Data-Driven Inventory Management Project

Wei Angel Huang

November 16, 2020

Abstract Inventory management is a classic problem in industry that is essential for the profitability of a company. In this project, I applied real-life data in a classic inventory management problem, and implemented four different methods to optimize inventory decisions. The first two methods were based on linear programming. I found optimal ordering policy to minimize total inventory while limiting the fraction of lost sales. The last two methods were based on neural network, which were used to predict the sales in the next period in order to select the optimal order quantity. Then I compared the performance on testing data set between the four different methods and evaluated the pros and cons of each method. The result showed that using Neural Network, we can achieve a lower inventory while limiting the fraction of lost sales.

The goal of this project was to apply data-driven decision-making tools to optimize inventory management. There are many choices of optimization function, such as maximize revenue, minimize cost, minimize probably of sale loss, etc. Here, we chose to minimize the total inventory (optimizing criteria), while limiting the fraction of lost sales to a fixed number (satisficing criteria).

1. Data

The sales data was extracted from a Kaggle dataset (https://www.kaggle.com/kyanyoga/sample-sales-data) that included longitudinal sales data for different models of toys. For simplicity, I extracted the "classic car" from all the toy types, since this category had the most abundant data. The methods used in this report can be easily adapted to other categories. The original data was organized in a way that every row is a transaction, and transactions occurred on one to two days per week on average. Within a day, there could be multiple transactions. To further simplify the problem, I chose month as the unit of analysis, and aggregated the total number of sales within each month. To gather enough data, I also ignored sub-categories of different model numbers under "classic car" and assumed that there is one homogeneous product that we are managing the inventory for. Note that in reality we would want to build different prediction models for each sub-category so as to optimize inventory for each. The final aggregated data for analysis was shown in Appendix Table A1.

The monthly sales data was recorded from January 2003 to May 2005. The sales quantity fluctuated from 120 to 3879. This fluctuation was clearly visualized in Figure 1, with y axis showing the sales and x axis showing the month in each year. It is worth pointing out that there was a seasonal trend in the data with October and November showing the highest sales, which was three to four times higher than normal, probably due to the Christmas season.

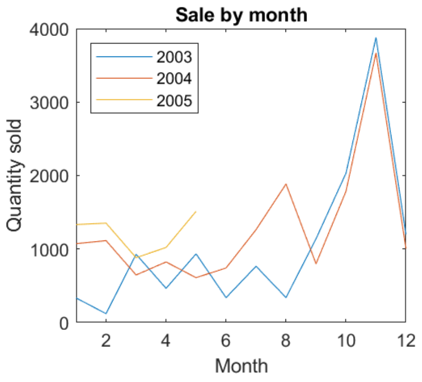


FIGURE 1. *Toy classic car sales data*

2. Linear Programming with all time periods

We started with a linear programming model to manage the inventory. At the beginning of each time period i ($1 \leq i \leq n$) (here each time period is a month, but this model can be easily generalized to other time units), we have X_i quantity of inventory on-hand. There will be D_i amount of demand during this period. Since we do not have data for the actual demand, we use the sales data to simulate the demand. We need to decide A_i, the amount to order at the end of period i, which arrives at the beginning of period $i + 1$. We employed a simple policy that if the inventory on hand falls below a certain threshold x, we will order A_i to make up the difference, otherwise we will not order.

$$A_i = \begin{cases} x - (X_i - D_i), & \text{if } X_i - D_i < x \\ 0, & \text{if } X_i - D_i \geq x \end{cases} \tag{1}$$

Thus, we have inventory X_{i+1} at the beginning of period $i + 1$:

$$X_{i+1} = max(X_i - D_i, 0) + A_i \tag{2}$$

Then the unsatisfied demand at the end of period i is:

$$L_i = max(D_i - X_i, 0) \tag{3}$$

The optimization target is the optimal order quantity x that will minimize the total inventory, i.e. our target function:

$$G = \sum_{i=1}^{n} X_i \tag{4}$$

At the same time, since unfulfilled demands are lost, we add a constraint that the fraction of lost sales should be less then a fixed number, here we use 0.05, i.e. we would like to have no more than 5% unfulfilled demands in total.

$$\sum_{i=1}^{n} L_i \leq .05(\sum_{i=1}^{n} D_i) \tag{5}$$

I simulated the sale process for each order policy with quantity x and showed the result in Figure 2. For each quantity x from 2000 to 4000, with increment of 1, the total inventory increased linearly (Figure 2, top), and the total unsatisfied demand decrease linearly until around 3600 and then flattened out at around 3900, by which time the unsatisfied demand is minimum, which is the first month's order 334, since we had 0 inventory on hand at the beginning (Figure 2, bottom). Based on the equation (5), we would only consider policy that has no more than 5% unsatisfied demand, which is below the red dotted horizontal line. This yielded available options of policy being $x \geq 3092$, i.e. the x-axis value for the intersect. Then we can consider minimizing equation (4), which is a positively linearly correlated to x (Figure 2, top). Thus the optimal policy threshold is $x = 3092$, the minimal total inventory is 91580, and the total unsatisfied demand is 1698 (< .05*total demand = 1699.6).

3. Linear programming with limited time periods

In the last section, we assumed that we knew data in all the time periods and x was chosen based on all the data. It is clearly not the case in reality. To come up with a more realistic policy that can guide our decision in real time, we upgraded our policy to only take into account data in a fixed time window m just before the decision period, we call this the "m-policy".

To be more specific, for each m parameter, we considered data from previous m time periods (periods $k - m + 1,\ k - m + 2,\ ...,\ k,\ k \geq m$) as training data, here, we chose up to 15 months ($2 \leq m \leq 15$). This is equivalent to when the total time periods $n = m$, then we found an x using the optimization procedure as in Section 2. Then we considered data in the next time period k as the test data to evaluate the policy. We compared average resulted inventory in month k per month for each m, and determined the best m and its corresponding x to be the optimal policy. The selection of x was similar as in Section 2 in that it minimized average inventory per month (equation (6)) while kept monthly unsatisfied demand under 5% (< .05*monthly

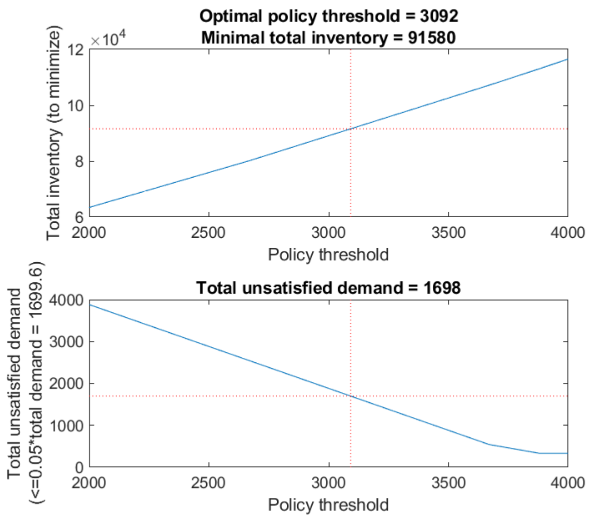


FIGURE 2. *Optimal policy to minimize inventory*

demand) (equation (7)).

$$\min_x(G) = \frac{1}{m}\sum_{i=k-m+1}^{k} X_i \tag{6}$$

$$\frac{1}{m}\sum_{i=k-m+1}^{k} L_i \leq \frac{.05}{m}\sum_{i=k-m+1}^{k} D_i \tag{7}$$

Figure 3 showed the optimal policy under each m parameter. The top plot showed that when we chose data from m time periods to guide ordering decision, the minimum average monthly inventory we can achieve. The inventory stayed at a relatively high level around 3100, with $m = 10$ months resulting in the highest average inventory. However, when using data from 11 previous months' for decision-making, we found a dip in average inventory to around 2679, which was also the minimum average inventory we can achieve given the parameter space we were searching from. This makes sense intuitively because we have found a seasonal trend in Figure 1, so having data up to almost a year was very helpful in guiding the inventory decisions. Interestingly, as we included data from more months ($m > 11$), the average inventory started to go up again, suggesting too big a time window is not beneficial (for example, it might start to introduce more noise).

The next constraint we care about was the unsatisfied demand. Since we had applied this constraint to get each x, we knew that each x resulted in less than 5% unsatisfied demand for each given m, the exact demand amount was different due to the difference in testing data set for each m (since there were different number of

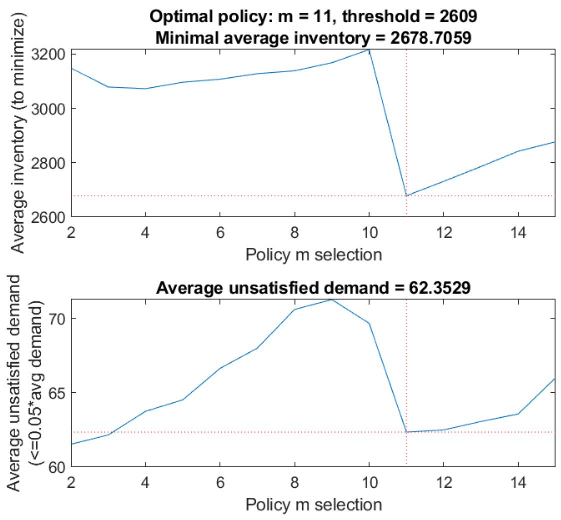


FIGURE 3. *Optimal m-policy to minimize inventory*

months used in training set). So we also plotted average unsatisfied demand against the parameter m in the bottom plot of Figure 3. It was clear that $m = 11$ also yielded one of the lowest average unsatisfied demand, besides $m = 3$ and $m = 4$. In summary, it was reassuring that when using data from previous $m = 11$ months, we were able to minimize average inventory, as well as keeping the average unsatisfied demand below 5%.

4. Neural Network

Before, we have been using one threshold for every test month, but it might make more sense to change how much we will order from month to month, especially given the seasonal trend we saw in Figure 1. Next, we were going to use a very different method from the above two – neural network – to predict the inventory in order to guide the ordering/inventory decision month by month. Since we have limited data, to keep things simple, we built a neural network with one hidden layer of $M = 3$ neurons.

Let $\mathbf{r}^{(n)} = \begin{bmatrix} r_1^{(n)} & r_2^{(n)} & \dots & r_m^{(n)} \end{bmatrix}^T$ denote the input layer, i.e. the vector of sales data from the previous m months in the nth training set. We built a neural network with one hidden layer $\mathbf{h}^{(n)}$ and one output layer $y^{(n)}$, described by the following set of equations:

$$\mathbf{h}^{(n)} = \phi(\mathbf{W}^{in}\mathbf{r}^{(n)} + \mathbf{b}^{in}), \qquad [\mathbf{W}^{in} : M \times m], \tag{8}$$

$$y^{(n)} = \mathbf{W}^{out}\mathbf{h}^{(n)} + \mathbf{b}^{out}, \qquad [\mathbf{W}^{out} : 1 \times M], \tag{9}$$

where:

$\mathbf{h}^{(n)}$	= M-dimensional activation of the hidden layer of the network
ϕ	= activation function
$\mathbf{W}^{in}$	= weight matrix that connects input layer to hidden layer
$\mathbf{b}^{in}$	= bias term that connects input layer to hidden layer
$\mathbf{W}^{out}$	= weight matrix that connects hidden layer to output layer
$\mathbf{b}^{out}$	= bias term that connects hidden layer to output layer
$y^{(n)}$	= scalar output of the network: the next month's order quantity

Using a non-linear activation function ϕ will ensure that the hidden layer performs a non-linear transformation of the input, which is the key to enable the mapping of non-linear relationship between input and output. A commonly used non-linear activation function is the linear rectification function (i.e. ReLU, standing for "Rectified Linear Units"):

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{else} \end{cases} \tag{10}$$

A commonly used target function for neural network is to minimize mean squared error (MSE) (i.e. "Loss" in equation (11)), which was computed from targets $y_1, y_2, \ldots, y_P$ and corresponding network predictions $\tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_P$.

$$Loss = \frac{1}{P}\sum_{n=1}^{P}\left(y^{(n)} - \tilde{y}^{(n)}\right)^2 \tag{11}$$

where P denotes the total number of data samples in the training set ($P = n - m$).

Here, we trained the neural network model using the gradient descent algorithm, which was carried out in the following three steps iteratively:

1. Evaluate the loss on the training data. For a mean squared error loss, this was given by equation (11).

2. Compute the gradient of the loss with respect to each of the network weights, such as in our case:

$$\frac{\partial L}{\partial \mathbf{W}^{in}}, \frac{\partial L}{\partial \mathbf{b}^{in}}, \frac{\partial L}{\partial \mathbf{W}^{out}}, \frac{\partial L}{\partial \mathbf{b}^{out}}$$

3. Update the network weights by descending the gradient that was calculated in

step 2:

$$\begin{aligned}
\mathbf{W}^{in} &\leftarrow \mathbf{W}^{in} - \alpha\frac{\partial L}{\partial \mathbf{W}^{in}} \\
\mathbf{b}^{in} &\leftarrow \mathbf{b}^{in} - \alpha\frac{\partial L}{\partial \mathbf{b}^{in}} \\
\mathbf{W}^{out} &\leftarrow \mathbf{W}^{out} - \alpha\frac{\partial L}{\partial \mathbf{W}^{out}} \\
\mathbf{b}^{out} &\leftarrow \mathbf{b}^{out} - \alpha\frac{\partial L}{\partial \mathbf{b}^{out}}
\end{aligned}$$

where α is the learning rate, which is a hyperparameter that can be tuned to adjust how big a step the algorithm take with respect to the gradient.

We used a standard ratio 60:20:20 to randomly split data into training:validation:test set, which means that we trained the model and tuned hyperparameter on 80% of the data and test the model on the rest 20% of data. This process was repeated for 100 times to obtain a stable measurement of the test performance. The result from test set was shown in Figure 4 in the format of mean ± standard deviation. To minimize mean square error of the demand prediction, the optimal policy is $m = 12$ months, which results in a minimal monthly error of 1034 (top panel) and an average monthly inventory of 1259 (middle panel), which was less than half of what we got from the previous methods. However, this low inventory also costed a high average proportion of unsatisfied demand, which was around 26%, much higher than our previous 5% threshold (bottom panel). In fact, all of the m selection yielded similar level of percentage of unsatisfied demand (bottom panel). This was not surprising since our target function was set to minimize absolute error, regardless of the sign of the error.

5. Neural Network with Lost-Demand Penalty

In the previous neural network model, we were able to achieve a low average inventory level but had no control over the lost sales. In reality, we might have preference on overstocking versus understocking. Here, in our example, we assume that we cared more about lost demand and set a 5% threshold.

To incorporate the constraint of this limited unsatisfied demand, one way is to make modifications to the target function. Since we would like to "punish" the model if it underestimates the demand, i.e. incurs unsatisfied demand, we added a penalty of cost q per unit of unsatisfied demand on top of MSE (equation (12)). Here q is another hyperparameter that we needed to tune. By increasing q, i.e. the cost of unsatisfied demand, we can bias the prediction towards more over-estimation, at the cost of not as accurately predicting the true demand. Then we picked the q value that resulted in around 5% unsatisfied demand.

$$Loss = \frac{1}{P}\sum_{n=1}^{P}\left[\left(y^{(n)} - \tilde{y}^{(n)}\right)^2 + q * \left(y^{(n)} - \tilde{y}^{(n)}\right)^+\right] \tag{12}$$

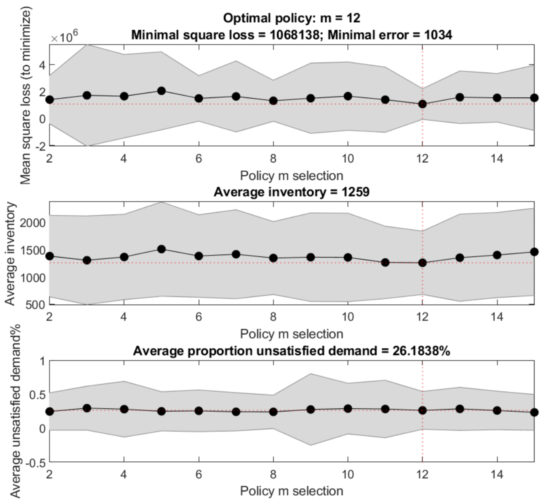


FIGURE 4. *Inventory prediction using neural network*

The training of Neural Network was implemented using the Deep Learning Toolbox in Matlab (Appendix Figure A1 showed an example GUI during training when $m = 12$ months). The top panel shows the structure of the neural network, which consist of $m = 12$ input neurons, and $M = 3$ neurons in hidden layer, and 1 neuron in output layer. ReLU was chosen as the activation function in the hidden layer. In the second panel, it shows the performance measurement, which is usually Mean Squared Error, however, here we added a cost of lost demand, via a customized function (mymse.m) and the corresponding customized functions for performing the gradient descent. The program normally took less than 6 validation checks and less than 1 second for each iteration to converge, which made sense given our small dataset.

The performance of this new neural network model with lost-demand penalty was shown in Figure 5. The bottom panel illustrated the average proportion of lost demand for each m selection. The solid red line denoted 5% unsatisfied demand, we only consider the m that yielded unsatisfied demand that is lower than this threshold. Within this subset of m, we identified that $m = 2$ gave lowest mean square loss (second panel from the top), and an average inventory 2505 (third panel from the top). This implied that we did not need too much data to predict next month's demand, especially given the constraint of unsatisfied demand. The top panel showed the loss function we were using to train the model (MSE + lost-demand cost). An interesting pattern we saw was that as m increased, the standard deviation (shade) of the data also increased for most of the panels. It was probably due to the fact that the greater the m, the smaller the size of training and test set to train and test the model, respectively. Here the tuned hyperparemeter "lost-demand cost" was $q = 10000$. Higher q normally

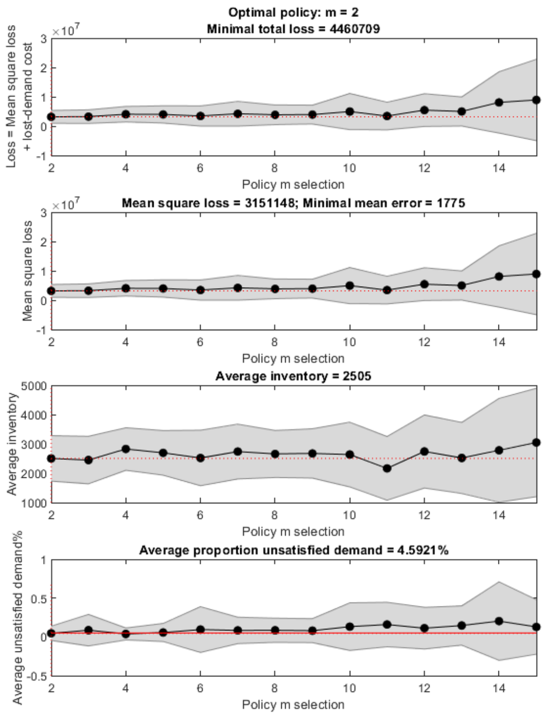


FIGURE 5. *Inventory prediction using neural network with lost-demand penalty (activation = ReLU)*

yielded lower unsatisfied demand and greater MSE, however, increasing q further from this point did not change too much of the percent of unsatisfied demand without yielding a much higher average inventory (than the linear programming solution). It was also worth pointing out that if we ignore the unsatisfied demand constraint, we can see in the first 3 panels, $m = 11$ gave the best (lowest) inventory, which was consistent with our m-policy result.

I also tried a different commonly used activation function – sigmoid function (Figure 6, which yielded an optimal $m = 5$, with a higher average inventory 2658, which was not much an improvement from the m-policy result.

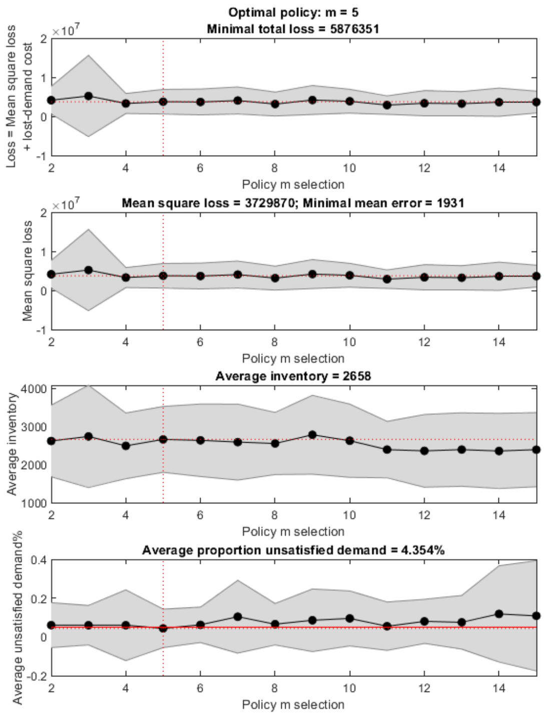


FIGURE 6. *Inventory prediction using neural network with lost-demand penalty (activation = Sigmoid)*

5. Conclusion

In this project, we implemented two methods based on linear programming in order to find the optimal inventory policy, and two methods based on neural network to predict demand in order to guide inventory decision. Now, let us compare the four methods from three angles: assumption, application, and performance.

First, the assumption and application of the methods were different. The first method assumes that we have all the data thus we can choose the optimal policy (reorder threshold) to minimize the overall inventory as well as keeping the overall unsatisfied demand below 5%. Then we came up with a more sensible policy that better resembles real life scenario and more informative. This also requires less data since it only uses previous *m* time periods. The neural network models we used were very simple with only one hidden layer, because it normally requires a large amount of data for it to show its benefit over other predicting methods.

Second, the applications of the methods were different. The first method requires a long history of data. Even though in this data sample, we only have 29 months of data, in real life, we might need to make decision based on daily data that last for years. In addition, we might have other variables to take into consideration, or other response variables that we need to predict (eg. we might need to build a separate model for each toy type/model). The cost of data storage and computation can quickly add up. Further more, even given the unlimited computational power, the evaluation of the performance of the first methods is biased because we made the decision based on the data we already have and assume that future data will look the same. To summarize in machine learning terms, the first method can only give us training accuracy, whereas the other three methods can give us test accuracy that can generalize to future unseen data.

Third, the results and performance of the methods were different (Table 1). To make a fair comparison, we converted the results of the first method from total measures into monthly measures. The optimal solution of this method corresponds to a reorder threshold of 3092, a minimum total inventory of 91580, i.e. average monthly inventory of 3157.93 (= 91580/29 months), and an unsatisfied demand of 5%, i.e. an average monthly unsatisfied demand of 58.55 (= 1698/29 months). In the m-policy solution, the optimal policy was to use previous 11 months' data, a reorder threshold of 2609, to achieve an average monthly inventory of 2678.71, and an average unsatisfied demand of 62.35. Overall, the m-policy yielded a lower monthly inventory but a little higher average unsatisfied demand, although it still meets the requirement of less than 5% unsatisfied demand. In the neural network model with MSE, we were able to achieve very low average inventory but with no contraint on unsatisfied demand, which was an unfair comparison to other methods. In the last model with MSE and Lost-Demand Cost model, we were able to achieve a lower monthly inventory as well as within the threshold unsatisfied demand (on average lower than 5%). The Neural Network showed an optimal *m* being 12, which is close to the m-policy result 11. Even though for the Neural Network with MES and Lost-Demand Cost Model, the optimal $m = 2$ was very different from the above methods, these was mainly due to the fact that the unsatisfied demand threshold limited the subset of m one can choose from. As we can see from Figure 5 and Figure 6, if we loose the unsatisfied-demand constraint, $m = 11$ was still one of the optimal solutions for minimal inventory. This type convergence between completely different methods was very encouraging to see.

TABLE 1. *Model selection and performance*

Model	Average monthly inventory	Average monthly unsatisfied demand	Reorder threshold	Optimal m
Linear Programming	3157.93	0.05	3092	-
Linear Programming with M-policy	2678.71	0.05	2609	11
NN with MSE	1259	0.26	-	12
NN with MSE + Lost-Demand Cost	2505	0.046	-	2

It should be noted that there are many more methods and policies that can be used to solve this problem that we did not touch upon due to the scope of this project. For example, one can choose a different target function or a combination of target functions (eg. assign value to the cost of inventory and unsatisfied demand, and then combine the cost of inventory and unsatisfied demand so that there is a single target cost function). Instead of weighing all m previous months equally, one can choose different weights for the previous months (eg. weight recent months more heavily) when trying to predict the inventory. Further, one can also use regression or other time series forecasting method to incorporate seasonality and get better prediction of the future inventory and adjust the threshold accordingly (although including more parameters will require more data).

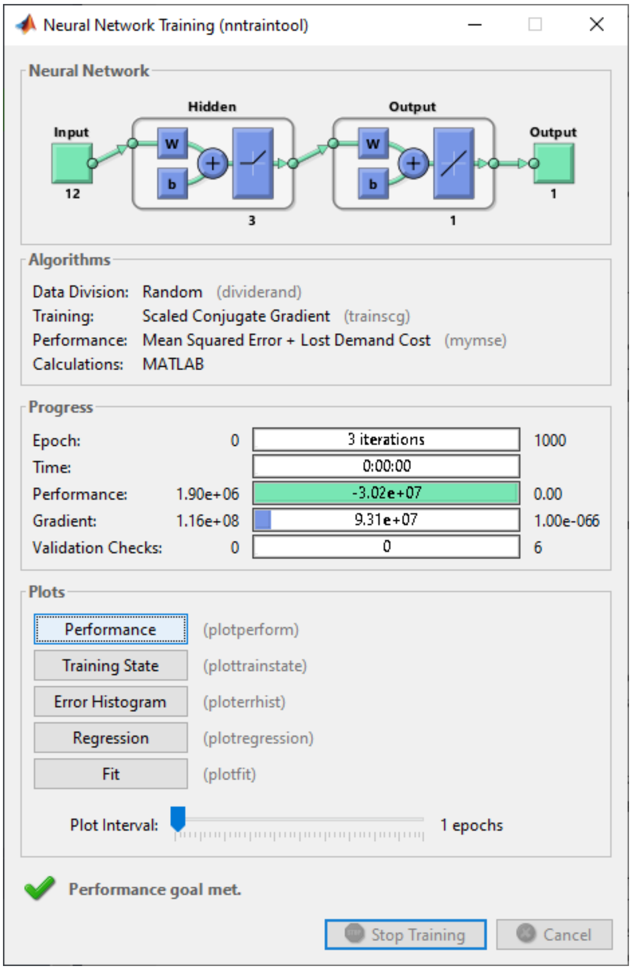
Overall, we used linear programming and neural network to solve an inventory management problem and found optimal solution among the methods we implemented. Even with a very small dataset, we observed that the Neural Network performed better than m-policy using linear programming, probably due to its flexible inventory that changed on a monthly basis. In addition, once trained, the neural network only needs 2 previous months' data to yield a decent performance, which made it a very useful tool in industry.

Appendix

TABLE A1. *Sales data for toy classic car.*

Year	*Month*	*Sale*
2003	1	334
2003	2	120
2003	3	929
2003	4	465
2003	5	934
2003	6	338
2003	7	765
2003	8	339
2003	9	1139
2003	10	2032
2003	11	3879
2003	12	1199
2004	1	1073
2004	2	1114
2004	3	647
2004	4	824
2004	5	610
2004	6	742
2004	7	1263
2004	8	1885
2004	9	799
2004	10	1786
2004	11	3669
2004	12	1006
2005	1	1333
2005	2	1352
2005	3	882
2005	4	1021
2005	5	1513

Notes: Data from https://www.kaggle.com/kyanyoga/sample-sales-data



Data Division: Random (dividerand)
Training: Scaled Conjugate Gradient (trainscg)
Performance: Mean Squared Error + Lost Demand Cost (mymse)
Calculations: MATLAB

```
Epoch:                 0   3 iterations   1000
Time:                      0:00:00
Performance:    1.90e+06   -3.02e+07      0.00
Gradient:       1.16e+08   9.31e+07       1.00e-066
Validation Checks:     0   0              6
```

Performance (plotperform)
Training State (plottrainstate)
Error Histogram (ploterrhist)
Regression (plotregression)
Fit (plotfit)

Plot Interval: 1 epochs

Performance goal met.

FIGURE A1. *Matlab nntraintool with customized target function (Deep Learning Toolbox)*