Submitted to *Management Science*
manuscript

End-to-End Learning for Optimization via Constraint-Enforcing Approximators

Rares Cristian
Operations Research Center, MIT, Cambridge, MA, raresc@mit.edu

Pavithra Harsha
IBM Watson AI Research, Yorktown Heights, NY pharsha@us.ibm.com

Georgia Perakis
Operations Research Center, MIT, Cambridge, MA georgiap@mit.edu

Briann Quanz
IBM Watson AI Research, Yorktown Heights, NY blquanz@us.ibm.com

Predictive methods are used to provide inputs for downstream optimization problems. Nevertheless, using the downstream task-based objective (e.g. the optimization task) to learn the intermediate predictive model, is often better than using the intermediate task objective, such as the prediction. The learning task in this approach is referred to as end-to-end learning. Nevertheless, using existing methods for end-to-end learning can prove difficult due to lack of differentiability in the resulting optimization problem function. To address this issue, we propose a neural network architecture that can learn to near-optimally solve such an optimization problem, particularly ensuring its output satisfies the feasibility constraints via alternate projections. We establish the method's convergence to an optimal solution and provide analytical results about the structure of the iterates from our proposed neural network's solution, proving that the method converges exponentially fast. Furthermore, the approach proposed in this paper is computationally more efficient than existing methods as the neural network produces high-quality approximations faster and scales better with the problem size. Our approach applies to a wide range of optimization problems including deterministic, single-stage as well as two-stage stochastic optimization problems. We illustrate how our proposed method applies to (i) the capacitated multi-product newsvendor, (ii) a two-stage multi-location cross-fulfilment newsvendor problem, (iii) the quadratic cost newsvendor problem, and (iv) a shortest path problem with a computer vision task of predicting edge costs from terrain maps. Finally, we illustrate how the proposed method out-performs existing approaches in terms of final task-based loss and training time.

1. Introduction

In many operations management problems, several input quantities need to be predicted from historical data prior to determining the corresponding best operational decision. Examples include predicting travel times in a vehicle routing problem, the demand distribution in a supply chain inventory optimization problem, among many others. A popular approach is to estimate these quantities using a machine learning model from historical data along with observed contextual features such as seasonal trends, location and price information among others. This machine learning model is used to create a forecast for a new observation which is subsequently used for decision making. In these problems, the quality of the resultant business objectives, such as the overall costs in the supply chain problem, is often the more important performance indicator, compared to the accuracy of the machine learning models, such as the mean square error of the demand estimate.

For example, consider the classical newsvendor problem in which one needs to forecast future demand. There is a holding cost associated with stocking more than the demand, and a backorder cost associated with each unit stocked below demand. Given perfect information about the distribution, the optimal stocking quantity is known to be a particular quantile dependent on the two costs (see Arrow et al. 1951). Nevertheless, such closed form solutions are not common in most application settings. For example, once there are multiple products with constraints on the stocking allocation, the optimal solution has no longer a closed form. A predict-then-optimize type of approach may take the following steps. (1) Assume the demand distribution comes from some family of distributions (such as Gaussian) and make a prediction using the mean and variance estimated from the data. Subsequently, one can solve the corresponding stochastic optimization problem. Alternatively (2) one may explicitly learn a distributional forecast (for example, learn the probability of each demand realization) and then subsequently solve the corresponding stochastic optimization problem to determine the allocation quantity. Clearly such distributional forecasts are necessary since using only a point forecast one cannot a priori know the correct statistic of the demand distribution to target. Indeed, simply making point forecasts by minimizing the mean-squared error between the prediction and the observed uncertainty has the potential to produce poor decisions as discussed for example in Cameron et al. (2021).

However, joint prediction and optimization, (also referred to as end-to-end learning), bypasses this issue. The primary goal in this case is to learn a forecast whose corresponding decision has the lowest possible cost (objective function). This goal is exactly incorporated into the training algorithm used to learn the forecasting model. Consider the newsvendor problem as an example. In this problem one needs to be able to implicitly learn the best forecast to make for the "correct quantile" of the distribution as that would minimize cost. At the same time, one needs to keep in mind that any model will always make some error. However, not all errors are the same. For

example, suppose the backorder cost is twice that of the holding cost. A model only minimizing mean-squared error between its predictions and the target would identify an over- and under-prediction the same. However, a model trained end-to-end would identify the true cost resulting from a prediction, and that over-prediction results in lower cost.

A significant issue in training end-to-end models comes from the required computational complexity. In particular, the loss function is now the output of a decision-making process (to determine the true cost) which is expensive to evaluate. In this paper, we propose a novel approach to end-to-end learning that is more efficient compared to existing methods and achieves similar performance. For many classes of these optimization problems like linear or quadratic, as well as for stochastic optimization problems like the resource allocation problems, where there are over and under utilization penalties, we observe that point forecasts are sufficient when using an end-to-end learning approach. Together with the proposed learning method in this paper, we can now efficiently solve computationally intractable stochastic optimization problems in an end-to-end sense.

More specifically, this paper presents the following contributions:

1) **Novel End-to-End Approach for Single and Two Stage Problems:** We achieve more efficient end-to-end training by replacing the complex optimization-based loss function with a simpler efficient approximation. We introduce a neural network architecture that can learn to near-optimally solve convex optimization problems. This is performed by introducing the key notion of approximate projections. We refer to this approach as ProjectNet. We further incorporate this network architecture into an end-to-end learning framework for solving the single-stage stochastic optimization problem. Furthermore, we present a way to extend our approach to two-stage stochastic optimization problems in section 4.
2) **Convergence and Rate of Convergence:** We prove bounds on how quickly a sequence generated by the ProjectNet architecture converges as well as bounds on the regret of the solution compared to using the exact original optimization problem. In addition, we also prove generalization bounds on how much data is needed to train the ProjectNet model.
3) **Point versus Distributional Forecasts for Stochastic Optimization:** Moreover, we prove that for a large class of stochastic optimization problems, if one uses a loss-like objective function, then point forecasts through an end-to-end learning approach produce the same optimal solutions as when making distributional forecasts (the latter is often needed in predict-then-optimize approaches).
4) **Computational Results on Several Applications:** We present computational results on three problems. (i) An electricity generation and planning problem using real-world data from PJM, an organization coordinating the movement of wholesale electricity around 13 US states.

(ii) A shortest path problem requiring sate of the art residual network and convolutional networks to learn from map data. (iii) A two-stage multi-warehouse cross-fulfillment newsvendor problem. We compare our methods along multiple axes. First, performance and accuracy as the amount of training data increases. Second, accuracy as the problem size (number of optimization variables) increases. Third, running time of our method as problem size increases. We show that our proposed method produces better (or in the worst case competitive) solutions in terms of average cost, while at the same time the method is computationally faster to train relative to other end-to-end learning methods. That is, the proposed approach consistently scales better in terms of runtime as problem size increases, being 2 to 10 times faster for various problems while retaining the same accuracy.

We show that our proposed approach is computationally efficient as it allows one to quickly approximate the solution, without explicitly solving the optimization problem itself. When training the predictive forecasting model via gradient descent, existing approaches require solving an optimization problem as a subproblem at each gradient update iteration. In contrast, we only require a simple pass through a neural network. We illustrate computationally for a variety of problems that ProjectNet produces better values in terms of the objective function as compared to the projected gradient descent method using the same step size and number of iterations. For example, for the matching problem we show computationally that the ProjectNet model produces solutions up to 12.5% better than gradient descent.

1.1. Some Related Literature

One challenge for a large class of decision making problems (for example, for linear optimization problems) stems from the fact that the gradient of the optimal solution with respect to the predicted quantities, e.g., the cost vector of the decision problem, is zero or undefined. Often, one aims to learn the forecasting function by using gradient descent. But if the gradient is zero (as in linear optimization), then end-to-end learning becomes next to impossible to perform. The reason why this gradient is zero is because in linear optimization problems, the optimal solution lies in a discrete set of vertices of the feasible region and hence, it is a piece-wise constant function of the cost vector. Nevertheless, linear optimization problems constitute a major class of problems of interest. Dobkin et al. (1979) (among others) have shown that any polynomial-time solvable problem can be formulated as a linear optimization problem. This includes for instance the shortest path problem, maximum matching among many others. Thus, most of the existing literature has focused on the linear optimization case.

End-to-end methods for convex problems. To deal with the lack of differentiability issue for linear optimization problems, Elmachtoub and Grigas (2022) construct a convex and differentiable

approximation of the objective. Furthermore, Elmachtoub et al. (2020) extend this framework to training with decision trees, and Liu and Grigas (2022) extend it to a setting where data and decisions need to be taken online over time. In the case of unconstrained quadratic objectives, Kao et al. (2009) train a model to directly minimize task loss. This work was extended in the OptNet framework of Amos and Kolter (2017) to constrained quadratic optimization by analyzing the optimality (KKT) conditions. In particular, this is performed by calculating the optimal solution (using a traditional solution method) and differentiating through the corresponding optimality (KKT) conditions. Donti et al. (2017a) further applies this methodology to stochastic optimization problems with probabilistic constraints. Subsequently, Wilder et al. (2019) propose to add a quadratic regularization term to linear optimization problems in order to obtain approximate solutions. Furthermore, Agrawal et al. (2019) extend the method of analyzing the optimality (KKT) conditions to more general convex optimization problems. For other examples of methods geared towards linear optimization, Mandi and Guns (2020) use an interior point method to retrieve the optimal solution and calculate the gradient by differentiating through the log-barrier terms. Vlastelica et al. (2020) view the optimization problem as a piece-wise constant function (which happens as we discussed, due to the presence of the zero-gradients) of the cost vector and design a continuous interpolation. Berthet et al. (2020) tackle the problem by adding a stochastic perturbation to the linear objective, producing nonzero gradients of the output. Unfortunately, a shortcoming of these approaches is that they are computationally expensive, as the underlying methods need to solve the nominal decision-making optimization problem (e.g., finding an optimal fulfilment strategy given a demand prediction, or a shortest path problem given a prediction of the travel times of the edges) at each gradient step.

To remedy this issue, we propose a novel neural network architecture which can learn the optimization problem solution, allowing one to approximate the solution fast, without explicitly solving the exact optimization problem itself. The main issue though in doing so, arises from ensuring the output of the network is a feasible solution to the optimization problem. To overcome this issue, we use an approximate projection method onto the feasible region after each layer of the neural network. We accomplish this by decomposing the problem into a sequence of projections onto "simpler" sets (for example, projecting onto a single constraint).

Learning decisions directly. Other approaches in the literature aim to directly learn a decision function rather a forecast. A significant issue to resolve is ensuring the output satisfies the problem constraints. For example, Ban and Rudin (2019) consider a newsvendor problem and model the decisions directly as a linear function of the features. In this case, the task of learning can be formulated exactly as a linear optimization problem which can be efficiently solved to optimality. However, this approach does not allow for more complex decision mappings. Alternatively, Frerix

et al. (2019) describe a feasible solution, not by its coordinates, but as a convex combination of the vertices and extreme rays describing the feasible region. Unfortunately, the primary downside of this approach lies in the often exponential size of the vertex set.

Closer to our approach, Donti et al. (2021) transform the output of the learning model into a feasible solution by projecting on the equality constraints, and subsequently performing gradient descent to satisfy the inequality constraints. The method of Qiu et al. (2024) proposes to ensure feasibility by learn a subset of variables and completing the rest automatically by leveraging fundamental properties of dual-optimal solutions for their specific use-case of AC power flow problems.

To ensure feasibility, the method proposed in this paper only performs a sequence of alternating projections onto simpler sets. Additionally, our method trains a surrogate model which explicitly learns and approximates solutions to the optimization problem (this may also be of independent interest). The spirit of the work in Shirobokov et al. (2020) is somewhat similar to ours but within a rather different context. That is, Shirobokov et al. (2020) consider simulation problems (a common task within fields in physics or engineering for instance) rather than optimization problems to solve after the initial forecast. The simulation problems considered in Shirobokov et al. (2020) are often highly expensive to perform, and as a result that paper proposes a surrogate generative network method to approximate the outcome. There has also been interesting work in creating continuous relaxations of algorithms to make them differentiable. For instance, Petersen et al. (2021) accomplishes this by introducing continuous relaxations of simple algorithmic concepts such as conditional statements, loops, and indexing, which can be pieced together to describe any algorithm.

Within the context of inventory optimization problems, the integration of learning and optimization was initially studied for example, in Ban and Rudin (2019), Bertsimas and Kallus (2020) and Oroojlooyjadid et al. (2020) for solving the feature-based newsvendor problem. For a more complex version of the problem, Qi et al. (2020) devise an end-to-end method for the multi-period replenishment problem. Unlike the previous classes of problems we discussed, this is a mixed integer linear optimization problem. Qi et al. (2020) propose a neural network model which can learn the binary values of which days to make an order for inventory as well as how much to order. The paper does so by pre-calculating the optimal solution (order quantities for each day) to many instances observed in the data. The paper then trains a network to learn the mapping from features to the optimal order quantities.

End-to-end learning for integer problems. Although this is not the focus of our paper, there has also been work in the recent years on end-to-end learning for hard combinatorial problems with integer constraints. In what follows, we only provide as a result a brief discussion on some related literature. Ferber et al. (2020) approach the problem by generating cutting planes, taking

the corresponding linear relaxation, and applying the approach of Wilder et al. (2019) to solve the new end-to-end problem. Mandi et al. (2020) apply the approach of Elmachtoub and Grigas (2022) to hard combinatorial problems. Guler et al. (2020) take a new approach using a divide and conquer algorithm when using a linear model to predict the uncertainty This approach is applicable to any optimization problem (with possibly nonlinear constraints) and with uncertainty in a linear objective function. The approach of Paulus et al. (2021) considers the case of uncertainty in constraints in addition to the objective, but restricts itself to linear constraints.

Learning efficient approximations of hard problems. Another stream of work is "learning to learn" methods and meta-learning (ways to learn algorithms that can solve optimization problems). For instance, Li and Malik (2016) learn a policy to solve unconstrained continuous optimization problems. They abstract the notion of an optimization algorithm to be a sequence of updates computed from some function of the objective function, the current location and past locations in the sequence. The paper restricts themselves to function values and gradients. Our approach on the other hand can be viewed as learning to solve a different convex problem instead, one which is faster to solve and provides a solution close to that of the original. See section 3.

There has also been focus on learning how to better optimize neural networks. For instance, learn new methods beyond stochastic gradient descent and its variations for training neural networks. See for example in Sergio and Colmenarejo (2016), Andrychowicz et al. (2016), Chen et al. (2017), Lv et al. (2017), Bello et al. (2017) and the references therein.

In a similar vein, there has been recent work in training neural networks to learn optimal solutions of optimization problems. Much of the work in that area, has focused on difficult mixed integer linear optimization problems (MILP) since the aim is to provide solutions more efficiently than solving the original MILP. For instance, one of the earliest proposals for solving the Travelling Salesman Problem (Hopfield and Tank 1985) was to transform the problem into a labelling problem (which edge should be in the path) and use Lagrange multipliers to penalize the solution's violations of the constraints. However, this method has been shown to be highly unstable and sensitive to initialization (Wilson and Pawley 1988). More effective methods for combinatorial problems on graphs have been developed by training recurrent neural networks using reinforcement learning. Examples include Vinyals et al. (2015) and Bello et al. (2016). These works are particularly useful for graph problems in which the RNN decides which next node to visit. However, these approaches are not developed with the end-to-end learning framework in mind.

Paper organization: The rest of the paper is organized as follows. In section 2 we formally describe the problem setting and the end-to-end framework. Then in section 3.2 we describe the ProjectNet architecture and how to apply it to learn forecasts end-to-end. We also investigate several theoretical properties and provide guarantees of our approach. In section 4 we extend our method to

two-stage stochastic optimization problems in which there are first-stage decisions to make after which the uncertainty is realized and a second-stage decision then has to be made. Finally, in section 5 we compare computationally our proposed method relative to other existing approaches. We perform these comparisons for the traditional as well as a two-stage multi-location cross-fulfilment newsvendor problem, a convex cost newsvendor problem and a shortest path problem.

2. End-to-end learning framework for single-stage stochastic optimization problems

We first formally present the problem class requiring the integration of machine learning (to forecast uncertain parameters) with optimization problems. Consider a convex objective function $g_u(w)$, where u is the uncertain parameter(s) that must be predicted. We assume g_u and its derivative is "simple" to evaluate. For instance, for linear objectives $g_u(w) = c^T w$, the cost vector parameters are $u = c$, while for quadratic objectives, $g_u(w) = q^T w + w^T Q w$, the parameters are $u = (q, Q)$. We define the following optimization problem with multiple types of constraints $h_i(w) \leq 0$, for a convex function h_i and linear equality constraints $l_j(w) = 0$:

$$\begin{aligned} w^*(u) = \arg\min_w \quad & g_u(w) \\ \text{subject to } & h_i(w) \leq 0, \quad i = 1, \ldots, p_1 \\ & l_j(w) = 0, \quad j = 1, \ldots, p_2. \end{aligned} \tag{1}$$

For ease of notation, we let $\mathcal{P}$ also denote the feasible region of problem (1). Suppose we are given N data points $(x^1, u^1), \ldots, (x^N, u^N)$ with features $x^n \in \mathbb{R}^p$ and realized costs $u^n \in \mathbb{R}^d$ (we discussed examples of what these can represent above). Given a model f_θ parameterized by some θ, for some out-of-sample data x, we make a prediction $f_\theta(x)$ (e.g., through a neural network) and a corresponding decision $w^*(f_\theta(x)) \in \mathcal{P}$. Afterwards, for a realized cost vector u, we incur overall cost $g_u(w^*(f_\theta(x)))$. However, u itself comes from an unknown distribution dependent on features x. Let D_x be the distribution of u conditioned on observing features x. Then, the optimal decision $w^*(D_x)$ is given by minimizing the expected cost, as given by the following stochastic optimization problem:

$$w^*(D_x) = \arg\min_{w \in \mathcal{P}} \mathbb{E}_{u \sim D_x} \left[g_u(w) \right]. \tag{2}$$

We first present some traditional methods of approaching this problem and later illustrate the differences and the advantages of our end-to-end method.

Traditional approaches: predict-then-optimize. A traditional method, which we will refer to as a *predict-then-optimize* approach, learns the forecasting function independently of the downstream optimization problem. For example, one may attempt to learn the distribution D_x itself. We briefly

discuss two methods to accomplish this. First, we may assume D_x belongs to some class of distributions such as the normal distribution, in which case, one only needs to predict the mean and variance from the data. Using this prediction, we can then solve or approximate the solution to (2). A second approach is to assume the uncertainty can only take some finite number of discrete values and predict the probability of each value to occur. Then again, one would solve problem (2) using this discrete distribution learned.

On the other hand, the problem simplifies if the objective function is linear as a function of the uncertainty. That is, we can write

$$g_u(w) = u^T \Phi(w) \tag{3}$$

for some function Φ. The simplest example is when $\Phi(w) = w$ and the objective $g_u(w) = u^T w$ is fully linear. Another example, is the case where the objective is quadratic in terms of w. For example, $g_u(w) = \sum_{i,j} u_{i,j} w_i w_j$. In these cases with the objective function being linear in the uncertainty, we can rewrite the stochastic optimization problem as

$$\begin{aligned} w^*(D_x) = &\arg\min_{w \in \mathcal{P}} \mathbb{E}_{u \sim D_x} \left[g_u(w) \right] \\ = &\arg\min_{w \in \mathcal{P}} \mathbb{E}_{u \sim D_x} \left[u^T \Phi(w) \right] \\ = &\arg\min_{w \in \mathcal{P}} \mathbb{E}_{u \sim D_x} \left[u \right]^T \Phi(w). \end{aligned} \tag{4}$$

This implies one only needs to predict the mean of the distribution D_x and this is optimal independent of the optimization problem itself. This may be done simply by minimizing the mean squared error between the forecast and the true in-sample cost. Let the forecast be some $f_\theta(x)$, parameterized by θ (such as a neural network with weights θ). Then, we aim to minimize

$$\theta_{\text{predict-then-optimize}} = \arg\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \| f_\theta(x^n) - u^n \|_2^2 . \tag{5}$$

However, in all of these cases, making a forecast independent of the optimization problem loses out on important gains. Clearly, making perfect forecasts will result in optimal decisions. But in practice, any forecasting model will incur some error. Within the context of the optimization problem, not all errors result in the same cost. For example, consider the case where the downstream optimization problem is linear. Let $\bar{u}_x$ be the mean of the distribution D_x and consider two possible forecasts u_1 and u_2. It is often the case that u_1 is closer in mean-squared distance to $\bar{u}_x$ but the corresponding decision $w^*(u_2)$ has lower objective function value (that is, $g_{\bar{u}_x}(w^*(u_2)) \leq g_{\bar{u}_x}(w^*(u_1))$). We illustrate this idea in the example of Figure 1. In this example we observe that point $w^*(u_2)$ is actually the same as the optimal solution $w^*(\bar{u}_x)$, while $w^*(u_1)$ is not. This is despite the fact that the cost vector u_2 has higher distance from $\bar{u}_x$ than u_1 does. This suggests that a lower mean-squared error in prediction does not necessarily result in decisions with a lower objective function value. This is due to the fact that a traditional predict-then-optimize method does not make use of this fact when learning its forecasts.

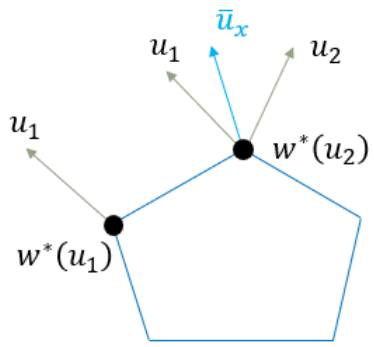


Figure 1 **Lower mean-squared error in prediction does not imply lower objective value of the corresponding decisions.**

The end-to-end approach. As we already discussed, the final goal of an end-to-end approach is to minimize the cost (objective function) of the final decision. We wish to learn a forecasting function $f_\theta(x)$ so that the decisions $w^*(f_\theta(x))$ minimize the expected cost of $g_u(w^*(f_\theta(x))$. This is expressed as follows:

$$\theta_{\text{end-to-end}} = \arg\min_\theta \mathbb{E}_x \mathbb{E}_{u\sim D_x} \left[g_u(w^*(f_\theta(x)) \right]. \tag{6}$$

As an approximation, we replace the expectation with samples drawn from the empirical distributions given by the data. This leads to the following learning task which we call the end-to-end problem we wish to solve.

$$\theta_{\text{end-to-end}} = \arg\min_\theta \sum_{n=1}^{N} g_{u^n}(w^*(f_\theta(x^n)). \tag{7}$$

See Figure 2. Given that f_θ is some neural network, one would generally aim to solve this problem using gradient descent. In order to do so, one must be able to compute the gradient

$$\frac{\partial w^*(\nu)}{\partial \nu} \tag{8}$$

For each gradient computation one must solve an expensive optimization problem $w^*(f_\theta(x^n))$. Due to the computational time required to solve the optimization problem $w^*(f_\theta(x^n))$ at each gradient step, end-to-end methods easily become intractable as the dimension of the problem increases as

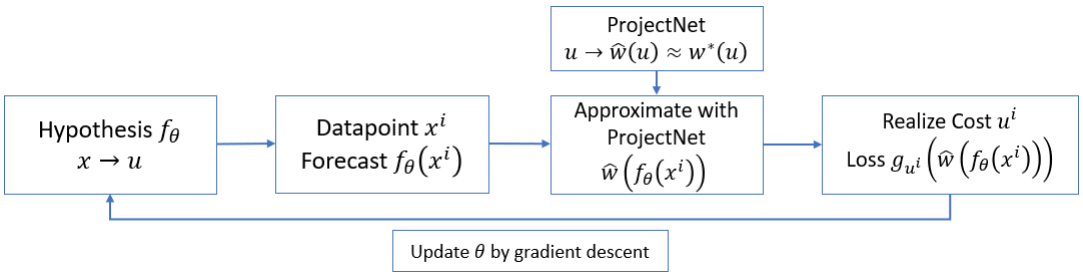


Figure 2 **Flow diagram for end-to-end learning using ProjectNet.**

this step is significantly more expensive than a simple forward and backward pass normally required to train neural networks.

We propose to address this issue by using some approximation $\hat{w}(u)$ which is faster to compute. We show that can safely make this switch. Moreover, we will observe that it suffices for $\hat{w}$ to approximate w^* well only on the distribution of data that we observe. In general, the algorithms used to compute w^* work for any possible input, however this is not necessary in our case. This leads us to learn an approximation $\hat{w}$ that performs well, and potentially more efficiently, on only the distribution of data we observe.

We first formalize the fact that one can learn using some approximation $\hat{w}$ instead of the exact w^*. Let $\hat{f}$ be the model one learns when minimizing the expected cost when using $\hat{w}$ and let f^* be the model when using the true w^*. That is,

$$\hat{f} = \arg\min_f \mathbb{E}_x \mathbb{E}_{u\sim D_x}[g_u(\hat{w}(f(x))] \tag{9}$$

$$f^* = \arg\min_f \mathbb{E}_x \mathbb{E}_{u\sim D_x}[g_u(w^*(f(x))] \tag{10}$$

Then, having learned $\hat{f}$, we make out-of-samples decisions $w^*(\hat{f}(x))$. The objective value of $w^*(\hat{f}(x))$ compared to the optimal decision $w^*(f^*(x))$ if one had trained using the exact oracle w^* is now bounded only by the approximation error of $\hat{w}$ compared to w^*:

THEOREM 1. *[Approximation impact on end-to-end]One can decompose the difference in cost of making decisions based off of $\hat{f}$ from the optimal cost from making decisions based off f^* by decomposing the error into two components:*

$$\mathbb{E}_x \mathbb{E}_{u\sim D_x}\left[g_u(w^*(\hat{f}(x))) - g_u(w^*(f^*(x)))\right] \leq \epsilon_1 + \epsilon_2 \tag{11}$$

where

1. $\epsilon_1 = \mathbb{E}_x \mathbb{E}_{u\sim D_x} |g_u(w^*(f^*(x))) - g_u(\hat{w}(f^*(x)))|$. *This describes how well $\hat{w}$ approximates w^* on input from f^**
2. $\epsilon_2 = \mathbb{E}_x \mathbb{E}_{u\sim D_x}\left|g_u(w^*(\hat{f}(x))) - g_u(\hat{w}(\hat{f}(x)))\right|$. *Similarly, this describes how well $\hat{w}$ approximates w^* on input from $\hat{f}$.*

Notice that to bound the error terms ϵ_1, ϵ_2 the approximation $\hat{w}$ only needs to perform well on inputs from $f^*(x)$ and $\hat{f}(x)$. Moreover, while we do not know f^*, we do have instances of $f^*(x)$. Specifically, the data $f^*(x_1) = u_1, \ldots, f^*(x_N) = u_N$ that we observe.

Proof. The proofs follows from a sequence of standard manipulations and triangle inequalities. Consider adding and subtracting the expectation of the term $g_u(w^*(\hat{f}(x)))$:

$$\mathbb{E}_x \mathbb{E}_{u\sim D_x}\left[g_u(w^*(\hat{f}(x))) - g_u(w^*(f^*(x)))\right] = \tag{12}$$

$$\mathbb{E}_x \mathbb{E}_{u\sim D_x}\left[g_u(w^*(\hat{f}(x))) - \Big(g_u(w^*(\hat{f}(x))) - g_u(w^*(\hat{f}(x)))\Big) - g_u(w^*(f^*(x)))\right] \tag{13}$$

and grouping the first two and second two terms:

$$\mathbb{E}_x\mathbb{E}_{u\sim D_x}\left[g_u(w^*(\hat{f}(x))) - g_u(w^*(f^*(x)))\right] \leq \epsilon_2 + \mathbb{E}_x\mathbb{E}_{u\sim D_x}\left[g_u(w^*(\hat{f}(x))) - g_u(w^*(f^*(x)))\right]. \quad (14)$$

Now consider adding and subtracting the expectation of the term $g_u(\hat{w}(f^*(x)))$. We find

$$\mathbb{E}_x\mathbb{E}_{u\sim D_x}\left[g_u(w^*(\hat{f}(x))) - g_u(w^*(f^*(x)))\right] \leq \quad (15)$$

$$\epsilon_2 + \mathbb{E}_x\mathbb{E}_{u\sim D_x}\left[g_u(w^*(\hat{f}(x))) - (g_u(\hat{w}(f^*(x))) - g_u(\hat{w}(f^*(x)))) - g_u(w^*(f^*(x)))\right]. \quad (16)$$

The last two terms in the expectation can be bounded by ϵ_1 so we are now left with

$$\mathbb{E}_x\mathbb{E}_{u\sim D_x}\left[g_u(w^*(\hat{f}(x))) - g_u(w^*(f^*(x)))\right] \leq \epsilon_2 + \epsilon_1 + \mathbb{E}_x\mathbb{E}_{u\sim D_x}\left[g_u(w^*(\hat{f}(x))) - g_u(\hat{w}(f^*(x)))\right]. \quad (17)$$

Finally, note the expectation on the right hand side is always non-positive. Indeed, $\hat{f}$ is the minimizer of $\mathbb{E}_x\mathbb{E}_{u\sim D_x}[g_u(w^*(f(x)))]$ over $f \in \mathcal{F}$ by definition (see (9)). Therefore, $\mathbb{E}_x\mathbb{E}_{u\sim D_x}\left[g_u(w^*(\hat{f}(x)))\right] \leq \mathbb{E}_x\mathbb{E}_{u\sim D_x}[g_u(\hat{w}(f^*(x)))]$. □

Moreover, there are additional advantages to using an approximation $\hat{w}$ instead of the exact w^* beyond only speed considerations. Specifically, this approximation aspect is crucial for linear optimization problems — an incredibly broad class of problems with countless applications. As noted earlier, the values of these gradients for commonly occurring linear decision problems are typically zero (as we already discussed above, this is because the optimal solution is always a vertex, and hence a piece-wise constant function, for which the gradient is zero). Moreover, if the gradient is zero this would result in no update of the weights of the neural network, making it impossible to learn. Therefore, using an approximation $\hat{w}$ instead of w^* in solving (7) for which the gradient of $\hat{w}$ is non-zero would resolve this issue.

See the diagram in Figure 2 for an illustration of our proposed end-to-end method. The exact structure of $\hat{w}$ and how it is learned can be found in section 3.2. An advantage of our approach is that the computationally expensive optimization problem $w^*(u)$ is never explicitly solved, even during the learning stage to approximate $w^*(u)$. Rather we replace it with the forward pass of a simple neural network $\hat{w}$ which we will denote as a *ProjectNet* model architecture.

Once a forecasting model $\theta_{\text{end-to-end}}$ has been learned, for new out-of-sample features x, we make the forecast $u = f_\theta(x)$ which then allows us to determine the decision $w^*(u)$ by solving the optimization problem of interest. Depending on whether the forecast is point or distributional in nature, the down stream problem of identifying $w^*(u)$ is a deterministic or a stochastic optimization problem. We have already shown that making point forecasts is optimal for objective functions that are linear in terms of the uncertainty. Next we show this is also true for nonlinear loss-type objective functions found for example, in inventory and resource management problems. We primarily focus on these classes of problems in the computational section of this paper although the proposed methodology is applicable to convex stochastic optimization problems in general.

Point Forecasts vs. Distributional Forecasts. We showed in the case of objectives that are linear in the uncertainty, it is sufficient to predict a point forecast (the mean of D_x) instead of the entire distribution D_x itself (see for eq. (4)). Using only a predict-then-optimize framework, separating the prediction from the optimization, point forecasts are not sufficient for other more complex objectives. As an example, consider the newsvendor problem. In this case, there is unknown demand u and a decision w to be made on the stock to allocate. There is a holding cost h, for every unit of stock unsold and a backorder cost b, for every unit of demand that is unmet. The objective function can be represented as

$$g_u(w) = \max(h(w-u), b(u-w)), \tag{18}$$

which is a nonlinear objective. In this problem, it is known that the optimal stocking decision is the $(b/(h+b))^{th}$ quantile of the demand distribution D_x (see Arrow et al. 1951). However, a predict-then-optimize point forecast would incorrectly target the mean.

We show that for end-to-end learning problems, point forecasts are sufficient for a large class of convex stochastic problems.

Proposition 1. *Consider a single-stage stochastic optimization problem with a loss-type objective function $g_u(w)$. That is, objectives for which u and w belong to the same feasible space and*

$$u = \arg\min_{w \in \mathcal{P}} g_u(w). \tag{19}$$

Then, for any distribution D of the uncertainty u, there exists a single point forecast d which produces the same solution as solving the original stochastic problem:

$$\arg\min_{w \in \mathcal{P}} \mathbb{E}_{u \sim D}\left[g_u(w)\right] = w^*(d). \tag{20}$$

An end-to-end method with objective as in (6) would find exactly this forecast d at optimality.

Proof. Let d be the solution to the problem using distributional forecast D:

$$d = \arg\min_{w} \mathbb{E}_{u \sim D}[g_u(w)]. \tag{21}$$

Now consider making a forecast of exactly d. Then, $w^* = \arg\min_w g_d(w) = d$, since g_d is a loss function. □

It is a major advantage to be able to restrict ourselves only to point forecasts without losing any decision-making ability compared to when making distributional forecasts. This is because point forecasts require far fewer parameters to predict. Moreover, there may be an exponentially large number of possible scenarios, making the nominal optimization problem intractable with

distributional forecasts. In contrast, with the right forecast via end-to-end learning, the nominal problem is an easy to solve deterministic model with the point forecast.

A large class of resource management problems where over-utilization and under-utilization are both penalized fall in this category. Note that specifically for the unconstrained newsvendor problem, the work of Ban and Rudin (2019) shows that quantile regression based point forecasts are sufficient. As additional examples, the capacitated newsvendor problem and the multi-location cross-fulfilment newsvendor problem both fall under this category of loss-type functions. In this paper, we present computational experiments on both problems.

3. Approximation Framework

We find that we have two competing objectives when choosing some approximation $\hat{w}$: (1) the objective that $\hat{w}(u)$ is close to $w^*(u)$, and (2) that $\hat{w}(u)$ can be computed significantly more quickly than $w^*(u)$. At a high level, our approach consists of learning to solve a different version of the original optimization problem, one which is faster to solve and provides a solution close to that of the original.

First, we formalize what we mean by being able to compute $\hat{w}$ "faster." This is highly dependent on the method used to solve a given optimization problem. Here we will consider iterative interior point methods. And an approximation $\hat{w}$ is computed "faster" depending on the number of iterations needed. We consider the following sets of algorithms to produce approximations $\hat{w}$:

$$\hat{w}_{r,T}(u) = w_T, \tag{22}$$

$$w_{t+1} = \pi_{\mathcal{P}}\left(w_t - \eta \cdot R(u, w_t)\right), \quad t = 1, \ldots, T-1 \tag{23}$$

$$\text{initialized } w_0 \in \mathcal{P} \tag{24}$$

where $\pi_{\mathcal{P}}$ is the projection operator onto the convex feasible region $\mathcal{P}$, and $R(u, w)$ is any update rule. For example, if $R(u, w_t) = \nabla g_u(w_t)$, the above algorithm reduces to projected gradient descent. Finally, let $\hat{w}_R(u)$ denote the limit of this sequence $\hat{w}_{R,T}(u)$ as $T \to \infty$. Now, we can restate the two objectives when choosing the approximation function $\hat{w}$:

1. Minimize the regret of $\hat{w}_R(u)$ compared to $w^*(u)$. Specifically, minimize $\mathbb{E}[g_u(\hat{w}_R(u)) - g_u(w^*(u))]$.
2. Maximize the convergence rate of $\hat{w}_{R,T}(u)$ to $\hat{w}_R(u)$.

Now we gain some more intuition on the effect the update rule R has on the convergence of $\hat{w}_R(u)$. Suppose there exists some function $r_u(w)$ for which $\nabla r_u(w) = R(u, w_t)$. Then, $\hat{w}_{R,T}(w)$ essentially performs gradient descent to minimize the function $r_u(w)$ over $w \in \mathcal{P}$. From an analysis point of view, we can instead consider the problem

$$\arg\min_{w \in \mathcal{P}} r_u(w) \tag{25}$$

and now the point of convergence $\hat{w}_R(u)$ is the solution to the above problem.

In this paper we will make the choice that the update rule is simply the gradient of g_u plus some additional linear term $L(u)$. Therefore, we will choose

$$R(u,w) = \nabla g_u(w) + \frac{\gamma}{\eta} \cdot L(u)w \tag{26}$$

where $L(u)$ is a matrix, possibly dependent on u. The update step then takes the form $\hat{w}_{t+1} = \pi_{\mathcal{P}}\left(\hat{w}_t - \eta \nabla g_u(w) - \gamma \cdot L(u)w\right)$. For ease of notation, we let $\hat{w}_{L,T}(u)$ denote the sequence of points using this update rule and $\hat{w}_L(u)$ be the point of convergence (assuming it exists, more on this in the next section). From this, we see that $\hat{w}_L$ is equal to

$$\hat{w}_L(u) = \arg\min_{w \in \mathcal{P}} g_u(w) + \frac{\gamma}{\eta} w^T L(u) w. \tag{27}$$

3.1. Analytical properties

First, note that the matrix $L(u)$ needs to be positive semidefinite in order for the problem in (27) to be convex as well as for the sequence to converge properly. In terms of analysis, we will assume that $L(u)$ is positive semidefinite. We will see algorithmically how we might be able to ensure this in section 3.2.2.

This choice of update rule has some nice properties we can make use of to directly answer objectives (1) and (2) mentioned above. In terms of approximation error, we find the following.

Proposition 2. *The decision $\hat{w}_L(u)$ when evaluated against the true solution $w^*(u)$ has regret*

$$\begin{aligned} g_u(\hat{w}_L(u)) - g_u(w^*(u)) &\leq \frac{\gamma}{\eta}(w^*(u))^T L w^*(u) && (28)\\ &\leq \frac{\gamma}{\eta}\sigma_{\max}(L) \cdot D^2 && (29) \end{aligned}$$

where $\sigma_{\max}(L)$ is the largest eigenvalue of L and D is the diameter of the feasible region $\mathcal{P}$.

Proof. This follows directly from the optimality of $\hat{w}_L(u)$ with respect to the objective $g_u(w) + \gamma/\eta\ w^T L w$. So, its objective value is lower than the objective value of $w^*(u)$:

$$\begin{aligned} g_u(\hat{w}(u)) + \frac{\gamma}{\eta}\hat{w}(u)^T L \hat{w}(u) &\leq g_u(w^*(u)) + \frac{\gamma}{\eta} w^*(u) L w^*(u) && (30)\\ g_u(\hat{w}(u)) - g_u(w^*(u)) &\leq \frac{\gamma}{\eta} w^*(u) L w^*(u) - \frac{\gamma}{\eta}\hat{w}(u)^T L \hat{w}(u) && (31)\\ g_u(\hat{w}(u)) - g_u(w^*(u)) &\leq \frac{\gamma}{\eta} w^*(u) L w^*(u) && (32) \end{aligned}$$

Finally, a standard property of eigenvalues states that $\max_{\|w\|=1} w^T L w = \sigma_{\max}(L)$. Therefore, since $w \in \mathcal{P}$ and hence $\|w\| \leq D$, it follows that $\max_{w \in \mathcal{P}} w^T L w \leq \sigma_{\max}(L) \cdot D^2$ □

And in terms of convergence rate, given g has α-Lipschitz gradient, we converge exponentially as follows.

Proposition 3. *The sequence $w_1, \ldots, w_t, \ldots$ converges to $\hat{w}_L(u)$ at an exponential rate $O(\exp(-t\sigma_{\min}(L)/(\alpha + \sigma_{\max}(L)))$. where $\sigma_{\min}(L), \sigma_{\max}(L)$ are the smallest and largest eigenvalues of L, respectively, and g_u has α-Lipschitz gradient (with respect to w).*

Proof. We will rely on traditional results in convex optimization for convergence rates for strongly convex and smooth convex functions. Specifically, for μ-strongly convex and β-smooth convex functions $r_u(w)$, a sequence $w_{t+1} = w_t - 1/\beta \nabla r_u(w_t)$ will converge as

$$r_u(w_t) - r_u(w^*(u)) \leq \left(1 - \frac{\mu}{\beta}\right)^t (r_u(w_0) - r_u(w^*(u))) . \tag{33}$$

Therefore, it suffices to show that $\mu \geq \sigma_{\min}(L)$ and $\beta \leq \sigma_{\max}(L) + \alpha$ for $r_u(w) = g_u(w) + \gamma/\eta w^T L w$.

We prove strong convexity first. r_u is μ-strongly convex if and only if $\nabla^2 r_u$ □

Putting these two propositions together suggests learning the function $L(u)$ which minimizes the upper bound on regret in proposition 2 on the data available while keeping the eigenvalues of $L(u)$ bounded as in proposition 3. Consider solving

$$\begin{aligned} \min_L \ & \sum_{n=1}^N w^*(u^n)^T L(u^n) w^*(u_n) \\ \text{s.t. } & \sigma_{\min}(L(u^n)) \geq \underline{\lambda}, \quad \forall n, \\ & \sigma_{\max}(L(u^n)) \leq \overline{\lambda}, \quad \forall n. \end{aligned} \tag{34}$$

While appealing, this formulation has several drawbacks. From a computational point of view, this is a difficult problem. If $L(u)$ is a linear mapping, the problem is a linear semidefinite problem. However, this only ensures that the model $L(u)$ outputs a positive semidefinite (PSD) matrix only on the input data, and not necessarily out-of-sample. If $L(u)$ is a constant function, that is we always use the same matrix independently of u, this problem becomes more tractable.

Aside from computational complexity, there is another issue with the above. In practice, we will not perform enough iterations to converge, and for the sake of speed, will only use relatively few iterations. While ensuring the rate of convergence is useful, it leaves a lot missing when trying to calculate the regret after a small number of iterations, say only 5 or 10 update iterations. Practically, a more useful problem is to learn $L(u)$ by solving

$$\min_L \sum_{n=1}^N g_{u^n}(\hat{w}_{L(u^n),T}(u^n)). \tag{35}$$

Moreover, we can bound the generalization gap of learning L from data. Suppose we are given a dataset of N i.i.d. samples. Let $C(L)$ be the expected cost, and $\hat{C}(L)$ the empirical cost:

$$C(L) = \mathbb{E}_u[g_u(\hat{w}_{L(u),T}(u)] \tag{36}$$

$$\hat{C}(L) = \frac{1}{N} \sum_{n=1}^N g_{u^n}(\hat{w}_{L(u^n),T}(u^n)) \tag{37}$$

Now suppose we are learning the function $L(u)$ from a hypothesis class $\mathcal{L}$. This is a set of functions mapping $u \in \mathbb{R}^d$ to a matrix $L \in \mathbb{R}^{d\times d}$. Essentially, a mapping $\mathbb{R}^d \to \mathbb{R}^{d^2}$. Rademacher complexity aims to define the complexity of this set of functions $\mathcal{L}$.

DEFINITION 1. [Multidimensional Rademacher Complexity]The empirical Rademacher complexity of the hypothesis class of function $\mathcal{L}$ from $\mathbb{R}^d \to \mathbb{R}^{d^2}$ is given by

$$\mathcal{R}_N(\mathcal{L}) = \mathbb{E}_{u^1,\ldots,u^N}\mathbb{E}_\sigma\left[\sup_{L\in\mathcal{L}}\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{d^2}\sigma_{nk}L_k(u^n)\right] \tag{38}$$

where σ_{nk} are i.i.d. variables uniformly sampled from $\{-1,1\}$ (also known as Rademacher random variables).

THEOREM 2. *[Generalization bound] With probability* $1-\delta$*, for any function* $L \in \mathcal{L}$*,*

$$C(L) \le \hat{C}(L) + T\eta\alpha\cdot\mathcal{R}_N(\mathcal{L}) + \sqrt{\frac{\log(1/\delta)}{N}} \tag{39}$$

where $g_u(w)$ *is* α*-Lipschitz (with respect to* w*) and* $\mathcal{R}_N(\mathcal{L})$ *denotes the rademacher complexity of the hypothesis class* $(\mathcal{L})$*.*

For many hypothesis classes $\mathcal{L}$, we can bound $\mathcal{R}_N(\mathcal{L})$ by a term that converges to 0 as $N\to\infty$ and at a rate $O(1/\sqrt{N})$ for common function classes like linear functions. See for example Bartlett and Mendelson (2002).

Approximation framework. The approximation $\hat{w}(u)$ is generated by T iterations of the update rule given by

$$\hat{w}_{t+1} = \pi_{\mathcal{P}}\left(\hat{w}_t - \eta\nabla g_u(w) - \gamma\cdot L(u)w\right) \tag{40}$$

for $t=0,\ldots,T-1$, choosing $\hat{w}_0=0$, $\pi_{\mathcal{P}}$ the projection operator on the feasible region $\mathcal{P}$ and $L(u)$ a positive semidefinite matrix. We learn such an $L(u)$ by solving (35)

$$\min_{L\in\mathcal{L}}\sum_{n=1}^{N} g_{u^n}(\hat{w}_{L(u^n),T}(u^n)). \tag{41}$$

where $\mathcal{L}$ defines the set of possible functions L. As examples in this paper, we will consider $\mathcal{L}$ to be the set of constant functions (that is, $L(u)$ is a constant matrix independent of u), and the set of linear functions of u.

3.2. ProjectNet: tractable algorithms

We propose tractable learning methods of solving the problems introduced in the previous section. This consists of addressing two key points. (1) We present a differentiable method of projection onto linear constraints and (2) we present a method to control the eigenvalues of the matrix $L(u)$. We denote this model as ProjectNet. Finally, (3) we integrate the ProjectNet model into the end-to-end framework.

3.2.1. Ensuring Feasibility First, we discuss how to perform the projection operator $\pi_{\mathcal{P}}$ as part of computing $\hat{w}_{r,T}(u)$. Projection itself is a difficult optimization problem given by $\pi(w) = \arg\min_{y\in\mathcal{P}} \|w-y\|^2$. We resolve this issue by performing an approximate projection as follows. We only assume that projection onto a single constraint can be done through a differentiable method. We use Dykstra's projection algorithm (Dykstra 1983) that provides a sequence of differentiable steps to approximate the projection. Let $\mathcal{P}_1,\dots,\mathcal{P}_J$ be any J intersecting convex sets that make up the feasible region. For example, $\mathcal{P}_i = \{w : h_i(w) \le 0\}, \mathcal{P}_{j+p_1} = \{l_j(w) = 0\}$. We cyclically project onto $\mathcal{P}_1$ through $\mathcal{P}_J$ until we reach some desired accuracy (distance from satisfying both constraints). We define π_j as the projection operators onto sets $\mathcal{P}_j$. After k steps of the iterative projection, we reach an approximation $\tilde{\pi}^k$ defined in Algorithm 1. The only requirement is that each individual projection π_j can be done by a differentiable method. That is, one must be able to compute $\partial\pi_j(w)/\partial(w)$. For example, for linear optimization problem, we let $\mathcal{P}_1 = \{w : Aw = b\}, \mathcal{P}_2 = \{w : w \ge 0\}$, so that $\mathcal{P} = \mathcal{P}_1 \cap \mathcal{P}_2$. The projections π_1, π_2 can be evaluated easily as follows:

$$\pi_1(w) = \arg\min_{y:Ay=b} \|w-y\|_2^2 \tag{42}$$

$$= w - A^T(AA^T)^{-1}(Aw-b) \tag{43}$$

$$\pi_2(w) = \arg\min_{y\ge 0} \|w-y\|_2^2 = ReLu(w). \tag{44}$$

In the case of linear subspaces, the method simplifies to $\tilde{\pi}^k(w) = \pi_2(\pi_1(\dots(\pi_2(\pi_1((w))\dots))$. This is depicted in Fig. 3. For example, in the case of a polyhedral feasible region, the sequence of points w_k is guaranteed to converge to a point in $\mathcal{P} = \mathcal{P}_1 \cap \mathcal{P}_2$ at a geometric rate. In particular, Deutsch and Hundal (1994) show that there exists $\rho < 1, a > 0$ so that for any integer k:

$$\left\|\tilde{\pi}^k(w) - \pi(w)\right\|_2 \le a\cdot\rho^k, \tag{45}$$

where $\pi(w)$ is the exact projection of w onto the feasible region $\mathcal{P}$. However, as the sequence of projections converges closer to a vertex, the corresponding gradients $\partial\tilde{\pi}^k(w)/\partial w$ will also approach zero. Indeed, in the limit, if we have exact projections onto vertices of the feasible polytope, then the gradient is zero. This suggests that one must be careful when choosing the number of iterations k so that they are large enough to provide good approximations, but at the same time consider the trade-off in keeping the corresponding gradients from becoming too small.

3.2.2. Controlling eigenvalues We now focus on controlling the eigenvalues of $L(u)$. We do so as follows. We propose any auxiliary neural network model which outputs two quantities, an upper triangular matrix $M(u)$ and a diagonal matrix $D(u)$. Then,

1. The resulting matrix $M(u)M(u)^T$ is symmetric and invertible as long as the diagonal entries of $M(u)$ are positive.

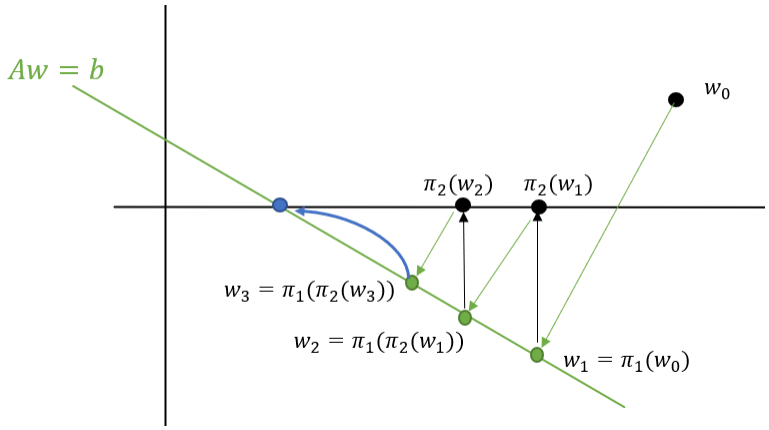


Figure 3 **Iterative projection method. The sequence of projections converges to a feasible solution (depicted as the black point).**

Algorithm 1 Dykstra's Projection Method

function $\tilde{\pi}^k(w)$
  Initialize $w_J^0 = w$, and $p_1^0 = \cdots = p_J^0 = 0$.
  for $t = 1, \ldots, k$ **do**
    $w_0^t = w_d^{t-1}$
    for $j = 1, \ldots, J$ **do**
      $w_j^t = \pi_j(w_{j-1}^t + z_j^{t-1})$
      $z_j^t = w_{j-1}^t + z_j^{t-1} - w_j^k$
  return w_J^k

2. The matrix $(M(u)M(u)^T)D(u)(M(u)M(u)^T)^{-1}$ exists and moreover its eigenvalues are equal to the diagonal entries of $D(u)$.
3. To control all eigenvalues to be between $\underline{\lambda}$ and $\overline{\lambda}$, we can apply the transformation $\rho(D(u)) =$. So, we can set

$$L(u) = (M(u)M(u)^T)\rho(D(u))(M(u)M(u)^T)^{-1}. \tag{46}$$

where again $M(u), D(u)$ can be generated from any choice of neural net architecture. The inverse operation is differentiable, and implemented in most existing software.

3.2.3. Model Architecture Recall from section 3.1 our goal is to learn a function $L(u)$ which outputs an update rule by solving

$$\min_{\theta} \sum_{n=1}^{N} g_{u^n}(\hat{w}_{L(u^n)}(u^n)). \tag{47}$$

To solve this, we make two approximations to make the formulation tractable. Instead of using $\hat{w}_L$, the point of convergence, we instead use $\hat{w}_{L,T}$ for some choice of T iterations. Moreover, instead

of using the exact projection π when computing a single step of the iterative process, we use $\tilde{\pi}_{\mathcal{P}}$ from section 3.2.1 instead. This can be seen in the architecture of the ProjectNet in figure 4.

Improvement over gradient descent. We now show the runtime improvements of our proposed approach compared to traditional projected gradient descent. For ease of clarity, we add the full details of the setup in Appendix A.1. We present computational results comparing the two approaches on a maximum matching problem with $n = 50$ nodes and $n^2 = 2,500$ edges/variables.

We also train a ProjectNet model with $T_0 = 5$ iterations, and compare the objective value of its solution for iterations up to $T_1 = 35$ on testing data. See figure 5a. In particular, we measure the average relative regret of decisions. That is, given realized edge weights u and decision $\hat{w}$, the relative regret is the percent difference in objective between the objective of $\hat{w}$ and the optimal decision in hindsight: $(g_u(\hat{w}) - g_u(w^*(u))/g_u(w^*(u)$. We see that indeed the ProjectNet method improves consistently in accuracy as the number of iterations T_1 is extended from the T_0 steps that were used during training.

Note that when compared to the traditional gradient descent approach, the ProjectNet approach performs better using fewer iterations. It maintains this edge even for steps $T_1 > T_0$, which it has not trained upon. However for larger T_1 traditional gradient descent is better able to converge to the optimal solution and it overtakes the ProjectNet method. But for the end-to-end framework it is beneficial to use a smaller number of iterations T_1 since this is computationally more efficient, and keeps the gradient $\nabla\hat{w}(u)$ from approaching zero. In the regime of smaller T_1, the ProjectNet method also has an advantage in terms of objective function value, with up to 12.5% improvement. See Figure 5b.

3.3. End-to-End Learning via ProjectNet

First, ProjectNet is trained to learn solutions to the optimization problem $w^*(u)$. In particular, given some cost vectors $u^1, \ldots, u^N$, we aim to learn some $\hat{w}$ parametrized by the update linear layer L which minimizes empirical cost as introduced in (41).

Note that we never need to solve the nominal optimization problem $w^*(u^n)$ during this process. In addition, we may use any data $u^1, \ldots, u^n$ that we wish, not necessarily only the vectors from the

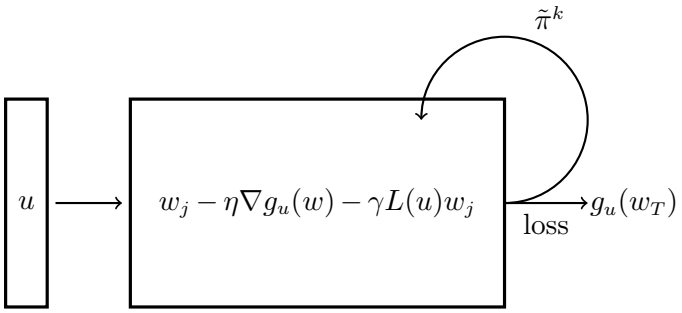
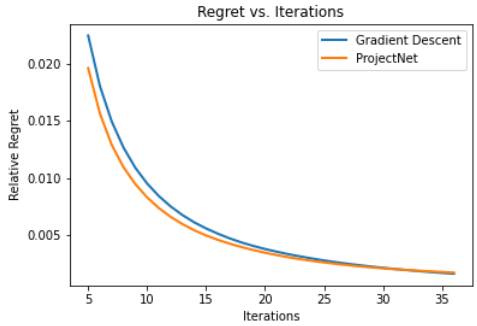
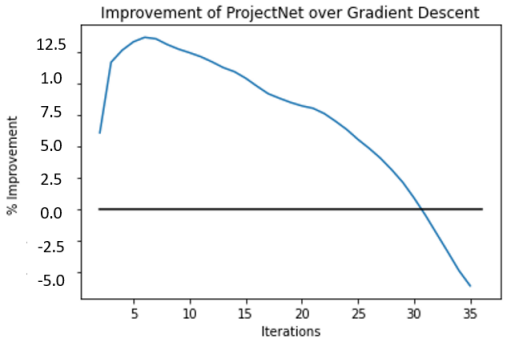


Figure 4 ProjectNet architecture



(a) Regret of ProjectNet compared to gradient descent as iterations T increase.



(b) Percent improvement in relative regret of the ProjectNet model compared to gradient descent.

Figure 5 Comparison of ProjectNet to Gradient Descent

Algorithm 2 End-to-End Learning via ProjectNet

function ProjectNet End-to-End$((x^1,u^1),\ldots,(x^N,u^N))$

$\hat{w}(\cdot) \leftarrow$ TrainProjectNet()

Initialize θ at random.

for each epoch **do**

for $n=1,\ldots,N$ **do**

Compute $\nabla_\theta g_{u^n}(\hat{w}(f_\theta(x_N)))$.

Update θ by any gradient method.

return θ

original training data of (x^n,u^n). We can train using, say, T_0 iterations of the recurrent network. The entire end-to-end method to learn forecasts $f_\theta(\cdot)$ is now described in Algorithm 2 which follows the steps in diagram 2 shown earlier. In short, we make a forecast $f_\theta(x^n)$ and differentiate through the approximate corresponding solution $\hat{w}(f_\theta(x^n))$ to update θ. During the training step of ProjectNet, we may use T_0 iterations, while when subsequently evaluating $\hat{w}(f_\theta(x_n))$, we may use any $T_1 \geq T_0$ iterations to improve the accuracy of the ProjectNet's approximation.

4. Extension to Two-Stage Stochastic Optimization

We now consider two-stage linear stochastic optimization problems. Let w denote the first-stage decision and let $V(w,u)$ denote the second-stage cost of decision w under the realization u of uncertainty. As an example, we can consider a multi-warehouse cross-fulfillment newsvendor problem. The first-stage decision w is the amount of product to allocate to each warehouse. After the decision is made, the demand u is realized, and finally one must fulfill the demand using the initial stocking decision. This would correspond to $V(w,u)$ being a form of a matching problem (which warehouse should fulfill which client). For details and computational results, see section 5.1.

In general we assume the first stage problem can be formulated as

$$\begin{aligned} w^*(D) = \arg\min_w \quad & c^T w + \mathbb{E}_{u\sim D}\left[V(w,u)\right] \\ \text{subject to} \quad & Aw = b \\ & w \geq 0. \end{aligned} \tag{48}$$

As for the second stage problem, we allow the uncertainty to impact either the objective or the constraints, and these could depend on the decision w taken in the first stage as well. The problem takes the general form as below, where w is the second-stage decision variable, $T(w,u)$ is some matrix which depends on the first-stage decision w and on the realization u:

$$\begin{aligned} V(w,u) = \min_v \quad & d(w,u)^T v \\ \text{subject to} \quad & T(w,u)\begin{bmatrix} w \\ v \end{bmatrix} = h(w,u) \\ & v \geq 0. \end{aligned} \tag{49}$$

Assumption 1. *We assume the model has relatively complete recourse.*

That is, any feasible w in the first stage leads to a feasible second stage problem for any uncertainty realization. This assumption is often satisfied, for example, this is the case in the traditional and cross-fulfilment newsvendor problem (see for example Shapiro and Philpott (2007), Birge and Louveaux (2011)).

Suppose we observe features x^n and corresponding realizations of uncertainty u^n, for N data points, $n = 1, \ldots, N$. Given a point forecast $f_\theta(x^n)$, the corresponding decision is $w^*(f_\theta(x^n))$. Afterwards, the value of the uncertainty u^n is realized and we can determine the cost to be $Z(w^*(f_\theta(x^n)), u^n)$, where $Z(w,u) := c^T w + V(w,u)$. Hence, the cost minimization problem to learn θ is as follows

$$\min_\theta \sum_{n=1}^N Z(w^*(f_\theta(x_n)), u_n) = \min_\theta \sum_{n=1}^N c^T(w^*(f_\theta(x_n))) + V(w^*(f_\theta(x_n)), u_n). \tag{50}$$

An additional difficulty in solving this problem is that there are nested optimization problems. That is, computing $w^*(f_\theta(x))$ and passing its solution as input to problem V. Therefore, to further simplify this problem, consider making a first-stage decision w directly from data by some $q_\vartheta(x)$ instead of making an intermediate forecast. Note that this problem differs from the learning problem in the previous section. Here, the goal is to learn a decision rule $q_\vartheta(x)$, whereas previously we made intermediate forecasts $f_\theta(x)$, which was then used to solve the single stage optimization problem of interest to obtain $w^*(f_\theta(x))$. We then have

$$\min_\vartheta \sum_{n=1}^N c^T q_\vartheta(x_n) + V(q_\vartheta(x_n), u_n). \tag{51}$$

The difficulty now lies in taking the gradient $\nabla_w Z(w,u) = c + \nabla_w V(w,u)$, since V is a complex optimization problem. Therefore, we propose to use ProjectNet to learn this $V(w,u)$. Specifically, it learns the second-stage decisions $\hat{v}$ in problem (49). Then, we can approximate $V(w,u)$ by $d(w,u)^T\hat{v}$. Notice that this problem of approximating $\hat{v}$, is the same as the ProjectNet problem described for the deterministic problem in 1. Finally, we must ensure that the decisions $q_\vartheta(x)$ satisfy first-stage feasibility constraints $Aq_\vartheta(x_n) = b, q_\vartheta(x_n) \geq 0$. This may be accomplished by applying our approximate projection method.

The primary difference in this case compared to the initial single-stage problem we presented in section 2 is that the uncertainty can lie in the constraints. However, this does not pose any problems, as the output of our proposed ProjectNet architecture is still differentiable with respect to parameters in the constraints. First, let us rewrite the update function of the ProjectNet for the second-stage problem. In this case, we aim to learn the optimal second-stage variables v and recall the objective function is linear $d(w,u)^T v$, so the gradient is always $d(w,u)$. Finally, we have

$$v_{j+1} = \tilde{\pi}^k \left(v_j - \eta d(w,u) - \gamma L(v_j)\right)$$

and we notice that the constraints only appear in the approximate projection $\tilde{\pi}^k$. From Algorithm 1 and Equations (42) one can see $\tilde{\pi}^k$ is a sequence of steps differentiable with respect to the constraints. Hence, the gradient of the approximation $d(w,u)^T v_j$ is always well-defined.

Point Forecasts vs. Distributional Forecasts In the same spirit as for single-stage problems, it is sufficient to make point forecasts with an end-to-end approach when the objective $Z(\cdot,\cdot)$ is a loss-type function:

Proposition 4. *Consider a two-stage stochastic optimization problem as described in section 4, with loss-type objective function $Z(w,u)$. That is, $Z(w,w) = 0$ for feasible w. Then, for any distributional forecast, there exists a single point forecast that produces the same solution. In other words, for any distribution D, there exists a single point forecast d so that*

$$\arg\min_w \mathbb{E}_{u\sim D}[Z(w,u)] = \arg\min_w Z(w,d). \tag{52}$$

Proof. Let w^* be the solution to the problem using distributional forecast D:

$$w^* = \arg\min_w \mathbb{E}_{u\sim D}[Z(w,u)]. \tag{53}$$

Now consider making a forecast of exactly $d = w^*$. Then, $w^* = \arg\min_w Z(w,d)$, since $Z(\cdot,\cdot)$ is a loss function (i.e., the minimum is achieved at $Z(w,w)$). □

5. Computational Results

In this section we present computational results illustrating that the ProjectNet method introduced in this paper is effective in several end-to-end learning settings. We show this on several tasks: (1) a two-stage multi-warehouse cross-fulfillment newsvendor problem in which the first stage consists of allocating supply to many warehouses, and the second consists of optimally fulfilling the realized demand, (2) a real-world electricity planning problem (3) a shortest path problem in which the forecasting step is a computer vision task of predicting edge costs from terrain maps. Moreover, we include additional synthetic experiments on several variations of the multi-item newsvendor problem having linear or quadratic costs and capacity constraints in Appendix A.1.

5.1. Multi-warehouse cross-fulfillment newsvendor

As an illustration of a two-stage stochastic optimization problem, we consider a multi-warehouse newsvendor problem with cross-fulfillment. We consider a setting of n warehouses and m clients with unknown future demand. In the first stage, one must decide on the amount of product to allocate to each individual warehouse. In the second stage, the demand at each client is realized and one must determine the optimal plan to fulfil the demand given the decisions made. In particular, there are traveling unit costs c_{ij} to transport a unit of product from warehouse i to client j. For every unit of unmet demand at client j, there is a backorder unit cost of b_j and for every unit of product left unused at a warehouse i there is a holding unit cost of h_i.

Let w_i denote the first-stage decision of amount of product to allocate at warehouse i and $V(w,d)$ the minimum cost of fulfilling a demand of $d=(d_1,\ldots,d_m)$ for the clients.

$$\begin{aligned} V(w,d) = \min_{v\geq 0} \quad & \textstyle\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}v_{ij} + \sum_{j=1}^{m} b_j \left(d_j - \sum_{i=1}^{n} v_{ij}\right)^+ + \sum_{i=1}^{n} h_i \left(\sum_{i=1}^{n} v_{ij} - s_i\right)^+ \\ \text{subject to} \quad & \textstyle\sum_{i=1}^{n} v_{ij} \leq w_i \end{aligned} \tag{54}$$

Finally, there is a unit cost of c of allocating a single product to any warehouse. Hence, the cost of the first and second stages when making decision w against future realization of d is

$$Z(w,d) = c\cdot\sum_{i=1}^{n} w_i + V(w,d) \tag{55}$$

We assume we are given data $(x^1,d^1),\ldots,(x^N,d^N)$ consisting of observed features x^n and corresponding demand realization of d^n. We generate this data as follows. Each x^n is drawn from a normal gaussian distribution, and d^n is given by a deterministic quadratic function of x^n. In particular, $(d^n)_j = (q^T x^n)_j^2$ for a fixed vector q which is initially generated at random. To learn the decision rule $f_\theta(x)$, we solve

$$\min_{\theta} \sum_{n=1}^{N} Z(f_\theta(x^n), d^n). \tag{56}$$

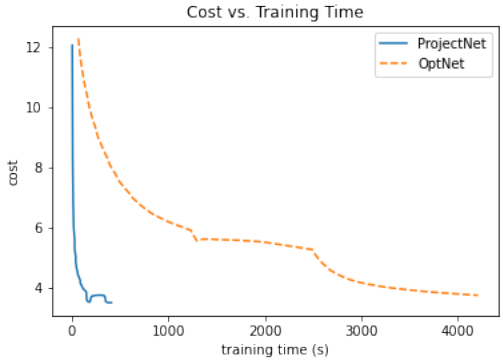


Figure 6 Task-based cost as a function of training time in the cross-fulfillment problem with 40 nodes.

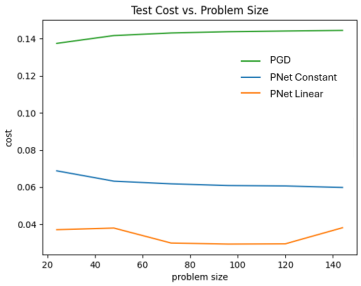
The primary difficulty lies in calculating $V(w,d)$ and the gradient $\partial V(w,d)/\partial w$. We use ProjectNet to approximate these.

We compare with the traditional predict then optimize method which separates the prediction and optimization, as well as the end-to-end OptNet method (Amos and Kolter 2017). For the predict-then-optimize method we simply learn a forecasting model which aims to minimize the mean-squared error. Since the demand distribution is a deterministic function of the features, this method alone would retrieve the optimal solution given infinite data. Given there will inherently be some error in the model, this will not happen and we see in the experiments that the end-to-end methods greatly outperform this approach. We adopt a similar decision rule idea for two-stage decision problems that we proposed in Section 4 for the ProjectNet method in the case of the OptNet method. We use the following formulation to calculate the fulfilment cost in the OptNet method after the allocation decision w has been made and the demand d is realized:

$$\begin{aligned} V_{\text{optnet}}(w,d) = \min_{v,q,p\geq 0} \quad & \textstyle\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}v_{ij} + \sum_{j=1}^{m} b_j q_j + \sum_{i=1}^{n} h_i p_j + \alpha(\|v\|^2 + \|p\|^2 + \|q\|^2) \\ \text{subject to} \quad & q_j \geq d_j - \textstyle\sum_{i=1}^{n} v_{ij}, \quad p_j \geq \sum_{i=1}^{n} v_{ij} - s_i, \quad \sum_{i=1}^{n} v_{ij} \leq w_i \end{aligned} \tag{57}$$

where again we introduce variables p, q to denote the amount of overstocked or understocked units in order to linearize the objective, respectively. Again, we choose the regularization term $\alpha = 0.01$.

In Table 1 we observe that both end-to-end methods clearly outperform the predict then optimize baseline in terms of accuracy. Yet again in this setting we observe a significant decrease in the training time for ProjectNet over Optnet, running more than twice as fast for a 20-location problem. As the problem size grows to double (40 locations), the running time of OptNet increases significantly, nearly twenty times, while the ProjectNet method only increased threefold. For the 40 location example, we see in Figure 6 the comparison of the cost of the decisions made by each approach as a function of the training time.



(a) Problem size cost comparison.

Size	ProjectNet	ProjectNet Linear	CVXPY	OptNet
24	0.12	0.09	2.32	1.54
48	0.103	0.11	2.64	3.14
72	0.108	0.164	2.59	8.41
96	0.11	0.33	2.59	19.06
120	0.10	0.18	2.67	21.78
144	0.07	0.27	3.19	35.47

(b) Running time (in seconds) of each approach with increasing problem size.

Figure 7 ProjecetNet accuracy and runtime on electricity scheduling.

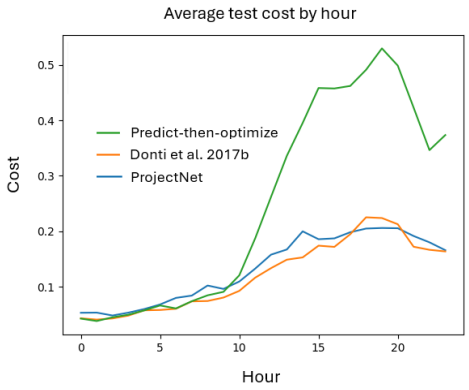
5.2. Electricity planning

We now consider an electricity generation and planning problem using data from PJM, an electricity routing company coordinating the movement of electricity throughout 13 states. Our objective is to plan electricity generation over the next 24 hours of the data. The operator incurs a unit cost γ_e for excess generation and a cost γ_s for shortages. The cost of generating $w_1, \ldots, w_{24}$ while true demand is $u_1, \ldots, u_{24}$ is given by $g_u(w) = \sum_{i=1}^{24} \gamma_s \max\{u_i - w_i, 0\} + \gamma_e \max\{w_i - u_i, 0\} + 1/2(w_i - u_i)^2$. Moreover, there are additional ramp-up constraints, that the generation from one hour to the next cannot differ by more than $r = 0.4$. The constraints are given by $|w_{i+1} - w_i| \leq r,\ i = 1, \ldots 23$ and $w_i \geq 0$.

ProjectNet accuracy and runtime We first focus on the accuracy of the ProjectNet model to approximate the optimization problem as well as the runtime required. We compare against traditional projected gradient descent (PGD). In addition, we consider two versions of ProjectNet, one where $L(u)$ is a constant function (using the same matrix L for all u, we denote this PNet Constant in the table) and where $L(u)$ is a linear function of u (we denote this as ProjectNet Linear in the table). We also compare against the runtime of OptNet and the CVXPY layer developed in Agrawal et al. (2019). We perform experiments along multiple axes. First, as we increase the amount of training data available, and second as we increase the size of the optimization problem. We increase the size by increasing the planning horizon from one day up to 5 days (hence, having to solve an optimization problem from 24 to 120 variables). All methods (our two versions of ProjectNet, Projected Gradient Descent (PGD), OptNet, and the CVXPY approach) will run 10 update iterations for its approximations.

# locations	Predict-then-Optimize	ProjectNet	OptNet
20	7.58	2.94 / 27s	2.93 / 60s
40	9.95	3.54 / 88.6s	3.52 / 1200s

Table 1 Cross-fulfilment newsvendor results. Recording average cost on test set and average running time per epoch.



(a) Average test cost by hour on electricity problem.

	Runtime per epoch
ProjectNet	1.45 (s)
Donti et al. (2017a)	5.33 (s)

(b) Running time (in seconds) of each approach with increasing problem size.

Figure 8 ProjecetNet accuracy and runtime on electricity scheduling.

End-to-end results Finally, we implement the ProjectNet into the end-to-end framework to train a model to make predictions and corresponding decisions. We use a two-layer (each layer of width 200) network with an additional residual connection from the input to the output layer. In addition, we also perform data augmentation, creating new features like non-linear functions of the temperature, one-hot-encodings of holidays and weekends, and yearly sinusoidal features. The setup is similar to that used in Donti et al. (2017b). The same model and data are used for all models. Hyperparameters are chosen to be the same as parameters chosen from the original paper that introduced the dataset and the benchmark method. We show the average cost incurred by each method during each hour of the day during the last year of data which we use as testing data. See Figure 8a. While the ProjectNet-based method incurs a small increase in cost (less than 5%) it is significantly faster to train as seen in Table 7b and 8b.

5.3. Warcraft Shortest Path

We use the Warcraft II tile dataset (Guyomarch 2017) which was first introduced in (Vlastelica et al. 2020) to test their end-to-end approach for combinatorial problems On this dataset, we compare our end-to-end method against their method as well as with a traditional two step predict then optimize method. The task consists of predicting costs of travelling over a terrain map and subsequently determining the shortest path between two points. In particular, each datapoint consists of a terrain map defined by a 12×12 grid where each vertex represents the terrain with a fixed unknown cost. The forecasting aspect is to determine the vertex weights given such an image, and the optimization aspect is to determine the shortest path from the top left to bottom right vertices. See figure 9 (top left) for a sample of terrain tiles.

The nominal shortest path problem can be formulated as follows, where $w_{i,j}$ is a variable deciding if edge from node i to node j should be chosen, $u_{i,j}$ is the cost of choosing edge from node i to j,

and for simplicity $O(i)$ is the set of edges leaving node i and $I(i)$ is the set of incoming edges into node i. Finally, the path begins at node a and ends at node b:

$$\begin{aligned} w^*(u) = \arg\min_{i,j} \quad & u_{i,j} w_{i,j} \\ \text{subject to } & \textstyle\sum_{j \in I(i)} w_{j,i} - \sum_{j \in O(i)} w_{i,j} = 0, \forall i \neq a, b \\ & \textstyle\sum_{j \in O(a)} w_{a,i} = 1, \quad \sum_{i \in I(b)} w_{i,b} = 1 \end{aligned} \tag{58}$$

Figure 9 (bottom left) illustrates an example of the path learned by the ProjectNet method, with the bottom right figure illustrating the true shortest path given perfect knowledge of vertex weights.

The forecasting task is much more complex in this example than the previous ones. In particular, it is a computer vision problem to learn vertex weights given an image. It is always important to choose the right forecasting model dependent on the problem at hand. A widely used model for computer vision is the so-called residual network (He et al. 2016). Traditionally, this is a deep network with 18 layers. In contrast, as in Vlastelica et al. (2020), we used only the first 5 layers of the architecture's structure for all experiments. The baseline model is a two-stage predict-then-optimize method which first trains a model by training on minimizing mean-squared error in predicting edge weights, then independently optimizing to find the shortest path. This method performs significantly worse than the end-to-end approaches. Hyperparameters are chosen to be the same as parameters chosen from the original paper that introduced the dataset and the benchmark method.

Comparing as in Vlastelica et al. (2020), we report the percentage of test instances for which various methods found an optimal path in Table 2. We can see the ProjectNet method's accuracy is

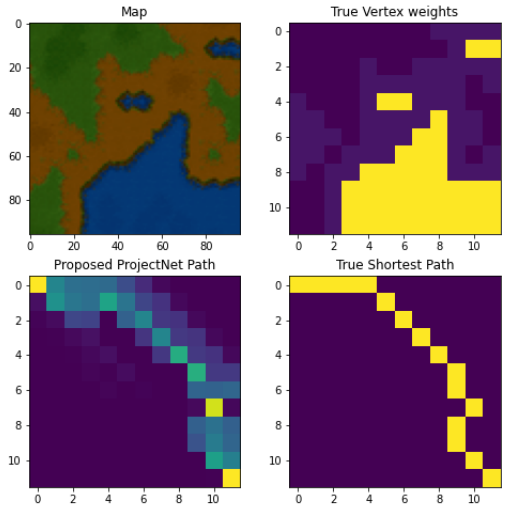


Figure 9 Sample terrain and path proposed by ProjectNet.

competitive and more crucially, the running time of our approach is 19% faster than the end-to-end method of (Vlastelica et al. 2020).

Table 2 Percentage of testing data for which optimal path was found on the warcraft shortest path problem. Runtime reports average running time in seconds per epoch.

Method	Matches	Runtime
ResNet Baseline	40.2%	9.2s
Vlastelica et al. (2020)	86.6%	81.3s
ProjectNet	83.0%	68.3s

Conclusions In this paper we studied the optimization under uncertainty problem. The traditional approach for tackling such a problem with uncertainty is the predict then optimize approach (e.g., first perform the prediction tasks, and then use these forecasts as inputs for downstream optimization problem). Rather in this paper we proposed a tractable end-to-end learning approach. We introduced a novel method to solve the end-to-end learning problem by introducing a novel neural network based method. The proposed approach learns to approximately solve an easier underlying optimization problem, ensuring its output satisfies the feasibility constraints. We established analytical results that justify our modelling choices. Furthermore, we applied this end-to-end learning approach to various supply chain problems, as well as to maximum matching and shortest path problems. We have shown in computational experiments that the ProjectNet method is computationally more efficient than other end-to-end methods while still being competitive in terms of task-based loss against other existing end-to-end methods.

References

Agrawal A, Amos B, Barratt S, Boyd S, Diamond S, Kolter JZ (2019) Differentiable convex optimization layers. *Advances in neural information processing systems* 32.

Amos B, Kolter JZ (2017) OptNet: Differentiable optimization as a layer in neural networks. Precup D, Teh YW, eds., *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, 136–145 (PMLR), URL `https://proceedings.mlr.press/v70/amos17a.html`.

Andrychowicz M, Denil M, Gomez S, Hoffman MW, Pfau D, Schaul T, Shillingford B, De Freitas N (2016) Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems* 29.

Arrow KJ, Harris T, Marschak J (1951) Optimal inventory policy. *Econometrica: Journal of the Econometric Society* 250–272.

Ban GY, Rudin C (2019) The big data newsvendor: Practical insights from machine learning. *Oper. Res.* 67:90–108.

Bartlett PL, Mendelson S (2002) Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research* 3(Nov):463–482.

Bello I, Pham H, Le QV, Norouzi M, Bengio S (2016) Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940* .

Bello I, Zoph B, Vasudevan V, Le QV (2017) Neural optimizer search with reinforcement learning. *International Conference on Machine Learning*, 459–468 (PMLR).

Berthet Q, Blondel M, Teboul O, Cuturi M, Vert JP, Bach F (2020) Learning with differentiable perturbed optimizers. *ArXiv* abs/2002.08676.

Bertsimas D, Kallus N (2020) From predictive to prescriptive analytics. *Management Science* 66(3):1025–1044.

Birge JR, Louveaux F (2011) *Introduction to stochastic programming* (Springer Science & Business Media).

Cameron C, Hartford J, Lundy T, Leyton-Brown K (2021) The perils of learning before optimizing. *arXiv preprint arXiv:2106.10349* .

Chen Y, Hoffman MW, Colmenarejo SG, Denil M, Lillicrap TP, Botvinick M, Freitas N (2017) Learning to learn without gradient descent by gradient descent. *International Conference on Machine Learning*, 748–756 (PMLR).

Deutsch F, Hundal H (1994) The rate of convergence of dykstra's cyclic projections algorithm: The polyhedral case. *Numerical Functional Analysis and Optimization* 15(5-6):537–565, URL `http://dx.doi.org/10.1080/01630569408816580`.

Dobkin D, Lipton RJ, Reiss S (1979) Linear programming is log-space hard for p. *Information Processing Letters* 8(2):96–97, ISSN 0020-0190, URL `http://dx.doi.org/https://doi.org/10.1016/0020-0190(79)90152-2`.

Donti P, Amos B, Kolter JZ (2017a) Task-based end-to-end model learning in stochastic optimization. *Advances in neural information processing systems* 30.

Donti P, Kolter Z, Amos B (2017b) Task-based end-to-end model learning in stochastic optimization. *NIPS*.

Donti PL, Rolnick D, Kolter JZ (2021) DC3: A learning method for optimization with hard constraints. *CoRR* abs/2104.12225, URL `https://arxiv.org/abs/2104.12225`.

Dykstra RL (1983) An algorithm for restricted least squares regression. *Journal of the American Statistical Association* 78(384):837–842.

Elmachtoub A, Liang JCN, Mcnellis R (2020) Decision trees for decision-making under the predict-then-optimize framework. III HD, Singh A, eds., *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, 2858–2867 (PMLR), URL `https://proceedings.mlr.press/v119/elmachtoub20a.html`.

Elmachtoub AN, Grigas P (2022) Smart "predict, then optimize". *Management Science* 68(1):9–26.

Ferber A, Wilder B, Dilkina B, Tambe M (2020) Mipaal: Mixed integer program as a layer. *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 1504–1511.

Frerix T, Nießner M, Cremers D (2019) Linear inequality constraints for neural network activations. *CoRR* abs/1902.01785, URL `http://arxiv.org/abs/1902.01785`.

Guler AU, Demirovic E, Chan J, Bailey J, Leckie C, Stuckey PJ (2020) Divide and learn: A divide and conquer approach for predict+ optimize. *arXiv preprint arXiv:2012.02342* .

Guyomarch J (2017) Warcraft ii open-source map editor. URL `http://github.com/war2/war2edit`.

He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hopfield J, Tank D (1985) Neural computation of decisions in optimization problems. *Biological cybernetics* 52:141–52, URL `http://dx.doi.org/10.1007/BF00339943`.

Kao Yh, Roy B, Yan X (2009) Directed regression. Bengio Y, Schuurmans D, Lafferty J, Williams C, Culotta A, eds., *Advances in Neural Information Processing Systems*, volume 22 (Curran Associates, Inc.).

Li K, Malik J (2016) Learning to optimize. *arXiv preprint arXiv:1606.01885* .

Liu H, Grigas P (2022) Online contextual decision-making with a smart predict-then-optimize method. *arXiv preprint arXiv:2206.07316* .

Lv K, Jiang S, Li J (2017) Learning gradient descent: Better generalization and longer horizons. *International Conference on Machine Learning*, 2247–2255 (PMLR).

Mandi J, Guns T (2020) Interior point solving for lp-based prediction+optimisation. *ArXiv* abs/2010.13943.

Mandi J, Stuckey PJ, Guns T, et al. (2020) Smart predict-and-optimize for hard combinatorial optimization problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 1603–1610.

Oroojlooyjadid A, Snyder LV, Takáč M (2020) Applying deep learning to the newsvendor problem. *IISE Transactions* 52(4):444–463.

Paulus A, Rolínek M, Musil V, Amos B, Martius G (2021) Comboptnet: Fit the right np-hard problem by learning integer programming constraints. *International Conference on Machine Learning*, 8443–8453 (PMLR).

Petersen F, Borgelt C, Kuehne H, Deussen O (2021) Learning with algorithmic supervision via continuous relaxations. *Advances in Neural Information Processing Systems* 34:16520–16531.

Qi M, Shi Y, Qi Y, Ma C, Yuan R, Wu D, Shen ZJM (2020) A practical end-to-end inventory management model with deep learning. *Available at SSRN 3737780* .

Qiu G, Tanneau M, Van Hentenryck P (2024) Dual conic proxies for ac optimal power flow. *Electric Power Systems Research* 236:110661.

Sergio YCMWH, Colmenarejo G (2016) Learning to learn for global optimization of black box functions. *stat* 1050:18.

Shapiro A (2003) Monte carlo sampling methods. *Handbooks in operations research and management science* 10:353–425.

Shapiro A, Philpott A (2007) A tutorial on stochastic programming. *Manuscript. Available at www2. isye. gatech. edu/ashapiro/publications. html* 17.

Shirobokov S, Belavin V, Kagan M, Ustyuzhanin A, Baydin AG (2020) Black-box optimization with local generative surrogates. *Advances in Neural Information Processing Systems* 33:14650–14662.

Vinyals O, Fortunato M, Jaitly N (2015) Pointer Networks. *arXiv e-prints* arXiv:1506.03134.

Vlastelica MP, Paulus A, Musil V, Martius G, Rolinek M (2020) Differentiation of blackbox combinatorial solvers. *ArXiv* abs/1912.02175.

Wilder B, Dilkina B, Tambe M (2019) Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI*, 1658–1665 (AAAI Press).

Wilson GV, Pawley GS (1988) On the stability of the travelling salesman problem algorithm of hopfield and tank. *Biol. Cybern.* 58(1):63–70, ISSN 0340-1200, URL `http://dx.doi.org/10.1007/BF00363956`.

Appendix

Appendix A: Synthetic experiments

A.1. Maximum Matching

In what follows, we aim to show the improvement of this approach over using a traditional projected gradient method. We present computational results comparing the two approaches on a maximum matching problem. We consider a fully-connected bipartite graph with n nodes in each part, and values u_{ij} assigned to the edge connecting nodes i and j from opposite parts. For the experiment, we use $n = 50$, inducing an optimization problem with $n^2 = 2,500$ edges/variables. The maximum matching problem is given by

$$\begin{aligned} w^*(u) = \arg\max_w \quad & \textstyle\sum_{i,j} u_{ij} w_{ij} \\ \text{subject to } & \textstyle\sum_{j=1}^n w_{ij} \le 1, \quad \forall i = 1, \dots, n \\ & \textstyle\sum_{i=1}^n w_{ij} \le 1, \quad \forall j = 1, \dots, n \\ & 0 \le w \le 1, \end{aligned} \tag{59}$$

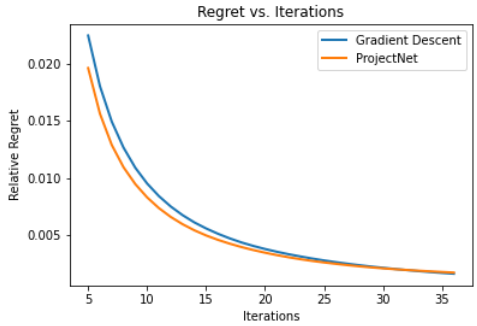
where u_{ij} describes the value of choosing edge from node i to j, and w_{ij} represents the variable that decides whether to choose edge (i, j). These variables can be viewed as flow, and the constraints ensure the flow out of a node, or into a node, is at most 1. At optimality, the solution is guaranteed to be integer, and hence equivalent to choosing a single edge.

Finally, note that the formulation has inequality constraints, however our framework was specified using only equality and nonnegativity constraints. In general, we can transform any problem with inequality constraints $Aw \le b$ into one with equality constraints as in (1) by adding slack variables s: $Aw + Is = b, s \ge 0$. We define the projected gradient descent sequence of points $w_{t+1} = \pi(w_t + \eta \cdot u)$, for edge weights u. We also train a ProjectNet model with $T_0 = 5$ iterations, and compare the objective value of its solution for iterations up to $T_1 = 35$ on testing data. See figure 10a. In particular, we measure the average relative regret of decisions. That is, given realized edge weights u and decision $\hat{w}$, the relative regret is the percent difference in objective between the objective of $\hat{w}$ and the optimal decision in hindsight: $(g_u(\hat{w}) - g_u(w^*(u))/g_u(w^*(u)$. We see that indeed the ProjectNet method improves consistently in accuracy as the number of iterations T_1 is extended from the T_0 steps that were used during training.

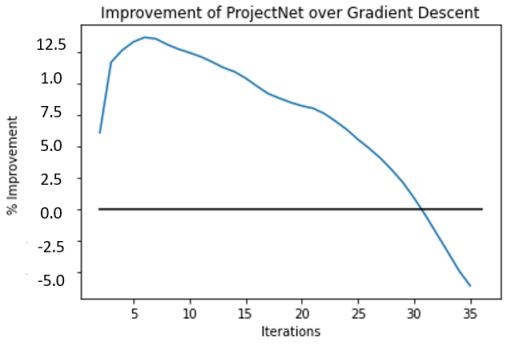
Note that when compared to the traditional gradient descent approach, the ProjectNet approach performs better using fewer iterations. It maintains this edge even for steps $T_1 > T_0$, which it has not trained upon. However for larger T_1 traditional gradient descent is better able to converge to the optimal solution and it overtakes the ProjectNet method. But for the end-to-end framework it is beneficial to use a smaller number of iterations T_1 since this is computationally more efficient, and keeps the gradient $\nabla \hat{w}(u)$ from approaching zero. In the regime of smaller T_1, the ProjectNet method also has an advantage in terms of objective function value, with up to 12.5% improvement. See Figure 10b.

A.2. Multi-product newsvendor

As an example, let us consider the multi-product newsvendor problem. We are given a total of K products to allocate inventory for. Each one has a local demand realization that is random. There is a holding cost



(a) Regret of ProjectNet compared to gradient descent as iterations T increase.



(b) Percent improvement in relative regret of the ProjectNet $T_1 = 0$ model compared to gradient descent.

Figure 10 Comparison of ProjectNet to Gradient Descent

h_j for each product $j = 1, \ldots, K$ (the cost paid for each unit of stock that remains unsold) and a lost sale cost b_j (the cost for each unit of unmet demand) specific to each product. The objective of this problem is to decide how much inventory of product to allocate. The cost of decision w and realization u is given by

$$g_u(w) = \sum_{j=1}^{K} h_j(w_j - u_j)^+ + b_j(u_j - w_j)^+. \tag{60}$$

We additionally impose a constraint on the total amount of stock C that can be stored across all products. Given a known demand u, the nominal optimization problem is then given by the following:

$$\begin{aligned} w^*(u) = \arg\min_{w \geq 0} \quad & g_u(w) \\ \text{subject to} \quad & \textstyle\sum_{j=1}^{k} w_j \leq C. \end{aligned} \tag{61}$$

We assume we are given $N = 500$ datapoints $(x^n, u^n), n = 1, \ldots, N$ of feature observations x^n and corresponding demand observations u^n. Each x^n is generated from a Gaussian distribution with zero covariance and random mean between [-1,1]. Then, we construct a random 2-layer neural network with ReLU activation. Passing in x^n to this network generates the data for u^n. The goal is to learn some forecasting function $f_\theta(x)$ which minimizes the in-sample cost exactly as in equation (7). To reiterate, the key difficulty lies in taking the gradient of $w^*(u)$ with respect to u. Hence, we approximate this with our ProjectNet approach.

We compare against four other methods. (1) A traditional predict-then-optimize approach which only predicts the uncertain parameters, independent of the optimization problem. (2) The OptNet framework of Amos and Kolter (2017) also used for end-to-end learning. This approach requires quadratic objectives, hence we add quadratic regularization terms to the objective as described in Wilder et al. (2019). (3) The traditional sample average approximation (SAA) method which does not incorporate features. And (4) an extension of SAA to use feature information as proposed in Bertsimas and Kallus (2020). In particular, we use a K-nearnest neighbor (KNN) method to determine the weights. Hyperparameters (such as K) are chosen by hyperaparameter tuning. Next we describe the specific formulations that we use for t hese other methods.

In particular, we reformulate the optimization problem to use with OptNet as follows:

$$\begin{aligned} w^*_{\text{optnet}}(u) = \arg\min_{w,p,q \geq 0} \quad & h_j \cdot p_j + b_j \cdot q_j + \alpha(\|w\|^2 + \|p\|^2 + \|q\|^2) \\ \text{subject to} \quad & p_j \geq w_j - u_j, \quad q_j \geq u_j - w_j, \quad \textstyle\sum_{j=1}^{k} w_j \leq C \end{aligned} \tag{62}$$

# Products	SAA	Predict-then-optimize	SAA (KNN)	OptNet	ProjectNet
50	<1s / 20.1	1.1s / 19.5	<1s / 19.6	73s / 18.7	16s / 18.5
100	<1s / 6.6	1.3s / 6.3	<1s / 6.2	504s / 5.9	52s / 5.6

Table 3 Running Time and Task-Based Cost Comparison. Left entry of each cell is the running time per epoch (models trained for the same number of epochs until convergence). The right entry is the average decision cost.

where we introduce variables p_j, q_j to describe the amount of overstocked or understocked units in order to linearize the objective. We also introduce the regularization terms $\|w\|^2 + \|p\|^2 + \|q\|^2$ to problem has nonzero gradient with respect to u. We chose a small value of $\alpha = 0.01$ so that it approximates the original problem well (indeed, at $\alpha = 0.01$, $w^*_{\text{optnet}}(u) = w^*(u)$). The training task to learn the forecasting function is now

$$\min_{\theta} \sum_{n=1}^{N} g_{u^n}(w^*_{\text{optnet}}(f_\theta(x^n))) \tag{63}$$

The SAA formulation is given by

$$\begin{array}{ll} \min_{w \geq 0} & \sum_{n=1}^{N} g_{u^n}(w) \\ \text{subject to} & \sum_{j=1}^{k} w_j \leq C. \end{array} \tag{64}$$

Note that this problem does not depend on features x. The approach in (Bertsimas and Kallus 2020) extends this by making use of features. In particular, instead of minimizing over all data u^n, we only minimize over the k-nearest neighbors to the out-of-sample features x. That is, given an out-of-sample point x, we calculate the decision

$$\begin{array}{ll} \min_{w \geq 0} & z_n g_{u^n}(w) \\ \text{subject to} & \sum_{j=1}^{k} w_j \leq C, \end{array} \tag{65}$$

where $z_n = 1$ if x^n is one of the k-nearest neighbors of x and $z_n = 0$ otherwise.

The results of the experiment can be found in Table 3. We observe that the end-to-end methods based on ProjectNet and OptNet takes better advantage of the problem structure to provide lower-cost decisions. Crucially, the end-to-end method based on ProjectNet is computationally more efficient, 10 times faster to train than the OptNet framework which needs to solve the original optimization problem at each iteration. There is a slight increase of at most 5% in cost due to the nature of approximation of ProjectNet. This gap may potentially be further reduced by more parameter tuning.

A.3. Optimality in the No-feature Case

In this example, we consider the case with no feature information in which we make the same single decision for any datapoint. In this particular case, we can find the exact optimal solution and compare against our proposed method using approximate projections. The experiment is as follows. Suppose we are given historical data $u^1, \ldots, u^N$ of observed demand. Then, we wish to find the single optimal decision

$$\begin{array}{ll} \min_{w \geq 0} & \sum_{n=1}^{N} g_{u^n}(w) \\ \text{subject to} & \sum_{j=1}^{k} w_j \leq C, \end{array} \tag{66}$$

through sample average approximation (SAA). Furthermore, SAA is guaranteed to converge to an optimal solution given enough samples from the underlying distribution (Shapiro 2003). This problem is generally

Capacity	SAA	Gradient Descent with Approximate Projection
10	32.528	32.53
20	9.661	9.669
30	4.173	4.181

Table 4 Comparison of SAA and gradient descent with approximate projections.

solved by traditional optimization methods. In this newsvendor case, this can be rewritten as a linear program. However, we may also solve this by gradient descent, ensuring feasibility by approximate projection on the constraint. Our problem becomes

$$\min_{w} \sum_{n=1}^{N} g_{u^n}(\tilde{\pi}(w)), \tag{67}$$

where $\tilde{\pi}$ is the approximate projection operator onto the constraints $\{w \geq 0, \sum_{j=1}^{K} w_j \leq C\}$. Experimentally, we find that there is an optimality gap of at most 0.1% of the proposed approach over SAA showing that using approximate projections comes at minimal cost and gives rise to near optimal solutions. See Table 4 for more details.

A.4. The Newsvendor Problem when Costs are Quadratic

Finally, we consider the newsvendor problem with quadratic costs. In particular, the penalty of over-allocating or under-allocating product scales quadratically. We use an identical setting from Donti et al. (2017a) which also considers this problem from an end-to-end perspective. In this setting, similarly to Donti et al. (2017a), we assume that the demand takes values over a discrete set of possible values $d_1, \ldots, d_K$. Moreover, we make a distributional forecast $f_\theta(x)_k$ to determine the probability that given observed features x, that the demand is d_k. Given a single product with a realized demand of d and a supply allocation of w, the objective value is given as follows,

$$g_d(w) = c_0 w + \frac{1}{2} q_0 w^2 + c_b(d-w)^+ + q_b((d-w)^+)^2 + c_h(w-d)^+ + q_h((w-d)^+)^2 \tag{68}$$

with different known holding and backorder costs $c_0, q_0, c_h, c_b, q_h, q_b$. Given a distributional forecast $f_\theta(x)$, the optimal stocking quantity w is given by

$$\begin{aligned} w^*(f_\theta(x)) = \arg\min_{w \geq 0} c_0 w + \frac{1}{2} q_0 w^2 + \sum_{k=1}^{K} f_\theta(x)_k \big(& c_b(d-w)^+ + q_b((d-w)^+)^2 + \\ & c_h(w-d)^+ + q_h((w-d)^+)^2 \big). \end{aligned} \tag{69}$$

Note that this formulation is a piece-wise quadratic optimization problem. Finally, given data $(x^1, d^1), \ldots, (x^N, d^N)$, one learns $f_\theta(x)$ by the following risk-minimization problem: $\min_\theta \sum_{n=1}^{N} g_{d^n}(w^*(f_\theta(x^n)))$. As before, we approximate solutions $w^*(p)$ for a distribution p, by using the ProjectNet architecture. We compare our method against the one of Donti et al. (2017a) where this same optimization problem was posed. See Table 5 for a comparison of training times and average decision cost of each method. First, note that as this problem is quadratic, the OptNet framework used in Donti et al. (2017a) can exactly calculate the (non-zero) gradients of the problem. But again, we see the ProjectNet

method is faster by a factor of 2 and scales better with increasing problem size while achieving slightly lower cost.

K (possible demands)	Predict-then-optimize	Donti et al. (2017a)	ProjectNet
5	15.98 / $<$ 0.1s	13.30 / 4.2s	13.07 / 2.4s
10	28.40 / $<$ 0.1s	26.0 / 9.9s	25.65 / 4.2s

Table 5 Performance comparison on quadratic newsvendor. Left side of each column represents average cost on test set, and right side represents average training time over 100 epochs.

Appendix B: Visualizing Toy Examples

In what follows, we will gain some intuition behind the output structure of the ProjectNet approach by considering some low-dimensional toy examples that are as a result easy to visualize. Subsequently in the next section, we will generalize these observations and also provide analytical results.

First, we investigate the output of the model given various different input cost vectors. Consider the following optimization problem we wish to learn via the ProjectNet:

$$\begin{aligned} w^*(u) = \arg\min_w \quad & u_1 w_1 + u_2 w_2 \\ \text{subject to } & w_1 + 2w_2 \geq 1, \quad 2w_1 + w_2 \geq 1, \quad w_1, w_2 \geq 0. \end{aligned} \tag{70}$$

As data, we generate random data $u^1, \ldots, u^N$ with each $u^n = [u_1^n \; u_2^n]$ being drawn uniformly at random from the unit square. We use this to train a ProjectNet model $\hat{w}$. We then examine the output on "test" vectors that are chosen uniformly spaced along a circle. In particular, M cost vectors u^m defined as

$$u^m = \left(\cos\left(m \cdot \frac{\pi}{2M}\right), \sin\left(m \cdot \frac{\pi}{2M}\right)\right). \tag{71}$$

In Figure 11, we plot the values of $\hat{w}(u^m)$. We see that the output of a trained ProjectNet is concentrated around vertices and continuously transitions from one vertex to an adjacent vertex as the cost vector changes. We want solutions to be concentrated around vertices as those are the optimal solutions, but in order to ensure the gradient is nonzero, the property that the ProjectNet's output transitions continuously between vertices is crucial.

Next, we also present the sequence of steps taken at each iteration of the ProjectNet model. In particular, we compare against a traditional projected gradient descent approach. We consider a different optimization problem in three variables:

$$\begin{aligned} w^*(u) = \arg\min_w \quad & u_1 w_1 + u_2 w_2 + u_3 w_3 \\ \text{subject to } & w_1 + w_2 + w_3 = 1, \quad w_1, w_2, w_3 \geq 0. \end{aligned} \tag{72}$$

Again, we generate random cost vectors and train a ProjectNet model $\hat{w}$ to approximate $w^*(u)$. In Figure 12 we notice that the ProjectNet method is able to converge faster by making use of the learned component of the update rule $L(w_t)$.

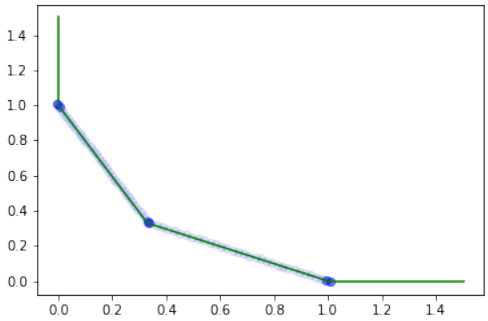


Figure 11 **ProjectNet output varies continuously between vertices.**

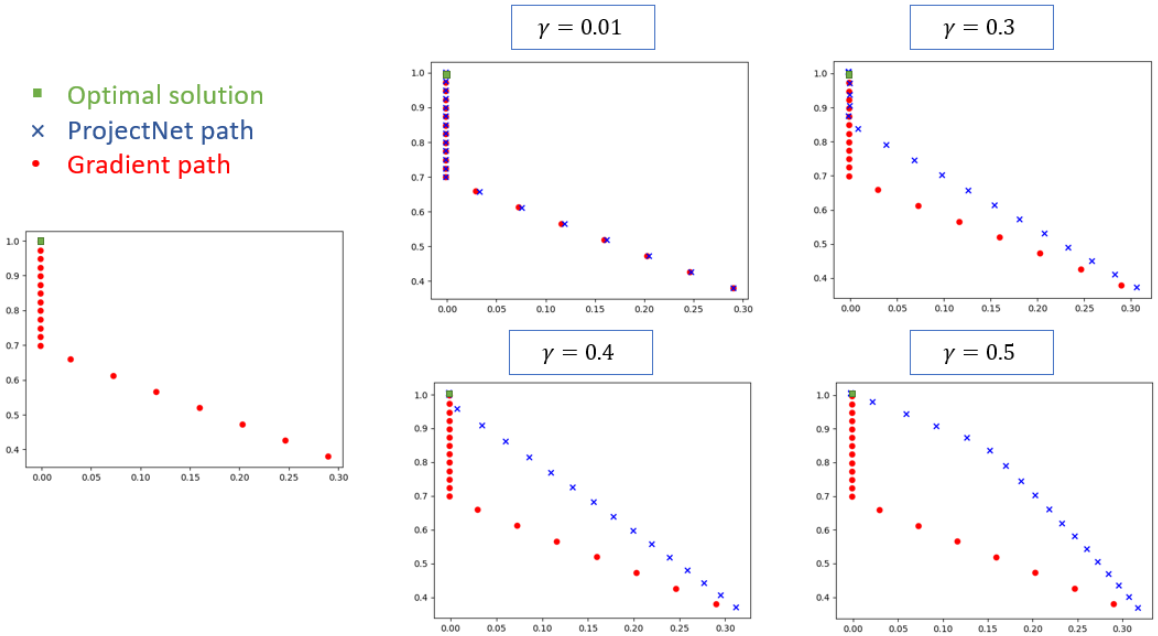


Figure 12 **Comparison of the path taken in gradient descent versus the ProjectNet model.**