

Automatic Inventory Control: A Neural Network Approach

Nicholas Hall

ECE 539
12/18/2003

TABLE OF CONTENTS

| | |
|--------------------|----|
| INTRODUCTION | 3 |
| CHALLENGES | 4 |
| APPROACH..... | 6 |
| EXAMPLES | 11 |
| EXPERIMENTS | 13 |
| RESULTS..... | 15 |
| CONCLUSION..... | 17 |
| REFERENCES..... | 18 |

INTRODUCTION

Managing an inventory is one of the biggest problems that many businesses face. Whether they manufacture goods, distribute, or resell them, any business that has a physical inventory eventually runs into these problems:

- How many of each part should be ordered
- Determining when each part should be ordered
- How to estimate when demand will go up for a part
- How to estimate when demand will go down for a part
- How to predict when the entire business' sales will go up or down

Many of these factors are related, but in essence they can be summarized with the rule:

Parts should arrive just before a customer orders them, but no sooner.

Obviously, this is a very idealized goal, and almost impossible to realize, but many businesses, such as Walmart and Amazon.com, spend millions of dollars each year to try to achieve as close as possible to this.

In reality, it is better to accept that it is impossible to predict customer demand 100% and therefore it will be impossible to meet this goal. Backorders can be prevented by filling a warehouse with as many parts as you predict will be sold in 10 years, but then the cost of the inventory would be incredibly high. Inventory cost can be kept down by moving to a Just-In-Time inventory solution, where products are ordered or manufactured immediately after a customer orders them, but this delays every order and leads to dissatisfaction among customers. Therefore, some compromise must be made between the total number of backorders allowed and the average time a particular product is kept in-stock. If this average time a product is kept in-stock is measured in months, then 12 divided by this time measures how many times the warehouse sells all of its products in a year, on average, and is called the "turnover time".

CHALLENGES

BACKGROUND INFORMATION:

This project is being done using inventory data from Manufacturer's Supply. Manufacturer's Supply is a mail order and Internet business that sells snowmobile, lawnmower, go-kart, and other small engine parts. With a large warehouse and over 30,000 products available for customers to order, it has been a challenge in the past to keep enough products in stock in order to minimize backorders, while at the same time minimizing the average time a product is kept in stock at the warehouse. In the past, inventory management was done almost entirely by a few people who worked on the shipping floor, knew inventory levels and knew how products were moving. As the volume of orders grew, it became increasingly difficult for these people to be aware of current inventory levels and what everyone was ordering. Tools were created in order to help these people quickly visualize current inventory levels and order history for particular parts, which greatly helped them keep up with the trends and understand the inventory. Unfortunately this system still depends on a large human component, and if someone new comes in it would take a long time before they could understand the inventory at the same level. Also, as the volume of sales continues to increase, it will be difficult to fully understand the inventory even with advanced visualization tools. Ideally, a computer program could be developed to replace these human experts and automate the entire parts ordering process. This program would save people time, and would probably increase the accuracy as sales volume increases.

PROBLEMS:

- Manufacturer's Supply's product lines are high seasonal. Several hundreds of certain parts will be sold each month during the winter season, but none will be sold during the summer. Likewise, the weather influences product sales a great deal. In a winter with heavy snowfall they might sell several times the number of snowmobile products than in a winter with little snow.
- Only a few of a certain product may sell over a period of several years, making stocking of that product impractical. If in one month someone buys many of these parts unexpectedly, the person ordering parts will realize that their order was one-time and not stock up on these parts that rarely sell. However, it is difficult for a computer program to understand this so the feature set must be designed to differentiate between this unusual, non-recurring "spike", and the normal selling pattern for this product.

- Product lead times, the time it takes for a product to arrive after it is ordered from a vendor, are not very well defined. Some vendors ship parts so they arrive the very next day, but some vendors require several months to manufacture parts after they are ordered. Likewise, some vendors will sometimes respond with parts in only a matter of days after they are ordered but at other times require over a month to fulfill the order.
- Although several years of inventory sales data are available for every product, it is only broken down in sales per month.

APPROACH

To predict how to order each product in the inventory, a program and a library were developed to use the multilayer perceptron algorithm (MLP). Because of the large amount of data, the amount of time required to analyze this data, and the likelihood of needing additional features such as the ability to save a network's state and restart where it left off, it was decided not to use a Matlab approach. Instead, a MLP library was developed from scratch that allows for all the features that this program needs. This library was designed to be generic and not just for this specific problem, and is fully object-based, with the following classes:

- Example – An Example object should be created for each set of inputs and outputs.
- Attribute – By specifying an Attribute each input and output can be uniquely named to ease in debugging and data analysis.
- Node – Each input, hidden, and output node is the same type of object, which makes it easy to link the different layers together.

A program called “predictnet” was created to use this MLP library to create the MLP network, train and test on a number of products. The following were the original input feature vectors:

- 12 input features for the number for that product which was sold in the 12 months prior to the one we are training for.
- 12 input features for the number for that product which was sold in the 12 months prior to those, so it is training on 2 years of inventory data.

The above inputs will be hereafter referred to as the “**standard inputs**”. It was hoped that this inventory data would be enough for the MLP to be able to converge to a reasonable value. However, the results were not as good as hoped for. After analyzing the data, the input features were changed to 27, which will hereafter be referred to as the “**expert inputs**”:

- 12 input features for the number for that product which was sold in the 12 months prior to the one we are training for, which is the same as was used in the standard inputs.
- Instead of simply using the prior year's inventory data as with the standard inputs, 12 input features were created that use the quantity for each product sold for one month in the previous year divided by the quantity sold for that product for the same month of the year before that. This was done because this ratio is very important in predicting future sales. If the number sold has doubled from the prior

year for every month so far, it is likely that the next month will approximately double as well.

- Recognizing the fact that many of the product sales are seasonal, it is apparent that other factors that influence purchasing decisions for customers, such as the weather, are also seasonal. Although there are many exceptions, in a general way, if this winter has been snowy so far, it is likely to remain snowy all season. A dry summer is likely to continue to be dry, as weather trends stretch over periods of several months. Since the weather has influenced product purchases so far this season, it is likely to continue to influence purchases in the same way for the rest of the season. Therefore, it is not strictly necessary to use the weather data as an input as long as the inventory data for this season is available. The year was broken down into 3 seasons, each 4 months long, with the 4 months prior to the month we're training for designated "Season 3", the 4 months before that "Season 2", and the 4 months before that "Season 1". Three input nodes were created using this data, each a ratio between two of the seasons: Season 3/Season 2, Season 3/Season 1, Season 2/Season 1. The four month long seasons were used instead of more conventional three month seasons because it required only 3 ratios instead of the 6 ratios required with three month seasons.
- When someone buys a large quantity of parts outside the normal buying trend, it is assumed they will be backordered, and that is fine. However, it is ideal if this unusual "spike" does not influence purchasing the next month. Since the previous month is usually the factor of greatest influence, this input feature divides the quantity sold for that month by the number sold all year. If a product's sales are roughly constant, this input will be approximately 1/12. However, if this is just an unusual spike, this input will approach 1.

These inputs will all be scaled between -5 and 5 , and centered around 0 , which will optimize the speed and ability for the MLP to learn.

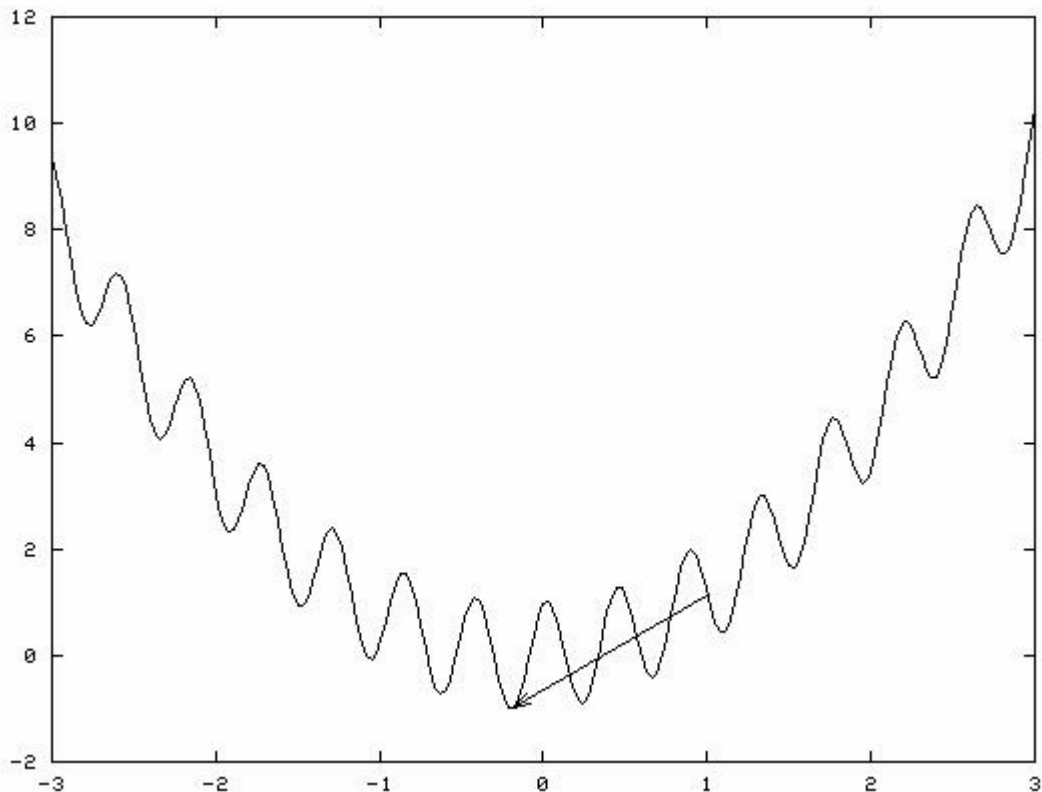
Since it is not possible to represent every influential factor as an input to this neural network, it is assumed that it will not operate with 100% accuracy on the testing data. Even if it is only slightly off, it is likely to underestimate on about half the products and overestimate on about half, causing a large number of backorders with a very high turnover rate. Since it would be ideal to minimize backorders but still keep the turnover rate around 3 months, there is one output node which will represent how many products will be sold over the next 2 months. By ordering every month enough so that the inventory gets up to this level, it provides a cushion by always ordering so there will be enough in stock for the next 2 months.

SIMULATION:

After the network has been trained on the training set, it is tested with the rest of the examples for that particular month. Although this is a useful statistic for comparing how well each network performs, that absolute error is not very important for this application because it is assumed that there will always be error. The business managing its inventory cares mostly about the percentage of products that are backordered and how many turns the inventory makes each year. In order to approximate these statistics, the predictnet program was given the capability to “simulate” the network. It does this by loading the part numbers for all the products in the inventory which were not in the training set, going back 13 months, and initializing a virtual inventory for all these products to 0. It then proceeds to predict how many will be sold in the next two months, orders the corresponding number, and at the end of the month, assumes they have arrived so it adds them to the current inventory level. After it predicts how many will be sold, it checks how many were actually sold that month and decreases the inventory by that amount. If the inventory ever goes below zero for a particular product it is assumed that a special order beyond the automatic monthly stocking order was needed to be placed and so the inventory is held at zero but a backorder counter keeps track of how many of each product is backordered, except for the first month since everything will be backordered when the inventory is initialized. This virtual inventory simulation provides both individual results for each product in the inventory, as well as total counts at the end of the simulation time period listing how many products were sold, how many products were ordered from suppliers, how many products were backordered, and an approximate number of turns the inventory made.

REFERENCE PROGRAM:

To determine whether the MLP approach can beat a much simpler approach, a benchmark program called predictweights was also developed which uses simulated annealing to tackle the problem in a very similar way to the expert input approach described above. Simulated annealing is a random walk method based that operates similarly to hill-climbing but modifies the process in an attempt to avoid getting stuck at local minimums. It is a generalization of the Monte Carlo method for examining the states of n-body systems, and employs a generalization of the Metropolis algorithm. By instead of always trying to go “downhill” in the search space, it tries to go downhill most of the time, and makes safer choices as it gets nearer to the bottom. It does this by starting off at an initial “temperature”, which it decreases as time goes on. The higher its temperature, the more likely it is to go uphill on a step. This allows it to move quickly downhill at high temperatures but step more slowly as it gets closer to the goal. It can be proved that if the temperature is lowered logarithmically, the algorithm will eventually converge to the global minimum, however, this is very slow so the algorithm usually approximates this with a fixed scaling factor in an attempt to get close to the global minimum.



Simulated Annealing Algorithm:

$$\text{Output} = \sum_{i=1}^{27} w_i I_i$$

where :

I_{1-12} = # sold each month of past year

$I_{13-23} = \frac{\text{\# sold each month of past year}}{\text{\# sold each month of year before that}}$

I_{24-26} = seasonal ratios

I_{27} = spike factor

while (temperature > stopping temperature)

new_weights = random_perturb(weights, temperature)

new_error_rate = 0

for (entire training set)

new_error_rate = new_error_rate + | predicted sold – actual sold |

if (new_error_rate < error_rate)

// If new weights work better, move to that state

weights = new_weights

error_rate = new_error_rate

else if ($e^{-\Delta E/\text{temperature}}$ <= rand_num_between_0_and_1)

// If new weights are worse, move to that state with some probability

weights = new_weights

error_rate = new_error_rate

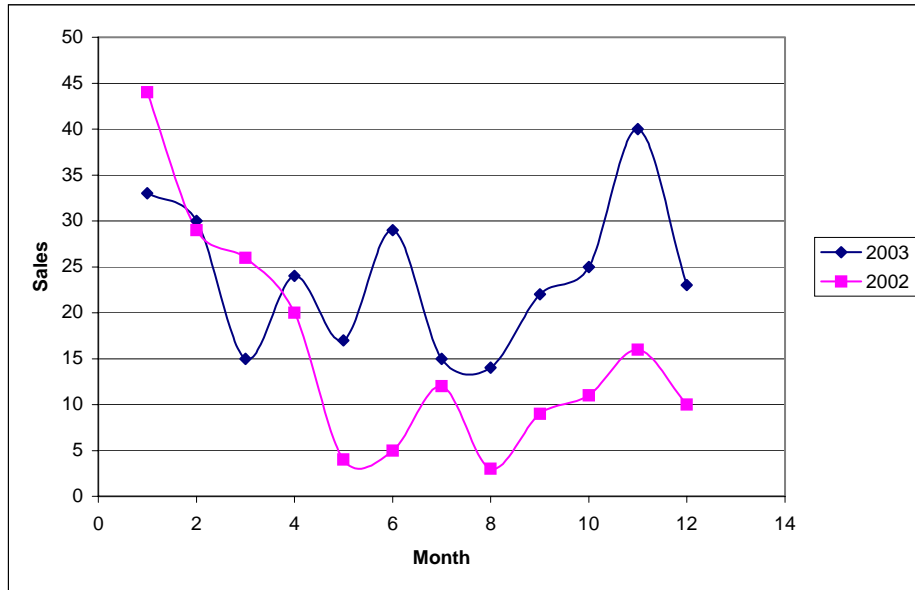
// Otherwise keep the weights as they are

*temperature = alpha*temperature*

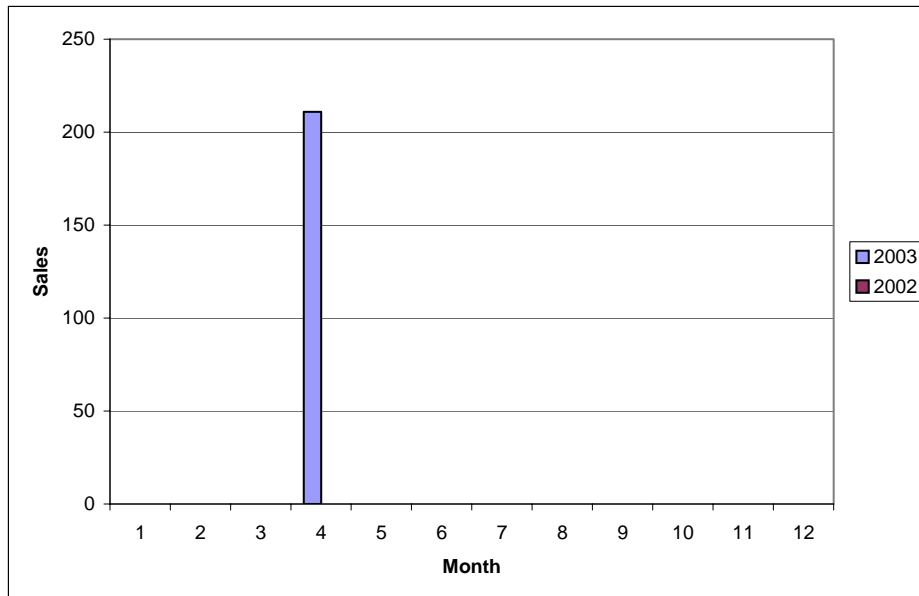
EXAMPLES

To aid the visualization of some of the inventory data, the following are some example graphs for individual products.

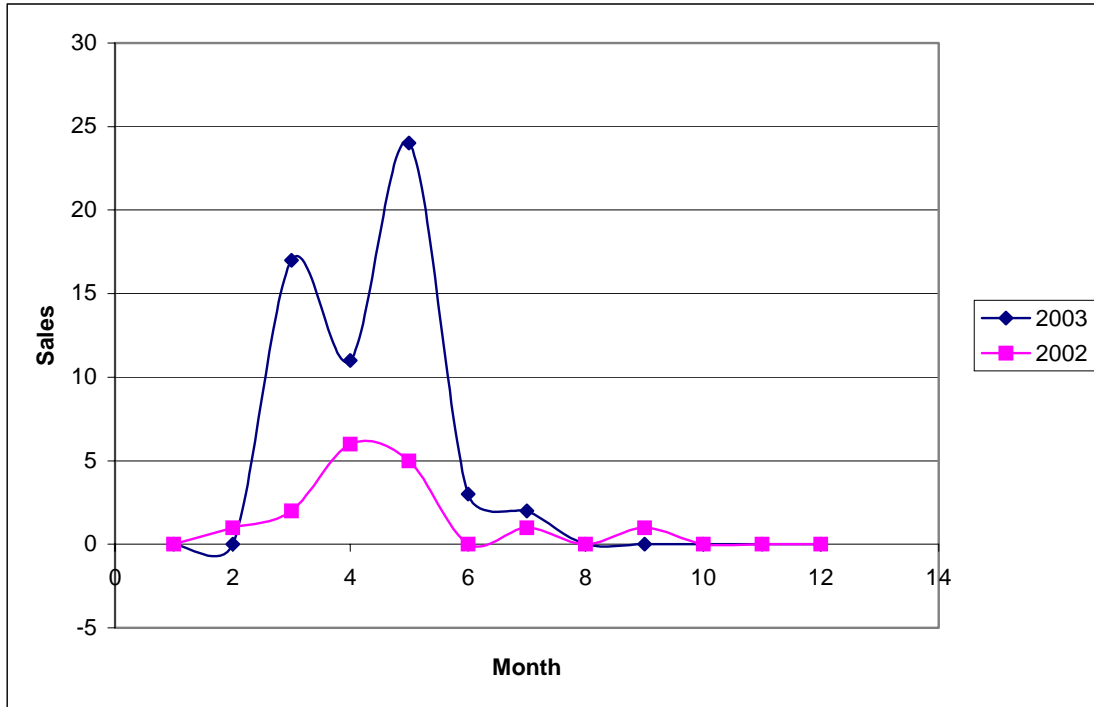
2003 reflects a very similar pattern to 2002: If it were scaled it would approximate properly:



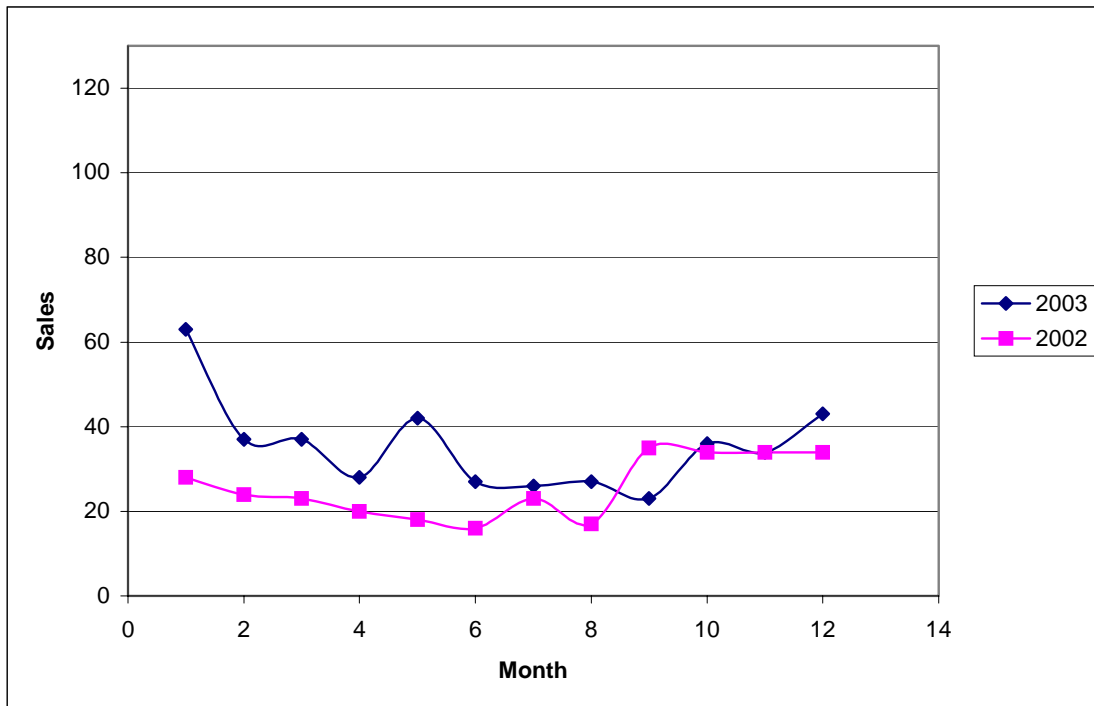
This part has a very large “spike” for one month, but was never sold any other month for the two-year period:



This graph shows a very seasonal nature for one part. Month 3 corresponds to February and month 5 corresponds to December. There are large volumes sold during this span but sales are almost non-existent during the rest of the year:



This graph shows how some products have fairly flat sales volume all year round:



EXPERIMENTS

Extensive trials were done to determine the proper parameters for the network. Different numbers of hidden nodes were used, alpha was changed, and the number of epochs used for training was varied. Most of the training was done using May 2002 as the month to predict for. During this month, about 3000 different part numbers were sold, so there are an abundance of examples to choose from for training.

The first set of trials used the original input vectors as described in the Approach section. Some of the more significant trials will be described. All trials are with a 13 month simulation period, as described in the simulation part of the Approach section. All of these trials used 100,000 epochs for training to try to achieve the best convergence.

Remaining in stock signifies the total quantity of products remaining in stock after the end of the simulation period, expressed as a percentage of the total quantity of products sold during that period. Ordered signifies the quantity of products ordered from suppliers expressed the same way. Backordered signifies the quantity of products which were ordered by customers but were not in-stock at the time, also expressed in the same way.

| # of Training Examples | # of Hidden Nodes | % Training per Epoch | Alpha | % Remaining In Stock | % Ordered | % Backordered | # Turns |
|------------------------|-------------------|----------------------|-------|----------------------|-----------|---------------|-------------|
| 100 | 100 | 50% | 0.1 | 17.1% | 117% | 63.7% | |
| 300 | 100 | 50% | 0.1 | 96.2% | 196% | 43.2% | 4.45 |
| 300 | 150 | 10% | 0.1 | 60.7% | 161% | 60.9% | 5.86 |

In order to see if the results would be better, the input features were changed to the “expert inputs” as described in the Approach section. Some of the trials follow:

| # of Training Examples | # of Hidden Nodes | % Training per Epoch | Alpha | % Remaining In Stock | % Ordered | % Backordered | # Turns |
|------------------------|-------------------|----------------------|-------|----------------------|-----------|---------------|-------------|
| 300 | 100 | 10% | 0.1 | 38.3% | 138% | 64.0% | 7.36 |
| 300 | 150 | 10% | 0.1 | 47.1% | 147% | 54.9% | 6.69 |
| 1500 | 150 | 1.5% | 0.1 | 56.9% | 159% | 85.95% | 6.08 |
| 1500 | 150 | 1.5% | * | 110% | 210% | 56.7% | 4.06 |

* indicates that this trial used an alpha that started at 1 at epoch 0 and decreased linearly as the current epoch increased, until it reached 0.01 at epoch 100,000

The much simpler simulated annealing program yielded some interesting results:

| Future Constant | % Remaining In Stock | % Backordered | # Turns |
|------------------------|-------------------------------------|--------------------------|----------------|
| 3 | 27.0% | 9.9% | 9.42 |
| 4 | 41.3% | 9.3% | 8.45 |

The future constant is used to scale the probability that the algorithm wants to buy a product. Some manual experimentation is required as the number of turns and the amount of backorders move oppositely to one another based on this factor, so the user must determine what trade-off he wants when he sets this factor's value.

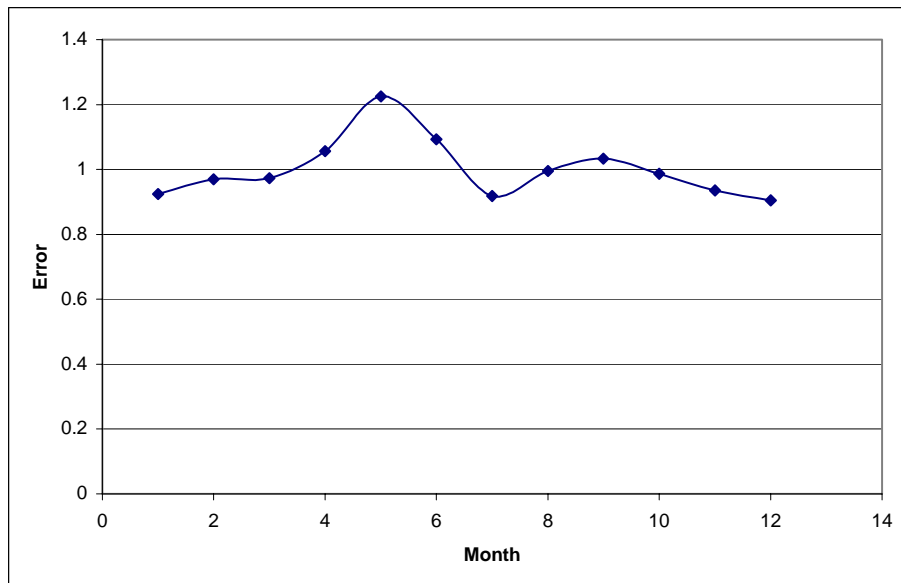
RESULTS

From the data, it is not possible to derive precisely what the actual backorder rate and number of turns that Manufacturer's Supply is currently running at using human inventory control. However, the backorder rate is currently estimated to be between 2% and 5%.

It is remarkable that the simpler simulated annealing approach beat the multi-layer perceptron algorithm in every case. With a backorder rate of a little over 9% it is still higher than the human expert's rate, but that is including the large "spikes" as backorders, which the human expert's backorder estimate may not include since they are usually seen by humans as outside the normal purchase pattern and therefore not conventional backorders. Since the simulated annealing program was developed first, it had more time to run, and since there were fewer parameters which needed experimenting with, each run could comfortably take longer. The final results from the simulated annealing program each used about a week's amount of CPU time on a AMD Athlon 1800+, which is quite a large amount of processing time.

The simulated annealing algorithm only trains on one month, but then tests over all the months of the previous year. By breaking down the testing results by month, one can see a slight change in error as the months get further from the current one, but they decrease again as the month approaches the month corresponding to one year before the training month. This seems to indicate there is some slight seasonal factor which is not being accounted for, but it is relatively small.

Month 1 is the training month. An error of 1 indicates the average error for the entire test run, so below or above that line indicates the percentage below or above average error for the trial:



It was surprising that the MLP algorithm fared quite a bit worse than the simple simulated annealing approach. The 43% backorder rate is not too bad when the complexity and unpredictability of the data is considered, but is still much worse than the simulated annealing approach. However, because this program was created later and there were many more parameters to experiment with, each trial needed to be designed to take less time to run. Most took around 8 hours of CPU time. With more time, it is likely that the algorithm would converge with better results. This also demonstrates one property of the MLP algorithm: backpropagation is slow. The simulated annealing algorithm is much simpler so in a given amount of CPU time it accomplishes more work. It is also easier with simulated annealing to ensure that it will converge to a point close to the minimum if enough time is given to it, which is more difficult to demonstrate with backpropagation.

CONCLUSION

While multi-layer perceptron neural networks are powerful, there are better matches to this inventory control problem, at least with the current implementation of multi-layer perceptrons. With more modifications it may be possible to increase the prediction rate to provide a much better simulation. By increasing the amount of time that the algorithm is allowed to run for it may be possible to obtain better results.

Even though the simulated annealing algorithm was developed simply as a benchmark program, and not as the serious attempt for inventory control, it appears to be worth more serious study. Even though it takes a very long time to run, its results indicate that it has high potential. By training over at least two months that are at different times in the year the slight season affects that are apparent should be further decreased, which would improve the results even more.

REFERENCES

S. Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice-Hall, 1999

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

K. Bansal, S. Vadhavkar, A. Gupta, "Neural Networks based Data Mining and Knowledge Discovery in Inventory Applications", <http://scanner-group.mit.edu/htdocs/DATAMINING/Papers/finalmicrosoft.html>

R.J. Tersine, *Principles of Inventory and Materials Management*, North Holland, 1982

R.G. Broeckelmann, *Inventory Classification Innovation*, St. Lucie Press, 1999

J.H. Fuchs, *Computerized Inventory Control Systems*, Prentice-Hall, 1978