End-to-End Learning via Constraint-Enforcing Approximators for Linear Programs with Applications to Supply Chains

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, prediction problems are used to model forecast inputs for downstream optimization problems and it often suffices to check the performance of the final task-based objective, instead of intermediate task objectives, such as prediction error. 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 linear programs, particularly ensuring its output satisfies the feasibility constraints. We further apply this to a multi-location newsvendor problem with cross fulfillment. We also analyze this problem with explicit fulfillment rules, and show the end-to-end problem can be solved with the exact derivative, without the need for approximations. We show that both these methods out-perform the predict following by optimize approach.

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.

Recent work 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 simple class of decision making problems, linear programs, stems from the fact that the gradient of the optimal solution with respect to the predicted quantities, say 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. For example, (Elmachtoub and Grigas 2021) deals with this by constructing a convex and differentiable approximation of the objective. Alternatively, as shown in (Donti, Kolter, and Amos 2017), the exact gradients can be calculated in the case of quadratic optimization. Hence, (Wilder, Dilkina, and Tambe 2019) proposes to add a simple quadratic regularization term to linear programs in order to obtain approximate solutions. In addition, see (Kotary et al. 2021) for a general survey for end-to-end combinatorial learning problems and references therein. 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 inequality constraints. Instead of relying on gradient descent to ensure feasibility, our method will simply perform a sequence of alternating projections onto simpler sets. Additionally, our method explicitly learns and approximates the optimization problem and makes an intermediate forecast instead of directly outputting a decision, improving interpretability.

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.

To illustrate our proposed method, we will focus on the following multi-warehouse newsvendor problem with cross fulfillment (NCF). Through adding additional structure to the NCF problem, in the form of fulfillment rules, we show that the derivatives may be determined explicitly, without any need for approximation.

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 the model as a 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, unlike many existing approaches.

(3) We describe closed form solutions for the NCF problem under a set of fulfillment rules. This additionally allows us to apply the end-to-end learning using the exact solution to this abridged problem, without the need for approximations such as ProjectNet.

End-to-End Learning Framework

We first describe our end-to-end learning approach with a generic optimization plus decision problem, and then show its application to our target problem. Consider the following linear optimization task, with feasible region denoted by $\mathcal{P}$:

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

Suppose we are given N data points $(x_1, c_1), \ldots, (x_N, c_N)$ with features $x_n \in \mathbb{R}^p$ and realized costs $c_n \in \mathbb{R}^d$. Given a model f_θ parameterized by some θ (e.g., neural network), for some out-of-sample data x, we make prediction $f_\theta(x)$ and a corresponding decision $w^*(f_\theta(x)) \in \mathcal{P}$. Afterwards, for a realized cost vector realized c, we incur overall cost $c^T w^*(f_\theta(x))$. This gives rise to the following end-to-end risk minimization problem to learn the parameters θ:

$$\min_\theta \sum_{n=1}^{N} c_n^T w^*(f_\theta(x_n)) \tag{2}$$

Note that in order to solve this optimization problem using a gradient descent method, we need to compute the gradients $\partial w^*(v)/\partial v$ at points $v = f_\theta(x_n)$. However, as noted earlier, the values of these gradients for linear decision problems are typically zero. Therefore, we need to approximate w^*. In this work we accomplish this using a neural network. In particular, we separately train a neural network $\hat{w}$ so that $\hat{w}(c) \approx w^*(c)$. 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^*.

Ensuring Feasibility

The difficulty of learning w^* lies in ensuring that its output satisfies the constraints $A\hat{w}(c) = b$ and $\hat{w}(c) \geq 0$. 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. However, we face a similar issue as before, that is, that the gradient of the projection onto a polytope is often zero (this is due to the fact that the projection may be onto a vertex).

As a result, we propose the following iterative and differentiable, algorithm that approximates projections: Let $P_1 = \{w : Aw = b\}$, $P_2 = \{w : w \geq 0\}$, so that $P = P_1 \cap P_2$. We alternatively project onto P_1 followed by P_2 until we reach some desired accuracy. That is, the approximate projection $\tilde{\pi}$ is defined as:

$$\tilde{\pi}(w) = \pi_2(\pi_1(\ldots(\pi_2(\pi_1 w))\ldots)), \tag{3}$$

where

$$\begin{aligned} \pi_1(w) &= \arg\min_{v: Av=b} \|w - v\|_2^2 \\ &= w - A^T(AA^T)^{-1}(Aw - b) \qquad (4) \\ \pi_2(w) &= \arg\min_{v \geq 0} \|w - v\|_2^2 = ReLu(w). \qquad (5) \end{aligned}$$

Indeed, the sequence of points $w_n = \pi_2(\pi_1(w))$ converges to a point in $P = P_1 \cap P_2$. Moreover, the convergence rate is geometrical. See Theorem 1 of (Gubin, Polyak, and Raik 1967).

Model Architecture

Let us now describe the proposed architecture of the model $\hat{w}$. Thus is similar in form to a neural network with a single learnable layer L. The input to the network is the cost vector c, and the output is some solution approximate feasible solution. Alternatively, we may think of this approach as learning an algorithm: we begin at some w_0 and update to a new point as follows:

$$w_{k+1} = \tilde{\pi}\left(w_k + \gamma^k \cdot L \begin{bmatrix} c \\ w_k \end{bmatrix}\right) \tag{6}$$

First, we use L to compute the direction to move in, as a linear function of the cost and the current position. This is then also scaled by γ^k to decrease the step size as we increase the number of iterations. Finally, we (approximately) project back onto the feasible region using $\tilde{\pi}$. The loss after T rounds is $c^T w_T$. See Fig. 1. We can now simply learn L via traditional gradient-based methods. We will refer to this model as ProjectNet.

By construction, our model is differentiable - a key property needed for an end-to-end learning approach. In addition, the output itself should be concentrated around vertices as well as continuously and quickly transition 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. 2.

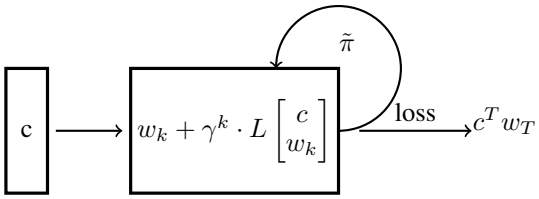


Figure 1: ProjectNet architecture

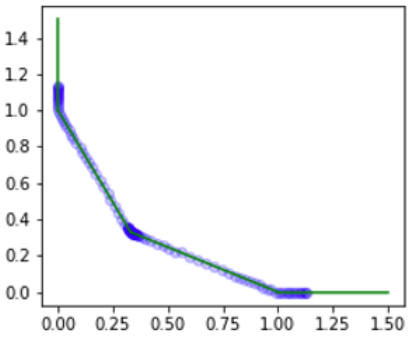
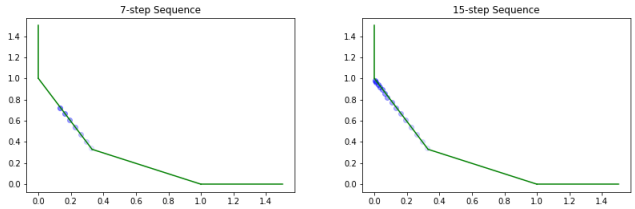


Figure 2: The output of the learned model is concentrated around vertices and smoothly transitions from one vertex to an adjacent one. A darker shade of blue implies more outputs lie at those points.

We may also view this learning problem as a task to learn an algorithm that solves optimization problems. At each iteration, we apply a mapping defined by θ and project back onto the feasible region. During training, we only perform T_0 iterations of this mapping. 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 simple to apply the learned mapping for more iterations after training is complete. See for instance Fig. 3. Again using the toy example from before, the model was trained using $T_0 = 7$ iterations. 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.

ProjectNet for Max Matching We now present some results for a ProjectNet model trained to learn solutions to a maximum matching optimization problem on a bipartite graph. We are given a fully-connected bipartite graph with n nodes in each part, and values c_{ij} assigned to the edge connecting nodes i and j from opposite parts. The corresponding max matching problem is given by

$$w^*(c) = \begin{array}{ll} \arg\max_w & \sum_{i,j} c_{ij} w_{ij} \\ \text{subject to} & \sum_{j=1}^n w_{ij} \leq 1, \quad \forall i = 1, \ldots, n \\ & \sum_{i=1}^n w_{ij} \leq 1, \quad \forall j = 1, \ldots, n \\ & 0 \leq w \leq 1 \end{array} \tag{7}$$



7-step sequence 15-step sequence

Figure 3: We show the progression of the learned algorithm over 7 and 15 iterations respectively. The model itself learned using only 7 steps. The initial point is near the middle vertex, while the true optimal solution is the left-most vertex. We see that during the learned 7 steps, the algorithm moves towards the optimal solution, and continuing to apply the same learned weights for additional iterations, the model moves closer to the optimal solution (as in the right plot).

In general, we can transform any problem with inequality constraints $Aw \leq b$ into one with equality constraints as in (1) by adding slack variables s: $Aw + Is = b, s \geq 0$.

We generate a dataset of training examples where each entry c_{ij} is sampled uniformly from $[0, 1]$. This is done for a bipartite graph with 20 nodes in each part, and hence a total of 400 edges. The ProjectNet is the trained to learn an (approximately) feasible solution $\hat{w}(c)$, deciding which edges should appear in the optimal solution. We empirically show 3 key properties of the resulting learned model: 1) good relative percentage error $(c^T \hat{w}(c) - c^T w^*(c))/c^T(w^*(c))$. See Table 1 In addition, we compare against a traditional neural network trained to minimize mean squared error between its prediction and the true solution, without explicitly taking into account feasibility. We see that while it achieves a lower MSE, its objective cost is significantly higher — on average over 4 times higher. See Table 2. Moreover, 2) The approximation $\hat{w}(c)$ is indeed nearly feasible. By construction we ensure that it violates any constraint by at most 0.01. That is, we keep applying the alternate projections until this tolerance level is met. We show that after projecting this $\hat{w}(c)$ onto the feasible region, the result still has nearly the same objective value. Finally 3) we show that continuing to apply the learned mapping L for T_1 iterations improves the accuracy of the model.

T_1	Error (%)	Error (%) Post Projection
7	18.69	19.57
10	17.24	18.21
15	16.49	17.43
20	16.26	17.15
25	16.15	17.04
30	16.12	16.99

Table 1: Relative percentage error when trained with $T_0 = 7$ and evaluated on test set with varying iteration count T_1.

Model	Objective Error (%)	MSE
Project Net	16.12	6.077
Feed Forward Net	71.21	4.45

Table 2: Comparison of Porject Net and Feed Forward network

Learning Forecasts

The model $\hat{w}(\cdot)$ we proposed above, needs to provide good approximations of the cost vectors $f_\theta(x_1), \ldots, f_\theta(x_N)$ in a neighborhood. Nevertheless, as we update θ, the current model $\hat{w}$ may no longer provide good approximations since the forecasts may have changed too much. Hence, we occasionally retrain ProjectNet. To train this, we may also need to generate additional data, for instance by adding small element-wise random perturbations. We define $\mathcal{N}_\theta(x_1, \ldots, x_n)$ to be this new perturbed dataset we use to train our ProjectNet model: $\mathcal{N}_\theta(x_1, \ldots, x_n) = \{f_\theta(x_i) \cdot \delta : \delta_i \sim [1-\bar{\delta}, 1+\bar{\delta}]\}$.

function PREDICTOPTIMIZE($A \in \mathbb{R}^{m\times d}$, $b \in \mathbb{R}^m$, $((x_1, c_1), \ldots, (x_n, c_n))$)
  Initialize θ.
  for Each Epoch **do**
    $\hat{w} \leftarrow$ TRAINPROJECTNET($\mathcal{N}_\theta(x_1, \ldots, x_n)$)
    for $i = n, \ldots, N$ **do**
      $\theta \leftarrow \theta - \eta \cdot \partial c_N^T \hat{w}(f_\theta(x_N))/\partial\theta$
    end for
  end for
  return θ
end function

where TRAINPROJECTNET is simply any gradient method used to train a ProjectNet on given data. Note that during the training step of ProjectNet, we may use T_0 iterations, while when evaluating $\hat{w}(f_\theta(x_N))$ we may use any T_1 iterations to improve the accuracyof the prediction.

We now illustrate the application of our model to a supply chain inventory optimization problem.

Multi-warehouse Newsvendor Problem with Cross-Fulfillment (NCF)

Let us consider a network of M warehouses (these are the nodes in the network). Each node i has a local demand realization D_i that is random, a holding cost h_i for each unit of inventory that was stored and remained unsold and a lost sale cost u_i associated with any unfulfilled demand. Finally, every node i can fulfill an order of node j with a shipping costs c_{ij}.

The objective of this problem is to decide how much inventory q_i to allocate to each node i in order to minimize the expected fulfillment cost. In what follows, we express this problem as an optimization problem. In particular, we use auxiliary decision variables a_{ij} that represent the number of units that node i will ship to node j to fulfill any excess local demand. Following that definition, a_{ii} is the number of units that node i uses to fulfill its own local demand. Notice that a_{ij} is a function of both $\boldsymbol{q}$ as well as $\boldsymbol{D}$ across the whole network.

$$\min_{q_i \geq 0} \; E_{\mathbf{D}}[Z(\mathbf{D}, \mathbf{q})] \qquad \text{where } Z(\mathbf{D}, \mathbf{q}) = \min_{a_{ij} \geq 0} \Big[\sum_{i=1}^{M} h_i \big[q_i - \sum_{j=1}^{M} a_{ij}\big] + u_i \big[D_i - \sum_{j=1}^{M} a_{ji}\big]^+ + \sum_{j=1}^{M} c_{ij} a_{ij} \Big]$$
$$\text{s.t:} \quad \sum_{j=1}^{M} a_{ij} \leq q_i \quad \forall i \tag{8}$$

To ensure non-trivial solutions, we assume that $h_i < c_{ij} < u_j$, for each pair of nodes (i, j). The left side of inequality is to ensure that the fulfilled quantity does not exceed the demand, and the right to capture that it is preferable to fulfill the demand of one warehouse from another than lose a sale (common approach in practice).

In a practical setting the number of nodes is large and the distribution of demand is hard to characterize. Thus a common method to come up with a solution is a Sample Average Approximation (SAA) approach. Suppose we are given a forecasting function $D_\theta(x)$ and generate K samples from this distribution. Then, we may approximate the expectation minimization problem as

$$\hat{q}(D_\theta(x)) = \arg\min_{q_i \geq 0} \frac{1}{K} \sum_{k=1}^{K} Z(\mathbf{D}_\theta^k(x), \mathbf{q}), \tag{9}$$

Then, given historical observations $(x_1, \mathbf{D_1})$, $\ldots$, $(x_N, \mathbf{D_N})$, we solve the following end-to-end risk minimization problem to learn forecast D_θ:

$$\min_{\theta} \sum_{i=1}^{N} Z(\mathbf{D}_i, \hat{q}(D_\theta(x_i))). \tag{10}$$

In the following section we will discuss how by adding additional structure to the NCF problem, the gradients above can be explicitly computed, without the use of any approximation scheme such as ProjectNet. This in turn greatly simplifies the end-to-end training.

Fulfillment Rules

A fulfillment rule is a set or rules that describes how to prioritize inventory fulfillment across warehouses. These rules are fixed beforehand and as a result they are given to the optimization problem as an input. Any complete ordering of the (i, j) pairs can be part of a fulfillment rule. We define $O(i, j)$ in the following way:

$$O(i,j) = \{lk : lk \text{ has higher fulfillment priority than } ij\} \tag{11}$$

In practice, the rule is usually decided by the ordering of the fulfillment costs with the goal of maximizing the revenue margins. An example can be seen in the simple network of figure (4). Assume that for each location i the corresponding costs c_{ii} are 10. Then the ordering of the fulfillment costs is as follows $c_{11}, c_{22}, c_{33}, c_{12}, c_{21}, c_{13}, c_{31}, c_{23}, c_{32}$. This implies that for instance $O(1, 3) = \{c_{11}, c_{22}, c_{33}, c_{12}, c_{21}\}$.

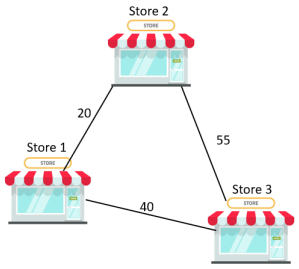


Figure 4: Example network with 3 stores and their corresponding symmetric cross fulfillment costs.

We can generate a fulfillment rule for any network through sorting its corresponding fulfillment costs in ascending order. However, our proposed model is general enough to work for any general ordering, even for those that are not dependent on cost.

Introducing a fulfillment rule allows us to fully characterize a_{ij} optimally. In particular:

$$a_{ij} = \min(q_i - \sum_{il \in O(i,j)} a_{il} \ , \ D_i - \sum_{lj \in O(i,j)} a_{lj}) \quad (12)$$

Using the above structure we can now characterize the optimal solution under a given set of fulfillment rules.

Characterizing the Optimal Solution

Given a set of fulfillment rules, we no longer have to solve $Z(\mathbf{D}, \mathbf{q})$ of problem (8) to get the a_{ij}. Instead we can directly use equation (12). Thus, problem (8) reduces to a single stage unconstrained problem. In addition, the objective function is convex as a sum of convex functions, allowing us to characterize the optimality equations for the q_i.

Indeed taking the derivative with respect to every q_i we get the system of equations as seen in (13).

$$\begin{aligned} &h_i + \sum_{j=1}^{M}\sum_{k=1}^{M}(c_{jk} - h_j)\frac{\partial E[a_{jk}]}{\partial q_i} + \\ &\sum_{j=1}^{M} u_j \frac{\partial E\left[D_j - \sum_{k=1}^{M} a_{ki}\right]^+}{\partial q_i} = 0 \quad \forall i \end{aligned} \quad (13)$$

The derivatives in the above system can be calculated in closed form since we have fully characterized the structure of a_{ij}. Although the equations fully describe the optimal solution, each derivative depends on all the decision variables as well as the demand. Changing any of the inventory decision variables may potentially also affect all the others in a nonlinear way. Thus, it is hard to solve this problem for general probability functions and multiple warehouses.

In general, fulfillment rules do not always capture the optimal allocation strategy for every demand realization. However, the derivatives of the objective as seen in (13) allow the end-to-end learning framework to use the exact w^* instead of the $\hat{w}$, thus solving the problem without any further approximations.

In the following section we will present the solution for a special case network with only two warehouses in order to get intuition on the structure of the solution.

Special case: Two warehouses setting

To illustrate some of the ideas of this paper, we now consider a simple network of two warehouses where c_{11} and c_{22} are the costs of fulfilling a single unit of local demand in warehouses 1 and 2 respectively. These are lower than the cross-fulfillment cost $c_{12} = c_{21}$. In addition, they share the same holding and underage costs, that is, $h_1 = h_2 = h$ and $u_1 = u_2 = u$. We are interested in optimizing with respect to q_1 and q_2. In that case, system (13) simplifies to:

$$\begin{aligned} k &= P(D_1 + D_2 < q_1^* + q_2^*) \\ F_{D_1}(q_1^*) &= \frac{u - c_{11} - (h + u - c_{12}) * k}{c_{12} - c_{11}} \\ F_{D_2}(q_2^*) &= \frac{u - c_{22} - (h + u - c_{12}) * k}{c_{12} - c_{22}} \end{aligned} \quad (14)$$

As a comparison, if we ignore the cross fulfillment effects and solve each problem individually using the classical newsvendor approach (see (Arrow, Harris, and Marschak 1951)) then the optimal solution satisfies:

$$F_{D_i}(q_i^*) = \frac{u_i}{u_i + h_i}. \quad (15)$$

To solve system (14) we need to find a $k \in [0, 1]$. Notice that k is the overall service level across the network. Since both $F_{D_1}(q_1^*)$ and $F_{D_2}(q_2^*)$ are decreasing with respect to k, we can efficiently search for the optimal value using a bisection search method.

Notice that k is dependent of the correlation of the demand among the two warehouses. Negative correlation translates into higher service level and this in turn implies smaller optimal inventory across the network. In the extreme case where the optimal k approaches 100%, the solution of each location coincides with the solution of the classic newsvendor setting (see (15)) with holding costs h and underage cost $c_{12} - c_{11} - h$ and $c_{12} - c_{22} - h$ respectively. Since $k = 1$, there will be no loss of demand in each location, and therefore, the new underage cost is the extra cost of shipping from the farthest location minus the holding costs for not having to store this extra unit.

Sample Experiments

We now apply our approach to a 2-location NCF problem. As a baseline, we compare against a traditional two-stage approach in which the stocking decision is made by a forecast function fitted to predict only the demand, independent of optimization problem We also compare against an end-to-end approach using a simple fulfillment rule as described earlier. In this simple 2-location case, the optimal solution also follows the following fulfillment rule: each location fulfills as much demand as it has product stocked, and ship any remaining product from one location to the other if the other still has unfulfilled demand. As noted earlier, this allows us to exactly compute the cost as a differentiable function of our decisions.

To simplify the problem in (10), we assume that the forecast $D_\theta(x)$ is a point forecast. So, our stocking decision q is simply the forecast quantity $q^*(D_\theta(x)) = D_\theta(x)$. In the table below we summarize the fulfillment costs of each approach's stocking decisions. The holding cost at each location is 1 and the lost sale cost is 4. And we choose a uniform shipping cost between all nodes. Note that a shipping cost of 4 is equivalent to not allowing any cross-shipment: the cost is the same as incurring a lost sale cost. Therefore, we see the gap between our approaches and the two-stage approach decrease. The end-to-end methods still show an improvement because they are still able to capture the newsvendor aspect of the problem and can stock at a better quantile. In each case, the models learned by end-to-end training generate a tailored θ for each cross-cost, while the two-stage method learns the same value. Because the rule-based approach is optimal for the 2-location case, it outperforms the ProjectNet. However, in general there may not exist any rule that describes the optimal solution, in which case the ProjectNet would have an advantage.

	Predict and Optimize		
Cross-Cost	ProjectNet	Rule-Based	Two-Stage
1	1.91	1.91	2.75
2	3.20	3.10	3.72
3	4.30	4.24	4.68
4	5.40	5.27	5.64

Table 3: Feature data is generated entry-wise at random using a Gaussian distribution. We consider the demand to be a linear function of the features with added Gaussian noise. Each method is trained to learn a linear function that makes a decision for the stocking level at both warehouse locations.

References

Arrow, K. J.; Harris, T.; and Marschak, J. 1951. Optimal inventory policy. *Econometrica: Journal of the Econometric Society*, 250–272.

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

Donti, P.; Kolter, Z.; and Amos, B. 2017. Task-based End-to-end Model Learning in Stochastic Optimization. In *NIPS*.

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

Elmachtoub, A. N.; and Grigas, P. 2021. Smart "Predict, then Optimize". *Management Science*, 0(0): null.

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

Gubin, L.; Polyak, B.; and Raik, E. 1967. The method of projections for finding the common point of convex sets. *USSR Computational Mathematics and Mathematical Physics*, 7(6): 1–24.

Kotary, J.; Fioretto, F.; Van Hentenryck, P.; and Wilder, B. 2021. End-to-End Constrained Optimization Learning: A Survey. *CoRR*, abs/2103.16378.

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.