# CHAPTER III

# PROPOSED SOLUTIONS

## 3.1 Objectives

The main objective of this dissertation is to propose a new cost function to capture the parallel behavior between the desired target and the output efficiently in order to reduce the amount of oscillation in the learning process.

## 3.2 Scope of Work and Limitations

In this dissertation, the scope of work is constrained as follows:

1. Study the cost function using in the supervised learning neural network.
2. Constructing the new cost function to mimimise the training time.
3. Implement the cost function with the benchmark test sets which the approximation problems.
4. Comparing the parallel between the desired target and the network output by measuring the values of *cosine* between these two vectors.

## 3.3 Network Architecture

Figure 3.1 depicts the network architecture of the proposed model which composes of input, hidden and output layers. Let $\mathbf{o1} = \{\mathbf{o1_1}, \mathbf{o1_2}, \ldots, \mathbf{o1_i}, \ldots, \mathbf{o1_p}\}$ be the set of input vector. An input vector $\mathbf{o1_i}$ in a $d$ dimensional space is defined as $\mathbf{o1_i} = [o1_{i,1}, o1_{i,2}, \ldots, o1_{i,k}, \ldots, o1_{i,d}]^T$. Each $o1_{i,k}$ is fed forward to $h$ hidden neurons whose outputs are combined with weights $w32_j$ into the network output $o3_i$.

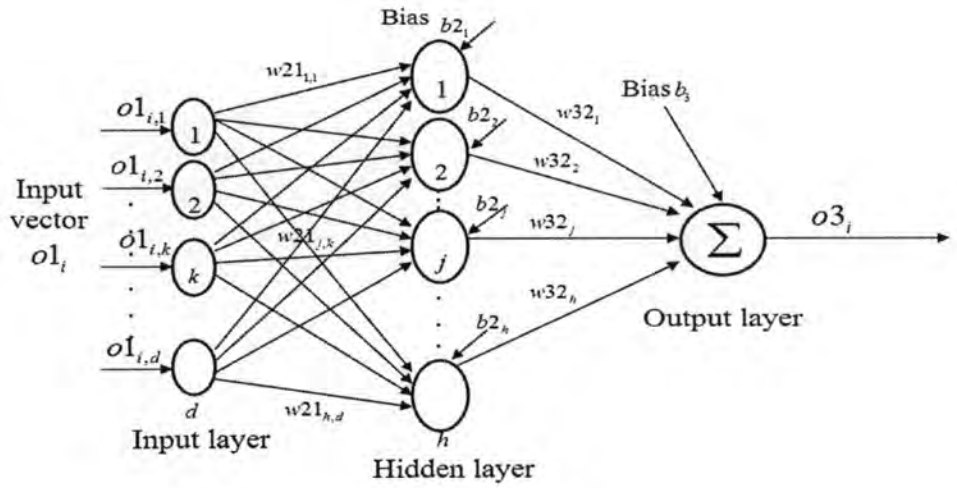



Figure 3.1: The structure of the neural network.

In this research, we use a single-layer network with the hyperbolic tangent function as an activation function. From Figure 3.1, we use the following notations.

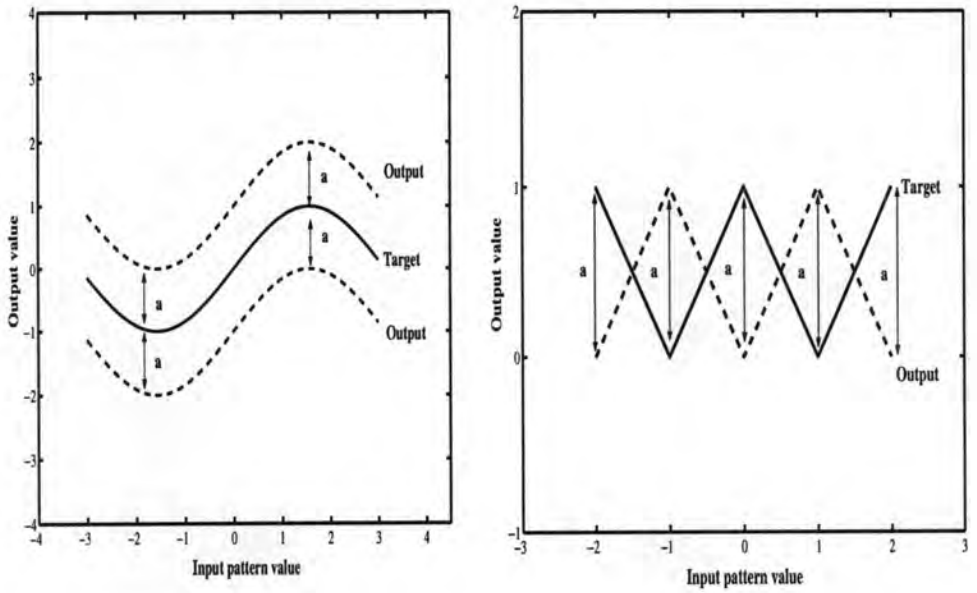
| | | | |
|---|---|---|---|
| $\mathbf{o1_i}$ | = | $[o1_{i,1}, o1_{i,2}, \ldots, o1_{i,d}]^T$ | : input vector of the $i^{th}$ pattern |
| **o2** | = | $[o2_1, o2_2, \ldots, o2_h]^T$ | : output vector from hidden unit |
| $\mathbf{o3_i}$ | | | : output vector of the $i^{th}$ pattern |
| **b2** | = | $[b2_1, b2_2, \ldots, b2_h]^T$ | : bias vector of the hidden layer |
| **b3** | = | $[b3]$ | : bias vector of the output layer |
| **w21** | = | $[w21_{1,1}, w21_{1,2}, \ldots, w21_{h,d}]^T$ | : weight vector from input layer to hidden layer |
| **w32** | = | $[w32_1, w32_2, \ldots, w32_h]^T$ | : weight vector from hidden layer to output layer |
| $d$ | | | : number of input dimension |
| $h$ | | | : number of hidden unit |
| $p$ | | | : number of learning pattern |
| $i, j, k$ | | | : indices of dimensions |

## 3.4 A New Behavior-Based Cost Function

According to apply the neural cellular automata on this work that implemented back-propagation neural network (BPNN) with a structure of feedforward multilayer perceptron for simulating the diffusion characteristics of water, there is still a constraint of forecasting efficiency due to the cost function used for the backpropagation neural network. Although the Backpropagation (BP) algorithm is a well known learning algorithm in supervised learning neural network, it has a slow convergence rate. The Levenberg-Marquardt (LM) algorithm is an improve-

ment of the BP method to expedite the learning step. Therefore, this research applies the LM algorithm in learning process.

From equation 2.25, the cost function is used for minimizing the difference between target and output values obtained from the backpropagation neural network of every training pattern. The BPNN becomes stable when the selected error criterion ($a$) defined by cost function is reached.




(a) (b)

Figure 3.2: The acceptable error criterion ($a$) the cost function used in the backpropagation neural network.

However, the output and target values may not have parallel differences as some output values can be higher or lower than the target value (as shown in Figure 3.4(a)). Furthermore, the output values may fluctuate within an acceptable error criterion ($a$) as shown in Figure 3.4(b).

Therefore, this work proposes a new behavior-based cost function capable of giving parallel difference between the output and target values in which the output is not necessarily the same as that of the target. Only certain difference at output pattern is acceptable.

Accordingly the idea of our proposed cost function is shown in Figure 3.3. This Figure shows the thick line representing the target vector, and the dotted line representing the output vector. The alphabet $a$ represents the acceptable distance between the target and the output values, which are constantly different at every time step. The target values can be higher or lower than the output values, which are parallel to each other.

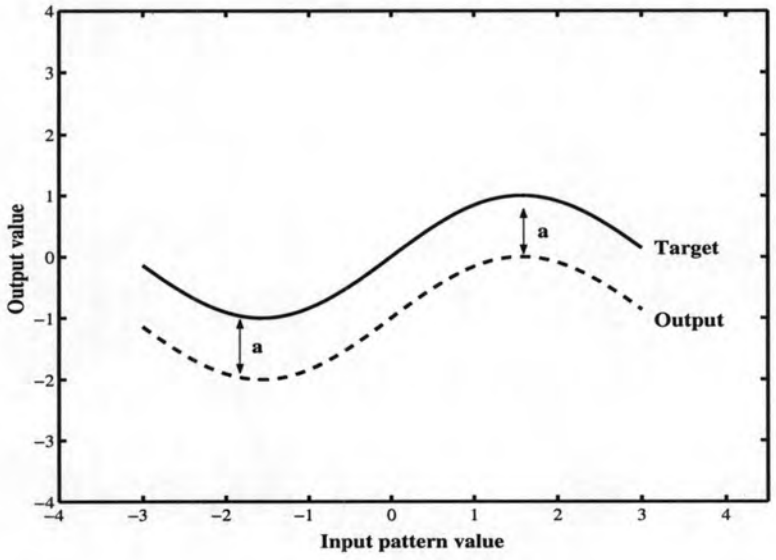



Figure 3.3: This figure shows the idea of creating a new behavior-based cost function.

From the mentioned principles, the proposed cost function is aimed to adjust the desired target and the network output so that they are parallel for every training pattern. The dot product was applied to achieve a new behavior-based cost function. The proposed cost function ($C = [c_1, c_2, ..., c_p]^T$) is defined by the following equation.

$$c_i = \frac{O_i \cdot T_i}{\|O_i\| \, \|T_i\|} \tag{3.1}$$

and

$$O_i = Vo_{i+1} - Vo_i \quad where \quad Vo_i = \begin{bmatrix} o1_i \\ o3_i \end{bmatrix} \tag{3.2}$$

$$T_i = Vt_{i+1} - Vt_i \quad where \quad Vt_i = \begin{bmatrix} o1_i \\ t_i \end{bmatrix} \tag{3.3}$$

In this proposed method, the hyperbolic tangent (tanh) is used as an activation function, instead of logistic sigmoid function in order to cope the negative values. An activation function, denoted by $f(v_k)$, defines the output of a neuron $k$ in term of the induced local field $f(v_k)$. The tanh function ($f(v_k)$ is defined by

$$f(v_k) = \alpha(\frac{2}{1 + e^{-2(v_k)}} - 1) \tag{3.4}$$

where, $v_k$ refers to the output from each neuron $k$.
$\alpha$ refers to the output range of the hidden unit and output unit.

So, the output from the hidden layer ($o2_j$) passed through the tanh function is defined by

$$f(o2_j) = \frac{2}{1 + e^{-2(o2_j)}} - 1 \tag{3.5}$$

where,

$$o2_j = \sum_{k=1}^{d} w21_{j,k} * o1_{i,k} + b2_j \tag{3.6}$$

Thus, the output of the input pattern $i^{th}$ ($o3_i$) passed through the tanh function is defined by

$$f(o3_i) = \frac{2}{1 + e^{-2(o3_i)}} - 1 \tag{3.7}$$

where,

$$o3_i = \sum_{j=1}^{h} w32_j * f(o2_j) + b3 \tag{3.8}$$

Using LM algorithm, the performance index $F(w)$ in our learning procedure is computed by

$$F(w) = C^T C \tag{3.9}$$

The weight adjustment ($\Delta w$) is obtained as following.

$$\Delta w = [J^T J + \mu I]^{-1} J^T C \tag{3.10}$$

and the Jacobian matrix ($J(w)$ ) is defined by the following matrix.

$$J(w) = \begin{bmatrix} \frac{\partial c_1}{\partial w_1} \frac{\partial c_1}{\partial w_2} \frac{\partial c_1}{\partial w_3} \cdots \frac{\partial c_1}{\partial w_n} \\ \frac{\partial c_2}{\partial w_1} \frac{\partial c_2}{\partial w_2} \frac{\partial c_2}{\partial w_3} \cdots \frac{\partial c_2}{\partial w_n} \\ \vdots \\ \frac{\partial c_i}{\partial w_1} \frac{\partial c_i}{\partial w_2} \frac{\partial c_i}{\partial w_3} \cdots \frac{\partial c_i}{\partial w_n} \\ \vdots \\ \frac{\partial c_p}{\partial w_1} \frac{\partial c_p}{\partial w_2} \frac{\partial c_p}{\partial w_3} \cdots \frac{\partial c_p}{\partial w_n} \end{bmatrix} \tag{3.11}$$

where, $w = [b3, b2_1, b2_2, \ldots, b2_h, w32_1, w32_2, \ldots, w32_h, w21_{1,1}, \ldots, w21_{h,d}]$ consists of all weight and bias vectors of the network.

Finally, the weight adjustment is defined as followed.

$$\begin{aligned} b3^{t+1} &= b3^t + \eta(\Delta b3) \\ b2_j^{t+1} &= b2_j^t + \eta(\Delta b2_j) \\ w32_j^{t+1} &= w32_j^t + \eta(\Delta w32_j) \\ w21_{j,k}^{t+1} &= w21_{j,k}^t + \eta(\Delta w21_{j,k}) \end{aligned} \tag{3.12}$$

where, $\eta$ refers to the learning rate.
$t$ refers to the index of the iteration.

### 3.4.1 Learning Procedure

From the proposed cost function, we use standard LM learning algorithm in order to speed up the convergence rate. For the proposed algorithm, the performance index $F(w)$ is computed from the proposed cost function $(C)$, equation 3.1.

## Setting $\alpha$ value

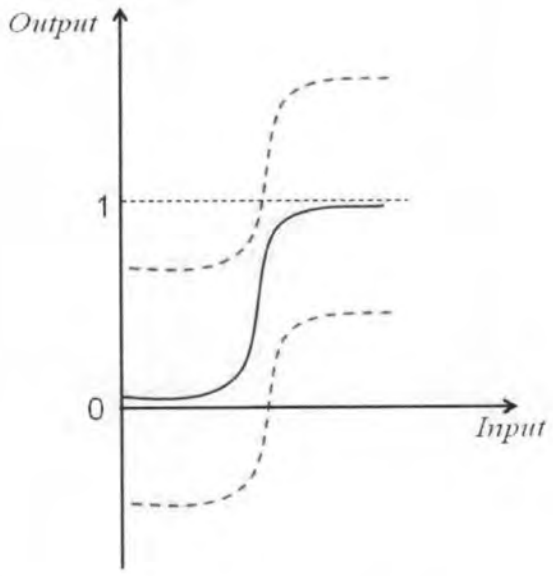
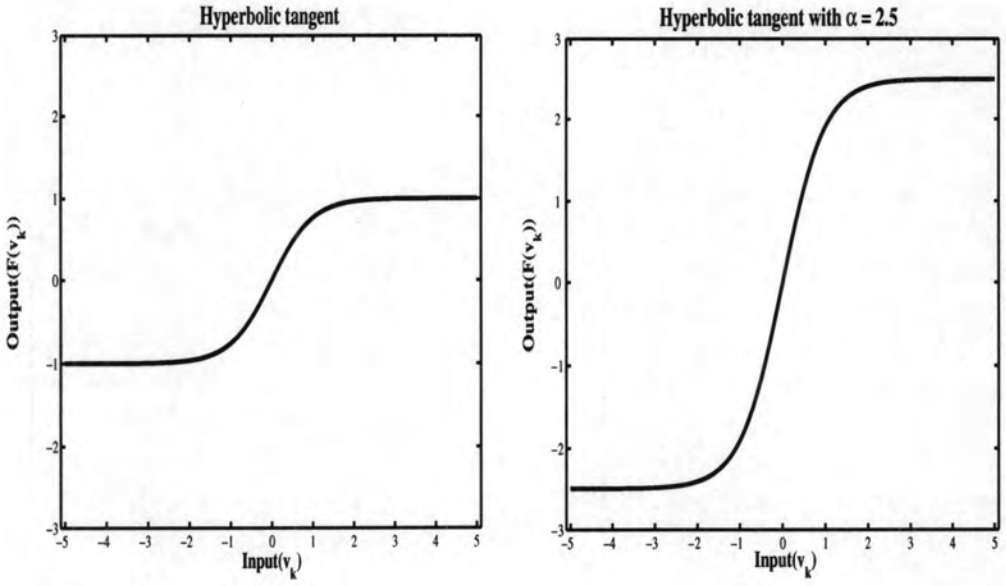



Figure 3.4: The concept of selecting the properly activation function.

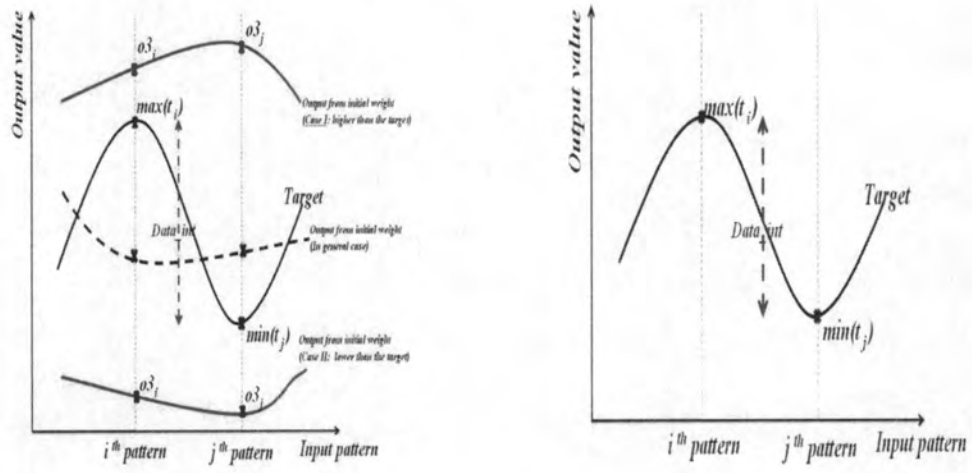
This research uses hyperbolic tangent function (Equation 3.4) as an activation function in order to cope the negative value for expanding the weight adjusting area. From the Figure3.4, the sigmoid function displays in the red line. According to the reason that we want to learn the behavior that have to parallel with the target, activation function should have enough space for adjusting the output to parallel with target value as shown in the dotted blue lines.

There is another variable $\alpha$ in the tanh function, to set the maximum and minimum values with the target range. Figure 3.5 shows the comparison of the standard hyperbolic tangent function for which the output range lied between minus one and one and the hyperbolic tangent function which the output range between $-2.5$ and 2.5. Figure 3.5(a) shows the normal hyperbolic tangent function. Figure 3.5(b) shows the hyperbolic tangent function with $\alpha$ equal to 2.5 so that the output range is between -2.5 and 2.5. Therefore, if $\alpha$ value is set appropriately and enough for weight adjusting area, the weight adjustment is flexible and help decreasing the training time.




(a) (b)

Figure 3.5: The comparison between standard hyperbolic tangent function and the hyperbolic tangent function with $alpha = 2.5$.




(a) (b)

Figure 3.6: The figure (a) shows the idea of setting the $\alpha$. (b) The first criterion.

From the initial weight, there are five criteria to set the $\alpha$ value. Figure 3.6 (a) shows the concept of setting the $\alpha$ value. Black line shows the target vector by $i^{th}$ input pattern giving the maximum target value. $j^{th}$ input pattern gives the minimum target value. Form initial weigh, we dont know whether output value will be higher or lower than target. However, it will stand in the same area of the

target as shown in the dotted line. Therefore, we must set the $\alpha$ value to cover in every case. Figure 3.6 (b) shows the concept of setting the $\alpha$ value in the first criterion. At least, $\alpha$ should have value to cover the target range. Therefore, the first criterion must be set the $\alpha$ to equal to the target range or equal to the different between the maximum and minimum value of the target. The first criterion $set(1)$ equals to the target interval ($data\_int$) by using the following equation.

$$set(1) = data\_int = max(t) - min(t) \tag{3.13}$$

where, $max(t)$ refers to the maximum value of the target vector of the $i^{th}$ pattern.
$min(t)$ refers to the minimum value of the target vector of the $j^{th}$ pattern.

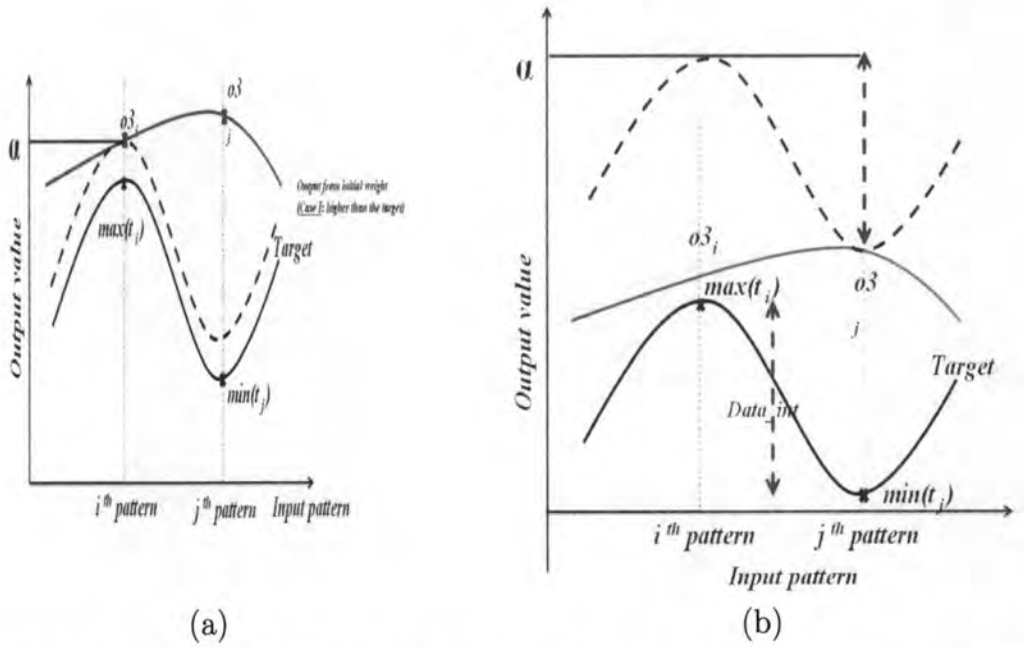



Figure 3.7: The figures show the idea of setting the $\alpha$ in (a) the second criterion. (b) The third criterion.

Figure 3.7 (a) shows the concept of setting the $\alpha$ value in case of initial weigh give the output at $i^{th}$ pattern ($o3_i$) higher than the target. The green line shows the output that receive form the initial weight. Consider the $i^{th}$ position that give the maximum target value, we set the $o3_i$ as the second criterion of the $\alpha$. Broken line shows that if we set the $\alpha$ equal to $o3_i$, we will have enough space for adjusting weight. Therefore, the second criterion ($set(2)$) equals to the absolute value of $o3_i$ in order to avoid the negative value.

$$set(2) = |o3_i| \tag{3.14}$$

Figure 3.7 (b) shows the concept of setting the $\alpha$ in the third criterion. If we consider $o3_j$ that is the minimum target value, we have to add the area at least $data\_int$ to get enough the target range value. Therefore, the third criterion

is obtained by adding the target range to the output at point $j$ ($o3_j$) to increase the adjusting area.

$$set(3) = |o3_j + data\_int| \tag{3.15}$$

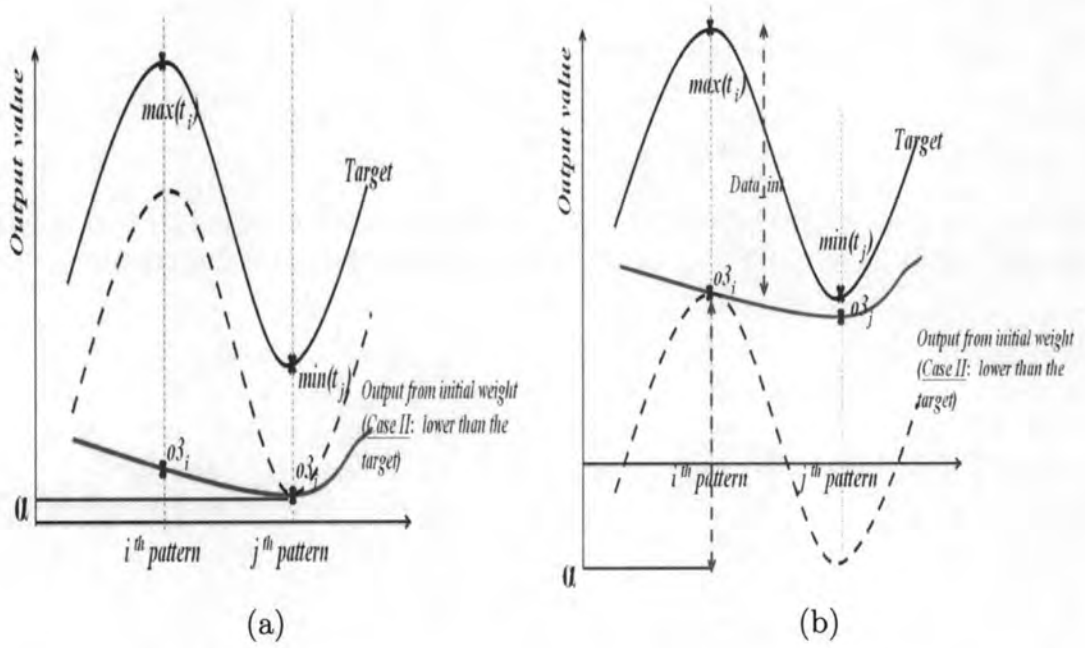



Figure 3.8: This figures show the idea of setting the $\alpha$ in (a) the fourth criterion. (b) The fifth criterion.

Figure 3.8 (a) shows the concept of setting the $\alpha$ value in case of initial weight give the output at $i^{th}$ pattern ($o3_i$) lower than target. The blue line shows the output that receive form the initial weight. Consider the $j^{th}$ position that give the minimum target value, we set the output value ($o3_j$) as the fourth criterion of the $\alpha$. Broken line shows that if we set the $\alpha$ equal to $o3_j$, we will have enough space for adjusting weight. Therefore, the fourth criterion ($set(4)$) equals to the absolute value of $o3_j$ in order to avoid the negative value.

$$set(4) = |o3_j| \tag{3.16}$$

In the same way as the third criterion, Figure 3.8 (b) shows the concept of setting the $\alpha$ in the fifth criterion. If we consider $o3_i$ that is the maximum target value, we have to add the area at least *data_int* to get enough the target range. Therefore, the fifth criterion equals to the absolute value of the difference between output at point $i$ and the target interval to increase the adjusting area.

$$set(5) = |o3_i - data\_int| \tag{3.17}$$

This approach can be applied in general case (dotted line in Figure 3.9). For training network, normally, we do not know whether initial weight will give

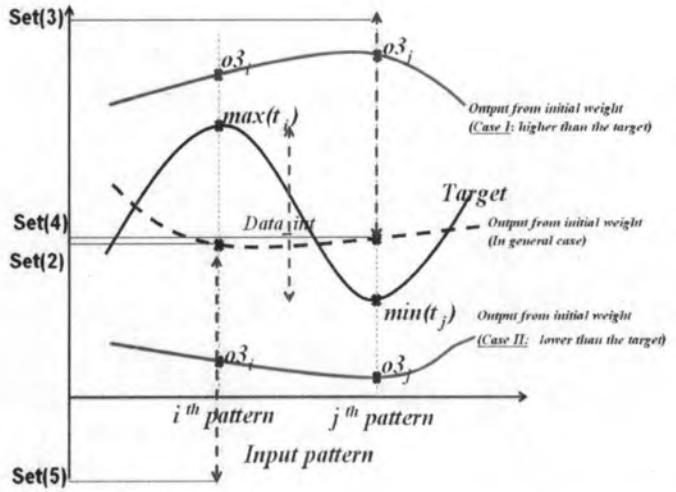



Figure 3.9: This figure shows the idea of setting the $\alpha$ in the general case.

the output value higher (green line) or lower (blue line) or the same area (dotted line) with the target. Therefore, we need to consider for all criteria that apply to set $\alpha$ value. After compute every *set* value, the highest value of all criteria is set to be the $\alpha$ value, the output interval, in the hyperbolic tangent function.

$$\alpha = max(\mathbf{set}) \tag{3.18}$$

## Learning Algorithm

1. Let $t = 1$.
2. Randomly select the initial weights.
3. Initialize the parameter $\mu$ ($\mu = 0.1$) and the decay rate $\beta$ ($\beta = 0.01$)
4. Compute the performance index $\mathbf{F(w)}$ from Equation 3.9
5. **While** $\mathbf{F_w} > \epsilon$ **and** $t \leq T$ **do**
6. Save the current value of the weight and bias
7. Compute the adjusting weight ($\Delta w$) from Equation 3.10
8. Recompute the performance index as an $\mathbf{F(w)^{new}}$
9. **If** $\mathbf{F(w)^{new}} < \mathbf{F(w)}$ **then**
10. Adjust the weight vectors from step 7 as an Equation 3.12
11. $\mu = \mu \cdot \beta$
12. **Else** $\mu = \mu/\beta$
13. restore the weight values
14. **End If**
15. $\mathbf{F(w)} = \mathbf{F(w)^{new}}$
16. $t = t + 1$
17. **Endwhile**

In this research, from Equation 3.10, we set $\mu$ equals to 0.1 [12] and the decay rate $\beta$ equals to 0.01 [12], where the $\epsilon$ refer to the acceptable error. In general, $\mu$ is multiplied by the decay rate $\beta$ whenever the $\mathbf{F}(\mathbf{w})^{\mathbf{new}}$ decreases, whereas $\mu$ is divided by the decay rate $\beta$ whenever the $\mathbf{F}(\mathbf{w})^{\mathbf{new}}$ increases in the next iteration.

## 3.5 Shifting the result

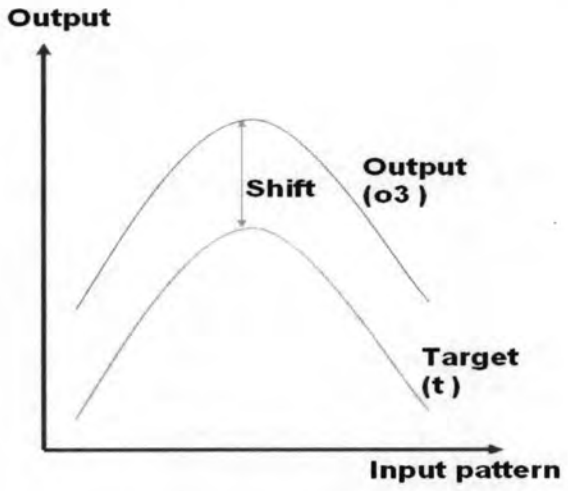



Figure 3.10: This figure shows the idea of shifting the result.

Figure 3.10 shows the concept of shifting the output value to get closer with the target. Red line shows the output vector and blue line shows the target vector. The goal is looking for the appropriate shift value ($shift$).

For every input pattern, the relationship function $R$ between the target ($t$) and the output ($o3$) is defined by equation 3.19 where $shift$ refers to the shift output value to tend to the target. After minimizing the relationship function $R$ by $shift$, we find the critical point of $Shift$ by using equation 3.21. After that, we apply equation 3.22 to be the $shift$ value.

$$R = \sum_{i=1}^{p} ((o3_i - shift) - t_i)^2 \tag{3.19}$$

$$\partial_{shift} R = \sum_{i=1}^{p} (2(shift - o3_i - t_i)) \tag{3.20}$$

$$\sum_{i=1}^{p} (2(shift - o3_i - t_i)) = 0 \tag{3.21}$$

$$Shift = \frac{1}{p} \left( \sum_{i=1}^{p} o3_i - \sum_{i=1}^{p} t_i \right) \tag{3.22}$$

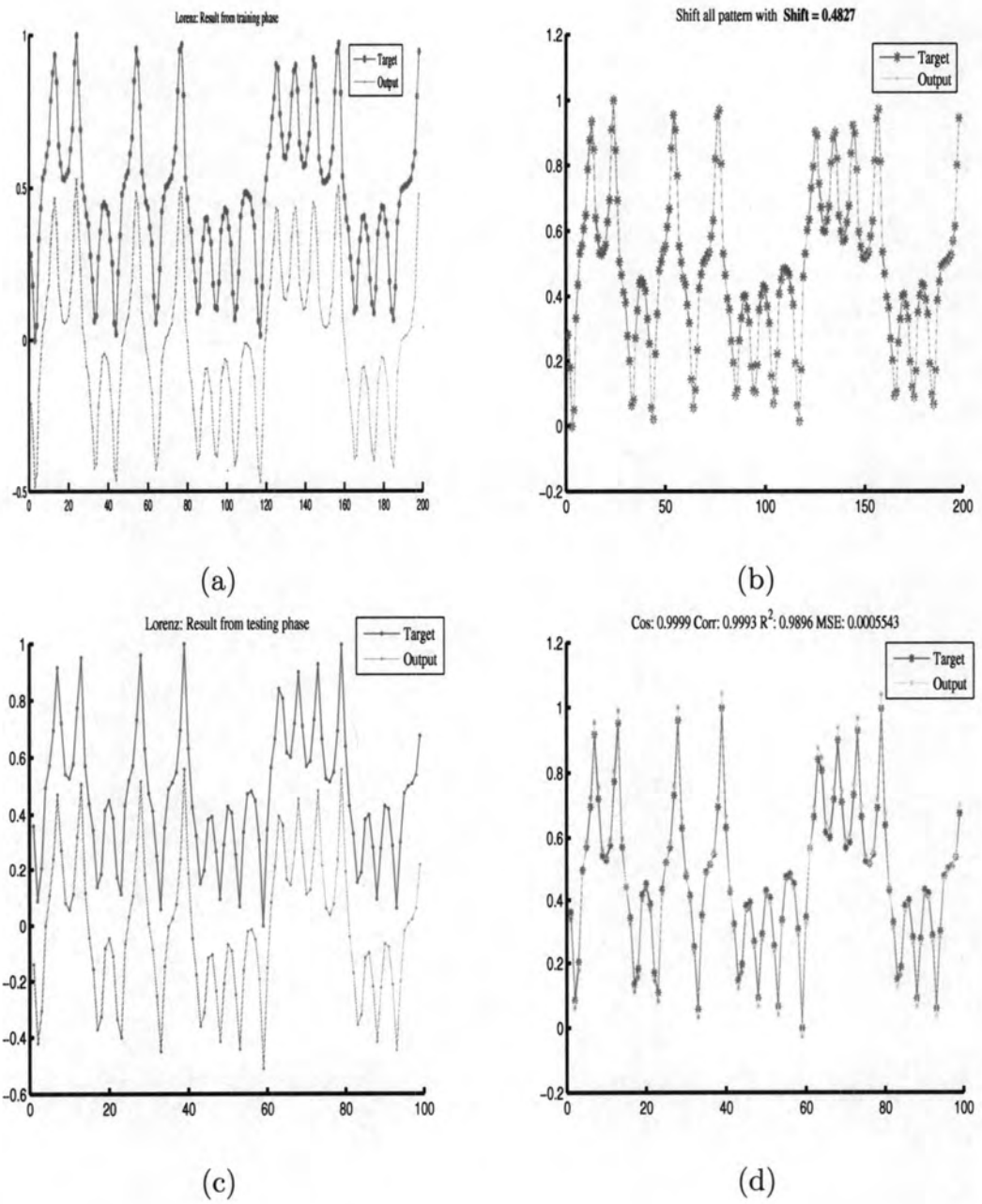



Figure 3.11: This figures show the idea of shifting the result. (a) Result from the training patterns (b) Result after shift the training patterns (c) Result from the testing patterns (d) Result after shift the testing patterns.

Figure 3.11 applies the shifting technique with the Lorenz testing set. Figure 3.11 (a) shows the result from the proposed cost function in the training phase. Figure 3.11 (b) shows the result from shifting output to get closer with the target in training phase with the $Shift$ value 0.4827. Figure 3.11 (c) shows the result from the proposed cost function in the testing phase. In Figure 3.11(d), we use the $Shift$ value shifting the output to approach the target value in the testing phase. The error measurements are measured the cosine value ($Cos$), co-relation ($Corr$), co-efficient of determination ($R^2$), and means squared error ($MSE$), respectively.

## 3.6 Using Neural Cellular Automata To Simulate The Color Diffusion

According to the principles of BP algorithm and neural cellular automata, this research applies and implement those principles and implement to forecasting of water diffusion characteristic which is one of the approximation problems. Color intensity of each pixel of diffusion image is used as input for the neural network, and the color intensity of the same pixel at the next time step is the target for neural network's learning. The advantages of using BPNN can be explained that it is easily understood and implemented as a program, and gives an acceptable result. However, the disadvantage of this method is the output may fluctuate within an acceptable error.

The predicted diffusion of surface disturbance is generated by pouring an amount of liquid on the surface of water. The liquid is colored and slightly heavier than water. The spontaneous spreading of this liquid generates a two-dimensional characteristic contour of color on the water surface. We can visually observe this phenomenon. The entire water surface can be captured on film at any instant of time. At each time step, the entire behavior is viewed as an image. The color intensity of each pixel corresponds to the intensity of the color of the poured liquid at the coordinates $(i, j)$ of that pixel. The intensity is assigned a value ranging from 0 to 255.

Let $I(t)_{i,j}$ be the intensity of pixel $(i, j)$ at time $t$. Based on the concept of Markovian process, the predicted value of $I(t)_{i,j}$ is computed from the values of the neighboring pixels and the current pixel at time $t - 1$. In this work, the neighboring pixels are confined in a square of size $3 \times 3$. Our assumption is that the event occurred at pixel $i$ is the sequel of the events of its neighboring pixels. In case of a square of size $3 \times 3$, the relation of $I(t)_{i,j}$ and its neighboring pixels is

$$
\begin{aligned}
I(t)_{i,j} = f(&I(t-1)_{i-1,j-1}, I(t-1)_{i-1,j}, \\
&I(t-1)_{i-1,j+1}, I(t-1)_{i,j-1}, I(t-1)_{i,j}, \\
&I(t-1)_{i,j+1}, I(t-1)_{i+1,j-1}, I(t-1)_{i+1,j}, \\
&I(t-1)_{i+1,j+1})
\end{aligned} \tag{3.23}
$$

Generally, this assumption can be extended to the situation where the event of the current pixel is affected by its square neighboring pixels of size $a \times a$. In our experiment, a square of size $5 \times 5$ is also considered. The value $I(t)_{i,j}$ is computed by

$$
\begin{aligned}
I(t)_{i,j} = f(&I(t-1)_{i-2,j-2}, I(t-1)_{i-2,j-1}, \\
&I(t-1)_{i-2,j}, I(t-1)_{i-2,j+1}, \\
&I(t-1)_{i-2,j+2}, I(t-1)_{i-1,j-2}, \\
&I(t-1)_{i-1,j-1}, I(t-1)_{i-1,j}, \\
&I(t-1)_{i-1,j+1}, I(t-1)_{i-1,j+2}, \\
&I(t-1)_{i,j-2}, I(t-1)_{i,j-1}, \\
&I(t-1)_{i,j}, I(t-1)_{i,j+1}, \\
&I(t-1)_{i,j+2}, I(t-1)_{i+1,j-2}, \\
&I(t-1)_{i+1,j-1}, I(t-1)_{i+1,j}, \\
&I(t-1)_{i+1,j+1}, I(t-1)_{i+1,j+2}, \\
&I(t-1)_{i+2,j-2}, I(t-1)_{i+2,j-1}, \\
&I(t-1)_{i+2,j}, I(t-1)_{i+2,j+1}, \\
&I(t-1)_{i+2,j+2})
\end{aligned} \tag{3.24}
$$

It is obvious that this prediction problem can be transformed to the problem of functional approximation problem for a supervised neural network. Hence, the intensity of pixel $(i, j)$ can be predicted by using a neural network. The entire image can be captured by a set of neural networks formed in a 2-D cellular fashion. Each cell corresponds to each pixel in the image. The value of cell $(i, j)$ is computed by a neural network.

Figure 3.12 shows our proposed 2D tightly coupled neural network. Every cell has the same neural structure and synaptic weights. The input vector of each network consists of $\{(I(t-1)_{i-1,j-1}, I(t-1)_{i-1,j}, I(t-1)_{i-1,j+1}, I(t-1)_{i,j-1}, I(t-1)_{i,j}, I(t-1)_{i,j+1}, I(t-1)_{i+1,j-1}, I(t-1)_{i+1,j}, I(t-1)_{i+1,j+1})\}$ and the corresponding target is $I(t)_{i,j}$. Base on this concept, only one neural network is needed for training. It is used to predict the intensity of every pixel.

For the network Architecture, color intensity of pixel $(i, j)$ and the neighboring pixel at time $t-1$ is considered as input vector from the network, where

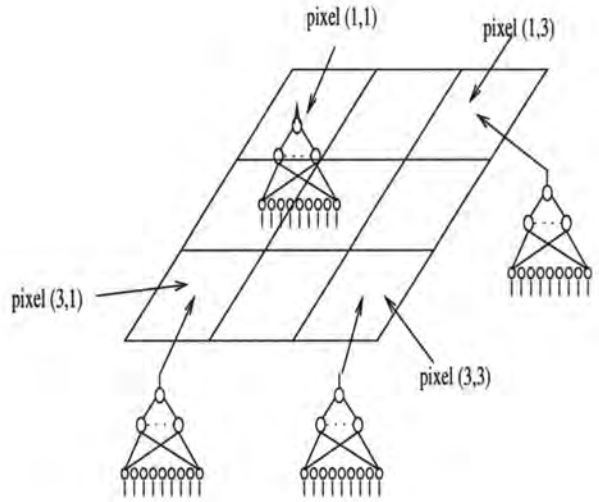



Figure 3.12: The structure of our proposed 2D tightly coupled cellular neural network.

as color intensity at time $t$ is an target vector. We do this process for all pixels at every time step of those. Afterwards, we prune all the redundant input patterns, and pass all the rest of input patterns to the network for training.

From Figure 3.1, network architecture composes of input, hidden and output layers. There are nine inputs in the input layer, one refers to the color intensity of pixel $(i, j)$ at time $t-1$, and the rest of inputs refers to its neighboring pixels. For output layer, there is only one neuron which refers to the color intensity of the pixel $(i, j)$ at time $t$.

## Simulation procedure

The learning algorithm initiates the data by performing hand-on experiments. The data from each experiment are grouped as the training set and the testing set. The procedures are as follows. To monitor the diffusion process of local surface disturbance by the pouring of colored liquid, we capture the planar spreading characteristic of colored liquid on water surface into video files. These files are, then, decomposed into image files with equal time. In order to avoid a vast number of inputs into the network, we crop each image at its center to achieve $100 \times 100$ pixels. Color intensity of each pixel is considered as an input. From the proposed method, the installation of blocking object into the water may cause the noisy input for the neural network due to the difference of color intensity between water surface and blocking object. To avoid this problem, the color intensity of it is adjusted to zero in order to ease the learning process of neural network. Finally, we train the proposed network by using 100 images, and then follow the procedures below.

**Step 1** Create the training data for every pixel, image and time step

**Step 2** Prune all the redundant input pattern.

**Step 3** Training neural network.
Train all the input patterns until the network converges or reaches the maximum iterations.

**Step 4** Test the trained neural network
Test the achieved network for creating the color intensity of image at the next time step.

**Step 5** Present all the forecasted images as an animation.

**Step 6** Error measurement