Decision-Focused Learning without Decision-Making: Learning Locally Optimized Decision Losses

Anonymous Author(s)
Affiliation
Address
email

Abstract

Decision-Focused Learning (DFL) is a paradigm for tailoring a predictive model to a downstream optimization task that uses its predictions in order to perform better *on that specific task*. The main technical challenge associated with DFL is that it requires being able to differentiate through the optimization problem, which is difficult due to discontinuous solutions and other challenges. Past work has largely gotten around this this issue by *handcrafting* task-specific surrogates to the original optimization problem that provide informative gradients when differentiated through. However, the need to handcraft surrogates for each new task limits the usability of DFL. In addition, there are often no guarantees about the convexity of the resulting surrogates and, as a result, training a predictive model using them can lead to inferior local optima. In this paper, we do away with surrogates altogether and instead *learn* loss functions that capture task-specific information. To the best of our knowledge, ours is the first approach that entirely replaces the optimization component of decision-focused learning with a loss that is automatically learned. Our approach (a) only requires access to a black-box oracle that can solve the optimization problem and is thus *generalizable*, and (b) can be *convex by construction* and so can be easily optimized over. We evaluate our approach on three resource allocation problems from the literature and find that our approach outperforms learning without taking into account task-structure in all three domains, and even hand-crafted surrogates from the literature.

1 Introduction

Predict-then-optimize [7, 8] is a framework for using machine learning to perform decision-making. As the name suggests, it proceeds in two stages—first, a predictive model takes as input *features* and makes some *predictions* using them, then second, these predictions are used to parameterize an optimization problem that outputs a *decision*. A large number of real-world applications involve both prediction and optimization components and can be framed as predict-then-optimize problems—for e.g., recommender systems in which missing user-item ratings need to be predicted [13], portfolio optimization in which future performance needs to be predicted [17], or strategic decision-making in which the adversary behavior needs to be predicted [14].

Historically, learning a predictive model was often done independently of the downstream optimization task. However, recent work [8, 16, 25, 9, 3, 25, 27, 26, 6] has shown that it is possible to achieve *better task-specific performance* by tailoring the predictive model to the downstream task. A large subset of these approaches, which we will refer to as *decision-focused learning (DFL)* [25], learn a

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

predictive model by differentiating through the entire prediction and optimization pipeline end-to-end. This procedure yields a loss function we call the *decision loss*, which optimizes for the quality of decisions induced by the predictive model.

Training with the decision loss can be challenging because the solutions to an optimization are often discontinuous in terms the predictions. For example, consider the optimization $\arg\min(\hat{y}_1, \hat{y}_2)$ that picks the smaller of two predicted values $\hat{y}_1$ and $\hat{y}_2$. The solution will rarely change for $\hat{y}_1 \pm \epsilon$. This results in an uninformative decision loss with gradients that are zero or undefined, neither of which are useful for learning a predictive model. In addition, even if the gradients are non-zero, there may be local optima in the decision loss that further complicate training. To address these challenges, DFL approaches often leverage a handcrafted surrogate task that provides more useful gradients. These surrogate optimization problems may be constructed by relaxing the original problem [8, 16, 25, 9], adding regularization to the objective [3, 25, 27], or even using entirely an different optimization problem that shares the same decision space [26].

Designing good surrogates is an art, requiring manual effort and insight about each optimization problem. We propose a fundamentally different approach: learning a decision loss *directly* for a given task, circumventing surrogate problem design entirely. Our framework represents the loss as a function in a particular parametric family and selects parameters which provide an informative loss for the optimization task. We call the resulting loss a *locally optimized decision loss (LODL)*.

Our starting point is the observation that a good decision loss should satisfy three properties: it should (i) be *faithful* to the original task, i.e., the decision quality is consistent with the original problem; (ii) provide informative gradients everywhere (i.e., defined and non-zero); (iii) be convex in prediction space to avoid local minima. These demands are in tension—the first requirement prevents the loss function from being modified too much to achieve the other two. It is not obvious apriori that any tractable parametric family should be able to simultaneously satisfy all three properties for the complex structure induced by many optimization tasks. We resolve this tension by separately modeling the loss function *locally* for the neighborhood around each individual training example. Faithful representation of the decision loss is easier to accomplish locally in each individual neighborhood than globally across instances, allowing us to introduce convex parametric families of loss functions which capture structural intuition about properties important for optimization. To fit the parameters, we sample points in the neighborhood of the true labels, evaluate the decision loss associated with these sampled points, and then train a loss function to mimic the decision loss.

We evaluate $LODL$s on three resource allocation domains from the literature [12, 25, 24]. Perhaps surprisingly, we find that $LODL$s outperforms handcrafted surrogates in two out of the three. In our analysis, we discover a linear correlation between decision quality and the agreement of the learned $LODL$ with the decision loss in a specific neighborhood of the true label. Our approach motivates a new line of research on decision-focused learning.

2 Background

We consider the following predict-then-optimize problem. Given features $\boldsymbol{x}$, a predictive model $\boldsymbol{M_\theta}$ produces predictions $\hat{\boldsymbol{y}} = \boldsymbol{M_\theta}(\boldsymbol{x})$. These predictions $\hat{\boldsymbol{y}}$ are used to parameterize an optimization problem that is solved to yield decision $\boldsymbol{z}^*(\hat{\boldsymbol{y}})$:

$$\boldsymbol{z}^*(\hat{\boldsymbol{y}}) = \arg\min_{\boldsymbol{z}} \quad f(\boldsymbol{z}, \hat{\boldsymbol{y}}) \;\; s.t. \;\; g_i(\boldsymbol{z}) \leq 0 \text{ for } i \in \{1, \ldots, m\}. \tag{1}$$

Predictions are evaluated w.r.t. to *decision loss (DL)* of the decision they induce—the value of the objective function of the optimization under the ground truth parameters $\boldsymbol{y}$: $DL(\hat{\boldsymbol{y}}, \boldsymbol{y}) = f(\boldsymbol{z}^*(\hat{\boldsymbol{y}}), \boldsymbol{y})$. Thus, for a dataset $[(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_N, \boldsymbol{y}_N)]$, we aim to learn a model $\boldsymbol{M_\theta}$ that generates predictions $[\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_N]$ that minimize DL:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{n=1}^{N} DL(\boldsymbol{M_\theta}(\boldsymbol{x}_n), \boldsymbol{y}_n).$$

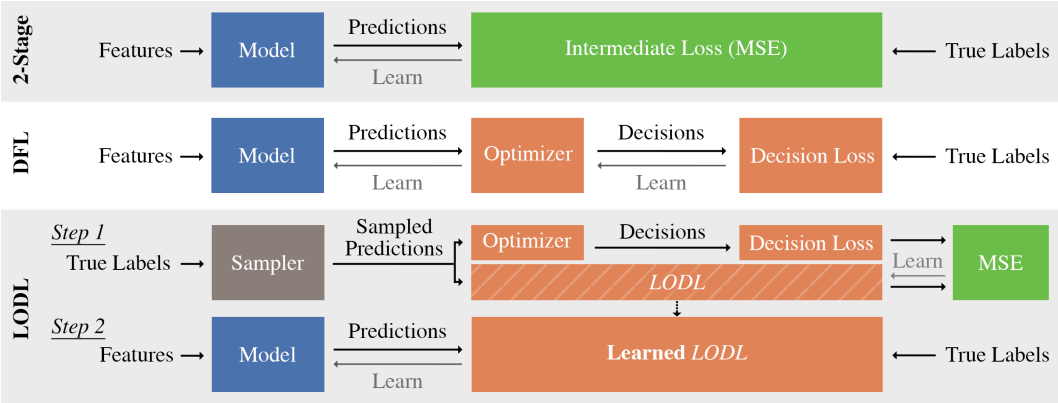


Figure 1: Schematic highlighting how the predictive model $\boldsymbol{M_\theta}$ is learned in different approaches. $LODL$ does not require backpropagating through the optimization problem.

This is in contrast with standard supervised learning approaches in which the quality of a prediction is measured by a somewhat arbitrary intermediate loss (e.g., mean squared-error) that does not contain information about the downstream decision-making task. In this paper, we refer to models that use an intermediate loss as *2-stage* and those that directly optimize for DL as *decision-focused learning (DFL)*. Figure 1 outlines how a predictive model is learned using these approaches.

3 Related Work

A great deal of recent work on decision-focused learning and related topics aims to incorporate information about a downstream optimization problem into the training of a machine learning models. Some optimization problems (especially strongly convex ones) can be directly differentiated through [7, 2, 1]. For others, particularly discrete optimization problems with ill-defined gradients, a variety of approaches have been proposed. Most of these construct surrogate loss functions via smoothing the optimization problem [18, 25, 9, 26, 23, 15]. Alternatively, Elmachtoub and Grigas [8] propose a closed-form convex surrogate loss with desirable theoretical properties; however, this loss applies only when the optimization problem has a linear objective function. Similarly, Mulamba et al. [20] provide a contrastive learning-based surrogate which does not require differentiation through optimization, but which is also developed specifically for linear objectives. When the predictive model is itself a linear function, Guler et al. [11] propose an approach to search directly for the best model.

Perhaps the most related work to ours is by Berthet et al. [4]. They differentiate through linear optimization problems during training by adding randomized perturbations to smooth the induced loss function. Specifically, they randomly perturb the predictions $\hat{y}$ with random noise ϵ and solve for $z^*(\hat{y} + \epsilon)$. This can be interpreted as replacing the decision loss with its averaged value in a neighborhood around $\hat{y}$, where the averaging smooths the function and ensures differentiability. There are two key differences between our approach and theirs. First, they apply random perturbations to optimization *in the training loop* in order to produce a smoother surrogate loss. By contrast, we use random perturbations to learn a loss function *prior to training*; during training optimization is removed entirely. This allows us to use the same random samples to inform each training iteration instead of drawing new samples per-iteration. Second, their approach applies only to linear objective functions while ours applies to arbitrary optimization problems.

4 Locally Optimized Decision Losses ($LODL$)

In this paper, we do away with the need for custom task-specific relaxations of the optimization problem $\boldsymbol{z}^*(\hat{\boldsymbol{y}})$ by instead translating task-specific information from the decision loss DL into a loss function $LODL_{\boldsymbol{\phi}}(\hat{\boldsymbol{y}}, \boldsymbol{y})$ that approximates the behavior of DL *and* is convex by construction.

The goal is to find a set of parameters $\boldsymbol{\phi}^*$ such that training a predictive model using $LODL_{\boldsymbol{\phi}^*}$ yields a trained model $\boldsymbol{M}_{\boldsymbol{\theta}^*_{LODL}}$ that performs just as well as the (hard to compute) model which optimizes DL directly. We now detail the main steps in this process.

4.1 Representing the $LODL$

The key design choice in instantiating our framework is choice of the parametric family used to represent the $LODL$. Optimization problems can induce a complex loss landscape which is not easily summarized in a closed-form function with concise parameterization. Accordingly, we design a set of families which capture phenomena particularly important for common families of predict-then-optimize problems. There are two key steps: first, creating a representation which is *local* to each training instance and second, choosing the parametric family which will be *optimized* over to produce each local representation.

Local loss functions We introduce a *separate* set of parameters $\boldsymbol{\phi}_n$ for each $[(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n), \ldots, (\boldsymbol{x}_N, \boldsymbol{y}_N)]$ in the training set. We take this step because learning a *global* approximation to $DL(\hat{\boldsymbol{y}}, \boldsymbol{y})$ (for arbitrary $\hat{\boldsymbol{y}}$) is hard; it requires learning a closed-form approximation to the general optimization problem $\boldsymbol{z}^*(\hat{\boldsymbol{y}})$ which may not always exist. Introducing separate parameters per-instance gives two key advantages.

1. **Learning for a specific $\boldsymbol{y}_n$:** Instead of learning a $\mathbb{R}^{\dim(\boldsymbol{y})+\dim(\hat{\boldsymbol{y}})}$ function $LODL_{\phi}(\hat{\boldsymbol{y}}, \boldsymbol{y})$ to imitate $DL(\hat{\boldsymbol{y}}, \boldsymbol{y})$, we instead learn N different $\mathbb{R}^{\dim(\hat{\boldsymbol{y}})}$ functions $LODL_{\phi_n}(\hat{\boldsymbol{y}})$ that imitate $DL(\hat{\boldsymbol{y}}, \boldsymbol{y}_n)$ for each $n \in N$. Doing this significantly reduces the dimensionality of the learning problem—this is especially relevant when N is not very large in comparison to $\dim(\boldsymbol{y})$ (as in our experiments). It also circumvents the need to enforce invariance properties on the global loss. For example, many optimization problems are invariant to permutations of the ordering of dimensions in $\boldsymbol{y}$, making it difficult to measure the quality of $\hat{\boldsymbol{y}}$'s across different instances. In a local loss, the ordering of the dimensions is fixed and so this issue is no longer relevant.
2. **Learning for only $\hat{\boldsymbol{y}}_n \approx \boldsymbol{y}_n$:** In addition to the simplification above, we don't try to learn a faithful approximation $\forall \hat{\boldsymbol{y}} \in \mathbb{R}^{\dim(\boldsymbol{y})}$—we instead limit ourselves to learning an approximation of $DL(\hat{\boldsymbol{y}}, \boldsymbol{y}_n)$ only when $\hat{\boldsymbol{y}}$ *is in the neighborhood of* $\boldsymbol{y}_n$. This is because we assume that our predictive model will always get us in the neighborhood of the true labels, and the utility of $LODL$ is in helping distinguish between these points.

Parametric families for the local losses Having made the decision to allow separate parameters for each training instance, the second design choice is how to represent each local loss, i.e., the specific parametric family to use. Our choice must be sufficiently expressive to capture the local dynamics of the optimization problem while remaining sufficiently efficient to be replicated across the N training instances. We propose that the structure of the loss function should capture the underlying rationale for why decision-focused learning provides an advantage over 2-stage in the first place; this is the key behavior which will underpin improved decision quality. We identify three key phenomena which motivate the design of the family of loss functions:

1. **Relative importance of different dimensions:** Typically, the different dimensions of a prediction problem are given equal weight, e.g., the MSE weights errors in each coordinate of $\boldsymbol{y}$ equally. However, there may be some dimensions along which DL is more sensitive to local perturbations. For example, a knapsack problem may be especially sensitive to errors in the value of items which are on the cusp of being chosen. In such cases, DFL can capture the relative importance of accurately predicting different dimensions.
2. **Cost of correlation:** Given the possibly large dimensionality of $\boldsymbol{y}$, in practice, the predictive model $M_{\boldsymbol{\theta}}$ does not typically predict $\boldsymbol{y}$ directly. Instead, the the structure of the optimization is exploited to make multiple predictions $[\hat{y}_1, \ldots, \hat{y}_K]$ that are then combined to create $\hat{\boldsymbol{y}}$. For example, in a knapsack problem, we might train a model which separately predicts the value of each item (i.e., predicts each entry of $\hat{\boldsymbol{y}}$ separately) using features specific to that item, instead of jointly predicting the entire set of values using the features of all of the items. However, Cameron

et al. [5] show that ignoring the correlation between different sub-$\boldsymbol{y}$ scale predictions (as in 2-stage) can result in poor optimization performance, and that DFL can improve by propagating information about the interactions between entries of $\boldsymbol{y}$ to the predictive model.

3. **Directionality of predictions:** In the $\arg\min(\hat{y}_1, \hat{y}_2)$ example from the introduction, the prediction $\hat{y}_1 = 2, \hat{y}_2 = 1$ produces the same decision as $\hat{y}_1 = 2000, \hat{y}_2 = 1$. On the other hand, the prediction $\hat{y}_1 = 0.5, \hat{y}_2 = 1$ leads to a different decision. As a result, over-predicting and under-predicting often have different associated costs for some optimization problems. Predictive models trained with DFL can take into account this behavior while those trained by typical symmetric 2-Stage losses like MSE cannot.

Given these insights, we propose three corresponding families of loss functions for $LODL_\phi$, each of which is convex by design and has a global minima at the true label $\boldsymbol{y}_n$—a desirable property because DL also has its minima at $\hat{\boldsymbol{y}} = \boldsymbol{y}$.

1. **WeightedMSE:** To take into account the relative importance of different dimensions, we propose a weighted version of MSE:

$$\text{WeightedMSE}(\hat{\boldsymbol{y}}) = \sum_{l=1}^{\dim(\boldsymbol{y})} w_l \cdot (\hat{y}_l - y_l)^2,$$

   where 'weights' w_l are the free parameters in the $LODL$, i.e., $\boldsymbol{\phi} = \boldsymbol{w}$.

2. **Quadratic:** To take into account the effect of correlation of different dimensions on each other, we propose learning a quadratic function that has terms of the form $(\hat{y}_i - y_i)(\hat{y}_j - y_j)$:

$$\text{Quadratic}(\hat{\boldsymbol{y}}) = (\hat{\boldsymbol{y}} - \boldsymbol{y})^T H (\hat{\boldsymbol{y}} - \boldsymbol{y}),$$

   where $\boldsymbol{\phi} = H$ is a learned low-rank symmetric Positive semidefinite (PSD) matrix. This family of functions is convex as long as H is PSD, which we enforce by parameterizing $H = L^T L$, where L is a low-rank triangular matrix L of dimension $\dim(\boldsymbol{y}) \times k$ and k is the desired rank.

   This loss function family has an appealing interpretation because learning $LODL$ is similar to estimating the partial derivative of DL with respect to its first input $\hat{\boldsymbol{y}}_n$. Specifically, consider the first three terms of the Taylor expansion of DL with respect to $\hat{\boldsymbol{y}}_n$ at $(\boldsymbol{y}_n, \boldsymbol{y}_n)$:

$$\begin{aligned} DL(\overbrace{\boldsymbol{y}_n + \boldsymbol{\epsilon}}^{\hat{\boldsymbol{y}}_n}, \boldsymbol{y}_n) &= \overbrace{DL(\boldsymbol{y}_n, \boldsymbol{y}_n)}^{\text{constant}} + \overbrace{\nabla_{\hat{\boldsymbol{y}}_n} DL(\boldsymbol{y}_n, \boldsymbol{y}_n)}^{0 \leftarrow (\boldsymbol{y}_n, \boldsymbol{y}_n) \text{ is a minima}} \boldsymbol{\epsilon} + \boldsymbol{\epsilon}^T \overbrace{\nabla^2_{\hat{\boldsymbol{y}}_n} DL(\boldsymbol{y}_n, \boldsymbol{y}_n)}^{\text{Hessian } H} \boldsymbol{\epsilon} + \dots \\ &\approx DL(\boldsymbol{y}_n, \boldsymbol{y}_n) + (\hat{\boldsymbol{y}}_n - \boldsymbol{y}_n)^T H (\hat{\boldsymbol{y}}_n - \boldsymbol{y}_n) \end{aligned}$$

   Quadratic $LODL_\phi$ can be seen as a 2nd-order Taylor-series approximation of DL at $(\boldsymbol{y}_n, \boldsymbol{y}_n)$ where the learned H approximates the Hessian of DL. Note that WeightedMSE is a special case of this Quadratic loss when $H = diag(\boldsymbol{w})$.

3. **DirectedWeightedMSE and DirectedQuadratic:** To take into account the fact that overpredicting and underpredicting can have different consequences, we propose modifications to the two loss function families above. For WeightedMSE, we redefine the weight vector $\boldsymbol{w}$ as below, and learn both $\boldsymbol{w}_+$ and $\boldsymbol{w}_-$. Similarly for Quadratic, we define 4 copies of the parameter L based on the directionality of the predictions.

$$w_l = \begin{cases} w_+, & \text{if } \hat{y}_i - y_i \geq 0 \\ w_-, & \text{otherwise} \end{cases} \qquad L_{ij} = \begin{cases} L_{ij}^{++}, & \text{if } \hat{y}_i - y_i \geq 0 \text{ and } \hat{y}_j - y_j \geq 0 \\ L_{ij}^{+-}, & \text{if } \hat{y}_i - y_i \geq 0 \text{ and } \hat{y}_j - y_j < 0 \\ L_{ij}^{-+}, & \text{if } \hat{y}_i - y_i < 0 \text{ and } \hat{y}_j - y_j \geq 0 \\ L_{ij}^{--}, & \text{otherwise} \end{cases}$$

4.2 Learning $LODL_\phi$

Given families proposed in Section 4.1, our goal is to learn some $\boldsymbol{\phi}_n^*$ for every $n \in N$ such that $LODL_{\boldsymbol{\phi}_n}(\hat{\boldsymbol{y}}_n) \approx DL(\hat{\boldsymbol{y}}_n, \boldsymbol{y}_n)$ for $\hat{\boldsymbol{y}}_n$ "close" to $\boldsymbol{y}_n$. We propose a supervised approach to learning $\boldsymbol{\phi}_n^*$ which proceeds in two steps (Figure 1): (1) we build a dataset mapping $\hat{\boldsymbol{y}}_n \rightarrow DL(\hat{\boldsymbol{y}}_n, \boldsymbol{y}_n)$ in

the region of $\boldsymbol{y}_n$, and then (2) we use this dataset to estimate $\boldsymbol{\phi}_n^*$ by minimizing the mean-squared error to the true decision loss:

$$\boldsymbol{\phi}_n^* = \arg\min_{\boldsymbol{\phi}_n} \frac{1}{K} \sum_{k=1}^{K} (LODL_{\boldsymbol{\phi}_n}(\boldsymbol{y}_n^k) - DL(\boldsymbol{y}_n^k, \boldsymbol{y}_n))^2.$$

This framework has the key advantage of reducing the design of good surrogate tasks (a complex problem requiring in-depth knowledge of each optimization problem) to supervised learning (for which many methods are available). Indeed, future advances, e.g. in representation learning, can simply be plugged into our framework.

The major remaining step is to specify the construction of the dataset for supervised learning of $\boldsymbol{\phi}_n^*$. We propose to construct this dataset by sampling a set of K points $[\boldsymbol{y}_n^1, \ldots, \boldsymbol{y}_n^K]$ in the vicinity of $\boldsymbol{y}$ and calculate $DL(\boldsymbol{y}_n^k, \boldsymbol{y}_n)$ for each. In this paper, we consider three sampling strategies:

1. **All-Perturbed:** Add zero-mean Gaussian noise to the true label $\boldsymbol{y}_n$:
$$\boldsymbol{y}_n^i = \boldsymbol{y}_n + \boldsymbol{\epsilon}^k = \boldsymbol{y}_n + \alpha \cdot \mathcal{N}(0, I),$$
where α is a normalization factor and I is a $dim(\boldsymbol{y}) \times dim(\boldsymbol{y})$ identity matrix.
2. **1-Perturbed or 2-Perturbed:** Estimating the behavior of $DL(\boldsymbol{y}_n + \epsilon^i, \boldsymbol{y}_n)$ for small ϵ can alternatively be interpreted as estimating $(\frac{\delta}{\delta \hat{\boldsymbol{y}}_n})^{dim(\boldsymbol{y}_n)} DL(\hat{\boldsymbol{y}}_n, \boldsymbol{y}_n)$ (i.e., the $dim(\boldsymbol{y}_n)^{th}$ partial derivative of DL w.r.t. its first input $\hat{\boldsymbol{y}}_n$) at $(\boldsymbol{y}_n, \boldsymbol{y}_n)$ because all the dimensions of $\hat{\boldsymbol{y}}_n$ are being varied simultaneously. While computing $(\frac{\delta}{\delta \hat{\boldsymbol{y}}_n})^{dim(\boldsymbol{y}_n)}$ is computationally challenging, it can be estimated using the simpler 1st or 2nd partial derivatives. This corresponds to perturbing only one or two dimensions at a time.

5 Experiments

To validate the efficacy of our approach, we run experiments on three resource allocation tasks from the literature. Because these problems are maximizations, we use the term *decision quality (DQ)* to represent the negative decision loss (higher is better). A more complete description of the experimental setup is provided in the appendix.

Linear Model This domain involves learning a linear model when the underlying mapping between features and predictions is cubic. With limited model capacity, one cannot model *all* the data accurately and so the loss function must prioritize *where* to make errors. Such problems are common in the explainable AI literature [21, 10, 12] where predictive models must be interpretable. Concretely, the aim is to choose the top $B = 1$ out of $N = 50$ resources using a linear model.

Predict: Given a feature $x_n \sim U[0, 1]$, use a linear model to predict the utility $\hat{y}$ of resource n, where the true value is $y_n = 10x_n^3 - 6.5x_n$. Combining predictions yields $\hat{\boldsymbol{y}} = [\hat{y}_1, \ldots, \hat{y}_N]$.

Optimize: Choose the $B = 1$ (budget) resources with highest utility: $\boldsymbol{z}^*(\hat{\boldsymbol{y}}) = \arg\operatorname{topk}(\hat{\boldsymbol{y}})$

Surrogate: Because the $\arg\max$ operation is piecewise constant, DFL requires a surrogate—we use the soft Top-K proposed by Xie et al. [27] that reframes the Top-K problem with entropy regularization as an optimal transport problem. This surrogate is *not* convex in the predictions.

Web Advertising This is a submodular optimization task taken from [25]. The aim is to determine on which websites to advertise given features about different websites.

Predict: Given features $\boldsymbol{x}_m$ associated with some website m, predict the click-through rates (CTRs) for a fixed set of $N = 10$ users on $M = 10$ websites $\hat{\boldsymbol{y}}_m = [\hat{y}_{m,1}, \ldots, \hat{y}_{m,N}]$. The task is based on the Yahoo! Webscope Dataset [28] which contains multiple CTR matrices. To obtain the features $\boldsymbol{x}_m$ for each website m, the true CTRs $\boldsymbol{y}_m$ for the website are scrambled by multiplying with a random $N \times N$ matrix A, i.e., $\boldsymbol{x}_m = A\boldsymbol{y}_m$.

Optimize: Given the matrix of CTRs, determine on which $B = 2$ (budget) websites to advertise such that the expected number of users that click on the ad *at least once* is maximized, i.e., $\boldsymbol{z}^*(\hat{\boldsymbol{y}}) = \arg\max_{\boldsymbol{z}} \sum_{j=0}^{N} (1 - \prod_{i=0}^{M} (1 - z_i \cdot \hat{y}_{ij}))$, where all the $z_i \in \{0, 1\}$.

Surrogate: Instead of requiring that $\boldsymbol{z}_i \in \{0, 1\}$, the multi-linear relaxation from Wilder et al. [25] allows fractional values. The DL induced by the relaxation is *non-convex*.

Portfolio Optimization This is a Quadratic Programming domain popular in the literature [7, 24]. The aim is to choose a distribution over N stocks in Markowitz portfolio optimization [17, 19] that maximizes the expected profit minus a quadratic risk penalty. We choose this domain as a stress test—it is highly favorable for DFL because the optimization problem naturally provides informative gradients and thus requires no surrogate.

Predict: Given historical data $\boldsymbol{x}_n$ about stock n, predict the future stock price y_n. Combining the predictions $\hat{y}_n$ across a consistent set of $N = 50$ stocks together yields $\hat{\boldsymbol{y}} = [\hat{y}_1, \ldots, \hat{y}_N]$. We use historical price and volume data from 2004 to 2017 downloaded from QuandlWIKI dataset [22].

Optimize: Given a historical correlation matrix Q between pairs of stocks, choose a distribution $\boldsymbol{z}$ over stocks that maximizes $\boldsymbol{z}^T\boldsymbol{y} - \lambda \cdot \hat{\boldsymbol{y}}^T Q \hat{\boldsymbol{y}}$, where $\lambda = 0.1$ is the risk aversion constant.

5.1 Results

We train either a linear model (for the Linear Model domain) or a 2-layer fully-connected neural network (for the other domains) using the learned loss functions $LODL_{\boldsymbol{\phi}^*}$ approach and compare it to the following baselines:

1. **Random**: The predictions are sampled uniformly from $[0, 1]^{dim(\boldsymbol{y})}$.
2. **Optimal**: The predictions are equal to the true labels $\boldsymbol{y}$.
3. **2-Stage**: The model is trained on the standard MSE loss ($\frac{1}{N}\sum_{n=1}^{N} ||\hat{\boldsymbol{y}}_n - \boldsymbol{y}_n||^2$).
4. **DFL**: Decision-focused learning using the specified surrogate.
5. **NN:** To determine how important convexity is for $LODL$ we define $LODL_\phi = NN_\phi(\hat{\boldsymbol{y}})$ in which NN_ϕ is a 4-layer fully-connected Neural Network (NN). There are no guarantees about the convexity or minima of the loss learned using this family.

Table 1: The decision quality achieved by each approach—**higher is better**. DirectedQuadratic consistently performs well.

Loss Function	**Normalized DQ On Test Data**		
	Linear Model	Web Advertising	Portfolio Optimization
Random	0	0	0
Optimal	1	1	1
2-Stage (MSE)	-0.953 ± 0.000	0.476 ± 0.147	0.320 ± 0.015
DFL	0.828 ± 0.383	0.854 ± 0.100	**0.348 ± 0.015**
NN	**0.962 ± 0.000**	0.814 ± 0.137	-0.105 ± 0.084
WeightedMSE	-0.934 ± 0.060	0.576 ± 0.151	0.308 ± 0.018
DirectedWeightedMSE	**0.962 ± 0.000**	0.533 ± 0.137	0.322 ± 0.015
Quadratic	-0.752 ± 0.377	**0.931 ± 0.040**	0.272 ± 0.020
DirectedQuadratic	**0.962 ± 0.000**	0.910 ± 0.043	0.325 ± 0.014

Table 1 shows the main results. We find that, in all domains, training predictive models with $LODL$ outperforms training them with a task-independent 2-stage loss. Surprisingly, it also outperforms DFL which has the benefit of having handcrafted surrogates. This is strong evidence in favor of our hypothesis that we can automate away the need for handcrafting surrogates. We now analyze the results in terms of the domains:

1. **Linear Model:** In this domain, the directionality of predictions is very important. As we describe in Section 4.1, predicting values higher than the true utilities for the B resources with highest utility, does not change the decision. However, if their predicted utility is lower than that of the $B - 1^{\text{th}}$ best resource, the decision quality is affected. As a result, we see that the "Directed" methods perform significantly better than their competition.
2. **Web Advertising**: In this domain, Wilder et al. [25] suggest that the decision quality is linked to being able to predict the quantity $\sum_j \hat{y}_{ij}$ (the sum of CTRs across all users j for a given website

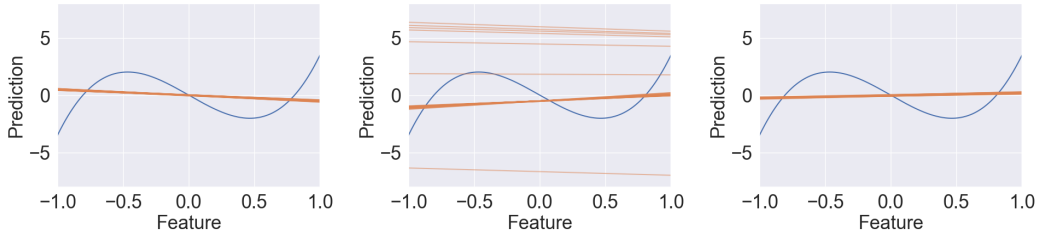
i). However, because the input features for every $\hat{y}_{ij}$ are the same $\boldsymbol{x}_i$, the errors can be correlated and so the sum can be biased. As a result, the ability to penalize the correlations between two predictions $\hat{y}_{ij}$ and $\hat{y}_{jk}$ is important to being able to perform well on this task—which results in the Quadratic methods outperforming the others.

3. **Portfolio Optimization**: Given that this stress-test domain was built to be favorable to DFL, it outperforms all other approaches with statistical significance. While the directed LODL methods do not outperform DFL, they nonetheless significantly outperform 2-stage at $p < 0.05$.

We now analyze the results in terms of the methods:

1. **Our DirectedQuadratic $LODL$ consistently does well:** In addition to consistently high expected values (always better than 2-stage), the associated variance is lower as well.
2. **Lack of convexity can cause inconsistent results**: While NN does well in the first two domains, it fails catastrophically in the Portfolio Optimization domain.
3. **DFL has a large variance in performance**: This is best illustrated by Figure 2b. Linear models with a positive slope pick the largest B values whereas those with a negative slope choose the smallest. We can see that, while DFL generally does well (induces linear models with positive slope), it can get stuck in local optima (induces stray lines with negative slope).

Figure 2: Graphs showing the true (blue) and 100 learned (orange) mappings between the features and predictions in the Linear Model domain. 2-stage does badly, DFL learns the correct slope on average but is subject to random failures, and DirectedQuadratic does well.



(a) 2-Stage (b) Decision-Focused Learning (c) DirectedQuadratic

5.2 Ablations

We study the impact of the sampling strategy and number of samples on the performance of the $LODL$ methods in the Web Advertising domain.

1. **Varying the sampling strategy:** In Table 2, we find that the *the best sampling strategy is loss family-specific*. Specifically, NN and DirectedWeightedMSE perform best with the "2-Perturbed" strategy, while the remaining $LODL$s perform best with the "All-Perturbed" strategy.

Table 2: Ablations across different sampling methods in the Web Advertising domain. All methods and sampling strategies use 500 samples. The best sampling strategy is loss family-specific.

Approach	Normalised Test DQ (1-Perturbed)	Normalised Test DQ (2-Perturbed)	Normalised Test DQ (All-Perturbed)
NN	0.855 ± 0.121	$\mathbf{0.888 \pm 0.086}$	0.802 ± 0.159
WeightedMSE	0.496 ± 0.138	0.533 ± 0.139	$\mathbf{0.576 \pm 0.151}$
DirectedWeightedMSE	0.470 ± 0.150	$\mathbf{0.533 \pm 0.160}$	0.500 ± 0.130
Quadratic	0.773 ± 0.250	0.877 ± 0.097	$\mathbf{0.918 \pm 0.048}$
DirectedQuadratic	0.770 ± 0.187	0.842 ± 0.109	$\mathbf{0.845 \pm 0.080}$

2. **Varying the number of samples used:** Table 4 (in the appendix) shows that all models perform better with more samples. The variance reduces as the number of samples increases (especially for Quadratic), suggesting that better approximations of DL lead to more consistent outcomes.

5.3 Correlation between 'quality' of $LODL$ and decision quality

Recall that $LODL$ losses are fit using a Gaussian sampling strategy centered around the true labels. It is natural to ask how well this proxy loss correlates with the decision quality on test data. We do this by measuring the mean absolute error (MAE) of LODL relative to the ground truth decision loss for points in the *Gaussian neighborhood* around the true labels.

This Gaussian neighborhood is only an approximation of the true distribution of interest—the distribution of predictions generated by the predictive model that is trained using the $LODL$ loss. We can measure the MAE on this distribution, which we call the *empirical neighborhood*.

Table 3: A table showing the relationship between the quality of the learned loss for different classes of $LODL$s, and the performance of a model trained on said loss. The Empirical Neighbourhood MAE is linearly correlated with DQ while the Gaussian Neighbourhood MAE is not.

Approach	MAE in Gaussian Neighborhood	MAE in Empirical Neighborhood	Normalized DQ on Test Data
NN	0.0094 ± 0.0006	0.0222 ± 0.0173	0.802 ± 0.159
WeightedMSE	0.0104 ± 0.0000	0.0448 ± 0.0171	0.576 ± 0.151
DirectedWeightedMSE	$\mathbf{0.0092 \pm 0.0000}$	0.0558 ± 0.0164	0.500 ± 0.130
Quadratic	0.0096 ± 0.0000	$\mathbf{0.0086 \pm 0.0052}$	$\mathbf{0.918 \pm 0.048}$
DirectedQuadratic	0.0106 ± 0.0000	0.0191 ± 0.0079	0.845 ± 0.080

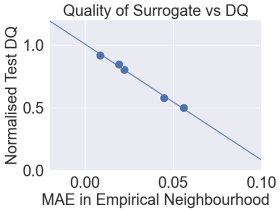


Table 3 shows the results. All methods are able to approximate the DL comparably well in the Gaussian neighborhood, but this does not correlate well with decision quality. In contrast, the error on the empirical neighborhood is tightly linearly correlated with decision quality. Furthermore, if we extrapolate the line of best fit to where the MAE is 0, i.e., when there is no discrepancy $LODL$ and DL, we find that the trend predicts the normalized DQ would be 1. This suggests that MAE on the empirical neighborhood is a useful metric to target—see Section 6 for more discussion.

6 Discussion and Conclusion

Our work proposes a conceptual shift from hand-crafting surrogate losses for decision problems to automatically learning them, and demonstrates experimentally that the $LODL$ paradigm enables us to learn high-quality models without such manual effort. Nevertheless, our current instantiation of this framework has limitations which are areas for future work.

First, to simplify the analysis, we considered $LODL$ that additively decompose across the dimensions of $\boldsymbol{y}$. Doing so allows us to isolate the effects of fit to the decision loss from the generalization performance across the dimensions of $\boldsymbol{y}$ and leads to a conceptually simple pipeline. Nevertheless, such LODLs require linearly many parameters in $\dim(\boldsymbol{y})$ (and quadratically many for DirectedQuadratic). This scaling may be undesirable in practice. However, we note that our main conceptual contribution—directly learning a loss for decision making—opens up the potential for future work to learn models that generalize across dimensions, which could offer benefits in scalability in addition to $LODL$'s current advantages in ease of use and model quality.

Second, in Section 5.3, we demonstrate that the fit of a $LODL$ to the empirical neighborhood around the ground truth label is highly correlated with the eventual decision quality. However, this metric is not accessible as a target during training—thus, we use the more tractable Gaussian neighborhood instead. While the Gaussian neighborhood method does yield models that perform well, it does not correlate well with the decision quality across $LODL$ parameterizations. It would be valuable to study the empirical neighborhood MAE to better understand the reasons for this discrepancy and potentially develop $LODL$s with even stronger performance.

In summary, LODL provides an alternate framework for machine learning which informs decision making, opening up new avenues towards models which are both high-performing and easily trained.

References

[1] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in Neural Information Processing Systems*, 32, 2019.

[2] Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J. Zico Kolter. Differentiable mpc for end-to-end planning and control. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[3] Brandon Amos, Vladlen Koltun, and J Zico Kolter. The limited multi-label projection layer. *arXiv preprint arXiv:1906.08707*, 2019.

[4] Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with differentiable pertubed optimizers. *Advances in Neural Information Processing Systems*, 33:9508–9519, 2020.

[5] Chris Cameron, Jason Hartford, Taylor Lundy, and Kevin Leyton-Brown. The perils of learning before optimizing. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36, 2022.

[6] Emir Demirović, Peter J Stuckey, James Bailey, Jeffrey Chan, Chris Leckie, Kotagiri Ramamohanarao, and Tias Guns. An investigation into prediction+ optimisation for the knapsack problem. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 241–257. Springer, 2019.

[7] Priya Donti, Brandon Amos, and J Zico Kolter. Task-based end-to-end model learning in stochastic optimization. *Advances in Neural Information Processing Systems*, 30, 2017.

[8] Adam N Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Management Science*, 2021.

[9] Aaron Ferber, Bryan Wilder, Bistra Dilkina, and Milind Tambe. MIPaaL: Mixed integer program as a layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1504–1511, 2020.

[10] Joseph Futoma, Michael Hughes, and Finale Doshi-Velez. POPCORN: Partially observed prediction constrained reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pages 3578–3588. PMLR, 2020.

[11] Ali Ugur Guler, Emir Demirovic, Jeffrey Chan, James Bailey, Christopher Leckie, and Peter J Stuckey. Divide and learn: A divide and conquer approach for predict+ optimize. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36, 2022.

[12] Michael Hughes, Gabriel Hope, Leah Weiner, Thomas McCoy, Roy Perlis, Erik Sudderth, and Finale Doshi-Velez. Semi-supervised prediction-constrained topic models. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, pages 1067–1076, 2018.

[13] Folasade Olubusola Isinkaye, Yetunde O Folajimi, and Bolande Adefowoke Ojokoh. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 16(3): 261–273, 2015.

[14] Debarun Kar, Benjamin Ford, Shahrzad Gholami, Fei Fang, Andrew Plumptre, Milind Tambe, Margaret Driciru, Fred Wanyama, Aggrey Rwetsiba, Mustapha Nsubaga, and Joshua Mabonga. Cloudy with a chance of poaching: Adversary behavior modeling and forecasting with real-world poaching data. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, page 159–167, 2017.

[15] Jayanta Mandi and Tias Guns. Interior point solving for LP-based prediction+optimisation. *Advances in Neural Information Processing Systems*, 33:7272–7282, 2020.

[16] Jayanta Mandi, Peter J Stuckey, Tias Guns, et al. Smart predict-and-optimize for hard combinatorial optimization problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1603–1610, 2020.

[17] Harry M Markowitz and G Peter Todd. *Mean-variance analysis in portfolio choice and capital markets*. John Wiley & Sons, 2000.

[18] Arthur Mensch and Mathieu Blondel. Differentiable dynamic programming for structured prediction and attention. In *International Conference on Machine Learning*, pages 3462–3471. PMLR, 2018.

[19] Richard O Michaud. The Markowitz optimization enigma: Is 'optimized' optimal? *Financial Analysts Journal*, 45(1):31–42, 1989.

[20] Maxime Mulamba, Jayanta Mandi, Michelangelo Diligenti, Michele Lombardi, Victor Bucarey, and Tias Guns. Contrastive losses and solution caching for predict-and-optimize. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, 2021.

[21] Sanjana Narayanan, Abhishek Sharma, Catherine Zeng, and Finale Doshi-Velez. Prediction-focused mixture models. *arXiv preprint arXiv:2110.13221*, 2021.

[22] Quandl. WIKI various end-of-day data, 2022. URL `https://www.quandl.com/data/WIKI`.

[23] Sebastian Tschiatschek, Aytunc Sahin, and Andreas Krause. Differentiable submodular maximization. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 2731–2738, 2018.

[24] Kai Wang, Bryan Wilder, Andrew Perrault, and Milind Tambe. Automatically learning compact quality-aware surrogates for optimization problems. *Advances in Neural Information Processing Systems*, 33:9586–9596, 2020.

[25] 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*, volume 33, pages 1658–1665, 2019.

[26] Bryan Wilder, Eric Ewing, Bistra Dilkina, and Milind Tambe. End to end learning and optimization on graphs. *Advances in Neural Information Processing Systems*, 32:4672–4683, 2019.

[27] Yujia Xie, Hanjun Dai, Minshuo Chen, Bo Dai, Tuo Zhao, Hongyuan Zha, Wei Wei, and Tomas Pfister. Differentiable top-k with optimal transport. *Advances in Neural Information Processing Systems*, 33:20520–20531, 2020.

[28] Yahoo! Webscope dataset, 2007. URL `https://webscope.sandbox.yahoo.com/`. ydata-ysm-advertiser-bids-v1.0.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] As supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] In the appendix
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] In all tables.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] In the appendix
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No] It isn't clear exactly what the license is, but all the datasets are publicly available.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]