths predicted by `DYS-net` (column 2), `CVX-net` (column 3), and `PertOpt-net` (column 4). Samples are taken from different grid sizes: 10-by-10 (row 1), 20-by-20 (row 2), and 30-by-30 (row 3).