

Applying Deep Learning to the Newsvendor Problem

Afshin Oroojlooy, Lawrence Snyder & Martin Takáč


To cite this article: Afshin Oroojlooy, Lawrence Snyder & Martin Takáč (2019): Applying Deep Learning to the Newsvendor Problem, IISE Transactions, DOI: [10.1080/24725854.2019.1632502](https://doi.org/10.1080/24725854.2019.1632502)

To link to this article: <https://doi.org/10.1080/24725854.2019.1632502>



Accepted author version posted online: 20 Jun 2019.



Submit your article to this journal 



Article views: 107

View Crossmark data 

Applying Deep Learning to the Newsvendor Problem

Afshin Oroojlooy, Lawrence Snyder, Martin Takáč

Department of Industrial and Systems Engineering

Lehigh University, Bethlehem, PA, USA

Abstract

The newsvendor problem is one of the most basic and widely applied inventory models. If the probability distribution of the demand is known, the problem can be solved analytically. However, approximating the probability distribution is not easy and is prone to error; therefore, the resulting solution to the newsvendor problem may be not optimal. To address this issue, we propose an algorithm based on deep learning that optimizes the order quantities for all products based on features of the demand data. Our algorithm integrates the forecasting and inventory-optimization steps, rather than solving them separately, as is typically done, and does not require knowledge of the probability distributions of the demand. One can view the optimal order quantities as the labels in the deep neural network. However, unlike most deep learning applications, our model does not know the true labels (order quantities), but rather learns them during the training. Numerical experiments on real-world data suggest that our algorithm outperforms other approaches, including data-driven and machine learning approaches, especially for demands with high volatility. Finally, in order to show how this approach can be used for other inventory optimization problems, we provide an extension for (r, Q) policies.

1 Introduction

The newsvendor problem optimizes the inventory of a perishable good. Perishable goods are those that have a limited selling season; they include fresh produce, newspapers, airline tickets, and fashion goods. The newsvendor problem assumes that the company purchases the goods at the beginning of a time period and sells them during the period. At the end of the period, unsold goods must be discarded, incurring a *holding cost* (sometimes referred to as an overage cost). In addition, if it runs out of the goods in the middle of the period, it incurs a *shortage cost* (or underage cost), losing potential profit. Therefore, the company wants to choose the order quantity that minimizes the expected sum of the two costs described above. The problem dates back to Edgeworth (1888); see Porteus (2008) for a history and Zipkin (2000), Porteus (2002), and Snyder and Shen (2019), among others, for textbook discussions.

The optimal order quantity for the newsvendor problem can be obtained by solving the following optimization problem:

$$\min_y C(y) = \mathbb{E}_d [c_p (d - y)^+ + c_h (y - d)^+], \quad (1)$$

where d is the random demand, y is the order quantity, c_p and c_h are the per-unit shortage and holding costs (respectively), and $(a)^+ := \max\{0, a\}$. In the classical version of the problem, the shape of the demand distribution (e.g., normal) is known, and the distribution parameters are either known or estimated using available (training) data. If $F(\cdot)$ is the cumulative density function of the demand distribution and $F^{-1}(\cdot)$ is its inverse, then the optimal solution of (1) can be obtained as

$$y^* = F^{-1}\left(\frac{c_p}{c_p + c_h}\right) = F^{-1}(\alpha), \quad (2)$$

where $\alpha = c_p / (c_p + c_h)$ (see, e.g., Snyder and Shen (2019)).

Extensions of the newsvendor problem are too numerous to enumerate here (see Choi (2012) for examples); instead, we mention two extensions that are

relevant to our model. First, in real-world problems, companies rarely manage only a single item, so it is important for the model to provide solutions for multiple items. Second, companies often have access to some additional data—called *features*—along with the demand information. These might include weather conditions, day of the week, month of the year, store location, and so on (Ban and Rudin 2018). The goal is to choose today's order quantity, given the observation of today's features. We will call this problem the *multi-feature newsvendor (MFNV) problem*. In this paper, we propose an approach for solving this problem that is based on deep learning, i.e., deep neural networks (DNN).

The remainder of this paper is structured as follows. A brief summary of the literature relevant to the MFNV problem is presented in Section 2. Section 3 presents the details of the proposed algorithm. Numerical experiments are provided in Section 4. Section 5 introduces an extension of the approach for (r, Q) policies, and the conclusion and a discussion of future research complete the paper in Section 6.

2 Literature Review

2.1 Current State of the Art

Currently, there are five main approaches in the literature that are applicable to the MFNV. The first category, which we will call the *estimate-as-solution* (EAS) approach, involves forecasting the demand and then simply using it as the order quantity. The forecasting usually is done via classical approaches like ARIMA, TRANSFER, and GARCH models (Box et al. 2015, Shumway and Stoffer 2010); or sometimes by deep neural networks (Efendigil et al. 2009, Qiu et al. 2014b, Vieira 2015). Although EAS is a relatively naive approach for solving the MFNV—unlike the subsequent four approaches—it is common both in practice and in the literature; see below. This approach involves first clustering the demand observations, then forecasting the demand, and then simply treating the point forecast as a deterministic demand value, i.e., setting the order quantity equal to the forecast. By clustering, we mean that all demand observations that have the same feature

values are put together in a set, called a cluster. For example, when there are 100 demand records for two products in two stores, there are four clusters, and on average each cluster has 25 records. The forecast may be performed in a number of ways, some of which we review in the next few paragraphs.

Figure 1 illustrates EAS and the other four approaches. In the figure, each square represents a cluster and the arrows indicate the steps used to move from the clustered data to the end result, i.e., the newsvendor solution.

Figure 1(a) shows that in the EAS approach, we simply calculate an estimate $\hat{\mu}_k$ of the mean of the demands in cluster k and then set the order quantity equal to $\hat{\mu}_k$.

This approach ignores the key insight from the newsvendor problem, namely, that we should not simply order the mean demand, but rather choose a level that strikes a balance between holding and stockout costs using the distribution of the demand. Nevertheless, the approach has been used in the literature on the newsvendor problem and other production–inventory-type problems. For example, [Yu et al. \(2013\)](#) propose a support vector machine (SVM) model to forecast newspaper demands at different types of stores, along with 32 other features. [Chi et al. \(2007\)](#) propose a SVM model to determine the replenishment point in a vendor-managed replenishment system, and a genetic algorithm is used to solve it. The common theme in these papers is that they provide only a forecast of the demand, which is then be treated as the solution to the MFNV or other optimization problem. This is the EAS approach.

The second approach for solving MFNV-type problems, which [Ban and Rudin \(2018\)](#) refer to as *separated estimation and optimization* (SEO), involves first estimating (forecasting) the demand distribution and then plugging the estimate into an optimization problem such as the classical newsvendor problem. The estimation step is performed similarly as in the EAS approach except that we estimate more than just the mean. For example, we might estimate both mean (μ_k) and standard deviation (σ_k) for each cluster, which we can then use in the optimization step. (See

Figure 1(b), in which we estimate the standard deviation in cluster k , σ_k , in addition to the mean μ_k , and then use the estimates $\hat{\mu}_k$ and $\hat{\sigma}_k$ as inputs to the classical newsvendor problem.) Or we might use the σ that was assumed for the error term in a regression model. The main disadvantage of this approach is that it requires us to assume a particular form of the demand distribution (e.g., normal), whereas demand distributions in practice are often unknown or do not follow a regular form. A secondary issue is that we compound the data-estimation error with model-optimality error. Ban and Rudin (2018) show that for some realistic settings, the SEO approach is provably suboptimal. This idea is used widely in practice and in the literature; a broad list of research that uses this approach is given by Turken et al. (2012). Ban and Rudin (2018) analyze it as a straw-man against which to compare their solution approach.

The third approach was proposed by Bertsimas and Thiele (2005) for the classical newsvendor problem. Their approach involves sorting the demand observations in ascending order $d_1 \leq d_2 \leq \dots \leq d_n$ and then estimating the α th quantile of the demand distribution, $F^{-1}(\alpha)$, using the observation that falls $100\alpha\%$ of the way through the sorted list, i.e., it selects the demand d_j such that $j = \left\lceil n \frac{c_p}{c_p + c_h} \right\rceil$. This quantile is then used as the order quantity, in light of (2). Since they approximate the α th quantile, we refer to their method as the *empirical quantile* (EQ) method. (See Figure 1(c), which depicts an adjusted EQ algorithm with clustered data: demands are sorted in each cluster, with the solution set equal to the α th quantile of the resulting implied demand distribution.) Importantly, EQ does not assume a particular form of the demand distribution and does not approximate the probability distribution, so it avoids those pitfalls. However, an important shortcoming of this approach in our context is that it does not use the information from features. In principle, one could extend their approach to the MFNV by first clustering the demand observations and then applying their method to each cluster. However, similar to the classical newsvendor algorithm, this would only allow it to consider categorical features and not continuous features, which are common in supply

chain demand data, e.g., [Ali and Yaman \(2013\)](#) and [Ban and Rudin \(2018\)](#). Moreover, even if we use this clustering approach, the method cannot utilize any knowledge from other data clusters, which contain valuable information that can be useful for all other clusters. Finally, when there is volatility among the training data, the estimated quantile may not be sufficiently accurate, and the accuracy of EQ approach tends to be worse.

In the newsvendor problem, the optimal solution is a certain quantile of the demand distribution. Thus, the problem can be modeled as a quantile regression problem, in a manner similar to the empirical quantile model of [Bertsimas and Thiele \(2005\)](#). [Taylor \(2000\)](#) was the first to propose the use of neural networks as a nonlinear approximator of the quantile regression to get a conditional density of multi-period financial returns. Subsequently, several papers used quantile-regression neural networks to obtain a quantile regression value. For example, [Cannon \(2011\)](#) uses a quantile-regression neural network to predict daily precipitation; [El-Telbany \(2014\)](#) uses it to predict drug activities; and [Xu et al. \(2016\)](#) uses a quantile autoregression neural network to evaluate value-at-risk. One can consider our approach as a quantile-regression neural network for the newsvendor problem. However, our approach is much more general and can be applied to other inventory optimization problems, even those whose optimal solutions is not simply a quantile, provided that a closed-form cost function exists. To demonstrate this, in Section 5 we extend our approach to solve an inventory problem that does not have a quantile-type solution, namely, optimizing the parameters of an (r, Q) policy.

A fourth approach for solving MFNV-type problems can be derived from the method proposed by [Bertsimas and Kallus \(2014\)](#), which applies several machine learning (ML) methods on a general optimization problem given by

$$y^*(x) = \underset{y}{\operatorname{argmin}} \mathbb{E}[c(y, d) | x], \quad (3)$$

where $\{(x_1, d_1), \dots, (x_N, d_N)\}$ are the available data—in particular, x_i is a d -dimensional vector of feature values and d_i is the uncertain quantity of

interest, e.g., demand values—and y is the decision variable. They test five algorithms to optimize (3): k -nearest neighbor (KNN), random forest (RF), kernel method, classification and regression trees (CART), and locally weighted scatterplot smoothing (LOESS). They use sample average approximation (SAA) as a baseline, and each algorithm provides substitute weights for the SAA method. For example, KNN identifies the set of k nearest historical records to the new observation x such that

$$\mathcal{N}(x) = \left\{ i = 1, \dots, n : \sum_{j=1}^n \mathbb{I}\{\|x - x_i\| \geq \|x - x_j\|\} \leq k \right\}.$$

Bertsimas and Kallus (2014) assign weights $w_i = 1/k$ for all $i \in \mathcal{N}(x)$ (and zero otherwise) and call a weighted SAA; for example, if applied to the newsvendor problem, the SAA might take the form

$$q = \inf \left\{ d_j : \sum_{i=1}^j w_i \geq \frac{c_p}{c_p + c_h} \right\}, \quad (4)$$

where d_j are the ascending sorted demands. (See Figure 1(d), where the weight w_k^r for each sample r in cluster k is obtained and the sorted demands in each cluster determines the solution via (4).) Similarly, in RF, there are T trees. The weight of each observation is obtained using

$$w_i = \frac{1}{T} \sum_{t=1}^T \frac{\mathbb{I}\{R^t(x) = R^t(x_i)\}}{|\{j : R^t(x_j) = R^t(x_i)\}|},$$

where $R^t(x)$ is the region of tree t that observation x is in. In other words, the RF algorithm counts all trees in which the new observation x is in the same region as historical observation x_i , $i = 1, \dots, n$, and normalizes them over all observations in tree t that have the same region. Finally, it normalizes the weights over all trees. Using these weights, the method of Bertsimas and Kallus (2014) as applied to the newsvendor problem calls the weighted SAA (4) to get the order quantity. Bertsimas and Kallus (2014) discuss asymptotic convergence of their methods and compare their performance with that of SAA.

The fifth approach for the MFNV, and the one that is closest to our proposed approach, was introduced by [Ban and Rudin \(2018\)](#); we refer to it as the *linear machine learning* (LML) method. They postulate that the optimal order quantity is related to the demand features via a linear function; that is, that $y^* = w^T x$, where x is the vector of features and w is a vector of (unknown) weights. They estimate these weights by solving the following nonlinear optimization problem, essentially fitting the solution using the newsvendor cost:

$$\begin{aligned} \min_w \quad & \frac{1}{n} \sum_{i=1}^n \left[c_p (d_i - w^T x_i)^+ + c_h (w^T x_i - d_i)^+ \right] + \lambda \|w\|_k^2 \\ \text{s.t.} \quad & (d_i - w^T x_i)^+ \geq d_i - w_1 - \sum_{j=2}^p w_j x_i^j; \quad \forall i = 1, \dots, n \\ & (w^T x_i - d_i)^+ \geq w_1 + \sum_{j=2}^p w_j x_i^j - d_i; \quad \forall i = 1, \dots, n \end{aligned} \quad (5)$$

where n is the number of observations, p is the number of features, and $\lambda \|w\|_k^2$ is a regularization term. (See Figure 1(e), which shows that a non-linear programming model is solved to obtain weights w^* , which then determine the order quantity) The LML method avoids having to cluster the data, as well as having to specify the form of the demand distribution. [Ban and Rudin \(2018\)](#) comprehensively analyze the effects of adding nonlinear combinations of features into the feature space, as well as the effects of regularization and of overfitting. (For more theoretical details on these concepts, see [Smola and Schölkopf \(2004\)](#).) However, this model does not work well when $p \gg n$, and its learning is limited to the current training data. In addition, if the training data contains only a small number of observations for each combination of the features, the model learns poorly. Finally, it makes the strong assumption that x and y^* have a linear relationship. We drop this assumption in our model and instead use DNN to quantify the relationship between x and y^* ; see Section 3. [Ban and Rudin \(2018\)](#) also propose a kernel regression (KR) model to optimize the order quantity, in which weighted historical demands are used to build an empirical cdf of the

demand. The weights are proportional to the distance of the newly observed feature value with historical feature values, i.e.,

$$w_i = \frac{K(x - x_i)}{\sum_{j=1}^n K(x - x_j)},$$

where $K(u) = \exp(-\|u\|_2^2 / 2h) / \sqrt{2\pi}$ and h is the kernel bandwidth that has to be tuned. Then they call weighted SAA (4) to obtain the order quantity. In addition, they provide a mathematical analysis of the generalization errors associated with each method.

In addition to the five methods discussed above, there is a large body of literature on data-driven inventory management that assumes we do not know the demand distribution and instead must directly use the data to make a decision. Besbes and Muharremoglu (2013) consider censored data (in which some demands cannot be observed due to stockouts) in the newsvendor problem. The paper proposes three models and algorithms to minimize the regret when real, censored, and partially censored demand are available. They propose an EQ-type algorithm (discussed above) for observable demand. For censored and partially censored demand, they propose two algorithms, as well as lower and upper bounds on the regret value for all algorithms. Burnetas and Smith (2000) propose an adaptive model to optimize price and order quantity for perishable products with an unknown demand distribution, assuming historical data of censored sales are available. They assume that the demand is continuous and propose two algorithms, one for a fixed price and another for the pricing/ordering problem. Their algorithm for choosing the order quantity provides an adaptive policy and works even when there is nearly no historical information, so it is suitable for new products. It starts from an arbitrary point q_0 and iteratively updates it with some learning rate and information about whether or not the order quantity q_t was sufficient to satisfy the demand in period t .

Neither of these two papers use features, which is the key aspect of our problem. One data-driven approach that does use features is by [Ban et al. \(2017\)](#), who propose a separated estimation and optimization model to choose the order quantity for new, short-life-cycle products from multiple suppliers over a finite time horizon, assuming that each demand has some feature information. They propose a data-driven algorithm, called the residual tree method, which is an extension of the scenario tree method from stochastic programming, and prove that this method is asymptotically optimal as the size of the data set grows. Their approach has separate steps for estimation (using regression) and optimization (using stochastic linear programming). Although their problem has some similarities to ours, such as utilizing data features, it is not immediately applicable since it is designed for finite-horizon problems with multiple suppliers, which if adjusted for a single product and single supplier, would simply result in a newsvendor problem with a fixed predicted demand.

2.2 Deep Learning

In this paper, we develop a new approach to solve the newsvendor problem with data features, based on deep learning. Deep learning, or deep neural networks (DNN), is a branch of machine learning that aims to build a model between inputs and outputs. Deep learning has many applications in image processing, speech recognition, drug and genomics discovery, time series forecasting, weather prediction, and—most relevant to our work—demand prediction. On the other hand, one major criticism of deep learning (in non-vision-based tasks) is that it lacks interpretability—that is, it is hard for a user to discern a relationship between model inputs and outputs; see, e.g. [Lipton \(2016\)](#). In addition, it usually needs careful hyper-parameter tuning, and the training process can take many hours or even days. We provide only a brief overview of deep learning here; for comprehensive reviews of the algorithm and its applications, see [Goodfellow et al. \(2016\)](#), [Schmidhuber \(2015\)](#), [LeCun et al. \(2015\)](#), [Deng et al. \(2013\)](#), [Qiu et al. \(2014a\)](#), [Shi et al. \(2015\)](#), and [Långkvist et al. \(2014\)](#).

DNN uses a cascade of many layers of linear or nonlinear functions to obtain the output values from inputs. A general view of a DNN is shown in Figure 2. At each node j ($j=1, \dots, n$) of a given layer l ($l=1, \dots, L$), the *input value*

$$z_j^l = \sum_{i=1}^n a_i^{l-1} w_{ij} \quad (6)$$

is calculated. A function $g_j^l(z_j^l)$, called the *activation function*, transforms the input in order to determine the output value of the node. The value of $g_j^l(z_j^l)$ is called the activation of the node, and is denoted by a_j^l . Typically, all nodes in the network have similar $g_j^l(\cdot)$ functions. The most commonly used activation functions are the sigmoid ($1/(1+e^{-z_j^l})$), tanh ($(1-e^{-2z_j^l})/(1+e^{-2z_j^l})$), and Relu ($\max\{0, z_j^l\}$) functions, which add non-linearity into the model. (For more details, see [LeCun et al. \(2015\)](#), [Goodfellow et al. \(2016\)](#)). The activation value of each node is the input for the next layer, and finally, the activation values of the nodes in the last layer determine the output values of the network. The general flow of the calculations between two layers of the DNN, highlighting z_j^l , a_j^l , w_{jk} , and z_j^{l+1} , is shown in Figure 2.

The goal of the DNN is to determine the weights w of the network such that a given set of inputs results in a true set of outputs. A loss function is used to measure the closeness of the outputs of the model and the true values. The most common loss functions are the hinge, logistic regression, softmax, and Euclidean loss functions. The goal of the network is to provide a small loss value, i.e., to optimize:

$$\min_w \frac{1}{n} \sum_{i=1}^n E(\theta(x_i; w), y_i) + \lambda R(w),$$

where $E(\cdot)$ is the loss function, w is the matrix of the weights, x_i is the vector of the inputs from the i th instance, $\theta(\cdot)$ is the DNN function, and $R(w)$ is a regularization function with weight λ . The regularization term prevents over-fitting and is typically the ℓ_1 or ℓ_2 norm of the weights. (Over-fitting means that

the model learns to do well on the training set but does not extend to the out-of-training samples; this is to be avoided.) Finally, y_i is the target value that the DNN attempts to predict, and in the context of the newsvendor problem, it is the optimal order quantity. In supervised problems like image classification the label is known, though in the newsvendor problem the optimal order quantity is not known and we provide a way to learn it. In particular, we use the demand observations in the training phase to obtain the newsvendor cost and then update the weights of the network in a direction that minimize the newsvendor cost.

In each DNN, the number of layers, the number of nodes in each layer, the activation function inside each node, and the loss function have to be determined. After selecting those characteristics and building the network, DNN starts with some random initial solution. In each iteration, the activation values and the loss function are calculated. Then, the back-propagation algorithm obtains the gradient of the network and, using one of several optimization algorithms (Rumelhart et al. 1988), the new weights are determined. The most common optimization algorithms are gradient descent, stochastic gradient descent (SGD), SGD with momentum, and Adam optimizer. (For details on each optimization algorithm see Goodfellow et al. (2016).) This procedure is performed iteratively until some stopping condition is reached; typical stopping conditions are (a) reaching a maximum number of iterations and (b) attaining $\|\nabla_w \ell(\theta(x_i; w), y_i)\| \leq \epsilon$ through the back-propagation algorithm.

Since the number of instances, i.e., the number of training records, is typically large, it is common (Goodfellow et al. 2016, Bottou 2010) to use a stochastic approximation of the objective function. That is, in each iteration, a mini-batch of the instances is selected and the objective is calculated only for those instances. This approximation does not affect the provable convergence of the method. For example, in networks with sigmoid activation functions in which a quadratic loss function is used, the loss function asymptotically

converges to zero if either gradient descent or stochastic gradient descent are used (Tesauro et al. 1989, Bottou 2010).

2.3 Our Contribution

To adapt the deep learning algorithm for the newsvendor problem with data features, we propose a revised loss function, which considers the impact of inventory shortage and holding costs. The revised loss function *allows the deep learning algorithm to obtain the minimizer of the newsvendor cost function directly, rather than first estimating the demand distribution and then choosing an order quantity*. Unlike in image classification, in the newsvendor problem the training set does not contain any true labels (order quantities), only feature values and corresponding demand observations. We use the feature values as the inputs of the DNN to obtain the order quantity. For this purpose, we use the demand observations only in the training phase to learn the optimal order quantities. The execution phase takes new feature values as inputs, outputs order quantities, and runs in real-time.

In the presence of sufficient historical data, this approach can solve problems with known probability distributions as accurately as (2) solves them.

However, the real value of our approach is that it is effective for problems with small quantities of historical data, problems with unknown/unfitted probability distributions, or problems with volatile historical data—all cases for which the current approaches fail.

3 Deep Learning Algorithm for Newsvendor with Data Features

In this section, we present the details of our approach for solving the newsvendor problem with data features. Assume there are n historical demand observations for m products. Also, for each demand observation, the values of p features are known. That is, the data can be represented as

$$\{(x_i^1, d_i^1), \dots, (x_i^m, d_i^m)\}_{i=1}^n,$$

where $x_i^q \in \mathbb{R}^p$ and $d_i^q \in \mathbb{R}$ for $i = 1, \dots, n$ and $q = 1, \dots, m$. The problem is formulated mathematically in (7) for a given period i , $i = 1, \dots, n$, resulting in the order quantities y_i^1, \dots, y_i^m :

$$E_i = \min_{y_i^1, \dots, y_i^m} \frac{1}{m} \left[\sum_{q=1}^m (c_h(y_i^q - d_i^q)^+ + c_p(d_i^q - y_i^q)^+) \right], \quad (7)$$

where E_i is the loss value of period i and $E = \frac{1}{n} \sum_{i=1}^n E_i$ is the average loss value. Since at least one of the two terms in each term of the sum must be zero, the loss function (7) can be written as:

$$E_i = \frac{1}{m} \sum_{q=1}^m E_i^q \quad (8)$$

$$E_i^q = \begin{cases} c_p(d_i^q - y_i^q), & \text{if } y_i^q < d_i^q, \\ c_h(y_i^q - d_i^q), & \text{if } d_i^q \leq y_i^q. \end{cases}$$

E , the average of the E_i values defined in (7), serves as the value to be minimized by the DNN, i.e., the DNN finds values for the variables y_i^1, \dots, y_i^m that obtain the minimum average cost. In other words, for each input x_i^q , the DNN obtains a single output y_i^q that will serve as the order quantity for the corresponding input features. Note though, that the decision variables of the model are not the y_i^q values directly; rather, they are the weights of the neural network, i.e., w_{jk} for $j = 1, \dots, nn_l$, $k = 1, \dots, nn_k$, and $l, k \in \{1, \dots, L\}$. The order quantity y_i^q can be written explicitly as a function of those weights, so that the output of the network, i.e., a_0^L , is the order quantity. The order quantity outputs are optimized through the training process by iteratively updating the weights of the network to minimize the loss function (7).

Note that, in contrast, the EAS approach would simply seek to *predict* d_i^q rather than to *optimize* y_i^q —say, by replacing the term inside the brackets in (7) with

$$\sum_{q=1}^m ||y_i^q - d_i^q||.$$

Using the newsvendor cost in the loss function, rather than simply the distance to the true demand, allows our model to capture the tradeoff between the holding and stockout costs, true to the aim of the newsvendor problem.

As noted above, there are many studies on the application of deep learning for demand prediction (see [Shi et al. \(2015\)](#)). Most of this research uses the Euclidean loss function (see [Qiu et al. \(2014b\)](#)). However, the demand forecast is an estimate of the first moment of the demand probability distribution; it is not, however, the optimal solution of model (7). Therefore, another optimization problem must be solved to translate the demand forecasts into a set of order quantities. This is the separated estimation and optimization (SEO) approach described in Section 2.1, which may result in a non-optimal order quantity ([Ban and Rudin 2018](#)). To address this issue, we propose using the newsvendor cost function (7) as the loss function (as well as a variant of it, essentially a revised Euclidean loss function, described in the next paragraph), so that instead of simply predicting the demand, the DNN minimizes the newsvendor cost function. Thus, running the corresponding deep learning algorithm gives the order quantity directly.

We found that squaring the cost for each product in (7) sometimes leads to better solutions, since the function is smooth, and the gradient is available in the whole solution space. Therefore, we also test the following revised Euclidean loss function:

$$E_i = \min_{y_i^1, \dots, y_i^m} \frac{1}{m} \left[\sum_{q=1}^m \left[c_p (d_i^q - y_i^q)^+ + c_h (y_i^q - d_i^q)^+ \right]^2 \right] \quad (9)$$

which penalizes the order quantities that are far from d_i much more than those that are close. Then we have

$$E_i^q = \begin{cases} \frac{1}{2} \|c_p(d_i^q - y_i^q)\|_2^2, & \text{if } y_i^q < d_i^q, \\ \frac{1}{2} \|c_h(y_i^q - d_i^q)\|_2^2, & \text{if } d_i^q \leq y_i^q. \end{cases} \quad (10)$$

The two propositions that follow provide the gradients of the loss functions with respect to the weights of the network. In both propositions, i is one of the samples, w_{jk} represents a weight in the network between two arbitrary nodes j and k in layers l and $l+1$,

$$a_j^l = g_j^l(z_j^l) = \frac{\partial(z_k^l)}{\partial w_{jk}} \quad (11)$$

is the activation function value of node j , and

$$\delta_j^l = \frac{\partial E_i^q}{\partial z_j^l} = \frac{\partial E_i^q}{\partial a_j^l} \frac{\partial a_j^l}{\partial z_j^l} = \frac{\partial E_i^q}{\partial a_j^l} (g_j^l)'(z_j^l). \quad (12)$$

Also, let

$$\begin{aligned} \delta_j^l(p) &= c_p(g_j^l)'(z_j^l) \\ \delta_j^l(h) &= c_h(g_j^l)'(z_j^l) \end{aligned} \quad (13)$$

denote the corresponding δ_j^l for the shortage and excess cases, respectively. Proofs of both propositions are provided in Appendix A.

Proposition 1. *The gradient with respect to the weights of the network for loss function (8) is:*

$$\frac{\partial E_i^q}{\partial w_{jk}} = \begin{cases} a_j^l \delta_j^l(p) & \text{if } y_i^q < d_i^q, \\ a_j^l \delta_j^l(h) & \text{if } d_i^q \leq y_i^q. \end{cases} \quad (14)$$

Proposition 2. *The gradient with respect to the weights of the network for loss function (10) is:*

$$\frac{\partial E_i^q}{\partial w_{jk}} = \begin{cases} (d_i^q - y_i^q) a_j^l \delta_j^l(p), & \text{if } y_i^q < d_i^q \\ (y_i^q - d_i^q) a_j^l \delta_j^l(h), & \text{if } d_i^q \leq y_i^q. \end{cases} \quad (15)$$

Our deep learning algorithm uses gradients (14) and (15) under the proposed loss functions (8) and (10), respectively, to iteratively update the weights of the networks. In order to obtain the new weights, an SGD algorithm with momentum is called, with a fixed momentum of 0.9. This gives us two different DNN models, using the linear loss function (8) and the quadratic loss function (10), which we call DNN- ℓ_1 and DNN- ℓ_2 , respectively.

In order to obtain a good structure for the DNN network, we use the HyperBand algorithm (Li et al. 2016). In particular, we generate 100 fully connected networks with random structures. In each, the number of hidden layers is randomly selected as either two or three (with equal probability). Let nn_l denote the number of nodes in layer l , then nn_1 is equal to the number of features. The number of nodes in each hidden layer is selected randomly based on the number of nodes in the previous layer. For networks with two hidden layers, we choose $nn_2 \in [0.5nn_1, 3nn_1]$, $nn_3 \in [0.5nn_2, nn_2]$, and $nn_4 = 1$. Similarly, for networks with three hidden layers, $nn_2 \in [0.5nn_1, 3nn_1]$, $nn_3 \in [0.5nn_2, 2nn_2]$, $nn_4 \in [0.5nn_3, nn_3]$, and $nn_5 = 1$. The nn_l values are drawn uniformly from the ranges given. For each network, the learning rate and regularization parameters are drawn uniformly from $[10^{-2}, 10^{-5}]$. In order to select the best network among these, following the HyperBand algorithm, we train each of the 100 networks for one epoch (which is a full pass over the training dataset), obtain the results on the test set, and then remove the worst 10% of the networks. We then run another epoch on the remaining networks and remove the worst 10%. This procedure iteratively repeats to obtain the final best networks.

4 Numerical Experiments

In this section, we discuss the results of our numerical experiments. In addition to implementing our deep learning models (DNN- ℓ_1 and DNN- ℓ_2), we implemented the EQ model by Bertsimas and Thiele (2005), modifying it so that first the demand observations are clustered according to the features and then EQ is applied to each cluster. We also implemented the LML and KR models by Ban and Rudin (2018) and the KNN and RF models by Bertsimas

and Kallus (2014). Finally, we test the SEO approach, obtaining the mean by training a DNN over the feature values and then assuming a normally distributed error term to use (2). We train the DNN with both ℓ_1 and ℓ_2 regularizers since we do the same for our DNN approach, and denote the corresponding results as SEO- ℓ_1 and SEO- ℓ_2 . Additionally, we provide results of another, simpler, version of the SEO approach in which we calculate the classical solution from (2) with parameters μ and σ set to the mean and standard deviation of the training data in each data cluster; the corresponding results are denoted by *parametric* SEO (PSEO). We do not include results for EAS since it is dominated by PSEO: PSEO uses the newsvendor solution based on estimates of μ and σ , whereas EAS simply sets the solution equal to the estimate of μ . In order to compare the results of the various methods, the order quantities were obtained with each algorithm and the corresponding cost function

$$Z = \sum_{i=1}^n \sum_{q=1}^m \left[c_p (d_i^q - y_i^q)^+ + c_h (y_i^q - d_i^q)^+ \right]$$

was calculated.

All of the deep learning experiments were done with TensorFlow (Abadi et al. 2016) in Python. (We have released our code as an open-source Python project.) ¹ Note that the deep learning, LML, KR, KNN, and RF algorithms are scale dependent, meaning that the tuned parameters of the problem for a given set of cost coefficients do not necessarily work for other values of the coefficients. Thus, we performed a separate tuning for each set of cost coefficients. In addition, we translated the categorical data features to binary representations (using one-hot encoding, in which each category is indicated by a binary variable). These two implementation details improve the accuracy of the learning algorithms. All computations were done on 16-core machines with cores of 1.8 GHz computation power and 32 GB of memory.

In what follows, we demonstrate the results of the ten algorithms in three separate experiments. First, in Section 4.1, we conduct experiments on a very

small data set in order to illustrate the differences among the methods. Second, the results of the ten algorithms on a real-world dataset are presented in Section 4.2. Finally, in Section 4.3, to determine the conditions under which deep learning outperforms the other algorithms on larger instances, we present the results of the ten approaches on several randomly generated datasets.

Note that there is an assumption inherent in all machine learning algorithms (including neural network, regression, random forest, k -nearest neighbor, etc.) that the underlying demand distribution is the same in the testing and training data, and that all samples are picked identically and independently. If this assumption is violated, there are no performance guarantees. Therefore, in all datasets, we randomly shuffle the available data so that this assumption holds.

4.1 Small Data Set

Consider the small, single-item instance whose demands are contained in Table 1.

In order to obtain the results of each algorithm, the first two weeks are used for training data and the third week is used for testing. To train the corresponding deep network, a fully connected network with one hidden layer is used. The network has eight binary input nodes for the day of week and item number. The hidden layer contains one sigmoid node, and in the output layer there is one inner product function. Thus, the network has nine variables.

Table 2 shows the results of the ten algorithms. The first column gives the cost coefficients. Note that we assume $c_p \geq c_h$ since this is nearly always true for real applications; however, none of the methods requires this. The table first lists the actual demand for each day, repeated from Table 1 for convenience. For each instance (i.e., set of cost coefficients), the table lists the order quantity generated by each algorithm for each day. The last column

lists the total cost of the solution returned by each algorithm, and the minimum costs for each instance are given in bold.

First consider the results of the EQ algorithm. The EQ algorithm uses c_h and c_p and returns the historical data value that is closest to the α th fractile, where $\alpha = c_p / (c_h + c_p)$. In this data set, there are only two observed historical data points for each day of the week. In particular, for $c_p / c_h = 1$, the EQ algorithm chose the smaller of the two demand values as the order quantity, and for $c_p / c_h > 1$, it chose the larger value. Since the testing data vector is nearly equal to the average of the two training data vectors, the difference between EQ's output and the real demand values is quite large, and consequently so is the cost. This is an example of how the EQ algorithm can fail when the historical data are volatile.

Consider the KNN algorithm. Since there are only two weeks of historical data, we opt to use all possible historical records without any validation and set $k = 14$. KNN gets the k historical records that are nearest to the new observation, each with a weight of $\frac{1}{k}$, and then chooses the point that weighted SAA selects. The demand of that point is the order quantity. So, as c_p / c_h increases, it selects larger values. However, the demands during the third week (the testing set) are close to the mean demand of the first two weeks (the training set); therefore, the increased order quantity chosen by KNN turns out to be too large. Similarly, in RF we select 2000 forests, and in KR we select $h = 0.5$ and use all data from the two weeks of the training set. Since both algorithms work with sorted demands, once c_p / c_h increases, they select larger demands from the training sets. Therefore, RF and KR also result in large cost values, for similar reasons as KNN.

Now consider the results of all versions of the SEO algorithm. For the case in which $c_h = c_p$ (which is not particularly realistic), SEO- ℓ_2 attains the best result among all the algorithms; however SEO- ℓ_1 does not perform well. PSEO's output is approximately equal to the mean demand, which happens to be

close to the week-3 demand values. This gives PSEO a cost of 2.5, which ties DNN- ℓ_2 for second place. For all other instances, however, the increased value of c_p / c_h results in an inflated order quantity and hence a larger cost.

Finally, both DNN- ℓ_1 and DNN- ℓ_2 outperform the LML algorithm by Ban and Rudin (2018), because LML uses a linear kernel, while DNN uses both a linear and non-linear kernel. Also, there are only two features in this data set, so LML has some difficulty to learn the relationship between the inputs and output. Finally, the small quantity of historical data negatively affects the performance of LML.

This small example shows some conditions under which DNN outperforms the other three algorithms. In the next section we show that similar results hold even for a real-world dataset.

4.2 Real-World Dataset

We tested the ten algorithms on a real-world dataset consisting of basket data from a retailer in 1997 and 1998 from Pentaho (2008). There are 13170 records for the demand of 24 different departments in each day and month, of which we use 75% for training and validation and the remainder for testing. We defined each cluster as a unique combination of item, day, and month, which resulted in 1856 clusters. The categorical data were transformed into their binary equivalents, resulting in 43 input features.

The results of each algorithm for 100 values of c_p and c_h are shown in Figure 3. In the figure, the vertical axis shows the normalized costs, i.e., the cost value of each algorithm divided by the corresponding DNN- ℓ_1 cost. The horizontal axis shows the ratio c_p / c_h for each instance. As before, most instances use $c_p \geq c_h$ to reflect real-world settings, though 32 instances use $c_p < c_h$ to test this situation as well.

As shown in Figure 3, for this data set, DNN- ℓ_1 outperforms the other algorithms for every value of c_p / c_h . Among the remaining algorithms, the results of SEO- ℓ_2 , SEO- ℓ_1 , KNN, and RF (in that order) are the closest to those

of DNN. On average, their corresponding cost ratios are 1.04, 1.08, 1.15, and 1.16, whereas the ratios for EQ, LML, KR, and PSEO are 1.26, 1.53, 1.16, 1.26, and 1.23 respectively. The average cost ratio of DNN- ℓ_2 is 1.13. Note that none of the other approaches are stable, in the sense that their cost ratios increase with the ratio c_p / c_h .

The superior performance of DNN- ℓ_1 vs. SEO- ℓ_1 and SEO- ℓ_2 suggests that *the strength of our method comes from its end-to-end nature, choosing order quantities directly from the data, and not simply from the improved performance of the deep learning models. If we use SEO but “boost” the estimation step by using deep learning instead of simpler forecasting methods, we still do not attain the performance of our integrated method.*

DNN- ℓ_2 requires more tuning than DNN- ℓ_1 , but the DNN- ℓ_2 curve in Figure 3 does not reflect this additional tuning. The need for additional tuning is suggested by the fact that DNN- ℓ_2 's loss value increases as c_p or c_h increase, suggesting that it might need a smaller learning rate (to avoid big jumps) and a larger regularization coefficient λ (to strike the right balance between cost and over-fitting). Thus, tuning DNN- ℓ_2 properly would require a larger search space for the learning rate and λ , which would make the procedure harder and more time consuming. In our experiment, we did not fully expend this extra effort; instead, we used the same procedure and search space to tune the network for both DNN- ℓ_1 and DNN- ℓ_2 , in order to compare them fairly.

Nevertheless, it is worth investigating how the performance of DNN- ℓ_2 could be improved if it is tuned more thoroughly. To that end, we selected integer values of $c_p / c_h = 3, \dots, 9$, and for each value, we applied more computational power and tuned the parameters using a grid search. We fixed the network as $[43, 350, 100, 1]$, tested it with 702 different parameters, and selected the best test result among them. The grid search procedure is explained in detail in Appendix B. The corresponding result is labeled as DNN- ℓ_2 -T in Figure 3. As the figure shows, this approach has better results than the original version of DNN- ℓ_2 ; however, DNN- ℓ_1 is still better.

The DNN algorithms execute more slowly than some of the other algorithms. For the basket dataset, the PSEO and EQ algorithms each execute in about 10 seconds. The DNN algorithm requires about 50 seconds (on a relatively large network, e.g., $[43,90,150,56,1]$) for each epoch of training, while the LML, KR, KNN, and RF algorithms require on average, respectively, about 40 seconds (per regularization value), 15 seconds (per bandwidth), 5 seconds (for a given k), and 4 seconds (per tree) for training for a given c_p and c_h . As the size of the search space for hyper-parameter tuning increases, so does the training time for DNN, LML, KR, RF, and KNN. For LML, we tested 30 different bandwidths— $2^h, h \in \{-20, \dots, 10\}$ —which resulted in 1200 seconds of training, on average. For KR, we tested bandwidth values of $10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 0.05, 0.1$, and 0.25 , with a total time of 110 seconds on average. KNN needs to tune k , for which we tested six values—5, 10, 15, 50, 100, and 200—which took 30 seconds on average. Similarly, for RF we tested five forest sizes—10, 20, 50, 100, and 150—which resulted in 1320 seconds of training on average. For DNN- ℓ_1 , DNN- ℓ_2 , SEO- ℓ_1 , and SEO- ℓ_2 we used the HyperBand algorithm to tune the network. We tested several different values of each of the hyper-parameters (as explained at the end of Section 3), resulting in a total of 881 epochs, which took 12.25 hours of training on average. The best network runs for 16 epochs, which took 600 seconds on average. Table 3 summarizes the hyper-parameters used during the tuning process for each method, and their approximate computation times. Note that the times reported in the table are for one instance of the basket dataset, i.e., one value of c_p / c_h .

On the other hand, DNN, SEO, and LML algorithms execute in less than one second, i.e., once the network is trained, the methods generate order quantities for new instances very quickly. In contrast, KR, KNN, and RF required approximately 15, 5, and $4t$ seconds, respectively, for inference, where t is the number of trees that is selected.

Since tuning the DNN hyper-parameters can be time-consuming, in Appendix C we propose a simple tuning-free network for the newsvendor problem.

Finally, we performed a small experiment to provide some intuition about which features have the most impact on the order quantity. In particular, we calculated the order quantity for each of the $7 \times 12 \times 24 = 2016$ possible combinations of the feature values, using the DNN model tuned for a uniform distribution with 100 clusters. For each individual feature value, we calculated the average order quantity; these are plotted in Figure 4. From the figure it is evident that—for this data set—the order quantity is affected most strongly by the product category, then by the day of the week, and then by the month of the year. The average order quantity ranges (max – min) for the product, day, and month are 682.9, 540.7, and 371.9, respectively.

This sort of approach could be used to analyze the results of the DNN algorithm for any set of categorical features. The results could be useful to managers attempting to decide whether to use a feature-based approach—including DNN or the other models discussed here—rather than treating the entire data set as a single cluster. For example, if the a supply chain manager for the supermarket data set did not have access to the product category labels, a feature-based optimization approach would be less valuable, since the day and month features provide less differentiation in the order quantities; in this case, ignoring the features and treating the entire data set as a single cluster would result in less error than it would if product category labels were available. Of course, these insights pertain only to this data set. We are not claiming that product is a stronger differentiator than month in general, but rather illustrating how the DNN model can be interpreted and used to generate such insights.

4.3 Randomly Generated Data

In this section we report on the results of an experiment using randomly generated data. This experiment allows us to test the methods on many more instances; however, the disadvantage is that these data are much cleaner than those typically encountered in real supply chains, i.e., they come from a single probability distribution with no noise. This should be kept in mind when interpreting these results. In short, the results in this section indicate that,

when the data are non-noisy, all of the methods perform more or less similarly, with some exceptions. In all cases, DNN's performance is competitive with, if not better than, the other methods; and since it also performs better on messier data sets (e.g., the real-world data set in Section 4.2), we recommend its use in general. We now present a more detailed discussion of this experiment.

We conducted tests using five different probability distributions for the demand (normal, lognormal, exponential, uniform, and beta distributions). For each distribution, we generated 257,500 records. The parameters for the five demand distributions are given in Table 4; these parameters were selected so as to provide reasonable demand values. All demand values are rounded to the nearest integer. Each group of 257,500 records is divided into training and validation (10,000 records) and testing (99 sets, each 2,500 records) sets.

In each of the distributions, the data were categorized into clusters, each representing a given combination of features. Like the real-world dataset, we considered three features: the day of the week, month of the year, and department. We varied the number of clusters (i.e., the number of possible combinations of the values of the features) from 1 to 200 while keeping the total number of records fixed at 257,500; thus, having more clusters is the same as having fewer records per cluster. In this experiment, an "instance" refers to a given combination of demand distribution (normal, exponential,...) and number of clusters (1, 10,...).

Each problem was solved for $c_p / c_h = 5$ using all ten algorithms (including both loss functions for DNN), without assuming any knowledge of the demand distribution. We conducted additional tests using additional c_p / c_h ratios; the results and conclusions were similar, so they are omitted here in the interest of conciseness.

In part, this experiment is designed to model the situation in which the decision maker does not know the true demand distribution. To that end, our implementations of the SEO and PSEO algorithm assumes the demands

come from a normal distribution (regardless of the true distribution for the dataset being tested), since this distribution is used frequently as the default distribution in practice. The other algorithms (DNN, LML, EQ, KNN, KR, and RF) do not assume any probability distribution. Additionally, since we know the underlying demand distributions, we also calculated and reported the optimal solution in each case. The average times required to tune or execute each of the algorithms, per instance, are similar to those in Table 3.

Figure 5 plots the average cost ratio (cost divided by optimal cost) for the five distributions. Each point on a given plot represents the average cost (over 99 testing sets) for one instance. Figure 6 contains magnified versions of the plots in Figure 5 for three of the distributions. From the plots, we can draw the following conclusions:

- If there is only a single cluster, then all ten algorithms produce nearly the same results. This case is essentially a classical newsvendor problem with 7,500 data observations, for which all algorithms do a good job of providing the order quantity in the test sets.
- As the number of clusters increases, i.e., the number of training samples in each cluster decreases, the methods begin to differentiate somewhat. In particular:
- DNN- ℓ_1 , SEO- ℓ_2 , PSEO, EQ, KR, and KNN perform the best and have roughly equal performance.
- The SEO methods perform well when the demands are normally distributed but less well otherwise. This is because one has to assume a demand distribution in order to use SEO, and we assumed normal. If the demands happen to come from a normal distribution, therefore, SEO works well. In practice, however, the demand distribution is usually unknown and often non-normal.
- SEO- ℓ_2 and EQ perform relatively well in general in this experiment because, when the data are non-noisy, it is easier to estimate a quantile. However, for both the small data set (Section 4.1) and the real-world data set (Section 4.2), which are noisier, SEO- ℓ_2 and EQ do not perform as well as in the simulated data.

- The performance of DNN- ℓ_2 is quite good *except* in the case of normal demands with 100 or 200 clusters. In these cases, the method would benefit from further tuning (similar to the additional tuning that we did for the basket data set in Section 4.2).
- LML and RF are nearly always worse than the other methods because there is not enough data for them to learn the distribution well. (As a result, we have omitted them from Figure 6.)

To confirm these findings statistically, Figures 7 and 8 plot 95% confidence intervals for each algorithm for normally and uniformly distributed demands (respectively). The confidence intervals are calculated using the mean and standard error of the cost ratio over the 99 test data sets. When two confidence intervals are non-overlapping, we can conclude that the performance of the two corresponding methods is statistically different. If a given method is excluded from a plot, it means that the method is much worse than the methods that are plotted. From these figures, we can draw the following conclusions:

- DNN- ℓ_1 is statistically better than all other methods for some cases (e.g., uniform demands with 100 clusters); is in statistical second place to PSEO for normal demands with 200 clusters and to DNN- ℓ_2 for uniform demands with 100 and 200 clusters; and is tied for first place in all other cases.
- PSEO is statistically better than all other methods for normal demands with 200 clusters and statistically worse than all other methods for uniform demands with any number of clusters. It is tied with other methods (not in first place) for most other instances.
- SEO- ℓ_2 in most cases is in a statistical tie with DNN- ℓ_1 . The exceptions are uniform demands with 10 and 100 clusters, and normal demands with 100 clusters.
- DNN- ℓ_2 , EQ, KNN, and KR are, in most cases, in a statistical tie.
- LML, SEO- ℓ_1 , and RF are statistically worse than all other methods, except in the case of normal demands with 1 cluster.

- In nearly every instance, no method obtains solutions that are statistically equal to the optimal solution. The exception is normal demands with 100 clusters, for which DNN- ℓ_1 is statistically tied with the optimal solution.

Suppose we take a naive approach toward the MFNV problem and ignore the data features, optimizing the inventory level as though there were only a single cluster. How significant an error is this? To answer this question, we solved the problem using DNN- ℓ_1 , grouping all of the data into a single cluster. (Note that this data set is different from the 1-cluster data sets discussed above. The data sets above assume there *is* only a single cluster, i.e., all demand records have identical feature values, whereas the data set here has multiple sets of feature values, but we are ignoring them to emulate the naive approach.) Figure 9 plots the ratio between the cost of the resulting solution and the cost of the DNN- ℓ_1 solution that accounts for the clusters, for the five probability distributions and for data sets with 10, 100, and 200 clusters. Clearly, the errors resulting from this naive approach can be significant: They range from 5.6% (for the exponential distribution with 200 clusters) to 677.9% (for the uniform distribution with 100 clusters). In general these errors will change with the probability distributions and their parameters, but it is clear that it is important to consider clusters when faced with featured data, and costly to ignore them.

4.4 Numerical Results: Summary

Our recommendations for which method to use are as follows. If the data set is noisy, like most real-world data sets, our experiments show that DNN is the most reliable algorithm, with the caveat that careful hyperparameter tuning is required. If the data are non-noisy (they come from a single probability distribution) and the number of historical samples is small (say, fewer than 10 records per combination of features), DNN tends to outperform the other methods. As the number of historical records begin to increase, either EQ, SEO, DNN, KR, RF, or KNN is a reasonable choice. Finally, if there are a large number of non-noisy historical demand records for each combination of

features (say, at least 10,000), then the algorithms all work roughly equally well, and it may be best to choose EQ or SEO, since they do not need any hyperparameter tuning.

5 DNN Model for (r, Q) Policies

In this section, we extend our DNN approach to optimize the parameters of an (r, Q) inventory policy, in order to demonstrate that the method can be adapted to other inventory problems, and especially to problems that cannot be solved simply by estimating the quantile of a probability distribution.

5.1 Model

Consider a continuous-review inventory optimization problem with stochastic demand, such that the mean demand per unit time is λ . Placing an order incurs a fixed cost K , and the order arrives after a deterministic lead time of $L \geq 0$ time units. Unmet demand is backordered. We assume the firm follows an (r, Q) inventory policy: Whenever the inventory position falls to r , an order of size Q is placed. The aim of the optimization problem is to determine r and Q .

If we know the true demand distribution, the optimal r and Q can be obtained by solving a convex optimization problem; see [Hadley and Whitin \(1963\)](#) or [Zheng \(1992\)](#). However, heuristic approaches are commonly used to obtain approximate values for r and Q , for a discussion of these, see [Snyder and Shen \(2019\)](#). We use the so-called expected-inventory level (EIL) approximation, which is arguably the most common approximation for the (r, Q) optimization problem. The EIL approximates the expected cost function as

$$g(r, Q) = c_h \left(r - \lambda L + \frac{Q}{2} \right) + \frac{K\lambda}{Q} + \frac{c_p \lambda n(r)}{Q}, \quad (16)$$

where

$$n(r) = \int_r^\infty (d - r) f(d) dd$$

and $f(d)$ is the demand distribution. The cost function (16) can be optimized through an iterative algorithm proposed by Hadley and Whitin (1963) (we will refer to this as the *EIL algorithm*), again assuming that the demand distribution is known.

Of course, in practice, the demand distribution is often not known, which is where DNN becomes a useful approach. In order to use DNN to obtain the policy parameters, we propose a DNN network similar to that used for the newsvendor problem, except that it has two outputs, r and Q . We use the cost function (16) as the loss function for the DNN, and in place of $n(r)$ we use the

unbiased estimator $\frac{1}{m} \sum_{i=1}^m (d_i - r_i)^+$. In addition, in order to avoid negative values for r and Q , we use r^+ and Q^+ in the DNN loss function, and also add a penalty for negative values of r and Q into the DNN loss function:

$$l(r, Q) = c_h \left(r^+ - \lambda L + \frac{Q^+}{2} \right) + \frac{K\lambda}{Q^+} + \frac{c_p \lambda n(r^+)}{Q^+} + \eta_Q Q^- + \eta_r r^-,$$

where η_r and η_Q are the penalty coefficients for negative r and Q , respectively.

The idea is as follows: We have a reliable estimate of the mean annual demand (λ). However, over the course of a year, the demand distribution changes, along with the features. Each time we need to place an order, we use the current features (without the actual demand value, since it is not observed yet) to determine r and Q ; we compare the current inventory position with r and place an order of size Q , if required. The EIL cost function essentially tells us the rate at which costs are accruing when we are placing the current order.

Note that, when we say that we use the current features to determine r and Q , we do not mean that r and Q are chosen as though the entire future would look like the current features. Rather, the model learns that when the features look like x , then the optimal action to take is y , anticipating the futures that might evolve from the state x . For example, if demands are usually small on Mondays and large on Tuesdays, then the model will learn that an order

placed on Monday should be sufficiently large to handle the demand spike on Tuesday, even though Monday's features by themselves suggest the demand will be small.

Figure 10 illustrates this process. The x -axis indicates a sample path of the clusters over time (first the system is in cluster 4, then cluster 7, etc.), as well as the magnitude of the demand distribution (high/medium/low) in each cluster. While the system is in cluster 7, the inventory position hits r_7 , so an order of size Q_7 is placed. The reorder point r_7 is relatively large because the subsequent lead time will encompass high- and medium-demand clusters. (Of course, when we place the order during cluster 7, we don't know which clusters will occur next, but the DNN will have learned what sorts of clusters are *likely* to occur.) On the other hand, by the time the lead time ends, we will be in a medium-to-low-demand phase, so the order quantity Q_7 is somewhat small. The next order occurs when we are in cluster 3. The lead-time demand is likely to be small, so r_3 is small, but the demand will have increased by the time the order arrives, so Q_3 is large.

We also use a KNN-based approach as a machine-learning-based benchmark. For a given feature value x , we use SAA for approximating $n(r)$, i.e.,

$$n(r) = \frac{1}{k} \sum_{i \in \mathbb{N}_x} (d_i - r_i)^+, \quad (17)$$

where \mathbb{N}_x is the set of k -neighbors for point x . Then, to obtain (r, Q) for a given feature x , we modify the EIL algorithm by replacing the rule for updating Q in the EIL algorithm with

$$Q = \sqrt{\frac{2\lambda \left[K + \frac{1}{k} p \sum_{i \in \mathbb{N}_x} (d_i - r_i)^+ \right]}{h}}.$$

The rest of the algorithm remains the same.

5.2 Numerical Experiments

In order to test the effectiveness of our proposed DNN algorithm when it is applied to the (r, Q) optimization problem, we tested our algorithm as well as the KNN algorithm on a problem with $K = 20$, $\lambda = 1200$, $c_p = 10$, $c_h = 1$, and $L = \mu / \lambda$ where μ is the annual demand of the product. As a benchmark against which to compare both approaches, we used the iterative EIL algorithm by Hadley and Whitin (1963) to obtain the r and Q that minimize the approximate cost function (16). Since the algorithm needs the demand distribution, similar to the approach in Section 4.3, we fit a normal distribution to each cluster and use it to obtain (r, Q) for that corresponding cluster. We refer to this approach as the EIL algorithm.

The (r, Q) inventory policy is applied over an infinite time horizon, and to measure the performance of each approach, one needs to run a long-run analysis. To this end, in addition to reporting the expected cost from the approximate (steady-state) cost function (16), we run a discrete-time simulation to obtain a cost estimate that accounts for the changes to the system features over time. In the simulation, for a given item, in each time period, the (r, Q) values for each of the three approaches (DNN, KNN, and EIL) are obtained according to the current feature values, i.e., item-department, day, and month, an order is made (if necessary), the demand and arriving shipment are observed, and the inventory statistics and costs are updated accordingly. Then, time is incremented by one unit, which determines the new set of features, and accordingly the new (r, Q) values. For each item, we run this simulation for 10000 periods and report the sum of the holding, shortage, and fixed ordering costs. This simulation is executed for each of the five demand distributions. For each distribution, in each period of the simulation, the demand is randomly selected from the dataset that was generated and used in Section 4.3. In each period, once the feature values are observed and the corresponding cluster is determined, a demand is randomly selected from the cluster in the test dataset, one period of the simulation is executed, and then the demand observation is deleted from the test dataset. Pseudocode for the simulation is shown in Algorithm 1. In the

algorithm, $IL_{i,t}$ is the inventory level of item i in period t , and $AO_{i,t}$ is the arriving order, i.e., the number of units of item i that will arrive at time t .

Algorithm 1 (r, Q) Policy Simulator

```

1: procedure Simulate an ( $r, Q$ ) system
2: Initialize  $IL_{i,t}, AO_{i,t}$  for all  $i, t$ , set horizon  $T$  and cluster  $C$ 
3: for  $i \leq C$  do
4:   for  $t \leq T$  do
5:     Generate  $d_{i,t}$  randomly
6:      $IL_{i,t} = IL_{i,t} - d_{i,t}$ 
7:     Get  $(r_{i,t}, Q_{i,t})$  for the current feature values.
8:     if  $IP_{i,t} < r_{i,t}$  then
9:        $AO_{i,t+l_i} = AO_{i,t+l_i} + Q_{i,t}$ 
10:    end if
11:     $\text{cost} = \text{cost} + c_h IL_{i,t}^+ + c_p IL_{i,t}^- + K$ 
12:     $IL_{i,t+1} = IL_{i,t} + AO_{i,t}$ 
13:  end for
14: end for
15: end procedure

```

When testing the DNN algorithm on this problem, we performed the same level of hyper-parameter tuning that we did on the newsvendor problem. All of the neural networks use the `Relu` activation function, where $\text{Relu}(x) = x^+$. We used the Adam optimizer (Kingma and Ba 2014) to optimize the weights of the network with random learning rate, $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-8$, a batch size of 128, and exponential decay with rate 0.96.

In what follows, we demonstrate the results of the three algorithms on two datasets that we used when testing the newsvendor problem: the basket data

set, which is presented in Section 5.2.1, and the five randomly generated datasets, presented in Section 5.2.2.

5.2.1 Basket Dataset

We obtained (r, Q) values using all three algorithms, and used the expected cost function (16) to evaluate the policies. The solution found by the EIL algorithm incurs a cost of 1,650,214; KNN results in 1,392,643; and DNN has a cost of 1,322,568, 19.9% and 5.0% better (respectively) than EIL and KNN. At first this may seem surprising, since the EIL algorithm is an exact algorithm to optimize the cost function (16) (though of course (16) is itself an approximation of the exact cost function). However, recall that the basket dataset is noisy and contains few historical observations (between 1 and 9) per cluster, but the EIL algorithm assumes the demands are normally distributed. This assumption is inaccurate for the basket dataset. KNN, too, works best when there is a large number of neighbors for each sample, which is not true for this dataset. In contrast, DNN considers the feature values and optimizes the weights of the network, and in doing so is able to learn the (r, Q) values that obtains smaller cost than KNN and EIL algorithm. Note that the whole dataset included 13170 records with 1856 clusters (recall that each cluster is a unique combination of item, day, and month). The test dataset includes 689 clusters out of 1856 total clusters, and on average there are six records per cluster. So, there are not enough observations in the test dataset to run the simulation and evaluate each policy.

5.2.2 Randomly Generated Data

In order to further explore the performance of the three algorithms, we tested their performance on the randomly generated datasets in Section 4.3. Just as in the newsvendor problem, we assume we do not know the demand distribution and instead approximate a normal distribution in each cluster to obtain the solution using EIL. The results of all demand distributions are shown in Figure 11, in which the results of each algorithm under the five demand distributions are presented. In the figure, the series are labeled with the algorithm and demand distribution, e.g., KNN-uniform denotes the results

of the KNN algorithm with uniformly distributed demand. Figures 11(a) and 11(b) show the expected cost calculated by (16) and the simulated cost, respectively. In both figures, the costs of KNN, DNN, and EIL are normalized by dividing them by the corresponding cost of the EIL algorithm.

As shown in Figure 11(a), when the measure is the EIL cost function, DNN and KNN obtain costs that are quite close to that of the EIL algorithm; on average their costs are 1.6% and 3.8% smaller than EIL, respectively. Note that KNN benefits from the fact that there are at least 37 neighbors for each test record, which allows it to get a reasonable approximation for the demand. On the other hand, with the simulation the costs are closer to that of the EIL algorithm, with DNN and KNN costs that are 0.3% and 1.3% larger, on average, than EIL. When the data is generated from a normal distribution, KNN obtains smaller expected cost than EIL, although the simulation results show that EIL consistently finds smaller costs than DNN and KNN in this case. This result implies that when the data distribution is known, EIL is able to get the best solution among the three algorithms, though the DNN solution is close, with around a 2.3% gap, on average, for 1, 10, 100, and 200 clusters. For other distributions, there is no consistent pattern; all algorithms work about as well as each other: on average DNN, attains a 0.15% smaller cost than EIL, while KNN obtains a 0.69% larger cost than that of EIL.

In addition, the costs obtained by the simulation and the expected cost function (16) are different. Both DNN and KNN had better performance than the EIL algorithm under the expected cost function (16). The reason is that the DNN, KNN, and EIL algorithms want to minimize the cost function (16), not the actual (r, Q) cost function. In other words, they find the solution that minimizes the EIL cost function, which is an approximation of the actual (r, Q) cost function. So, the optimal solution of cost function (16) is not necessarily optimal for the actual (r, Q) cost function. As a result, there are noticeable differences between Figures 11(a) and 11(b).

Let us more closely examine one instance, the normally distributed dataset, for which the EIL solution is optimal. When there is only one cluster, the

optimal solution from EIL is $(r, Q) = (70.24, 222.70)$, and KNN gets $(70.31, 219.09)$, whereas DNN obtains $(70.00, 222.53)$, which is quite close to EIL's solution. Similarly, when there are 10 clusters, the DNN (r, Q) are quite close to the optimal solutions, as shown in Table 5, though it is not case with KNN. As a result, the costs of the solutions obtained by the DNN and EIL algorithms are almost equal. Similar results also emerge from the instances with 100 and 200 clusters.

To summarize, if the true stationary distribution is available, our DNN method, the KNN method, and the classical EIL approach work almost equally well. However, EIL algorithm's performance deteriorates when the true demand distribution is not known, even if there is a relatively large amount of historical data. Under this condition, KNN obtains a smaller cost than the EIL algorithm. In contrast, DNN works well when the true demand distribution is unknown, even if the historical dataset is small and/or noisy. Thus, DNN is very powerful at finding solutions even for this problem with two decision variables; it works as well as EIL and KNN when the data are not noisy, and obtains smaller costs than EIL and KNN when the data are noisy.

6 Conclusion

In this paper, we consider the multi-feature newsvendor (MFNV) problem. If the probability distribution of the demands is known for every possible combination of the data features, there is an exact solution for this problem. However, approximating a probability distribution is not easy and produces errors; therefore, the solution of the newsvendor problem also may be not optimal. Moreover, other approaches from the literature do not work well when the historical data are scant and/or volatile.

To address this issue, we propose an algorithm based on deep learning to solve the MFNV. The algorithm does not require knowledge of the demand probability distribution and uses only historical data. Furthermore, it integrates parameter estimation and inventory optimization, rather than solving them separately. Extensive numerical experiments on real-world and simulated

data demonstrate the conditions under which our algorithm works well compared to the algorithms in the literature. The results suggest that when the volatility of the demand is high, which is common in real-world datasets, deep learning works very well. When the data can be represented by a well-defined probability distribution, in the presence of enough training data, a number of approaches, including DNN, have roughly equivalent performance.

Furthermore, we extend our DNN approach to the (r, Q) inventory optimization problem, to demonstrate that our approach is applicable in more general settings, especially those that cannot be solved by estimating a quantile. Our computational results show that the DNN approach works well when the historical data are noisy and/or sparse, and that it often outperforms the “exact” algorithm when the true demand distribution is unknown (since the exact algorithm must make an assumption about the distribution).

Motivated by the results of deep learning on both newsvendor and (r, Q) problems, we suggest that this idea can be extended to other supply chain problems. For example, since general multi-echelon inventory optimization problems are very difficult, deep learning may be a good candidate for solving these problems. Another direction for future work could be applying other machine learning algorithms to exploit the available data in the newsvendor problem.

7 Acknowledgment

This research was supported in part by NSF grants NSF:CCF:1618717, NSF:CMMI:1663256 NSF:CCF:1740796, and XSEDE:DDM180004. This support is gratefully acknowledged.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A system for large-scale machine learning. In *Proceedings*

of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA, 2016.

Özden Gür Ali and Kübra Yaman. Selecting rows and columns for training support vector regression models with large retail datasets. *European Journal of Operational Research*, 226 (3):471–480, 2013.

Gah-Yi Ban and Cynthia Rudin. The big data newsvendor: practical insights from machine learning analysis. *Forthcoming in Operations Research*, 2018.

Gah-Yi Ban, Jérémie Gallien, and Adam Mersereau. Dynamic procurement of new products with covariate information: The residual tree method. Technical report, London Business School, 2017.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

Dimitris Bertsimas and Nathan Kallus. From predictive to prescriptive analytics. *arXiv preprint arXiv:1402.5481*, 2014.

Dimitris Bertsimas and Aurélie Thiele. A data-driven approach to newsvendor problems. Technical report, Massachusetts Institute of Technology, Cambridge, MA, 2005.

Omar Besbes and Alp Muharremoglu. On implications of demand censoring in the newsvendor problem. *Management Science*, 59(6):1407–1424, 2013.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time Series Analysis: Forecasting and Control*. John Wiley & Sons, 2015.

Apostolos N Burnetas and Craig E Smith. Adaptive ordering and pricing for perishable products. *Operations Research*, 48(3):436–443, 2000.

Alex J Cannon. Quantile regression neural networks: Implementation in R and application to precipitation downscaling. *Computers & Geosciences*, 37(9):1277–1284, 2011.

Hoi-Ming Chi, Okan K Ersoy, Herbert Moskowitz, and Jim Ward. Modeling and optimizing a vendor managed replenishment system using machine learning and genetic algorithms. *European Journal of Operational Research*, 180(1):174–193, 2007.

Tsan-Ming Choi, editor. *Handbook of Newsvendor Problems*. Springer, New York, 2012.

Li Deng, Jinyu Li, Jui-Ting Huang, Kaisheng Yao, Dong Yu, Frank Seide, Mike Seltzer, Geoffrey Zweig, Xiaodong He, Julia Williams, et al. Recent advances in deep learning for speech research at Microsoft. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8604–8608. IEEE, 2013.

Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, pages 3460–3468, 2015.

F. Edgeworth. The mathematical theory of banking. *Journal of Royal Statistical Society*, 51: 113–127, 1888.

Tuğba Efendigil, Semih Önüt, and Cengiz Kahraman. A decision support system for demand forecasting with artificial neural networks and neuro-fuzzy models: A comparative analysis. *Expert Systems with Applications*, 36(3):6697–6707, 2009.

Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, and Kevin Leyton-Brown. Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NIPS Workshop on Bayesian Optimization in Theory and Practice*, volume 10, 2013.

Mohammed E El-Telbany. What quantile regression neural networks tell us about prediction of drug activities. In *Computer Engineering Conference (ICENCO), 2014 10th International*, pages 76–80. IEEE, 2014.

Jacob R Gardner, Matt J Kusner, Zhixiang Eddie Xu, Kilian Q Weinberger, and John P Cunningham. Bayesian optimization with inequality constraints. In *ICML*, pages 937–945, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.

G. Hadley and T. M. Whitin. *Analysis of Inventory Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1963.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Martin Längkvist, Lars Karlsson, and Amy Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42:11 – 24, 2014.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560*, 2016.

Zachary C Lipton. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490*, 2016.

Pentaho. Foodmart’s database tables. <http://pentaho.dlpage.phi-integration.com/mondrian/mysql-foodmart-database>, 2008. Accessed: 2015-09-30.

Evan L. Porteus. *Foundations of Stochastic Inventory Theory*. Stanford University Press, Stanford, CA, 2002.

Evan L. Porteus. The newsvendor problem. In D. Chhajed and T. J. Lowe, editors, *Building Intuition: Insights From Basic Operations Management Models and Principles*, chapter 7, pages 115–134. Springer, 2008.

X. Qiu, L. Zhang, Y. Ren, P. N. Suganthan, and G. Amaratunga. Ensemble deep learning for regression and time series forecasting. In *Computational Intelligence in Ensemble Learning (CIEL), 2014 IEEE Symposium on*, pages 1–6, Dec 2014a.

Xueheng Qiu, Le Zhang, Ye Ren, Ponnuthurai N Suganthan, and Gehan Amaratunga. Ensemble deep learning for regression and time series forecasting. In *CIEL*, pages 21–26, 2014b.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive Modeling*, 5(3):1, 1988.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61: 85–117, 2015.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 802–810. Curran Associates, Inc., 2015.

Robert H Shumway and David S Stoffer. *Time Series Analysis and Its Applications: With R Examples*. Springer Science & Business Media, 2010.

Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.

- Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.
- Lawrence V Snyder and Zuo-Jun Max Shen. *Fundamentals of Supply Chain Theory*. John Wiley & Sons, 2nd edition, 2019. Forthcoming.
- James W Taylor. A quantile regression neural network approach to estimating the conditional density of multiperiod returns. *Journal of Forecasting*, 19(4):299–311, 2000.
- Gerald Tesauro, Yu He, and Subutai Ahmad. Asymptotic convergence of backpropagation. *Neural Computation*, 1(3):382–391, 1989.
- Nazli Turken, Yinliang Tan, Asoo J Vakharia, Lan Wang, Ruoxuan Wang, and Arda Yenipazarli. The multi-product newsvendor problem: Review, extensions, and directions for future research. In *Handbook of Newsvendor Problems*, pages 3–39. Springer, 2012.
- Armando Vieira. Predicting online user behaviour using deep learning algorithms. *Computing Research Repository - arXiv.org*, <http://arxiv.org/abs/1511.06247>, 2015.
- Qifa Xu, Xi Liu, Cuixia Jiang, and Keming Yu. Quantile autoregression neural network model with applications to evaluating value at risk. *Applied Soft Computing*, 49:1–12, 2016.
- Xiaodan Yu, Zhiquan Qi, and Yuanmeng Zhao. Support vector regression for newspaper/magazine sales forecasting. *Procedia Computer Science*, 17:1055–1062, 2013.
- Yu-Sheng Zheng. On properties of stochastic inventory systems. *Management Science*, 38(1): 87–103, 1992.
- Paul H. Zipkin. *Foundations of Inventory Management*. Irwin/McGraw-Hill, New York, 2000.

Notes

¹We plan to release the open-source code once this paper is accepted for publication; at that time, a URL will be provided here.)

A Proofs of Propositions 1 and 2

These proofs are based on the general idea of the back-propagation algorithm and the way it builds the gradients of the network. For further details, see [LeCun et al. \(2015\)](#).

Proof of Proposition 1. To determine the gradient with respect to the weights of the network, we first consider the last layer, L , which in our network contains only one node. Note that in layer L , $y_i^q = a_1^L$. So, we first obtain the gradient with respect to w_{j1} , which connects node j in layer $L - 1$ to the single node in layer L , and then recursively calculate the gradient with respect to other nodes in other layers.

First, consider the case of excess inventory ($d_i^q \leq y_i^q$). Recall from (12) that

$$\delta_j^l = \frac{\partial E_i^q}{\partial a_j^l} (g_j^l)'(z_j^l) \quad . \text{ Then } \delta_1^L = c_h (g_1^L)'(z_1^L) , \text{ since } E_i^q = c_h (a_1^L - d_i^q) . \text{ Then:}$$

$$\begin{aligned} \frac{\partial E_i^q}{\partial w_{j1}} &= c_h \frac{\partial (y_i^q - d_i^q)}{\partial w_{j1}} \\ &= c_h \frac{\partial a_1^L}{\partial w_{j1}} \quad (\text{since } d_i^q \text{ is independent of } w_{j1}) \\ &= c_h \frac{\partial g_1^L(z_1^L)}{\partial w_{j1}} \\ &= c_h \frac{\partial g_1^L(z_1^L)}{\partial z_1^L} \frac{\partial z_1^L}{\partial w_{j1}} \quad (\text{by the chain rule}) \\ &= c_h (g_1^L)'(z_1^L) a_j^{L-1} \quad (\text{by (11)}) \\ &= \delta_1^L(h) a_j^{L-1} \quad (\text{by (13)}). \end{aligned} \tag{18}$$

Now, consider an arbitrary layer l and the weight w_{jk} that connects node j in

layer l and node k in layer $l+1$. Our goal is to derive $\delta_j^l = \frac{\partial E_i^q}{\partial z_j^l}$, from which

one can easily obtain $\frac{\partial E_i^q}{\partial w_{jk}}$, since

$$\frac{\partial E_i^q}{\partial w_{jk}} = \frac{\partial E_i^q}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}} = \delta_j^l a_j^l \quad (19)$$

using similar logic as in (18). To do so, we establish the relationship between δ_j^l and δ_k^{l+1} .

$$\begin{aligned} \delta_j^l &= \frac{\partial E_i^q}{\partial z_j^l} \\ &= \sum_k \frac{\partial E_i^q}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} \quad (20) \\ &= \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l} \end{aligned}$$

Also, from (6), we have

$$z_k^{l+1} = \sum_j w_{jk} a_j^l = \sum_j w_{jk} g_j^l(z_j^l)$$

Therefore,

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{jk} (g_j^l)'(z_j^l). \quad (21)$$

Plugging (21) into (20), results in (22).

$$\delta_j^l = \sum_k w_{jk} \delta_k^{l+1} (g_j^l)'(z_j^l). \quad (22)$$

We have now calculated δ_j^l for all $l=1, \dots, L$ and $j=1, \dots, nn_l$. Then, substituting (22) in (19), the gradient with respect to any weight of the network is:

$$\frac{\partial E_i^q}{\partial w_{jk}} = a_j^l \sum_k w_{jk} \delta_k^{l+1} g_j^{\prime l}(z_j^l). \quad (23)$$

Similarly, for the shortage case in layer L , we have:

$$\begin{aligned} \frac{\partial E_i^q}{\partial w_{j1}} &= -c_p \frac{\partial(d_i^q - y_i^q)}{\partial w_{j1}} \\ &= c_p \frac{\partial(a_1^L)}{\partial w_{j1}} \\ &= c_p \frac{\partial(g_1^L(z_1^L))}{\partial w_{j1}} \quad (24) \\ &= c_p \frac{\partial(g_1^L(z_1^L))}{\partial z_1^L} \frac{\partial(z_1^L)}{\partial w_{j1}} \\ &= c_p a_j^{L-1} (g_1^L)'(z_1^L) \\ &= \delta_1^L(p) a_j^{L-1}. \end{aligned}$$

Using the chain rule and following same procedure as in the case of excess inventory, the gradient with respect to any weight of the network can be obtained. Summing up (18), (23) and (24), the gradient with respect to the w_{jk} is:

$$\frac{\partial E_i^q}{\partial w_{jk}} = \begin{cases} a_j^l \delta_j^l(p) & \text{if } y_i^q < d_i^q, \\ a_j^l \delta_j^l(h) & \text{if } d_i^q \leq y_i^q. \end{cases}$$

□

Proof of Proposition 2. Consider the proposed revised Euclidean loss function defined in (10). Using similar logic as in the proof of Proposition 1, we get that the gradient of the loss function at the single node in layer L is

$$\begin{aligned} \frac{\partial E_i^q}{\partial w_{j1}} &= c_h (y_i^q - d_i^q) \frac{\partial(y_i^q - d_i^q)}{\partial w_{j1}} \quad (25) \\ &= (y_i^q - d_i^q) a_j^{L-1} \delta_1^L(h). \end{aligned}$$

in the case of excess inventory and

$$\begin{aligned}\frac{\partial E_i^q}{\partial w_{j1}} &= -c_p(d_i^q - y_i^q) \frac{\partial(d_i^q - y_i^q)}{\partial w_{j1}} \\ &= (d_i^q - y_i^q) a_j^{L-1} \delta_1^L(p).\end{aligned}\quad (26)$$

in the shortage case. Again following the same logic as in the proof of Proposition 1, the gradient with respect to any weight of the network can be obtained:

$$\frac{\partial E_i^q}{\partial w_{jk}} = \begin{cases} (d_i^q - y_i^q) a_j^L \delta_1^L(p) & \text{if } y_i^q < d_i^q \\ (y_i^q - d_i^q) a_j^L \delta_1^L(h) & \text{if } d_i^q \leq y_i^q. \end{cases}$$

□

B Grid Search for Basket Dataset

In this appendix, we discuss our method for performing a more thorough tuning of the network for DNN- ℓ_2 , as discussed in Section 4.2. We used a large, two-layer network with 350 and 100 nodes in the first and second layer, respectively. In order to find the best set of parameters for this model, a grid search is used. We considered three parameters, lr , λ , and γ , λ is the regularization coefficient, and lr and γ are parameters used to set the learning rate. In particular, we set lr_t , the learning rate used in iteration t , using the following formula:

$$lr_t = lr \times (1 + \gamma \times t)^{-0.75}.$$

We considered parameter values from the following sets:

$$\begin{aligned}\gamma &\in \{0.01, 0.005, 0.001, 0.0001, 0.00005, 0.000005\} \\ \lambda &\in \{0.01, 0.005, 0.001, 0.0005, 0.0001, 0.00005\} \\ lr &\in \{0.001, 0.005, 0.0005, 0.0001, 0.00005, 0.00003, 0.00001, 0.000009, 0.000008, 0.000005\},\end{aligned}$$

The best set of parameters among these 360 sets were

$\gamma = 0.00005$, $\lambda = 0.00005$, and $lr = 0.000009$. These parameters were used to test integer values of $c_p / c_h \in \{3, \dots, 9\}$ in Figure 3, for the series labeled DNN- ℓ_2 -T.

C A Tuning-Free Neural Network to Solve the Newsvendor Problem

To tune the hyper-parameters of the DNN in Section 4, we used an extension of the random search algorithm ([Bergstra and Bengio 2012](#)) called HyperBand ([Li et al. 2016](#))—in particular, to determine the network structure, learning rate, and regularization coefficient. However, a user of our model might not have the time, resources, or expertise to follow a similar procedure. Even cheaper procedures like Bayesian optimization ([Snoek et al. 2012](#), [Gardner et al. 2014](#)) are still too time consuming and too complex to implement. To address this issue, in this section we propose a computationally cheap approach to set up a network structure without extensive tuning. Our approach provides quite good results on a wide range of problem parameters.

The network structure should have a direct relation with the number of training samples n , the number of features p , and the range r_i that feature f_i , $i = 1, \dots, p$, can take values from. For example, a feature f_i which represents the day of week takes values between 1 and 7, and the one-hot-encoded version is a categorical feature with 7 categories; so, $r_i = 7$. For a continuous feature like the sales quantity, r_i may be an interval such as $[0, \infty]$. These characteristics—the number of features and the range of values for each feature—affect both the number of layers in the network and the numbers of nodes in each layer. For instance, if the number of features is small and the features take on only a few values, a trained DNN returns a solution that minimizes the average loss value. In this case a small network can provide quite good results. On the other hand, when the number of features is relatively large and each feature can take values from a large range or set, the DNN must be able to distinguish among a large number of cases. In this event, the DNN network must be relatively large.

Now, consider the newsvendor problem with p features. In the datasets that we considered, the features are quite simple, e.g., receipt date and item category. However, we wish to propose a general structure for prospective users of our model, so we assume one may use more complex features, either categorical or continuous. (However, we assume the input cannot be an

image, so we do not need a convolutional Goodfellow et al. (2016) network.) Thus, we propose a three-layer network in which the number of nodes in the first, second, and third hidden layers equal aq , bq , and cq , respectively, where a , b , and c are constants (by default we use $a = 1.5$, $b = 1$, and $c = 0.5$), and where q is defined as follows. Let q_v be the number of continuous features, let $P_c \subseteq \{1, \dots, p\}$ be the set of categorical features, and let

$$q_u = \min \left\{ \sum_{i \in P_c} r_i, \prod_{i \in P_c} r_i \right\}.$$

In words, q_u is the smallest number that can represent all combinations of categories. Let $q = q_u + q_v$. Finally, the number of input nodes also equals q , and the output layer includes a single node.

Using this approach, if the number of features is small, the number of DNN weights to optimize is small, and if the number of features is large, the number of weights is large. Using the default values of a , b , and c , the proposed

network has $m = \frac{1}{2}q(7q+1)$ weights, which should be smaller than the number of training records. If $m > n$, there is a chance of over-fitting, and if $m \gg n$, the DNN over-fits the training data with high probability, in which case the number of DNN variables must be reduced. In this case one should select smaller values of the coefficients a , b , and c to reduce the number of nodes in each layer. Finally, using the default coefficient values, the resulting network has size $[q, 1.5q, q, 0.5q, 1]$, so the number of nodes in the first hidden layer is larger than the number of features, and with a high probability the DNN is able to capture the information of the features and transfer them through the network. Setting the number of nodes in the first hidden layer smaller than that in the input layer may result in losing some input information.

We continue training until we meet one of the following criteria:

- the point-wise improvement in loss function value is less than 0.01%,
or
- the number of passes over the training data reaches MaxEpoch .

(We set $\text{MaxEpoch}=100$.)

Of course, we cannot guarantee that this approach will produce an optimal network structure, but it eliminates the work of determining the structure, and our experiments suggest that it performs well. We also note that one still must follow an approach to determine a suitable learning rate and regularization parameter (see [Snoek et al. \(2012\)](#), [Eggenberger et al. \(2013\)](#), [Domhan et al. \(2015\)](#), [Bergstra and Bengio \(2012\)](#)).

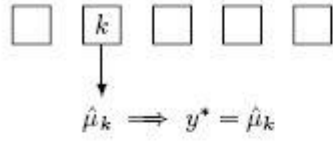
In order to see how well the fixed-size network works, we ran the same experiments as in Section 4.3. In these tests, we fixed the network structure to $[q, 1.5q, q, 0.5q, 1]$ with learning rate $= 0.001$ and $\lambda = 0.005$ for all demand distributions. In all cases except normally distributed demand, the network obtained near-optimal costs after at most 10 epochs (which, on average, took 10 minutes to train), when improvement stopped. For normally distributed demands, the algorithm ran for at least 50 epochs to get a converged network. Table 6 shows the results of the test datasets for all demand distributions, in which we provide the gap between the results of the fixed network and the results from the HyperBand algorithm. As provided in the table, when we train for 100 epochs, the fixed network obtains costs that are very close to those obtained using the HyperBand algorithm. For 1, 10, 100, and 200 clusters, it obtains average gaps of 0.67%, 1.9%, 43.5%, and 65.3% compared to the results of networks obtained by HyperBand algorithm.

In order to see the effect of training length, we ran all experiments for 300 epochs to see whether the solutions improve, which are provided in right side of Table 6. The average gaps decreased to 0.5%, 0.08%, 3.29%, and 62.6% for 1, 10, 100, and 200 clusters, respectively. Therefore, running the DNN for longer training periods can help to get smaller cost values.

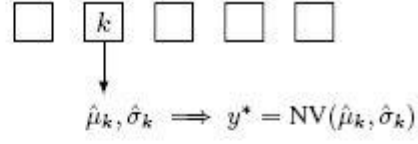
In sum, setting the network size using this approach is much cheaper than any extension of random search or Bayesian optimization, and it can provide near-optimal results for the newsvendor problem when there is a sufficiently large number of historical records. (In our experiment, this corresponds to

having fewer clusters.) When there is insufficient historical data available, additional tuning and/or training is required in order to obtain good results.

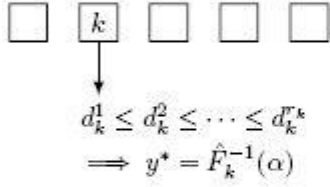
Accepted Manuscript



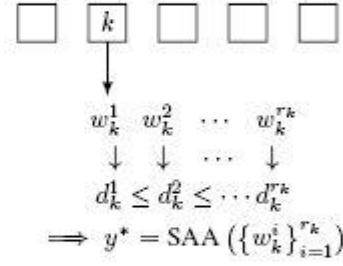
(a) Estimate-as-solution (EAS).



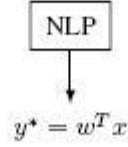
(b) Separated estimation and optimization (SEO).



(c) Empirical quantile (EQ).



(d) K-nearest neighbors (KNN) and random forest (RF).



(e) Linear machine learning (LML).

Fig. 1 Approaches for solving MFNV problem. Squares represent clusters.

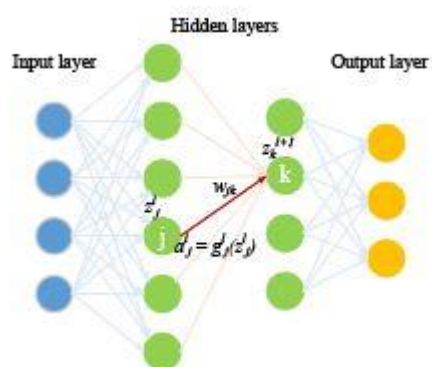


Fig. 2 A simple deep neural network.

Accepted Manuscript

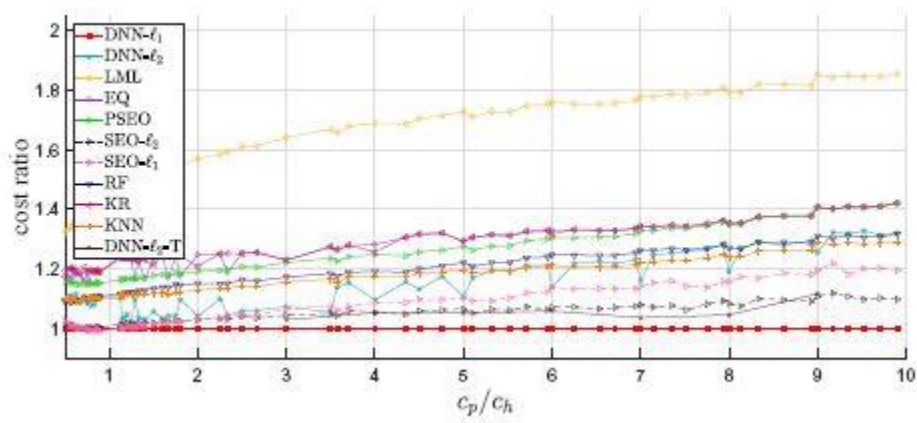


Fig. 3 Ratio of each algorithm's cost to DNN- ℓ_1 cost on a real-world dataset.

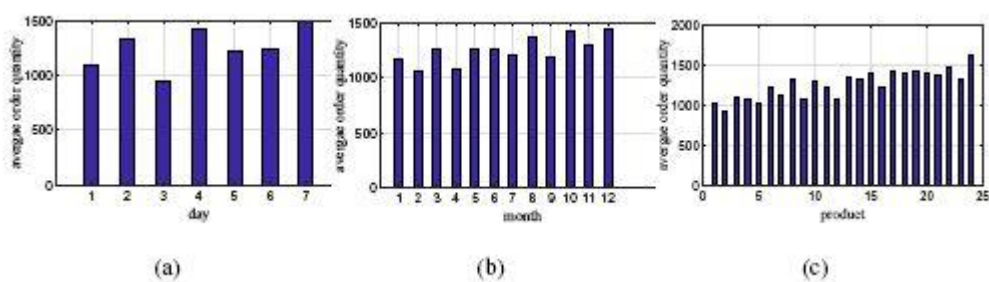


Fig. 4 The effect each feature on the order quantity for uniformly distributed data with 100 clusters.

Accepted Manuscript

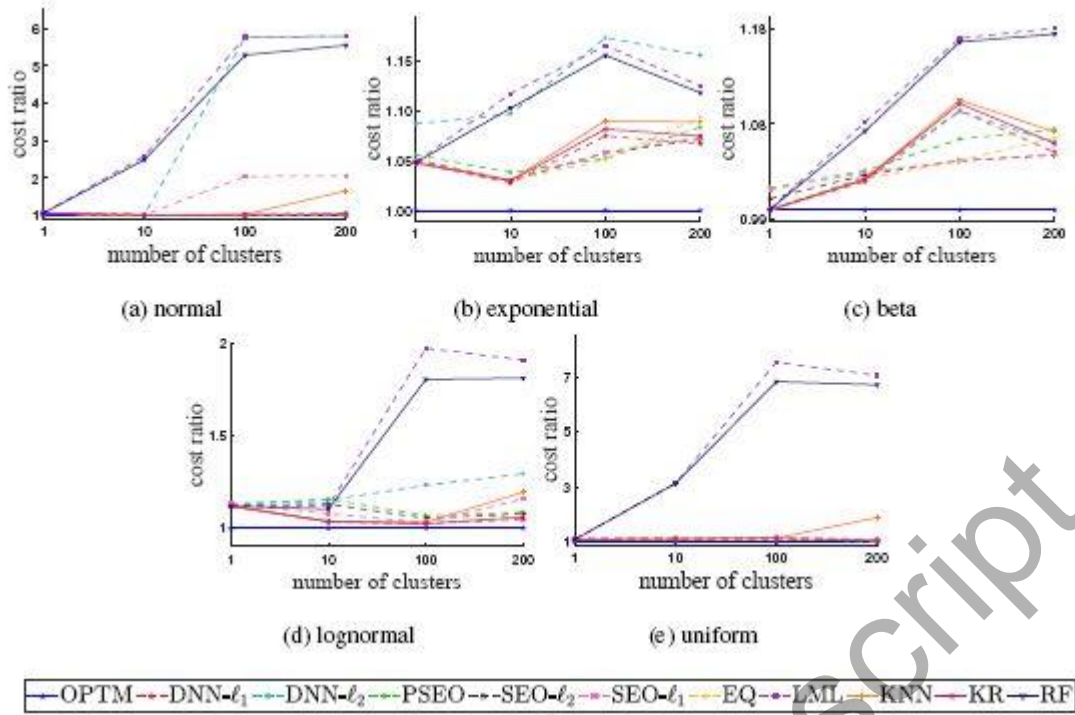


Fig. 5 Ratio of each algorithm's cost to optimal cost on randomly generated data from each distribution.

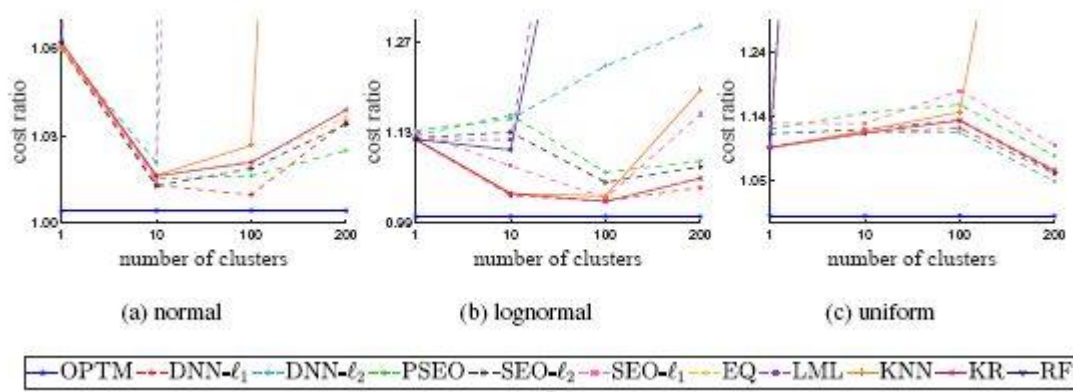


Fig. 6 Magnified results for normal, lognormal, and uniform distributions.

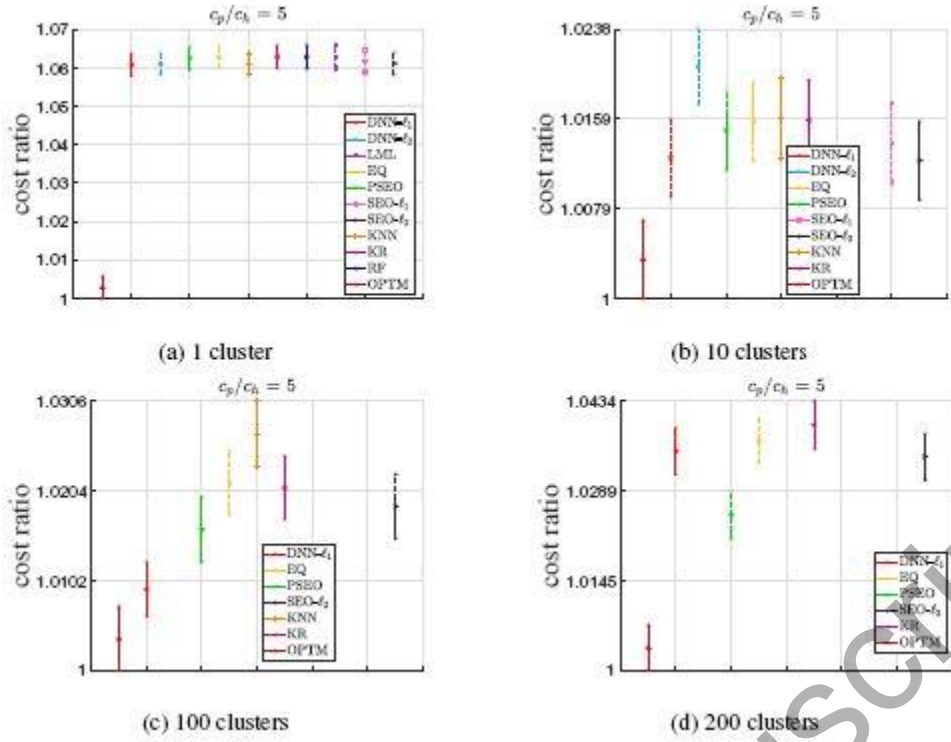


Fig. 7 Confidence intervals for each algorithm for normally distributed demands.

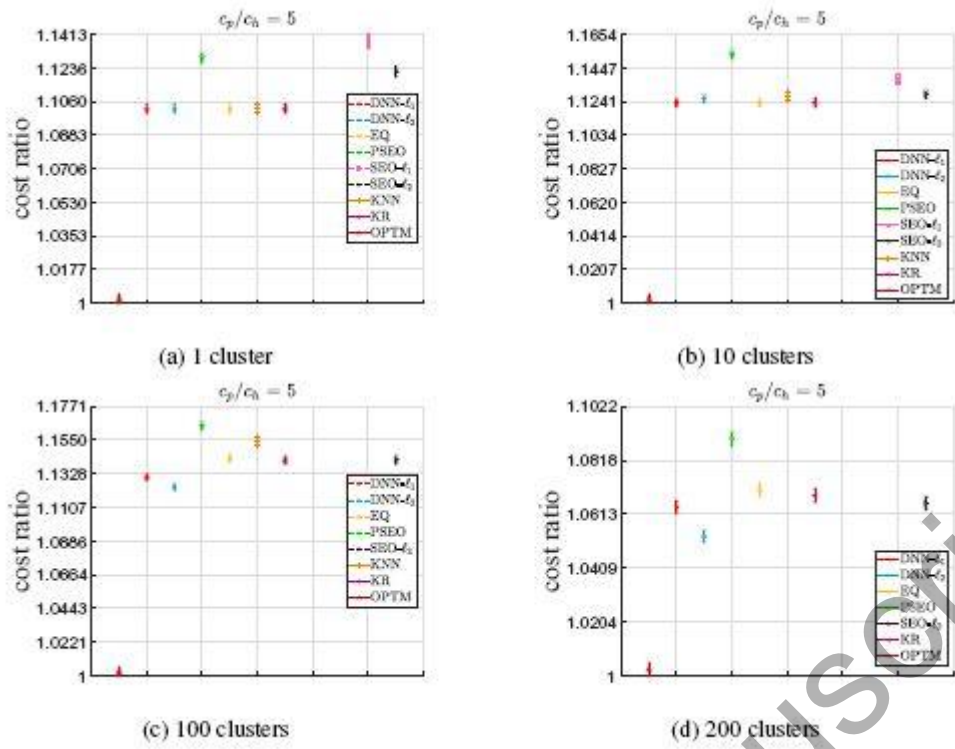


Fig. 8 Confidence intervals for each algorithm for uniformly distributed demands.

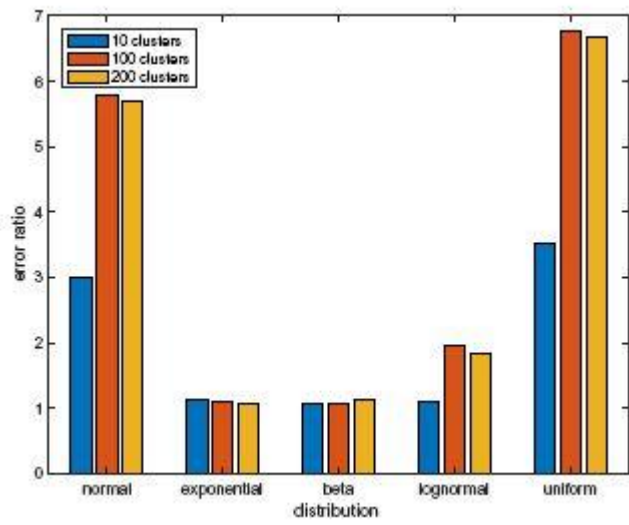


Fig. 9 Error ratio from ignoring clusters when solving MFNV.

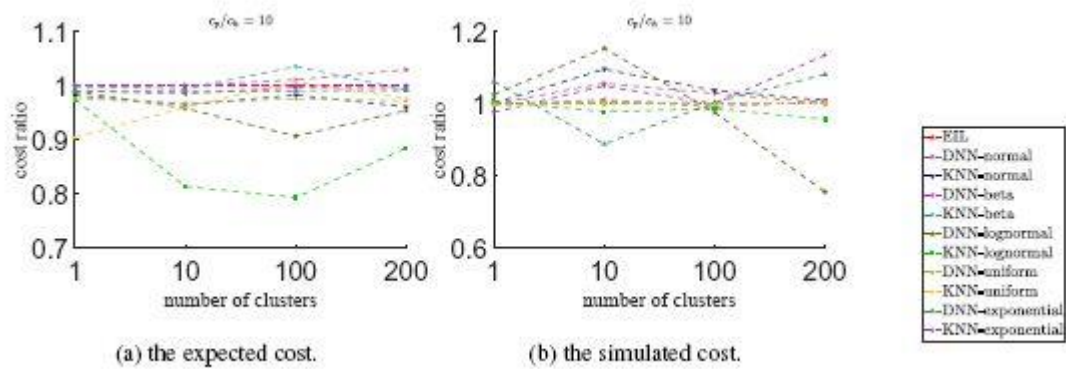


Fig. 11 The results for randomly generated datasets for the (r, Q) model.

Table 1 Demand of one item over three weeks.

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Week 1	1	2	3	4	3	2	1
Week 2	6	10	12	14	12	10	10
Week 3	3	6	8	9	8	6	5

Accepted Manuscript

Table 2 Order quantity proposed by each algorithm for each day and the corresponding cost. The bold costs indicate the best newsvendor cost for each instance.

		Day & Demand							
(c_p, c_h)	Algorithm	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Cost
	True demand	3	6	8	9	8	6	5	
(1,1)	DNN- ℓ_2	3.5	6.0	7.5	9.0	7.5	6.5	5.5	2.5
	DNN- ℓ_1	4.6	6.0	8.5	9.0	8.6	5.6	5.6	2.9
	EQ	1.0	2.0	3.0	4.0	3.0	2.0	1.0	29.0
	LML	1.3	2.2	3.1	4.0	4.9	5.8	6.7	20.3
	PSEO	3.5	6.0	7.5	9.0	7.5	6.5	5.5	2.5
	SEO- ℓ_1	3.2	6.0	5.3	6.4	6.4	5.8	5.0	7.3
	SEO- ℓ_2	3.6	6.2	7.9	9.1	7.8	6.7	5.3	2.0
	KR	4.0	6.0	6.0	6.0	6.0	4.0	4.0	11.0
	KNN	6.0	6.0	6.0	6.0	6.0	6.0	6.0	11.0
	RF	6.0	6.0	6.0	6.0	6.0	6.0	6.0	11.0
(2,1)	DNN- ℓ_2	5.8	7.3	8.9	9.7	8.9	8.1	7.0	10.7
	DNN- ℓ_1	6.0	6.0	7.9	9.3	9.3	6.4	6.1	5.9
	EQ	6.0	10.0	12.0	14.0	12.0	11.0	10.0	30.0
	LML	6.0	7.0	8.0	9.0	10.0	11.0	12.0	18.0
	PSEO	5.0	8.4	10.2	12.0	10.2	9.2	8.2	18.5
	SEO- ℓ_1	5.3	7.0	7.8	8.5	7.8	7.3	7.5	8.9
	SEO- ℓ_2	4.7	6.8	8.8	10.1	8.8	7.5	6.3	8.0
	KR	6.0	10.0	12.0	14.0	12.0	11.0	10.0	30.0
	KNN	10.0	6.0	10.0	10.0	10.0	10.0	10.0	25.0
	RF	6.0	10.0	10.0	10.0	11.0	11.0	11.0	24.0
(10,1)	DNN- ℓ_2	5.6	9.3	11.2	13.1	11.2	10.2	9.2	24.6

[illegible]

Table 3 Summary of hyper-parameter (HP) tuning process for each method. Times reported are approximate training times for a single problem instance.

				Approx. Avg.	Approx.
		# HP Values		Training Time	Total Training
	# HP	Tested	HP Values Tested	per HP (sec)	Time (sec)
SEO	–	–			10
EQ	–	–			10
LML	1	30	$2^h, h \in \{-20, \dots, 10\}$	40	1200
KR	1	7	$\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 0.05, 0.1, 0.25\}$	15	105
KNN	1	6	$\{5, 10, 15, 50, 100, 150\}$	5	30
RF	1	5	$\{10, 20, 50, 100, 150\}$	4 (per tree)	1320
DNN	4	100	(see Section 3)	600	44,050

Table 4 Demand distribution parameters for randomly generated data.

	Number of Clusters			
Distribution	1	10	100	200
Normal	$\mathcal{N}(50,10)$	$\mathcal{N}(50i,10i)$	$\mathcal{N}(50i,5i)$	$\mathcal{N}(50i,5i)$
Lognormal	$\ln \mathcal{N}(2,0.5)$	$\ln \mathcal{N}(1+0.1(i+1),$	$\ln \mathcal{N}(0.05(i+1),$	$\ln \mathcal{N}(0.02(i+1),$
		$0.5+0.1(i+1))$	$0.01(i+1))$	$0.005(i+1))$
Exponential	$\exp(10)$	$\exp(5+2(i+1))$	$\exp(5+0.2(i+1))$	$\exp(5+0.05(i+1))$
Beta	$20\mathcal{B}(1,1)$	$100\mathcal{B}(0.6(i+1),$	$100\mathcal{B}(0.1(i+1),$	$100\mathcal{B}(0.07(i+1),$
		$0.6(i+1))$	$0.1(i+1))$	$0.07(i+1))$
Uniform	$\mathcal{U}(1,21)$	$\mathcal{U}(5(i+1),$	$\mathcal{U}((i+1),$	$\mathcal{U}(0.5(i+1),$
		$15+5(i+1))$	$15+(i+1))$	$15+0.5(i+1))$

Table 5 EIL and DNN values of (r, Q) for the normally distributed dataset with 10 clusters.

Cluster	1	2	3	4	5	6	7	8	9	10
EIL r	70.4	141.5	212.1	277.7	350.4	416.8	485.7	555.9	626.9	697.9
KNN r	70.5	141.7	207.5	278.7	352.1	419.2	488.8	560.2	632.0	704.9
DNN r	71.0	138.0	210.2	277.8	354.0	409.0	479.0	551.0	639.0	703.0
EIL Q	222.7	226.6	230.2	233.4	238.1	241.9	245.2	250.2	253.1	259.5
KNN Q	219.1	219.1	330.4	219.1	219.1	219.1	219.1	219.1	219.1	219.1
DNN Q	218.2	226.4	226.2	228.3	233.3	259.6	247.4	246.1	249.1	251.0

Table 6 Results of 100 and 200 training epochs.

	100 epochs				300 epochs			
clusters	1	10	100	200	1	10	100	200
normal	0.000	0.004	2.006	3.083	0.000	0.004	0.005	3.083
lognormal	0.003	0.006	0.129	0.006	0.000	0.004	0.126	0.011
uniform	0.001	0.012	0.020	0.134	0.000	0.001	0.020	-0.004
beta	0.029	0.003	0.014	0.023	0.027	-0.006	0.007	0.021
exponential	0.000	0.071	0.008	0.019	0.000	0.001	0.006	0.018
average	0.0067	0.0192	0.4353	0.6531	0.0054	0.0008	0.0329	0.6260