

**SPLINE BASED NEURAL NETWORKS**

**A Thesis**

**Presented to the Institute of Science and Engineering**

**of**

**Işık University**

**In Partial Fulfillment of the Requirements for the Degree of**

**Master of Science**

**In**

**The Department of Computer Engineering**

**by**

**Hikmet Dalkılıç**

**June 2005**

Approval of the Institute of Science and Engineering

---

Prof .Dr. Hüsnu Erbay  
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

---

Prof Dr. Ahmet Aksen  
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

---

Prof. Dr. Selahattin Kuru  
Supervisor

Examining Committee Members

---

.....

\_\_\_\_\_

.....

\_\_\_\_\_

.....

\_\_\_\_\_

.....

\_\_\_\_\_

.....

\_\_\_\_\_

## **ABSTRACT**

### **SPLINE BASED NEURAL NETWORKS**

Hikmet Dalkılıç

In this thesis, we applied the Catmull-Rom splines and B-splines to the neural networks models, which are Multi Layer Perceptrons, Elman Networks, and Locally Recurrent Neural Networks, as adaptive activation functions. We derived the learning algorithms for the five new neural network models, which we proposed. This new models are called “Multi Layer Perceptrons with Adaptive B- Spline Activation Function”, “Elman Networks with Adaptive Catmull-Rom Spline Activation Function”, “Elman Networks with Adaptive B- Spline Activation Function”, “Locally Recurrent Neural Networks with Adaptive Catmull-Rom Spline Activation Function”, “Locally Recurrent Neural Networks with Adaptive B- Spline Activation Function”. We measure the performance of these networks on the xor problem and compare the performance of them for this problem. To simulate the networks and to compare their performances we developed a web-based neural network simulator written in PHP 4 called SBNN.

Keywords: Spline networks, spline activation functions, adaptive activation functions, Adaptive Catmull-Rom spline activation functions, Adoptive B- spline activation functions, SBNN.

## ÖZET

### SPLINE TABANLI YAPAY SİNİR AĞLARI

Hikmet Dalkılıç

Bu tez ile, Catmull-Rom spline fonksiyonları ve B-spline fonksiyonları uyarlanabilir aktivasyon fonksiyonları olarak, yapay sinir ağı modelleri olan Çok Katmanlı Ağlara, Elman ağlarına ve Yerel Geri Beslemeli ağlara uygulandı.

Bu uygulamalardan oluşturduğumuz 5 yeni yapay sinir ağı modeli için öğrenme algoritmalarının çıkarımları yapıldı. Bu yeni modeller sırasıyla “Uyarlanabilir Catmull-Rom Spline Aktivasyon Fonksiyonlu Çok Katmanlı Ağlar”, “Uyarlanabilir B-Spline Aktivasyon Fonksiyonlu Çok Katmanlı Ağlar”, “Uyarlanabilir Catmull-Rom Spline Aktivasyon Fonksiyonlu Elman Ağları”, “Uyarlanabilir B-Spline Aktivasyon Fonksiyonlu Elman Ağları”, “Uyarlanabilir Catmull-Rom Spline Aktivasyon Fonksiyonlu Yerel Geri Beslemeli ağlar”, ve son olarak “Uyarlanabilir B-Spline Aktivasyon Fonksiyonlu Yerel Geri Beslemeli ağlar” diye adlandırılır. Ağların performansı xor problemi kullanılarak ölçüldü ve sonuçları birbirleriyle karşılaştırıldı. Yapay sinir ağlarını oluşturulması ve performanslarının ölçülmesi için SBNN adında PHP 4 programlama dilini ile yazılmış web tabanlı bir yapay sinir ağı similatörü geliştirildi.

Anahtar Kelimeler: Spline ağları, spline aktivasyon fonksiyonları, Uyarlanabilir aktivasyon fonksiyonları, Uyarlanabilir Catmull-Rom spline aktivasyon fonksiyonları, Uyarlanabilir B-spline aktivasyon fonksiyonları, SBNN.

To My parents, Mehmet Ali & Fatma Dalkılıç

## ACKNOWLEDGEMENTS

To people who contributed in different ways to this thesis, for which I would like to express thanks.

Prof. Dr Selahattin Kuru for his guidance throughout my work and for keeping me motivated with his great advises when I loose my belief to finish this thesis before the deadline.

Dilek , my fiancée, for her understanding of my behaviors under the stress of the works, especially on the final days of the thesis and her support by making me feel loved every time when I need.

## TABLE OF CONTENT

	<u>PAGE</u>
ABSTRACT .....	iii
ÖZET .....	iv
ACKNOWLEDGEMENTS.....	vi
TABLE OF CONTENTS.....	vii
LIST OF FIGURES.....	x
1. INTRODUCTION.....	1
2. SPLINE FUNCTIONS.....	3
2.1 Spline Specification.....	3
2.2 Spline Function’s Mathematical Description .....	3
3. MULTI LAYER PERCEPTRON.....	7
3.1. Multi Layer Perceptron with Sigmoid Activation Functions.....	7
3.1.1 The Structure of Multi Layer Perceptron.....	7
3.1.2 Delta Learning Rule.....	10
3.2. Multi Layer Perceptron with Adaptive Catmull-Rom Spline Activation Functions.....	12
3.2.1 Gradient-Based Learning for Multi layer Perceptron with Adaptive Spline Activation Function.....	16
3.3. Multi Layer Perceptron with Adaptive Catmull-Rom Spline Activation Functions.....	20
4. ELMAN NEURAL NETWORKS.....	29
4.1. Elman Networks with Sigmoid Activation Function .....	29
4.2. Elman Networks With Adaptive Catmull-Rom Spline Activation Functions.....	31
4.3 Elman Networks With Adaptive B-Spline Activation Functions.....	36
5. LOCALLY RECURRENT NEURAL NETWORKS.....	41
5.1 Locally Recurrent Neural Networks with Sigmoid Activation Functions.....	41
5.2 Locally Recurrent Neural Networks with Adaptive Catmull-Rom Spline Activation Functions.....	42

5.3 Locally Recurrent Neural Networks with Adaptive B-Spline Activation Functions.....	47
6. PERFORMANS OFTHE NETWORKS.....	52
6.1 Comparison of the Multi Layer Perceptron.....	52
6.1.1. Comparison of the sigmoid activation functions and the B-spline activation function for MLP.....	52
6.1.2. Comparison of the Catmull-Rom spline activation functions and the B-spline activation functions for MLP.....	53
6.2 Comparison of the Elman Networks.....	54
6.2.1. Comparison of the sigmoid activation functions and the B-spline activation function for Elman Networks.....	54
6.2.2. Comparison of the Catmull-Rom spline activation functions and the B-spline activation functions for Elman Networks .....	54
6.2.3. Comparison of the Catmull-Rom spline activation functions and the sigmoid activation functions for Elman Networks .....	57
6.3 Comparison of the Locally Recurrent Neural Networks .....	58
6.3.1. Comparison of the sigmoid activation functions and the B-spline activation function for Locally Recurrent Neural Networks.....	59
6.3.2. Comparison of the Catmull-Rom spline activation functions and the sigmoid activation functions for Locally Recurrent Neural Networks.....	59
6.4 Comparison of execution time of all model.....	60
7. SPLINE BASED NEURAL NETWORK SIMULATOR (SBNN).....	62
7.1 Software Specification of SBNN.....	62
7.2 The aim of this Software.....	62
7.3 The Menu of the SBNN.....	63
7.4 Training a network.....	64
7.5 Running and deleting a network.....	66
7.6 Comparing the Network Performance.....	66
7.7 Comparing the Execution time performance .....	68
9. CONCLUSION AND RECOMMENDATIONS FOR FUTURE WOR.....	69
REFERRANCES.....	70
APPENDIX: CD containing Thesis text and software code .....	72



## LIST OF FIGURES

2.1. Parametric point function $p(u)$ for a curve section between control points $p_k$ ve $p_{k+1}$ .....	4
3.1. A neuron structure.....	8
3.2. An example of Multi Layer Perceptron.....	8
3.3. The graph of Cost Function.....	11
3.4. Catmull-Rom Spline curve span.....	14
3.5. Control points of the Catmull–Rom spline-based activation function.....	16
3.6. A neuron with adaptive spline activation function.....	18
3.7. B-spline curve behaviors at control points.....	20
4.1. An example of Elman Networks.....	29
5.1. Locally Recurrent Neural Networks.....	41
6.1. Performances of MLPs with sigmoid and adaptive b-spline activation functions for xor problem.....	53
6.2. Performances of MLPs with adaptive CR spline and adaptive b-spline activation functions for xor problem.....	54
6.3. Performances of MLPs with adaptive CR spline and adaptive b-spline activation functions for xor problem in more detail .....	54
6.4. Performances of Elman Networks with sigmoid and adaptive b-spline activation function for xor problem.....	55
6.5. Performances of Elman Networks with adaptive CR spline and adaptive b- spline activation functions for xor problem.....	56
6.6. Performances of Elman Networks with adaptive CR spline and adaptive b- spline activation functions for xor problem in more detail.....	56
6.7. Performances of Elman Networks with sigmoid and adaptive CR spline activation functions for xor problem.....	57
6.8. Performances of Elman Networks with sigmoid and adaptive CR spline activation functions for xor problem in more detail.....	58
6.9. Performances of LRNNs with sigmoid and adaptive b-spline activation functions for xor problem .....	59
6.10. Performances of LRNNs with sigmoid and adaptive CR spline activation functions for xor problem .....	60

6.11	Execution time graph for all the models.....	61
7.1	The main page of SBNN.....	63
7.2	The submenu of “NETWORKS TYPES”.....	63
7.3	The submenu of “COMPARE”.....	64
7.4	The initial parameter of a network to start training .....	65
7.5.	The table which shows the number of epoch to go on training.....	65
7.6.	The table of outputs for MLP.....	66
7.7	Comparison table of network models and The parameters table of the graphs, shows the graph of cost functions for the number of epoch.....	67
7.8.	An example of comparison graphics.....	67
7.9	Chosen parameters for the graph, shown in Figure 7.8.....	68
7.10	An example of Execution Time Comparison and report table.....	68

## CHAPTER 1. INTRODUCTION

This thesis aimed to use spline functions for the artificial neural networks as adaptive activation function [1, 2, and 3]. We used this kind of activation functions for the networks: “Multi Layer Perceptrons” [4], “Elman Networks” [5], and “Locally Recurrent Neural Networks” [6].

In chapter 2, we give a brief definition of the splines function and spline fitting[7,8] which is an extremely popular form of piecewise approximation using various forms of polynomials of degree  $n$ , or more general functions, in which they are fitted to the function at specified points, known as control points, nodes or knots.

In chapter 3, First of al, we explained the model “Multi Layer Perceptrons” which is using sigmoid activation function [9]. This is one of the most popular neural network types, which are commonly used for engineering problems. Multi Layer Perceptrons use the “generalized delta learning rule” [10, 11] as learning algorithm.

In the section 3.2, we described the model “Multi Layer Perceptrons with Adaptive Catmull-Rom Spline Activation Function” [12, 13, and 14]. This model is very similar to the MLP model the only the difference is the activation function. This model uses a spline based activation function called “Catmull-Rom Spline [15] Activation Function”. In this model not only the weights but also the control points of the activation functions are adapted at the learning phase.

In the section 3.3, we used B-spline as activation function instead of Catmull-Rom spline. For the model “Multi Layer Perceptron with Adaptive B-Spline Activation Function”

We defined and derived the learning algorithms of this network in a similar way, used in the model “Multi Layer Perceptron with Adaptive Catmull-Rom Spline Activation Function”.

In first section of chapter 4, we explained “Elman Networks” which is a member of “Recurrent Neural Networks” [16], and its learning rule.

In the section 4.2 we propose a new network type “Elman Networks with Adaptive Catmull-Rom Spline Activation Function” In this network we apply “Adaptive Catmull-Rom Spline Activation Function” to the “Elman Networks” and derive a learning algorithm for this network types.

In the section 4.3 we propose a new network type “Elman Networks with Adaptive B- Spline Activation Function” and derive a learning algorithm for this model.

We described the network type “Locally Recurrent Neural Networks” and it’s learning algorithm in the first section of Chapter 5.

In the section 5.2 we propose a new network type “Locally Recurrent Neural Networks with Adaptive Catmull-Rom Spline Activation Function” and derived a learning algorithm for this model.

In the section 5.3 we also proposed a new network type “Locally Recurrent Neural Networks with Adaptive B-Spline Activation Function” and derived a learning algorithm for this model.

In chapter 6, we compared the performances of this nine network models with the famous xor problem by using the software SBNN, developed to simulate these nine different models.

In chapter 7, we introduced the software SBNN which is developed to simulate neural networks, we used. SBNN is a web based program written in the programming language PHP 4 [17]. We used this tool to compare the performance of the network.

## CHAPTER 2. SPLINE FUNCTIONS

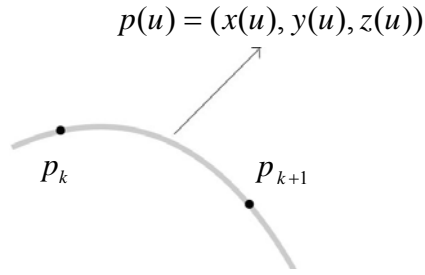
### 2.1 Spline Specification

Spline functions are piecewise polynomial functions. Commonly used in computer graphics. These functions are fascinating properties. By using spline functions we can easily interpolate or approximate all the control points with a smooth, continuous curve which has first and second derivatives at every point. Since the splines use low degree polynomials like cubic polynomials the cost of calculation are so small comparing with the high degree polynomials. In their most general form, splines can be considered as a mathematical model that associate a continuous representation of a curve or surface with a discrete set of points in a given space. We control the shapes of spline curves by changing the control points which may also called nodes or knots. One control points effects only the four consecutive segments of the function. Hence, we can make local changes by changing one or more control points. In polynomials interpolation we can't make the local changes. When you make a change on any parameter of polynomial interpolation, it affects all the shape of the curve.

### 2.2 Spline Function's Mathematical Description

The control points are described as (1). While the parameter  $u$  scan 0 to 1,  $p(u)$  scans the between two intermediate control points of four consequence points used for spline functions. Figure 2.1 shows these control points.

$$p_k = (x_k, y_k, z_k) \quad k = 0, 1, 2, \dots, n \quad (1)$$



**Figure 2.1.** Parametric point function  $p(u)$  for a curve section between control points  $p_k$  ve  $p_{k+1}$

The expression at (2) is called as coefficient matrix of Catmull-Rom Splines, and the matrix at (3) called as Catmull-Rom splines characteristic matrix. The functions as in (4) called Catmull-Rom blending functions.

$$\begin{bmatrix} a_x \\ b_x \\ c_x \\ d_x \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ x_k \\ x_{k+1} \\ x_{k+2} \end{bmatrix} \quad (2)$$

$$M_c = \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \quad (3)$$

$$\begin{aligned}
C_0(u) &= \frac{1}{2}(-u^3 + 2u^2 - u) \\
C_1(u) &= \frac{1}{2}(3u^3 - 5u^2 + 2) \\
C_2(u) &= \frac{1}{2}(-3u^3 + 4u^2 + u) \\
C_3(u) &= \frac{1}{2}(u^3 - u^2)
\end{aligned} \tag{4}$$

Catmull-Rom Splines interpolate the end points which are second and third points of four consecutive points. They span the region between the end points and create a smooth curve while  $u$  spans the distance 0 to 1. We can manipulate a specific segment of the curve by changing the four consecutive points and it gives the power of flexibility and locality to our curve.

If there is no need to the interpolation at end points we can use another technique which doesn't require interpolation at end points. In this technique Curves approximate the end points and but the segment at the right hand side and at the left hand side must have equal derivative at the meeting points (control points). These kinds of splines are called B-splines (5). The matrix (6) used in this description is called b-spline characteristic matrix. The functions in (7) called blending functions of b-spline matrix.

$$\begin{bmatrix} a_x \\ b_x \\ c_x \\ d_x \end{bmatrix} = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ x_k \\ x_{k+1} \\ x_{k+2} \end{bmatrix} \tag{5}$$

$$M_b = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \tag{6}$$

$$\begin{aligned}C_0(u) &= \frac{1}{6}(-u^3 + 3u^2 - 3u + 1) \\C_1(u) &= \frac{1}{6}(3u^3 - 6u^2 + 4) \\C_2(u) &= \frac{1}{6}(-3u^3 + 3u^2 + 3u + 1) \\C_3(u) &= \frac{1}{6}u^3\end{aligned}\tag{7}$$



## **CHAPTER 3. MULTI LAYER PERCEPTRONS**

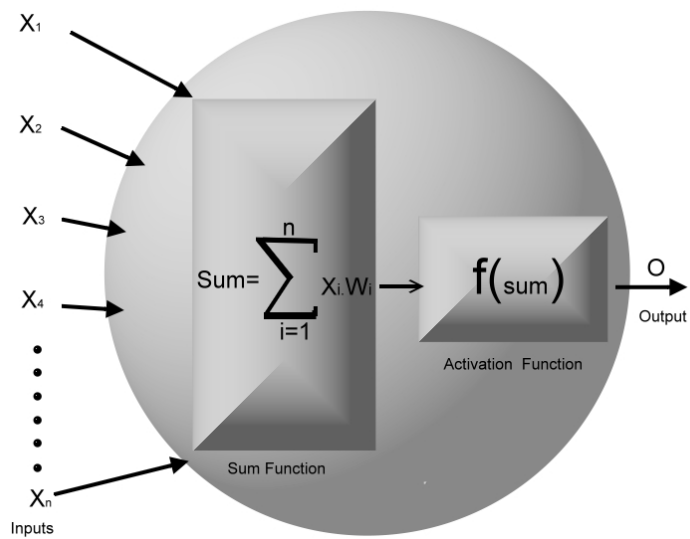
### **3.1 Multi Layer Perceptrons with Sigmoid Activation Functions**

Multi Layer Perceptron is one of the most popular and most frequently used artificial neural network model. When Minsky [18] showed that an older model Perceptron [19], proposed before the Multi Layer Perceptrons, couldn't solve the famous xor problem[20], researcher's motivation on artificial neural networks are dramatically decreased and almost all research on this area was ended. Researcher thought that the problem, which has no linear relations between, inputs and outputs like xor could not be solved by artificial neural networks .Only a few researchers continued to work on this area. One of the researchers who insisted on working on this area was Rumellhart and his friends solved xor problem by using the more than one layer. Rumellhart and his friends' solution had a great impact on starting the researches on this area. Currently, Multi Layer Perceptrons are producing solutions almost all engineering problems and it is the most commonly used artificial neural networks model in industry. Especially to solve the classification, generalization and Identification problems, Multi Layer Perceptrons are frequently used with the learning rule called "Delta Learning Rule".

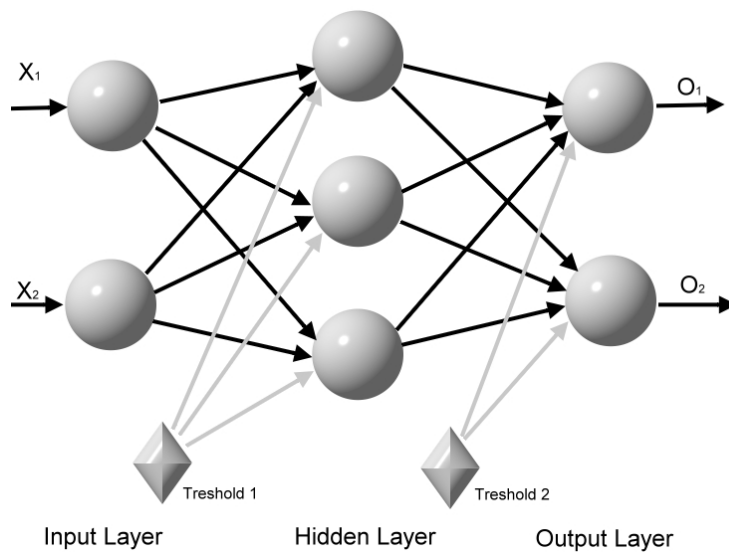
#### **3.1.1 The Structure of Multi Layer Perceptrons**

Like all other artificial neural network, Multi Layer Perceptrons consists of neurons. A neuron structure showed in Figure 3.1 by details.

A neuron has inputs, output, sum function and activation function. The inputs are multiplied by corresponding weights, which are the random number chosen the range between 0 and 1. The sum of these products is put in an activation function and the output of this activation function is the output of this neuron.



**Figure 3.1** A neuron structure



**Figure 3.2** An example of Multi Layer Perceptron

In Multi Layer Perceptron as shown in Figure 3.2, there are three main layers, which are Input Layer, Hidden Layer, and Output Layer. Now we will explain this tree layer respectively.

### **Input layer:**

This layer's duty is to transform the data coming from the outside world into the hidden layer. The inputs are not multiplied by any weight and also not passed throughout any activation functions. Every input layer neuron only one input and only one output which is exactly same to this input. Input layer has the number of neuron, equals to the number of inputs. Every output of input layers' neuron is sent to the first hidden layer as input.

### **Hidden layer:**

Hidden layer gets the data from inputs layer, puts them in some processes and sends the output of this processes to the output layer. Every output of input neuron is multiplied by a corresponding weight, which is randomly chosen from the range between 0, and 1. The sum of these weighted inputs is passed through the corresponding neurons' activation functions. The output of this activation function is the output of the corresponding neuron and sent to the output layer as input.

### **Output layer:**

Output layers' neurons are very similar to hidden layers' neurons. Every neuron in this layer gets the every output of last hidden layer as input. These inputs are multiplied by corresponding weights as in hidden layers. The sum of these weighted inputs is sent to activation function and the output of activation function of every neuron in output layer is built the output of output layer also built the output of network. The number of neuron in this layer is equal to the number of outputs of the network.

Multi Layer Perceptron uses supervised learning method. In this method, a specific amount of inputs and expected outputs are shown to the network. Network makes some generalization from this example and generates a solution set for the problem then uses this solution set for new sample to get an output.

We mentioned about “delta learning rule” which Multi Layer Perceptron commonly uses as learning rule. Now we will examine “delta learning rule” in details.

### 3.1.2 Delta Learning Rule

The other name of this rule is known as “least mean squares rule”. It is developed for supervised learning models. The basics of this learning rule are related to the philosophy of least mean squares rule. By the least mean squares rule we get the local minimums. In the philosophy of this method, you can find a local minimum by going to the opposite side of gradient. It may or may not be the global minimum but it is certainly at least a local minimum.

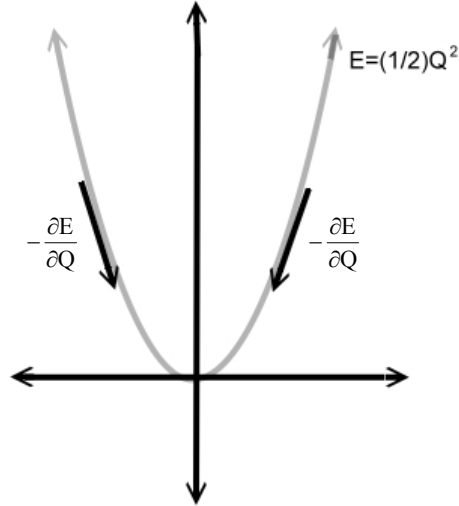
Delta Learning Rule use the cost function (1) which is half of the sum of the square root of difference between network outputs and desired outputs to find weight vectors.

$$E = \sum_k \frac{1}{2} (o_k - d_k)^2 \quad (1)$$

If we define the difference between network outputs and desired outputs as  $Q$ , then we get the equation (2).

$$E = \sum \frac{1}{2} Q^2 \quad (2)$$

The graph of this cost function is shown in Figure 3.3. As we can easily see in this Figure opposite direction (i.e. negative direction) of derivative take us to the minimum of this graph. In this manner we can get the following conclusion, to find the minimum point of E-W graph we should move on the direction of  $-\frac{\partial E}{\partial W}$ .



**Figure 3.3** The graph of Cost Function

MLP uses the equation (3) to adapt the weights. To find the derivative on the right hand side we use (4). To find  $\delta$ , used in (4), we use (5).

$$w_{kj}^l[t+1] = w_{kj}^l[t] - \mu_w \frac{\partial E_p[t]}{\partial w_{kj}^l[t]} \quad (3)$$

$$\frac{\partial E}{\partial w_{kj}^{l-r}} = \begin{cases} \delta_k^{l-r} \cdot f'(sum_k^{l-r}) \cdot o_j^{l-r-1} & \text{where } r \neq 0 \\ \delta_k^{l-r} \cdot o_j^{l-r-1} & \text{where } r = 0 \end{cases} \quad (4)$$

$$\delta_k^{l-r} = \begin{cases} \sum_k \delta_k^{l-r+1} \cdot w_{kj}^{l-r+1} & \text{where } r \neq 0 \\ (o_k^l - d_k) f'(sum_k^{l-r}) & \text{where } r = 0 \end{cases} \quad (5)$$

### 3.2. Multi Layer Perceptrons with Adaptive Catmull-Rom Spline Activation Functions

In this section, we will explain the adaptive cubic Catmull-Rom Spline Activation Functions; used in the Multi Layer Perceptrons, will be introduced. There are many reasons to use adaptive cubic Splines as activation function. Some of them are mentioned below.

- On Spline curves you can easily change a small segment of the curve by changing a few control points. Changing a control point affects only the four consecutive segments so the changes on the curve are totally local. On a polynomial curve you can not do this kind of local adaptations. When you change a parameter on a polynomial curve the whole curve is affected.
- The degree of the polynomial must be so high to interpolate all control points comparing with a spline function. Hence, the cost is significantly reduced by using a cubic spline instead of using a high degree polynomial.

Spline functions also have the following properties, which must be had by any activation function.

- boundedness constraint
- universal approximation property
- flexibility

A cubic Spline Activation Function is described as (1)

$$F(u) = [F_x(u) \ F_y(u)]^T = \underset{i=0}{\overset{N-3}{C}} F_i(u) \quad (1)$$

$C$  in the equation (1) is the concatenation operator.  $F_i(u)$  is  $i$ th curve span.  $U$  is the local parameter and takes the value in the range between 0 and 1. The indices of  $C$  are valid only for cubic polynomials.  $i$ th curve span function is defined as  $F_i(u)$  (2). In this equation,  $C_j(u)$  represents spline polynomials in another words spline blending functions.  $Q$  represents the control points.

$$F_i(u) = [F_{xi}(u) \ F_{yi}(u)]^T = \sum_{j=0}^3 Q_{i+j} C_j(u) \quad (2)$$

Every control points consist of two component x and y components.

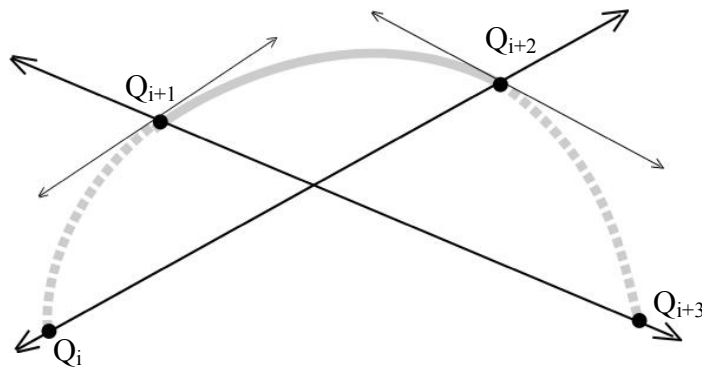
$$Q = \{Q_0, \dots, Q_n\} \quad Q_i = [q_{x,i} \ q_{y,i}]^T$$

When the Catmull-Rom splines are used as activation function  $C_j(u)$  can be written as (3).

$$\begin{aligned} C_0(u) &= \frac{1}{2}(-u^3 + 2u^2 - u) \\ C_1(u) &= \frac{1}{2}(3u^3 - 5u^2 + 2) \\ C_2(u) &= \frac{1}{2}(-3u^3 + 4u^2 + u) \\ C_3(u) &= \frac{1}{2}(u^3 - u^2) \end{aligned} \quad (3)$$

Considering the equations (3), the equation (2) can be written as the equation (4).  
 The derivative of this equation can be computed easily.

$$F_i(u) = [u^3 \ u^2 \ u \ 1] \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \quad (4)$$



**Figure 3.4** Catmull-Rom Spline curve span



The derivatives at the end points are taken as below, shown in Figure 3.4.

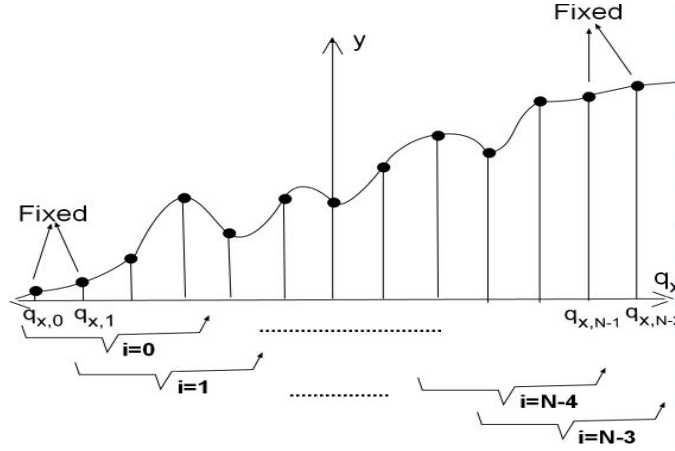
$$\text{While } u=0 \quad \frac{\partial F_i(0)}{\partial u} = \frac{1}{2}[-Q_i + Q_{i+2}].$$

$$\text{While } u=1 \quad \frac{\partial F_i(1)}{\partial u} = \frac{1}{2}[-Q_{i+1} + Q_{i+3}]$$

To reduce the cost of the calculation the control points abscissas are chosen equally spaced on the x-axis and centered at the origin. The abscissas of the control points are not adapted also. Hence,  $F_{xi}(u)$  becomes a linear function instead of a cubic polynomial function.  $\Delta x$  shows the equal distances between the abscissas of the consecutive control points.

Since we choose the abscissa of control point's equally distanced and centered at the origin there is no need to store the abscissas of the control points. All we need to store number of control points and the distance between the control points.

The initial values for control points are derived from the most popular activation function called sigmoid activation function. Since the activation function must be a limiting function, last two points and first two points are fixed to satisfy this property shown in Figure3.5.



**Figure 3.5** Control points of the Catmull–Rom spline-based activation function with a fixed step  $\Delta x$ . The extreme points  $q_{x;0}$ ,  $q_{x,1}$ , and  $q_{x,N-1}$ ,  $q_{x;N-2}$  are fixed

### 3.2.1 Gradient-Based Learning for Multi Layer Perceptrons with Adaptive Spline Activation Function

#### Definition of the parameters:

- $O_k^l$  Output of the  $k$ th neuron in the  $l$ th layer;
- $w_{kj}^l$  Weight of the  $k$ th neuron in the  $l$ th layer with respect to the  $j$ th neuron in the previous layer. ( $w_{k0}^l$  are the bias terms);
- $Sum_k^l$  Net output (i.e., linear combiner output) of the  $k$ th neuron in the  $l$ th layer;
- $N + 1$  Number of control points for each neuron in the network;
- $\Delta x$  Sampling step along the x-axis for each activation function;
- $i_k^l$  Curve span index of the activation function for the  $k$ th neuron in the  $l$ th layer ( $0 \leq i_k^l \leq N-2$ );
- $u_k^l$  Local parameter for the  $i_k^l$ th curve span of the  $k$ th neuron in the  $l$ th layer ( $0 \leq u_k^l \leq 1$ );

- $q_{k,n}^l$  Ordinate of the  $n$ th control point of the  $k$ th neuron in the  $l$ th layer ( $0 \leq n \leq N$ ). The control point abscissas  $q_{x,n}$  do not appear since we assume the  $x$ -axes are uniformly sampled;
- $F_{k,i_k}^l(\cdot)$   $i_k^l$  th Spline patch of the activation function for the  $k$ th neuron in the  $l$ th layer;
- $C_{k,m}^l(\cdot)$   $m$ th CR polynomial(blending function) for the  $k$ th neuron in the  $l$ th layer ( $0 \leq m \leq 3$ ).

### Forward Computation:

The parameters  $u$  and  $i$  gathered from the equations (5) and sent as inputs to the equation (6).

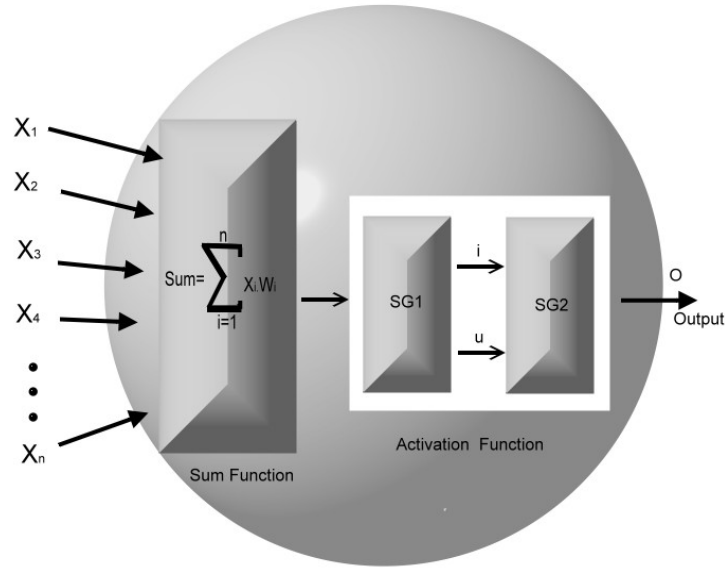
$$z_k^l = \frac{Sum}{\Delta x} + \frac{N-2}{2}$$

$$i_k^l = \lfloor z_k^l \rfloor$$

$$u_k^l = z_k^l - \lfloor z_k^l \rfloor$$
(5)

Output of this equation (6) is also the output of the neuron.

$$O_k^l = F_{k,i_k}^l(u_k^l) = \sum_{m=0}^3 q_{k,(i_k+m)}^l C_{k,m}^l(u_k^l)$$
(6)



**Figure 3.6** A neuron with adaptive spline activation function

The block, which is defined at (5), called GS1 and the block, defined in (6) called GS2 shown in Figure 3.6.

### Backward Computation (Learning Phase):

$p$  shows the learning sample,  $t$  shows the iteration indices,  $o$  shows the output of the neuron,  $d$  shows the desired output,  $E$  shows the cost function.  $\mu_w$  and  $\mu_q$  are learning coefficient for weights and the control points, respectively.

For every neuron an  $e$  parameter is defined as in (7)

$$e_k^l[t] = \begin{cases} (o_k^l[t] - d_k[t]) & l = M \\ \sum_{p=1}^{N_{l+1}} \delta_p^{l+1}[t] w_{pk}^{l+1} & l = M-1, \dots, 1 \end{cases} \quad (7)$$

$\delta$  parameter is also defined as in(8) for every neuron.

$$\delta_k^l[t] = e_k^l[t] \left( \frac{\partial F_{k,i_k}^l(s_k^l[t])}{\partial \text{Sum}_k^l[t]} \right) \quad (8)$$

Since we fixed last and first two points, the fixed values are used for the derivative of these points as in (9).

$$\frac{\partial F_{k,i_k}^l(s_k^l[t])}{\partial \text{sum}_k^l[t]} = \frac{q_{k,1}^l - q_{k,0}^l}{\Delta x} \quad s_k^l < q_{k,1}^l \quad (9)$$

$$\frac{\partial F_{k,i_k}^l(s_k^l[t])}{\partial \text{sum}_k^l[t]} = \frac{q_{k,N}^l - q_{k,N-1}^l}{\Delta x} \quad s_k^l > q_{k,N-1}^l$$

If we get  $w_{k0}^l$  weights as offset values, the equations (10) and (11) are used to adapt weights and control points respectively.

$$w_{kj}^l[t+1] = w_{kj}^l[t] + \mu_w \delta_k^l[t] \begin{cases} o_j^{l-1} & j \neq 0 \\ 1 & j = 0 \end{cases} \quad (10)$$

$$q_{k,(i_k+m)}^l[t+1] = \partial q_{k,(i_k+m)}^l[t] + \mu_q e_k^l[t] C_{k,m}^l(u_k^l[t]) \quad (11)$$

$$m = 0, \dots, 3$$

### 3.3. Multi Layer Perceptron with Adaptive B-Spline Activation Functions

In this section we will use the adaptive b-spline activation functions for Multi layer Perceptron by the same technique used in section 3.2. Actually we only change the spline types. Hence, the computations are very similar to 3.2. The main difference between Catmull-Rom splines and B-Splines is their positions at the control points. While Catmull-Rom spline is interpolating control points, B-spline approximates the control points as in Figure 3.7 .



**Figure 3.7** B-spline curve behaviors at control points

B-spline characteristic matrix and blending functions described (1) and (2) respectively as we mentioned in chapter 2.

$$M_b = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \quad (1)$$

$$\begin{aligned}
C_0(u) &= \frac{1}{6}(1-u)^3 \\
C_1(u) &= \frac{1}{6}(4-6u^2+3u^3) \\
C_2(u) &= \frac{1}{6}(1+3u+3u^2-3u^3) \\
C_3(u) &= \frac{1}{6}u^3
\end{aligned} \tag{2}$$

Considering the equation (1), we can define the equation (3) for the ordinate of the control points as in section 3.2.

$$F_i(u) = [u^3 \ u^2 \ u \ 1] \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \tag{3}$$

The derivative of the equation (3) for  $u$  is written as the equation (4).

$$\frac{\partial F_i(u)}{\partial u} = [3u^2 \ 2u \ 1] \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \tag{4}$$

The derivatives at the end points are computed as below as computed in section 3.2.

$$\text{While } u=0 \quad \frac{\partial F_i(0)}{\partial u} = \frac{1}{2}[-Q_i + Q_{i+2}].$$

$$\text{While } u=1 \quad \frac{\partial F_i(1)}{\partial u} = \frac{1}{2}[-Q_{i+1} + Q_{i+3}]$$

If we choose the control points in the same way we used in section 3.2. We constrain the control points' abscissas to be equidistant and not adaptable and also centered at the origin.

$$F_{xi}(u) = [u^3 \ u^2 \ u \ 1] \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} Q_{x,i} \\ Q_{x,i} + \Delta x \\ Q_{x,i} + 2\Delta x \\ Q_{x,i} + 3\Delta x \end{bmatrix} \quad (5)$$

When we solve the equation (5) we get the equation (6).

$$F_{xi}(u) = u\Delta x + q_{x,i} + \Delta x = u\Delta x + q_{x,i+1} \quad (6)$$

### Forward Computation:

The equation (6) can be written as the equation (7).

$$Sum = u\Delta x + q_{x,i+1} \quad (7)$$



$q_{x,i+1}$  , in the equation (7) can be expressed as (8).

$$q_{x,i+1} = -\frac{N\Delta x}{2} + (i+1)\Delta x \quad (8)$$

By putting the equation (8) into the equation (7), we get the equation (9). To find  $u$  and  $i$  parameters the equation (9) can be written as the equation (10). Finally  $u+i$  calculated as in (10). If the left hand side of the equation named as  $z$  then the equations at (11) can be written. The block (11) is called as SG1. We get the necessary parameters  $u$  and  $i$  in the equations (11) for the block SG2.

$$Sum = u\Delta x + -\frac{N\Delta x}{2} + (i+1)\Delta x \quad (9)$$

$$\frac{Sum}{\Delta x} + \frac{N-2}{2} = u+i \quad (10)$$

$$z_k^l = \frac{Sum}{\Delta x} + \frac{N-2}{2}$$

$$i_k^l = \lfloor z_k^l \rfloor \quad (11)$$

$$u_k^l = z_k^l - \lfloor z_k^l \rfloor$$

As in the section 3.2, the equation (12) called as SG2 block and use  $u$  and  $i$  as inputs. Outputs of SG2 block is also the output of neuron.

$$O_k^l = F_{k,i_k^l}^l(u_k^l) = \sum_{m=0}^3 q_{k,(i_k^l+m)}^l C_{k,m}^l(u_k^l) \quad (12)$$

### **Backward Computation (Learning Phase with adaptive b-spline activation function)**

We will use the same gradient reduction technique, used in MLP with adaptive CR-spline activation functions. . The cost function is defined at (13). The equation (14) for the weights and the equation (15) for the control points will be used for adaptation.  $\mu_w$  and  $\mu_q$  represent the learning parameter for weights and control points.

$$E_p[t] = \sum_k \frac{1}{2} (o_k^l[t] - d_k^l[t])^2 \quad (13)$$

$$w_{kj}^l[t+1] = w_{kj}^l[t] - \mu_w \frac{\partial E_p[t]}{\partial w_{kj}^l[t]} \quad (14)$$

$$q_{k,(i_k^l+m)}^l[t+1] = q_{k,(i_k^l+m)}^l[t] - \mu_q \frac{\partial E_p[t]}{\partial q_{k,(i_k^l+m)}^l[t]} \quad (15)$$

where  $m = 0, \dots, 3$

By using the chain rule, the equation (16) can easily be written. When we write the derivatives on the right hand side, we get the equation (17)

$$\frac{\partial E_p[t]}{\partial w_{kj}^l} = \frac{\partial E_p[t]}{\partial o_k^l} \frac{\partial o_k^l[t]}{\partial u_k^l} \frac{\partial u_k^l[t]}{\partial \text{Sum}_k^l[t]} \frac{\partial \text{Sum}_k^l[t]}{\partial w_{kj}^l} \quad (16)$$

where  $l$  represents output layer

$$\frac{\partial E_p[t]}{\partial w_{kj}^l} = (o_k^l[t] - d_k[t]) f'(u_k^l[t]) \frac{1}{\Delta x} o_j^{l-1}[t] \quad (17)$$

where  $l$  represents output layer

We can also write the equation (17) in more detail like in (18). Now we will define the  $\delta_k^l[t]$  in the equation (19). Hence, the equation (18) can be written as the equation (20) shortly. This equation is used for the output layer. For the last hidden layer we use the equation (21). For the hidden layer we can write the equation (22) as the equation (24) with the new  $\delta$  value.

$$\frac{\partial E_p[t]}{\partial w_{kj}^l} = (o_k^l[t] - d_k[t]) [3(u_k^l)^2 - 2(u_k^l) + 1] \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} o_k^{l-1}[t] \quad (18)$$

where  $l$  represents output layer

$$\delta_k^l[t] = \frac{\partial E_p[t]}{\partial \text{Sum}_k^l[t]} = (o_k^l[t] - d_k[t]) f'(u_k^l[t]) \frac{1}{\Delta x} \quad (19)$$

$$\frac{\partial E_p[t]}{\partial w_{kj}^l[t]} = \delta_k^l[t] o_j^{l-1}[t] \quad (20)$$

Equation (21) can be written as the equation (22). Now we will define the  $\delta$ , as in (23), for the last hidden layer.

$$\frac{\partial E[t]}{\partial w_{kj}^{l-1}[t]} = \sum_k \frac{\partial E[t]}{\partial o_k^l[t]} \frac{\partial o_k^l[t]}{\partial u_k^l} \frac{\partial u_k^l}{\partial \text{sum}_k^l[t]} \frac{\partial \text{sum}_k^l[t]}{\partial o_k^{l-1}[t]} \frac{\partial o_k^{l-1}[t]}{\partial w_{kj}^{l-1}[t]} \quad (21)$$

$$\frac{\partial E[t]}{\partial w_{kj}^{l-1}[t]} = \sum_k \delta_k^l[t] w_{kj}^l \frac{\partial o_k^{l-1}[t]}{\partial w_{kj}^{l-1}[t]} \quad (22)$$

$$\delta_k^{l-1}[t] = \delta_k^l[t] w_{kj}^l \quad (23)$$

$$\begin{aligned} \frac{\partial E[t]}{\partial w_{kj}^{l-1}[t]} &= \sum_k \delta_k^{l-1}[t] \frac{\partial o_k^{l-1}[t]}{\partial u_k^{l-1}[t]} \frac{\partial u_k^{l-1}[t]}{\partial \text{Sum}_{kj}^{l-1}[t]} \frac{\partial \text{Sum}_k^{l-1}[t]}{\partial w_{kj}^{l-1}[t]} \\ &= \sum_k \delta_k^{l-1}[t] f'(u_k^{l-1}[t]) \frac{1}{\Delta x} o_j^{l-2} \end{aligned} \quad (24)$$

When we generalize these results for all hidden layers and output layer we get the  $\delta$  value as in (25) and Derivative for the weights as in (26).

$$\delta_k^{l-r}[t] = \begin{cases} \sum_k \delta_k^{l-r+1}[t] w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \\ (o_k^l[t] - d_k[t]) f'(u_k^l[t]) \frac{1}{\Delta x} & \text{where } r = 0 \end{cases} \quad (25)$$

$$\frac{\partial E}{\partial w_{kj}^{l-r}} = \begin{cases} \delta_k^{l-r} \cdot f'(u_k^{l-r}[t]) \frac{1}{\Delta x} o_j^{l-r-1} & \text{where } r \neq 0 \\ \delta_k^{l-r} o_j^{l-r-1} & \text{where } r = 0 \end{cases} \quad (26)$$

For the control points adaptation of the output layers' neurons we use the equation (27). For the control points' adaptation of the last hidden layer, we use the equation (28)

$$\begin{aligned} \frac{\partial E_p[t]}{\partial q_{k,(i_k^l+m)}^l[t]} &= \frac{\partial E_p[t]}{\partial o_k^l[t]} \frac{\partial o_k^l[t]}{\partial q_{k,(i_k^l+m)}^l[t]} = (o_k^l[t] - d_k[t]) \frac{\partial o_k^l[t]}{\partial q_{k,(i_k^l+m)}^l[t]} \\ &= (o_k^l[t] - d_k[t]) \frac{\partial F_{k,(i_k^l+m)}^l(s_k^l[t])}{\partial q_{k,(i_k^l+m)}^l[t]} = (o_k^l[t] - d_k[t]) C_{k,m}^l(u_k^l[t]) \\ &\quad \text{for } m = 0, \dots, 3 \end{aligned} \quad (27)$$

$$\begin{aligned} \frac{\partial E_p[t]}{\partial q_{k,(i_k^{l-1}+m)}^{l-1}[t]} &= \sum_k \frac{\partial E_p[t]}{\partial o_k^l[t]} \frac{\partial o_k^l[t]}{\partial u_k^l[t]} \frac{\partial u_k^l[t]}{\partial \text{Sum}_k^l[t]} \frac{\partial \text{Sum}_k^l[t]}{\partial o_k^{l-1}[t]} \frac{\partial o_k^{l-1}[t]}{\partial q_{k,(i_k^{l-1}+m)}^{l-1}[t]} \\ &= \sum_k (o_k^l[t] - d_k[t]) f'(u_k^l[t]) \frac{1}{\Delta x} w_{kj}^l[t] C_{k,m}^{l-1}(u_k^{l-1}[t]) \\ &\quad \text{for } m = 0, \dots, 3 \end{aligned} \quad (28)$$

To generalize this conclusion to the all hidden layers and output layer, we define a new parameter  $e$  as in (29). Hence, we define the derivative of cost function for control points as in (30). For the fixed points we use the derivatives in (31).

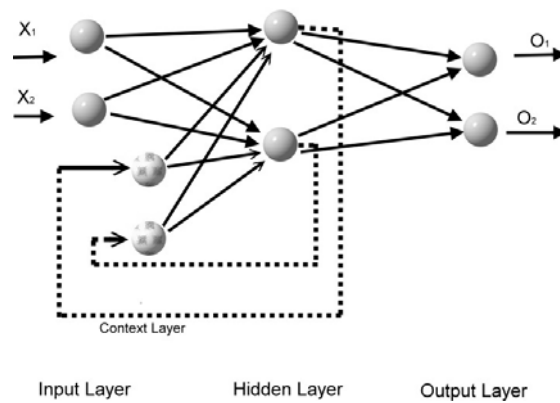
$$e_k^{l-r}[t] = \begin{cases} (o_k^l[t] - d_k[t]) & \text{where } r = 0 \\ \sum e_k^{l-r+1}[t] f'(u_k^{l-r+1}[t]) \frac{1}{\Delta x} w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \end{cases} \quad (29)$$

$$\frac{\partial E_p[t]}{\partial q_{k,(i_k^{l-r}+m)}^{l-r}[t]} = e_k^{l-r}[t] C_{k,m}^{l-r}(u_k^{l-r}[t]) \quad \text{for } m = 0, \dots, 3 \quad (30)$$

$$\begin{aligned} \frac{\partial F_{k,i_k^l}^l(s_k^l[t])}{\partial \text{sum}_k^l[t]} &= \frac{q_{k,1}^l - q_{k,0}^l}{\Delta x} \quad s_k^l < q_{k,l}^l \\ \frac{\partial F_{k,i_k^l}^l(s_k^l[t])}{\partial \text{sum}_k^l[t]} &= \frac{q_{k,N}^l - q_{k,N-1}^l}{\Delta x} \quad s_k^l > q_{k,N-1}^l \end{aligned} \quad (31)$$

## CHAPTER 4. ELMAN NETWORK

Elman networks [17] are in the class of recurrent neural networks. These networks store the hidden layer's output and give these outputs to the neuron with the other inputs in the next iteration. These outputs, very similar to other inputs, also have weights. A neuron can not understand differences of outputs from the other inputs. Neurons act them as if they are inputs, came from previous layer.



**Figure 4.1** An example of Elman Networks

### 4.1. Elman Networks with Sigmoid Activation

Elman Network has a new *Sum* function (1) for the hidden layers comparing with Multi Layer Perceptron. But for the output layer *Sum* function (2) is the same with the function used in Multi Layer Perceptron.

$$Sum_k^l[t] = \sum_{m=1}^j o_m^{l-1}[t]w_{km}^l[t] + \sum_{m=1}^k o_m^l[t-1]w_{k,(j+m)}^l \quad (1)$$

$$Sum_k^l[t] = \sum_{m=1}^j o_m^{l-1}[t] w_{km}^l[t] \quad (2)$$

### Learning Algorithm:

Elman networks also use Generalized Delta Learning Rule like MLP. Cost function and weight's adaptation function, described as (3) and (4) respectively.

$$E_p[t] = \sum_k \frac{1}{2} (o_k^l[t] - d_k^l[t])^2 \quad (3)$$

$$w_{kj}^l[t+1] = w_{kj}^l[t] + \eta \frac{\partial E_p[t]}{w_{kj}^l[t]} \quad (4)$$

The  $\delta$  parameters can be written as in (5) and  $\frac{\partial E}{\partial w}$  also can be written in (6).

$$\delta_k^{l-r}[t] = \begin{cases} \sum_k \delta_k^{l-r+1}[t] w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \\ (o_k^l[t] - d_k^l[t]) f'(sum_k^{l-r}[t]) & \text{where } r = 0 \end{cases} \quad (5)$$

$$\frac{\partial E_p[t]}{\partial w_{kj}^{l-r}[t]} = \begin{cases} \delta_k^{l-r}[t] f'(sum_k^{l-r}[t]) o_j^{l-r-1}[t] & \text{for hidden layer} \\ \delta_k^{l-r}[t] f'(sum_k^{l-r}[t]) o_{j+k}^{l-r}[t-1] & \text{for context layer} \\ \delta_k^{l-r}[t] o_j^{l-r-1}[t] & \text{where } r = 0 \end{cases} \quad (6)$$



#### 4.2. Elman Networks with Adaptive Catmull-Rom Spline Activation Function

In Standard Elman Networks, sigmoid function is used as activation function. We will use Catmull-Rom Spline Activation Function, used for Multi Layer Perceptrons before, for the Elman Networks. Cost function (1) is same as before.

$$E_p[t] = \sum_k \frac{1}{2} (o_k^l[t] - d_k^l[t])^2 \quad (1)$$

##### Forward Computation:

We get SG1 block output by the equations (2).

$$z_k^l = \frac{Sum}{\Delta x} + \frac{N-2}{2}$$
$$i_k^l = \lfloor z_k^l \rfloor \quad (2)$$
$$u_k^l = z_k^l - \lfloor z_k^l \rfloor$$

##### Backward Computation:

We use the equation (3) for the weight adaptation.

$$w_{kj}^l[t+1] = w_{kj}^l[t] - \mu_w \frac{\partial E_p[t]}{\partial w_{kj}^l[t]} \quad (3)$$

We used the  $\delta$ , as in the equation (4), and derivative formula as in the equation (5) for the Elman Networks with sigmoid activation functions.

$$\delta_k^{l-r}[t] = \begin{cases} \sum_k \delta_k^{l-r+1}[t] w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \\ (o_k^l[t] - d_k[t]) f'(sum_k^{l-r}[t]) & \text{where } r = 0 \end{cases} \quad (4)$$

$$\frac{\partial E_p[t]}{\partial w_{kj}^{l-r}[t]} = \begin{cases} \delta_k^{l-r}[t] f'(sum_k^{l-r}[t]) o_j^{l-r-1}[t] & \text{for hidden layer} \\ \delta_k^{l-r}[t] f'(sum_k^{l-r}[t]) o_{j+k}^{l-r}[t-1] & \text{for context layer} \\ \delta_k^{l-r}[t] o_j^{l-r-1}[t] & \text{where } r = 0 \end{cases} \quad (5)$$

$$\begin{aligned} f'(sum_k^l[t]) &= \frac{\partial o_k^l[t]}{\partial Sum_k^l[t]} = \frac{\partial o_k^l[t]}{\partial u_k^l[t]} \frac{\partial u_k^l[t]}{\partial Sum_k^l[t]} = f'(u_k^l[t]) \frac{1}{\Delta x} \\ &= [3(u_k^l[t])^2 \ 2(u_k^l[t]) \ 1] \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} \end{aligned} \quad (6)$$

If we apply (6) to (4) and (5) we derive the equations (7) and (8).

$$\delta_k^{l-r}[t] = \begin{cases} \sum_k \delta_k^{l-r+1}[t] w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \\ (o_k^l[t] - d_k[t]) f'(u_k^{l-r}[t]) \frac{1}{\Delta x} & \text{where } r = 0 \end{cases} \quad (7)$$

$$= \begin{cases} \sum_k \delta_k^{l-r+1}[t] w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \\ (o_k^l[t] - d_k[t]) [3(u_k^{l-r}[t])^2 - 2(u_k^{l-r}[t]) + 1] \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} & \text{where } r = 0 \end{cases}$$

$$\frac{\partial E_p[t]}{\partial w_{kj}^{l-r}[t]} = \begin{cases} \delta_k^{l-r}[t] [3(u_k^{l-r}[t])^2 - 2(u_k^{l-r}[t]) + 1] \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} o_j^{l-r-1}[t] & \text{for hidden layer} \\ \delta_k^{l-r}[t] [3(u_k^{l-r}[t])^2 - 2(u_k^{l-r}[t]) + 1] \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} o_{j+k}^{l-r}[t-1] & \text{for context layer} \\ \delta_k^{l-r}[t] o_j^{l-r-1}[t] & \text{where } r = 0 \end{cases} \quad (8)$$

For the control points adaptation we use the equation (9). The difference between Elman Networks with adaptive Catmull-Rom spline activation function and Multi Layer Perceptron with adaptive Catmull-Rom spline activation function is *Sum* function. Hence, there is no difference adapting the control points. For the output layers, the equation (10) is used.

$$q_{k,(i_k^l+m)}^l[t+1] = q_{k,(i_k^l+m)}^l[t] - \mu_q \frac{\partial E_p[t]}{\partial q_{k,(i_k^l+m)}^l[t]} \quad (9)$$

where  $m = 0, \dots, 3$

$$\begin{aligned} \frac{\partial E_p[t]}{\partial q_{k,(i_k^l+m)}^l[t]} &= \frac{\partial E_p[t]}{\partial o_k^l[t]} \frac{\partial o_k^l[t]}{\partial q_{k,(i_k^l+m)}^l[t]} = (o_k^l[t] - d_k[t]) \frac{\partial o_k^l[t]}{\partial q_{k,(i_k^l+m)}^l[t]} \\ &= (o_k^l[t] - d_k[t]) \frac{\partial F_{k,(i_k^l+m)}^l(s_k^l[t])}{\partial q_{k,(i_k^l+m)}^l[t]} = (o_k^l[t] - d_k[t]) C_{k,m}^l(u_k^l[t]) \end{aligned} \quad (10)$$

for  $m = 0, \dots, 3$

For the last hidden layer,

$$\begin{aligned} \frac{\partial E_p[t]}{\partial q_{k,(i_k^{l-1}+m)}^{l-1}[t]} &= \sum_k \frac{\partial E_p[t]}{\partial o_k^l[t]} \frac{\partial o_k^l[t]}{\partial u_k^l[t]} \frac{\partial u_k^l[t]}{\partial Sum_k^l[t]} \frac{\partial Sum_k^l[t]}{\partial o_k^{l-1}[t]} \frac{\partial o_k^{l-1}[t]}{\partial q_{k,(i_k^{l-1}+m)}^{l-1}[t]} \\ &= \sum_k (o_k^l[t] - d_k[t]) f'(u_k^l[t]) \frac{1}{\Delta x} w_{kj}^l[t] C_{k,m}^{l-1}(u_k^{l-1}[t]) \end{aligned} \quad (11)$$

for  $m = 0, \dots, 3$

When we generalize these results we get the equation (12) and (13)

$$e_k^{l-r}[t] = \begin{cases} (o_k^l[t] - d_k[t]) & \text{where } r=0 \\ \sum e_k^{l-r+1}[t] f'(u_k^{l-r+1}[t]) \frac{1}{\Delta x} w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \end{cases} \quad (12)$$

$$= \begin{cases} (o_k^l[t] - d_k[t]) & \text{where } r=0 \\ \sum e_k^{l-r+1}[t] [3(u_k^{l-r+1}[t])^2 \ 2(u_k^{l-r+1}[t]) \ 1] \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \end{cases}$$

$$\frac{\partial E_p[t]}{\partial q_{k,(i_k^{l-r}+m)}^{l-r}[t]} = e_k^{l-r}[t] C_{k,m}^{l-r}(u_k^{l-r}[t]) \quad \text{for } m = 0, \dots, 3 \quad (13)$$

We use the derivatives (14) for the fixed control paints.

$$\begin{aligned} \frac{\partial F_{k,i_k^l}^l(s_k^l[t])}{\partial \text{sum}_k^l[t]} &= \frac{q_{k,1}^l - q_{k,0}^l}{\Delta x} \quad s_k^l < q_{k,l}^l \\ \frac{\partial F_{k,i_k^l}^l(s_k^l[t])}{\partial \text{sum}_k^l[t]} &= \frac{q_{k,N}^l - q_{k,N-1}^l}{\Delta x} \quad s_k^l > q_{k,N-1}^l \end{aligned} \quad (14)$$

### 4.3 Elman Networks With Adaptive B-Spline Activation Functions

In this section we will apply adaptive b-spline activation function to the Elman Networks. The cost functions is described as in (1)

$$E_p[t] = \sum_k \frac{1}{2} (o_k^l[t] - d_k^l[t])^2 \quad (1)$$

#### Forward Computation:

We will use the same forward computation as we use for the CR spline activation functions. The block in (2) is called SG1.

$$\begin{aligned} z_k^l &= \frac{Sum}{\Delta x} + \frac{N-2}{2} \\ i_k^l &= \lfloor z_k^l \rfloor \\ u_k^l &= z_k^l - \lfloor z_k^l \rfloor \end{aligned} \quad (2)$$

#### Backward Computation:

We will use the equation (3) to adapt the weights as in CR splines.

$$w_{kj}^l[t+1] = w_{kj}^l[t] - \mu_w \frac{\partial E_p[t]}{\partial w_{kj}^l[t]} \quad (3)$$

The  $\delta$  and the derivative  $\frac{\partial E}{\partial w}$  are described as in (4) and (5) respectively for the standard Elman Networks, which use sigmoid function as activation function. We can easily adapt these two equations for the b-spline activation function.

$$\delta_k^{l-r}[t] = \begin{cases} \sum_k \delta_k^{l-r+1}[t] w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \\ (o_k^l[t] - d_k[t]) f'(sum_k^{l-r}[t]) & \text{where } r = 0 \end{cases} \quad (4)$$

$$\frac{\partial E_p[t]}{\partial w_{kj}^{l-r}[t]} = \begin{cases} \delta_k^{l-r}[t] f'(sum_k^{l-r}[t]) o_j^{l-r-1}[t] & \text{for hidden layer} \\ \delta_k^{l-r}[t] f'(sum_k^{l-r}[t]) o_{j+k}^{l-r}[t-1] & \text{for context layer} \\ \delta_k^{l-r}[t] o_j^{l-r-1}[t] & \text{where } r = 0 \end{cases} \quad (5)$$

The only change is  $f'(sum_k^l[t])$  which can be derived as in (6).

$$\begin{aligned} f'(sum_k^l[t]) &= \frac{\partial o_k^l[t]}{\partial Sum_k^l[t]} = \frac{\partial o_k^l[t]}{\partial u_k^l[t]} \frac{\partial u_k^l[t]}{\partial Sum_k^l[t]} = f'(u_k^l[t]) \frac{1}{\Delta x} \\ &= [3(u_k^l)^2 \ 2(u_k^l) \ 1] \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} \end{aligned} \quad (6)$$

Now we can write the  $\delta$  and the derivative  $\frac{\partial E}{\partial w}$  as in (7) and (8).

$$\delta_k^{l-r}[t] = \begin{cases} \sum_k \delta_k^{l-r+1}[t] w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \\ (o_k^l[t] - d_k[t]) f'(u_k^{l-r}[t]) \frac{1}{\Delta x} & \text{where } r = 0 \end{cases} \quad (7)$$

$$= \begin{cases} \sum_k \delta_k^{l-r+1}[t] w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \\ (o_k^l[t] - d_k[t]) [3(u_k^{l-r}[t])^2 - 2(u_k^{l-r}[t]) + 1] \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} & \text{where } r = 0 \end{cases}$$

$$\frac{\partial E_p[t]}{\partial w_{kj}^{l-r}[t]} = \begin{cases} \delta_k^{l-r}[t] [3(u_k^{l-r}[t])^2 - 2(u_k^{l-r}[t]) + 1] \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} o_j^{l-r-1}[t] & \text{for hidden layer} \\ \delta_k^{l-r}[t] [3(u_k^{l-r}[t])^2 - 2(u_k^{l-r}[t]) + 1] \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} o_{j+k}^{l-r}[t-1] & \text{for context layer} \\ \delta_k^{l-r}[t] o_j^{l-r-1}[t] & \text{where } r = 0 \end{cases}$$

(8)



We will use the equation (9) to adapt the control points. Since the only difference between Elman Networks and MLP is the *Sum* function, there is no difference between Elman Networks and MLP for the adaptation of control points.

$$q_{k,(i_k^l+m)}^l[t+1] = q_{k,(i_k^l+m)}^l[t] - \mu_q \frac{\partial E_p[t]}{\partial q_{k,(i_k^l+m)}^l[t]} \quad (9)$$

where  $m = 0, \dots, 3$

For output layer, (10) is used.

$$\begin{aligned} \frac{\partial E_p[t]}{\partial q_{k,(i_k^l+m)}^l[t]} &= \frac{\partial E_p[t]}{\partial o_k^l[t]} \frac{\partial o_k^l[t]}{\partial q_{k,(i_k^l+m)}^l[t]} = (o_k^l[t] - d_k[t]) \frac{\partial o_k^l[t]}{\partial q_{k,(i_k^l+m)}^l[t]} \\ &= (o_k^l[t] - d_k[t]) \frac{\partial F_{k,(i_k^l+m)}^l(s_k^l[t])}{\partial q_{k,(i_k^l+m)}^l[t]} = (o_k^l[t] - d_k[t]) C_{k,m}^l(u_k^l[t]) \end{aligned} \quad (10)$$

for  $m = 0, \dots, 3$

For last hidden layer, (11) is used

$$\begin{aligned} \frac{\partial E_p[t]}{\partial q_{k,(i_k^{l-1}+m)}^{l-1}[t]} &= \sum_k \frac{\partial E_p[t]}{\partial o_k^l[t]} \frac{\partial o_k^l[t]}{\partial u_k^l[t]} \frac{\partial u_k^l[t]}{\partial \text{Sum}_k^l[t]} \frac{\partial \text{Sum}_k^l[t]}{\partial o_k^{l-1}[t]} \frac{\partial o_k^{l-1}[t]}{\partial q_{k,(i_k^{l-1}+m)}^{l-1}[t]} \\ &= \sum_k (o_k^l[t] - d_k[t]) f'(u_k^l[t]) \frac{1}{\Delta x} w_{kj}^l[t] C_{k,m}^{l-1}(u_k^{l-1}[t]) \end{aligned} \quad (11)$$

for  $m = 0, \dots, 3$

By generalizing these results, we get the (12) and (13).

$$e_k^{l-r}[t] = \begin{cases} (o_k^l[t] - d_k[t]) & \text{where } r = 0 \\ \sum e_k^{l-r+1}[t] f'(u_k^{l-r+1}[t]) \frac{1}{\Delta x} w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \end{cases} \quad (12)$$

$$= \begin{cases} (o_k^l[t] - d_k[t]) & \text{where } r = 0 \\ \sum e_k^{l-r+1}[t] [3(u_k^{l-r+1}[t])^2 \ 2(u_k^{l-r+1}[t]) \ 1] \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \end{cases}$$

$$\frac{\partial E_p[t]}{\partial q_{k,(i_k^{l-r}+m)}^{l-r}} = e_k^{l-r}[t] C_{k,m}^{l-r}(u_k^{l-r}[t]) \quad \text{for } m = 0, \dots, 3 \quad (13)$$

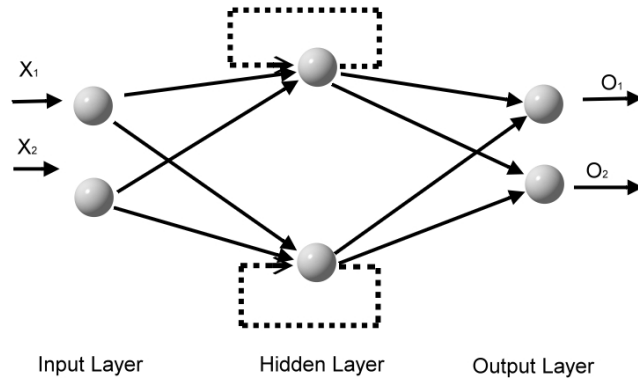
We use the derivatives (14) for the fixed control points.

$$\begin{aligned} \frac{\partial F_{k,i_k}^l(s_k^l[t])}{\partial \text{sum}_k^l[t]} &= \frac{q_{k,1}^l - q_{k,0}^l}{\Delta x} \quad s_k^l < q_{k,l}^l \\ \frac{\partial F_{k,i_k}^l(s_k^l[t])}{\partial \text{sum}_k^l[t]} &= \frac{q_{k,N}^l - q_{k,N-1}^l}{\Delta x} \quad s_k^l > q_{k,N-1}^l \end{aligned} \quad (14)$$

## CHAPTER 5. LOCALLY RECURRENT NEURAL NETWORKS

### 5.1 Locally Recurrent Neural Networks with Sigmoid Activation Function

LRNN is a member of recurrent networks. The only difference from the MLP is the *Sum* function of the hidden layers' neuron. A hidden layer's neuron stores the output of itself and sent to the *Sum* function of itself as weighted input.



**Figure 5.1** Locally Recurrent Neural Networks

The *Sum* function for hidden Layer's neuron which is the only difference of LRNN from the MLP, shown in (1). The *Sum* function (2) for output layer is the same with the MLP.

$$Sum_k^l[t] = \sum_{m=1}^j o_m^{l-1}[t]w_{km}^l[t] + o_k^l[t-1]w_{k,(j+1)}^l \quad (1)$$

$$Sum_k^l[t] = \sum_{m=1}^j o_m^{l-1}[t]w_{km}^l[t] \quad (2)$$

### Learning Algorithm:

LRNN networks use Generalized Delta Learning rule. It uses (3), (4), and (5) for the adaptation of the weights.

$$w_{kj}^l[t+1] = w_{kj}^l[t] + \eta \frac{\partial E_p[t]}{w_{kj}^l[t]} \quad (3)$$

$$\delta_k^{l-r}[t] = \begin{cases} \sum_k \delta_k^{l-r+1}[t] w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \\ (o_k[t] - d_k[t]) f'(sum_k^{l-r}[t]) & \text{where } r = 0 \end{cases} \quad (4)$$

$$\frac{\partial E_p[t]}{\partial w_{kj}^{l-r}[t]} = \begin{cases} \delta_k^{l-r}[t] f'(sum_k^{l-r}[t]) o_j^{l-r-1}[t] & \text{for hidden layer} \\ \delta_k^{l-r}[t] f'(sum_k^{l-r}[t]) o_{j+1}^{l-r}[t-1] & \text{for feedback weight} \\ \delta_k^{l-r}[t] o_j^{l-r-1}[t] & \text{where } r = 0 \end{cases} \quad (5)$$

## 5.2 Locally Recurrent Neural Networks with Adaptive Catmull-Rom Spline Activation Function

Standard LRNN networks use the sigmoid activation function. We will apply the CR-Splines activation function to the LRNN. The cost function is the same as before as in (1).

$$E_p[t] = \sum_k \frac{1}{2} (o_k[t] - d_k[t])^2 \quad (1)$$

### Forward Computation:

We use the block SG1 without changing as in (2)

$$\begin{aligned} z_k^l &= \frac{Sum}{\Delta x} + \frac{N-2}{2} \\ i_k^l &= \lfloor z_k^l \rfloor \\ u_k^l &= z_k^l - \lfloor z_k^l \rfloor \end{aligned} \quad (2)$$

### Backward Computation:

We use (3) to adapt weights. Standard LRNN has the  $\delta$  and the derivative  $\frac{\partial E}{\partial w}$  as in (4) and (5) respectively

$$w_{kj}^l[t+1] = w_{kj}^l[t] - \mu_w \frac{\partial E_p[t]}{\partial w_{kj}^l[t]} \quad (3)$$

$$\delta_k^{l-r}[t] = \begin{cases} \sum_k \delta_k^{l-r+1}[t] w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \\ (o_k^l[t] - d_k[t]) f'(sum_k^{l-r}[t]) & \text{where } r = 0 \end{cases} \quad (4)$$

$$\frac{\partial E_p[t]}{\partial w_{kj}^{l-r}[t]} = \begin{cases} \delta_k^{l-r}[t] f'(sum_k^{l-r}[t]) o_j^{l-r-1}[t] & \text{for hidden layer} \\ \delta_k^{l-r}[t] f'(sum_k^{l-r}[t]) o_{j+1}^{l-r}[t-1] & \text{for feedback weight} \\ \delta_k^{l-r}[t] o_j^{l-r-1}[t] & \text{where } r=0 \end{cases} \quad (5)$$

We can write  $f'(sum_k^l[t])$  for LRNN with CR Spline functions as in (6).

$$\begin{aligned} f'(sum_k^l[t]) &= \frac{\partial o_k^l[t]}{\partial Sum_k^l[t]} = \frac{\partial o_k^l[t]}{\partial u_k^l[t]} \frac{\partial u_k^l[t]}{\partial Sum_k^l[t]} = f'(u_k^l[t]) \frac{1}{\Delta x} \\ &= [3(u_k^l[t])^2 \ 2(u_k^l[t]) \ 1] \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} \end{aligned} \quad (6)$$

We can write (4) and (5) for the LRNN as (7) and (8) considering the (6).

$$\delta_k^{l-r}[t] = \begin{cases} \sum_k \delta_k^{l-r+1}[t] w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \\ (o_k^l[t] - d_k[t]) f'(u_k^{l-r}[t]) \frac{1}{\Delta x} & \text{where } r = 0 \end{cases} \quad (7)$$

$$= \begin{cases} \sum_k \delta_k^{l-r+1}[t] w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \\ (o_k^l[t] - d_k[t]) [3(u_k^{l-r}[t])^2 \ 2(u_k^{l-r}[t]) \ 1] \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} & \text{where } r = 0 \end{cases}$$

$$\frac{\partial E_p[t]}{\partial w_{kj}^{l-r}[t]} = \left\{ \begin{array}{l} \delta_k^{l-r}[t][3(u_k^{l-r}[t])^2 - 2(u_k^{l-r}[t]) + 1] \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} o_j^{l-r-1}[t] \\ \text{for hidden layer} \\ \\ \delta_k^{l-r}[t][3(u_k^{l-r}[t])^2 - 2(u_k^{l-r}[t]) + 1] \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} o_{j+1}^{l-r-1}[t-1] \\ \text{for context layer} \\ \\ \delta_k^{l-r}[t] o_j^{l-r-1}[t] \\ \text{where } r=0 \end{array} \right. \quad (8)$$

There is no change in the way of adapting the control points.

$$q_{k,(i_k^l+m)}^l[t+1] = q_{k,(i_k^l+m)}^l[t] - \mu_q \frac{\partial E_p[t]}{\partial q_{k,(i_k^l+m)}^l[t]} \quad (8)$$

where  $m = 0, \dots, 3$

For output layer, we use (9). For last hidden layer, we use (10). When we generalize the results, we got before, we can write the equations (11) and (12) for LRNN.

$$\begin{aligned}
\frac{\partial E_p[t]}{\partial q_{k,(i_k^l+m)}^l[t]} &= \frac{\partial E_p[t]}{\partial o_k^l[t]} \frac{\partial o_k^l[t]}{\partial q_{k,(i_k^l+m)}^l[t]} = (o_k^l[t] - d_k[t]) \frac{\partial o_k^l[t]}{\partial q_{k,(i_k^l+m)}^l[t]} \\
&= (o_k^l[t] - d_k[t]) \frac{\partial F_{k,(i_k^l+m)}^l(s_k^l[t])}{\partial q_{k,(i_k^l+m)}^l[t]} = (o_k^l[t] - d_k[t]) C_{k,m}^l(u_k^l[t]) \\
&\quad \text{for } m = 0, \dots, 3
\end{aligned} \tag{9}$$

$$\begin{aligned}
\frac{\partial E_p[t]}{\partial q_{k,(i_k^{l-1}+m)}^{l-1}[t]} &= \sum_k \frac{\partial E_p[t]}{\partial o_k^l[t]} \frac{\partial o_k^l[t]}{\partial u_k^l[t]} \frac{\partial u_k^l[t]}{\partial \text{Sum}_k^l[t]} \frac{\partial \text{Sum}_k^l[t]}{\partial o_k^{l-1}[t]} \frac{\partial o_k^{l-1}[t]}{\partial q_{k,(i_k^{l-1}+m)}^{l-1}[t]} \\
&= \sum_k (o_k^l[t] - d_k[t]) f'(u_k^l[t]) \frac{1}{\Delta x} w_{kj}^l[t] C_{k,m}^{l-1}(u_k^{l-1}[t]) \\
&\quad \text{for } m = 0, \dots, 3
\end{aligned} \tag{10}$$

We use the derivatives (11) for the fixed control points.

$$\begin{aligned}
\frac{\partial F_{k,i_k^l}^l(s_k^l[t])}{\partial \text{sum}_k^l[t]} &= \frac{q_{k,1}^l - q_{k,0}^l}{\Delta x} \quad s_k^l < q_{k,l}^l \\
\frac{\partial F_{k,i_k^l}^l(s_k^l[t])}{\partial \text{sum}_k^l[t]} &= \frac{q_{k,N}^l - q_{k,N-1}^l}{\Delta x} \quad s_k^l > q_{k,N-1}^l
\end{aligned} \tag{11}$$



$$e_k^{l-r}[t] = \begin{cases} (o_k^l[t] - d_k[t]) & \text{where } r = 0 \\ \sum e_k^{l-r+1}[t] f'(u_k^{l-r+1}[t]) \frac{1}{\Delta x} w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \end{cases} \quad (12)$$

$$= \begin{cases} (o_k^l[t] - d_k[t]) & \text{where } r = 0 \\ \sum e_k^{l-r+1}[t] [3(u_k^{l-r+1}[t])^2 \ 2(u_k^{l-r+1}[t]) \ 1] \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \end{cases}$$

$$\frac{\partial E_p[t]}{\partial q_{k,(i_k^{l-r}+m)}^{l-r}[t]} = e_k^{l-r}[t] C_{k,m}^{l-r}(u_k^{l-r}[t]) \quad \text{for } m = 0, \dots, 3 \quad (13)$$

### 5.3 Locally Recurrent Neural Networks with Adaptive B-Spline Activation Function

In this section, we will apply adaptive b-spline activation function to the LRNN. The cost function as in (1)

$$E_p[t] = \sum_k \frac{1}{2} (o_k^l[t] - d_k^l[t])^2 \quad (1)$$

### Forward Computation:

We use the same SG1 block, we used before as in (2).

$$\begin{aligned} z_k^l &= \frac{Sum}{\Delta x} + \frac{N-2}{2} \\ i_k^l &= \lfloor z_k^l \rfloor \\ u_k^l &= z_k^l - \lfloor z_k^l \rfloor \end{aligned} \quad (2)$$

### Backward Computation:

To change the weights we will use the equation (3). The  $\delta$  and the derivative  $\frac{\partial E}{\partial w}$  for the standard LRNN are as in (4) and (5) respectively.

$$w_{kj}^l[t+1] = w_{kj}^l[t] - \mu_w \frac{\partial E_p[t]}{\partial w_{kj}^l[t]} \quad (3)$$

$$\delta_k^{l-r}[t] = \begin{cases} \sum_k \delta_k^{l-r+1}[t] w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \\ (o_k^l[t] - d_k[t]) f'(sum_k^{l-r}[t]) & \text{where } r = 0 \end{cases} \quad (4)$$

$$\frac{\partial E_p[t]}{\partial w_{kj}^{l-r}[t]} = \begin{cases} \delta_k^{l-r}[t] f'(sum_k^{l-r}[t]) o_j^{l-r-1}[t] & \text{for hidden layer} \\ \delta_k^{l-r}[t] f'(sum_k^{l-r}[t]) o_{j+1}^{l-r}[t-1] & \text{for feedback weight} \\ \delta_k^{l-r}[t] o_j^{l-r-1}[t] & \text{where } r=0 \end{cases} \quad (5)$$

We can write the  $f'(sum_k^l[t])$  as in (6). The equation (4) and (5) can be adapt for LRNN with adaptive b-spline activation function as in (7) and (8) respecting the equation (5).

$$\begin{aligned} f'(sum_k^l[t]) &= \frac{\partial o_k^l[t]}{\partial Sum_k^l[t]} = \frac{\partial o_k^l[t]}{\partial u_k^l[t]} \frac{\partial u_k^l[t]}{\partial Sum_k^l[t]} = f'(u_k^l[t]) \frac{1}{\Delta x} \\ &= [3(u_k^l)^2 \ 2(u_k^l) \ 1] \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} \end{aligned} \quad (6)$$

$$\delta_k^{l-r}[t] = \begin{cases} \sum_k \delta_k^{l-r+1}[t] w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \\ (o_k^l[t] - d_k[t]) f'(u_k^{l-r}[t]) \frac{1}{\Delta x} & \text{where } r=0 \end{cases} \quad (7)$$

$$\begin{aligned} &\begin{cases} \sum_k \delta_k^{l-r+1}[t] w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \\ (o_k^l[t] - d_k[t]) [3(u_k^{l-r}[t])^2 \ 2(u_k^{l-r}[t]) \ 1] \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} & \text{where } r=0 \end{cases} \end{aligned}$$

$$\frac{\partial E_p[t]}{\partial w_{kj}^{l-r}[t]} = \begin{cases} \delta_k^{l-r}[t][3(u_k^{l-r}[t])^2 - 2(u_k^{l-r}[t]) + 1] \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} o_j^{l-r-1}[t] \\ \text{for hidden layer} \\ \delta_k^{l-r}[t][3(u_k^{l-r}[t])^2 - 2(u_k^{l-r}[t]) + 1] \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} o_{j+1}^{l-r}[t-1] \\ \text{for context layer} \\ \delta_k^{l-r}[t] o_j^{l-r-1}[t] \\ \text{where } r=0 \end{cases} \quad (8)$$

The way we use to adapt control points exactly the same, we used before for previous network models. (9) is used to adapt control points. For output layer, we use the equation (10).

$$q_{k,(i_k^l+m)}^l[t+1] = q_{k,(i_k^l+m)}^l[t] - \mu_q \frac{\partial E_p[t]}{\partial q_{k,(i_k^l+m)}^l[t]} \quad (9)$$

where  $m = 0, \dots, 3$

$$\begin{aligned} \frac{\partial E_p[t]}{\partial q_{k,(i_k^l+m)}^l[t]} &= \frac{\partial E_p[t]}{\partial o_k^l[t]} \frac{\partial o_k^l[t]}{\partial q_{k,(i_k^l+m)}^l[t]} = (o_k^l[t] - d_k[t]) \frac{\partial o_k^l[t]}{\partial q_{k,(i_k^l+m)}^l[t]} \\ &= (o_k^l[t] - d_k[t]) \frac{\partial F_{k,(i_k^l+m)}^l(s_k^l[t])}{\partial q_{k,(i_k^l+m)}^l[t]} = (o_k^l[t] - d_k[t]) C_{k,m}^l(u_k^l[t]) \end{aligned} \quad (10)$$

for  $m = 0, \dots, 3$

For the last hidden layer, we use the equation (11). If we generalize the results, we got, we can write (12) and (13) for the LRNN. We use the derivatives (14) for the fixed control points.

$$\frac{\partial E_p[t]}{\partial q_{k,(i_k^{l-1}+m)}[t]} = \sum_k \frac{\partial E_p[t]}{\partial o_k^l[t]} \frac{\partial o_k^l[t]}{\partial u_k^l[t]} \frac{\partial u_k^l[t]}{\partial \text{Sum}_k^l[t]} \frac{\partial \text{Sum}_k^l[t]}{\partial o_k^{l-1}[t]} \frac{\partial o_k^{l-1}[t]}{\partial q_{k,(i_k^{l-1}+m)}[t]}$$

$$\sum_k (o_k^l[t] - d_k[t]) f'(u_k^l[t]) \frac{1}{\Delta x} w_{kj}^l[t] C_{k,m}^{l-1}(u_k^{l-1}[t])$$

for  $m = 0, \dots, 3$

(11)

$$e_k^{l-r}[t] = \begin{cases} (o_k^l[t] - d_k[t]) & \text{where } r = 0 \\ \sum e_k^{l-r+1}[t] f'(u_k^{l-r+1}[t]) \frac{1}{\Delta x} w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \end{cases}$$
(12)

$$= \begin{cases} (o_k^l[t] - d_k[t]) & \text{where } r = 0 \\ \sum e_k^{l-r+1}[t] [3(u_k^{l-r+1}[t])^2 - 2(u_k^{l-r+1}[t]) + 1] \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \end{bmatrix} \begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \frac{1}{\Delta x} w_{kj}^{l-r+1}[t] & \text{where } r \neq 0 \end{cases}$$

$$\frac{\partial E_p[t]}{\partial q_{k,(i_k^{l-r}+m)}[t]} = e_k^{l-r}[t] C_{k,m}^{l-r}(u_k^{l-r}[t]) \quad \text{for } m = 0, \dots, 3$$
(13)

$$\frac{\partial F_{k,i_k^l}^l(s_k^l[t])}{\partial \text{sum}_k^l[t]} = \frac{q_{k,1}^l - q_{k,0}^l}{\Delta x} \quad s_k^l < q_{k,1}^l$$

$$\frac{\partial F_{k,i_k^l}^l(s_k^l[t])}{\partial \text{sum}_k^l[t]} = \frac{q_{k,N}^l - q_{k,N-1}^l}{\Delta x} \quad s_k^l > q_{k,N-1}^l$$
(14)

## CHAPTER 6. PERFORMANCE OF THE NETWORKS

In this chapter, we will compare the performance of the network on the famous xor problem, which is commonly used as criteria to measure the performance of the artificial neural networks. We use the artificial neural network simulator SBNN which is explained in Appendix. On all the comparison we used following values

- Number of hidden layers :1
- Number of neuron per hidden layers :10
- Initial weights :randomly
- Number of epoch :100,000
- Learning rate for weight :0.5
- MSE period :100
- Number of control points for spline activation functions :80
- Distance between the control points :1

While we compare the networks we use a computer which has Intel Celeron 1.7 GHz cpu, 256 ddr ram with operating system Ms. windows xp.

### 6.1 Comparison of the Multi Layer Perceptron

In this section, we will compare the performance of Multi Layer Perceptron with three different kinds of activation function, sigmoid activation functions, adaptive Catmull-Rom spline activation functions, and adaptive b-spline activation functions with the values, written on the above.

#### 6.1.1. Comparison of the sigmoid activation functions and the B-spline activation function for MLP

When we look at the graph on the Figure 6.1 we see that under the 1900 epoch sigmoid function has a better performance but after the 1900 epoch adaptive b-spline activation function has significantly better performance. The performance of sigmoid

activation function is getting worse while the iteration number is increasing comparing with the b-spline activation functions.

### 6.1.2. Comparison of the Catmull-Rom spline activation functions and the B-spline activation functions for MLP

Catmull-Rom spline activation function has a great performance at the beginning of the epochs. As we can see at the Figure, 6.2 b-spline functions are learning very slowly comparing with the CR spline functions. But after a certain amount of epoch nearly 6800 epoch the performance of the B-spline functions are increasing on the other hand the performance of CR spline functions are dramatically decreasing we can easily see it in the Figure 6.3

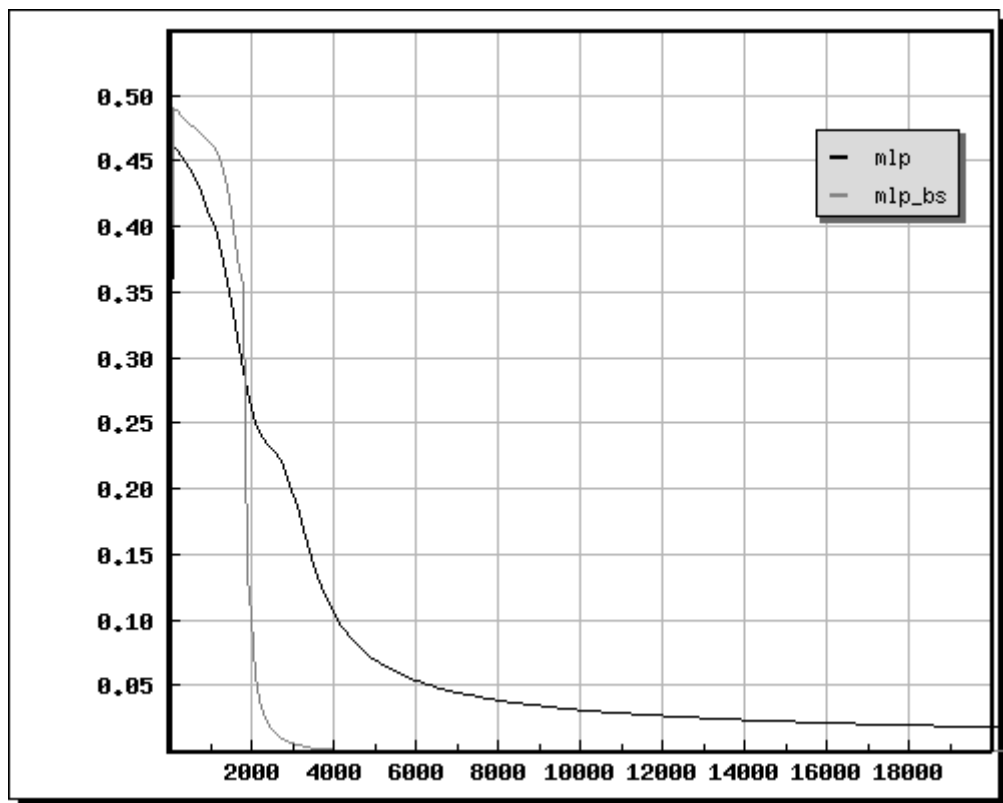
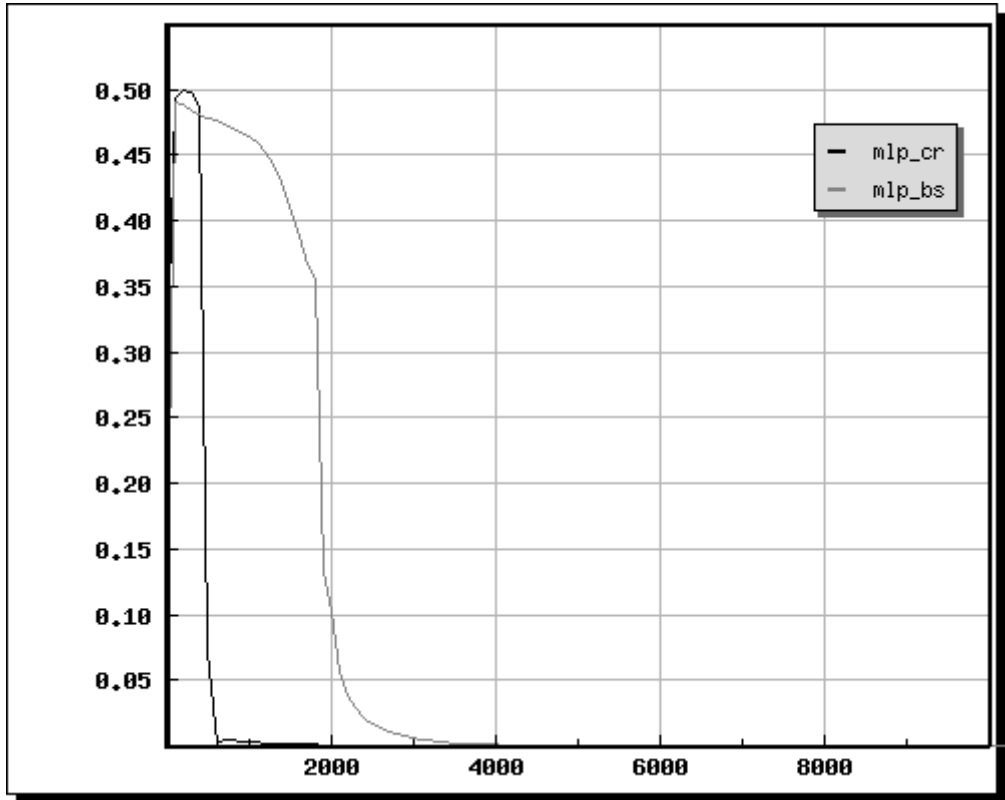
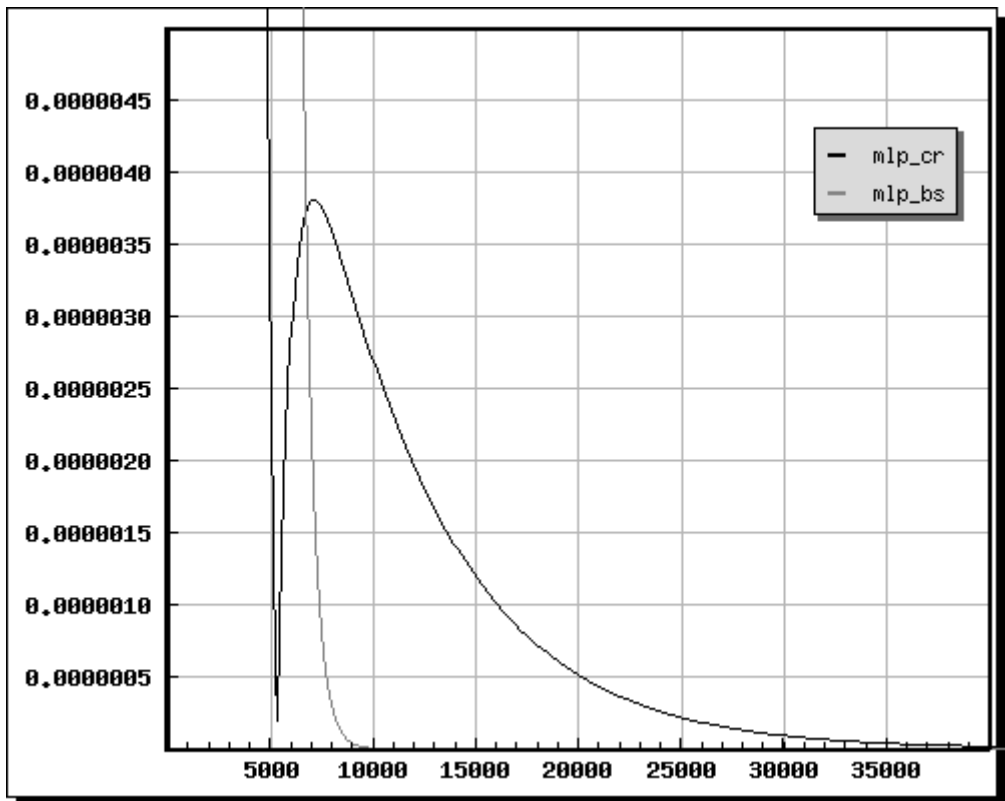


Figure 6.1 Performances of MLPs with sigmoid and adaptive b-spline activation functions for xor problem



**Figure 6.2** Performances of MLPs with adaptive CR spline and adaptive b-spline activation functions for xor problem



**Figure 6.3** Performances of MLPs with adaptive CR spline and adaptive b-spline activation functions for xor problem in more detail



## 6.2 Comparison of the Elman Networks

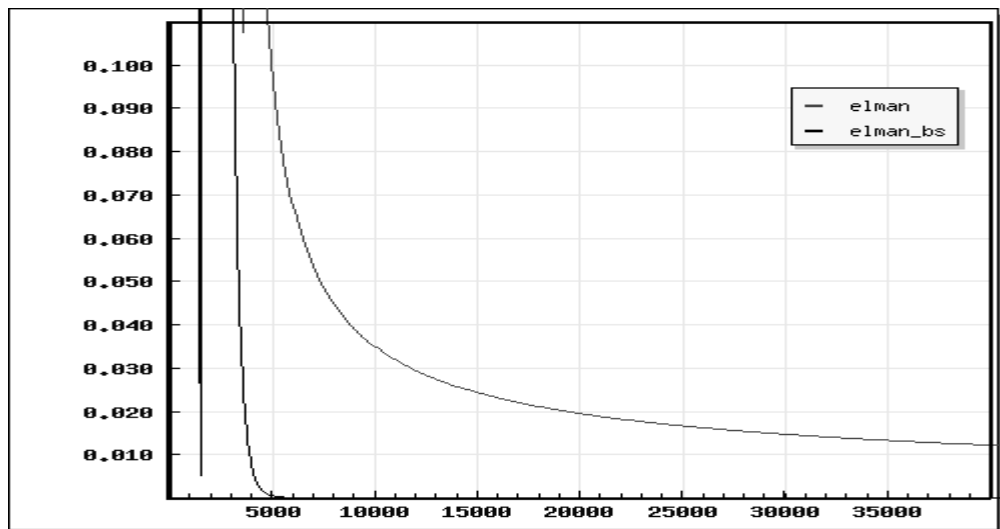
In this section, we will compare the performance of Multi Layer Perceptron with three different kinds of activation function, sigmoid activation functions, adaptive Catmull-Rom spline activation functions, and adaptive b-spline activation functions for Elman Network.

### 6.2.1. Comparison of the sigmoid activation functions and the B-spline activation function for Elman Networks

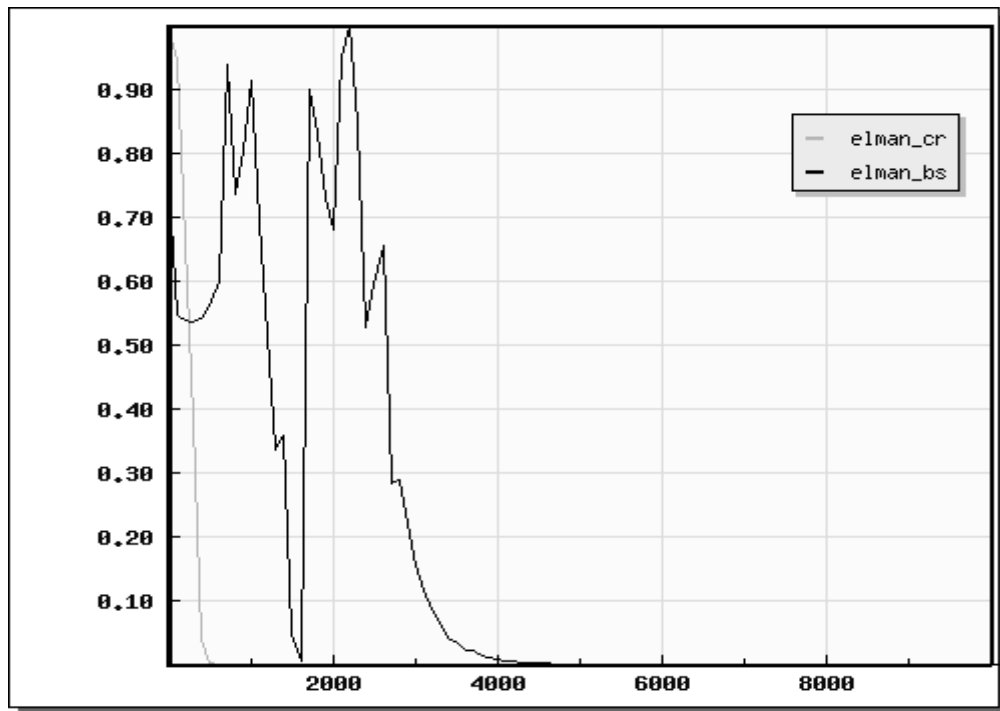
As we can easily see at the Figure 6.4, b-spline activation function has significantly better performance than the sigmoid function.

### 6.2.2. Comparison of the Catmull-Rom spline activation functions and the B-spline activation functions for Elman Networks

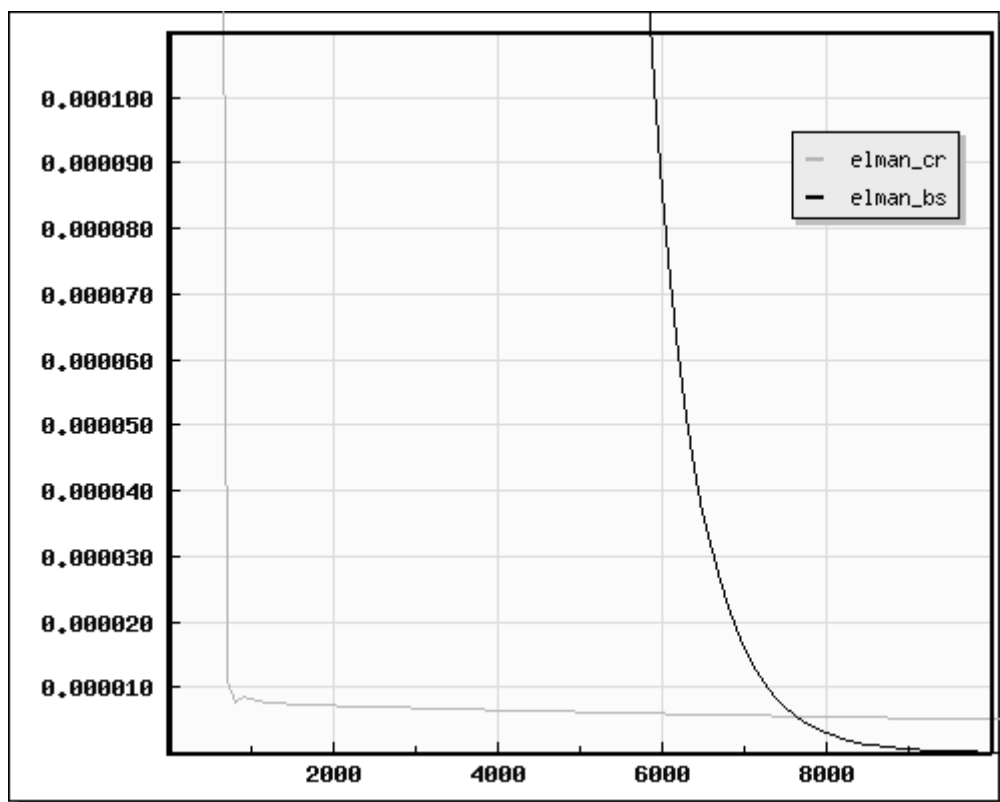
If we look at the graphs at Figure 6.5 and Figure 6.6 we see that this two functions effects the performance of Elman Network in the same way as they effects the performance in MLP. CR spline functions effects the performance very fast but while the number of epochs are increasing the positive effect of CR spline function on the performance are decreased. On the other hand, b-spline activation functions show its effect a bit later.



**Figure 6.4** Performances of Elman Networks with sigmoid and adaptive b-spline activation function for xor problem



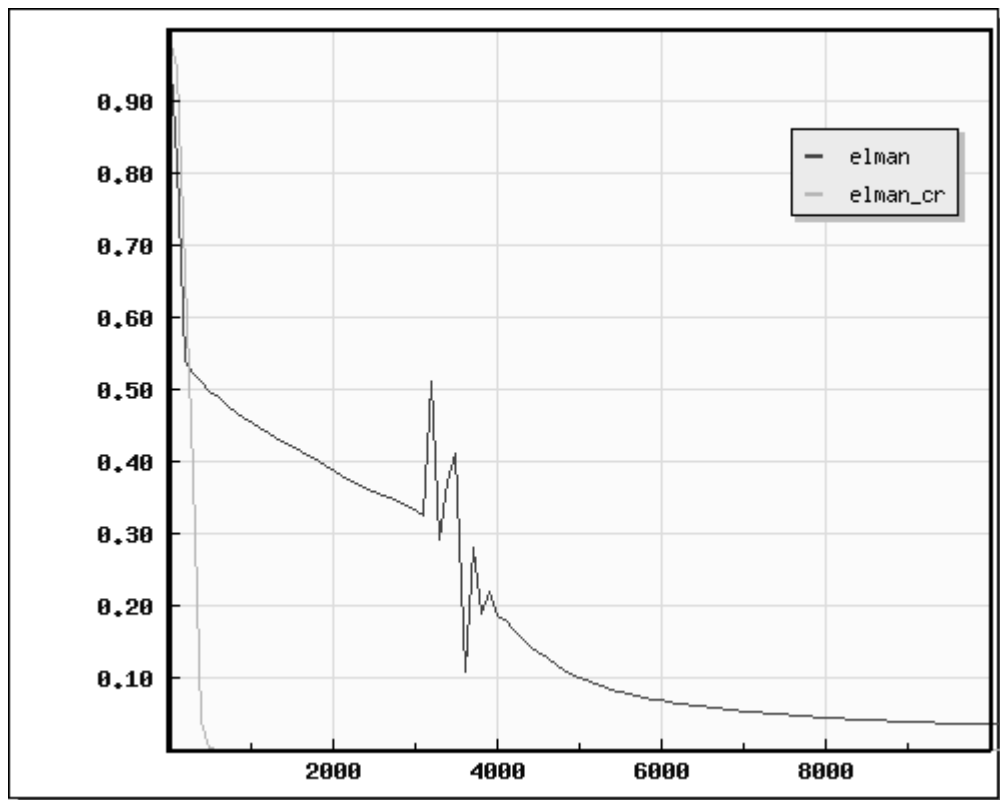
**Figure 6.5** Performances of Elman Networks with adaptive CR spline and adaptive b-spline activation functions for xor problem



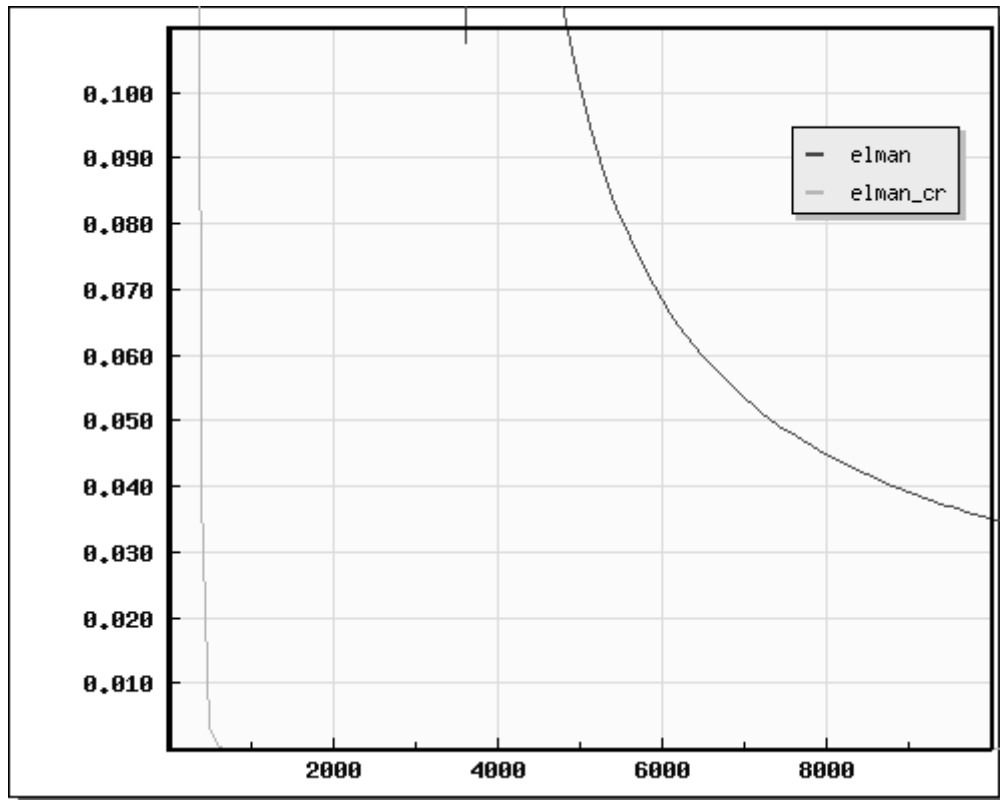
**Figure 6.6** Performances of Elman Networks with adaptive CR spline and adaptive b-spline activation functions for xor problem in more detail

### 6.2.3. Comparison of the Catmull-Rom spline activation functions and the sigmoid activation functions for Elman Networks

When we look at the Figure 6.7 CR splines has a quick performance but after a while sigmoid function is reaching the performance of the CR splines. To understand whether the sigmoid function will catch or pass the CR spline we should look in more detail like the Figure 6.8



**Figure 6.7** Performances of Elman Networks with sigmoid and adaptive CR spline activation functions for xor problem



**Figure 6.8** Performances of Elman Networks with sigmoid and adaptive CR spline activation functions for xor problem in more detail

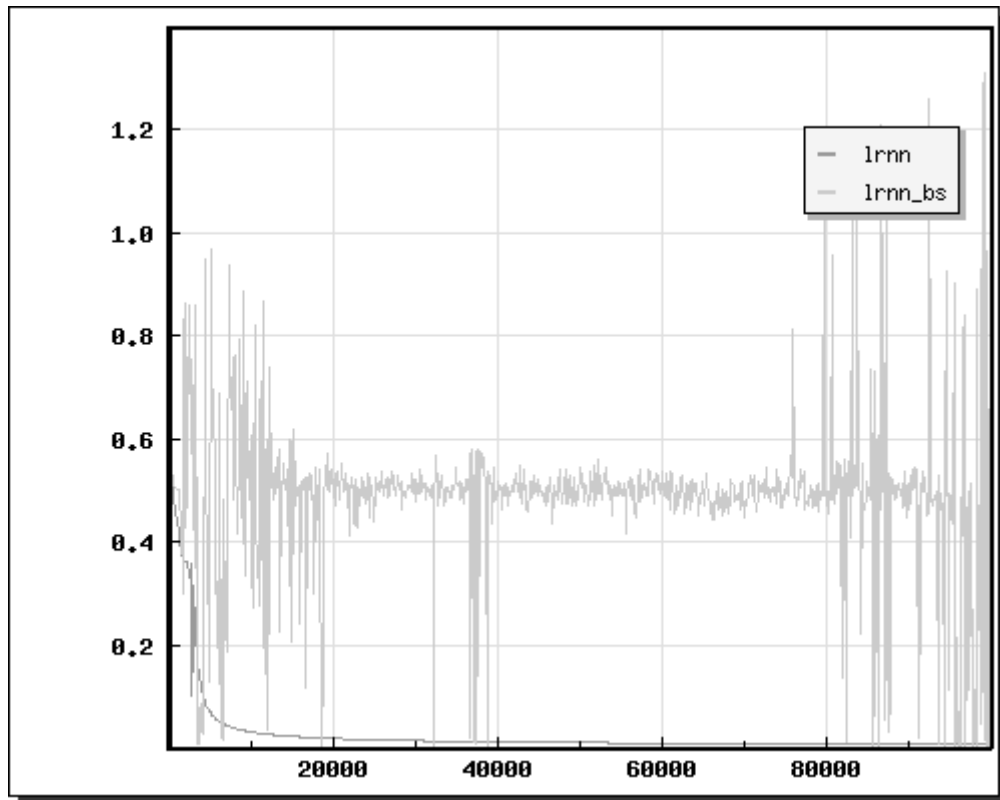
### 6.3 Comparison of the Locally Recurrent Neural Networks

In this section we will compare the performance of Multi Layer Perceptron with three different kinds of activation function, sigmoid activation functions, adaptive Catmull-Rom spline activation functions, and adaptive b-spline activation functions for Locally Recurrent Neural Networks.

#### 6.3.1. Comparison of the sigmoid activation functions and the B-spline activation function for Locally Recurrent Neural Networks

As we can see at the Figure 6.9 the B-spline activation function for Locally Recurrent Neural Networks couldn't be learned any information. It is oscillating in a

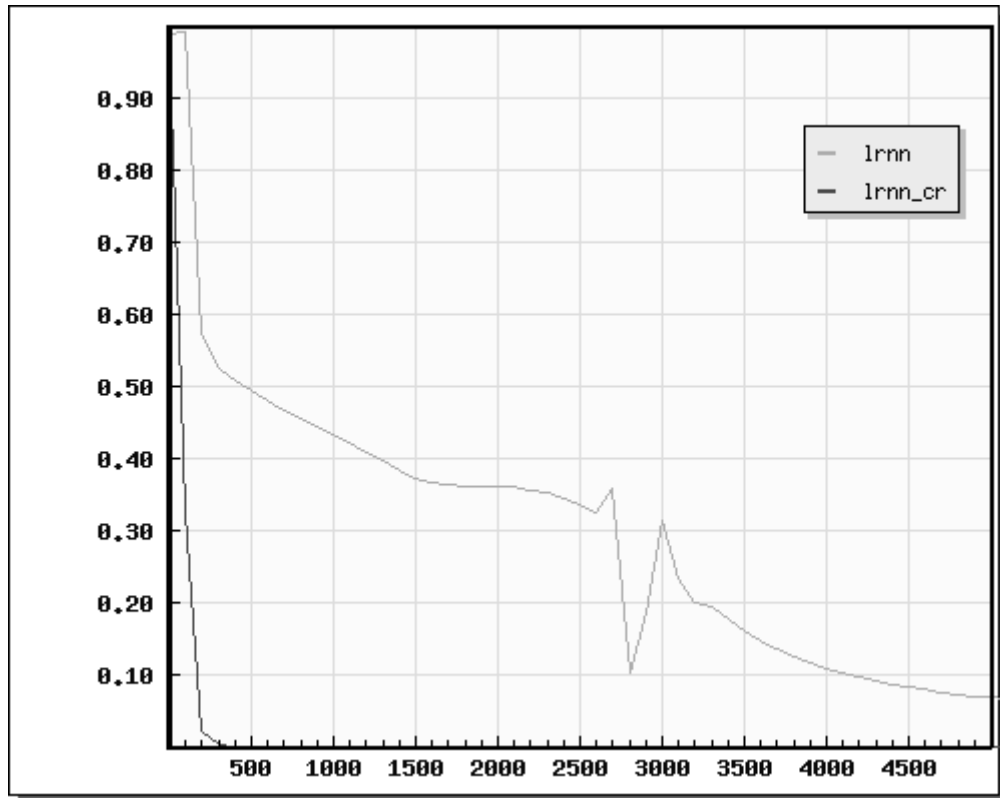
range and no proper direction. Since that, we will not compare b-spline with any activation function.



**Figure 6.9** Performances of LRNNs with sigmoid and adaptive b-spline activation functions for xor problem

### **6.3.2. Comparison of the Catmull-Rom spline activation functions and the sigmoid activation functions for Locally Recurrent Neural Networks**

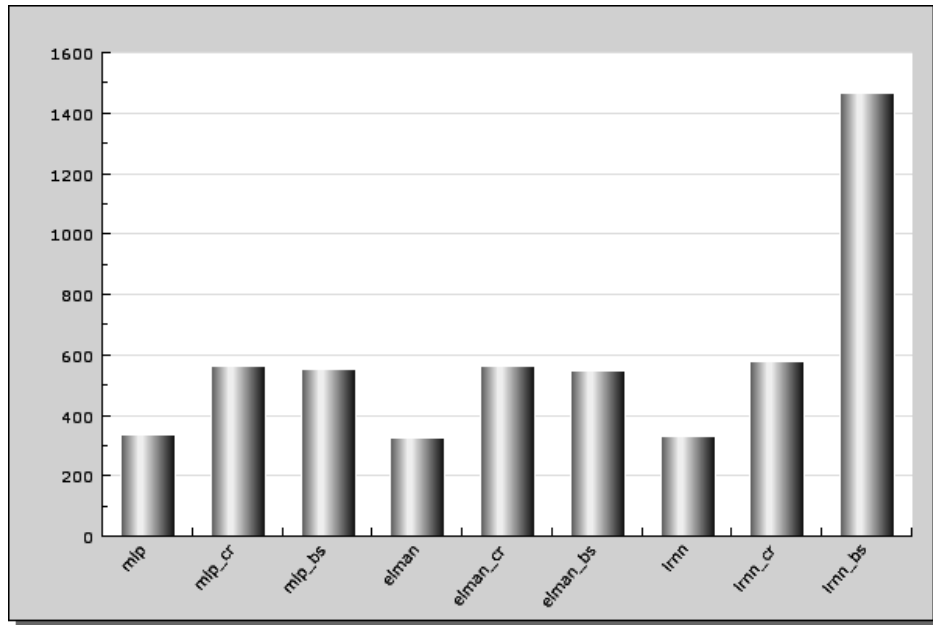
As we can see in the Figure 6.10, the CR function is better performance than the sigmoid function. This performance CR function is not decreasing while the number of epoch is increasing.



**Figure 6.10** Performances of LRNNs with sigmoid and adaptive CR spline activation functions for xor problem

#### 6.4 Comparison of execution time of all model

All the models that we compare here are different impact to performance. But we also should the execution time to evaluate these models. We can easily compare all the models by looking the graph at Figure 6.11



**Figure 6.11** Execution time graph for all the models

## **CHAPTER 7.SPLINE BASED NEURAL NETWORK SIMULATOR (SBNN)**

### **7.1 Software Specification of SBNN**

The software, we used to simulate artificial neural networks, is written in the PHP 4 web-based programming language. To use this software stand-alone, a web server and at least PHP 4 must be installed and graphic extension of the PHP 4 “gd.dll” must also be installed to see the graphs properly. We recommend Apache as a web server. If you don’t want to spend time by installation and adjustment of this software there is an alternative software “PHPTriad” which install and adjust all the software you need. You should do one more thing; open the php.ini file and remove the semicolon from the front of the “gd.dll” row. Hence, you activate the graphic extension of PHP 4. Now you can start using the program.

### **7.2 The aim of this Software**

The main reason we developed this software is to simulate the neural networks model that we used and compare the performance of this networks. There are nine different neural network models, we simulated by this software, which are MLP with sigmoid activation function, MLP with adaptive CR spline activation function, MLP with adaptive b-spline activation function, ELMAN NETWORK with sigmoid activation function, ELMAN NETWORK with adaptive CR spline activation function, ELMAN NETWORK with adaptive b-spline activation function, LOCALLY RECURRENT NEURAL NETWORKS with sigmoid activation function, LOCALLY RECURRENT NEURAL NETWORKS with adaptive CR spline activation function, and LOCALLY RECURRENT NEURAL NETWORKS with adaptive b-spline activation function. We used xor problem to measure performance of this nine models. But the software can easily be adapt to manipulate the other real world problems.

### **7.3 The Menu of the SBNN**

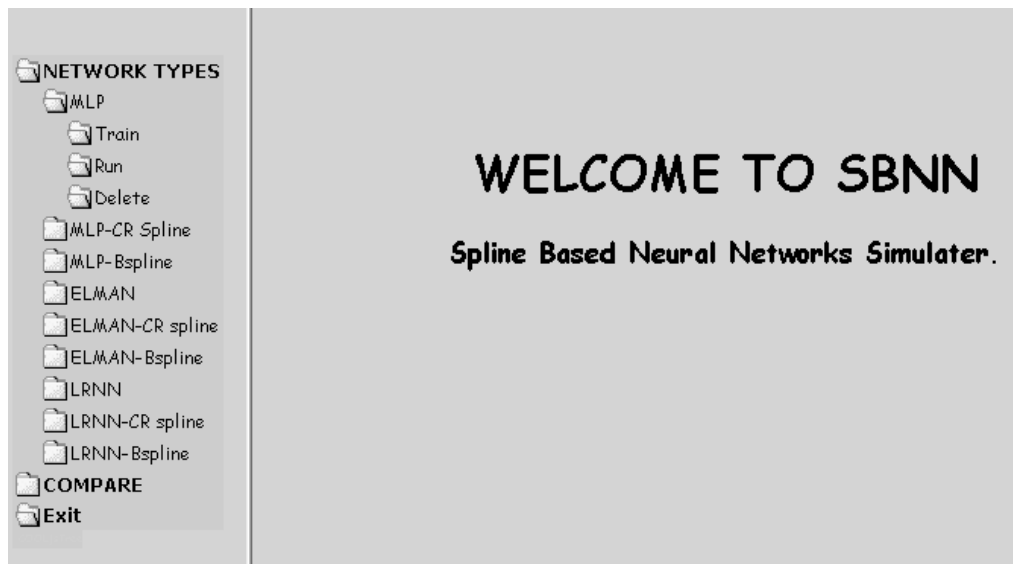
As we can see Figure 7.1, there are two main menu “NETWORKS TYPES” and “COMPARE”. There is also and “Exit” button. There are nine submenus of



“NETWORKS TYPES”, each one represents a kind of neural networks “MLP”, “MPL-CR Spline”, “MLP-BSpline”, “ELMAN”, “ELMAN -CR Spline”, “ELMAN-BSpline”, “LRNN”, “LRNN -CR Spline”, and “LRNN -BSpline”. Each submenu of “NETWORKS TYPES” is also three sub menu which are Train, Run and Delete. We can see them in the Figure 7.2.



**Figure 7.1** The main page of SBNN



**Figure 7.2** The submenu of “NETWORKS TYPES”

There two sub menu of “COMPARE”, “MSE” and “Execution Time” as shown in Figure 7.3. There is also one more button “Exit” on the menu frame. As it is guessed it closes the program.

#### 7.4 Training a network

When you want train a network for xor problem, First of all you should choose a network model ,in the submenu of “NETWORK TYPES”, then choose the “Train” button in the sub menu of the network model, you choose. You will see the table in Figure 7.4.



**Figure 7.3** The submenu of “COMPARE”

This table contains essential values to start training. You can change these values as your need you can also select the random button to select weights or you can define a constant value to the all weights as initial value. After the selection push, the train button then Network will be trained and saved automatically. It gets a certain amount of time related to the values you choose. You can also go on training later by doing it in the same way but you will not be able to change the values except the “Number of Epoch”. As in shown Figure 7.5

**Network not found. Create a new network.**

Number Of Epoch	1000	Number of Hidden Layers	1
Weight Learning Rate	0.5	Number of Neurons per Layer	10
Spline Control Point Learning Rate	0.05	MSE Period	100
Number of Control Points	80	Initial Weights : <input type="checkbox"/> Random	0.5
Distance Between Control Points	1	Network Type: Multi Layer Perceptron (Backpropogation) with Sigmoid Activation Function	<input type="button" value="Train"/>

**Figure 7.4** The initial parameter of a network to start training

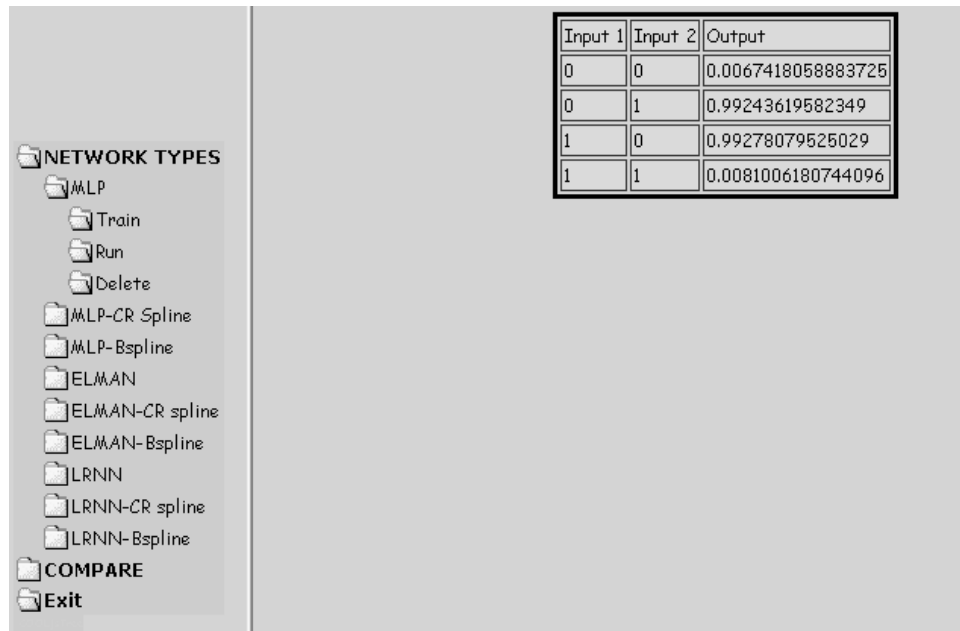
**Network found.**

Number of Epoch	1000	Number of Hidden Layers	
Weight Learning Rate		Number of Neurons per Layer	
Spline Control Point Learning Rate		Network Type: Multi Layer Perceptron (Backpropogation) with Sigmoid Activation Function	<input type="button" value="Train"/>
Number Of Control Points			
Distance Between Control Points			

**Figure 7.5** The table which shows the number of epoch to go on training

## 7.5. Running and deleting a network

To see the outputs of this trained network for the xor problem you should run this network. To run this network just pushes the “run” button. You will see the result like in Figure 7.6. If you want to delete the network, you created and trained, just push the delete button.



The screenshot shows a software interface with a menu on the left and a table of outputs on the right. The menu is titled "NETWORK TYPES" and includes options: MLP, Train, Run, Delete, MLP-CR Spline, MLP-Bspline, ELMAN, ELMAN-CR spline, ELMAN-Bspline, LRNN, LRNN-CR spline, LRNN-Bspline, COMPARE, and Exit. The table on the right displays the output for an MLP network for the XOR problem.

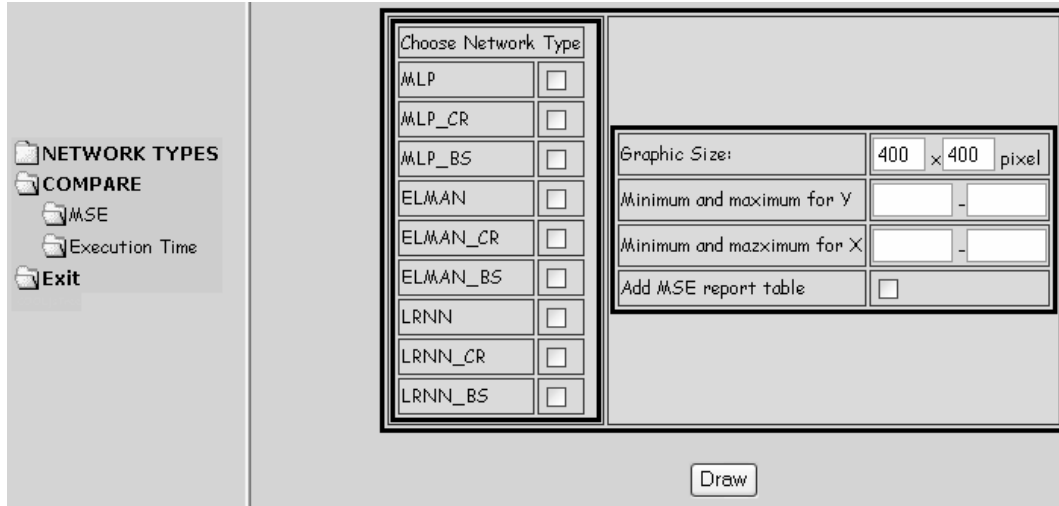
Input 1	Input 2	Output
0	0	0.0067418058883725
0	1	0.99243619582349
1	0	0.99278079525029
1	1	0.0081006180744096

**Figure 7.6** The table of outputs for MLP

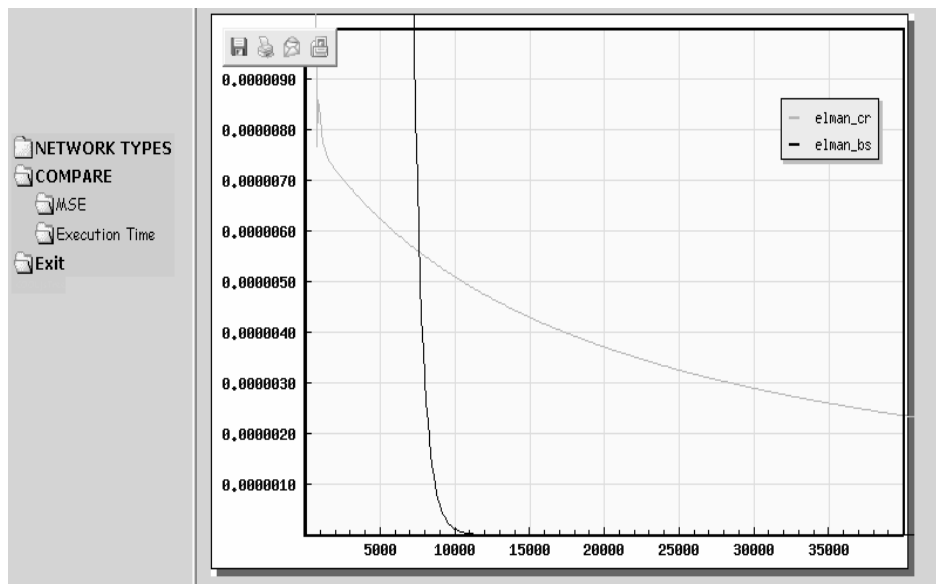
## 7.6. Comparing the Network Performance

To compare the network performance, choose “MSE, submenu of “COMPARE” and you will see the view at Figure 7.7. You can choose the networks you want to compare and the size of graph that shows cost functions together on one graph. You can also change accuracy of the graph. If you want to see the graphs in detail, you can zoom in the graphs by changing maximums and minimums for X and Y-axes. An example of comparison is shown in Figure 7.8. For this graphics,

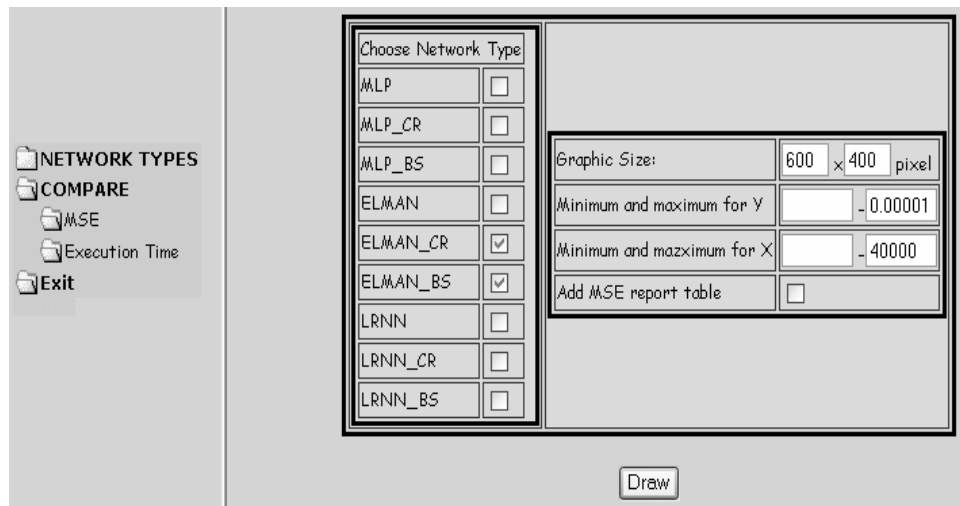
parameters are chosen as in Figure 7.9. We can also take the comparison as table by choosing the “Add MSE report table”.



**Figure 7.7** Comparison table of network models and The parameters table of the graphs, shows the graph of cost functions for the number of epoch.



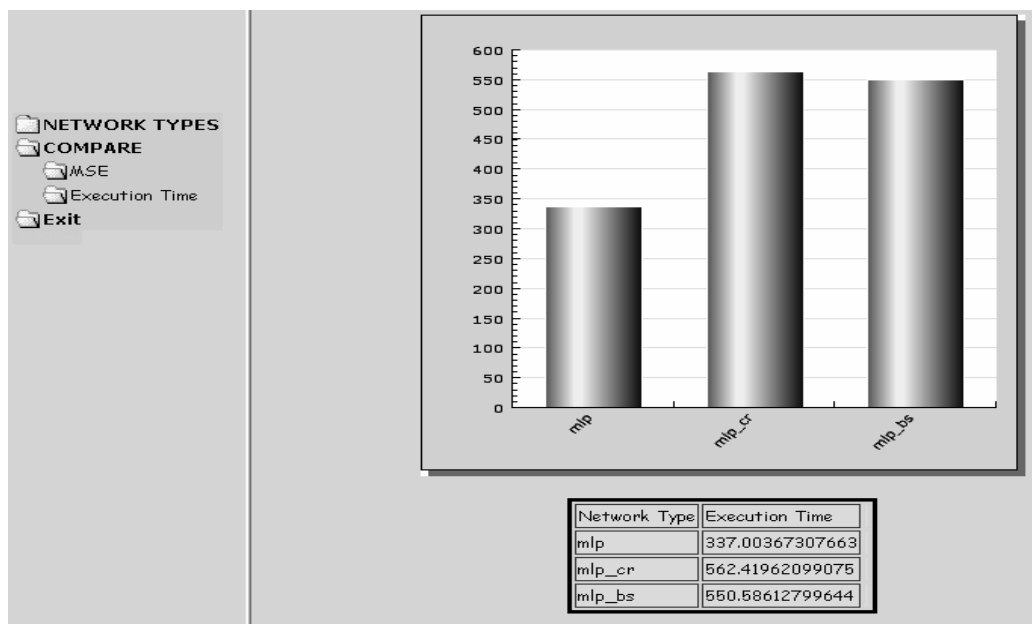
**Figure 7.8** an example of comparison graphics



**Figure 7.9** Chosen parameters for the graph, shown in Figure 7.8

### 7.7 Comparing the Execution Time Performance

If we want to compare the execution you should choose the “Execution Time” button, submenu of “COMPARE”, and choose which network types you want to compare than you will get a bar graph as in Figure 7.10. You can also get this comparison as table if you chose the “Add Execution Time report table”.



**Figure 7.10** An example of Execution Time Comparison and report table

## CHAPTER 9. CONCLUSION AND RECOMMENDATIONS FOR FUTURE WORK

In this thesis, five new neural network models are proposed: *MLP with B-spline activation function*, *ELMAN NETWORKS with CR-spline activation function*, *ELMAN NETWORKS with B-spline activation function*, *LRNN with CR-spline activation function*, and *LRNN with B-spline activation function*,

We derived mathematical explanations of these models. Then we developed web-based software *SBNN*, written in the programming language PHP 4 to simulate these five models and four other models: *MLP*, *ELMAN NETWORKS*, and *LRNN*. We use this artificial neural network simulator to compare the performance of this nine artificial neural network models on the famous xor problem.

As a future work, artificial neural network simulator, *SBNN*, can be developed for educational purposes. Since *SBNN* is a web-based program, it can be adapted for e-learning. Students can use it on the internet as an online tool.

On the other hand, these new neural networks models can be applied different kinds of real world problems and can be tested the performance of these networks. *SBNN* can be used for this purpose by a little bit developing.

## REFERENCES

- [1] Helmut A. Mayer, Roland Schwaiger, “Evolution of Cubic Spline Activation Functions for Artificial Neural”, *10th Portuguese Conference on Artificial Intelligence*, 2001
- [2] Helmut A. Mayer, Marc Strapetz, Roman Fuchs, “Simultaneous Evolution of Structure and Activation Function Types in Generalized Multi-Layer Perceptrons”, *WSES International Conference on Neural Networks and Applications*, 2001
- [3] Helmut A. Mayer, Roland Schwaiger, “Differentiation of Neuron Types by Evolving Activation Function Templates for Artificial Neural Networks”, *World Congress on Computational Intelligence, International Joint Conference on Neural Networks*, 2002.
- [4] Rumelhart D.E, Hinton G.E, Williams R.J, “Learning representations by backpropagation errors” *Nature*, vol. 323, (533-536), 1986.
- [5] Elman J.L., “Finding the structure in time”, *Cognitive Science*, Vol. 14, (179-211), 1990.
- [6] A. D. Back and A.C. Tsoi “Locally Recurrent Globally Feedforward Networks, a Critical Review of Architectures”. *IEEE Trans Pattern Analysis and Machine Intelligence*, Vol 5, No 2, (229-239), 1994.
- [7] Hearn D. ,Baker M., “Computer Graphics Second Edition ”, *Prentice Hall Press* (305-335), 1994.
- [8] Angel E., “Interactive Computer Graphics Third edition”, *Addison Wesley press* (477-526), 2003
- [9] Cybenko G., “Approximation by superposition a sigmoidal Function”, *Mathematics of CONTROL, signals, and Systems* 2(930-944) , 1989.
- [10] Öztemel E., “Yapay Sinir ağları”, *Papatya yayıncılık* (75-114), and(165-170) , 2003



- [11] “Delta Learning Rule Derivation”,  
*[http://www.eeng.dcu.ie/~ee490/ANNs/ann\\_delta\\_deriv.htm](http://www.eeng.dcu.ie/~ee490/ANNs/ann_delta_deriv.htm)*, 2005.
- [12] Guarnieri S., Piazza F., “Multilayer Feedforward Networks with Adaptive Spline Activation Function”, *IEEE Trans. Neural Networks*, Vol. 10, no: 3, 1999.
- [13] Solazzi, M. Piazza, F. Uncini, A. “Nonlinear blind source separation by spline neural Networks.” *IEEE International Conference, Acoustics, Speech, and Signal Processing*, 2001.
- [14] Mirko Solazzi and Aurelio Uncini, “Spline Neural Networks for Blind Separation of Post-Nonlinear-Linear Mixtures”, *IEEE Transactions on Circuits and Systems I Fundamental Theory and Applications*, Vol. 51, No. 4, pp 817 – 829, 2004.
- [15] Catmull E. and Rom R., “A class of local interpolating splines” *Computer-Aided Geometric Design*, R. E. Barnhill and R. F. Riesenfeld, Eds. New York: Academic, ( 317–326), 1974.
- [16] Horne B.G. and Giles C.L. “An Experimental Comparison of Recurrent Neural Networks.” *Advances in Neural Information Processing Systems*, volume 7, (697-704), Cambridge, MA, MIT Press, 1995.
- [17] “PHP programming language”, *<http://www.php.net/>*, 2005.
- [18] Minsky, M.L. , Papert, S.A. “Perceptrons”, *MIT Press* , 1969.
- [19] Rosenblatt, F. “The Perceptron: A probabilistic model for information storage and organization in the brain”, *Psychoanalytic Review*, 65, (386-408), 1958.
- [20] Widrow B. Ve Lehr M.A. ,“ 30 years of adaptive neural networks: Perceptron, Madaline, Backpropogation”, *proceeding of the IEEE*, vol 78, No 9, 1990.
- [21] Vecci, L., Piazza, F., “Learning and approximation capabilities of adaptive spline activation function neural networks.” *Neural Networks* (259-270,) 1998.
- [22] Elman, J.L , “Distributed representations, simple recurrent networks, and grammatical structure.”, *Machine Learning*”, 1991.

APPENDIX: CD containing Thesis text and software code of the SBNN.