Abstract

In multi-echelon inventory systems the performance of a given node is affected by events that occur at many other nodes and at many other time periods. For example, a supply disruption upstream will have an effect on downstream, customer-facing nodes several periods later as the disruption "cascades" through the system. There is very little research on stock-out prediction in single-echelon systems and (to the best of our knowledge) none on multi-echelon systems. However, in real the world, it is clear that there is significant interest in techniques for this sort of stock-out prediction. Therefore, our research aims to fill this gap by using DNN to predict stock-outs in multi-echelon supply chains.

1 Introduction

A multi-echelon network is a chain of nodes which aims to provide a product or service to its customers. Each network consists of production and assembly lines, warehouses, transportation systems, retail process, etc., and each of them is connected to at least one other node. The most downstream nodes of the network face the customers, which usually present an external stochastic demand. Also, the most upstream nodes interact with third-party vendors, which offer an unlimited source of raw materials and goods. A sample of a multi-echelon network is shown in Figure 1 which is a distribution network, e.g a retail supply chain.

The network’s goal is to find a compromise between the profit and service level (a number between zero and one that determines the percent of the customer’s orders that are satisfied on time) to its customers. For example, a retail network may decide to change the number of retail stores to increase its service availability and reach more sales, which results also in a higher cost for the system. In this case, the relevant decisions are how many, where, and when they should be opened/closed to maximize the profit. Facility location and network design are the common mathematical programming
problems to provide the optimal decision in those questions. Similarly, the problems in production and assembly lines, are where, when, how, and how much to produce of which item. Scheduling and capacity management are the common problems in this area. Distribution systems also must decide when, where, how much of which item should be moved. The transportation problem is the most famous problem which answers these questions. Inventory management systems also decide when, where, how and how much of which item should be ordered. In well-run companies, there are multiple systems which optimize those problems to provide the best possible balance between service level and profit. In this thesis we focus on inventory management systems to provide an algorithm that answers some of the questions in an environment with stochastic demand.

Inventory management systems also balance between the service level and profit, which equivalently can be done by balancing the stock-out level and holding safety stock. Stock-outs are expensive and common in supply chains, e.g., distribution systems face 6% – 10% stock-out for non-promoted items and 18% – 24% for promoted items [Gartner Inc.]. The stock-outs result in significant lost revenue for the supply chain. When a company is faced with a stock-out, roughly 70% of customers do not wait for inventory to be replenished, but instead purchase the items from a competitor [Bharadwaj et al. 2002]. Thus, in order not to lose customers and maximize profit, companies should have an inventory management system to provide high service level with small cost.

Supply chains have different tools to balance between the service level and the stock-out costs, and all of those tools use a kind of optimization or decision making procedure. For example, some companies, e.g., those that produce huge products, such as ships that cannot hold inventory and have to balance their service level and service costs. Others can hold inventory and in this case the optimization problem finds a compromise between the holding and stock-out costs. Usually these models assume a given service level, and minimize the corresponding costs for that. As mentioned, the other relevant questions of an inventory management system are when, where, and how much of each item should be ordered, moved, stored, or transported. These questions can be optimally answered by finding the optimal $(R, Q)$ or $(s, S)$ policy, or by finding the optimal safety stock for each node of the network. This is called the Stochastic Service Model (SSM) approach, which considers stochastic demand and stochastic lead times due to upstream stockouts. The optimal base-stock level can be found for serial systems by solving a sequence of single variable convex problems [Clark and Scarf, 1960]. Similarly, by converting an assembly system (in which each node has at most one successor) to an equivalent serial system, the optimal solution can be achieved [Rosling, 1989].

For more general network topologies, no efficient algorithm exists for finding optimal base-stock levels, and in some cases the form of the optimal inventory policy is not even known [Zipkin, 2000].

Another approach for dealing with multi-echelon problems is the Guaranteed Service Model (GSM) approach. GSM assumes the demand is bounded above, or equivalently the excessive demand can be satisfied from the outside of the system, e.g., a third party vendor. It assumes a Committed Service Time (CST) for each node, which is the latest time that the node will satisfy the demand of its successor nodes. By this definition, instead of optimizing the inventory level, the GSM model optimizes the lead times to provide the CST for each node, or equivalently it finds the base stock values for each node of the network, to minimize the holding costs (since there is a CST, theoretically a stock-out cost never happens). This approach can handle much more general supply chain topologies, typically using either dynamic programming [Graves, 1988; Graves and Willems, 2000] or MIP techniques [Magnanti et al., 2006].

Due to their intractability, SSM models are not implemented in any commercial software packages we are aware of, despite their long history in the research literature. GSM models are implemented in several commercial software packages (e.g., Llamasoft [Llamasoft, Inc., 2016], Logility [Logility, Inc., 2016], and SAP [SAP Inc., 2016]) and are widely used in practice, but the underlying models are somewhat “fragile” in the sense that they depend heavily on certain assumptions about the system’s operation. Complicating factors such as additional stochastic elements, side constraints, or alternate objective functions can not be handled by existing algorithms and implementations. Moreover, these algorithms use a parametric approach, assuming that the demand distributions are known. To implement such a model in practice, a distribution has to be approximated, and then an optimization problem has to be solved—this is the SEO approach with all its merits and weaknesses. For a review of GST and SST Models see Eruguz et al. [2016] and Simchi-Levi and Zhao [2011].

The sense among (at least some) supply chain practitioners is that the current set of inventory optimization models are sufficient to optimize most systems as they function normally. What keeps
these practitioners up at night is the deviations from “normal” that occur on a daily basis and that pull the system away from its steady state. In other words, there is less need for new inventory optimization models and more need for tools that can help when the real system deviates from the assumptions that were modeled in the first place.

On the other hand, consider an ideal condition without any abnormal or unpredicted events. The SST approach considers a given service level and provides the optimal policy that obtains the lowest possible cost. The policy guarantees having fewer stock-outs than the given service level on average. However, it does not provide information about the points in time at which a stock-out may happen. Since stock-outs are expensive, companies prefer to somehow predict the stock-outs in their supply chain so that they be able to prevent them. Thus, an algorithm that takes a snapshot of the supply chain at a given point in time, makes predictions about how individual components of that supply chain will perform and whether they will face a stock-out or not in the near future, would be interesting.

Note that the systems that are modeled using the GSM approach also faces stock-outs, even though the GSM model itself assumes they do not. The GSM approach assumes a bound on the demand value; when the real-world demand exceeds that bound, it may not be possible to satisfy the demand externally (as the GSM model assumes), so in the real world the GSM approach also may face stock-out. Therefore, stock-out prediction can be interesting for fans of both SSM and GSM approaches.

In a single node network, the stock-out probability and stock-out prediction can be obtained, if the probability distribution of the demand is known (see Appendix A). However, to the best of our knowledge, there is not any algorithm to provide stock-out predictions in multi-echelon networks. To address this need, in this chapter we propose an algorithm to provide stock-out prediction in multi-echelon networks, which works for any network topology, providing stock-out predictions for each node of the system.

The remainder of this chapter is organized as follows. In Section 2, we introduce our algorithm. Section 3 describes three naive algorithms to predict stock-outs. To demonstrate the efficiency of the proposed algorithm in terms of solution quality, we compare our results with the best naive algorithms in Section 4. Finally, Section 5 concludes the chapter and proposes future studies.

## 2 Solution Method

We develop an approach to provide stock-out predictions for multi-echelon networks with available data features. Our algorithm is based on deep learning, which is a non-parametric algorithm. In the area of supply chain it is usually has applied to obtain demand prediction [Efendigil et al., 2009, Vieira, 2015], obtaining quantile regression [Taylor, 2000, Kourentzes and Crone, 2010, Cannon, 2011, Xu et al., 2016], and forecasting [Ko et al., 2010, Kourentzes and Crone, 2010, Qiu et al., 2014, Crone et al., 2011]. It also is successfully applied to the newsvendor problem [Oroojlooyjadid et al., 2016]. The basics of deep learning are reviewed in Goodfellow et al. [2016].

Consider a general multi-echelon network with \( n \) nodes and assume the topology of the network is known. For each node of the network, assume the history of the Inventory Level (IL), Inventory in Transit (IT), and the stock-out status in the historical periods are given. Stock-out status for each node is a True or False boolean, where True stands for the case that the node experienced a stock-out. Note that we use 1 interchangeably instead of True and 0 for False. The demand distribution is known or unknown; nevertheless, the historical demand information is available. The goal is to provide a stock-out prediction for each node of the network for the next period.

The available information that can be provided as input to the DNN algorithm is the values of the \( p \) available features, e.g., day of week, month of year, weather information, etc., along with the available historical information of each node, e.g. IL, IT. So, the available information at time \( t \) can be written as:

\[
[f^1_t, \ldots, f^p_t, [IL_i, IT_i]]_{i=1},
\]

where \( f^1_t, \ldots, f^p_t \) stands for the value of the \( p \) features in time step \( t \). However, the DNN needs a fixed input size and cannot handle the variable input size in (1). So, we just consider historical information from \( k \) most recent periods instead of the full information. Although this results in partial information of the network, it unifies and reduces the input size, which is more computationally affordable, and selecting large enough \( k \) provides a good level of information about the system. So,
the input of the DNN is
\[ [f^1_t, \ldots, f^p_t, [IL_i, IT_i]]_{t=k+1} \].

Also, the DNN algorithm predicts the stock-out status of each node for the next period. Therefore, the output of the DNN is the stock-out prediction for time \( t+1 \), for each node of the network, i.e. \( y_t = [y^1_t, \ldots, y^n_t] \) which is an array of length \( n \). Each of the \( y^i_t, i \in 1, \ldots, n \), is a binary value, 1 if the node in period \( t \) has stock-out and 0 otherwise.

The DNN algorithm randomly initializes the variables. Loss functions measure the outputs’ quality to see how well the DNN is working. Since the problem has binary outputs, among the current loss functions the following loss functions are tested:

- Hinge loss function
- Euclidean loss function
- Soft-Max loss function

Hinge (7) and Euclidean (9) loss functions are reviewed in section C. Consider the soft-max function (2) which is a generalized version of logistic regression.

\[ \sigma(z_u) = \frac{e^{z_u}}{\sum_{v=1}^{U} e^{z_v}}; \quad \forall u = 1, \ldots, U \]  

where \( U \) is the number of possible categories, \( z_u = \sum_{i=1}^{M_{L-1}} a^{L-1}_i w_{i,u}, L \) is the number of layers in the DNN network, and \( M_{L-1} \) represents the number of nodes in layer \( L - 1 \). Similarly, (3) demonstrates the soft-max loss function:

\[ E = -\frac{1}{N} \sum_{i=1}^{N} \sum_{u=1}^{U} I\{y_i = u - 1\} \log \frac{e^{z_u}}{\sum_{v=1}^{U} e^{z_v}}, \]

where \( N \) is the total number of training samples, in our problem \( U = 2 \), and \( E \) is the loss function value which provides the corresponding cost of a given classification. Hinge and Soft-max loss functions provide a binary output and their output represents True and False value as required in the prediction procedure. However, the Euclidean loss function provides a continuous value, which should be changed to a binary output. Since the training labels are either 0 or 1, rounding the outputs provides a binary value, zero or one. So, any output less than 0.5 is rounded to 0, and all outputs greater than or equal to 0.5 are rounded to 1.

Based on the loss function, the gradients of the weights of the network are obtained using back-propagation or automatic differentiation. Then, the network’s weights are updated through a first or second order algorithm, e.g. gradient descent algorithm, SGD, SGD with momentum, LBFGS, etc. This procedure repeats iteratively to meet one of the stopping criteria, i.e.

- The loss function value < 1e - 6
- The number of iterations reaches MaxIter

The loss function provides a measure for monitoring the improvement of the DNN algorithm through the iterations. However, it cannot be used to measure the quality of prediction, and it is not meaningful by itself. Since the prediction output is a binary value, test error—the number of wrong predictions divided by the number of samples—is an appropriate measure. Moreover, statistics on false positives (type I error—which is the incorrect rejection of a true null hypothesis) and false negatives (type II error—which is failure to reject a true null hypothesis) are helpful and we use them to get more insights about how the algorithm works.

### 3 Naive Approaches

In this section three naive approaches to predict stock-outs are proposed. These algorithms are used as baselines for measuring the quality of the DNN algorithm. They are easy to implement, but they do not consider the system state at any nodes other than the node for which we are predicting stockouts.
(The proposed DNN approach, in contrast, uses the state at all nodes to provide a more effective prediction.)

In all of the naive algorithms we use $IP_t$ to denote the Inventory Position in period $t$. Also, $v, u$ are the numbers of the training and testing records, respectively, and $d = [d_1, d_2, \ldots, d_v]$ is the demand of the customers in each period. Finally, $RNG(x_t)$ is a function which gives the range of $x_t$, and function approximator takes a list of numbers, fits a normal distribution to it, and returns the corresponding parameters of the normal distribution.

**Algorithm 1 Naive Algorithm 1**

1: procedure *NAIVE*-

2: for $i = 1 : n$ do

3: $s = []$; set $\alpha$;

4: # TRAINING PROC

5: for $t = 1 : v$ do

6: if $y_{t+1} = 1$ then

7: $s += IP_t$;

8: end if

9: end for

10: $\mu_s, \sigma_s = \text{approximator}(s)$;

11: $\eta_\alpha = \mu_s + \Phi^{-1}_\alpha(\sigma_s)$;

12: # TESTING PROC

13: for $t = 1 : u$ do

14: if $IP_t < \eta_\alpha$ then

15: prediction($t$) = 1;

16: else

17: prediction($t$) = 0;

18: end if

19: end for

20: return prediction

21: end for

22: end procedure

Naive algorithm 1 obtains the list of all periods in the training database that face a stock-out in their immediately next period and gathers their corresponding inventory positions, i.e. $s = \{[IP_t \text{ if } y_{t+1} = 1]\}_{t=0}^v$. Then, it approximates a probability distribution of those inventory positions by fitting a normal distribution $N(\mu_s, \sigma_s)$ and predicts a stock-out if $IP_t$ is greater than or equal to the $\alpha$th quantile of that distribution. The value of $\alpha$ is determined by the modeler.

Naive algorithm 2 groups the inventory positions into multiple ranges, calculates the frequency of a stock-out in the training data for each range, and then predicts a stock-out if the $IP_t$'s range led to a stock-out more than 50% of the time in the training data.

Naive algorithm 3 uses classical inventory theory that says the inventory level in period $t + L$ equals $IP_t$ minus the lead-time demand, where $L$ is the lead time [Snyder and Shen (2011), Zipkin (2000)]. The algorithm estimates the lead-time demand distribution by fitting a normal distribution based on the training data, then predicts a stockout in period $t + 1$ if $IP_t$ is less than or equal to the $\alpha$th quantile of the estimated lead-time demand distribution, where $\alpha$ is a parameter chosen by the modeler.

In algorithms 1 and 3, the value of $\alpha$ has to be selected by the modeler and whatever it is, it directly affects $\eta_\alpha$. Small $\alpha$ results in a small $\eta_\alpha$, so that the algorithm predicts fewer stock-outs. Generally, as $\alpha$ decreases, the number of false positive errors decreases compared to the number of false negative errors, and vice versa. Thus, selecting an appropriate value of $\alpha$ is important and directly affects the output of the algorithm. Indeed, the value of $\alpha$ has to be selected according to the preferences of the company running the algorithm. For example, a company may have very expensive stock-outs. So, it may choose a very large $\alpha$ so that the algorithm predicts a lot of stock-outs, along with many more false positive errors, and then checks them one by one to prevent the stock-outs. In this situation the number of false positive errors increases; however, the company faces fewer false negative errors,
Algorithm 2 Naive Algorithm 2

1: procedure NAIVE-2
2: for $i = 1 : n$ do
3:     $rng = \text{RNG}(IP)$;
4:     Divide $rng$ to $k$ equal intervals $[l_i, u_i], \forall i = 1, \cdots, k$;
5:     $SO_i = NSO_i = 0, \forall i = 1, \cdots, k$;
6:     # TRAINING PROC
7:     for $t = 1 : v$ do
8:         if $IP_t \in [l_i, u_i]$ then
9:             if $y_{t+1} = 1$ then
10:                $SO_i + = 1$;
11:            else
12:                $NSO_i + = 1$;
13:        end if
14:     end if
15: end for
16: # TESTING PROC
17: for $t = 1 : u$ do
18:     if $IP_t \in [l_i, u_i]$ then
19:         if $SO_i > NSO_i$ then
20:             prediction($t$) = 1;
21:         else
22:             prediction($t$) = 0;
23:        end if
24:     end if
25: end for
26: return prediction
27: end for
28: end procedure

Algorithm 3 Naive Algorithm 3

1: procedure NAIVE-3
2: set $\alpha$;
3: for $i = 1 : n$ do
4:     # TRAINING PROC
5:     $\mu_d, \sigma_d = \text{approximator}(d)$;
6:     $\eta_\alpha = \mu_d + \Phi_{\alpha}^{-1}(\sigma_d)$;
7:     # TESTING PROC
8:     for $t = 1 : v$ do
9:         if $IP_t < \eta_\alpha$ then
10:            prediction($t$) = 1;
11:        else
12:            prediction($t$) = 0;
13:        end if
14:     end for
15: return prediction
16: end for
17: end procedure
which are costly. In order to determine an appropriate value of \( \alpha \), the modeler should consider \( c_p \) and \( c_n \), which stand for the cost of false positive and negative errors, respectively. According to these costs, an appropriate \( \alpha \) can be determined.

Similarly, in the DNN algorithm one should have control over the number of stock-out predictions, as well as control over the number of false positive and false negative errors. The original DNN algorithm provides one prediction, in which the false positive and negative errors are weighted equally. DNN does not have a parameter like \( \alpha \) that directly controls the errors. Instead, the loss function can provide control over the number of stock-out predictions, since the DNN’s output is directly affected by its loss function.

The loss functions mentioned in Section 2 do not have any weighting coefficient, and place equal weight between selecting 0 (predicting no stock-out) and 1 (predicting stock-out). So, those loss functions should be revised to provide appropriate control over the DNN’s prediction. That can be done by weighing the corresponding loss of each of the output "0" or "1", and the appropriate weights could be \( c_n \) and \( c_p \) for each case "0" and "1", respectively. In this way, when \( c_p < c_n \), the DNN tries to have a smaller number of cases with \( \text{True} \{ y_i = 0 \} \), so it predicts more stock-outs to result in a smaller number of false negative errors and a larger number of false positive errors. Similarly, when \( c_p > c_n \), the DNN predicts a smaller number of stock-outs to have smaller number of \( \text{True} \{ y_i = 1 \} \). So, it predicts a smaller number of false positive errors and a larger number of false negative errors. Also, in case that \( c_p \) and \( c_p \) are equal, it works similar to the original soft-max loss function. Following this weighting, the revised hinge, Euclidean, and soft-max loss functions are shown in (4), (5), and (6) respectively.

\[
E = \frac{1}{N} \sum_{i=1}^{N} E_i 
\]

\[
E_i = \begin{cases} 
  c_n \max(0, 1 - y_i \hat{y}_i), & \text{if } y_i = 0 \\
  c_p \max(0, 1 - y_i \hat{y}_i), & \text{if } y_i = 1,
\end{cases} 
\]

\[
E = \frac{1}{N} \sum_{i=1}^{N} E_i 
\]

\[
E_i = \begin{cases} 
  c_n \|y_i - \hat{y}_i\|^2_2, & \text{if } y_i = 0 \\
  c_p \|y_i - \hat{y}_i\|^2_2, & \text{if } y_i = 1,
\end{cases} 
\]

\[
E = -\frac{1}{N} \sum_{i=1}^{N} \sum_{u=1}^{U} w_u I\{y_i = u - 1\} \log \frac{e^{z_{u_i}}}{\sum_{v=1}^{U} e^{z_{v_i}}},
\]

where \( U = 2 \) and \( w_1 = c_n \) and \( w_2 = c_p \). Thus, these loss functions allow one to manage the number of false positive and negative errors.

4 Numerical Experiments

In order to check the validity and accuracy of our algorithm, we conducted a series of numerical experiments. Since there is no publicly available data of the type needed for our algorithm, we built a simulation model that assumes each node follows a base-stock policy and can make an order only if its predecessor has enough stock to satisfy it so that only the retailer nodes face stock-outs. The simulation records several state variables for each of the \( n \) nodes for each of the \( T \) time periods. Figure 2 shows the flowchart of the simulation algorithm used.

To see how our algorithm works with different network topologies, we conducted multiple tests on several supply chain network topologies, ranging from a simple series system to complex networks containing (undirected) cycles and little or no symmetry. These tests are intended to show how the DNN approach is robust on simple or very complex networks. From the list of the analyzed networks, the results of the following networks are presented in this section:

- Serial network with 11 nodes.
- Tree network with 11 nodes.
Observe demand for node \( i \)

\[
IL_i = IL_i - d_i \quad t > T
\]

Stop

\[
\text{Set } t = t + 1
\]

Observe transportation lead time \( L_i \) for node \( i \)

\[
IT_i = IT_i + L_i, \quad i = O_i
\]

Yes

Observe demand for node \( i \)

\[
IL_i = IL_i + IT_i, \quad t
\]

No

Obtain related costs of the period

\[
O_i = \min \{IL_\text{prec}(i), S_i - IP_i\}
\]

No

\[
i > n
\]

Yes

\[
IL_i = IL_i - d_i
\]

\[
i = i + 1
\]

Set \( i = 1 \)

Figure 2: The simulation algorithm which is used to simulate a supply network

Figure 3: The analyzed networks in stock-out prediction

- Distribution network with 13 nodes.
- Complex network I with 11 nodes, three retailers and one warehouse.
- Complex network II with 11 nodes, one retailer and two warehouses.

The corresponding networks are shown in Figure 3.

For each of the network topologies, \( 10^6 \) periods of data are generated by the simulation, of which 75% are used for training (and validation) and the remaining 25% for testing. Also, for all of the problems we used a fully connected DNN network with 350 and 150 sigmoid nodes in the first and second layers, respectively. The inputs are the inventory levels and on-order inventories for each node from each of the 11 most recent periods, and the output is the binary stock-out predictor for each of the nodes. A general view of the DNN network is shown in Figure 4. Among the loss functions reviewed in Section 2, the soft-max loss function had the best accuracy in initial numerical experiments. Thus, the soft-max loss function was selected and its results are provided. To this end,
the weighted soft-max function and its gradient (see appendix B) are implemented in caffe, which is a DNN computation framework [Jia et al., 2014], and all of the tests are done on machines with 16 AMD cores and 32 GB of memory. In order to optimize the network, the SGD algorithm—batch of 50—with momentum is used, and each problem is run for at most 27 epochs. Finally, since the values of $c_n$ and $c_p$ and $\alpha$ are not known in our study, we tested 99 values of $\alpha$ and 99 values of $(c_n, c_p)$.

On the other hand, the deep learning algorithm is scale dependent, i.e. the parameters of the network, ($\gamma$, learning rate, momentum, etc.) are dependent on the value of $c_p$ and $c_n$. Thus, a set of appropriate parameters of the network for a given set of cost coefficients $(c_p, c_n)$ does not necessarily work for another set $(c'_p, c'_n)$. This means that for each set of $(c_p, c_n)$, the DNN’s parameters should be tuned. However, the tuning procedure is expensive, so we tuned the parameters for $c_p = 2$ and $c_n = 1$ and used it for other sets of costs, in all network topologies. However, in complex network [17] we did not get good convergence with similar parameters, so we tuned the network for 10 other cost coefficients to make sure that we can get competitive results for all sets of costs.

In what follows, we demonstrate the results of the four algorithms in seven experiments. Section 4.1 presents the results of the serial network, and Sections 4.2, 4.3, 4.4, and 4.5 provide the results of the tree, distribution, complex I, and complex II networks respectively. Then, in Section 4.6 we analyze the results of a distribution network with multiple items with dependent demand. Finally, Section 4.7 shows the results of a multi-period prediction in a distribution network. In each of the network topologies, the false positive vs. false negative errors are plotted for all algorithms to compare their accuracies. Also, two other figures are provided which show the accuracy vs. false positive and negative errors to provide better insights into the way that the algorithm works compared to other algorithms.

### 4.1 Results: Serial Network

Figure 5 shows the serial network with 11 nodes. The training dataset is used to train all four algorithms and the corresponding results are shown in Figures 6 and 7. Figure 6 plots the false-negative errors vs. the false-positive errors for each approach and for a range of $\alpha$ values for the naive approaches and a range of weights for the weighted DNN approach. Points closer to the origin indicate more desirable solutions. Since there is just one retailer, the algorithms predict stock-out for $2.5e5$ samples and the number of errors in both Figures should be compared to $2.5e5$. The DNN
approach always dominates the naive approaches, with the unweighted version providing a slightly better accuracy but the weighted version providing more flexibility. For any given false-positive error, the number of false-negative errors of DNN and WDNN algorithms are smaller than the naive approaches, similarly for a given false-negative error. The three naive approaches are similar to each other. Moreover, Figure 7 plots the errors vs. the accuracy of the predictions.

Figure 6: False positives vs. false negatives for serial network

Figure 7: Accuracy of each algorithm for serial network

4.2 Results: Tree Network

Figure 8 shows the tree network with 11 nodes and Figures 9 and 10 present the results obtained. Since there are 10 retailers, it is more challenging than the serial network. The algorithms predict stock-out for 2.5e6 samples and the number of errors in both figures should be compared to 2.5e6.
Figure 8: The tree network

Figure 9 shows the false-negative errors vs. the false-positive errors for each approach and for a range of $\alpha$ values for the naive approaches and a range of weights for the weighted DNN approach. DNN and weighted DNN dominate the naive approaches. Other three naive approaches do almost similar to each other. Moreover, Figure 10 plots the errors vs. the accuracy of the predictions.

Figure 9: False positives vs. false negatives for tree network

Figure 10: Accuracy of each algorithm for tree network
4.3 Results: Distribution Network

Figure 11 shows the distribution network with 13 nodes, and the corresponding results of the four algorithms are shown in Figures 12 and 13.

As Figure 12 shows, the DNN approach dominates the naive approaches; however, weighted DNN does not dominate in the end tails that is because of the tuning of the network’s parameters. Among the three naive approaches, Naive-3 apparently dominates Naive-1, since the demand data comes from a normal distribution without any noise, and the algorithm also approximates a normal distribution which needs around 12 samples to get a good estimate of the mean and standard deviation. Indeed, the experiment is biased in favor of algorithm Naive-3. Moreover, Figure 13 plots the errors vs. the accuracy of the predictions.

Figure 12: False positives vs. false negatives for distribution network

Figure 13: Errors vs. accuracy of predictions for distribution network
Figure 13: Accuracy of each algorithm for distribution network

4.4 Results: Complex Network I, two warehouses

Figure 14 shows a complex network with two warehouses, and the corresponding results of four algorithms are shown in Figures 15 and 16.

Figure 14: The complex network, two warehouses

Figure 15 plots the false-negative errors vs. the false-positive errors for each approach and for a range of $\alpha$ values for the naive approaches and a range of weights for the weighted DNN approach. The DNN approach dominates the naive approaches for most cases, but does worse when false-positives are tolerated in favor of reducing false-negatives. The average accuracy rates for this system are 91% for WDNN and 97% for DNN, which show the importance of parameter tuning for each weight of the weighted DNN approach. Tuning it for each weight individually would improve the results significantly (but increase the computation time). Moreover, Figure 16 plots the errors vs. the accuracy of the predictions.
4.5 Results: Complex Network II, three retailers

Figure 17 shows the complex network with three retailers and the corresponding results of each algorithm are shown in Figures 18 and 19.

Figure 17: The complex network, three retailers

Figure 18 plots the false-negative errors vs. the false-positive errors for each approach and for a range of $\alpha$ values for the naive approaches and a range of weights for the weighted DNN approach. Like the other network topologies, the tuned parameters for the case of $c_p = 2$ and $c_n = 1$ are used for all
Table 1: The parameters used for each network

| Network  | lr   | γ   | λ   |
|----------|------|-----|-----|
| Tree     | 0.001| 0.0005 | 0.0005 |
| Distribution | 0.0005 | 0.001 | 0.0005 |
| Serial   | 0.001 | 0.0005 | 0.0001 |
| Complex-I | 0.05  | 0.000005 | 0.000005 |
| Complex-II | 0.001 | 0.0005 | 0.0005 |

other cases of \((c_p, c_n)\). However, for some of the \((c_p, c_n)\) values the network did not converge. Thus, for 10 values of \((c_p, c_n)\) we used the parameters shown in Table 1. Even though, we used one set of parameters for all ten \((c_p, c_n)\) values, their results dominate the naive algorithms. This also suggests that parameter tuning would result in dominance of the DNN approach. Moreover, Figure 19 plots the errors vs. the accuracy of the predictions.

Figure 18: False positives vs. false negatives for complex network II

Figure 19: Accuracy of each algorithm for complex network II

In order to get more insight, the accuracy of each algorithm for each of the networks are presented in Table 2 in which N1, N2, and N3 stand for Naive-1, Naive-2, and Naive-3 algorithms. As shown, DNN provides the best accuracy compared to the other algorithms. Also, WDNN works better than or equal to the other algorithms in the serial, tree, and complex II networks. In the distribution network, it provides almost as good results as DNN—just 1% difference; however, tuning WDNN for any coefficient would decrease this difference. A similar reason is behind the big gap in the Complex I network. Moreover, the column N3<N1 shows the number of samples in which one of Naive-3’s
predictions has fewer false positive and fewer false negative errors than at least one of the predictions of Naive-1 algorithm. Similarly, the last column shows the comparison for the Naive-3 and WDNN algorithms. Apparently, Naive-3 provides better prediction than Naive-1, since it has the benefit of approximating a true form of probability distribution. Indeed, the simulated data comes from a normal distribution and Naive-3 also approximates the demand with a normal distribution. Thus algorithm Naive-3 performs very well when compared to algorithm Naive-1 in which we do not know the true distribution of $I_{L_t} + IT_t$.

4.6 Result: Multi-Item Dependent Demand Multi-echelon Problem

In the real world, usually the demands of the items depends on the demands of the other items. Thus, this information might help to provide a more accurate stock-out prediction. To analyze this, we generated the data of seven items such that their demand is dependent on each other, e.g. when the demand of one item is high, the demand of some other item is low, and vice versa. The demand pattern of seven items in seven days of a week is shown in Figure 20. The corresponding data is generated for the distribution system, shown in Figure 11.

- The demand of each item is dependent on the other items’ demand.
- The network is optimized with DNN.
- The average prediction accuracy for retailers is 95.78.

4.7 Multi-Period Prediction

In order to see how the algorithm works for multi-period prediction, we used the distribution network (Figure 11). The output of DNN algorithm is changed, such that there are $n \times q$ output values in the DNN algorithm, where $q$ is the number of prediction periods.

The algorithm ran for three problems. The first one predicts the stock-out in the next two days, the second and third do it for three days and seven days ahead. The accuracy of the prediction for each day for each of those are plotted in Figure 21. Also, the accuracy of one-day prediction is plotted as a reference. Increasing the number of prediction days results in lower accuracy for the latter days. Also, it causes a lower accuracy for the first days, such that in the seven-days-ahead prediction, the accuracy of the first day is lowered. Thus, it seems that developing $q$ separate DNNs for getting the
prediction of \( q \) days could be a better idea than having the stock-out prediction of the next \( q \) days in one network.

Figure 21: The average accuracy of the seven-days in multi-period prediction

| Days | Average Accuracy |
|------|-----------------|
| 1    | 0.86            |
| 2    | 0.88            |
| 3    | 0.90            |
| 4    | 0.92            |
| 5    | 0.94            |
| 6    | 0.96            |
| 7    | 0.98            |

5 Conclusion and Future Works

In this chapter, the stock-out prediction of multi-echelon networks is considered. In single-node networks, a known probability distribution of demand provides the stock-out prediction. However, there is no algorithm to predict stock-out in multi-echelon networks.

To address this need, an algorithm based on deep learning is proposed. Also, three naive algorithms are introduced to provide a baseline for stock-out prediction. All of the DNN and naive algorithms do not require knowledge of the demand probability distribution and use only historical data. Extensive numerical experiments show that the DNN works well compared to the three naive algorithms. The preliminary results suggest that our method holds significant promise for predicting stock-outs in complex, multi-echelon supply chains, and several research directions are now evident, including:

- Expand the current model to handle other types of uncertainty, e.g. lead times, disruptions, etc.
- Test the effectiveness of various state variables and expand it to include many more types of state variables. These state variables will fall into several categories; for example, variables related to inventory (such as inventory levels, on-order items, backorders), those related to the movement of products (observed lead times, transportation costs, delays), and those related to exogenous factors (economics, weather, exchange rates).
- Train the model to make predictions more than one period ahead. Our current model appears to be able to make predictions accurately up to roughly 3 periods ahead, but its accuracy degrades quickly after that. We will investigate why this is happening and modify the algorithm to improve the results. A similar approach will be followed for correlated demands.
- Extend this approach into an optimization tool so that ML can be used to choose inventory levels, not just predict stockouts, in multi-echelon systems.

Also, in order to test the efficacy of our approach, simulation can be a powerful source of data, and we will exploit it heavily to test our models. To the extent possible, we will also test our models on real supply chain data, if there are any available.

References

Sundar Bharadwaj, Thomas W Gruen, and Daniel S Corsten. Retail Out of Stocks: A Worldwide Examination of Extent, Causes, and Consumer Responses. 2002.
Alex J Cannon. Quantile regression neural networks: Implementation in R and application to precipitation downscaling. *Computers & geosciences*, 37(9):1277–1284, 2011.

Andrew J Clark and Herbert Scarf. Optimal policies for a multi-echelon inventory problem. *Management science*, 6(4):475–490, 1960.

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.

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.

Ayse Sena Eruguz, Evren Sahin, Zied Jemai, and Yves Dallery. A comprehensive survey of guaranteed-service models for multi-echelon inventory optimization. *International Journal of Production Economics*, 172:110–125, 2016.

Gartner Inc.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. [http://www.deeplearningbook.org](http://www.deeplearningbook.org)

Stephen C. Graves. Safety stocks in manufacturing systems. *Journal of Manufacturing and Operations Management*, 1:67–101, 1988.

Stephen C. Graves and Sean P. Willems. Optimizing strategic safety stock placement in supply chains. *Manufacturing and Service Operations Management*, 2(1):68–83, 2000.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093*, 2014.

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. 2010.

Llamasoft, Inc. Multi-echelon inventory optimization. [http://www.llamasoft.com/solutions/inventory-optimization/](http://www.llamasoft.com/solutions/inventory-optimization/), 2016.

Logility, Inc. Logility. [https://www.logility.com/](https://www.logility.com/) 2016.

Thomas L. Magnanti, Zuo-Jun Max Shen, Jia Shu, David Simchi-Levi, and Chung-Piaw Teo. Inventory placement in acyclic supply chain networks. *Operations Research Letters*, 34:228–238, 2006.

Afshin Oroojlooyjadid, Lawrence Snyder, and Martin Takáč. Applying deep learning to the news-vendor problem. *http://arxiv.org/abs/1607.02177*, 2016.

Xueheng Qiu, Le Zhang, Ye Ren, Ponnuthurai N Suganthan, and Gehan Amaratunga. Ensemble deep learning for regression and time series forecasting. In *IEEE Symposium Series on Computational Intelligence*, pages 21–26, 2014.

Kaj Rosling. Optimal inventory policies for assembly systems under random demands. *Operations Research*, 37(4):565–579, 1989.

SAP, Inc. SAP enterprise inventory and service-level optimization. [http://go.sap.com/product/scm/inventory-management.html](http://go.sap.com/product/scm/inventory-management.html) 2016.

David Simchi-Levi and Yao Zhao. Performance evaluation of stochastic multi-echelon inventory systems: A survey. *Advances in Operations Research*, 2012, 2011.

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.

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

Paul H. Zipkin. *Foundations of Inventory Management*. McGraw-Hill, Irwin, 2000.
A  Stock-out prediction for One Agent Network

Consider a one-agent network. The goal is obtaining the stock-out probability and as a result stock-out prediction, i.e. we want to obtain the probability:

\[ p(IL_t + IT_t < d) \]

With available knowledge of demand distribution \( d \) we can find the desired probability.

In order to predict whether the node will see a stock-out or not, a threshold \( \alpha \) should be determined. Similar to the approach, described in Section 3, one can determine it as

\[ \alpha = F^{-1}\left(\frac{c_p}{c_p + c_n}\right). \]

Thus, \( c_p \) and \( c_n \) costs provide the optimal value of \( \alpha \).

B  Gradient of Weighted Soft-max Function

Also, the gradient of the weighted softmax function is shown in

- Let

\[ p_j = \frac{e^{z_j}}{\sum_{u=1}^{U} e^{z_u}}. \]

- Then gradient of Soft-Max loss is:

\[ \frac{\partial E}{\partial z_j} = p_j - y_j \]

- And gradient of weighted Soft-Max is:

\[ \frac{\partial E_w}{\partial z_j} = w_j(p_j - y_j) \]

C  Activation and Loss Functions

The most common loss functions are Hinge Loss function (7), Logistic Loss function (8), and Euclidean Loss function (9).

\[ E = \max(0, 1 - y_i \hat{y}_i) \]  \hspace{1cm} (7)

\[ E = \log(1 + e^{y_i \hat{y}_i}) \]  \hspace{1cm} (8)

\[ E = ||y_i - \hat{y}_i||^2_2 \]  \hspace{1cm} (9)

which \( y_i \) is the observed value of the sample \( i \), and \( \hat{y}_i \) is the output of the deep learning. The Hinge loss function is appropriate for 0, 1 classification. Similarly, Logistic loss function does 0, 1 classification; however, it is a convex function which is easier to optimize. The Euclidean loss function minimizes the difference between the observed and calculated values and penalizes the closer predictions much less than far predictions.

Also, each node of the network has a activation function which the most commonly used activation functions are:

- Sigmoid \((z) = \frac{1}{1+e^{-z}}\)
- Tanh \((z) = \frac{2e^z-1}{2e^z+1}\)
- InnerProduct \((z) = z\)