Applying Deep Learning to the Newsvendor Problem

AFSHIN OROOJLOOY, LAWRENCE SNYDER, MARTIN TAKÁČ

Department of Industrial and Systems Engineering

Lehigh University, Bethlehem, PA, USA

COR@L Technical Report 17T-004

Applying Deep Learning to the Newsvendor Problem

Afshin Oroojlooy, Lawrence Snyder, Martin Takáč[1]

[1]Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, USA

Abstract

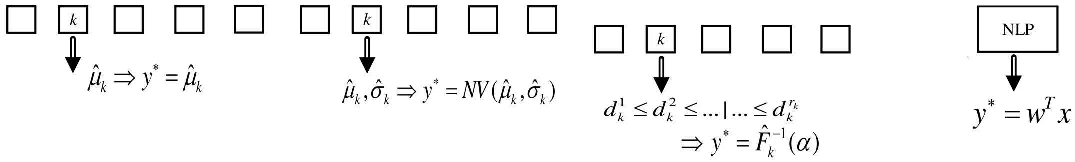
The newsvendor problem is one of the most basic and widely applied inventory models. There are numerous extensions of this problem. One important extension is the multi-item newsvendor problem, in which the demand of each item may be correlated with that of other items. If the joint probability distribution 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. 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.

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, it must dispose of unsold goods, incurring a *holding cost*. In addition, if it runs out of the goods in the middle of the period, it incurs a *shortage 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 (2011), among others, for textbook discussions.

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

$$\min_y C(y) = E_D\left[c_p(D-y)^+ + c_h(y-D)^+\right], \tag{1}$$



(a) Estimate-as-solution (EAS). (b) Separated estimation and optimization (SEO). (c) Empirical quantile (EQ). (d) Linear machine learning (LML).

Figure 1: Approaches for solving MFNV problem.

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

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

(see e.g., Snyder and Shen (2011)).

Extensions of the newsvendor problem are too numerous to cover 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 consider multiple items, with correlated demands that follow a multivariate probability distribution. 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, etc. The goal is to choose today's base-stock level, given the observation of today's features. We will call this problem the multi-feature newsvendor (MFNV) problem. An important extension of the newsvendor problem—the version we consider in this paper—is a combination of these two variants, i.e., a multi-item newsvendor problem with multi-dimensional features.

The remainder of this paper is structured as follows. A brief summary of the literature relevant to this problem is presented in Section 2. Section 3 presents the details of the proposed algorithm. Numerical experiments are provided in Section 4, and the conclusion and a discussion of future research complete the paper in Section 5.

2 Literature Review

2.1 Current State of the Art

Currently, there are four main approaches in the literature for solving the multi-item newsvendor problem with data features. The first category, which we will call the *estimate-as-solution* (EAS) approach, involves first forecasting the demand and then simply treating the point forecast as a deterministic demand value, i.e., setting the newsvendor solution equal to the forecast. (See Figure 1a.) The forecast may be performed in a number of ways, some of which we review in the next few paragraphs. This approach ignores the key insight from the newsvendor problem, namely, that we should not simply order up to the mean demand, but rather choose a level that strikes a balance between underage and overage costs using the distribution of the demand. Nevertheless, the approach is common in the literature. For example, Yu et al. (2013) propose a support vector machine (SVM) model to forecast newspaper demands. They considere the sales data of newspapers at different types of stores, along with 32 other factors such as education distribution in the area of the shop, age distribution, average income level, and gender distribution. Wu and Akbarov (2011) use a weighted support vector regression (SVR) model to forecast warranty claims; their model gives more priority to the most recent warranty claims. 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. Carbonneau et al. (2008) present a least squares SVM (LS-SVM) model to forecast a manufacturer's demand. They compare it with standard forecasting methods such as average, moving average, trend, and multiple linear regression, as well as neural network and recurrent neural network algorithms. According to their numerical experiments, the recurrent neural network and the LS-SVM algorithm have the best results. Ali and Yaman (2013) forecast grocery sales, with datasets containing millions of records, and for each record there are thousands of features. Since general SVM methods are not able to solve such a large problem, they proposed an algorithm to reduce the number of rows and columns of the datasets with a small loss of accuracy. Lu and Chang (2014) propose an iterative algorithm to predict sales. They use independent component analysis (ICA) to obtain hidden features of their datasets, k-mean clustering to cluster the sales data, and finally SVR to provide the prediction. Viaene et al. (2000) propose an LS-SVM classifier model with 25 features to predict whether a direct mail offer will result in a purchase. Since a huge dataset was available, an iterative approach based on the Hestenes–Stiefel conjugate gradient algorithm was proposed to solve the model.

Classical parametric approaches for forecasting include ARIMA, TRANSFER, and GARCH models (Box et al. 2015, Shumway and Stoffer 2010); these are also used for demand forecasting (see Cardoso and Gomide (2007), Shukla and Jharkharia (2011)). Similarly, Taylor (2000) use a normal distribution to forecast demand one or more time steps ahead; however, their model does not perform well when demands are correlated over time and when the demands are volatile. These and other limitations have motivated the use of DNN to obtain demand forecasts. For example, Efendigil et al. (2009) propose a DNN model to forecast demand based on recent sales, promotions, product quality, and so on. Vieira (2015) propose a deep learning algorithm to predict online activity patterns that result in an online purchase. Taylor (2000), Kourentzes and Crone, Cannon (2011), Xu et al. (2016) use DNN for quantile regression, with applications to exchange rate forecasting, for example. For reviews of the use of DNN for forecasting, see Ko et al. (2010), Kourentzes and Crone, Qiu et al. (2014b), Crone et al. (2011).

The common theme in all of the papers in the last two paragraphs is that they provide only a forecast of the demand, which must 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 Rudin and Vahn (2013) refer to as *separated estimation and optimization* (SEO), involves first estimating (forecasting) the demand 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 1b.) 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 empirical demand distributions 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. Rudin and Vahn (2013) prove that for some realistic settings, the SEO approach is provably suboptimal. We are not aware of this approach being proposed explicitly in the literature, but it is a natural idea to use in practice, and Rudin and Vahn (2013) analyze it as a straw-man against which to compare their SVM approach. A broad list of research that uses this approach is given by Turken et al. (2012).

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 $d_1 \leq d_2 \leq \cdots \leq d_n$ and then estimating the αth 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 $j = \lceil n \frac{c_p}{c_p + c_h} \rceil$. This quantile is then used as the base-stock level, in light of (2). Since they approximate the αth quantile, we refer to their method as the *empirical quantile* (EQ) method. (See Figure 1c.) 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 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 Rudin and Vahn (2013). 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.

The approach that is closest to our proposed approach was introduced by Rudin and Vahn (2013); we refer to it as the *linear machine learning* (LML) method. They postulate that the optimal base-stock level is related to the demand features via a linear function; that is, that

$$y^* = w^T x, \tag{3}$$

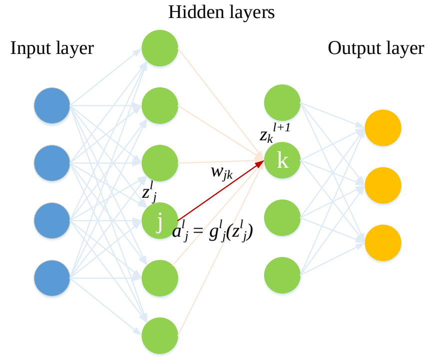
where w is a vector of (unknown) weights.

They estimate these weights by solving the following nonlinear optimization problem, essentially fitting the solution using the newsvendor cost:

$$\begin{aligned} &\min_w \tfrac{1}{n} \textstyle\sum_{i=1}^{n} \left[c_p (d_i - w^T x)^+ + c_h (w^T x - d_i)^+ \right] + \lambda ||w||_k^2 \\ &(d_i - w^T x)^+ \geq d_i - w_1 - \sum_{j=2}^{p} w_i x_i^j; \;\; \forall i = 1, \ldots, n \\ &(w^T x - d_i)^+ \geq w_1 + \sum_{j=2}^{p} w_i x_i^j - d_i; \;\; \forall i = 1, \ldots, n \end{aligned} \tag{4}$$

where n is the number of observations, p is the number of features, and $\lambda ||w||_k^2$ is a regularization term. The LML method avoids having to cluster the data, as well as having to specify the form of the demand distribution. However, this model does not work for a large number of instances, 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.

Figure 2: A simple Neural Network



2.2 Deep Learning

In this paper, we develop a new approach to solve the multi-product 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. We provide only a brief overview of deep learning here; for comprehensive reviews of the algorithm and its applications, see Schmidhuber (2015), LeCun et al. (2015), Deng et al. (2013), Qiu et al. (2014a), Shi et al. (2015), and Längkvist 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. The goal is to determine the weights 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:

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

where w is the matrix of the weights, x_i is the vector of the inputs from the ith instance, 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.

In each node j–out of N– of a layer l–out of L–, the input value

$$z_j^l = \sum_{i=1}^{N} a_i^{l-1} w_{i,j}$$

is calculated and the value of the function $g_j^l(z_j^l)$ provides 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 functions. The most commonly used activation functions are sigmoid and tanh. The activation function value of each node is the input for the next layer, and finally, the activation function values of the nodes in the last layer determine the output values of the network.

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), and SGD with momentum. 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 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.

2.3 Our Contribution

To adapt the deep learning algorithm for the newsvendor problem, 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.*

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 multi-product newsvendor problem with data features. Assume there are n historical demand observations for m items. Also, for each demand observation, the values of p features are known. That is, the data can be represented as

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

where $x_i^j \in \mathbb{R}^p$ and $d_i^j \in \mathbb{R}$ for $i = 1, \dots, n$ and $j = 1, \dots, m$. The problem is formulated mathematically in (5), resulting in the order quantities $y_i^1, \dots, y_i^m$:

$$E = \min_{y_i^1, \dots, y_i^m} \frac{1}{n} \left[\sum_{j=1}^{m} c_h (y_i^j - d_i^j)^+ + c_p (d_i^j - y_i^j)^+ \right], \;\; \forall i = 1, \dots, n, \tag{5}$$

where E is the loss value. 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 (5). 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 (Rudin and Vahn (2013)). To address this issue, we propose two loss functions, the newsvendor cost function and a revised Euclidean loss function, 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 the loss function given by (5) does not always lead to the best solution when compared with squared cost function, so we also test the folowing revised Euclidean loss function:

$$E = \min_{y_1,\cdots,y_n} \frac{1}{n}\left[\sum_{i=1}^{n}\left[c_p(d_i-y_i)^+ + c_h(y_i-d_i)^+\right]^2\right], \tag{6}$$

which penalizes the order quantities that are far from d_i much more than those that are close. (For ease of notation, we exclude the index j from (6) and hereafter.) Since at least one of the two terms in this function must be zero, we have

$$\begin{aligned} E &= \frac{1}{n}\sum_{i}^{n} E_i\ , \\ E_i &= \begin{cases} \frac{1}{2}||c_p(d_i-y_i)||_2^2\ , & \text{if } \ y_i < d_i, \\ \frac{1}{2}||c_h(y_i-d_i)||_2^2\ , & \text{if } \ d_i \le y_i. \end{cases} \end{aligned} \tag{7}$$

Similarly, for cost function (5) the loss function can be written as:

$$\begin{aligned} E &= \frac{1}{n}\sum_{i}^{n} E_i \\ E_i &= \begin{cases} |c_p(d_i-y_i)|\ , & \text{if } \ y_i < d_i, \\ |c_h(y_i-d_i)|\ , & \text{if } \ d_i \le y_i. \end{cases} \end{aligned} \tag{8}$$

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, and $a_j = g_j(z_j) = \frac{\partial(z_k)}{\partial w_{jk}}$ is the activation function value of node j. Proofs of both propositions are provided in Appendix C.

Proposition 1.The gradient with respect to the weights of the network for loss function (7) is:

$$\frac{\partial E_i}{\partial w_{jk}} = \begin{cases} c_p(d_i-y_i)a_j g_k'(z_k) & y_i < d_i \\ c_h(y_i-d_i)a_j g_k'(z_k) & d_i \le y_i. \end{cases} \tag{9}$$

Proposition 2. The gradient with respect to the weights of the network for loss function (8) is:

$$\frac{\partial E_i}{\partial w_{jk}} = \begin{cases} c_p a_j g_k'(z_k) & y_i < d_i, \\ c_h a_j g_k'(z_k) & d_i \le y_i. \end{cases} \tag{10}$$

Under the proposed loss functions (7) and (8), our deep learning algorithm uses gradient (9) and sub-gradient (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 (7) and the quadratic loss function (8), which we call DNN-ℓ_2 and DNN-ℓ_1, respectively.

In order to obtain a good structure for the DNN network, we generate 80 fully connected networks with random structures. In each, the number of hidden layers is randomly selected between two and three (with equal probability). Let nn_k denote the number of nodes in layer k; then nn_1 is equal to the number of features. The number of nodes in the hidden layers are selected randomly based on the number of nodes in the previous layer. For networks with two hidden layers, we choose $nn_2 \in [0.5 * nn_1, 3 * nn_1]$, $nn_3 \in [0.5 * nn_2, nn_2]$, and $nn_4 = 1$. Similarly, for networks with three hidden layers, $nn_2 \in [0.5*nn_1, 3*nn_1]$, $nn_3 \in [0.5*nn_2, 2*nn_2]$, $nn_4 \in [0.5*nn_3, nn_3]$, and $nn_5 = 1$. 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, an algorithm based on the HyperBand algorithm (Li et al. 2016) is run, in which each of the 80 networks are trained for one epoch (which is a full pass over the training dataset), the results on the test set are obtained, and then the worst 10% of the networks are removed. We then run another epoch on the remaining networks and remove the worst 10%. This procedure iteratively repeats to obtain the final network.

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. Also, the LML model by Rudin and Vahn (2013), and the SEO approach by calculating 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. Since SEO always provides a better solution compared to EAS, its result is not provided. In order to compare their results, the order quantities were obtained with each algorithm and the corresponding cost function

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

was calculated. We implemented the loss function for the deep learning algorithm in TensorFlow 0.9.0 (Abadi et al. (2015)) using Python. All of the deep learning experiments were done with TensorFlow. Note that the deep learning and LML 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, the cost coefficients were scaled.In addition, we translated the categorical data features to their corresponding binary representations. These two implementation details improve the accuracy of the learning algorithms. All computations were done on a 1.8 GHz machine with 32 GB of memory.

In what follows, we demonstrate the results of the four 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 four 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 four approaches on several randomly generated datasets.

Table 1: Demand of one item through 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

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 four 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. The table first lists the actual demand for each day, repeated from Table 4.1 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 αth 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 $c_p/c_h \leq 1$, 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.

Now consider the results of the SEO algorithm. For the case in which $c_h = c_p$ (which is not particularly realistic), SEO's output is approximately equal to the mean demand, which happens to be close to the week-3 demand values. This gives SEO a cost of 2.5, which ties DNN-ℓ_2 for first 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 Rudin and Vahn (2013), because LML uses a linear kernel, while DNN uses both a linear and non-linear kernel. Also, there are only two featuresin 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.

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
	SEO	3.5	6.0	7.5	9.0	7.5	6.5	5.5	**2.5**
(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
	SEO	5.0	8.4	10.2	12.0	10.2	9.2	8.2	18.5
(10,1)	DNN-ℓ_2	5.6	9.3	11.2	13.1	11.2	10.2	9.2	24.6
	DNN-ℓ_1	6.9	10.2	10.2	10.2	10.2	10.2	10.2	**22.8**
	EQ	6.0	10.0	12.0	14.0	12.0	11.0	10.0	30.0
	LML	8.0	10.0	12.0	14.0	16.0	18.0	20.0	53.0
	SEO	8.2	13.6	16.0	18.4	16.0	15.0	14.0	56.2
(20,1)	DNN-ℓ_2	5.8	9.6	11.6	13.5	11.6	10.6	9.6	**27.2**
	DNN-ℓ_1	6.1	11.3	11.3	11.3	11.3	11.3	11.3	28.6
	EQ	6.0	10.0	12.0	14.0	12.0	11.0	10.0	30.0
	LML	8.0	10.0	12.0	14.0	16.0	18.0	20.0	38.0
	SEO	9.4	15.4	18.1	20.8	18.1	17.1	16.1	70.1

4.2 Real-World Dataset

We tested the four 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 the remainder for testing. 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 and also is provided in Appendix B. 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 a handful of instances use $c_p < c_h$ to test this situation as well.

As shown in Figure 3, for this data set, the DNN-ℓ_1 and DNN-ℓ_2 algorithms both outperform the other three algorithms for every value of c_p/c_h. Among the three remaining algorithms, the results of the SEO algorithm are the closest to those of DNN. On average, its corresponding cost ratio is 1.23, whereas the ratios for EQ and LML are 1.26 and 1.53,

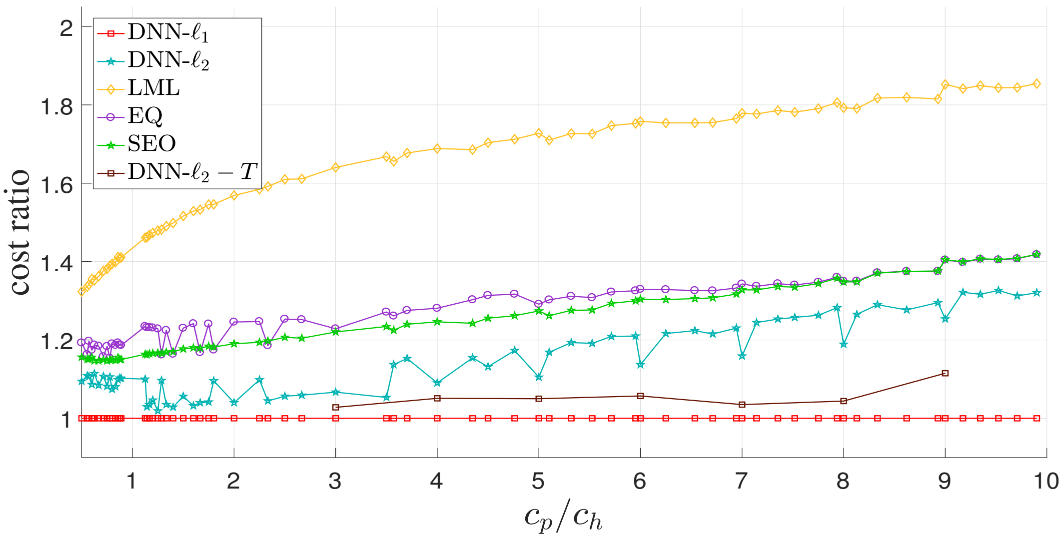


Figure 3: Ratio of each algorithm's cost to DNN-ℓ_1 cost on a real-world dataset.

respectively. The average cost ratio of DNN-ℓ_2 is 1.13. However, none of the other approaches are stable; their cost ratios increase with the ratio c_p/c_h.

Moreover, we note that 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. In order to see how much the results of DNN-ℓ_2 could be improved by better tuning, we applied more computation power there and tuned its parameters using a grid search. We selected some of the integer values of c_p/c_h and fixed the network as $[43, 350, 100, 1]$. Then, the network was tested with 702 different parameters and the best test result among them was selected. The grid search procedure is explained in Appendix C. The corresponding result is labeled as DNN-ℓ_2-T in Figure 3. As the figure shows, this approach has better results than the initial result; however, DNN-ℓ_1 is still better.

The DNN algorithms execute more slowly than some of the other algorithms. For the basket dataset, the SEO and EQ algorithms each execute in about 10 seconds. The DNN algorithms require about 600 seconds for training, while the LML algorithm requires about 1200 seconds for training. On the other hand, Both the DNN 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.

4.3 Randomly Generated Data

In order to further explore the performance of DNN versus the other methods, we generated 10,000 records and divided them into training (75%) and testing (25%) sets. 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 10,000. In each cluster i, data are drawn from

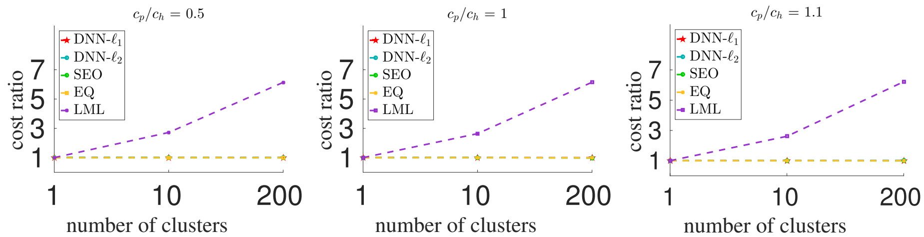


Figure 4: Ratio of each algorithm's cost to DNN-ℓ_1 cost on randomly generated data.

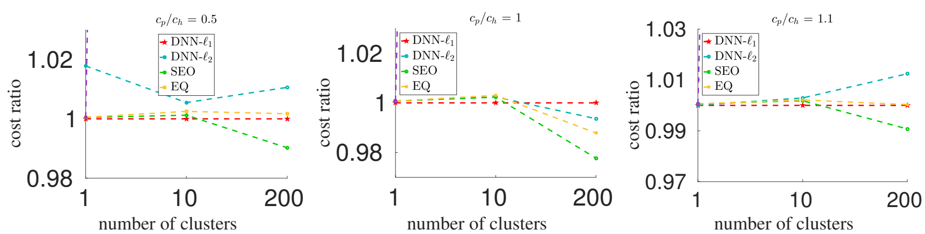


Figure 5: Ratio of each algorithm's cost to DNN-ℓ_1 cost on randomly generated data.

a $\mathcal{N}(50i, 25i^2)$ distribution. Each problem was solved for three values of c_p and c_h using all four algorithms (including both loss functions for DNN).

Figure 4 shows the results. As shown in the figure, if there is only a single cluster, all four algorithms produce nearly the same results. This case is essentially a classical newsvendor problem with 10,000 data observations, for which all algorithms do a good job of providing the order quantity. As the number of clusters increases, i.e., the number of samples in each cluster decreases, the performance of LML deteriorates somewhat. This is because there is not enough data for LML to learn the distribution well. In order to demonstrate the performance of the DNN-ℓ_1, DNN-ℓ_2, EQ, and SEO algorithms, a magnified version of Figure 4 is shown in Figure 5 (LML is omitted.). Also, for further clarification, the details of the costs is provided in Appendix D. As the number of clusters increases, the performance of SEO improves compared to that of DNN. This is because DNN does not have not enough data to learn the distribution well. Moreover, depending on the value of c_p/c_h, DNN-ℓ_2 sometimes has a smaller cost than DNN-ℓ_1.

Although SEO (and sometimes EQ) outperforms DNN when the number of clusters is large in Figure 5, we note that this experiment is somewhat biased in favor of those methods. In particular, the SEO algorithm *assumes* the demands are distributed from a known distribution and without any noise, and these data *are* in fact normally distributed, thus giving SEO an advantage since it only needs around 12 samples to get a good estimate of the mean and standard deviation. Moreover, as Bertsimas and Thiele (2005) mention, EQ works well with normally distributed data and is almost as good as SEO. In sum, when the data come from a known distribution, DNN, SEO, and EQ work well. However, when the distribution is unknown, DNN should be used, especially if there are sufficient training data available.

5 Conclusion

In this paper, we consider the multi-item newsvendor problem with data features. If the probability distribution of the demands is known, 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 multi-item, multi-feature newsvendor problem. 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 random 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. Moreover, when the data can be represented by a well-defined probability distribution, in the presence of enough training data, deep learning works as well as other approaches.

Motivated by the results of deep learning on the real world data-set, 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.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

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.

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

George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

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

Real Carbonneau, Kevin Laframboise, and Rustam Vahidov. Application of machine learning techniques for supply chain demand forecasting. *European Journal of Operational Research*, 184(3):1140–1154, 2008.

G Cardoso and F Gomide. Newspaper demand prediction and replacement model based on fuzzy clustering and rules. *Information Sciences*, 177(21):4799–4809, 2007.

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.

Sven F Crone, Michele Hibon, and Konstantinos Nikolopoulos. Advances in forecasting with neural networks? empirical evidence from the nn3 competition on time series prediction. *International Journal of Forecasting*, 27(3):635–660, 2011.

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.

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.

Mark Ko, Ashutosh Tiwari, and Jörn Mehnen. A review of soft computing applications in supply chain management. *Applied Soft Computing*, 10(3):661–674, 2010.

Nikolaos Kourentzes and Sven Crone. Advances in forecasting with artificial neural networks. *Working paper (Lancaster University Management School, Lancaster, UK),*.

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. ISSN 0167-8655.

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.

Chi-Jie Lu and Chi-Chang Chang. A hybrid sales forecasting scheme by combining independent component analysis with K-Means clustering and support vector regression. *The Scientific World Journal*, 2014, 2014.

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.

Cynthia Rudin and Gah-yi Vahn. The big data newsvendor : practical insights from machine learning analysis. *Cambridge, Mass. : MIT Sloan School of Management*, 2013.

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.

Manish Shukla and Sanjay Jharkharia. ARIMA models to forecast demand in fresh supply chains. *International Journal of Operational Research*, 11(1):1–18, 2011.

Robert H Shumway and David S Stoffer. *Time series analysis and its applications: with R examples*. Springer Science & Business Media, 2010.

Lawrence V Snyder and Zuo-Jun Max Shen. *Fundamentals of supply chain theory*. John Wiley & Sons, 2011.

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.

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.

Stijn Viaene, Bart Baesens, Tony Van Gestel, Johan AK Suykens, Dirk Van den Poel, Jan Vanthienen, Bart De Moor, and Guido Dedene. Knowledge discovery using least squares support vector machine classifiers: A direct marketing case. In *Principles of Data Mining and Knowledge Discovery*, pages 657–664. Springer, 2000.

Armando Vieira. Predicting online user behaviour using deep learning algorithms. *Computing Research Repository - arXiv.org*, http://arxiv.org/abs/1511.06247, 2015.

Shaomin Wu and Artur Akbarov. Support vector regression for warranty claim forecasting. *European Journal of Operational Research*, 213(1):196–204, 2011.

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.

Paul H. Zipkin. *Foundations of Inventory Management*. Irwin/McGraw-Hill, New York, 2000.

A The proof of Proposition 1 and Proposition 2

The proof of Proposition 1 and Proposition 2 are provided in this section.

Proposition 1. The gradient of the loss function 7 is:

$$\frac{\partial E_i}{\partial w_{jk}} = \begin{cases} c_p(y_i - d_i)a_j g'_k(z_k) & y_i < d_i \\ c_h(y_i - d_i)a_j g'_k(z_k) & d_i \leq y_i. \end{cases}$$

Proof: Consider the proposed revised Euclidean loss function defined in (7). The gradient of the loss function in cases of excess inventory and shortage are shown in (11) and (12) respectively.

$$\begin{aligned} \frac{\partial E_i}{\partial w_{jk}} &= c_h(y_i - d_i)\frac{\partial(y_i - d_i)}{\partial w_{jk}} = c_h(y_i - d_i)\frac{\partial(y_i)}{\partial w_{jk}} \\ &= c_h(y_i - d_i)\frac{\partial(g_k(z_k))}{\partial w_{jk}} = c_h(y_i - d_i)\frac{\partial(z_k)}{\partial w_{jk}}g'_k(z_k) \\ &= c_h(y_i - d_i)a_j g'_k(z_k), \end{aligned} \tag{11}$$

where i is one of the samples, w_{jk} represents one of the weights in the network, and $\frac{\partial(z_k)}{\partial w_{jk}} = g_j(z_j) = a_j$. For the shortage case also we have:

$$\begin{aligned} \frac{\partial E_i}{\partial w_{jk}} &= -c_p(d_i - y_i)\frac{\partial(d_i - y_i)}{\partial w_{jk}} = c_p(d_i - y_i)\frac{\partial(y_i)}{\partial w_{jk}} \\ &= c_p(d_i - y_i)\frac{\partial(y_i)}{\partial w_{jk}} = c_p(d_i - y_i)\frac{\partial(g_k(z_k))}{\partial w_{jk}} \\ &= c_p(d_i - y_i)\frac{\partial(z_k)}{\partial w_{jk}}g'_k(z_k) = c_p(d_i - y_i)a_j g'_k(z_k). \end{aligned} \tag{12}$$

Therefore, the gradient is:

$$\frac{\partial E_i}{\partial w_{jk}} = \begin{cases} c_p(d_i - y_i)a_j g'_k(z_k) & y_i < d_i \\ c_h(y_i - d_i)a_j g'_k(z_k) & d_i \leq y_i. \end{cases}$$

Proposition 2. The gradient of the loss function (8) is:

$$\frac{\partial E_i}{\partial w_{jk}} = \begin{cases} c_p a_j g'_k(z_k) & y_i < d_i, \\ c_h a_j g'_k(z_k) & d_i \leq y_i. \end{cases}$$

Proof: Consider the loss function (8). To determine the gradient with respect to the weights of the network, we first note that in the case of excess inventory ($d_i \leq y_i$),

$$\begin{aligned} \frac{\partial E_i}{\partial w_{jk}} &= c_h\frac{\partial(y_i - d_i)}{\partial w_{jk}} = c_h\frac{\partial(\hat{y}_i)}{\partial w_{jk}} \\ &= c_h\frac{\partial(g_k(z_k))}{\partial w_{jk}} = c_h\frac{\partial(z_k)}{\partial w_{jk}}g'_k(z_k) \\ &= c_h a_j g'_k(z_k), \end{aligned} \tag{13}$$

where i is one of the samples, w_{jk} represents one of the weights in the network, and $\frac{\partial(z_k)}{\partial w_{jk}} = g_j(z_j) = a_j$. For the shortage case also we have:

$$\begin{aligned} \frac{\partial E_i}{\partial w_{jk}} &= -c_p \frac{\partial(d_i - y_i)}{\partial w_{jk}} = c_p \frac{\partial(y_i)}{\partial w_{jk}} \\ &= c_p \frac{\partial(y_i)}{\partial w_{jk}} = c_p \frac{\partial(g_k(z_k))}{\partial w_{jk}} \\ &= c_p \frac{\partial(z_k)}{\partial w_{jk}} g'_k(z_k) = c_p a_j g'_k(z_k). \end{aligned} \tag{14}$$

Summing up (13) and (14), the gradient is:

$$\frac{\partial E_i}{\partial w_{jk}} = \begin{cases} c_p a_j g'_k(z_k) & y_i < d_i, \\ c_h a_j g'_k(z_k) & d_i \leq y_i. \end{cases}$$

B Experiments on Real Dataset

To support Figure 3, Table 3 shows the details of each algorithm's results. For each combination of cost coefficients c_h and c_p, the newsvendor cost is reported. Figure 3 shows the normalized values of this table.

C Grid Search for Basket Dataset

In order to obtain a good solution for the DNN-ℓ_2 model, a large network, i.e. two layers with 350 and 100 nodes in the first and second layer respectively, is used. In order to find the best set of parameters for this model a grid search is used. Three parameters of *lr*, λ, and γ were considered, which γ is the parameter for decaying the learning rate through the following formula:

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

For each of the parameters, the set of

$$\gamma \in \{0.001, 0.0008, 0.0009\}$$

$$\lambda \in \{0.01, 0.005, 0.001, 0.0005\}$$

$$\text{lr} \in \{0.001, 0.0007, 0.0005, 0.0001\}$$

were considered and the testing result of the 48 problems were obtained. According to the test results, the new set of parameters

$$\gamma \in \{0.001, 0.005, 0.0005, 0.0001, 0.00005, 0.000005\}$$

$$\lambda \in \{0.01, 0.009, 0.007, 0.005, 0.001, 0.0005, 0.0003\}$$

$$\text{lr} \in \{0.0001, 0.0005, 0.00008, 0.00005, 0.00003, 0.00001, 0.000005\}$$

used and the cost for each set of parameters obtained. According to the cost of these 294 problems, the new set of parameters

$$\gamma \in \{0.01, 0.005, 0.001, 0.0001, 0.0005, 0.00005\}$$

Table 3: The results of the real-world data set with different cost coefficients.

c_p	c_h	DNN-ℓ_1	DNN-ℓ_2	SEO	EQ	LML	c_p	c_h	DNN-ℓ_1	DNN-ℓ_2	SEO	EQ	LML
1	2	124146	133859	143913	148472	164751	6	7	629404	691547	727046	750679	888441
1	3	137417	152912	160295	163914	173471	6	8	663730	733754	760879	778486	916171
1	4	144273	165672	172092	174405	177780	6	9	688757	766955	790750	816813	939529
1	5	151241	177108	182149	184263	180561	7	1	239553	247996	318109	321695	426128
1	6	153691	185181	191051	193865	182366	7	2	371641	391432	458719	472465	619832
1	7	157386	192714	197776	203075	183689	7	3	467580	488345	559830	554484	744296
1	8	160843	199288	203308	212108	184701	7	4	539901	562339	639237	669949	834352
1	9	161094	204443	208071	221168	185472	7	5	601328	618439	703597	699860	900997
2	1	144108	145982	171861	179881	226533	7	6	649676	672279	756843	800671	953750
2	2	385081	203413	229364	236522	284687	7	7	1311995	759641	802775	802775	996389
2	3	229891	241495	263583	272271	313177	7	8	731843	806940	842125	868940	1031198
2	4	250005	267478	287826	296944	329504	7	9	763133	843726	876971	880424	1060148
2	5	261991	292685	305934	316754	339955	8	1	250940	261991	338263	338773	449809
2	6	273589	309279	320590	327828	346939	8	2	391459	414813	489331	502944	662991
2	7	281565	326053	333229	342280	351863	8	3	498587	528026	600479	624198	803337
2	8	288539	340411	344184	348810	355560	8	4	579950	615574	687444	719524	906083
2	9	294467	355492	354420	360488	358751	8	5	644137	664679	760038	799799	984824
3	1	173831	178698	212549	210762	285696	8	6	702048	735959	820322	857057	1046027
3	2	248496	255234	292997	306270	377514	8	7	750172	772090	871999	923991	1096832
3	3	574214	310990	344046	344046	427030	8	8	1498281	875504	917457	931920	1138731
3	4	331546	350744	380439	395896	458086	8	9	832200	917273	957156	987201	1173953
3	5	353635	384183	408737	415168	479402	9	1	254213	283461	357153	356970	470851
3	6	373676	413324	431738	445416	494256	9	2	412264	437719	517578	541569	702305
3	7	387612	434356	450610	461163	505575	9	3	523699	556175	637646	632400	857088
3	8	399138	455627	466753	485499	513812	9	4	613242	643004	732238	764686	971696
3	9	410051	476297	480884	491742	520408	9	5	686564	715839	811460	806376	1061575
4	1	196184	206096	244665	251472	331496	9	6	747900	772044	878991	918810	1132503
4	2	287446	297760	343722	359762	453045	9	7	803321	831357	936420	933363	1190322
4	3	350737	358397	410161	430740	523018	9	8	848323	887806	987020	1047311	1239500
4	4	756157	424856	458728	473044	569373	9	9	1684718	986566	1032139	1032139	1281072
4	5	431824	463805	496377	513928	602144	10	1.01	265791	292800	377026	376916	492851
4	6	460170	501022	527167	544542	626353	10	1.03	270559	292530	381021	380816	498970
4	7	480738	531870	553270	575451	644744	10	1.05	273892	363281	384971	384716	504983
4	8	496894	558302	575651	593888	659014	10	1.07	276335	315897	388891	388616	510991
4	9	511100	590499	595040	613648	670755	10	1.09	280702	370950	392774	392517	516942
5	1	213419	224049	271893	275569	368704	10	1.12	289646	318888	398538	398367	525777
5	2	320732	338830	387037	401954	516360	10	1.16	295382	377104	406148	406167	537354
5	3	395799	411506	466606	462326	606570	10	1.20	301817	316214	413626	413967	548543
5	4	450983	459499	526125	554031	667124	10	1.23	310879	328767	419167	419817	556766
5	5	937452	532351	573411	573411	711716	10	1.26	312809	342656	424649	425414	564971
5	6	533012	576090	611854	632418	745420	10	1.29	320027	329004	430079	431259	572998
5	7	560653	620807	644350	648840	771767	10	1.33	327581	362691	437217	439051	583567
5	8	586391	653398	672516	696223	792866	10	1.36	331131	370627	442491	444895	591313
5	9	606136	672019	697512	704984	809882	10	1.40	338567	355176	449482	452687	601580
6	1	227098	240079	296150	301896	399201	10	1.44	346509	356019	456408	461751	611645
6	2	348202	368487	425097	440686	571392	10	1.49	355572	414991	464886	471317	623947
6	3	434307	455712	515583	539643	679562	10	1.53	361196	386019	471533	478970	633571
6	4	498145	517370	585994	612540	755002	10	1.60	370607	383246	482945	492362	650061
6	5	550280	575834	641589	677351	810650	10	1.68	381369	401133	495701	505532	668414
6	6	1124513	650301	688093	709566	854048	10	1.75	391696	393389	506647	517986	684349

$$\lambda \in \{0.01, 0.005, 0.001, 0.0005, 0.0001, 0.00005\}$$

$$\text{lr} \in \{0.001, 0.005, 0.0005, 0.0001, 0.00005, 0.00003, 0.00001, 0.000009, 0.000008, 0.000005\}$$

were considered and the best set of parameters is selected among the 360 sets.

D Experiments on Randomly Generated Data

In order to obtain the results on the simulation data, a similar tuning procedure as mentioned in Appendix B is used. Table 4 shows the detailed results of the simulated data which we discussed in Section 4.3. As shown, as the number of training samples in each cluster decreases, the performance of DNN degrades.

Table 4: Detailed costs of the four algorithms on randomly generated data.

# Clusters	c_p	c_h	DNN-ℓ_1	DNN-ℓ_2	SEO	EQ	LML
1	1	0.5	13436	13676	13443	13444	13444
1	1	1	19659	19674	19674	19673	19673
1	1	1.1	20607	20608	20620	20619	20619
10	1	0.5	76274	76693	76369	76467	207304
10	1	1	110818	111080	111074	111153	292354
10	1	1.1	116190	116529	116403	116437	303504
200	1	0.5	703399	710868	696546	704613	2036633
200	1	1	1042052	1035399	1018809	1029408	4457149
200	1	1.1	1078734	1092147	1068727	1079067	4879109