uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Tapu Kumar Ghose

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.C.S.

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Dynamic Pricing in Electronic Commerce Using Neural Network

TITRE DE LA THÈSE / TITLE OF THESIS

Thomas Tran

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

Liam Peyton

Anil Somayaji

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Dynamic Pricing in Electronic Commerce using Neural Network.

by

Tapu Kumar Ghose

A thesis submitted to the

Faculty of Graduate and Postdoctoral studies

In partial fulfillment of the requirements for the degree

Masters in Computer Science

Ottawa-Carleton Institute for Computer Science

School of Information Technology and Engineering

University of Ottawa.

June, 2010

Your file Votre référence
ISBN: 978-0-494-69033-8
Our file Notre référence
ISBN: 978-0-494-69033-8

Abstract

There exist intelligent agents to aid online sellers to dynamically calculate a competitive price for their products in online markets. However, these intelligent agents usually make a number of assumptions for dynamic pricing. Some intelligent agents assume that sellers consist of prior knowledge about the online market parameters. In other words, the agents assume that the sellers are well aware of other competitors' pricing strategies, consumers purchase preferences, consumers' reservation price, profit made by other competing sellers etc. In addition, other agents assume that price is the only attribute that determines consumers' purchase decision. On the contrary, in real life sellers have limited or no prior knowledge about the market parameters. In addition, nowadays along with price other attributes such as after sale service, product quality etc. contribute in determining consumers' purchase decision. In this thesis, we propose an approach where sellers have limited knowledge on market parameters. We also assume that buyers' purchase decisions are based on multiple attributes. We are using a feed-forward neural network approach for calculating a competitive price dynamically to increase the sellers' revenue. Product price, product quality, delivery time, after sales service and seller's reputation are taken into consideration while determining the competitive price of the product by our model. In our experimental evaluation we showed that once the sellers, by considering the five attributes, set an initial price of the product, our model adjusts the price of the product

automatically with the help of neural network in order to raise the revenue. In setting the initial price of a product, we assume that sellers use their prior knowledge about the prices of the product offered by other competing sellers. Any other prior knowledge like buyer demand or competitor's price setting behaviors is not used in our evaluation. The experimental results portray the effect of considering the five attributes in earning revenue by the sellers. Before concluding with directions for future works, we discuss the value of our approach in contrast with related work.

Acknowledgements

I would like to thank my supervisor, Dr. Thomas T. Tran, for his patience and guidance throughout my research. His extraordinary mentorship and encouragements lead to the completion of the research work. I would also like to thank him for providing all the necessary facilities and financial support required for this task. I take the opportunity to thank my family members for their supports and inspiration without which it would not be possible.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Motivation

In today's sophisticated world's competitive online market, buyers take the aid of shopbots in determining which seller to select for purchasing any product. In the words of Greenwald et al. [31], "Shopbots are helping more and more buyers minimize expenditure and maximize satisfaction". A traditional shopbot is an online comparison service that filters products information based on different dimensions, such as price, product quality, sellers' reputation etc. According to Kephart and Greenlawald [26], shopbots are internet agents that search information regarding price, quality of goods and other attributes along with services by searching different vendors. Shopbots, by providing valuable information, assist buyers in making purchase decision. Markopoulos and Kephart [25] analyzed the importance of price information provided by shopbots to the buyers. They figured out that price information of a product is 6% to 10% as valuable as the product itself.

The buyers in the online economy take the aid of shopbots to minimize their expenditure. Along with the existence of multiple competitive sellers, the appearance of shopbots turned the task of setting a dynamic selling price for products by a seller into a

challenging one. Similar to shopbots, pricebots entered the market to aid sellers in the online economy in setting a selling price for products. Sellers make use of pricebots to maximize their profits. Greenwald and Kephart [28] described pricebot as an agent that employ automated pricing algorithm.

In dynamic pricing products prices always respond to the fluctuation of the market and hence the prices keep on changing with the tick of a clock. Every seller wants to set the selling price of their products so that their revenue is increased.

If a seller X sets the price of a product too high (say, P_h) then he/she could make greater profit from buyers. However, the number of buyers purchasing the product at price P_h would be zero or very few, because buyers would prefer to purchase the product from a different seller Y if that seller offers the same product at a lower price. Moreover, when the buyers, who bought the product from seller X at price P_h, realize that seller X has set the price too high compared to other sellers in the market then the buyers would be much less likely come back to seller X for purchasing any products in future. Ultimately, the seller X would not be able to make satisfactory revenue due to setting the price of the product too high.

On the other hand, if seller X sets the price of the product too low (say, P_l) then he/she might attract a greater number of buyers. However, since P_l is too low, seller X would not make as much revenue as expected. In such case, seller X could increase the price of the

product by a certain amount in order to lift up his/her revenue. In addition, setting the product price too low might have some negative psychological impact on buyers' purchasing behavior. Some buyers may think that because of the low quality of the product seller X has set the price of the product too low compared to other competing sellers in the market.

Therefore, determining selling prices of products is a challenging task for the sellers to sustain in the market. The purpose of the dynamic pricing problem is to determine selling prices such that sellers receive satisfactory revenue. Dynamic pricing, or personalized pricing, has been defined as estimating a shopper's desire, measuring his/her means, and then charging accordingly [1]. The motivation of this thesis lies here with the development of an algorithm to dynamically determine selling prices of products such that sellers earn better revenue compared to revenue earned by using traditional pricing algorithm.

1.2 Research Problem

Usually, a customer before buying a product selects a store/seller for the purchase. The selection may be done under multiple attributes (preferences), such as best price offered, after-sale services, product quality, delivery time, sellers' reputation etc. Therefore, the sellers have to provide a competitive price for a product in response to variation in the market parameters such as competitors' prices and consumers purchase preferences.

There exist intelligent agents, called pricebots, which enable online sellers to dynamically calculate a competitive price for a product. According to Dasgupta et al. [2], "these intelligent agents provide a convenient mechanism for implementing automated dynamic pricing algorithms for the sellers in an online economy". However, some intelligent agents use a number of assumptions for the dynamic pricing in online markets. Some intelligent agents assume that sellers are provided with complete knowledge of market parameters, while some other agents consider product price as the only attribute that determines consumers' purchase decision [2, 9, 10, 12, 22]. In recent decades extensive research has been done in dynamic pricing. Some of the research made an assumption that there is only one seller in the market [17]. The authors, in their model, assumed a market with imperfect competition such as a monopolist market. Their main concern was to maximize sellers revenue by dynamically pricing the products where products were supposed to be sold within a given time horizon. However, in real life sellers have limited or no prior knowledge about the market parameters (e.g., buyer's reservation price, competitive sellers' price and profit etc). In addition, in reality there exist several competitive sellers in online market. The goal of this work is to address the problem of dynamic pricing in a competitive online economy, where sellers have limited or no prior knowledge about the market parameters and where a buyer's purchase decision is determined by multiple attributes.

1.3 Thesis Contributions

- In this thesis, we propose an approach of dynamic pricing using feed-forward neural network where buyers purchase decision is dependent on multiple preferred purchase attributes. As discussed in Section 1.2, some of existing intelligent agents for dynamic pricing make an assumption that price is the only attribute that a buyer would be interested before making any purchase. On the contrary, in present days besides price other attributes such as after sale service, product quality etc. contribute in determining consumers' purchase decision. In our approached model of dynamic pricing we considered five attributes namely product price, product quality, delivery time, after sales service and sellers' reputation. However, our model is general enough to work for any number of attributes. Some agents assume that there exists only one seller in the market. In contrast, we can see multiple sellers competing in today's competitive market of online economy. Our model expects that there will be multiple competing sellers in the market. The approach requires the sellers, by considering the five attributes, to set an initial price of the product by using their prior knowledge about the prices of the product offered by other competing sellers. Our approach adjusts the selling price of products automatically with the help of neural network in order to raise seller revenue. The experimental results portray the effect of considering the five attributes in earning revenue by the sellers.

- Partial work of this thesis has been published in MCeTech'09 conference: Tapu Kumar Ghose and Thomas T. Tran. "Dynamic Pricing for Electronic Commerce using Neural Network", *Proc. 4th International MCETECH Conference on e-Technologies*, Ottawa, ON, Canada, May 2009.

- The work of this thesis has appeared in AI'2010 conference: Tapu Kumar Ghose and Thomas T. Tran. "A Dynamic Pricing Approach in E-Commerce based on Multiple Purchase Attributes", *Proc. The 28th Canadian Conference on Artificial Intelligence*, Ottawa, ON, Canada, May 31 – June 2, 2010.

1.4 Thesis Outline

The remaining of the thesis is organized as follows: Chapter 2 provides the background information on neural network and back propagation technique followed by related work. Chapter 3 presents our proposed approach for dynamic pricing using neural networks. Chapter 4 represents results and analysis from our experimental evaluation followed by a brief discussion on our model. Finally, chapter 5 concludes the thesis with future research directions.

Chapter 2: Background and Related Work

2.1 Neural Network

Neural network is a mathematical model which is composed of a large number of highly interconnected groups of artificial neurons that are used to process information [19]. Generally, neural network is an adaptive system that learns from example [5]. The structure of the neural network is based on the information flowing through the network. The complexity of the structure can be determined by the connection between the neurons [19].

The market of the present world is very competitive. This fact makes the sellers to change their offered prices of products frequently in order to sustain in competitive online economy. In other words, products prices always respond to the fluctuation of the market and hence the prices keep on changing with the tick of a clock. Therefore, in establishing a dynamic price our model employs an algorithm whose output changes with time for a given scenario. Since traditional programming methods always provide the same output for a given problem, they do not work in our case. In the traditional programming, we code line by line, instruction by instruction how the program should perform a calculation

or task. Hence, a traditional programming always leads to one specific output for a specific input. In contrast, our model employs an algorithm where the output is not static. Therefore, we aim to use an adaptive method where instead of telling the program how to do something, we show it examples of what we want to be done. We provide it with data on inputs and show it what outputs we want and the adaptive system builds the actual function autonomously. There exist several randomized and adaptive methods. Among them we choose feed-forward neural network to address the problem as it has the ability to determine trends and extract patterns from imprecise data and provides output depending on the determined trends.

2.1.1 Framework of Neural Network

Neural Networks consist of a set of nodes and links. Generally, all nodes fall under one of the three layers: input layer, hidden layer and output layer. The nodes in input layer accept information from outside the network, while the nodes in output layer send information outside the network. Each node, also known as unit, is connected to one or more other nodes by directed links. Each link contains a numerical weight, for instance $W_{i,j}$ indicates the strength of the connection between unit i and unit j, where the link is directed from unit i (u_i) to unit j (u_j). The numerical weight, $W_{i,j}$, ranges from 0 to 1.

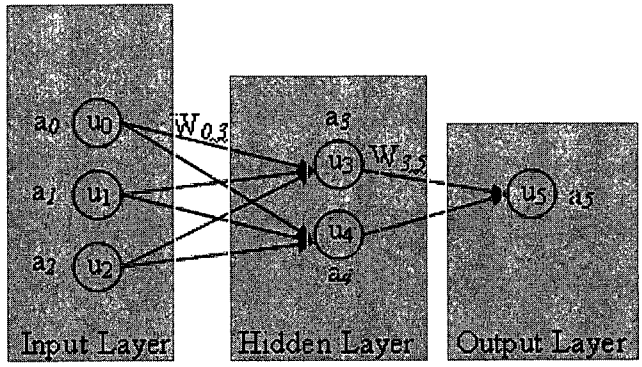

Figure 1: A general Neural Network.

Each unit u_i has an activation value a_i which acts as output of the unit. The activation values of the units of the input layer are set to some predefined values. A link from u_i to u_j serves to propagate the activation a_i from u_i to u_j. Each unit u_i first computes a weighted sum of its input. The computed sum is then passed as parameter to an activation function f to derive output a_i of the unit as follows:

$$a_i = f(\sum_{j=0}^{i-1} W_{j,i} a_j) \qquad (1)$$

where, $\sum_{j=0}^{i-1} W_{j,i} a_j$ is the weighted sum of the inputs to unit u_i and f is the activation function applied to the weighted sum. This can be summarized by the following figure:

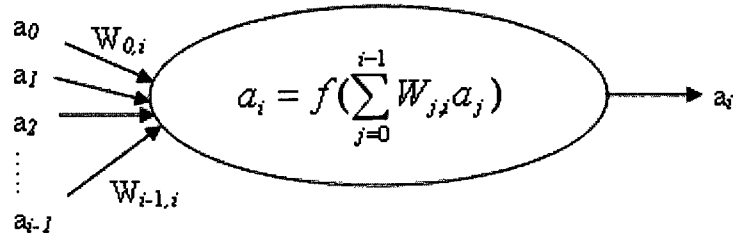

Figure 2: A simple Neuron.

The activation function *f* controls the amplitude of the output of the units. In the simplest case *f* is the identity function, f(x) = x, and the unit's output is just its net input [19]. Step function, logistic sigmoid and symmetric sigmoid can also be used as activation function. Step function gives 0 or 1 as output, whereas, the range of output for symmetric sigmoid function varies from -1 to 1. On the other hand, the output of logistic sigmoid function ranges from 0 to 1. Among these functions, symmetric sigmoid provides higher range of output. However, it provides negative values too. Our approached model of neural network produces a competitive price of a product as output of the network. Since the price of a product cannot be a negative value, we have chosen logistic sigmoid function as the activation function. The function is as follows:

$$f(x) = \frac{1}{1+e^{-x}}$$

In worst case scenario, assume that all the numerical weights of the links between the input layer and the hidden layer of Figure 1 are zeroes. In such case if identity function is used as the activation function, according to equation 1, the activation value or output of each unit of the hidden layer would be zero. Consequently, the output of the entire network would be zero. However, the aim of our network is to produce a competitive price of a product in order to increase the sellers' revenue. To overcome this problem we include a bias unit in our network.

A bias unit is connected to any unit of the input layer. We assume that the bias unit is connected to unit u_0 of the input layer. Every product has its own production cost below which sellers are unwilling to sell the product. Therefore, we are setting the production cost of the product as the output of the bias unit. In addition, we set the numerical weight of the links associated with the bias unit to 1. With the help of the additional bias unit we can ensure that our network will never provide the price of a product below its production cost.

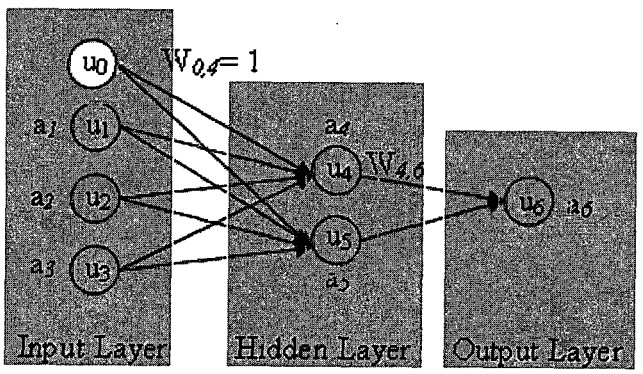

Figure 3: Neural Network with additional Bias Unit.

In our model of neural network, a unit is activated when the weighted sum of the inputs exceed the input of the bias unit. The weighted sum of the inputs also includes the bias unit's input.

2.1.2 Feed-forward Neural Network

There are three types of units that can be found in the neural network: input units, hidden units and output units (Figure 1). Input units receive data from outside of neural network. Output units send data outside the neural network. The input and output data of hidden units are remained within the network. Neural network can be divided into two major categories: feed-forward network and feed-back network. Feed-back network feeds its output back into its own inputs. In feed-forward network each unit in a specific layer receives input only from the units in the immediately preceding layer. Unlike feed-back networks, there is no loop in the feed-forward networks, i.e., the output of any layer does not affect the same layer [4].

The most common feed-forward network consists of three layers: input layer, hidden layer and output layer. Every unit of the input layer is connected to one or more units of the hidden layer, and every unit of the hidden layer is connected to one or more units of the output layer.

The advantage of adding the hidden layer is that it enlarges the space of hypothesis that the network can represent. Any continuous activation function of inputs with arbitrary accuracy can be represented with a single hidden layer [21]. If two hidden layers are used then discontinuous functions can also be represented. The problem of choosing the right number of hidden layers is still not well understood [4, 20].

2.1.3 Back Propagation Algorithm

The selection of links among the units in neural networks plays an important role in performing any specific task. The assignments of the weights to the links must be done appropriately such that the networks produce a better approximation of the desired output. The objective behind most algorithms for neural networks is to adjust the weight of each link of the networks to minimize some measure of the errors on the training set [4].

In order to train a neural network to perform a specific task, the weights of each unit must be adjusted in such a way that the error between the desired output and actual output is reduced. This requires neural network to calculate rate of change of errors with respect to weights, i.e., how errors change as weights are changed.

The back-propagation algorithm is the most widely used method for determining the error derivative of the weights, denoted by ΔE. ΔE can be computed as follows [4]:

$$\text{Error, } E = \tfrac{1}{2}\,Err^2 \approx \tfrac{1}{2}\,(X - Y)^2, \text{ where } X = \text{desired output, } Y = \text{actual output}$$

The squared error can be reduced by calculating partial derivative of error E with respect to every weight $W_{j,i}$.

$$\Delta E = \frac{\partial E}{\partial W_{j,i}} = \frac{\partial}{\partial W_{j,i}}(\frac{1}{2}Err^2) = Err \times \frac{\partial Err}{\partial W_{j,i}}$$

$$= Err \times \frac{\partial}{\partial W_{j,i}}(X - Y)$$

From equation (1), we know the actual output of each unit is represented by $f(\sum_{j=0}^{i-1} W_{j,i} a_j)$. Therefore, by substituting the value of actual output Y from equation (1) we get,

$$\Delta E = Err \times \frac{\partial}{\partial W_{j,i}}(X - f(\sum_{j=0}^{i-1} W_{j,i} a_j))$$

$$= Err \times (0 - (a_j \times f'(\sum_{j=0}^{i-1} W_{j,i} a_j)))$$

$$= -Err \times a_j \times f'(\sum_{j=0}^{i-1} W_{j,i} a_j) \qquad (2).$$

The algorithm computes the error derivative of the weights, ΔE, by computing the amount of error at each unit. ΔE gives the direction in which the weight, W, has its steepest slope. Therefore, in order to reduce error we move a small step in the opposite direction of ΔE (i.e., - ΔE). The small step is determined by a constant called learning rate, α. By using equation (2), the error E can be reduced by updating the weights as follows:

$$W = W + \alpha \times (-\Delta E), \text{ where } \alpha \text{ is the learning rate}$$

$$= W - \alpha \times \Delta E \qquad (3).$$

The learning rate, α, is a constant value (real number) which ranges from 0 to 1. It determines by what amount we change the weights W at each step.

The error, Err_k for the output units u_k is simply the difference between desired output (X) and actual output (Y).

$$Err_k = (X - Y)$$

Substituting the value of Err_k into equation (2), we get

$$\Delta E_k = -(X - Y) \times a_j \times f'(\sum_{j=0}^{k-1} W_{j,k} a_j) \qquad (4).$$

Using equation 3 and 4, the weights $W_{j,k}$ for the links between hidden layer units u_j and output layer units u_k are then updated as follows:

$$W_{j,k} = W_{j,k} - \alpha \times \Delta E_k, \text{ where } \alpha \text{ is the learning rate.}$$

Since the units u_k of the output layer are connected to the units u_j of the hidden layer, the units u_j have influence for some fraction of the error ΔE_k. Therefore, the error ΔE_k is divided according to the weights of the links between the units u_j and the units u_k. The divided error is then propagated back to determine the errors Err_j for each unit u_j of the hidden layer. This is accomplished by multiplying the weights $W_{j,k}$ of each link between u_j and u_k with ΔE_k and then summing up the products.

$$Err_j = \sum_{j=0}^{k-1} W_{j,k} \Delta E_k$$

Substituting the value of Err_j into equation 2, we get

$$\Delta E_j = - \sum_{j=0}^{k-1} W_{j,k} \Delta E_k \times a_i \times f'(\sum_{i=0}^{j-1} W_{i,j} a_i) \qquad (5).$$

Using equation 3 and 5, the weights $W_{i,j}$ for the links between hidden layer units and input layer units are then updated as follows:

$$W_{i,j} = W_{i,j} - \alpha \times \Delta E_j, \text{ where } \alpha \text{ is the learning rate.}$$

In summary, the process of training the three-layered network using back-propagation algorithm to reduce the errors at each layer by updating the weights is as follows:

i. Compute the errors ΔE_k at the output units which is simply the difference between desired output (X) and actual output (Y).

ii. Update the weights between the hidden layer and the output layer by using the errors ΔE_k.

iii. Propagate the errors ΔE_k back to the hidden layer to find the errors ΔE_j at the hidden layer units.

iv. Update the weights between the input layer and the hidden layer by using the errors ΔE_j determined in step (iii).

2.2 Literature Review

Over the past few years there can be observed a noticeable rise in interest of dynamic pricing in commercial and research communities. Several analytical models have been developed for dynamic pricing in online economies [1, 16, 22, 9, 11, 2]. The main aim behind dynamic pricing is to adjust the product price in order to capitalize sellers' revenue. In spite of rich literatures in the field, majority of the research works do not consider the competition markets where there exist multiple sellers for selling the same product. [12].

2.2.1 Monopolist Market and Single Purchase Attribute

Setting the exact price of a product that would maximize revenue is one of the most challenging tasks for the sellers. Li et al. [24] studied the enterprises' dynamic decision problem on price strategies (dynamic pricing decision) in duopolistic retailing market under uncertain market state. They assumed that two enterprises simultaneously choose their strategic variable in each period to maximize their expected revenue. They used Markov stochastic game frame to build up their model. In their design they showed two distinct type of reinforcement learning methods, Nash Q-learning and Best-response Q-learning, in their simulation. Their numerical study concluded that the Best-response Q-learning method outperformed the Nash Q-learning method in dynamic pricing decision in duopolistic retailing market.

Chinthalapati et al. [9] used machine learning based approach to study price dynamics in an electronic retail market. The paper gave an overview on what steps the sellers might take in order to attract the customers to buy goods. Volume discount, consumer segmentation and sales promotion can be some of them. In the study they considered a multi seller environment which consisted of two competitive sellers. In addition, the authors have taken price attributes into consideration that would determine a customer's buying decision. The study was carried on two scenarios. First, "*no information case*" where none of the sellers had any information about customer queue levels, inventory levels, or prices at the competitors. Second, "*partial information case*" where all the

sellers had information about the customer queue levels and inventory levels of the competitors. The authors demonstrated a Reinforcement Learning (RL) based pricing algorithm to reset the prices at random intervals with the aim of maximizing discounted cumulative profit. The resetting of prices based on few factors such as number of back orders, inventory levels, and replenishment lead times. The authors discussed about two types of consumers: captives and shoppers. Captives are not price sensitive and buy the product at per unit cost. They get preference over the shoppers with regard to supply of items in the absence of stock. Shoppers receive their products only after delivering the goods to the captives. Shoppers are price sensitive. They are willing to bear with the waiting based inconveniences imposed by the sellers. In a nutshell, the authors considered two competitive sellers in the markets. They also assumed price as the only attributes of the product in determining customers' buying decisions.

In derivative following (DF) strategy, initially, product prices are set randomly and profitability is observed. The product prices are increased in the same direction unless the observed profitability falls. If the observed profitability falls then product prices are decreased as long as profit is encountered. It requires keeping track of past average profit of each state, and increases the prices till the profitability level falls [9]. Dasgupta et al. [10] studied dynamic pricing in a multi-agent economy which consisted of buyers and competing sellers. They divided buyers into two categories: bargain hunting buyers and random selecting buyers. Each buyer has a valuation price P_{val} below which he/she unwilling to pay for a product. A bargain hunting buyer employs a shopbot to select the

seller that offers the lowest price and purchases the good if the price offered by the seller is below P_{val}. A random selecting buyer selects a seller at random and purchases the good if the price offered by the seller is below P_{val}. They had taken price as the only attribute which take part in buyers purchase decision. They considered that each seller had limited information about the competitors' prices. In their work, they refined the DF algorithm and introduced a model optimizer (MO) algorithm that re-estimates a relationship between price and profit for a seller at every interval more efficiently and builds an internal model by nonlinear regression of historical data with time-discounted weighting. The simulations showed that MO outperforms DF even though it has no additional information about the market. C. Brooks et al. used neural network for dynamic pricing where a monopolist market has been considered [7, 8].

In contrast, our model considers four more attributes (product quality, delivery time, after sale service and sellers' reputation) other than price. In fact our model is general enough to work for any number of attributes. Moreover, our model is not limited to two competitive sellers. Our model is general enough to work for both monopolist market and a competitive market with multiple sellers.

2.2.2 Pricing for Products with Finite Time Horizon

Dimicco et. al [34], by using Learning Curve Simulator, analyzed performance of two adaptive pricing algorithms: Goal-Directed (GD) and Derivative-Following (DF). They

considered both monopoly and competitive economy of finite markets where goods like airlines ticket, sport events ticket, perishable goods have to be sold by finite time horizon. The goal of the GD strategy is to sell entire inventory by a given finite time period. It follows a strategy of lowering the prices when sales are low and raising prices when sales are high. In contrast, DF relies on historical data. It adjusts price based on revenue earned on previous day due to previous days price change. It follows a strategy of changing the price in same direction if revenue earned by following previous days price change is less. The direction of changing price is reversed otherwise. The authors found that GD outperforms DT for slower moving markets, whereas, DF is superior when there is an early demand peak in the market. Moreover, in monopoly condition, it is recommended by the authors to use similar strategies like GD since its goal is to sell the entire inventory. Another finding is that when GD and DF co-exist, GD success is detrimented by DF strategies. However, adaptive pricing strategies result in price war if buyers are too pricing sensitive.

Alexandre et. al [13] discussed on the problems of dynamic pricing in finite time horizon. They considered a retailer who has to set the price of a good to optimize the total expected revenues over a period of time T. Their model is dependent on demand curve of the products. They assumed both demand and price were to be continuous variables. They studied the problems of optimizing sales revenues based on a parametric model in which the parameters were unknown. The sellers had to set the price at a level in order to maximize current revenue and at the same time learn about the parameter values in order

to increase the future revenues. It has been showed that among different strategies for learning, one-step look-ahead rule produced good short term performance.

Kong [11], in his paper, examined seller strategies for dynamic pricing in a market for which a seller has finite time horizon to sell its inventory. For this purpose a dynamic pricing strategy is developed using neural network based on online learning (called SDNN strategy, Sales-Directed Neural Network). The SDNN strategy takes in account the dynamics and resulting uncertainties of the market place. Neural network used here consists of three layers: input, hidden and output layer. The only unit of the input layer takes real price of the product as input, whereas, output is the sales quantity. The SDDN strategy continually used the observed sales to calculate the error between the current sale and desired sale. The error is then propagated backward through the network and small changes are made to the weights in each layer. The strategy did not use any information regarding buyer population or competitor's pricing strategies. The only knowledge included in the SDNN strategy was the demand curve. Here, the functionality relationship between price and sales quantity was represented by the demand curve.

Dasgupta et. al [14], in their paper, employed push strategies mechanism for dynamic pricing. The authors considered time-limited goods in a supplier driven marketplace where goods are sold by maintaining strict deadline. These goods become nominal after a certain deadline. Examples of such goods include electricity, airline tickets, goods with expiry date like dairy products etc. In their paper the authors had considered dynamic

pricing in two scenarios. In the first case they assumed that seller were unable to sample buyers' demand curve. In such scenario the sellers used a heuristic dynamic pricing technique called Maximum Return Algorithm to estimate and refine the demand curve of goods at each buyer. The supplier then offers a price at every buyer that maximizes the sum of the immediate profit to the supplier for the currently estimated demand curve, and the estimated value of the surplus stock that the supplier is unable to sell in the market. In the second case, the authors studied a scenario where the deadline to sell the products is not strict. In such case, the authors showed that the sellers sample the buyer demand and use it to dynamically negotiate an exchange point for the good which simultaneously improves both the sellers' profits and buyers' utility, as compared to trading without negotiation. The authors used price and quantity of the products as the only criterion that determine buyer's purchase decision.

In contrary, our model is not dependent on buyer's population in the market, which is why we did not consider demand curve in the model, because demand curve cannot be produced without the buyer's population. Therefore, our model of dynamic pricing does not require the sellers to figure out the demand curve of the products. In our approach of dynamic pricing our objective is to aid the online sellers in determining a selling price for the products. In the approach we are not concerned whether the products are perishable or not. In other words, our model is not limited to goods with finite time horizon.

2.2.3 Kephart in Dynamic Pricing

Kephart et al. [27], for their work, considered a picture where a monopolist seller willing to maximize his/her revenue, provided buyers demand curve is random and unpredictable. The authors, in their model, employed a trial and error technique to optimize sellers' revenue. They claimed that fixed pricing or two-parameter dynamic pricing technique are preferred when demand curve is unpredictable. According to them, trial and error technique, besides providing satisfactory revenue, helps in enriching underlying consumer preferences information which could turn out to be handy for the sellers if they want to receive higher profits by capitalizing on the information. Their model did not take demand curve of consumers into consideration. Instead, the model uses historic information regarding profits earned while different price schedules are practiced by the sellers. With the aid of this information, the model attempts to learn the most profitable price schedule. To accomplish this task the authors employed a modification of the amoeba algorithm. In their simulation, the authors worked with five different price schedules, namely, pure bundling, linear pricing, two part tariff, mixed bundling and non linear pricing. The authors have related sellers' response to topography of profit landscape, degree of exploration and frequency of shocks. They showed that non-linear pricing schedule work best when the demand shock is very infrequent. Among the five price schedules two-part tariff and mixed bundling are less complex and they can be learn quicker than other price schedules.

Greenwald and Kephart [28] explored no-regret learning for probabilistic pricing algorithm. They examined both high information and low-information settings. The goal of their work is to identify most profitable pricebot algorithms by investigating dynamics of interaction among different pricebot algorithms. They made an assumption that buyers follow two strategies (Bargain Hunter and Any Seller) for selecting a seller to purchase the goods. Buyers practicing Bargain Hunter select the seller who offers lowest price compared to other sellers. Buyers of this category usually take the aid of shopbots. On the other hand, buyers practicing Any Seller strategy select any random seller provided that seller offers less prices than buyer's valuation. Buyers of this category usually prefer product quality or other attributes over product price. The authors, in their work, studied two typed of pricebot algorithms: informed and naive. An informed algorithm requires relevant profits information as input, whereas, naive algorithm functions need no information. They run simulations with two to five adaptive pricebots algorithm that employ no regret pricing strategies. They found that, for both informed and naïve, No Internal Regret (NIR) and No External Regret (NER) pricebots converge closely to Nash equilibrium. In their model they considered an economy for single homogeneous goods. On the contrary, our model is not restricted to homogeneous goods.

Tesauro and Kephart [30] showed that unaccepted behaviors of pricebots such as eternal price war can be avoided by introducing foresight in the pricing algorithm. For this task they proposed two heuristic approaches based on adaption of classic minimax fixed-depth search algorithims and dynamic programming (DP) style algorithms. For the first

approach they found that it is adequate to curtail price war by using any amount of look-ahead in the pricing algorithms. For the second approach the authors used three versions of DP algorithms: synchronous, asynchronous and incremental. The incremental version is an extension of asynchronous version where a random element is adjusted by a small increment towards the calculated optimal price. Their experimental results portrayed that, among the three versions, incremental version converge to a unique self-consistent solution. Our model is not concerned about how many sellers are there in the market, whereas, in their experiment they assumed that there are only two competing sellers participating in the economy who alternatively take turns in adjusting their prices at each time step.

Greenwald et. al [31] studied four different price-setting strategies: game-theoric pricing (GT), myoptimal pricing (MY), derivative following (DF), and Q-learning (Q). These strategies are different from each other in terms of infomational and computational requirements. They analyzed their results for both homogeneous and heterogeneous settings. They found that when all pricebots use same pricing strategy (homogeneous setting), DF outperforms MY and GT. In contrast, Q strategy demonstrate superior performance over all strategies when different pricing algorithms are practiced (heterogeneous setting). Kephart and Tesauro [29] studied nature and behavior of Q-learning in a model market in which two interacting agents co-exit. They analyzed symmetric and asymmetric solutions for the model. They found that, between symmetric and asymmetric, choice of solution obtained by Q-learning depends on discount

parameter and on randomness of exploration. The trend is that with increasing discount parameter, asymmetric solution is preferred over symmetric solution. However, there are some limitations in GT, MY, DF and Q-learning. GT strategy makes an assumption that all other competing sellers use game-theoretic [31], however, in present world different sellers employ distinct pricing strategies. In comparison, our model is not concerned about what strategies are being used by other competing sellers. MY strategy assumes that prices set by other competing sellers will remain unchanged [31]. In the contrary, sellers are always willing to change their offered price for the sake of sustaining in competing market. Hence, our model always keeps an eye on the random prices set by other sellers. Q-learning strategy uses a lookup table representation of the Q-function and requires extensive size of computational requirement. It makes use of both buyers' demand curve and knowledge about competitors pricing strategies. On the other hand, our model does not rely on buyer demand curve.

2.2.4 Overview of Our Approach

In our proposed approach, the dynamic pricing described uses feed-forward neural network to determine a competitive selling price for the products and use little prior knowledge about market parameters. Moreover, instead of taking only price attribute, we took five different attributes based on which a customer's purchase decision can be made. However, our model is general enough to work for any number of attributes. For instance, if a seller wish to determine a competitive price for his/her product without

considering sellers reputation attribute, then he/she can use our model by deleting the corresponding input unit from the input layer of our network. On the other hand, if a seller plan to derive his/her product's price by taking another additional attribute into consideration, then our model can be used by adding an extra input unit into the input layer. In a word, our model is suitable for flexible number of attributes.

Chapter 3: Design and Implementation of Proposed Model using Feed-Forward Neural Network

3.1 Introduction

In present days the customers' buying decision is not only determined by the product price. Along with the product price, customers purchase decision is also triggered by other attributes such as product quality, delivery time, after sale service, sellers' reputation etc. The preferable purchase attributes of product may vary from buyers to buyers. To demonstrate this, let us assume that the price for a product, P, based on different attributes is offered by three different sellers according to Table 1. If a buyer B_A is concerned with the quality of products, then he/she may prefer to buy the product, P, from the seller S_B who offers the best price amongst the three sellers when price is set based on product quality. Similarly, a buyer B_B would choose seller S_A when he/she wish to enjoy after sale service.

Table 1: Price offered by different sellers for a product, P, based on different attributes

Seller	Product Price	Product Quality	Delivery Time	After Sale Service	Seller's Reputation
S_A	10.50	10.75	12.25	11.52	11.98
S_B	11.50	10.65	12.85	11.54	11.62
S_C	10.25	10.95	12.50	11.61	11.25

In order to increase the revenue, a seller has to set the selling price of products in such a way that the price is capable of attracting as many buyers as possible, provided that there are enough inventories for the products. Since buyers preferred purchase attributes of a product vary, in order to attract buyers from wider range the sellers have to consider multiple purchase attributes in setting the product price. For instance, the seller S_A from the above example could have attracted both the buyers B_A and B_B if S_A would have considered both product quality and after sales service attributes while setting selling price for the product P. Moreover, sometimes it may happen that a buyer is appearing with more than one preferred purchase attributes, or in other words, a buyer may wish to enjoy multiple preferred purchase attributes before making any purchase decision. Therefore, in order to attract greater number of buyers, instead of providing different selling prices based on different attributes, the sellers can provide a single price that has been set by considering multiple purchase attributes. Under the above circumstances, we propose a model for dynamic pricing which takes multiple purchase attributes into consideration and provide a single price that covers all the considered attributes. As discussed in the following subsections, our model takes five purchase attributes of products into consideration in providing a selling price for a product, namely product price, product quality, delivery time, after sales service and seller's reputation.

Once the sellers set an initial price of the product, our model adjusts the price of the product automatically with the help of neural network in order to increase the revenue earned. In setting the initial price of a product, we assume that sellers use their prior

knowledge about the prices of the product offered by other competing sellers. No other prior knowledge is used in the evaluation. In the following chapter we simulate an e-commerce market place with three different sellers. The sellers employ three distinct pricing techniques: our model, simple pricing algorithm and derivative-following (DF).

From the experimental evaluation demonstrated in Section 4.2.4 and Section 4.3 we can see that our approach performs better than a traditional simple pricing algorithm. A seller, by taking five attributes of our model into consideration, could employ a simple pricing algorithm to determine a competitive price of products. A simple pricing algorithm may take at least the production cost of a product as initial selling price of the product. If a buyer prefers to enjoy any additional attributes such as after sale service, then the algorithm may wish to add some additional price for each supplementary attributes. Finally, the algorithm would provide a selling price of the product. Since in online economy prices of products do not remain static, a seller has to frequently update his/her offered price of the products. While updating the prices, there can arrive two different scenarios for a seller who employs the simple pricing algorithm. First, the algorithm, while updating the price by adding extra amounts for additional attributes, does not use any information on how other competing sellers in the market set their selling price. Since the algorithm has no knowledge about market parameters, it uses some random extra prices for additional attributes. Hence, the price can be too low or too high which may lead to inadequate revenue for a seller. In other words, sellers employing a simple pricing algorithm run a risk of earning less revenue. In the second scenario let us assume

that a seller who employed the simple algorithm, do some manual search on the prices offered by other vendors. The algorithm uses information obtained from the seller's search to determine a selling price for products. However, the manual search could be time consuming. It might take from hours to days or even longer to gather information by manual search. Since prices change within very short span of time in online market, the information acquired by manual search during relatively large span of time might become outdated. Consequently, the algorithm would be using obsolete information which may lead to inappropriate output. In contrary, our model outputs a competitive selling price of products by providing importance to the five attributes based on historical data, which implies it does not rely on any manual search. In addition, our technique, while determining competitive selling price, considers the sellers make use of their prior knowledge of the prices set by other competing sellers in providing initial input to the model. This indicates that our model keep an eye on the competitive price set by other sellers in competing market and utilizes fresh information of price. In other words, our technique performs better than manual search in terms of time and non-obsolete information.

3.2 Purchase Attributes

Best Buy and Future Shop are the largest retailers of electronic goods in the United States and Canada respectively. By analysing the prices offered by them, we found that their prices are set based on product price, delivery time, after sales service and product quality

(e.g., sealed products vs. opened box products). They offer different price for the same product based on different purchase attributes. For instance, if a buyer prefers to receive the purchased product earlier then they add some additional value to the selling price on the basis of faster delivery time. Similarly, a buyer is asked with higher selling price than the usual selling price if he/she wants to enjoy after sales services like warranty. According to a quarterly survey that was conducted on October 2008 in United States, eBay was ranked as No. 1 online retailer [43]. Besides product price and product quality, eBay also makes use of sellers' reputation in setting selling price of a product. In online shopping transaction it is difficult for a buyer to measure quality of the purchased goods well ahead of delivery. The buyers, hence, have to rely on the products' information provided by sellers. In such scenario seller reputation becomes an important means of reliance [35] before the outcome of the transaction. Chen et al. [37] studied the attributes or factors of e-commerce website that have effect on cosumers' on-line shopping preference and figured out that, beside other attributes, trust and delivery would play role in buyers purchase satisfaction. In the words of Dasgupta et al. [2], "micro-economic literature and online consumer surveys suggest that a consumer's purchase decision is determined by multiple product attributes including price, delivery time, seller reputation, product quality and after-sale service". From information of above mentioned literature review, the most common attributes that can play vital role in determining customers' purchase decision would include product price, product quality, delivery time, after-sale service, and sellers' reputation.

Under the above circumstances, in our model we choose to include product price, product quality, delivery time, after-sale service, and sellers' reputation as the purchase attributes of product in setting a selling price for a product, *P*. However, our model is not restricted to these five attributes. For simplicity we have chosen the said attributes. Our model is general enough to work for any number of attributes. We use feed-forward neural network to determine a competitive price for the products in order to raise sellers' revenue. The prices of products do not remain constant. It varies with time. Since traditional programming methods always provide the same output for a given problem, they do not work in our case. As discussed in Section 2.1, we choose feed-forward neural network to address the problem as it has the ability to determine trends and extract patterns from imprecise data and provides output depending on the determined trends.

3.3 Structure of the Proposed Model

The objective of the thesis is to determine a competitive price for a product such that the number of buyers willing to buy the product at the determined price is increased and hence the sellers' revenue is raised. In our model, for determining the price, we used a feed-forward neural network which contains three layers: input layer, hidden layer and output layer.

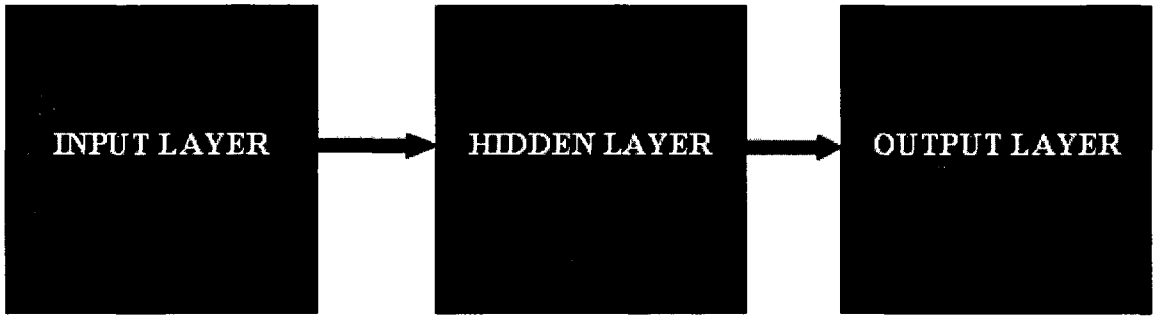

Figure 4: Block diagram of the proposed model.

3.3.1 Units of the Input Layer

In our model we are considering five important attributes which contribute in buyers' purchase decisions. These attributes are product price, product quality, delivery time, after sale service, and sellers' reputation. As discussed in Section 3.2 we believe that these are the most common attributes that buyers would consider to make purchase decisions. Therefore, the network we designed consists of five units in the input layer, one for each attribute. Each attribute can be assigned to any of the five units of the network if the unit is still unoccupied by any attribute. In our case, we are assigning product price, product quality, delivery time, after sale service and sellers reputation to u_1, u_2, u_3, u_4 and u_5, respectively.

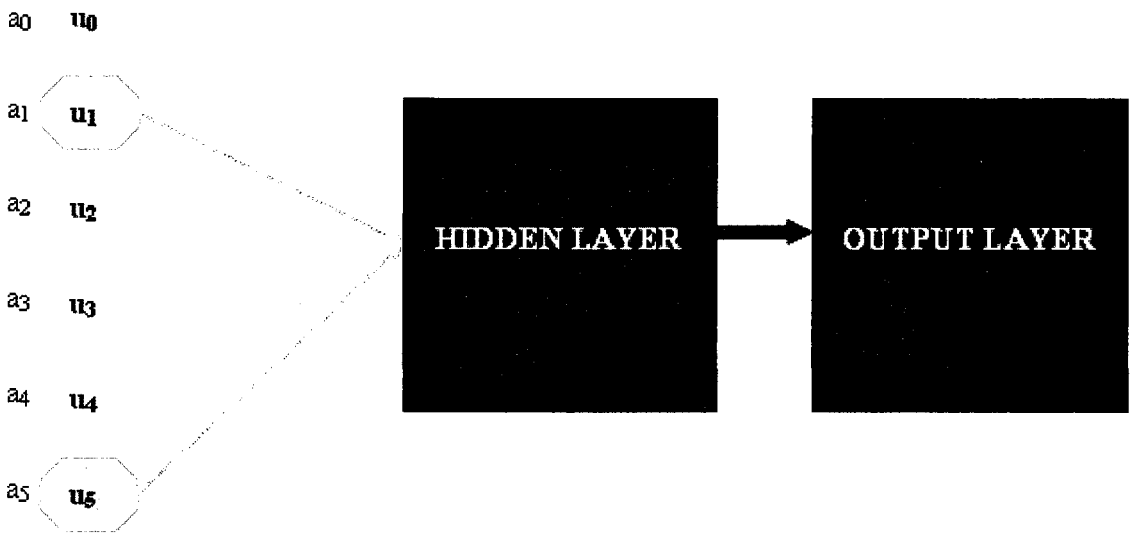

Figure 5: Units of Input Layer of the proposed model.

The input layer also consists of one extra unit u_0 as the bias unit as explained in Section 2.1.1. We set the value of a_0 to the production cost of the product. Usually, sellers are not willing to sell their products below the production cost of the corresponding products. Hence, we considered the production cost of the product as the output of the bias unit. Initially, all the values a_1, a_2, a_3, a_4 and a_5 of the input units are set by the sellers. These values represent input to the five units of input layer u_1, u_2, u_3, u_4 and u_5 respectively. In setting these values, we assume that sellers use their prior knowledge about the prices of the product offered by other competing sellers in the market. No other prior knowledge is used in our model.

3.3.2 Inputs of the model

All the units of the input layer accept numerical values as input, i.e., we provide a numerical value of the product based on different purchase attributes.

Product price: Different sellers offer different prices for an identical product in the online economy. A brand new "Sony Vaio (VGNNS330DS)" laptop of same configuration is offered at CDN 699.99 and CDN 679.95 by Best Buy[1] and Future Shop[2] respectively, as shown by Figure 6 and 7 below [40, 39].

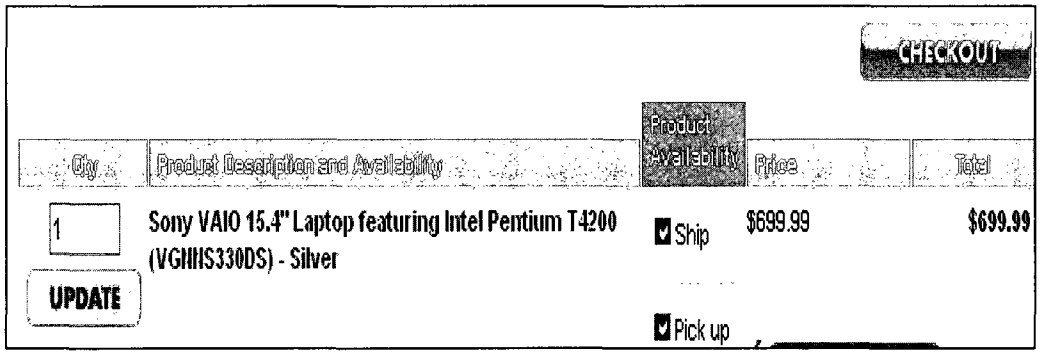

Figure 6: Price offered by Best Buy for Sony Vaio (VGNNS330DS) laptop.

[1] United States largest electronics retailer that also operates in Canada, Mexico, and China.
[2] Canada's largest electronic retailer.

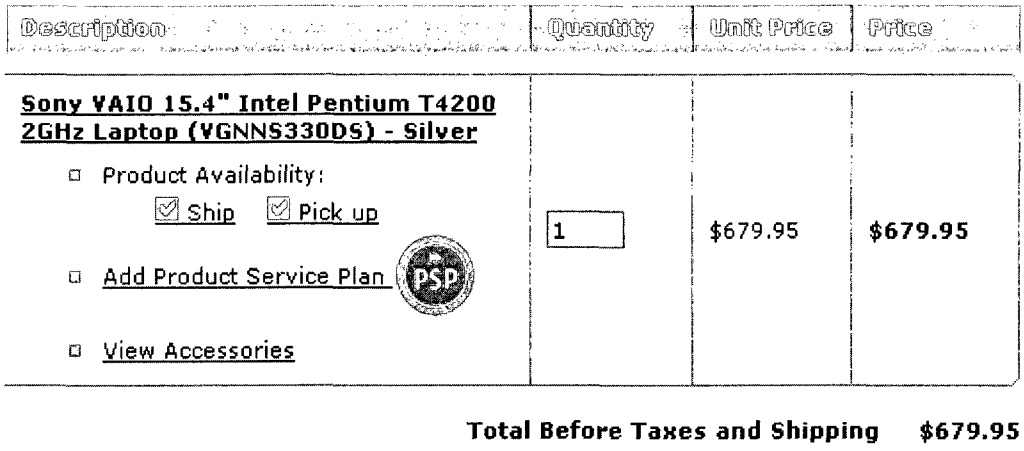

Figure 7: Price offered by Future Shop for Sony Vaio (VGNNS330DS) laptop.

Usually a consumer would like to buy a product from the lowest price offered seller. Therefore, when a buyer purchase decision is made by the product price offered, the buyer would choose Future Shop to purchase the laptop. The first unit, u_1, of the input layer of our network takes price of the product as input, a_1, based on the "product price" attribute. The sellers study the price offered by other competitive sellers in the market before providing an input to the first input unit, u_1.

Product quality: Vendors offer different prices for same product with different qualities. For instance, a consumer can get a laptop (Latitude E5500) from Dell at two different prices as portrayed in Figure 8 and Figure 9. Dell offers a Latitude E5500 laptop at CDN 1160.00 [41]. However, the same laptop with higher battery quality or battery life time can be bought from Dell at CDN 1258.00 [41]. A businessman, who travels more compared to a student, would prefer to pay more for the sake of better battery life time.

The second input unit, u_2, of our network accepts product price based on product quality as the input, a_2.

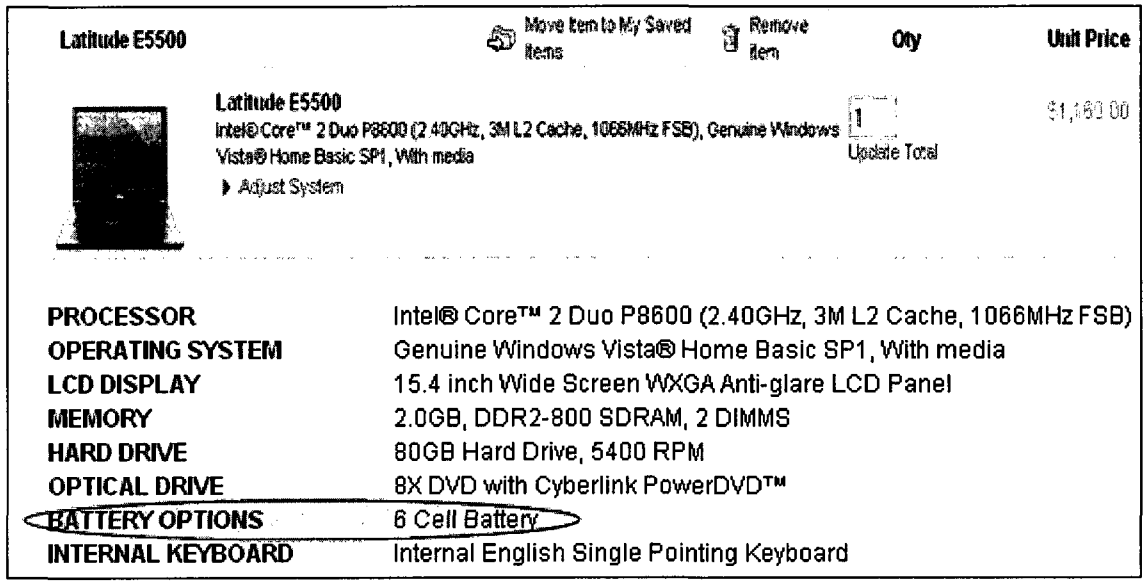

Figure 8: Price offered by Dell for a Latitude E5500 laptop with low quality of battery.

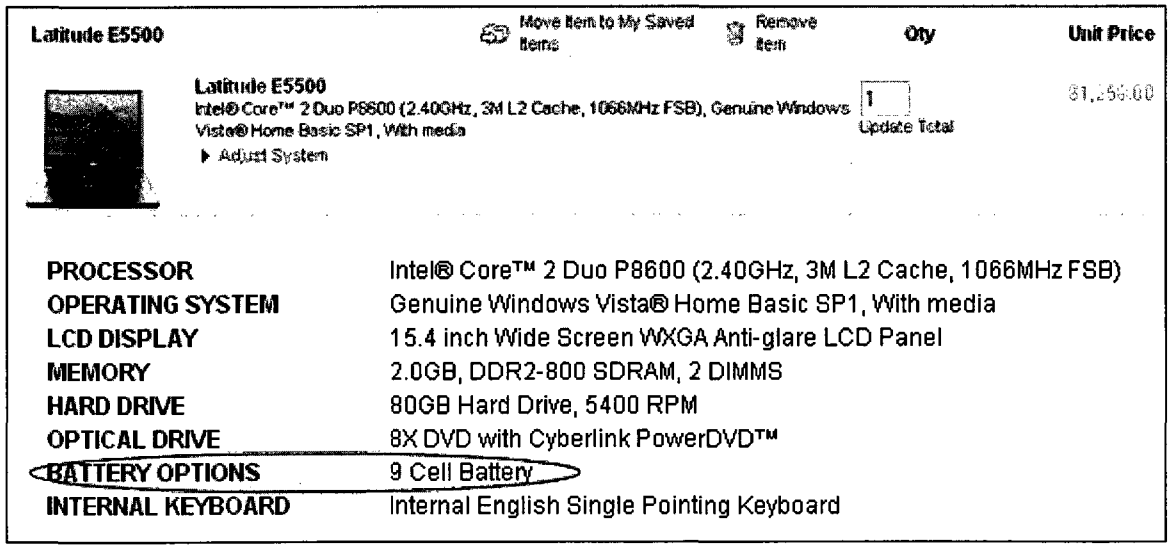

Figure 9: Price offered by Dell for a Latitude E5500 laptop with high quality of battery.

Delivery time: If a buyer's preferred attribute is delivery time, then the seller would deliver the product as soon as possible. In such case the sellers might have to go for express mail option in sending the desired product to the buyer. Therefore, the seller may add some extra money to the cost of the product so that the expense of the express mail does not affect the sellers' revenue. Since the "delivery time" attribute is taken care of by the third input unit, u_3, the seller would set the value of a_3 to some price x such that the revenue is not affected by the extra cost of express mailing service. For instance, the price of a Nikon D3000 DSLR camera set by Future Shop is CDN 799.98. However, the vendor charge an additional cost of CDN 14.87 (Figure 10) if a consumer wishes the product to be mailed in faster means (air) so that it can be delivered in less time. The third input unit, u_3, of our network receives price of the product based on delivery time.

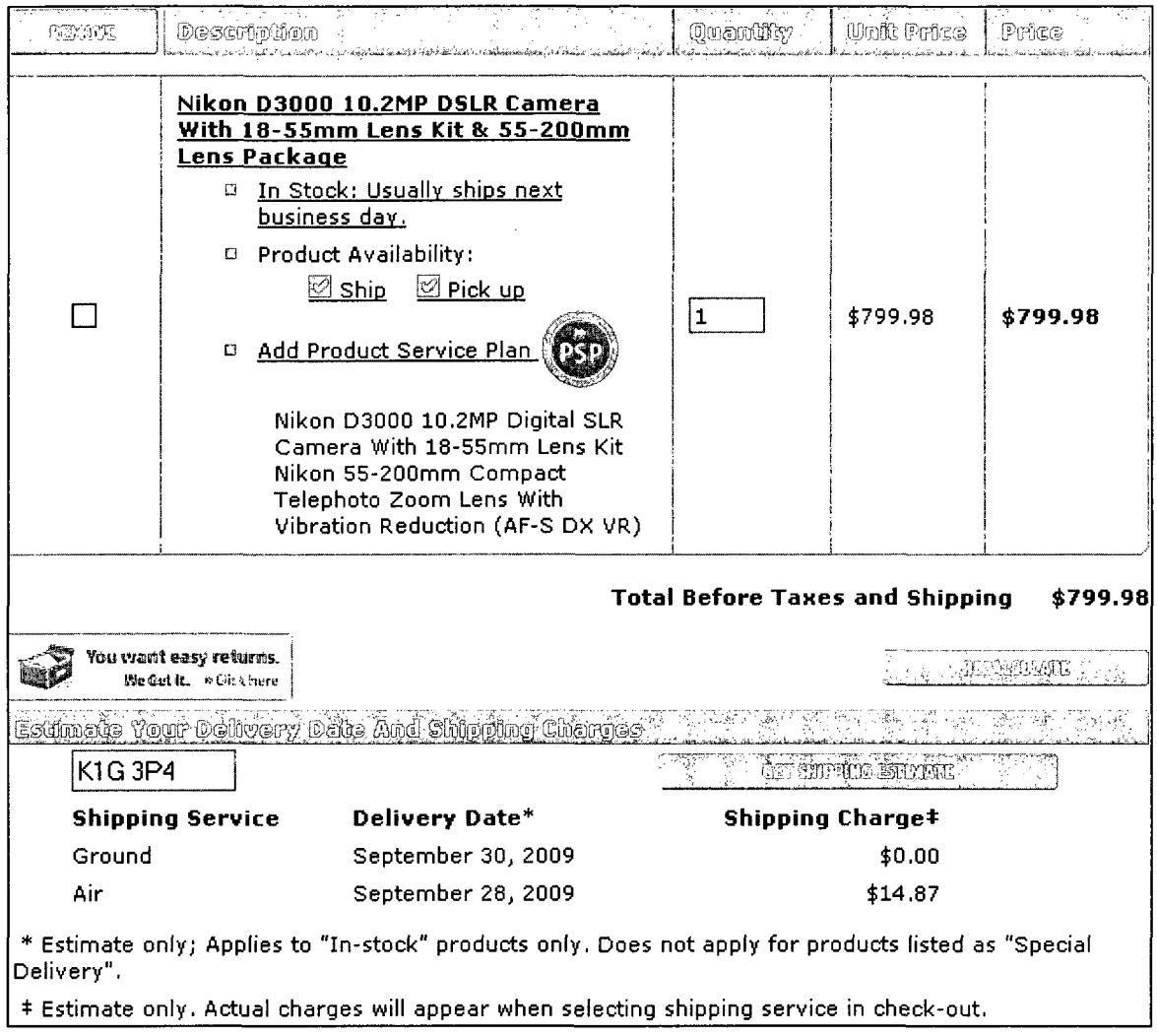

Figure 10: Future Shop charges additional price for faster delivery.

After sales service: To facilitate a buyer with after sales services it is a common practice for the vendors to charge a specific amount in addition to the selling price of a product. For example, as shown in Figure 11 the price of a "Sony Vaio (VGNNW130DT)" laptop offered by Future Shop is CDN 979.99 [39].

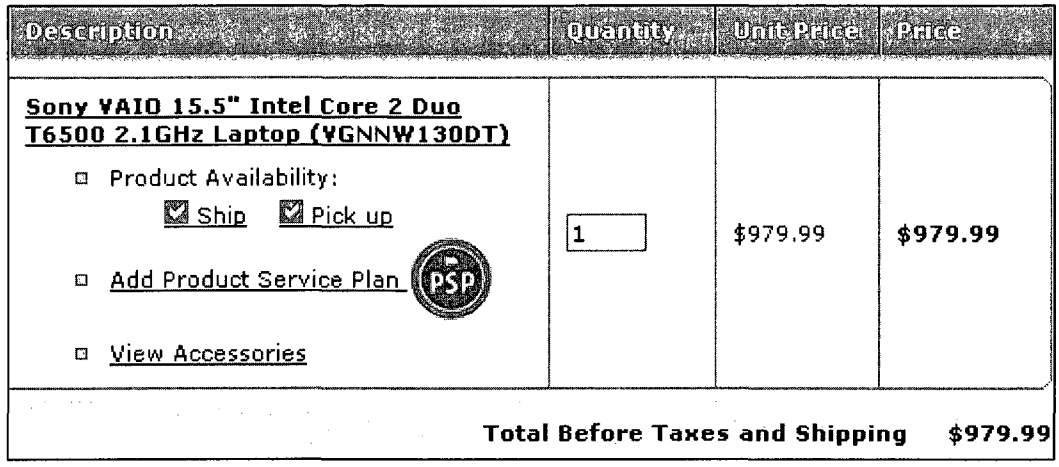

Figure 11: Price offered by Future Shop for a Sony Vaio (VGNNW130DT) laptop.

However, if a consumer would like to enjoy warranty for additional two years, then Future Shop charges additional CDN 274.99 for adding on-site product service plan for two years (Figure 12). That is, total price of the laptop including the mentioned after sale service would be CDN 1254.98 [39]. The fourth input unit, u_4, of our network accepts a price of a product based on after sale service attribute as input, a_4.

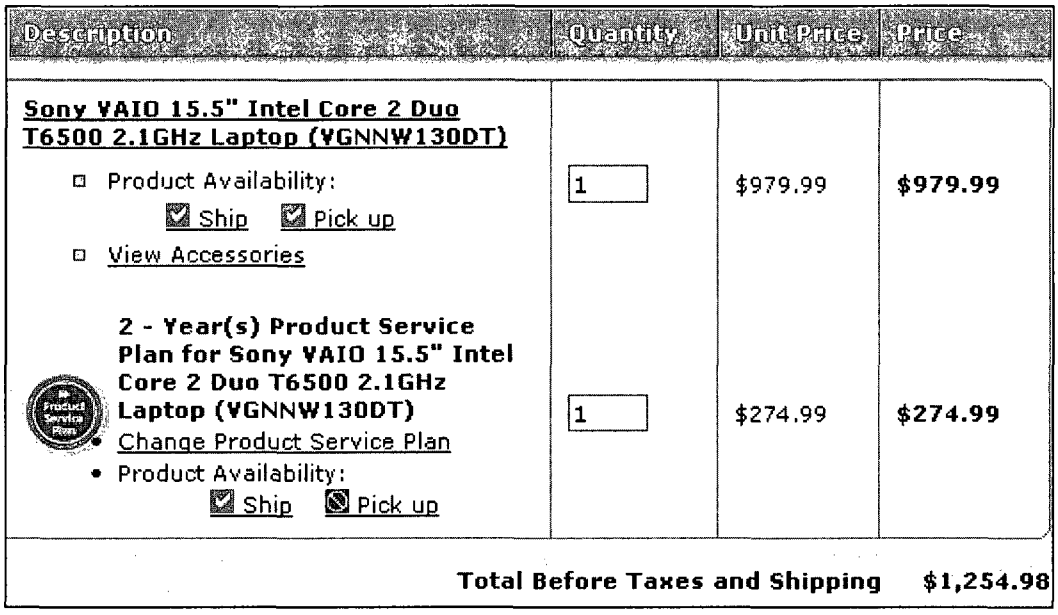

Figure 12: Additional price charged by Future Shop for a Sony Vaio (VGNNW130DT) laptop with two years service plan.

Seller's reputation: Product price may vary with a particular seller's reputation. In online market since buyers have to pay the fare of purchased product before delivery, they run a risk of trusting seller's integrity. Before purchasing a product, a consumer carries a certain level of expectations regarding the products quality [38]. In online shopping transaction it is difficult for a buyer to measure quality of the purchased goods well ahead of delivery. The buyers, hence, have to rely on the products' information provided by sellers. However, after receiving the purchased product, the buyer may find that the seller has provided asymmetric information of the products. In such scenario seller reputation becomes an important means of reliance [35, 38] before outcome of the transaction. Based on his/her reputation, a seller may or may not add some extra money to the cost of the product. We can find enormous examples on online where two sellers with different

reputation values offering distinct prices for a similar product. For example, at *eBay* a BL-4C battery for Nokia 6100 cell phone is offered at two distinct prices by two sellers of different reputation (Figure 13). A seller with 99.1% users' positive feedback asks little higher price (CDN 3.51) than a price (CDN 2.93) offered by a seller with 98.5% users' affirmative feedback [41]. Like all other input units, the fifth input unit, u_5, of our network accepts product price based on sellers reputation as input, a_5.

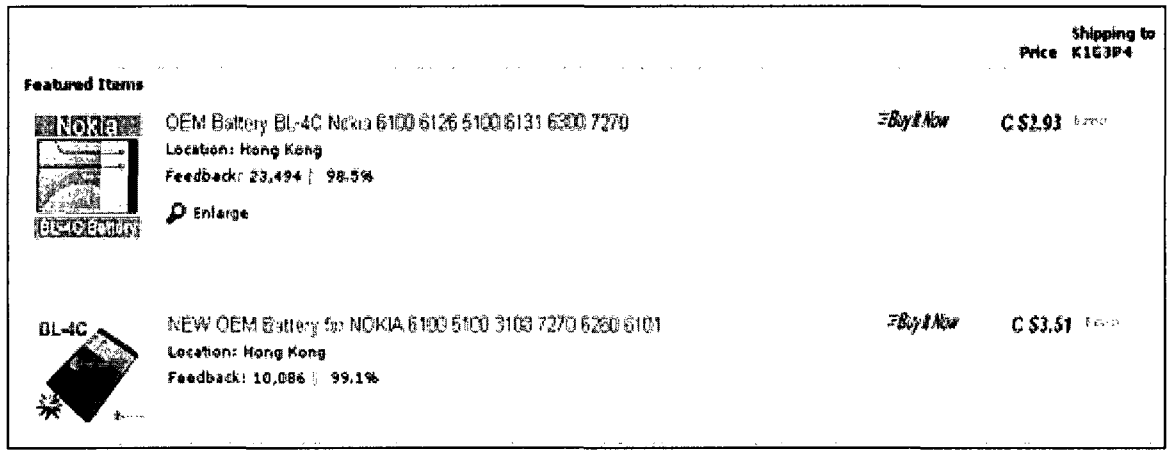

Figure 13: Price offered for a BL-4C cell by two sellers of different reputation at eBay.

In a nutshell, the input layer of our model consists of separate units for the five purchase attributes of product, namely product price, product quality, delivery time, after sales service and sellers' reputation. Based on each purchase attribute, we provide the selling price of a particular product as input to the corresponding unit.

Please note that our model can accept more attributes. One additional unit in the input layer needs to be added for each new attribute. On the other hand, in order to remove an attribute from the network the corresponding unit from the input layer, along with all the

links that are connected to the unit, has to be eliminated. This implies that our model will work for any number of attributes. In addition, our model is not limited to these five attributes. The model can accept any other attributes as input. We have taken these five attributes to present a clear concept about how an attribute would look like.

3.3.3 Units of the Hidden and Output layer

The number of units in the hidden layer cannot be well defined in advance. The number keeps changing from one network configuration to another. In our model we used three units in the hidden layer. From our experimental evaluation we found that the outputs are relatively the same for the network with three hidden units and four hidden units; however, the evaluation with four hidden units takes more execution time than that with three hidden units. On the other hand, network with three hidden units performed better than network with two hidden units. Therefore, we chose three hidden units in our experimental evaluation. We may increase/decrease the number of hidden units based on the performance of our experimental evaluation. The output layer of our model consists of a single unit which provides a competitive price of the product as output.

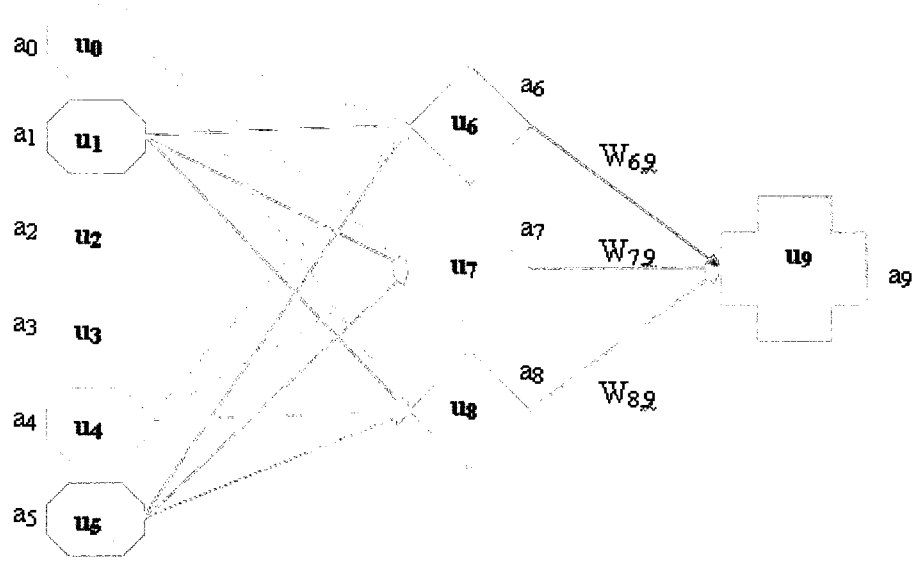

Figure 14: A three-layered feed-forward Neural Network for price determination.

3.4 Dynamic Pricing Algorithm

The price of the product determined by our network (Figure 14) can be found by using the final output a_9. As indicated in Section 2.1.1, we use the logistic sigmoid function as the activation function and the final output a_9 can be calculated with the aid of equation (1) as follows:

$$a_9 = f(W_{6,9}a_6 + W_{7,9}a_7 + W_{8,9}a_8) \qquad (6).$$

where,

$$a_6 = f(W_{0,6}a_0 + W_{1,6}a_1 + W_{2,6}a_2 + W_{3,6}a_3 + W_{4,6}a_4 + W_{5,6}a_5)$$

$$a_7 = f(W_{0,7}a_0 + W_{1,7}a_1 + W_{2,7}a_2 + W_{3,7}a_3 + W_{4,7}a_4 + W_{5,7}a_5)$$

$$a_8 = f(W_{0,8}a_0 + W_{1,8}a_1 + W_{2,8}a_2 + W_{3,8}a_3 + W_{4,8}a_4 + W_{5,8}a_5)$$

and the logistic sigmoid function f, as the activation function, is given by:

$$f(x) = \frac{1}{1+e^{-x}}$$

Equation (6) can be represented pictorially as Figure 14 where bias unit is set as the production cost of the product. Initially, we assume that the buyers have equal preference on all the five attributes that we are considering. Since there are five links between the input layer's units (except bias unit) and hidden layer's units; we set 0.2 as weight of the links between the input units and the hidden units (5 x 0.2 = 1.0). Similarly, 0.33 is set as weight of the links between the hidden units and the output units; as there are three links between the layers (3 x 0.33 ≈ 1.0).

We sub-divide the process of dynamic pricing of our model using neural network into two phases: training phase and price determination phase. In the training phase we train our network with a set of training patterns. A training pattern consists of a set of inputs and a desired output. The desired output indicates what the output of a model of neural network ought to be when a specific set of input is given to the network. A typical set of training patterns for our model is depicted in Table 2.

Table 2: General set of training pattern of our model

Product Price	**Product Quality**	**Delivery Time**	**Sellers' Reputation**	**After Sales Service**	**Desired Output**
$Input_{11}$	$Input_{12}$	$Input_{13}$	$Input_{14}$	$Input_{15}$	$Output_1$
$Input_{21}$	$Input_{22}$	$Input_{23}$	$Input_{24}$	$Input_{25}$	$Output_2$
$Input_{31}$	$Input_{32}$	$Input_{33}$	$Input_{34}$	$Input_{35}$	$Output_3$

Each row of Table 2 represents a training pattern which contains a set of inputs and a corresponding desired output. The purpose of the training process is to adjust the weights between the links such that the errors are minimized. To obtain this goal we feed units of the input layer of our network with the corresponding input values ($Input_{ij}$) from each training pattern. We then determine the output from our network and compare it with the corresponding desired output ($Output_i$) of the training pattern to calculate error. Finally, we update weights between the links depending on the calculated errors. In our model, during the process of training, the errors between the links are minimized by using back-propagation technique. The training process can be portrayed by the following steps:

i) Input values from a training pattern to units of the input layer of the network.

ii) If the current training pattern is the first training pattern of the training set, then associate the links between input units and hidden units with equal weight, i.e., 0.2. Also, associate the links between hidden units and output unit equally, i.e., 0.33.

iii) Determine the value from the output layer.

iv) Compute the error, i.e., the difference between desired output of the training pattern and the value obtained in step *iii*.

v) If the error is more than zero then go to step *viii*.

vi) If the error is approximately zero and there is more training pattern left, then take the next training pattern and go to step *i*.

vii) If the error is approximately zero and there is no more training pattern left, then terminate the training process.

viii) Update the weights of the links using back-propagation technique to minimize the error.

ix) Go to step *iii*.

The training phase updates weights between the links of the network as needed so that it can provide better output. Once the training process is complete, our model of network is ready to determine a competitive price for a product, P, from the price determination phase as follows:

i) Set the production cost of the product, P, as the input to bias unit of the input layer and set the weights of the links associated with bias unit[3] to 1.

ii) Set the values (a_i) of the input units for the corresponding purchase attributes of product (as mentioned in Section 3.1) by using prior knowledge about the prices of product offered by other competing sellers.

iii) Run the network and derive the price from the output layer.

iv) Set the price from the output layer as the product price.

[3] As mentioned in Section 2.1.1, every product has its own production cost below which sellers are unwilling to sell the product. Therefore, we are setting the production cost of the product as the output of the bias unit. In addition, we set the numerical weight of the links associated with the bias unit to 1 such that our model would never provide the price of a product below its production cost.

In step *ii* while setting the values of input units for the corresponding attributes, we assume that the sellers use their prior knowledge about the prices offered by other competitive sellers. A seller may employ a web crawler or a web-spider that browse the World Wide Web in methodical, automated manner to harvest up-to-date information about the price offered by other competitive sellers in the market. Search engines like Google make use of web-spidering for providing up-to-date data. We kept the inclusion of such a web-crawler to our model as a future work.

3.5 Error Minimization and Price Determination

Initially, the error size may be large depending on how the initial weights of the links and the values of the input units are chosen. The error is minimized at each iteration from step *iii* through step *ix* of the training phase. Once the price of a specific product is determined from the output layer from step *iv* of the price determination phase, the weights of the links remain unchanged. In step *ii* of price determination phase we assume that sellers use their prior knowledge of price offered by other sellers in the market. This indicates that our model keep an eye on the competitive price set by other sellers in the competing market.

In dynamic online economy, the price of products keeps on changing with the tick of clock. In order to sustain in the competitive online economy a seller needs to update his/her price in response to price fluctuation by other competing sellers in the market.

While updating the price by using our model, we go through the training and price determination phases of our model to recalculate the price. Before recalculating the price we analyze the revenue earned by using the selling price, *Pr*, for the product, *P*, that was generated from our model.

If the revenue earned is greater than zero, i.e., if the seller earned at-least some revenue by using *Pr*, then in step *ii* of training phase instead of taking 0.2 as the weight, $W_{i,j}$, between the input units and the hidden units, we use the weights, $W_{i,j}$, that were determined during the last iteration of the training phase at the time of determining *Pr* and go through the process again. For instance, assume that the value of $W_{1,6}$ was 0.38 when the product price was determined from step *iv* of price determination phase. In such scenario, we would like to set the value of $W_{1,6}$ to 0.38 instead of 0.2 in step *ii* of training phase and run the process again. Moreover, at the time of determining *Pr* we store the values of input units from step *i* and values of output unit from step *iv* of price determination phase as the historical data. We use this historical data as an additional training set during the training phase. We assume that all the training patterns used in the training phase are derived from historical data.

On the other hand, if there was no revenue earned then the entire process is run by providing a new set of inputs in step *ii* of price determination phase. In providing new set of inputs, we suggest that a seller choose slightly lower values than the set of inputs that were used most recently.

3.6 Implementation of the Algorithm

For implementing the algorithm described in section 3.2, C++ programming language has been used. As described in the earlier subsections of 3.1, our model contains five input units in the input layer including a bias unit, three units in the hidden layer and one unit in the output layer.

3.6.1 Structure of Each Unit

A unit of each layer consists of three properties. They are:

1. Number of input links from previous layer (N),
2. Weight of incoming links (W) and
3. Activation value (AV).

"*Number of input links from previous layer (N)*" of a unit *u* indicates the total number of incoming units from immediate previous layer of the network that is connected to *u*. Each of these incoming units is associated with some weight value which is denoted by "*weight of incoming links (W)*". The "*activation value (AV)*" implies net output of the unit.

The following is a snapshot of the structure of a unit or node of each layer in our model.

```
class TNode{
      int nInputLinks;  // number of input links from previous layer
      float *weights;   // weights of incoming links
      float aValue;     // activation value
public:
      TNode();
      ~TNode();
      void setNumInputLinks(int num);
      int getNumInputLinks();
      void setWeights(std::vector<float> wts);
      float* getWeights();
      void setValue(float val);
      float getValue();
};
```

3.6.2 Generation of the Network

Our model contains five input units, three hidden units and a single output unit. We generate the network of our model as follows:

```
genLayer(nInputNodes,nHiddenNodes,nOutputNodes,epochs,lRate,epsilon);
```

The first three parameters of the function indicate the number units in the input layer, hidden layer and output layer respectively. The fourth parameter (*epochs*) is the number

of iteration that will be carried out while training the network. Epoch is the maximum number of times the complete data set of training pattern can be used by the network for training. We will discuss about the training pattern in the following chapter. The fifth parameter tells about the learning rate of the network. The learning rate determines by what amount the weights between the links are updated at each step of iteration during training period. The last parameter specifies the amount of error that can be tolerated by our network in the learning phase.

3.6.3 Initialization of Weights Vector

At the initial stage of training the network, we set the weights between the links of the layers equally. We had the intention to give even priority to all the attributes that we discussed. Since there are five input units and three hidden units, each hidden unit is connected to five of the units from the previous layer (input layer). Consequently, we associate each link between input units and hidden units with equal weights, i.e., 0.2. Similarly, the single unit of the output layer is linked to three of the units from hidden layer hence we associate an equal amount of 0.33 as weight to each link between hidden units and out unit.

The content of initial weights vector looks as follows:

0.20
0.20
0.20
0.20
0.20
0.33
0.33
0.33

The initial weights vector contains weights of the links between input units and hidden units followed by weights of the links between hidden units and output unit. The contents of the weights vector are then distributed among appropriate links between the layers as follows:

```
void TLayer::setHiddenLayerWeights(std::vector<float> wts){
      for(int i = 0; i < nInputNodes; i++)
            HLW.push_back(wts.at(i));
}

void TLayer::setOutputLayerWeights(std::vector<float> wts){
      for(int i = nInputNodes; i < wts.size(); i++)
            OLW.push_back(wts.at(i));
}
```

Here, both the functions taking initial weights vector (*wts*) as input parameter. The initial weights of the links between input units and hidden units are stored in HLW vector,

whereas, the weights of the links between hidden units and output unit are located in OLW vector.

3.6.4 Activation Function

Identity function, step function, logistic sigmoid function and symmetric sigmoid function are some of the well known activation function that is being used in neural network. Among these functions, symmetric sigmoid provides higher range of output. However, it provides negative values. Since our model of neural network produces a competitive price of a product as output of the network and the price of a product cannot accept any negative value, we have chosen logistic sigmoid function as the activation function.

```
float TLayer::sigmoid(float x){
      return 1/(1+exp(-x));
}
```

3.6.5 Train Network

The goal of our model is to provide a competitive price for the products so that the revenue earned by the sellers is increased. Setting the correct weights of each link among different units of all the layers plays the most vital role in neural networks. Therefore, the objective here is to choose weights for each link as accurately as possible such that the

network performance is optimized. This can be done by training the network by a set of predefined training pattern. A training pattern consists of a set of inputs with desired output. We train the network by using same sets of training patterns over and over again. After each iteration in training the network, we evaluate the errors by finding the difference between desired output and actual output of the network during that specific iteration. A sample code of error evaluation is given below.

```
void TLayer::evaluateErrors(float targetVal){
      float error;
      // update output layer weights
      oError.clear();
      for(int i = 0; i < nOutputNodes; i++){
            error = (sigmoid(targetVal - prCost) -
output[i].getValue()) * output[i].getValue() * (1 -
output[i].getValue());
            oError.push_back(error);
      }

      // update hidden layer weights
      float eSum = 0.0;                    // sum of errors
      hError.clear();
      for(i = 0; i < nHiddenNodes; i++){
            for(int j = 0; j < nOutputNodes; j++)
                  eSum += oError.at(j) * OLW.at(i);
```

```
            error = eSum * hidden[i].getValue() * (1 -
hidden[i].getValue());
            hError.push_back(error);
      }
}
```

We first calculate the amount of errors in output layer by calculating the difference between final output of output layer and target output of the training sets. Since the single unit of output layer is connected to each of the unit of hidden layer, hidden layer units are partially responsible for the errors found in output layer. We, therefore, propagate the calculated error of output layer back to hidden layer units to figure out the errors presented in the hidden layer. The errors in output layer and hidden layer are stored in *oError* and *hError* error vectors respectively.

These calculated errors are used to update the weights of all the links to minimize the errors. The errors calculated in the output layer are utilized to update the weights of the links between hidden units and output unit. The updated weights are then stored in OLW vector. The errors found in hidden layer are used to update the weights of the links between input units and hidden units. Consequently, the content of HLW vector is updated.

```
void TLayer::updateWeights(){
	// update output layer weights
	for(int i = 0; i < nOutputNodes; i++){
		for(int j = 0; j < nHiddenNodes; j++)
			OLW.at(j) = OLW.at(j) + lRate * oError.at(i) *
hidden[j].getValue();
	}

	// update hidden layer weights ei * wij
	for(i = 0; i < nHiddenNodes; i++){
		for(int j = 0; j < nInputNodes; j++)
			HLW.at(j) = HLW.at(j) + lRate * hError.at(i) *
input[j].getValue();
	}
}
```

After updating the weights of all the links, the network is run again. We keep on running the network over and over again to train the network until the errors found in the output layer is below some threshold value specified by *epsilon*. Following is a snapshot of train network function.

```
void TLayer::trainNetwork(){
	std::vector<float> trainingSetVal;
	float targetVal;
	std::fstream inputTrainingSet;
```

```
	inputTrainingSet.open("trainingSet.txt");


	if(!inputTrainingSet)
		std::cout<<"ERROR: Training set file named trainingSet.txt
not found.\n";
	else{
		float t;
		while(true){
			for(int i = 0; i < epochs; i++){
				while(!inputTrainingSet.eof()){
					for(int j = 0; j < nInputNodes + 1; j++){
						inputTrainingSet>>t;
						trainingSetVal.push_back(t);
					}
					inputTrainingSet>>targetVal;
					setInputLayerInputs(trainingSetVal);
					runNetwork();
					evaluateErrors(targetVal);
					updateWeights();
					trainingSetVal.clear();
				} // end while
				inputTrainingSet.clear();
				inputTrainingSet.seekg(0,std::ios::beg);
			} // end for
			if(oError.at(0) < epsilon)
				break;
		}// end while
```

```
        } // end else
        inputTrainingSet.close();
}
```

3.6.6 Set Input Layer Units

Once the training of the network is done we are ready to run the network to provide a competitive selling price of the products for the sellers. The first step to run the network is to set input to the units of input layer.

```
void TLayer::setInputLayerInputs(std::vector<float> inputVal){
        prCost = inputVal.at(0);
        for(int i = 0; i < nInputNodes; i++)
                input[i].setValue(inputVal.at(i+1));
}
```

The function accepts a vector of input values as an input parameter which contains the input values of each units of the input layer.

3.6.7 Run Network

After setting input values to all units of the input layer we run the network to calculate the outputs of each unit of the hidden layer. The output from the units of hidden layer is

then used to derive the final output of the network. This final output represents the selling price of the products determined by the network. The following is the sample code for running the network.

```
void TLayer::runNetwork(){
	// hidder layer
	float netInput;
	for(int i = 0; i < nHiddenNodes; i++){
		netInput = prCost;
		for(int j = 0; j < nInputNodes; j++){
			netInput += sigmoid(input[j].getValue()) * HLW.at(j);
		}
		hidden[i].setValue(sigmoid(netInput - prCost));
	}

	// output layer
	for(i = 0; i < nOutputNodes; i++){
		netInput = prCost;
		for(int j = 0; j < nHiddenNodes; j++){
			netInput += hidden[j].getValue() * OLW.at(j);
		}
		output[i].setValue(sigmoid(netInput - prCost));
	}
}
```

3.6.8 Get Product Price

The aim of our model was to determine a competitive selling price for the products that enable sellers to increase their revenue. The final output of the network provides the competitive price of the products. In calculating the product price our model considered five attributes that would determine buyers purchase decision. As mentioned before, our model is general enough to work for any number of attributes.

```
float TLayer::getProductPrice(){
      std::cout<<output[0].getValue()<<"\n";
      return - log(1/output[0].getValue() - 1) + prCost;
}
```

Chapter 4: Test Application and Experimental Evaluation

In this thesis a test application is implemented to experiment the applicability and performance of the designed model. The implemented system is divided into two subsystems.

1. Agent module – for training the network.
2. Client module – to receive competitive price of a product from the network.

The agent module is intended for the system. The agent module is responsible for the following:

- Setting up the neural network by determining
    i. Number of input units in the input layer,
    ii. Number of hidden units in the hidden layer,
    iii. Number of output units in the output layer,
    iv. Number of iteration (epochs) used in training the network,
    v. Amount of errors (epsilon) allowed during training and
    vi. Learning rate of the network while training.
- Generate the network by using the above six information pieces.
- Set the weights of the links between input layer and hidden layer.

- Set the weights of the links between hidden layer and output layer.
- Train the network by using a set of training patterns.

The client module is meant for sellers. The sellers are sole user of the client module. The client module is responsible for the following:

- Set the inputs for all the units of input layer.
    - It is assumed that sellers use their prior knowledge of the product price set by other competitive sellers.
    - By using the prior knowledge they set inputs to all the units of input layer as described in section 3.1.1.
- Receive a competitive price of the product given by the network.

4.1 Design of the Test Application

Since the system is divided into two subsystems, the design of the test application consists of use cases and sequence diagram for both agent and client. The design also includes class diagram of the application.

4.1.1 Use-Cases of Agent Module

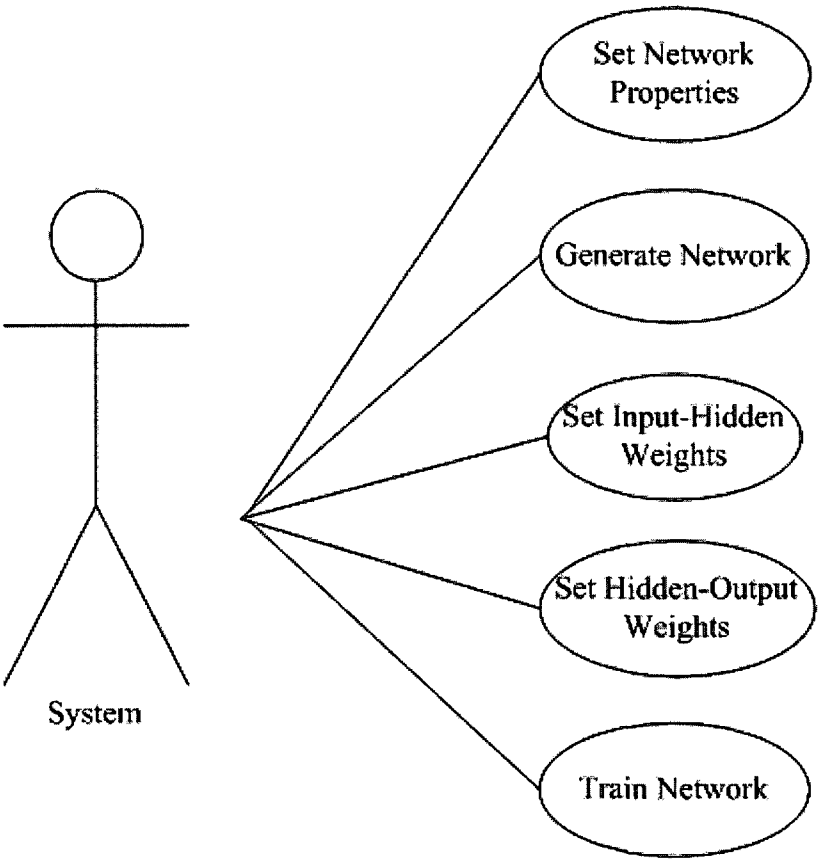

Figure 15: Use-Case diagram for Agent module of test application.

The use-cases are described below:

Use-case: Set Network Properties.

Actor: System.

Description: The system prepare the initial setting of the network by determining the number of input units, hidden units and output units in the input layer, hidden layer and

output layer respectively. In addition, how much iteration would be carried out during training the network is also defined. The system also identifies the tolerable errors by the network. Finally, during the training process what would be the learning rate of the network is classified.

Use-case: Generate Network.

Actor: System.

Description: The system generates the network by following the initial set up of the network.

Use-case: Set Input-Hidden Weights.

Actor: System.

Description: The system initializes the weights of each link between input layer and hidden layer. The weights among the links are distributed equally during initialization.

Use-case: Set Hidden-Output Weights.

Actor: System.

Description: The system initializes the weights of each link between hidden layer and output layer. The weights among the links are distributed equally during initialization.

Use-case: Train Network.

Actor: System.

Description: The system trains the network with the help of a set of training pattern. The weights of the links are updated in the training process in order to minimize the errors.

4.1.2 Use-Cases of Client Module

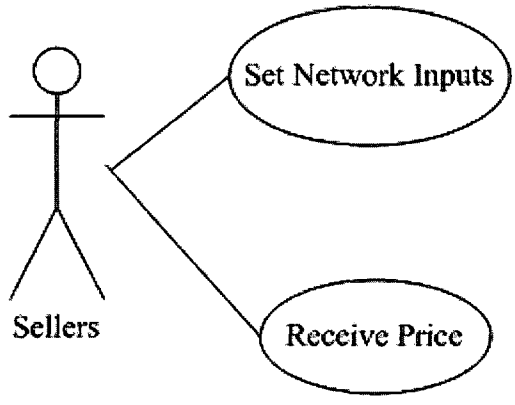

Figure 16: Use-Case diagram for Client module of test application.

The use-cases are described below:

Use-case: Set Network Inputs.

Actor: Sellers.

Description: The sellers set the initial inputs to the network by assigning each unit of the input layer with a value. In setting the initial values, sellers use their prior knowledge of how other competitive seller set the price of the product.

Use-case: Receive Price.

Actor: Sellers.

Description: The sellers receive a competitive price from the network.

4.1.3 Sequence Diagram of Agent Module

Figure 17 portrays the sequence diagram for Agent module of the application. The main objective of the system is to train the network. The system first sets the configuration of the network by determining number of units in different layers, tolerable errors (epsilon), learning rate and number of epochs used in learning phase. The network is then generated by following the given configuration. After creation of the network, the system initializes weights of all the links between the layers. As a final step, the system trains the network by using a set of training patterns. The same set of training patterns are run for several iterations. The amount of errors is calculated at each iteration. The weights of the links are then updated depending on the calculated errors. The training process continues until the amount errors do not fall under the specified tolerable errors.

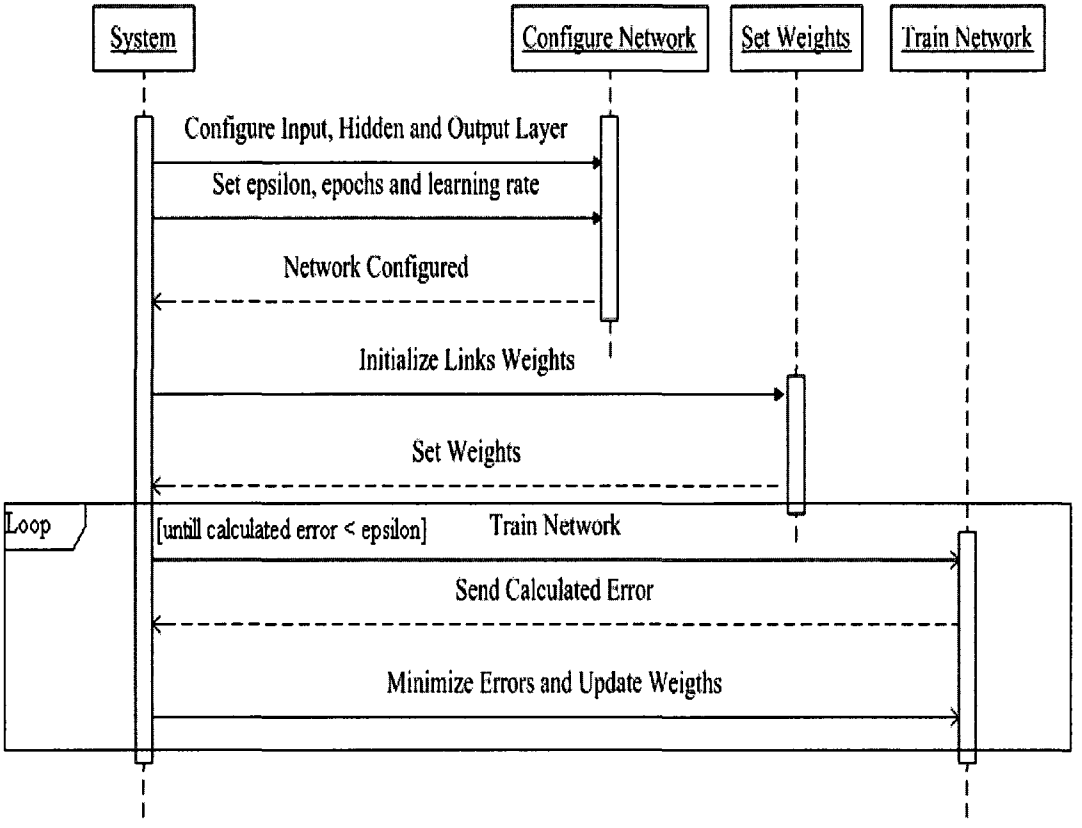

Figure 17: Sequence Diagram for Agent Module.

4.1.4 Sequence Diagram of Client Module

Figure 18 illustrates the sequence diagram for Client module of the application. Once the network has been trained by the system, the sellers are ready to request a competitive price of the products from the system. The sellers first set the initial price of the product based on different attributes as described in section 3.1.1. The sellers then request the system for a competitive price of the product. Based on the initial price set by the sellers, the system computes a price for the product. The computed price is then sent to the sellers.

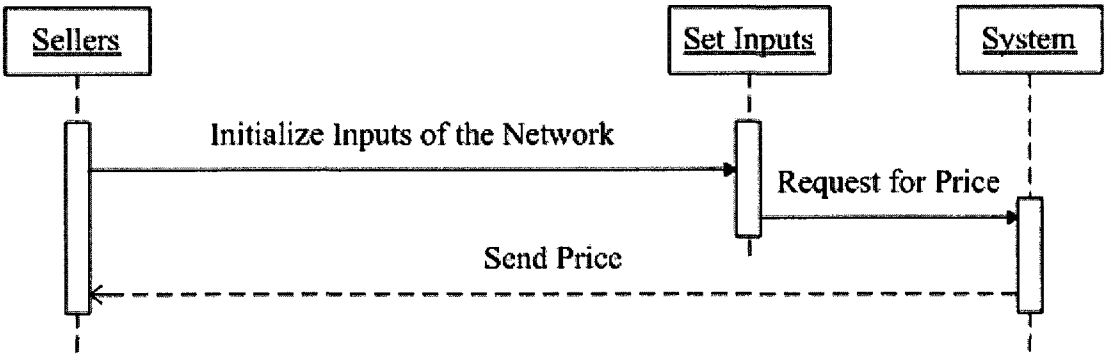

Figure 18: Sequence Diagram for Client Module.

4.1.5 Class Diagram

As mentioned earlier, the test application consists of two subsystems: Agent and Client. The system is the sole user of the Agent module, whereas, the sellers are the only user of the Client module. Figure 19 demonstrates the class diagram of the test application.

The main purpose of the system is to train the network in order to minimize the errors such that a competitive price of the products can be obtained by using the network. In training the network the system first set the configuration of the network by **Utility::main()**. Once configuring the network, the system calls **TLayer::TLayer(...)** to generate the network. The method accepts configuration of the network as input parameters. The weights of the links between input layer and hidden layer is then initialized by using **TLayer::setHiddenLayerWeights(...)**. Similarly, TLayer::setHiddenLayerWeights(...) is called to initialize the weights of the links

between hidden layer and output layer. After all the initialization has been done, the system starts training the network with the aid of **TLayer::trainNetwork()**.

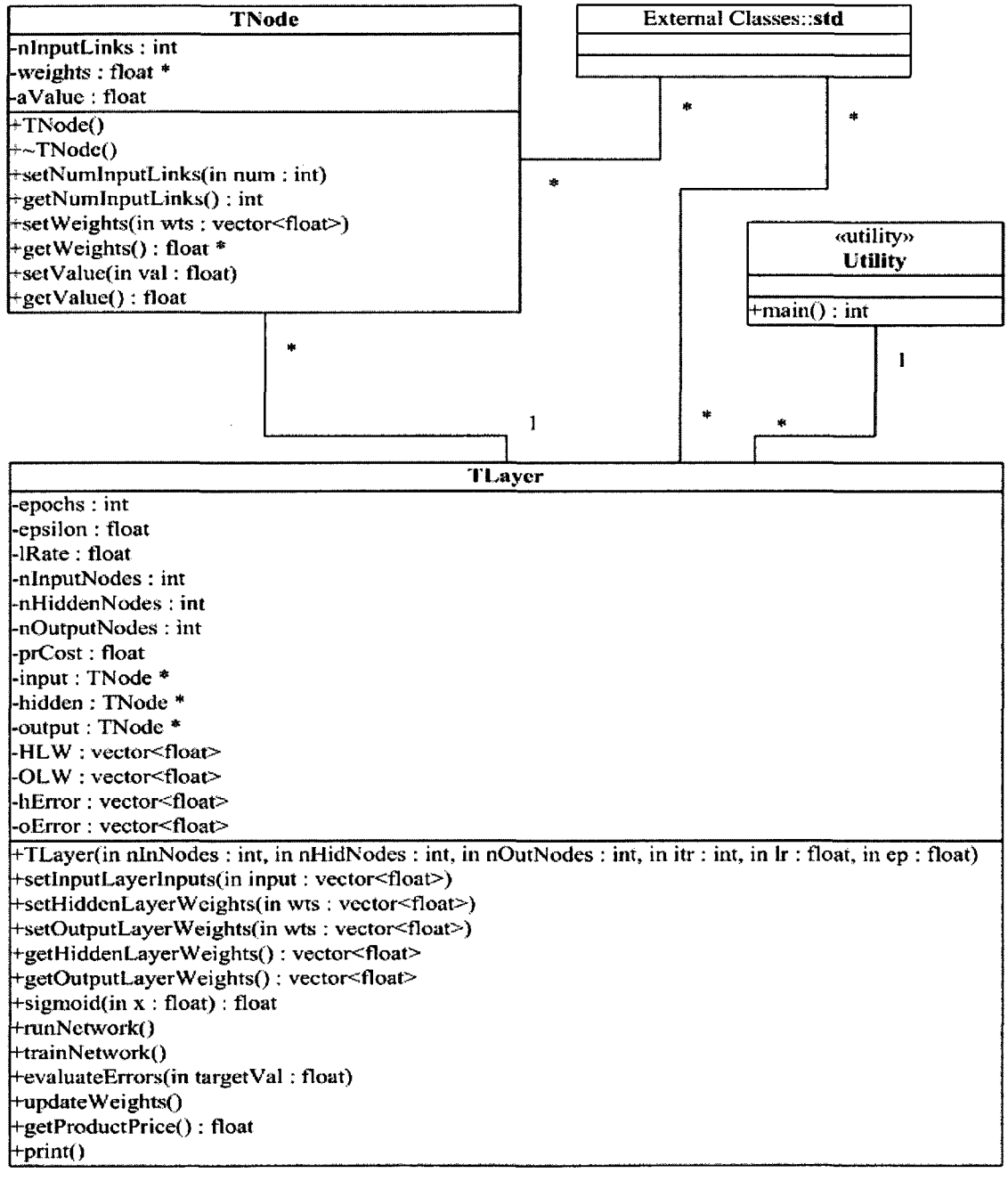

Figure 19: Class Diagram of the application.

After the network has been trained by the system, the network is in a position to generate a competitive price for the products. Before making a request to the system for a price, the sellers first set the initial price of the product by calling **TLayer::setInputLayerInputs(...)**. At this point the sellers call **TLayer::getProductPrice()** to ask the system for generating a competitive price of the product based on the given initial values. The system then calculate a price by using **TLayer::runNetwork()** and return the price to the sellers.

4.2 Experimental Results

For evaluating the performance of the application, the test application has been run on a desktop computer with the following configuration:

Operating System:	Microsoft Windows XP version 2002 SP2
Processor:	AMD Athlon™, 1.10 GHz
RAM:	512 MB

Our network is run with five input units, three hidden units and one output unit. We have gone through an experimental evaluation of our model in an e-commerce market place to examine if the model performs better than the simple pricing algorithm outlined in Section 3.1. We also analyzed if a seller earns more revenue by employing our model instead of the Derivative-Following (DF) strategy proposed in [9]. In derivative following

(DF) strategy, initially, product prices are set randomly and profitability is observed. The product prices are increased in the same direction unless the observed profitability falls. If the observed profitability falls then product prices are decreased as long as profit is encountered. It requires keeping track of past average profit of each state, and increases the prices till the profitability level falls [9]. In other words, DF follows a strategy of changing the price in same direction if revenue earned by following previous price change is high. The direction of changing price is reversed otherwise.

4.2.1 Train Network

We assume that sellers use their historical data as the training patterns to the network. We began our experimental evaluation by training the network of our model with 10 sets of training patterns. A training pattern consists of a set of inputs with desired output. We used the following sets of training patterns from Table 3 to train our network so that errors can be minimized as much as possible by using back propagation algorithm. We obtained the values in Table 3 by using prices offered by Future Shop and Best Buy based on different attributes of distinct electronic goods as of November 16, 2009. This implies that all the patterns are not of same product. For instance, the first row represents the price offered by Future Shop for a Nikon D5000 12.3MP Digital SLR Camera", whereas, second row shows the price for a "Samsung 22" LCD HDTV" offered by Best Buy. From our daily life example we cay say that the preference of purchase attributes varies from product to product. For example, we always look for quality of product

before buying any food item. On the other hand, preference is given to after sales service (extended warranty period etc.) when a buyer purchases any electronic goods. By keeping this fact in mind we have chosen training pattern from diverse products so that different purchase attributes are covered by the training patterns. From each training pattern values that we obtained as shown in Table 3 we can see that among other attributes, price offered based on "Product Price" is the lowest. Since no seller would like to sell his/her product less than the production cost, the prices offered based on different attributes must be greater than production cost. However, Production cost of a product is not revealed by any online seller. Therefore, for the value of production cost of each training pattern we used 1% less value of the price offered based on "Product Price" attribute. In addition, we used a random number generator to obtain a number greater than the production cost of the product in a training pattern and use this number as value of desired output of the corresponding pattern. In generating the value we restrict the random number generator in producing any value which is 25% greater than the production cost. The constraint is used such that the random number generator does not breed a number that is much greater than the production cost which would not be realistic. Mention to be made that in the following table the value of price offered based on "Sellers' Reputation" is not taken from Future Shop or Best Buy as it was not available on the websites. Hence, we took the average value of prices of all other four attributes to populate this particular column.

Table 3: Sets of training patterns used to train the model.

Production Cost	Product Price	Product Quality	Delivery Time	After Sales Service	Sellers' Reputation	Desired Output (Selling Price)
871.1901	879.99	1139.99	895.13	1029.98	986.2725	1066
296.9901	299.99	329.95	329.73	334.99	323.665	321
118.7901	119.99	159.99	129.26	149.98	139.805	138
791.9901	799.99	949.99	810.68	979.98	885.16	966
989.9901	999.99	1099.99	1054.99	1169.98	1081.2375	1128
178.1901	179.99	199.99	198.95	209.98	197.2275	203
1187.9901	1199.99	1299.99	1254.99	1529.98	1321.2375	1233
326.6505	329.95	349.99	340.37	389.95	352.565	375
128.6901	129.99	149.99	139.26	169.99	147.3075	146
1038.51	1049	1147	1104	1248	1137	1211

Initially, we intend to give identical preference to all the five considered attributes. Therefore, we distributed the weights of links between the layers equally. Since, there are five links between the input layer's units (except bias unit) and hidden layer's units; we set 0.2 as weight of the links between the input units and the hidden units ($5 \times 0.2 = 1.0$). Similarly, 0.33 is set as weight of the links between the hidden units and the output units; as there are three links between the layers ($3 \times 0.33 \approx 1.0$). We trained our network for nine different numbers of epochs[4] as mentioned in Table 4.

Table 4: Number of epochs used to train network.

Epochs	10	50	100	500	1000	5000	10000	50000	100000

[4] Epoch is the maximum number of times a complete data set of training patterns can be used by a network for training.

As the training continues, after each epoch, the network calculates amount of error. The calculated error is then used to update the weights of the links by using back propagation algorithm so that error is minimized in the next iteration. Practically the value of error never becomes zero, but approaches zero. We let our network to tolerate an error of amount 0.01 and 0.001. In other words the training phase terminate when the calculated error after an epoch is equal to or less than the value of tolerated error by the network. We run our network with five different learning rates as shown in Table 5.

Table 5: Learning rates used to train network.

Learning Rate	0.01	0.005	0.001	0.0005	0.0001

Analysis of the training process in the following section indicates that the model performs better if we use 50000 epochs with 0.005 learning rate during training the network.

4.2.2 Determine Training Parameters

The purpose of the model is to generate a competitive price for a product with respect to the price offered by other competing sellers in the market. The more number of training patterns are used to train the network in training phase, the better knowledge the model will contain. This would lead to generate a more competitive price. Hence, the

performance of the model depends on the training phase. Besides number of training patterns used, the training process is dependent on three parameters:

1. number of epochs used
2. Learning rate
3. Error tolerance

The better the model is trained, the better performance it provides. Therefore, use of proper values for the above three mentioned parameters plays a vital role in the model's performance. We trained our model by using different values (as mentioned in Section 4.2.1) for these parameters with 10 sets of training patterns (Table 3). Once training process is complete, our model is ready to determine a selling price for a product. We used our trained network to determine the price of a product (lets call it P). We then derive suitable values for the parameters by analyzing performance of the model through investigating the experimental results shown later in this section.

Since our model requires initial selling price of the product to be set by the seller, we used the prices of Table 6 as initial selling price based on five different attributes of the product. We studied the price offered by Future Shop and Best Buy (as on November 17, 2009) for a Sony Vaio 15.4" (VGNNS330DT) Laptop and used their offered price information based on different purchase attributes of the product to populate Table 6. We assumed the production cost of the laptop is 645.00.

Table 6: Initial Price of Product P.

Production Cost	645.00
Product Price	648.99
Product quality	745.99
Delivery time	670.34
After Sales Service	805.99
Sellers' Reputation	718.99

According to our model, the output produced from the output layer of our model is considered as the selling price, *Pr*, at which a sellers, *S*, would be selling *P*. If there are *M* out of *N* buyers in the market willing to buy *P* at a cost of *Pr*, then the revenue earned by *S* can be calculated as the product of *M* and *Pr*.

Figure 20 portrays the amount of revenue earned after selling a product P to a single buyer at the determined price given by the network while the network is allowed to accept an error tolerance of 0.01 during training phase. We also trained our network with an error tolerance of 0.001. Figure 21 depicts the network performance in case of 0.001 tolerable error.

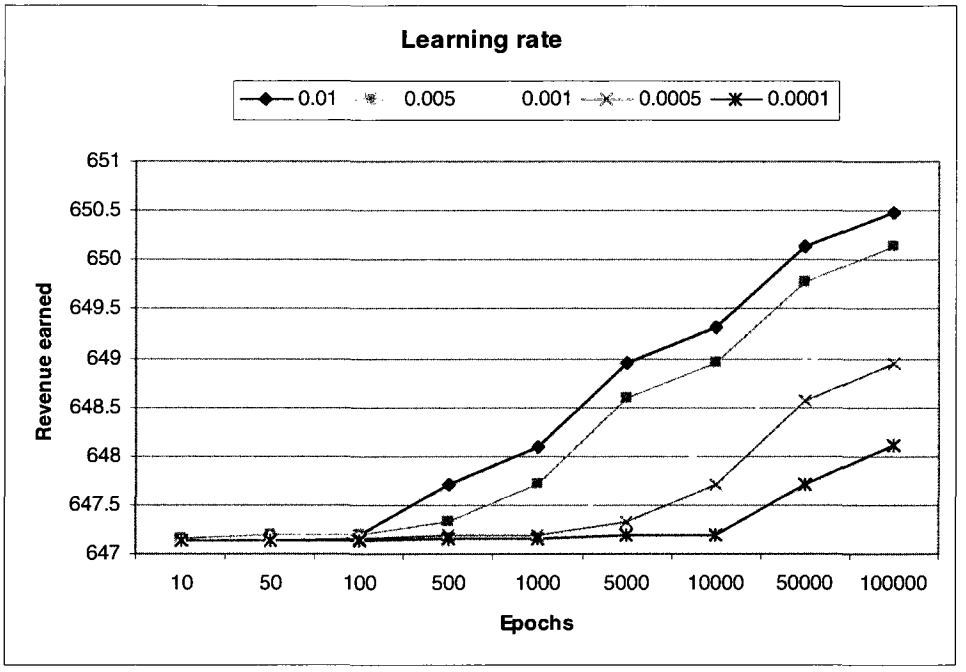

Figure 20: Network performance in terms of revenue earned (Err = 0.01)

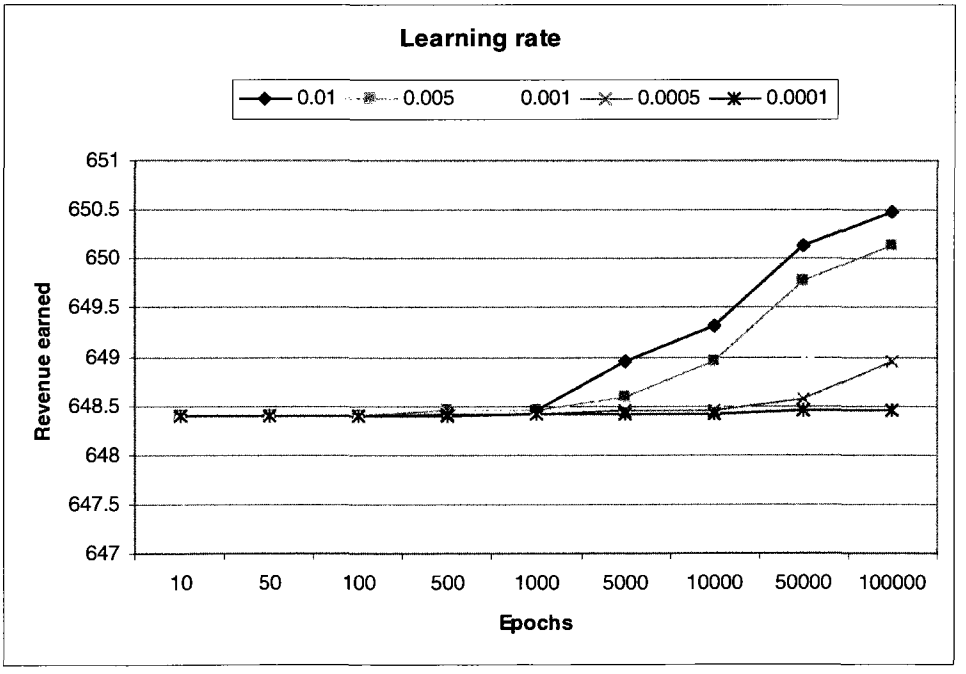

Figure 21: Network performance in terms of revenue earned (Err = 0.001)

While training our model, we can see that the amount of revenue earned per product after selling it to a single buyer is closely identical to each other for different learning rates when lower number of epochs is used. Amount of revenue earned is increased gradually with higher number of epochs used. It can also be noticed that the model performs better if 0.01 is chosen as learning rate compared to other four learning rates of Table 5. Here, one of our objectives was to identify the learning rate by using which a seller can earn more revenue after selling a single product. We figured out that, compared to other four learning rates, if 0.01 is used then better revenue can be earned after selling a single product.

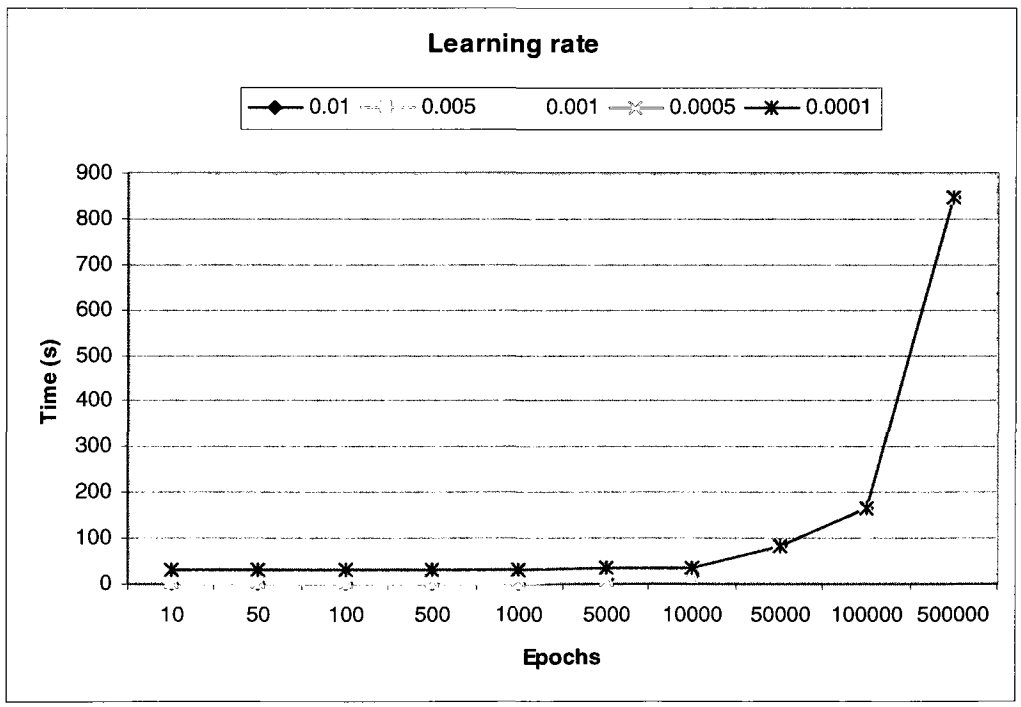

Figure 22: Network performance in terms of elapsed time (Err = 0.01)

Figure 22 indicates that the elapsed time for training our model increases gradually with increasing number of epochs. However, there is a rapid increase in elapsed time after 100,000 epochs. Therefore, we would not like to use more than 100,000 epochs during simulating an e-commerce market place in the following section. While training our model, we let our model to accept an error tolerance of 0.01 and 0.001. It is generally known that a neural network works better if less error tolerance is used [4]. However, using lower than 0.001 of error tolerance makes our training process more time consuming. Moreover, Figure 20 and Figure 21 indicate that from 5000 number of epochs onwards the model delivers similar output for error tolerance of 0.01 and 0.001.

Under the above circumstances of analysis, for training our model during simulating an e-commerce market place in the following section we would like to use 100,000 epochs, 0.01 learning rate and 0.01 error tolerance with 10 sets of training patterns shown in Table 3. Mention to be made that the order of using 10 sets of training patterns has no impact on the result.

Table 7: Training Parameters

Number of Epochs	100000
Learning rate	0.01
Error tolerance	0.01

While performing experimental evaluation, we figured out that our model consumes nearly similar execution time for two hidden units and three hidden units (Figure 22);

however, our model performs better when, instead of two hidden units, three hidden units are used (Figure 21). On the other hand, compared to elapsed time, the margin of difference in our model's outputs for the network with three hidden units and four hidden units are relatively small. Therefore, we chose three hidden units in our experimental evaluation.

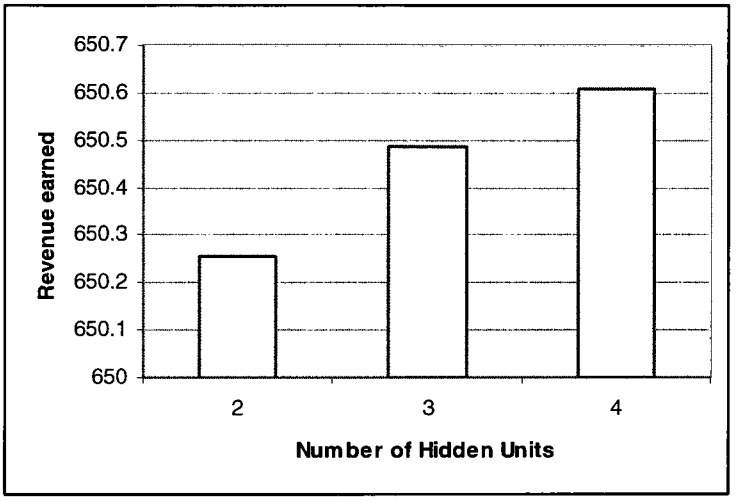

Figure 23: Revenue earned based on number of hidden layers used

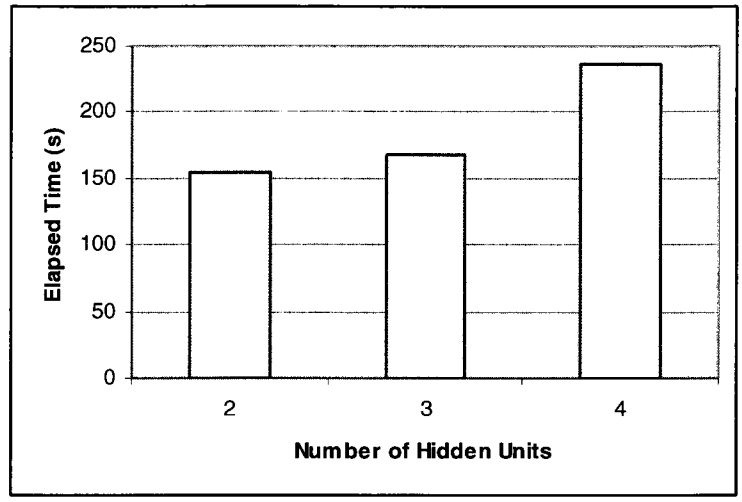

Figure 24: Elapsed time based on number of hidden layers used

4.2.3 Marketplace Setup

We performed an experimental evaluation of our model in an e-commerce market place to evaluate performance of our model. We consider a market place where three sellers (namely, Seller_simple, Seller_DF and Seller_om) wish to sell a product P to 200 different buyers with distinct preferable purchase attributes of products described in Section 3.3.2. Seller_simple employs a **simple** pricing algorithm described in Section 1.3.1. Seller_DF uses **d**erivative-**f**ollowing (**DF**) strategy that was described in Section 4.2 and Seller_om follows **o**ur **m**odel.

We run the market for ten rounds, with twenty buyers in each round. After each round we calculate the revenue earned by each seller. We then compare it with the revenue earned

by the corresponding seller in previous round to determine the direction of revenue earned. If the total revenue earned at the end of current round is strictly greater than the total revenue earned in previous round, then direction of revenue earned is positive. On the other hand, we consider the direction of revenue earned as negative.

At the end of each round, we allow the sellers to update their selling prices. In DF strategies, the price of product is updated, by some amount, in the direction of revenue earned. We consider that Seller_DF updates his/her selling price by a random amount between 0.00% and 5.00% of the current selling price.

On the other hand, Seller_simple updates the selling price either by using direction of revenue earned or by using information of prices set by other two sellers in the market during one of the previous rounds. We made an assumption that simple pricing algorithm performs manual search, which is time consuming, to gather information regarding other sellers' selling price. Therefore, the information may not be available to Seller_simple at the end of each round. In addition, we assume that if the information is available, then due to manual slow searching process the information of the immediate previous round is not available to Seller_simple. For simplicity, we consider that if the information is available, then Seller_simple updates the selling price of P by using average value of the prices set by other two sellers during the $(r\text{-}2)^{th}$ round where r is the current round. In contrary, if the information is not available, Seller_simple uses direction of revenue to update the price. In this case we assume that he/she add/subtract some random value

between 0.00% and 5.00% of the production cost to the current selling price based on direction of revenue earned. In the experimental evaluation we used a randomly-generated probability to determine if the information is available to Seller_simple. If the randomly generated probability is greater than 0.5 then information regarding other sellers' selling price is available to Seller_simple or vice-versa. We populate Table 8 by using this strategy. We then use values of Table 8 regarding information availability of other sellers' selling price to Seller_simple during simulating the market for 10 rounds.

Table 8: Information availability to Seller_simple at the end of each round

Rounds	1	2	3	4	5	6	7	8	9	10
Information Available	No	No	Yes	Yes	No	No	No	Yes	No	Yes

Seller_om always uses our model to update the price. While updating the selling price, we assume that a seller's price of the current round is invisible to other sellers in the market. However, sellers' price of previous round may be visible to Seller_DF and Seller_om.

There are five attributes of the product P based on which a seller can set a selling price for it. Depending on the price offered by the sellers, a buyer chose a seller to purchase the product P. For example, let us say the three sellers offer the prices as shown in Table 9 for a product P. If a buyer wishes to buy P and he/she is looking for a higher quality of product, then according to the offered prices, since Seller_simple offers the lowest price

for the product P when "Product Quality" is chosen as preferred attribute, the buyer would purchase the product from Seller_simple.

Table 9: Sample Price offered by different sellers.

	Seller_simple	Seller_DF	Seller_om
Product Price	50	70	63
Product Quality	55	65	63
Delivery Time	60	60	63
After Sale Service	65	55	63
Sellers' Reputation	70	50	63

For simplicity we assume that in all ten rounds of the market there are equal numbers of buyers for each given five attributes.

Table 10: Number of buyers based on preferred attributes.

	Number of Buyers
Product Price	4
Product Quality	4
Delivery Time	4
After Sale Service	4
Sellers' Reputation	4

4.2.4 Results

We begin our experimental evaluation of the market by assuming that at the beginning of the first round Seller_simple and Seller_DF use data from Table 6 to set their selling

price of the product P whose production cost is 645.00 and our model uses information from Table 6 and Table 7 to generate a selling price (650.49) for P.

Table 11: Selling price offered by sellers in round 1.

	Seller_simple	Seller_DF	Seller_om
Product Price	648.99	648.99	650.49
Product Quality	745.99	745.99	650.49
Delivery Time	670.34	670.34	650.49
After Sale Service	805.99	805.99	650.49
Sellers' Reputation	718.99	718.99	650.49

From Table 11 we can see that the seller employing our model (Seller_om) attracts 16 buyers by the price offered for P. The remaining 4 buyers whose preferred attribute is "Product Price" would either purchase P from Seller_simple or Seller_DF. For simplicity, we assume that each of them attract two of the remaining buyers. At the end of round 1 we calculate the revenue earned by the sellers. We then determine the direction (positive or negative) of revenue earned compared to previous revenue.

Table 12: Total Revenue earned after round 1.

Seller Name	Total Revenue earned	Direction of Revenue earned
Seller_simple	1297.98	Positive
Seller_DF	1297.98	Positive
Seller_om	10407.80	Positive

Now, before beginning round 2, we allow all the sellers to update their price. At the end of round 1, since direction of revenue earned for Seller_DF is positive, he/she updates

his/her price by adding 0.02% value of current selling price. From values of Table 8 we can see that at the end of first round the Seller_simple has no information regarding selling price set by other sellers. Hence, we add a randomly generated value between 0.0 and 5.0 percent of the production cost (here, 2.17%) to the current price. However, our model does not update the price at the end of round 1, because, at the end of round 1 our model finds that the initial price used to train the model is similar to the price offered by other two sellers in the market. This is due to our assumption that at the beginning of first round Seller_simple and Seller_DF uses data from Table 6 to set their selling price of the product P. In addition, our model uses same set of information from Table 6 to generate a selling price for P at the beginning of first round.

Table 13: Selling price offered by sellers in round 2.

	Seller_simple ($\Delta \approx +2.17\%$)	**Seller_DF ($\Delta \approx +0.02\%$)**	**Seller_om**
Product Price	662.99	650.29	650.49
Product Quality	759.99	747.49	650.49
Delivery Time	684.34	671.69	650.49
After Sale Service	819.99	807.60	650.49
Sellers' Reputation	732.99	720.43	650.49

Table 13 indicates that the Seller_om offers the lowest price for all attributes except "Product Price". Again, Seller_om manages to attract 16 buyers in round 2. Compared to round 1, Seller_simple fails to sell any product in round 2. On the other hand, due to failure of Seller_simple, Seller_DF earned more revenue than that was earned in round 1.

Table 14: Total revenue earned after round 2.

Seller Name	Total Revenue earned	Direction of Revenue earned
Seller_simple	1297.98	Negative
Seller_DF	3896.54	Positive
Seller_om	20815.58	Positive

Table 8 indicates that at the end of second round Seller_simple is unaware of the information regarding selling price set by other sellers. Moreover, at the end of round 2, Seller_simple finds his/her direction of revenue earned as negative. Hence, he/she updates the selling price by subtracting a random value between 0.0% and 5.0% of the production cost (here, 1.58%) from his/her current selling price. However, Seller_DF gets another chance to increase his/her selling price because of the positive direction of revenue earned in this round. This time he/she updates his/her price by adding a random value between 0.0% and 5.0% (here, 0.25%) of current selling price. As mentioned before, Seller_om updates the price by using our model. Before updating the price, we store the current pricing information for training purpose. As we described in Section 3.5, since Seller_om earned some revenue by using the current selling price that was determined by our model at the beginning of first round, we add current inputs and output of the network to the training set of Table 3 so that it can be used during the training process of the network, i.e., the following set is added at the last row of Table 3.

645.00	648.99	745.99	670.34	805.99	718.99	650.49

After saving the current pricing information, Seller_om updates the price. Before providing initial input to the network of our model to derive a new updated selling price from the model, Seller_om analyzes the prices offered by other two competing sellers. Seller_om uses the information of Table 15 and Table 7 to determine the new selling price. We obtain the values of Table 15 by taking the minimum price offered by Seller_simple and Seller_DF in previous round (Table 13).

Table 15: Initial input to network in round 3.

Production Cost	645.00
Product Price	650.29
Product quality	747.49
Delivery time	671.69
After Sales Service	807.60
Sellers' Reputation	720.42

The price offered by the three sellers during round 3 is given in Table 16. We can see that our model provide same price as the price of previous round. This is due to the insignificant difference between the initial input of the network used in previous round (Table 6) and current round (Table 15).

Table 16: Selling price offered by sellers in round 3.

	Seller_simple ($\Delta \approx -1.58\%$)	**Seller_DF ($\Delta \approx +0.25\%$)**	**Seller_om**
Product Price	652.78	666.5452	650.49
Product Quality	749.78	766.169	650.49
Delivery Time	674.13	688.4727	650.49
After Sale Service	809.78	827.792	650.49
Sellers' Reputation	722.78	738.4387	650.49

After end of round 3, we determine the revenue earned by each seller and then allows sellers to update their prices. As far information of Table 8, at the end of third round the information regarding other sellers' selling price is available to Seller_simple. As we mentioned earlier, due to manual slow searching process the information of the $(r\text{-}2)^{th}$ round is visible to Seller_simple, where r is the current round. Therefore, at the beginning of fourth round Seller_simple uses the prices offered by Seller_DF and Seller_om during second round (Table 13) to update his/her price. In this case, Seller_simple calculates the average value of the prices offered by other two sellers from Table 13 and set the values as the selling prices for round 4. We run the remaining seven rounds in the similar fashion.

Figure 25 summarizes the total revenue earned at the end of each round by the three different sellers who employed three distinct pricing algorithms. Initially, all the sellers managed to earn some revenue. Among the three sellers, the growth of revenue earned by Seller_simple was the slowest. Seller_simple failed to earn any revenue at the end of most of the rounds. Apart from first round, Seller_simple earned some revenue after the end of seventh and tenth round..The performace of Seller_DF in terms of earning revenue was better than Seller_simple, however, he/she could not beat Seller_om in any of the rounds. On the contrary, Seller_om, who employed our model, earned revenue at each round. Moreover, after each round, Seller_om earned higher revenue than that of revenue earned by other two sellers. At the end of tenth round, Seller_DF earned nearly 43%

more revenue than Seller_simple. On the other hand, Seller_om earned nearly eight times more revenue than Seller_DF.

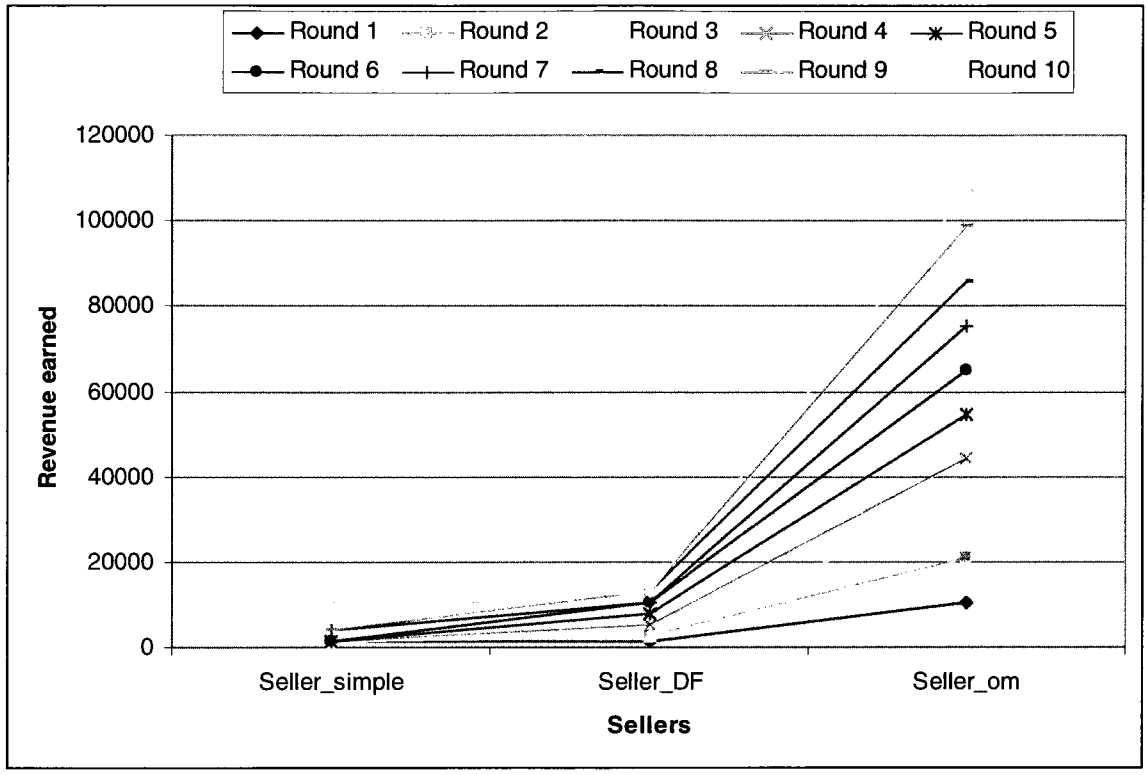

Figure 25: Total revenue earned by three sellers.

In our experimental evaluation we showed that once the sellers set an initial price of the product based on five different attributes, our model adjusts the price of the product automatically with the help of neural network in order to increase revenue. In other words, our model considers five different attributes in generating a selling price for a

product such that it can attracts buyers with different preferred attributes which leads to capitalizing the revenue earned. In setting the initial price of a product, we assume that sellers use their prior knowledge about the prices of the product offered by other competing sellers.

4.3 Enhanced Experimental Evaluation

In Section 4.2.3 we demonstrated a market place where three sellers (namely, Seller_simple, Seller_DF and Seller_om) participate in selling a product P to 200 different buyers with distinct preferable purchase attributes of products. In the experimental evaluation we considered that no seller has knowledge on the buyers preferred purchase attributes. Consequently, after each round the sellers have no clue on the earned revenue came from which of the five preferred purchase attributes. Therefore, we would like to perform another experimental evaluation of our model with the same market place described in Section 4.2.3. However, in this case we assume that Seller_simple and Seller_DF aware of the preferred purchase attributes from which they made revenue. In other words, we presume that after each round Seller_simple and Seller_DF can determine the direction of revenue (either positive or negative) earned for the five different attributes. These two sellers then update their selling price by using their knowledge of the attributes which was absent in our previous experimental evaluation.

We begin our experimental evaluation in the similar fashion that we demonstrated in Section 4.2.4. The first round of this evaluation is a carbon copy of the first round of previous evaluation. The difference between two versions of evaluation comes into scene at the time of updating selling price before beginning of the second round. At the end of round 1 we can find the revenue earned with its direction as shown in Table 12. Now, before beginning round 2, we allow all the sellers to update their price. At the end of round 1, Seller_DF finds that he/she earned some revenue from the price he/she offered for "Product Price" attribute. Hence, Seller_DF updates his/her price for "Product Price" attribute by adding a random value between 0.0% and 5.0% (here 1.29%) of current selling price. However, since Seller_DF did not earn anything from the price he/she offered for other four attributes, he/she update the prices for those attributes by subtracting 0.76% value of current selling price. From values of Table 8 we can see that at the end of first round the Seller_simple has no information regarding selling price set by other sellers. Hence, Seller_simple updates his/her selling price by adding/subtracting a randomly generated value to the current price. Since Seller_simple made some revenue from the price he/she offered for "Product Price" attribute, he/she adds a random value between 0.0% and 5.0% (here 1.93%) of the product cost for this attributes. As he/she failed to earn anything from the other four attributes, Seller_simple updates his/her selling price for those attributes by deducting 1.7% value of the production cost. On the other hand, our model follows the same technique for updating the selling price as portrayed in Section 4.2.4.

Table 17: Selling price offered by sellers in round 2.

	Seller_simple	Seller_DF	Seller_om
Product Price	661.49	657.36	650.49
Product Quality	735.03	740.32	650.49
Delivery Time	659.38	665.25	650.49
After Sale Service	795.06	799.87	650.49
Sellers' Reputation	708.06	713.53	650.49

We then calculate the revenue earned by the three sellers and follow the same procedure in updating the selling price before going to the next round.

Figure 26 summarizes the total revenue earned at the end of each round by the three different sellers. From this enhanced version of experimental evaluation we can see that Seller_simple and Seller_DF performed much better in terms of revenue earned compared to the evaluation of Section 4.2.4. Figure 26 portrays that both Seller_om and Seller_DF maintained a steady growth in earning revenue over the ten rounds. As the evaluation progress from one round to another round, the difference in revenue earned by these two sellers decreased. In contrast, Seller_simple could not earn as much revenue as compared to other two sellers. However, Seller_simple managed to carry on the growth of revenue earned from one round to another. At the end of tenth round, Seller_om, who employed our model, earned higher revenue in comparison to Seller_DF and Seller_simple.

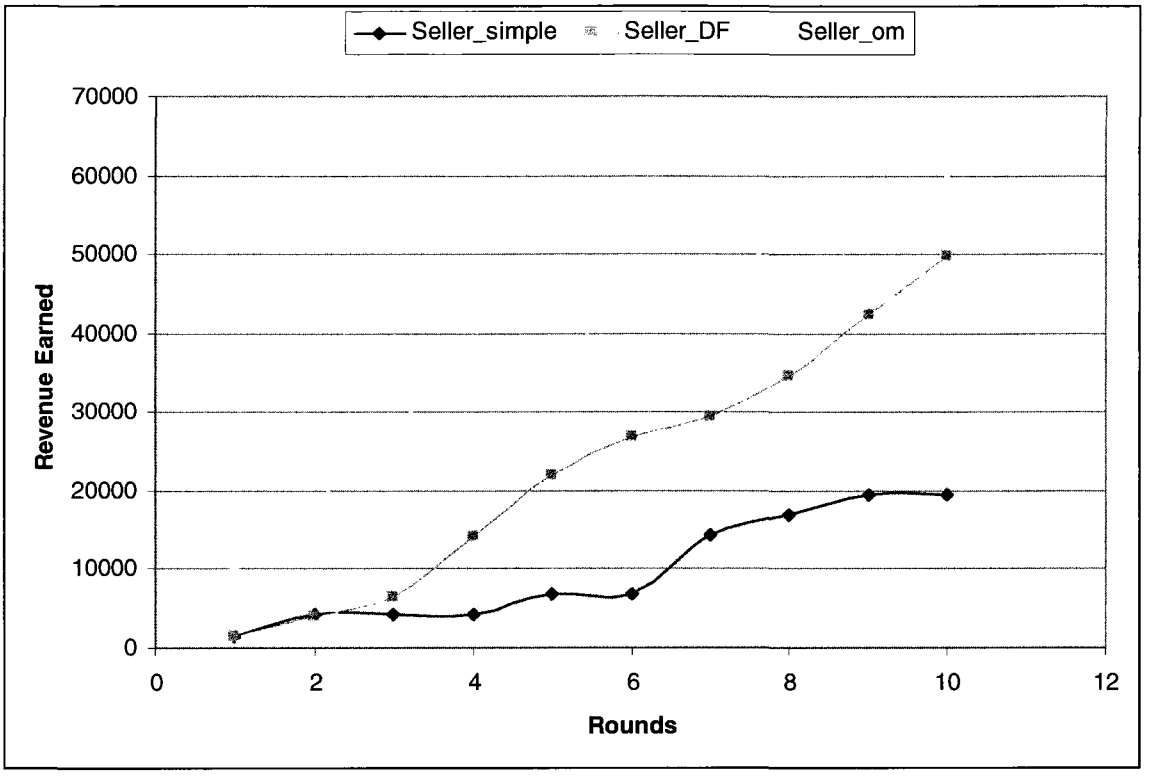

Figure 26: Comparison of three sellers in terms of revenue earned.

4.4 Discussion

The experimental results show that our model can attract more buyers compared to other two sellers, because we have considered multiple attributes in determining a selling price for the product P. Attracting more buyers from wider range of preferred attributes implies that more revenue can be earned.

Various pricing algorithms are followed in present online economy. Among them game-theoric pricing (GT), myoptimal pricing (MY), derivative following (DF), and Q-learning

(Q) are practiced widely. Game-theoretic (GT) strategy makes an assumption that all other competing sellers use game-theoretic [31]. However, in present world different sellers employ distinct pricing strategies. GT uses complete information regarding buyer population. Moreover, it does not use any historical data. In contrast, historical data plays an important role in understanding changing behavior of the market. We used little historical data of the price offered by other competitive sellers. In addition, our model is not concerned about what strategies are being used by other competing sellers. Similarly, Myoptimal (MY) strategy does not dependant on whether other sellers employing different pricing strategies or not, but it is concerned about buyers demand curve and also the prices set by other sellers in the economy. MY also assumes that prices set by other competing sellers will remain unchanged [31]. In the contrary, sellers are always willing to change their offered price for the sake of sustaining in competing market. Hence, our model always keeps an eye on the random prices set by other sellers. Q-learning strategy is based on reinforcement learning called Q-learning. It finds optimal policy when the opponents use stationary Markovian strategies. It uses a lookup table representation of the Q-function. It requires extensive size of computational requirement. It makes use of both buyers' demand curve and knowledge about competitors pricing strategies. On the other hand, our model does not rely on buyer demand curve.

In short, we attempt to address the problem of dynamic pricing in a competitive online economy, where a buyer's purchase decision is determined by multiple attributes. From the behavior of current market we can infer that buyer's purchase decision is no more

dependent solely on products price. Along with price, other attributes such as product quality, delivery time, after-sale service, and sellers' reputation also play vital role in determining customers' purchase decision. Hence, in our model we consider these mentioned five attributes so that we can attract more number of buyers which lead to earning more revenue. However, our model is general enough to work for any number of attributes. Our model requires initial price to be set by sellers by using their prior knowledge about the prices of the product offered by other competing sellers. Any other prior knowledge is not used by our model. In our experimental evaluation we showed that once the sellers set an initial price of the product, our model adjusts the price of the product automatically with the help of neural network in order to increase profits.

Chapter 5: Conclusions and Future Works

5.1 Conclusions

Dynamic pricing as a changing price in a marketplace is becoming characteristic of electronic commerce. Determining selling prices of products is a challenging task for the sellers to sustain in the market. The purpose of the dynamic pricing problem is to determine selling prices such that sellers receive better revenue. There exist intelligent agents to aid online sellers to dynamically calculate a competitive price for their products in online markets. However, these intelligent agents usually make a number of assumptions for dynamic pricing. Some intelligent agents assume that sellers consist of prior knowledge about the online market parameters, while other agents assume that price is the only attribute that determines consumers' purchase decision [2, 9, 10, 12, 22]. On the contrary, in real life sellers have limited or no prior knowledge about the market parameters. In addition, nowadays along with price other attributes such as after sale service, product quality etc. contribute in determining consumers' purchase decision. The proposed approach of dynamic pricing described in our work of this thesis used feed-forward neural network to determine product price dynamically. We used back propagation algorithm to minimize the errors while training the network with 10 sets of

training patterns. We considered buyer preferences over multiple product attributes. Along with product price, we have taken product quality, delivery time, after-sale service and sellers' reputation into consideration. However, our model is general enough to work for any number of attributes. We also considered that sellers have limited prior knowledge about market parameters like how other competing sellers set the prices. Our model assumed no other information. Our model aids the sellers of competitive market in the automation of determining the price of a product in order to earn better revenue. The approach requires the sellers, by considering the five attributes, to set an initial price of the product by using their prior knowledge about the prices of the product offered by other competing sellers. Our approach adjusts the selling price of products automatically with the help of neural network in order to raise seller revenue. The experimental results portray the effect of considering the five attributes in earning revenue by the sellers. We performed an experimental evaluation of our model in an e-commerce market place with 200 buyers, three sellers where all the sellers trying to sell a product P. The experimental results showed that the seller employing our model earned higher revenue than that of earned by other two sellers who followed simple pricing algorithm and derivative-following (DF) strategies. The experimental results show that our model can attract more buyers compared to other two sellers, because we have considered multiple attributes in determining a selling price for the product P. Attracting more buyers from wider range of preferred attributes implies that more revenue can be earned. Partial work of this thesis has been published in the proceedings of 4th International MCETECH Conference on e-Technologies, Ottawa, ON, Canada, May 2009. Furthermore, some work of this thesis

has appeared in Proceedings of the 28th Canadian Conference on Artificial Intelligence that has held in Ottawa, ON, Canada between May 31 and June 2, 2010.

5.2 Future Works

- We have made a comparison of our approach of dynamic pricing with simple pricing algorithm and derivative following (DF) strategies. We intend to compare our approach with some more complex pricing strategies.
    - We would like to compare our approach with other existing well known approach of dynamic pricing, like game-theoretic (GT), my-optimal (MY) etc.
    - We are planning to compare the total revenue earned by different sellers after selling the same product whose price is determined by different strategies (GT, MY and our designed approach).
    - In the comparison we will be considering each seller follows different pricing strategies.
- Our model made an assumption that sellers use their limited prior knowledge of market parameters in setting the initial price of the products. We would like to eliminate the assumption from our model.
    - We may employ a web crawler tool in our application in order to learn the information on prices set by other competitive sellers in the market.

- Using the gathered information we may set the initial price of the products such that the sellers no need to initialize the product prices while using our model.

References

[1] Anindya Ghose, Tridas Mukhopadhyay, Uday Rajan, Vidyanand Choudhary: Dynamic Pricing: A Strategic advantage for electronin retailers. In: Twenty-Third International Conference on Information Systems (2002)

[2] Prithviraj(Raj) Dasgupta and Yoshitsugu Hashimoto: Multi-attribute dynamic pricing for online markets using intelligent agents. In: AAMAS (2004)

[3] J. Brown and A. Goolsbee: Does the Internet Make Markets More Competitive? In: NBER Working Papers 7996, National Bureau of Economic Research (2000)

[4] Russell, J. Stuart, Peter Norvig: Artificial Intelligence: A modern Approach, Second Edition, Prentice Hall (2005)

[5] Neural Networks,
http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html

[6] An Introduction to Neural Networks,
http://www.cs.stir.ac.uk/~lss/NNIntro/InvSlides.html

[7] C. Brooks, R. Gazzale, J. MacKie-Mason, E. Durfee: Improving learning performance by applying economic knowledge. In: Proc. of the 3rd ACM Conference on Electronic Commerce (2003)

[8] C. Brooks, S. Fay, R. Das, J. MacKie-Mason, J. Kephart, E. Durfee: Automated strategy searches in an electronic goods market: learning and complex price schedules. In: Proc. of 1st ACM Conference on E-Commerce (1999)

[9] V. L. Raju Chinthalapati, Narahari Yadati, and Ravikumar Karumanchi: Learning Dynamic Prices in MultiSeller Electronic Retail Markets With Price Sensitive Customers, Stochastic Demands, and Inventory Replenishments. In: IEEE (2006)

[10] Prithviraj Dasgupta, Rajarshi Das: Dynamic Service Pricing for Brokers in a Multi-Agent Economy. In: IEEE (2000)

[11] Danxia Kong: One Dynamic Pricing Strategy in Agent Economy Using Neural Network Based on Online Learning. In: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (2004)

[12] Li Luo, Baichun Xiao, Jiejun Deng: Dynamic pricing decision analysis for parallel flights in competitive markets. In: IEEE, pp. 327-312 (2005)

[13] Alexandre X. Carvalho, Martin L. Puterman: Dynamic pricing and reinforcement learning. In: IEEE (2003)

[14] Prithviraj Dasgupta, Louise E. Moser, P. Michael Melliar-Smith: Dynamic Pricing for Time-Limited Goods in a Supplier-Driven Electronic Marketplace. In: Electronic commerce research (2005)

[15] Joan Morris Dimicco, Pattie Maes and Amy Greenwald: Learning Curve: A Simulation-Based Approach to Dynamic Pricing. In: Electronic Commerce research (2003)

[16] Gerald Tesauro and Jeffrey O. Kephart: Pricing in Agent-Economies using Multi-agent Q-learning. In: AAMAS (2002)

[17] G. Gallego and G. van Ryzin: Optimal dynamic pricing of inventories with stochastic demand over finite horizons. In: *Manage. Sci.*, vol. 40, no. 8, pp. 999–1020 (1994)

[18] CS-449: Neural Networks, http://www.willamette.edu/~gorr/classes/cs449/intro.html

[19] Artificial Neural Network, http://en.wikipedia.org/wiki/Artificial_neural_network

[20] Statistical Learning Methods, http://aima.cs.berkeley.edu/newchap20.pdf

[21] Module: Neural Networks, Multi-Layer-Perceptron. CS 4700: Foundations of Artificial Intelligence, http://www.cs.cornell.edu/Courses/cs4700/2008fa/PPT/CS4700-ANN4.ppt#374,1,CS 4700: Foundations of Artificial Intelligence

[22] J. Kephart, J. Hanson and A. Greenwald: Dynamic Pricing by Software Agents. In: Computer Networks, vol 32, no 6, pp. 731-752 (2000)

[23] Training the Neural Network using Back Propagation, http://richardbowles.tripod.com/neural/backprop/backprop.htm

[24] Li, C., Wang, H and Zhang, Y. "Dynamic pricing decision in a duopolistic retailing market", Proceedings of the 6th World Congress on Intelligent Control and Automation, June 2006, Dalian, China.

[25] Panos M. Markopoulos, Jeffrey O. Kephart: How valuable are shopbots? AAMAS 2002: 1009-1016

[26] Jeffrey O. Kephart, Amy R. Greenwald: Shopbot Economics. Autonomous Agents and Multi-Agent Systems 5(3): 255-287 (2002)

[27] Jeffrey O. Kephart, Christopher H. Brooks, Rajarshi Das: Pricing information bundles in a dynamic environment. ACM Conference on Electronic Commerce 2001: 180-190

[28] Amy R. Greenwald, Jeffrey O. Kephart: Probabilistic pricebots. Agents 2001: 560-567

[29] Jeffrey O. Kephart, Gerald Tesauro: Pseudo-convergent Q-Learning by Competitive Pricebots. ICML 2000: 463-470

[30] Gerald Tesauro, Jeffrey O. Kephart: Foresight-based pricing algorithms in agent economies. Decision Support Systems 28(1-2): 49-60 (2000)

[31] Amy R. Greenwald, Jeffrey O. Kephart, Gerald Tesauro: Strategic pricebot dynamics. ACM Conference on Electronic Commerce 1999: 58-67

[32] Amy R. Greenwald, Jeffrey O. Kephart: Shopbots and Pricebots. Agent Mediated Electronic Commerce (IJCAI Workshop) 1999: 1-23 (look at http://www.research.ibm.com/infoecon/paps/html/amec99_shopbot/shopbot.html)

[33] Jeffrey O. Kephart, Amy R. Greenwald: Shopbot Economics. Agents 1999: 378-379

[34] Joan Morris DiMicco, Amy R. Greenwald, Pattie Maes: Dynamic pricing strategies under a finite time horizon. ACM Conference on Electronic Commerce 2001: 95-104

[35] Heski Bar-Isaac and Steven Tadelis: Seller Reputation. In: Foundations and Trends in Microeconomics. vol. 4, no. 4, pg 273-351, 2008.

[36] Frederick Asselin and Brahim Chaib-draa: Toward a Protocol for the Formation of Coalitions of Buyers. In: The Fifth International Conference on Electronic Commerce Research (ICECR-5) 2002.

[37] Yinh-Hueih Chen, Ching-Yi Tsao, Chia-Chen Lin and I-Chieh Hsu: A Conjoint Study of the Relationship between Website Attributes and Consumer Purchase Intentions. In: Pacific Asia Conference on Information Systems (PACIS) 2008.

[38] Sandhya Beldona, Costas Tsatsoulis: Reputation Based Buyer Strategy For Seller Selection For Both Frequent and Infrequent Purchases. In: 4th International Conference on Informatics in Control, Automation & Robotics, ICINCO-RA (2) 2007: 84-91, Angers, France.

[39] www.futureshop.ca accessed on September 25, 2009.

[40] www.bestbuy.ca accessed on September 25, 2009.

[41] www.dell.ca accessed on September 25, 2009.

[42] www.ebay.ca accessed on September 25, 2009.

[43]. http://seekingalpha.com/article/105889-top-10-online-retailers-shopping-categories accessed on November 7, 2009.