International Journal of Intelligent Information Systems
2016; 5(2): 25-36
http://www.sciencepublishinggroup.com/j/ijiis
doi: 10.11648/j.ijiis.20160502.11
ISSN: 2328-7675 (Print); ISSN: 2328-7683 (Online)

Bread Shrimp Microbe Growth Simulation and Prediction System Based on Neural Network

Xiao Laisheng[1], Zheng Yuandan[2]

[1]Educational Information Center, Guangdong Ocean University, Zhanjiang, China
[2]Information College, Guangdong Ocean University, Zhanjiang, China

Email address:
xiaolaisheng@163.com (Xiao Laisheng)

To cite this article:
Xiao Laisheng, Zheng Yuandan. Bread Shrimp Microbe Growth Simulation and Prediction System Based on Neural Network. *International Journal of Intelligent Information Systems.* Vol. 5, No. 2, 2016, pp. 25-36. doi: 10.11648/j.ijiis.20160502.11

Received: January 22, 2016; **Accepted**: February 3, 2016; **Published**: March 12, 2016

Abstract: According to the requirements of a scientific research project, a set of bread shrimp microbial growth simulation and prediction system is designed and implemented in detail. The system is established by taking vibrio parahemolyticus in bread shrimp as research objects, according to effects of temperature, salt and time on their growth, and employing neural network technology. In order to improve its compatibility, the system is developed by using C# on Visual Studio 2008 platform, and its design and implementation are based on AForge.NET framework and sliding-window modeling method. The system consists of three parts: data management, data simulation and data prediction, which would provide an effective analytical tool for bread shrimp safe production. After tested carefully, the system can meet the requirements of the project design.

Keywords: Neural Network, AForge.NET, Simulation and Prediction System, Microbial Growth, Bread Shrimp

1. Introduction

Intelligent simulation and prediction for microbial growth refers to that under the premise of without microbial detection and analysis and according to the characteristic data of the food microbe in different processing, storage and circulation, the dynamic changes of the growth and survival of main pathogenic bacteria and spoilage bacteria in food is going to be determined by processing with computers, thus the quality and safety of food can be evaluated and predicted quickly. According to the existing limited experimental data, using neural network prediction algorithm, and through intelligent simulation and prediction neural network-based microbial growth simulation and prediction system can provide more data than one that come from real experiments, which could provide reference data for safe food production.

According to the requirements of a research project, sub project of National Science and Technology Support Program of China: Key Integrated and Demonstrated Technologies for Quality and Safety Control in Aquatic Product Processing Process (No. 2012BAD29B06), the authors of this paper have developed a set of bread shrimp microbial growth simulation and prediction system. The system is established by taking vibrio parahemolyticus in bread shrimp as research objects, according to effects of temperature, salt and time on their growth, and employing neural network technology. The system consists of three parts: data management, data simulation and data prediction, which would provide an effective analytical tool for bread shrimp safe production.

In order to improve its compatibility, the system is developed by using C# on Visual Studio 2008 platform, and its design and implementation are based on AForge.NET framework and sliding-window modeling method. Through intelligent simulation and prediction in the system, users can predict vibrio parahemolyticus growth and survival data in various conditions only providing a limited number of experimental vibrio parahemolyticus growth and survival data in the conditions. By this way, users do not need to perform experiments in each condition, greatly reducing the number of experiments and saving experimental time and costs.

2. Related Work

Food simulation and prediction microbiology is a new science based on the subjects such as microbiology, mathematics, statistics and computer applications, which needs the research foundation that a series of models that can

be able to describe and predict microbial growth and survival in specific conditions should be designed. The core of food microbe simulation technology is to form mathematical models, through which the rules of growth, live and death of microbe can be described [1-2].

Up to now, domestic scholars have done a lot of research for microbial growth simulation modeling [3-7]. In their work, all of the models including level-one, level-two and level-three models described microbial growth with mathematical equations, such as level-one model with linear model, Logistic model, Gompertz model, Baranyi & Roberts model etc, level model with square root model, Al Leave equation etc. In recent years overseas scholars have also done a lot of research for microbial growth simulation modeling [8-15], but they only put emphasis on mathematical modeling too. As imagined, microbial growth is a dynamic and continuous process, it is difficult to use single mathematical equation to simulate. Artificial neural network is a network constructed manually to simulate human brain function, absorbing some advantages of biological nerve, such as high parallelism, non-linear global role, good fault-tolerance and associative memory function, and very strong self adaptive and self learning ability. Therefore, neural network has very outstanding qualities of adaptive learning, parallel computing, distributing storage, associative memory. Theory has proved that neural network can approach any continuous real function with arbitrary precision. Hence, it is feasible to take neural network to simulate microbial growth process.

Just for that reason, scholars over the world have carried out a great deal of research on the aspect of applying neural network to microbial growth simulation [16-23]. From these related research literatures, we can know that the application of neural network applied to microbial growth was also carried out a more extensive research, and a good simulation effect was achieved too. But so far, applying neural network to simulate and predict growth for vibrio parahemolyticus in bread shrimp is still not reported. Moreover, in [24] one author of the paper tried to develop a universal platform for microbial growth simulation modeling with VC++ and MATLAB, but its compatibility is not good. In the light of such reasons, in order to meet the requirement of the research project (No. 2012BAD29B06) and to improve the compatibility of system, in this paper we have developed an intelligent simulation and prediction system for bread shrimp microbial growth by using C# on Visual Studio 2008 platform, and its design and implementation are based on AForge.NET framework and sliding-window modeling method.

3. AForge.NET

AForge.NET is an open source C# framework designed for developers and researchers in the fields of computer vision and artificial intelligence - image processing, neural networks, genetic algorithms, fuzzy logic, machine learning, robotics, etc.

The framework is comprised by the set of libraries and sample applications, which demonstrate their features. Typically, AForge.NET is comprised of a series of components, such as AForge.Imaging, AForge.Vision, AForge.Video, AForge.Neuro, AForge. Genetic, AForge.Fuzzy, AForge.Robotics, AForge.MachineLearning, etc. AForge.Imaging is a library with image processing routines and filters; AForge.Vision is a computer vision library; AForge.Video is a set of libraries for video processing; AForge.Neuro is a neural networks computation library; AForge.Genetic is an evolution programming library; AForge.Fuzzy is a fuzzy computations library; AForge.Robotics is a library providing support of some robotics kits; AForge.MachineLearning is a machine learning library. The framework is provided not only with different libraries and their sources, but with many sample applications, which demonstrate the use of this framework, and with documentation help files, which are provided in HTML Help format. The documentation is also available on-line. [25]

When a neural network-based system is developed, a neural networks computation library, AForge.Neuro, should be used. The C# library on AForge.Neuro contains six main entities shown in Figure 1.

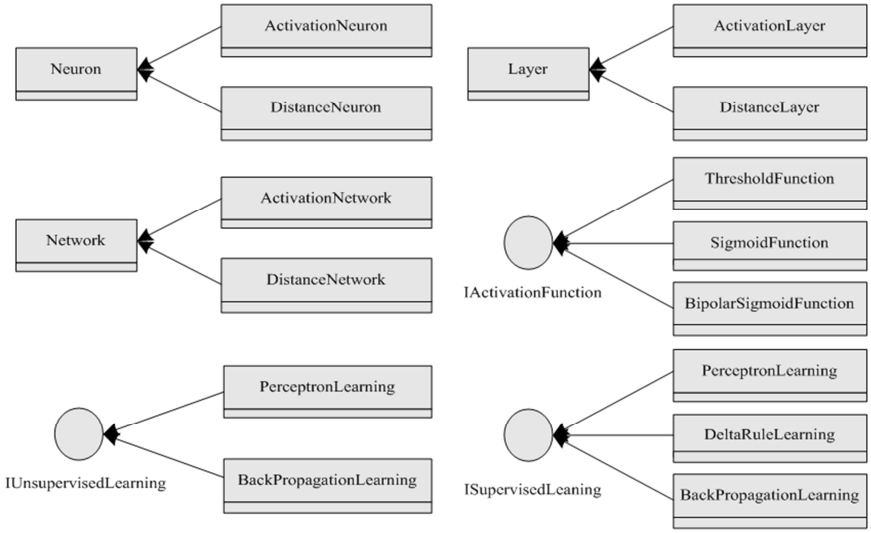

Figure 1. *Six main entities in C# library on AForge.Neuro.*

From Figure 1, we can see that there are six main entities in C# library on AForge.Neuro, namely, Neuron, Layer, Network, IActivationFunction, ISupervisedLearning, IUnsupervisedLearning. Neural network is made of a collection of neurons, in which each neuron is described by Neuron. Neuron is a base abstract class for all neurons, which encapsulates such common entities like a neuron's weight, output value, and input value. Other neuron classes inherit from the base class to extend it with additional properties and specialize it. Neural network is also made of some layers, in which each layer is described by Layer. Layer represents a collection of neurons. This is a base abstract class, which encapsulates common functionality for all neuron's layers. A network consists of a collection of neuron's layers, which is described be Network, what is a base abstract class, which provides common functionality of a generic neural network. To implement a specific neural network, it is required to inherit the class, extending it with specific functionalities of any neural network architecture. IActivationFunction is an activation function's interface. Activation functions are used in activation neurons - the type of neuron, where the weighted sum of its inputs is calculated and then the value is passed as input to the activation function, and the output value becomes the output value of the neuron. ISupervisedLearning is an interface for supervised learning algorithms - the type of learning algorithms where a system is provided with sample inputs, with desired output values during the learning phase. The aim of the system is to generalize learning data, and learn to provide the correct output value when it is presented with the input value only. IUnsupervisedLearning is an interface for unsupervised learning algorithms - the type of learning algorithms where a system is provided with sample inputs only during the learning phase, but not with the desired outputs. The aim of the system is to organize itself in such a way to find correlation and similarities between data samples.

The C# library on AForge.Neuro provides the following two types of neural network architectures: Activation Network and Distance Network. In engineering practice, we can select one of the two types of neural network architectures, but how to make a choice depends on concrete problems. Activation Network is commonly used neural network where each neuron computes its output as the activation function's output, and the argument is a weighted sum of its inputs combined with the threshold value. The network may consist of a single layer, or of multiple layers. Trained with supervised learning algorithms, the network allows to solve such tasks as approximation, prediction, classification, and recognition. Another neural network is Distance Network where each neuron computes its output as a distance between its weight values and input values. The network consists of a single layer, and may be used as a base for such networks like Kohonen Self Organizing Map, Elastic Network, and Hamming Network.

The C# library on AForge.Neuro also provides abundant learning algorithms, which could be used to train different neural networks and to solve different problems. Primary learning algorithms include Back Propagation Learning, Perceptron Learning, SOM Learning, Delta Rule Learning, Elastic Network Learning. Back Propagation Learning is one of the most popular, known and commonly used algorithms for multi-layer neural network learning. Because the algorithm is able to train multi-layer neural networks, the range of its applications is very great, and includes such tasks as approximation, prediction, object recognition, etc. Perceptron Learning could be used with a one-layer activation network, where each neuron has a threshold activation function. SOM Learning treats neural network as a 2D map of nodes, where each node may represent a separate class. The algorithm organizes a network in such a way, that it becomes possible to find the correlation and similarities between data samples. Delta Rule Learning utilizes the activation function's derivative, and may be applicable to single-layer activation networks only, where each neuron has a continuous activation function instead of a threshold activation function. The most popular continuous activation function is the unipolar and bipolar sigmoid function. Elastic Network Learning is similar to the idea of the SOM learning algorithm, but it treats network neurons not as a 2D map of nodes, but as a ring. During the learning procedure, the ring gets some shape, which represents a solution. [26]

Software design process based on AForge.NET could be described as follows:

Firstly in order to use the classes and interfaces of AForge. Neuro to construct and train the neural network model, it is needed to open VisualStudio2008 software and add the three dynamic link libraries AForge. Controls. dll, AForge. dll and AForge. Neuro. dll in its solution manager.

Secondly the activation network or distance network class should be introduced to instantiate the neural network model in accordance with the actual requirements. Subsequently, it is needed to choose a suitable learning algorithm for each model from the classes of BP Learning, Delta rule Learning, Perceptron Learning, SOM Learning or Elastic Network Learning, and set the parameters such as relevant learning rate and impulse value.

Finally the neural network model can be trained by using the training data sets generated by sliding-window method. At the same time, the error curves in the training process can be drawn onto the Chart control provided by AForge.NET framework so as to view the data conveniently. When the error of neural network is less than the preset error or the number of training is reached, the training process will be stopped [27].

Nowadays, C#-based AForge.NET is widely used in scientific and engineering research, industry applications, such as motion video detection, eye-tracking control system, diagnostics of products, etc. [28-31].

4. Neural Network Sliding-Window Modeling Method

In general, neural network modeling uses sliding-window

method. In order to facilitate computer programming, a neural network sliding-window modeling method is theoretically derived in detail by one of the authors of the paper Xiao Laisheng [32]. Specific method is described as follows.

4.1. Neural Network Architecture for Sliding-Window Modeling

Figure 2 Shows neural network architecture for sliding-window modeling.

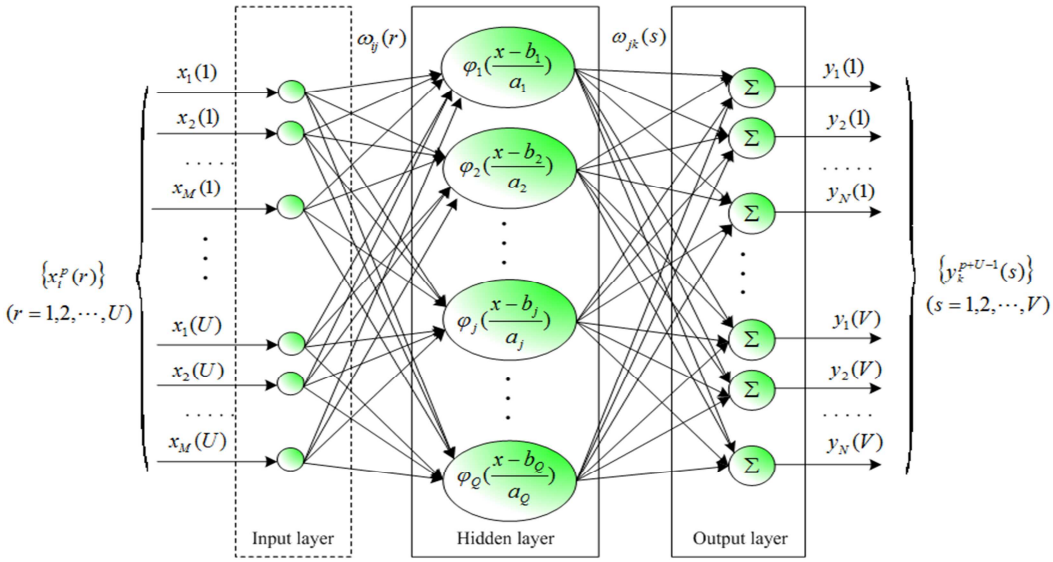

Figure 2. *Neural network architecture for sliding-window modeling.*

In Figure 2, assume that M is total number of nodes in input layer, $i=1,2,\cdots,M$; Q is total number of nodes in hidden layer, $j=1,2,\cdots,Q$; N is total number of nodes in output layer, $k=1,2,\cdots,N$; P is total number of input samples, $p=1,2,\cdots,P$; $\{x_i^p\}$ are inputs of network, $\{y_k^p\}$ are outputs of network, $\{d_k^p\}$ are their corresponding target outputs; ω_{ij} are connection weights between neurons in input layer and neurons in hidden layers, ω_{jk} are connection weights between neurons in hidden layer and neurons in output layers; a_j is scale parameter and b_j is translation parameter of wavelet. When Sliding-Windows is adopted, U is sliding-window input parameter, V is sliding-window output parameter. In this case, input weights of network ω_{ij} are extended equally to the number of $\omega_{ij}\times U$, which are marked as $\omega_{ij}(r), r=1,2,\cdots,U$, and output weights of network ω_{jk} are extended equally to the number of $\omega_{jk}\times V$, which are marked as $\omega_{jk}(s), s=1,2,\cdots V$.

Then input of jth neuron of wavelet basis can be expressed as $s_j^p=\sum_{i=1}^{M}\omega_{ij}x_i^p$, its output is $t_j^p=\phi(\frac{s_j^p-b_j}{a_j})$, output of kth level component in output layer is $y_k^p=\sum_{j=1}^{Q}\omega_{jk}t_j^p$, total error function is defined as $E=\frac{1}{P}\sum_{p=1}^{P}E^p=\frac{1}{2P}\sum_{p=1}^{P}\sum_{k=1}^{N}(d_k^p-y_k^p)^2$.

4.2. Learning Algorithm

When sliding-window is used, the learning algorithm of wavelet neural network is described as follows.

Step one, Initialize network weights, thresholds, scale and translation parameters of wavelet function.

Give corresponding initial values for each of the following parameters: scale parameter a_j and translation parameter b_j of the wavelet, weights of the network $\omega_{ij}(r), r=1,2,\cdots,U$ and $\omega_{jk}(s), s=1,2,\cdots V$, learning rate η and momentum factor λ.

Set sample counter $p=1$ and number of iterations $n=1$, and provided maximum number of iterations is N.

Step two, Input learning sample s $\{x_i^p(r)\}, r=1,2,\cdots,U$ and corresponding sliding-window desired outputs $\{d_k^{p+U-1}(s)\}, s=1,2,\cdots,V$, and calculate outputs of hidden layer $\{t_j^p\}$ and sliding-window outputs of output layer $\{y_k^{p+U-1}(s)\}, s=1,2,\cdots,V$.

Inputs of hidden layer are

$$s_j^p=\sum_{i=1}^{M}(\sum_{r=1}^{U}\omega_{ij}(r)x_i^p(r)),\quad j=1,2,\cdots,Q \tag{1}$$

Outputs of hidden layer are

$$t_j^p=\phi(\frac{s_j^p-b_j}{a_j})=\phi(\frac{\sum_{i=1}^{M}(\sum_{r=1}^{U}\omega_{ij}(r)x_i^p(r))-b_j}{a_j}),\quad j=1,2,\cdots,Q \tag{2}$$

Sliding-window outputs of output layer are

$$y_k^{p+U-1}(s)=\sum_{j=1}^{Q}\omega_{jk}(s)t_j^p,\ k=1,2,\cdots,N.\ s=1,2,\cdots V \quad (3)$$

In equations (1), (2), and (3), $x_i^p(r)$ is input of input layer, s_j^p is input of hidden layer, t_j^p is output of hidden layer, $y_k^{p+U-1}(s)$ is sliding-window output of output layer, $\phi(\cdot)$ is wavelet basis function.

Step three, Calculate target error and gradient vectors.

Define target error function E^p as

$$E^p=\frac{1}{2}\sum_{k=1}^{N}\sum_{s=1}^{V}(d_k^{p+U-1}(s)-y_k^{p+U-1}(s))^2 \quad (4)$$

Where, $\left\{d_k^{p+U-1}(s)\right\}$ is sliding-window desired outputs of output layer, $y_k^{p+U-1}(s)$ is sliding-window calculated outputs of output layer.

Energy function gradient respectively are

$$\delta_{ij}^p(r)=\frac{\partial E^p}{\partial\omega_{ij}(r)}=\frac{\partial E^p}{\partial y_k^{p+U-1}(s)}\frac{\partial y_k^{p+U-1}(s)}{\partial t_j^p}\frac{\partial t_j^p}{\partial\omega_{ij}(r)}$$

$$=-\sum_{k=1}^{N}\sum_{s=1}^{V}(d_k^{p+U-1}(s)-y_k^{p+U-1}(s))\omega_{jk}(s)\phi'(\frac{s_j^p-b_j}{a_j})\frac{x_i^p(r)}{a_j} \quad (5)$$

Where, $\omega_{ij}(r)$ can affect all level component of output layer, so

$$\frac{\partial E^p}{\partial y_k^{p+U-1}}=-\sum_{k=1}^{N}\sum_{s=1}^{V}(d_k^{p+U-1}(s)-y_k^{p+U-1}(s))$$

$$\delta_{jk}^p(s)=\frac{\partial E^p}{\partial\omega_{jk}(s)}=\frac{\partial E^p}{\partial y_k^{p+U-1}(s)}\frac{\partial y_k^{p+U-1}(s)}{\partial\omega_{jk}(s)}$$

$$=-(d_k^{p+U-1}(s)-y_k^{p+U-1}(s))t_j^p$$

$$=-(d_k^{p+U-1}(s)-y_k^{p+U-1}(s))\phi(\frac{s_j^p-b_j}{a_j}) \quad (6)$$

But, $\omega_{ij}(s)$ can only affect kth level component of output layer, so

$$\frac{\partial E^p}{\partial y_k^{p+U-1}}=-(d_k^{p+U-1}(s)-y_k^{p+U-1}(s))$$

$$\delta_{aj}^p=\frac{\partial E^p}{\partial a_j}=\frac{\partial E^p}{\partial y_k^{p+U-1}(s)}\frac{\partial y_k^{p+U-1}(s)}{\partial t_j^p}\frac{\partial t_j^p}{\partial a_j}$$

$$=-\sum_{k=1}^{N}\sum_{s=1}^{V}(d_k^{p+U-1}(s)-y_k^{p+U-1}(s))\omega_{jk}\phi'(\frac{s_j^p-b_j}{a_j})(\frac{s_j^p-b_j}{a_j^2}) \quad (7)$$

$$\delta_{bj}^p=\frac{\partial E^p}{\partial b_j}=\frac{\partial E^p}{\partial y_k^{p+U-1}(s)}\frac{\partial y_k^{p+U-1}(s)}{\partial t_j^p}\frac{\partial t_j^p}{\partial b_j}$$

$$=-\sum_{k=1}^{N}\sum_{s=1}^{V}(d_k^{p+U-1}(s)-y_k^{p+U-1}(s))\omega_{jk}\phi'(\frac{s_j^p-b_j}{a_j})(-\frac{1}{a_j}) \quad (8)$$

Step four, Error back propagation and modify network parameters.

Computational formulas are

$$\omega_{ij}(r)(t+1)=\omega_{ij}(r)(t)-\eta\delta_{ij}^p(r)+\lambda[\omega_{ij}(r)(t)-\omega_{ij}(r)(t-1)] \quad (9)$$

$$\omega_{jk}(s)(t+1)=\omega_{jk}(s)(t)-\eta\delta_{jk}^p(s)+\lambda[\omega_{jk}(s)(t)-\omega_{jk}(s)(t-1)] \quad (10)$$

$$a_j(t+1)=a_j(t)-\eta\delta_{aj}^p+\lambda[a_j(t)-a_j(t-1)] \quad (11)$$

$$b_j(t+1)=b_j(t)-\eta\delta_{bj}^p+\lambda[b_j(t)-b_j(t-1)] \quad (12)$$

Step five, Input next sample, namely set $p=p+1$. If $p\le P$, then go to *Step two.*

Step six, Calculate total error of the network

$$E=\frac{1}{P}\sum_{p=1}^{P}E^p=\frac{1}{2P}\sum_{p=1}^{P}\sum_{k=1}^{N}\sum_{s=1}^{V}(d_k^{p+U-1}(s)-y_k^{p+U-1}(s))^2 \quad (13)$$

And judge if E is smaller than preset progress value $\varepsilon(\varepsilon>0)$. If $E<\varepsilon$ or $n>N$, stop the learning of network, otherwise, set $n=n+1$ and $p=1$, go to Step two.

It is needed to explain that, in the process of derivation as an example the wavelet function is taken as activation function of the neural network. In fact, other functions can be used as activation functions too, such as Sigmoid Function: S shape function. But in engineering practice how the final effects to use these functions are depends on a series of simulation experiments, in which a lot of tests and comparisons should be done.

5. System Design and Implementation

5.1. Overa Design

5.1.1. Requirement Regulation

Firstly, users acquire the number of bacterial colonies on temperature, salinity, PH, low temperature, temperature +PH and temperature + salinity at several time points by experiments. Secondly the system storages related experimental data, and begins neural network simulation according to the experimental data and finally predicts the number of bacterial colonies of vibrio parahaemolyticus on each time point.

This system consists of the following three parts.

Data management

Data management includes data addition, data deletion, data loading, data query. Users add the experimental data into the database for management through data addition and can delete the added data if needed.

According to the experimental conditions, the number of bacterial colonies of vibrio parahaemolyticus could be inquired through data query. Data loading can load experimental data into the system under certain conditions. It is needed to load the relevant data before simulation.

Data simulation

According to the experimental data of a certain state, the

number of bacterial colonies at all time points in the state could be simulated through the BP neural network algorithm.

Data prediction

According to the experimental data of a certain state, the number of bacterial colonies at a certain time point in the state could be predicted through the BP neural network algorithm. This time point can be the one that the user has not acquired by experiments. In this way, users can predict the number of bacterial colonies at each time point through a limited number of experiments.

5.1.2. Overall Function Structure

According to the demand, the system is divided into three parts: data management module, data simulation module, data prediction module. Data management module includes functions of data addition, data deletion, data query and data loading. Overall function structure diagram is as shown in Figure 3.

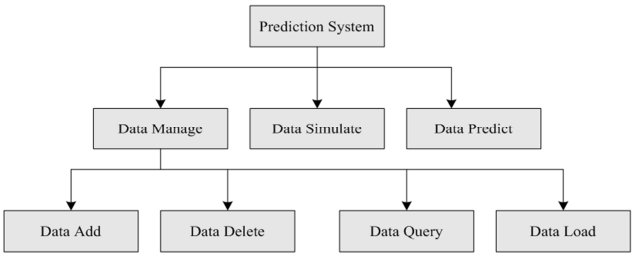

Figure 3. *Overall function structure.*

5.1.3. Basic Process Flow

The process of data simulation and prediction is shown in Figure 4. The growth experiment data of vibrio parahaemolyticus in bread shrimp in the six states of temperature, salinity, pH, temperature, temperature + PH, temperature + salinity at some time points are acquired through experiments in the laboratory. Then, these data are made as initial input data to a neural network and processed as input normalization, and finally the algorithm of Back Propagation Learning is taken for training the neural network. The trained neural network can be used for predicting the number of bacterial colonies of vibrio parahaemolyticus for each time point.

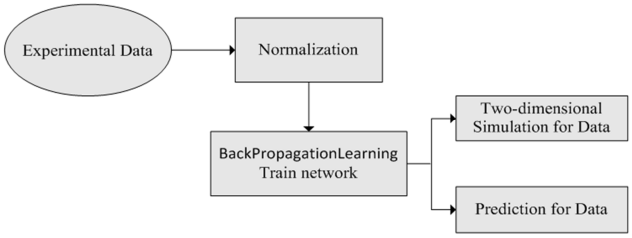

Figure 4. *Process of data simulation and prediction.*

5.1.4. Overall Scheme

The system is developed as an easy-to-use microbial growth simulation and prediction system, in which the growth experimental data of vibrio parahaemolyticus in bread shrimp in the six states of temperature, salinity, pH, low temperature, temperature + PH, temperature + salinity at some time points are taken as input data for neural network. Neural network algorithm, Back Propagation Learning, is used for neural network simulation. The program structure of the system is to take Visual Studio 2008 as its front, SQL Server2008 as its database, and use open source AForge.NET for its framework. The front desk interface displays two-dimensional data graphics, while in the background the system loads experimental data stored in the database, applies open source framework AForge.NET for neural network simulation, and outputs the prediction results to the front desk interface.

5.2. Detailed Design

5.2.1. Database Design

The background database of the system uses SQL Server database, which is used to store the experimental growth data of vibrio parahaemolyticus. Interface connects to the database through the System. Data. SqlClient provided by the C# namespace method. Experimental data of vibrio parahaemolyticus in bread shrimp in six states of temperature, salinity, pH, low temperature; temperature + PH, temperature + salinity are provided by experiments in laboratory, which can be changed with a certain period of time. Six pieces of corresponding data tables in the database are established to store data.

i. Temperature table: Temperature table is designed for storing the experimental growth data of vibrio parahaemolyticus in bread shrimp in different temperature and time. The temperature table is shown in Table 1.

Table 1. *Temperature table.*

Field name	Field description	Data type	Can be empty	Constraint	Remarks
temperature	temperature	float		Combined primary key	
hour	time	float		Combined primary key	
amount	number of bacterial colonies	float	no		

ii. Salt table: Salt table is designed for storing the experimental growth data of vibrio parahaemolyticus in bread shrimp in different salt and time. The salt table is shown in Table 2.

Table 2. *Salt table.*

Field name	Field description	Data type	Can be empty	Constraint	Remarks
salt	salt	float		Combined primary key	
hour	time	float		Combined primary key	
amount	number of bacterial colonies	float	no		

iii. PH table: PH table is designed for storing the experimental growth data of vibrio parahaemolyticus in bread shrimp in different PH and time. The PH table is shown in Table 3.

Table 3. *PH table.*

Field name	Field description	Data type	Can be empty	Constraint	Remarks
PH	PH	float		Combined primary key	
hour	time	float		Combined primary key	
amount	number of bacterial colonies	float	no		

iv. LowSurvival table: LowSurvival table is designed for storing the experimental growth data of vibrio parahaemolyticus in bread shrimp in different low temperature and time. The LowSurvival table is shown in Table 4.

Table 4. *LowSurvival table.*

Field name	Field description	Data type	Can be empty	Constraint	Remarks
Low Temperature	low temperature	float		Combined primary key	
hour	time	float		Combined primary key	
amount	number of bacterial colonies	float	no		

v. TempreturePh table: TemperaturePh table is designed for storing the experimental growth data of vibrio parahaemolyticus in bread shrimp in different temperature, PH and time. The temperature+PH table is shown in Table 5.

Table 5. *TemperaturePh table.*

Field name	Field description	Data type	Can be empty	Constraint	Remarks
temperature	temperature	float		Combined primary key	
ph	ph	float		Combined primary key	
hour	time	float		Combined primary key	
amount	number of bacterial colonies	float	no		

vi. TemperatureSalt table: TemperatureSalt table is designed for storing the experimental growth data of vibrio parahaemolyticus in bread shrimp in different temperature, salt and time. The temperature+salt table is shown in Table 6.

Table 6. *TemperatureSalt table.*

Field name	Field description	Data type	Can be empty	Constraint	Remarks
temperature	temperature	float		Combined primary key	
salt	salt	float		Combined primary key	
hour	time	float		Combined primary key	
amount	number of bacterial colonies	float	no		

5.2.2. System Function Module Detailed Design

i. Neural network training module: The basic principle of neural network training is that experimental data are used as its inputs and needed to be normalized, and the network is trained through an activation function. In the system we use Sigmoid Function as its activation function. After training, the network can be used to perform data prediction. The key codes are as follows.

```
privateconstint tempreture = 0;// tempreture -0，salty -1, ph-2，lowSurvival -3，temprePh ph-4，tempreSalty -5
privateconstint salty = 1;
privateconstint ph = 2;
privateconstint lowSurvival = 3;
privateconstint temprePh = 4;
privateconstint tempreSalty = 5;

//Two-dimensional
constint inputNum = 1;
constint outputNum = 1;
//Training data
int trainNum;
double[][] trainInput;
double[][] trainOutput;
//The maximum and minimum data are used for normalization
double[] maxInput = newdouble[inputNum];
double[] minInput = newdouble[inputNum];
```

```
double[] maxOutput = newdouble[outputNum];
double[] minOutput = newdouble[outputNum];
//Construct chart data
double[,] data;

privateActivationNetwork network; //neural network

public BPNeural()
        {
            init();
        }

privatevoid init()
        {

        }

publicvoid setChart(Chart chart)
        {
this.chart = chart;
        }

//Input normalization
privatedouble premnmxInput(double num, double min, double max)
        {
double xFactor = 2.0 / (max - min);
return (num - min) * xFactor - 1.0;
        }

//Output normalization
privatedouble premnmxOutput(double num, double min, double max)
        {
double yFactor = 1.7 / (max - min);
return (num - min) * yFactor - 0.85;
        }

//Output counter normalization
privatedouble getOriginalOutput(double num, double min, double max)
        {
double yFactor = 1.7 / (max - min);
return (num + 0.85) / yFactor + min;
        }

//Training sample data
privatevoid getTrainData()
        {
string queryString = null;
if (category == tempreture)
            queryString = "SELECT hour,amount FROM tempreture WHERE tempreture = " + condition1;
if (category == salty)
            queryString = "SELECT hour,amount FROM salt WHERE salt = " + condition1;
if (category == ph)
            queryString = "SELECT hour,amount FROM ph WHERE ph = " + condition1;
if (category == lowSurvival)
            queryString = "SELECT hour,amount FROM lowSurvival WHERE tempreture = " + condition1;
if (category == temprePh)
            queryString = "SELECT hour,amount FROM tempreturePh WHERE tempreture = " + condition1 +" AND ph = "+condition2;
if (category == tempreSalty)
            queryString = "SELECT hour,amount FROM tempretureSalt WHERE tempreture = " + condition1 + " AND salt = " + condition2;

//Initial max min data
for (int i = 0; i < inputNum; ++i)
            {
                maxInput[i] = double.MinValue;
                minInput[i] = double.MaxValue;
            }
for (int i = 0; i < outputNum; ++i)
            {
                maxOutput[i] = double.MinValue;
                minOutput[i] = double.MaxValue;
            }

// read maximum 50 points
int maxTempNum = 50;
double[][] tempInputData = newdouble[maxTempNum][];
double[][] tempOutputData = newdouble[maxTempNum][];
int num = 0;

using (SqlConnection connection = newSqlConnection(connectionString))
            {
SqlCommand command = newSqlCommand(queryString, connection);
            connection.Open();
SqlDataReader reader = command.ExecuteReader();
try
            {
while ((num < maxTempNum) && reader.Read())
                {
//handle data
//train input
                    tempInputData[num] = newdouble[inputNum];
for (int j = 0; j < inputNum; j++)
                    {
                        tempInputData[num][j] = double.Parse(reader[0].ToString());
// search for min value
if (tempInputData[num][j] < minInput[j])
                            minInput[j] = tempInputData[num][j];
// search for max value
```

```
   if (tempInputData[num][j] > maxInput[j])
                                        maxInput[j]     =
tempInputData[num][j];
                                }
   // trainOutput
                                tempOutputData[num]     =
newdouble[outputNum];
   for (int j = 0; j < outputNum; j++)
                                {
                                    tempOutputData[num][j]
= double.Parse(reader[1].ToString());
   // search for min value
   if (tempOutputData[num][j] < minOutput[j])
                                        minOutput[j]    =
tempOutputData[num][j];
   // search for max value
   if (tempOutputData[num][j] > maxOutput[j])
                                        maxOutput[j]    =
tempOutputData[num][j];
                                }
                                num++;

   // allocate and set trainInput trainOutput
                                trainNum = num;
                                trainInput              =
newdouble[trainNum][];
                                trainOutput             =
newdouble[trainNum][];
   for (int j = 0; j < trainNum; j++)
                                {
                                    trainInput[j]       =
newdouble[inputNum];
                                    trainOutput[j]      =
newdouble[outputNum];
                                }
   Array.Copy(tempInputData, 0, trainInput, 0, num);
   Array.Copy(tempOutputData, 0, trainOutput, 0, num);
                            }
                        }
   finally
                        {
   // Always call Close when done reading.
                            reader.Close();
                            connection.Close();
                        }

   //Construct data displayed in chart
                        data = newdouble[trainNum, 2];
   for (int i = 0; i < trainNum; i++)
                        {
                            data[i, 0] = trainInput[i][0];
                            data[i, 1] = trainOutput[i][0];
                        }

   // Display chart boundary label
                        labelMinHour.Text               =
minInput[0].ToString();
                        labelMaxHour.Text               =
maxInput[0].ToString();
                        labelMinAmount.Text             =
minOutput[0].ToString();
                        labelMaxAmount.Text             =
maxOutput[0].ToString();

   // normalization
   for (int i = 0; i < trainNum; ++i)
                        {
   for (int j = 0; j < inputNum; ++j)
                            {
                                trainInput[i][j]        =
premnmxInput(trainInput[i][j], minInput[j], maxInput[j]);
                            }
   for (int j = 0; j < outputNum; j++)
                            {
                                trainOutput[i][j]       =
premnmxOutput(trainOutput[i][j],        minOutput[j],
maxOutput[j]);
                            }
                        }
                    }
                }

   privatevoid trainNetwork(int inputNum, int hideNode, int
outputNum, double learningRate, double Momentum, int
iterate)
                {
   //Training network

   //create multi-layer neural network
                    network                             =
newActivationNetwork(newBipolarSigmoidFunction(2),
inputNum, hideNode, outputNum);
   //create teacher
   BackPropagationLearning         teacher             =
newBackPropagationLearning(network);
   //set learning rate and momentum
                    teacher.LearningRate = learningRate;
                    teacher.Momentum = Momentum;

   int iteration = 0;
   double error = 0;

   while (iteration < iterate)
                    {
                        error = teacher.RunEpoch(trainInput,
trainOutput) / trainNum;
                        ++iteration;
                    }
                }
```

ii. Two-dimensional simulation of data: The trained neural network can predict the growth of vibrio parahaemolyticus at any time, and the predicted value can be simulated by a two-dimensional image. The

key codes are as follows.

```
//Two-dimensional simulation
privatevoid simulateTwoDimension()
        {
            chart.RangeX = newRange((float)minInput[0], (float)maxInput[0]);
            chart.UpdateDataSeries("data", data);
            chart.UpdateDataSeries("solution", null);

double[,] solution = newdouble[50, 2];
double[] networkInput = newdouble[1];

// calculate X values to be used with solution function
for (int j = 0; j < 50; j++)
            {
                solution[j, 0] = chart.RangeX.Min + (double)j * chart.RangeX.Length / 49;
            }

// calculate solution
for (int j = 0; j < 50; j++)
            {
                networkInput[0] = premnmxInput(solution[j, 0], minInput[0], maxInput[0]);//normalization
                solution[j, 1] = getOriginalOutput(network.Compute(networkInput)[0], minOutput[0], maxOutput[0]);
            }
            chart.UpdateDataSeries("solution", solution);

        }
```

iii. *Data prediction:* The trained neural network can predict the growth of vibrio parahaemolyticus for a single time entered by user. The key codes are as follows.

```
publicdouble predict(double input)
        {
double[] networkInput = newdouble[1];
            networkInput[0] = premnmxInput(input, minInput[0], maxInput[0]);//normalization
return getOriginalOutput(network.Compute(networkInput)[0], minOutput[0], maxOutput[0]);
    }
```

5.3. System Implementation

5.3.1. Implementation of Data Management Module

In the data management module, the experimental data can be added, deleted, queried, and data can be also loaded from the database.

Data addition is as shown in Figure 5. Enter data that is needed to add in the input box, and then click the Add button to add.

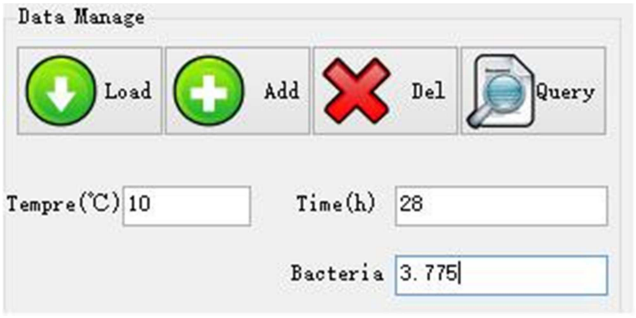

Figure 5. *Data Add.*

Data query interface is similar to data addition, enter the data into the corresponding input boxes to temperature and time, and then click on the Query button to search. If the queried data does not exist in the database, it will give tips.

Data loading is as shown in Figure 6, after clicking on the Load button, the experimental data of the corresponding state in the database will be loaded into the table.

Tempre(℃)	Time(h)	lgNt(cfu/mL)
10	0	2.977
10	20	3.35
10	28	3.775
10	44	4.172
10	52	4.365
10	68	5.129
10	74	5.85
10	94	6.418
10	100.5	7.066
10	120	8.07
10	138	9.395
10	145	9.926
14	0	3.082
14	14.5	3.361
14	24	4.175
14	38.5	4.875
14	47.5	5.671
14	62.5	6.146
14	70.5	6.939
14	88	7.593
14	94.5	7.998
14	110.5	8.455

Figure 6. *Data loading.*

Data deletion interface is similar to data loading interface, select a record in the table, and then click the Del button, you can delete the data from the database.

5.3.2. Implementation of Data Simulation Module

Data simulation is as shown in Figure 7. Select the condition that is needed to be simulated at the top of the list box, and then click on the Simul button, you can perform neural network training and two-dimensional data simulation.

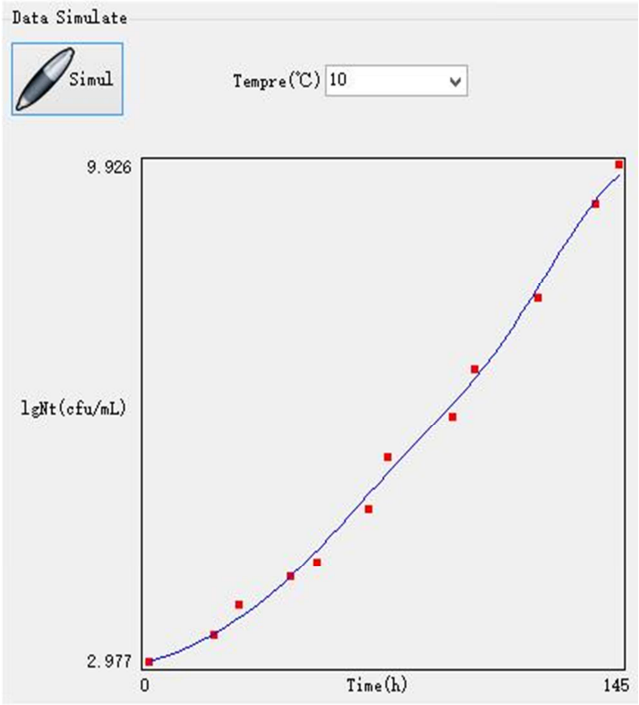

Figure 7. *Data simulation.*

5.3.3. Implementation of Data Prediction Module

Data prediction is as shown in Figure 8. Enter the prediction condition in the corresponding condition box, and then click the Pred button, and the results of the prediction will be displayed on the bacterial value box.

Figure 8. *Data prediction.*

6. Conclusions

According to the requirements of a research project, sub project of National Science and Technology Support Program of China: Key Integrated and Demonstrated Technologies for Quality and Safety Control in Aquatic Product Processing Process (No. 2012BAD29B06), the authors of this paper have developed a set of bread shrimp microbial growth simulation and prediction system. The system is established by taking vibrio parahemolyticus in bread shrimp as research objects, according to effects of temperature, salt and time on their growth, and employing neural network technology. The system consists of three parts: data management, data simulation and data prediction, which would provide an effective analytical tool for bread shrimp safe production. In order to improve its compatibility, the system is developed by using C# on Visual Studio 2008 platform, and its design and implementation are based on Aforge.NET framework and sliding-window modeling method.

The system is developed as an easy-to-use microbial growth simulation and prediction system, in which experimental growth data of vibrio parahaemolyticus in bread shrimp in the six states of temperature, salinity, pH, low temperature, temperature + PH, temperature + salinity at some time points are taken as input data for neural network. Neural network algorithm, Back Propagation Learning, is used for neural network simulation. The program structure of the system is to take Visual Studio 2008 as its front, SQL Server2008 as its database, and use open source AForge.NET for its framework. The front desk interface displays two-dimensional data graphics, while in the background the system loads experimental data stored in the database, applies open source framework AForge.NET for neural network simulation, and outputs the prediction results to the front desk interface. After tested carefully, the system can meet the requirements of the project design.

Through intelligent simulation and prediction in the system, users can predict vibrio parahemolyticus growth and survival data in various conditions only providing a limited number of vibrio parahemolyticus growth and survival data in the conditions. By this way, users do not need to perform experiments in each condition, greatly reducing the number of experiments and saving experimental time and costs.

Acknowledgment

This research work was funded by the sub project of National Science and Technology Support Programof China under Grant No. 2012BAD29B06 and the Science and Technology Project of Guangdong Province under Grant No. 2014B040401014.

References

[1] Xiao Laisheng, A Neural Network-Based Multi-Dimensional Simulation Modeling Approach for Food Microbial Growth, Advanced Science Letters, ISSN: 1936-6612, Volume 6, Pages 400-405(15 March 2012).

[2] Wang Zhengxia, Xiao Laisheng, "Prediction Model of Ocean Food Microbe Growth Based on Neural Network and Its Simulation", CCCA2011, Volume II, p160-165, ISBN: 978-1-61284-102-1, 2011.

[3] Zhou Kang, Liu Shouchun, Li Pinglan, Ma Changwei, Peng Zhaohui, New Advances in Predictive Food Microbial Growth Model, Microbiology, APR 20, 2008, 35(4): 589-594.

[4] Zhang Yuting, Meng Yaquan, Yan Guoting, Application of Matlab in microbial growth forecast model, Hebei Chemical Industry, Vol. 31, No.1, Jan. 2008, pp.20-22.

[5] Zhang Yuting, Wu Kun, Zhang Chunhui, Wang Yufeng, Selection and application of microorganism growth model in cold fresh pork, Meat Industry, 2005, No.11, Totally 295, pp.23-25.

[6] Yang Hongju, Nan Qingxian, Establishment of main corruption microbial growth model in cold pork, Storage and Process, 2004, No.3, pp.7-10.

[7] Liu Xinyou, Nan Haijun, Hao Yaqing, Gao Yuanjun, Tang Xueyan, Zhang Fang, Research on microbial growth model for fresh cut apples in storage period, Journal of Henan Agricultural Sciences, 2007, No.3, pp.88-91.

[8] I. Stamati, F. Logist, E. Van Derlinden, J.-P. Gauchi, J. Van Impe, Optimal experimental design for discriminating between microbial growth models as function of suboptimal temperature, Mathematical Biosciences 250 (2014) 69–80.

[9] I. Stamati, F. Logist, S. Akkermans, E. Noriega Fernández, J. Van Impe, On the effect of sampling rate and experimental noise in the discrimination between microbial growth models in the suboptimal temperature range, Computers and Chemical Engineering 85 (2016) 84–93.

[10] Si Zhu, Guibing Chen, Numerical solution of a microbial growth model applied to dynamic environments, Journal of Microbiological Methods 112 (2015) 76–82.

[11] Anastasia Lytou, Efstathios Z. Panagou, George-John E. Nychas, Development of a predictive model for the growth kinetics of aerobic microbial population on pomegranate marinated chicken breast fillets under isothermal and dynamic temperature conditions, Food Microbiology 55 (2016) 25e31.

[12] Albert Ibarz • Pedro E. D. Augusto, An autocatalytic kinetic model for describing microbial growth during fermentation, Bioprocess Biosyst Eng (2015) 38:199–205.

[13] Long Liu • Zhiguo Guo • Jianjiang Lu •Xiaolin Xu, Kinetic model for microbial growth and desulphurisation with Enterobacter sp., Biotechnol Lett (2015) 37:375–381.

[14] María Jesús Munoz-Lopez, Maureen P. Edwards, Ulrike Schumann and Rober s. Anderssen, Multiplicative modelling of four-phase microbial growth, Pacific Journal ofMathematics for Industry (2015) 7: 7.

[15] Yury V. Bukhman • NathanW. DiPiazza • Jeff Piotrowski • Jason Shao• Adam G. W. Halstead • Minh Duc Bui • Enhai Xie • Trey K. Sato, Modeling Microbial Growth Curves with GCAT, Bioenerg. Res. (2015) 8: 1022–1030.

[16] Yong-guang Yin, Yun Ding, A close to real-time prediction method of total coliform bacteria in foods based on image identification technology and artificial neural network, Food Research International 42 (2009) 191–199.

[17] M. Hajmeer, I. Basheer, A probabilistic neural network approach for modeling and classification of bacterial growth/no-growth data, Journal of Microbiological Methods 51 (2002) 217–226.

[18] M. Cheroutre-Vialette, A. Lebert, Application of recurrent neural network to predict bacterial growth in dynamic conditions, International Journal of Food Microbiology 73 (2002) 107–118.

[19] A. H. Geeraerd, C. H. Herremans, C. Cenens, J. F. Van Impe, Application of artificial neural networks as a non-linear modular modeling technique to describe bacterial growth in chilled food products, International Journal of Food Microbiology 44 (1998) 49–68.

[20] Adolf Willem Schepers, Jules Thibault, Christophe Lacroix, Comparison of simple neural networks and nonlinear regression models for descriptive modeling of Lactobacillus helveticus growth in pH-controlled batch cultures, Enzyme and Microbial Technology 26 (2000) 431–445.

[21] Francisco Fernández-Navarroa, Antonio Valero, César Hervás-Martínez, Pedro A. Gutiérrez, Rosa M. García-Gimeno, Gonzalo Zurera-Cosano, Development of a multi-classification neural network model to determine the microbial growth/no growth interface, International Journal of Food Microbiology 141 (2010) 203–212.

[22] Francisco Fernández-Navarroa, César Hervás-Martíneza, M. Cruz-Ramírez, Pedro Antonio Gutiérrez, Antonio Valero, Evolutionary q-Gaussian Radial Basis Function Neural Network to determine the microbial growth/no growth interface of Staphylococcus aureus, Applied Soft Computing 11 (2011) 3012–3020.

[23] Daniel S. Esser • Johan H. J. Leveau • Katrin M. Meyer, Modeling microbial growth and dynamics, Appl Microbiol Biotechnol (2015) 99:8831–8846.

[24] Wang Zhengxia, Xiao Laisheng, Lin Honghong, Qiu Shuzhong, Huang Chiyun, Lei Xiaoling, Intelligent General Predictive Platform for Sea Food Microorganism Growth, Computer Knowledge and Technology, Vol 7, No. 19, July 2011.

[25] http://www.aforgenet.com/aforge/framework/.

[26] http://www.codeproject.com/Articles/16447/Neural-Networks-on-C.

[27] Xiao-sheng LIU, Xiao HU, Ting-li WANG, Rapid assessment of flood loss based on neural network ensemble, Trans. Nonferrous Met. Soc. China 24(2014) 2636−2641.

[28] Chengying Gong, Hui He, Research of AForge.NET in Motion Video Detection, Applied Mechanics and Materials Vols. 496-500 (2014), pp 2150-2153.

[29] Suraj Verma*, Prashant Pillai and Yim-Fun Hu, Development of an eye-tracking control system using AForge.NET framework, Int. J. Intelligent Systems Technologies and Applications, Vol. 11, Nos. 3/4, 2012.

[30] ŽIDEK Kamil, RIGASOVá Eva, Diagnostics of Products by Vision System, Applied Mechanics and Materials Vol. 308 (2013) pp 33-38.

[31] Ondrej Krejcar, Utilization of C# Neural Networks Library in Industry Applications, ICeND 2011, CCIS 171, pp. 61-72, 2011.

[32] Laisheng Xiao, “A sliding-window modeling approach for neural network”, International Journal of Control and Automation, ISSN 2005-4297, Vol.7, No.8, Aug. 2014.