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

Rares Cristian[1], **Pavithra Harsha**[2], **Georgia Perakis**[1], **Brian L Quanz**[2], **Ioannis Spantidakis**[1]

[1] Massachusetts Institute of Technology
[2] IBM T. J Watson Research Center
raresc@mit.edu, pharsha@us.ibm.com, georgiap@mit.edu, blquanz@us.ibm.com, yspant@mit.edu

Abstract

In many real-world applications, predictive methods are used to provide inputs for downstream optimization problems. It has been shown that using the downstream task-based objective to learn the intermediate predictive model is often better than using only intermediate task objectives, such as prediction error. The learning task in the former approach is referred to as end-to-end learning. The difficulty in end-to-end learning lies in differentiating through the optimization problem. Therefore, we propose a neural network architecture that can learn to approximately solve these optimization problems, particularly ensuring its output satisfies the feasibility constraints via alternate projections. We show these projections converge at a geometric rate to the exact projection. Our approach is more computationally efficient than existing methods as we do not need to solve the original optimization problem at each iteration. Furthermore, our approach can be applied to a wider range of optimization problems. We apply this to a shortest path problem for which the first stage forecasting problem is a computer vision task of predicting edge costs from terrain maps, a capacitated multi-product newsvendor problem, and a maximum matching problem. We show that this method out-performs existing approaches in terms of final task-based loss and training time.

Introduction

In many practical problems, several input quantities are predicted from historical data prior to decision making. For instance, travel times in a vehicle routing problem, the demand distribution in a supply chain inventory optimization problem, etc. 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 information and prices 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 more important part is the quality of the business objectives, such as the overall costs in the supply chain problem, compared to the accuracy of the machine learning models, such as the mean square error of the demand estimate. But in many organizations, the forecasting and decision making teams are siloed with each focusing on their respective objectives. Indeed, independently solving the forecasting and decision-making problems can give rise to significantly suboptimal decisions (Cameron et al. 2021).

Recent work that has focused on combining the prediction (learning) stage with the downstream optimization task (the decision making) investigates ways to perform gradient descent through the end-to-end optimization problem itself. One challenge for a large class of decision making problems, linear programs, 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. This is because a small change in the cost vector either results in the same optimal solution, or a discontinuous jump to a new vertex. Linear programming constitutes a major class of problems of interest. (Dobkin, Lipton, and Reiss 1979) among others have showed that linear programming is P-complete from which it follows that every problem in P has an LP-formulation of polynomial size. This includes for instance the shortest path problem, maximum matching, etc.

Thus, much existing literature has focused on this setting. To deal with the differentiability issue, (Elmachtoub and Grigas 2022) constructs a convex and differentiable approximation of the objective. In the case of unconstrained quadratic objectives, (Kao, Roy, and Yan 2009) trains a model to directly minimize task loss. Crucially this did not take into account constraints in the optimization problem. This work was finally extended in (Amos and Kolter 2017) to constrained quadratic optimization by analyzing KKT optimality conditions. Donti, Amos, and Kolter (2017) further apply this methodology to stochastic programming problems with probabilistic constraints. Additionally, (Wilder, Dilkina, and Tambe 2019) applies the OptNet methodology to linear optimization problems by adding a quadratic regularization term in order to obtain approximate solutions. They also use this approach to a new setting of submodular maximization problems. Furthermore, the method of analyzing the KKT conditions was extended to more general convex optimization problems in (Agrawal et al. 2019). See for instance (Mandi and Guns 2020), (Vlastelica et al. 2020), (Berthet et al. 2020), (Niepert, Minervini, and Franceschi 2021) for various other methods to approximate the objective function and produce nonzero gradients. A major shortcoming of these approaches is that they are computationally

expensive algorithms, as they solve the original decision-making optimization problem at each gradient step.

Other approaches aim to satisfy feasibility directly, without altering the objective. Modeling the decision as only a linear function of the features, the corresponding empirical risk minimization problem can be formulated as a linear program as in (Ban and Rudin 2019). But this does not allow for more complex mappings. Alternatively, (Frerix, Nießner, and Cremers 2019) describes a solution, not by its coordinates, but by a double-description method, as a convex combination of the vertices and extremes rays describing the feasibility region. The primary downside lies in the often exponential size of the vertex set. Closer to our approach, (Donti, Rolnick, and Kolter 2021) transforms the output of the learning model into a feasible solution by projecting on any equality constraints, and subsequently performing gradient descent to satisfy the inequality constraints. Instead of relying on gradient descent to ensure feasibility, the method we propose in this paper will simply perform 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, which may be of independent interest. The work of (Shirobokov et al. 2020) takes a similar approach within a different context. Instead of having an optimization problem to solve after the initial forecast, they consider simulation problems (a common task within fields of physics or engineering for instance). These simulations are often highly expensive to perform, and they propose 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) does 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.

Another stream of work is "learning to learn" methods and meta-learning: ways to learn algorithms that can solve optimization problems. See for instance (Sergio and Colmenarejo 2016), (Andrychowicz et al. 2016), (Chen et al. 2017). These focus purely on optimization, with no forecasting or end-to-end component. In a similar vein, there has been much work in training neural networks to learn optimal solutions of optimization problems. Much of the work has focused in another direction, specifically on more difficult mixed integer linear programs since their aim is to provide solutions more efficiently than solving the original MILP. For instance, one of the earliest proposals (Hopfield and Tank 1985) to solving the Travelling Salesman Problem is 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 constraints. However, this technique 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. See for instance (Vinyals, Fortunato, and Jaitly 2015) and (Bello et al. 2016). These are particularly useful for path-based problems in which the RNN decides which next node to visit. However, these approaches are not developed with the end-to-end framework in mind.

Specifically, we propose a novel neural network architecture which can learn the optimization problem, allowing one to quickly approximate the solution, without explicitly solving the optimization problem itself. The main issue in doing so arises from ensuring the output of the network is a feasible solution to the optimization problem. We propose to solve this by projecting onto the feasible region after each layer of the network. However, a projection operator also has a zero gradient with respect to the input for similar reasons as above. So, we design an approximate projection method which produces more useful gradients.

Our paper presents the following contributions:

1) A neural network architecture that can learn to approximately solve linear optimization problems. This is done by introducing the key notion of approximate projections. We refer to this approach as ProjectNet.
2) We incorporate this network into an end-to-end learning framework. This is computationally efficient as it allows one to quickly approximate the solution, without explicitly solving the optimization problem itself. This is in contrast to many other existing approaches which do require solving the original optimization problem at each gradient update step.
3) We show experimental results on two end-to-end problems for a shortest path problem and a capacitated multi-product newsvendor problem. In particular, we show our method produces better or competitive decisions in terms of average cost, and is computationally faster to train than other end-to-end methods.
4) We also compare the accuracy of our ProjectNet model in solving optimization problems against projected gradient descent (PGD). We do this since our approach is essentially a generalization of PGD. This is in particular important since our method would be more efficient within the end-to-end framework than PGD. We show that it achieves up to 12.5% improvement over PGD using the same step size and number of iterations.

End-to-End Learning Framework

We first formally present the problem class requiring the integration of machine learning (in order to forecast uncertain parameters) with optimization problems. We then illustrate how our ProjectNet architecture can be used to solve each of these problems. Consider a convex objective function $g_u(w)$ where u is the uncertain parameter that must be predicted. We assume g and its derivative is simple to evaluate. For instance, for linear objectives $g_u(w) = c^T w$ for cost vector $u = c$ or for quadratic objectives, $g_u(w) = q^T w + w^T Q w$ for $u = (q, Q)$. We define the following optimization task with linear constraints and feasible region denoted by $\mathcal{P} = \{w \geq 0 : Aw = b\}$:

$$w^*(u) = \begin{array}{ll} \arg\min_w & g_u(w) \\ \text{subject to} & Aw = b \\ & w \geq 0. \end{array} \tag{1}$$

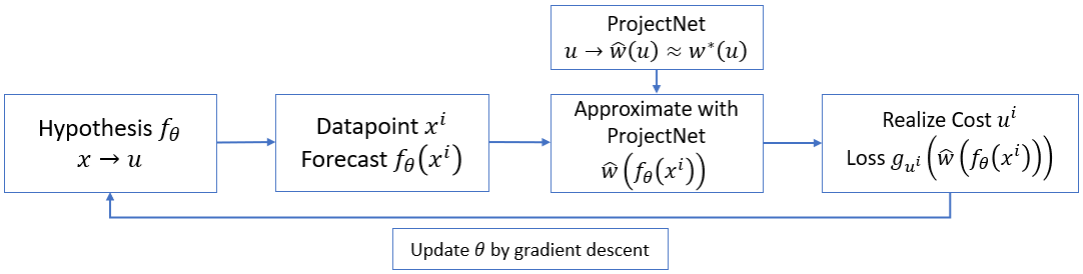


Figure 1: Flow diagram for solving deterministic problems End-to-End using ProjectNet.

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 $u^n \in \mathbb{R}^d$. Given a model f_θ parameterized by some θ (e.g., neural network), for some out-of-sample data x, we make cost prediction $f_\theta(x)$ and a corresponding decision $w^*(f_\theta(x)) \in \mathcal{P}$. Afterwards, for a realized cost vector realized u, we incur overall cost $g_u(w^*(f_\theta(x)))$. A traditional method of solving this is a two-stage approach in which the forecast is learned independently of the downstream optimization problem. This may be done simply by minimizing mean squared error between the forecast and the true in-sample cost: $f_{\text{2-stage}} = \arg\min_\theta \frac{1}{N}\sum_{n=1}^N \|f_\theta(x^n) - u^n\|_2^2$. However, we are only interested in the cost of the final corresponding decisions. This gives rise to the following end-to-end cost minimization problem to learn the parameters θ:

$$\min_\theta \sum_{n=1}^N g_{u^n}(w^*(f_\theta(x^n)) \tag{2}$$

Note that in order to solve this optimization problem using a gradient descent method, we need to compute the gradients $\partial w^*(u)/\partial u$ at points $u = f_\theta(x^n)$. A very common class of problems are linear programs, however as noted earlier, the values of these gradients for commonly occurring linear decision problems are typically zero. However, if the gradient is zero this would result in no update of the weights of the neural network, making it impossible to learn. Therefore, we need to approximate w^*. We replace the optimal w^* with an approximation $\hat{w}$ which we accomplish by using a neural network. In particular, we separately train a neural network $\hat{w}$ so that $\hat{w}(u) \approx w^*(u)$ and $\partial\hat{w}(u)/\partial u$ is nonzero. Note that in this case $\hat{w}$ is already differentiable by construction. This allows us to solve the risk minimization problem stated above using $\hat{w}$ as an approximation instead of the true optimal solution w^*. See the diagram in Fig.1 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 . An advantage of our approach is that the computationally expensive optimization problem $w^*(u)$ is never explicitly solved, but rather we replace it with the forward pass of a simple neural network $\hat{w}$.

ProjectNet

In each of the different examples in the previous section, the end-to-end learning framework requires some way to take the gradient of the output of an optimization problem with respect to its uncertain parameters. The ProjectNet architecture allows us to do this by providing approximate solutions to the optimization problem using only differentiable operations. We summarize this section as follows:

(1) We present a differentiable method of ensuring the output of any network satisfies any given set of constraints.

(2) We describe the ProjectNet architecture which is designed to approximately solve optimization problems.

(3) Finally, we integrate the ProjectNet into the end-to-end framework.

Ensuring Feasibility

One difficulty of learning w^* lies in ensuring that the output of a neural network satisfies the constraints $A\hat{w}(u) = b$ and $\hat{w}(u) \geq 0$. The simplest solution method for satisfying these constraints is to project after each iteration onto the feasible region, similar to a projected gradient descent method. Projection of a point onto the feasible region can be solve by a quadratic program:

$$\pi(w) = \arg\min_{y \in \mathcal{P}} \|w - y\|^2 \tag{3}$$

Hence one possibility is to use the OptNet method (Amos and Kolter 2017) to calculate the gradient $\partial\pi(w)/\partial w$.

However, we face a similar issue as before. That is, the gradient of the projection onto a polytope is often zero. Indeed, the set of points which projects onto any given vertex is a fully-dimensional cone and so the gradient for these would be zero.

As a result, we propose using Dykstra's projection algorithm (Dykstra 1983) that provides a sequence of differentiable steps to approximate the projection. Let $\mathcal{P}_1, \mathcal{P}_2$ be any two intersecting convex sets. We alternatively project onto $\mathcal{P}_1$ followed by $\mathcal{P}_2$ until we reach some desired accuracy (distance from satisfying both constraints). We define π_1, π_2 as the projection operators onto sets $\mathcal{P}_1, \mathcal{P}_2$ respectively. After k steps of the iterative projection, we reach an approximation $\tilde{\pi}^k$ defined in Algorithm 1. For linear optimization as described in the previous sections, we may let $\mathcal{P}_1 = \{w : Aw = b\}, \mathcal{P}_2 = \{w : w \geq 0\}$, so that $\mathcal{P} = \mathcal{P}_1 \cap \mathcal{P}_2$. The projections π_1, π_2 can be easily eval-

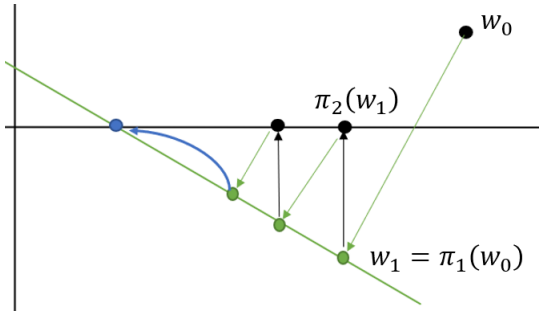


Figure 2: Iterative projection method. The sequence of projections converges to a feasible solution (depicted as the blue point).

Algorithm 1 Dykstra's Projection Method

function $\tilde{\pi}^k(w)$
  Initialize $w_0 = w$, and $p_0 = q_0 = 0$.
  for $t = 0, \ldots, k-1$ **do**
    $y_{t+1} = \pi_1(w_t + p_t)$
    $p_{t+1} = w_t + p_t - y_t$
    $w_{t+1} = \pi_2(y_t + q_t)$
    $q_{t+1} = y_t + q_t - w_{t+1}$
  return w_k

uated as follows:

$$\pi_1(w) = \arg\min_{y:Ay=b} \|w - y\|_2^2 \tag{4}$$

$$= w - A^T(AA^T)^{-1}(Aw - b) \tag{5}$$

$$\pi_2(w) = \arg\min_{y \geq 0} \|w - y\|_2^2 = ReLu(w). \tag{6}$$

In the case of linear subspaces, the method simplifies to $\tilde{\pi}^k(w) = \pi_2(\pi_1(\ldots(\pi_2(\pi_1((w))\ldots))$. This is depicted in Fig. 2. 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) shows that there exists $\rho < 1, a > 0$ so that for any integer k:

$$\left\|\tilde{\pi}^k(w) - \pi(w)\right\|_2 \leq a \cdot \rho^k \tag{7}$$

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 perfect projections onto vertices of the feasible polytope, then the gradient is zero. So, one must be careful to choose the number of iterations k to be large enough to provide good approximations, but to also keep the corresponding gradients from becoming too small.

Model Architecture

Let us now describe the proposed architecture of the model $\hat{w}$. This will be similar in form to a recurrent neural network. The input to the network is the prediction vector u, and the output is some approximate feasible solution. Alternatively, we may think of an RNN as learning an algorithm:

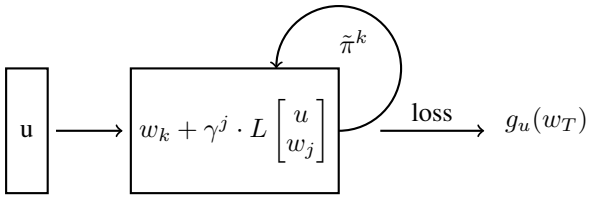
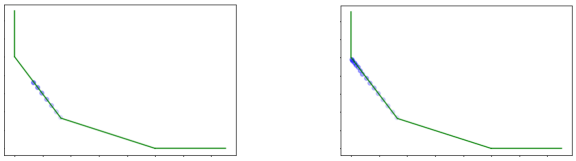


Figure 3: ProjectNet architecture



7-step sequence 15-step sequence

Figure 4: Progression of the learned algorithm over $T_0 = 7$ (left) and extrapolated to $T_1 = 15$ iterations (right).

we begin at some w_0 and update to a new point by a some function $w_{j+1} = \phi(w_j)$. In addition, we must enforce that each point w_j is (approximately) feasible. Instead we may want $w_{j+1} = \tilde{\pi}^k(\phi(w_j))$. Taking inspiration from projected gradient descent, one possible ϕ is simply $\phi(w) = w - \eta \cdot c$ where c is the cost vector and η is some step size. That is, at each iteration we move the current point in the direction of the cost, and then project onto the feasible region. ProjectNet is a generalization of this approach, in which we learn the direction to move in, taking into account the current solution w_j, the underlying uncertainty u, and implicitly the constraints as well. Consider the following update step:

$$w_{j+1} = \tilde{\pi}^k\left(w_j + \eta \cdot L \begin{bmatrix} u \\ w_j \end{bmatrix}\right) \tag{8}$$

First, we use a small neural network unit L to compute the direction to move in (for instance, this could be as simple as a linear function or a two-layer network). Finally, we (approximately) project back onto the feasible region using $\tilde{\pi}^k$. The final decision after T rounds is $\hat{w}(u) = w_T$ and the loss after T rounds is the cost of w_T. For example, in the deterministic optimization problem, this would be $g_u(w_T)$. We can now simply learn L via traditional gradient-based methods. We will refer to this model as ProjectNet. We mentioned the ProjectNet architecture is in a way a generalization of projected gradient descent. A major advantage of this approach over only projected gradient descent is that the ProjectNet learns an update rule L which depends on the optimization problem itself. In particular, for gradient descent the update step is always in the direction of the gradient. The ProjectNet method allows for this to change depending on the position of the current point.

Generalizing past T iterations Recall we may view this learning problem as a task to learn an algorithm that solves optimization problems. At each iteration, we apply a mapping defined by L and project back onto the feasible region. During training, we only perform T_0 iterations of this map-

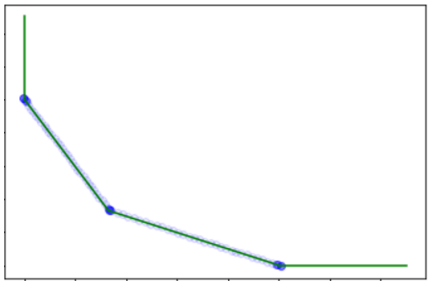


Figure 5: ProjectNet output for many input vectors.

ping. But, given learned weights we can then apply the mapping even further, hopefully converging closer to the optimal solution. For tractability purposes, it may difficult to train with a high value of T_0, but it is simple to apply the learned mapping for more iterations after training is complete. In particular, once ProjectNet has been trained using T_0 steps, we may then use some $T_1 > T_0$ steps when using it to learn forecasts. Again using the earlier toy example, the model was trained using $T_0 = 7$ iterations. See Fig. 4. The sequence of outputs at each of the iterations is given in the left figure for a given cost vector whose true corresponding optimal solution is the top left vertex. And we subsequently continue to apply the same learned mapping for a total of $T_1 = 15$ iterations. We can see indeed the final solution improves towards the true optimal solution.

Approximation Quality

By construction, our model is differentiable — a key property needed for an end-to-end learning approach. However, it should be close to the original problem in which the optimal solutions are discrete vertices. Indeed, we see that the output of a trained ProjectNet is concentrated around vertices while continuously and quickly transitioning from one vertex to an adjacent vertex as the cost vector changes. We illustrate this desired property of our learned models on a simple toy example which consists of two constraints $w_1 + 2w_2 \geq 1$ and $2w_1 + w_2 \geq 1$. The cost vector was varied uniformly and the final output of the learned model is given for each. See Fig. 5.

Now we aim to show the improvement of this approach over using a traditional projected gradient method. We present computational results comparing the two 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 $n^2 = 2500$ edges/variables.

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 6. We report the relative regret $(g_u(\hat{w}) - g_u(w^*(u)))/g_u(w^*(u))$. 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.

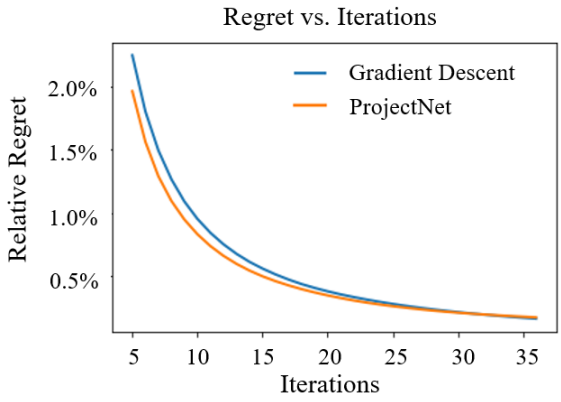


Figure 6: Regret of ProjectNet compared to gradient descent as iterations T increase.

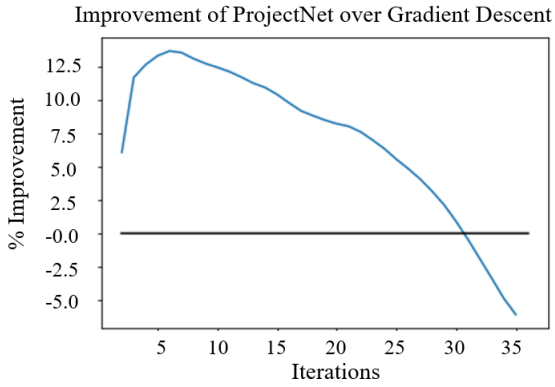


Figure 7: Percent improvement in regret of the ProjectNet $T_1 = 5$ model compared to gradient descent.

When compared to traditional gradient descent, the ProjectNet performs better after 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. 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 $\nabla w^*(u)$ from approaching zero. In the regime of smaller T_1, the ProjectNet method also has the edge in terms of objective value, with up to 12.5% improvement. See Figure 7.

End-to-End Learning via ProjectNet

First, a ProjectNet is trained to learn solutions to the optimization problem $w^*(u)$. In particular, given some cost vectors $u^1, \ldots, u^m$, we aim to learn some $\hat{w} = \hat{w}_L$ parametrized by the update layer L which minimizes the cost:

$$\arg\min_{L} \sum_{i=1}^{m} g_{u^i}(\hat{w}_L(u^i)) \tag{9}$$

This is done by gradient descent and is described explicitly in Algorithm 2. Note that we never need to solve the nominal optimization problem $w^*(u^i)$ during this process.

Algorithm 2 Training ProjectNet

function TRAINPROJECTNET($u^1, \ldots, u^N$)
  Initialize matrix L
  for each epoch **do**
    for $i = 1, \ldots, N$ **do**
      Initialize $w_0 = 0$
      for $j = 1, \ldots, T$ **do**
        $z_{j+1} = w_j - \eta L \cdot \begin{bmatrix} u^i \\ w_j \end{bmatrix}$
        $w_{j+1} = \tilde{\pi}^k(z_{j+1})$
      loss = $g_{u^i}(w_T)$
      Update L by any gradient method.
  return L

Algorithm 3 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($u^1, \ldots, u^N$)
  Initialize θ at random.
  for each epoch **do**
    for $i = 1, \ldots, N$ **do**
      Compute $\nabla_\theta g_{u^i}(\hat{w}(f_\theta(x_N)))$.
      Update θ by any gradient method.
  return θ

In addition, we may use any data $u^1, \ldots, u^n$ that we wish, not necessarily only the vectors from the original training data of (x^j, u^j). 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 3 which follows the steps in diagram 1 shown earlier. In short, we make a forecast $f_\theta(x^i)$ and differentiate through the approximate corresponding solution $\hat{w}(f_\theta(x^i))$ to update θ. During the training step of ProjectNet, we may use T_0 iterations, while when subsequently evaluating $\hat{w}(f_\theta(x^i))$, we may use any $T_1 \geq T_0$ iterations to improve the accuracy of the ProjectNet's approximation. We now illustrate the application of our model.

Computational Results

In this section we present computational results that our ProjectNet method is effective in end-to-end learning. We show this on two tasks: (1) a single-stage multi-product newsvendor problem, (2) a shortest path problem in which the forecasting step is a computer vision task of predicting edge costs from terrain maps. Additional computational experiments can be found in the appendices.

Multi-Product Newsvendor

Let us consider a multi-product newsvendor problem. Each of K products has a local demand realization that is random. There is a holding cost h_j for each product j (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

$$g_u(w) = \sum_{j=1}^{K} h_j(w_j - u_j)^+ + b_j(u_j - w_j)^+ \tag{10}$$

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:

$$w^*(u) = \begin{array}{ll} \arg\min_{w \geq 0} & g_u(w) \\ \text{subject to} & \sum_{j=1}^{k} w_j \leq C \end{array} \tag{11}$$

We compare against four other methods. (1) A traditional two-stage 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, Dilkina, and Tambe 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. Results of the experiment can be found in table 1. We see that the end-to-end method takes better advantage of the problem structure to provide lower-cost decisions. Moreover, it is more computationally efficient than the OptNet framework which needs to solve the original optimization problem at each iteration.

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

Table 1: Running Time and Task-Based Cost Comparison. Left entry of each cell is the running time, while the right entry is the average cost of the decision.

Optimality in the no-feature case Here we consider a simplified 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 through sample average approximation (SAA). Furthermore, SAA is guaranteed to converge to an optimal solution given enough samples from the underlying distribution. This problem is generally solved by traditional optimization methods. In this newsvendor case, this can be rewritten as a linear program. However,

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

Table 2: Comparison of SAA and gradient descent with approximate projections.

Method	Matches	Runtime
ResNet-18 Baseline	40.2%	9.2
(Vlastelica et al. 2020)	86.6%	81.3
ProjectNet	83.0%	68.3

Table 3: 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.

we may also solve this by gradient descent, ensuring feasibility by approximate projection on the constraint. Our problem becomes

$$\min_{w} \sum_{n=1}^{N} g_{u^n}(\tilde{\pi}(w)) \tag{12}$$

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. See Table 2.

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. We compare against this 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 8 (top left) in the appendix 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. The path begins at node a and ends at node b:

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

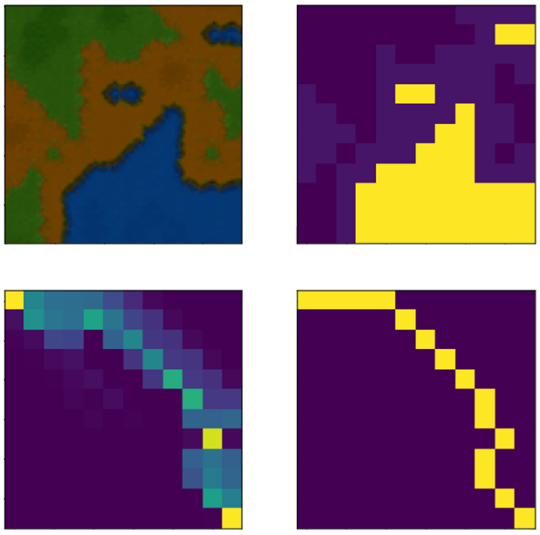


Figure 8: Sample terrain (top left), weight matrix (top right), path proposed by ProjectNet (bottom left), and optimal path (bottom right).

Figure 8 (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. In addition, as in (Vlastelica et al. 2020) we report the percentage of test instances for which various methods found an optimal path in Table 3. We can see the ProjectNet method is competitive with the rest of the field. More crucially, the running time of our approach is 19% faster than the end-to-end method of (Vlastelica et al. 2020).

Conclusions

In this paper we studied a fundamental problem of decision-making under uncertainty by using the end-to-end learning framework. We introduced a novel approach that solves the end-to-end learning problem to learn the optimal solution. Our proposed approach avoids the classical end-to-end optimization problem approach difficulty that relies on differentiating the objective function (e.g., the overall supply chain cost). We instead proposed and analyzed a novel neural network approach that learns to approximately solve an underlying optimization problem, ensuring its output satisfies the feasibility constraints.

We applied this end-to-end learning approach to three problems: a shortest path problem on the warcraft tile dataset, and a capacitated multiproduct newsvendor problem. We have shown in experimental results 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.; and Kolter, J. Z. 2019. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32.

Amos, B.; and Kolter, J. Z. 2017. OptNet: Differentiable Optimization as a Layer in Neural Networks. In Precup, D.; and Teh, Y. W., eds., *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, 136–145. PMLR.

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

Ban, G.-Y.; and Rudin, C. 2019. The Big Data Newsvendor: Practical Insights from Machine Learning. *Oper. Res.*, 67: 90–108.

Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio, S. 2016. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.

Berthet, Q.; Blondel, M.; Teboul, O.; Cuturi, M.; Vert, J.-P.; and Bach, F. 2020. Learning with Differentiable Perturbed Optimizers. *ArXiv*, abs/2002.08676.

Bertsimas, D.; and Kallus, N. 2020. From Predictive to Prescriptive Analytics. *Management Science*, 66(3): 1025–1044.

Cameron, C.; Hartford, J.; Lundy, T.; and Leyton-Brown, K. 2021. The Perils of Learning Before Optimizing. *arXiv preprint arXiv:2106.10349*.

Chen, Y.; Hoffman, M. W.; Colmenarejo, S. G.; Denil, M.; Lillicrap, T. P.; Botvinick, M.; and Freitas, N. 2017. Learning to learn without gradient descent by gradient descent. In *International Conference on Machine Learning*, 748–756. PMLR.

Deutsch, F.; and 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.

Dobkin, D.; Lipton, R. J.; and Reiss, S. 1979. Linear programming is log-space hard for P. *Information Processing Letters*, 8(2): 96–97.

Donti, P.; Amos, B.; and Kolter, J. Z. 2017. Task-based end-to-end model learning in stochastic optimization. *Advances in neural information processing systems*, 30.

Donti, P. L.; Rolnick, D.; and Kolter, J. Z. 2021. DC3: A learning method for optimization with hard constraints. *CoRR*, abs/2104.12225.

Dykstra, R. L. 1983. An Algorithm for Restricted Least Squares Regression. *Journal of the American Statistical Association*, 78(384): 837–842.

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

Frerix, T.; Nießner, M.; and Cremers, D. 2019. Linear Inequality Constraints for Neural Network Activations. *CoRR*, abs/1902.01785.

Guyomarch, J. 2017. Warcraft ii open-source map editor. http://github.com/war2/war2edit.

Hopfield, J.; and Tank, D. 1985. Neural Computation of Decisions in Optimization Problems. *Biological cybernetics*, 52: 141–52.

Kao, Y.-h.; Roy, B.; and Yan, X. 2009. Directed Regression. In Bengio, Y.; Schuurmans, D.; Lafferty, J.; Williams, C.; and Culotta, A., eds., *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc.

Mandi, J.; and Guns, T. 2020. Interior Point Solving for LP-based prediction+optimisation. *ArXiv*, abs/2010.13943.

Niepert, M.; Minervini, P.; and Franceschi, L. 2021. Implicit MLE: backpropagating through discrete exponential family distributions. *Advances in Neural Information Processing Systems*, 34: 14567–14579.

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

Sergio, Y. C. M. W. H.; and Colmenarejo, G. 2016. Learning to Learn for Global Optimization of Black Box Functions. *stat*, 1050: 18.

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

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

Vlastelica, M. P.; Paulus, A.; Musil, V.; Martius, G.; and Rolinek, M. 2020. Differentiation of Blackbox Combinatorial Solvers. *ArXiv*, abs/1912.02175.

Wilder, B.; Dilkina, B.; and Tambe, M. 2019. Melding the Data-Decisions Pipeline: Decision-Focused Learning for Combinatorial Optimization. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI*, 1658–1665. AAAI Press.

Wilson, G. V.; and Pawley, G. S. 1988. On the Stability of the Travelling Salesman Problem Algorithm of Hopfield and Tank. *Biol. Cybern.*, 58(1): 63–70.