INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17" x 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI
University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600
A nonlinear function approximator—the cooperation of nonparametric concepts and a neural network approach

Wang, Wei-Hua, Ph.D.
The Ohio State University, 1989
A Nonlinear Function Approximator
-- The Cooperation of Nonparametric Concepts
and A Neural Network Approach

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the Graduate
School of the Ohio State University

By

Wei-Hua Wang, B.S., M.S.

* * * * *

The Ohio State University

1989

Approved by

Dissertation Committee:
Richard A. Miller
Jordan Pollack
Sue Leurgans

Advisor
Department of Industrial and Systems Engineering
To My Mom and Dad
ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to Dr. Richard A. Miller, my advisor, for his guidance and insight throughout the research. Special thanks go to Dr. Jordan Pollack who constantly gave me valuable comments and insight for my dissertation and encouragement to help me pass through the tough period of my research. I would like to express my special gratitude to Dr. Sue Leurgans for her valuable comments and suggestions. I would also like to thank Dr. Alex Kirlik for his help in proof-reading my manuscripts and giving the insight comments.

Special thanks go to Mr. Cedric Sze, my manager in Computer and Control Lab., for his understanding and support. For working with him so many years, I learned not only the knowledge about computers but also the way to be an excellent boss.

I also wish to express my appreciation to all my friends for their friendship and support which made my life more joyful.

To my wife, Chao-Ping, I offer sincere thanks for her unshakable faith in me and the endurance in my delay of graduation. Her love and help make this dissertation possible. To my sister and brother, I thank them for understanding and support for so many years of taking care of our Mom and Dad for me.

Finally, I thank God to give me the chance to be what I want to be.

iii
VITA

Dec. 21, 1956 ........................................... Born - Taipei, Taiwan, R.O.C.
June 1980 ............................................ B.S., Tunghai University
                                            Taiwan, Republic of China
June 1984 ............................................ M.S., ISE Department,
                                            The Ohio State University,
                                            Columbus, Ohio
1984-Present ........................................ Graduate Administrative Associate
                                            Computer & Control Lab.,
                                            ISE Department
                                            The Ohio State University,
                                            Columbus, Ohio

FIELDS OF STUDY

Major Fields: Systems Engineering.

Studies in  Systems Modelling, Control Theory, CAD/CAM Integration,
            Neural Networks and Design of Intelligent Systems.
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS ................................................................. iii
VITA ........................................................................................................ iv
LIST OF TABLES ............................................................................. vii
LIST OF FIGURES ........................................................................... ix

CHAPTER
PAGE

I. INTRODUCTION ................................................................................. 1
   1.1 Problem Definition ...................................................................... 1
   1.2 The Approaches to the Problem ................................................. 4
   1.3 Research Objective .................................................................... 10

II. BACKGROUND AND LITERATURE REVIEW .................................. 12
   2.1 Nonparametric Function Approximation (NPFA) ....................... 13
       2.1.1 Polynomial Approximation (PA) .......................................... 19
       2.1.2 Kernel Approximation (KA) ................................................ 22
   2.2 Neural Networks (NN) .............................................................. 29
   2.3 Localized Distributed Representation: The Connection between
       Kernel Approximation and Neural Networks ............................... 50

III. UNIVARIATE MODEL .................................................................... 59
   3.1 Basic Units ................................................................................ 60
   3.2 The Structure of the Univariate Network and the Associated
       Computational Procedure ........................................................ 71
   3.3 Dynamic Learning Rate ............................................................ 78
   3.4 Experimental Results and Discussion ....................................... 86
IV. MULTIVARIATE MODEL ................................................................. 151

4.1 The Theory of Representation Functions of Several Variables by Functions of One Variable .................................................. 152

4.2 The Construction of a Multivariate Model Based on Kolmogorov Theorem and Experimental Results .................................. 158

4.2.1 The Number of Summands ..................................................... 158

4.2.2 The External Functions, G_q's .............................................. 160

4.2.3 The Internal Functions, \( \Phi_{pq} \)'s ........................................ 185

V. CONCLUDING REMARKS ............................................................. 213

5.1 Conclusion .................................................................................. 213

5.2 Future Research .......................................................................... 215

LIST OF REFERENCES ........................................................................ 218
LIST OF TABLES

TABLE PAGE

3.1. The Index of Accuracy and RMSE of $y=\cos(x)$ ............................................... 94
3.2. The Index of Accuracy and RMSE of $y=\sin(x)-\cos(x^2)$ ................................. 99
3.3. The Index of Accuracy and RMSE of $y=0.8x+1.5$ ........................................ 105
3.4. The Index of Accuracy and RMSE of Ramp Function ................................... 110
3.5. The Index of Accuracy and RMSE of Multi-ramp Function ............................ 115
3.6. The Index of Accuracy and RMSE of Triangle Function ................................. 120
3.7. The Index of Accuracy and RMSE of Step Function ..................................... 125
3.8. The Index of Accuracy and RMSE of Chaotic Function .............................. 130
3.9. The Index of Accuracy and RMSE of $y=\cos(x)$ via Dynamic Learning ... 139
3.10. The Index of Accuracy and RMSE of $y=\sin(x)-\cos(x^2)$ via Dynamic Learning ................................................................. 142
3.11. The Index of Accuracy and RMSE of $y=0.8x+1.5$ via Dynamic Learning ................................................................. 145
3.12. The Index of Accuracy and RMSE of Ramp Function via Dynamic Learning ................................................................. 148
4.1. The Index of Accuracy and RMSE of Case 1 ................................................ 170
4.2. The Index of Accuracy and RMSE of Case 2 ................................................ 173
4.3. The Index of Accuracy and RMSE of Case 3 ................................................ 176
4.4. The Index of Accuracy and RMSE of Case 4 ................................................ 179
4.5. The Index of Accuracy and RMSE of Case 5 ................................................ 182
4.6. The Index of Accuracy and RMSE After 100 Trials ..................................... 185
4.7. $z=0.2xy+1.2\sin(x+y)$, The Index of Accuracy and RMSE of Approach 1 ................................................................. 196
4.8. $z=\sin(x)+1.2\cos(y^2)$, The Index of Accuracy and RMSE of Approach 1 ................................................................. 197
<table>
<thead>
<tr>
<th>TABLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.9. $z=0.2xy+1.2\sin(x+y)$, The Index of Accuracy and RMSE of Approach 2</td>
<td>203</td>
</tr>
<tr>
<td>4.10. $z=\sin(x)+1.2\cos(y^2)$, The Index of Accuracy and RMSE of Approach 2</td>
<td>204</td>
</tr>
<tr>
<td>4.11. $z=0.2xy+1.2\sin(x+y)$, The Index of Accuracy and RMSE of Approach 3</td>
<td>209</td>
</tr>
<tr>
<td>4.12. $z=\sin(x)+1.2\cos(y^2)$, The Index of Accuracy and RMSE of Approach 3</td>
<td>210</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1. Black Box</td>
<td>3</td>
</tr>
<tr>
<td>1.2. Global Representation Scheme</td>
<td>4</td>
</tr>
<tr>
<td>1.3. Local Representation Scheme</td>
<td>5</td>
</tr>
<tr>
<td>1.4. Function Approximation Scheme</td>
<td>7</td>
</tr>
<tr>
<td>2.1. Geometric View of a Best Approximation</td>
<td>15</td>
</tr>
<tr>
<td>2.2. The Behavior of $x^p$ for $p=0.5$, $p=1$, $p=2$, $p=500$</td>
<td>16</td>
</tr>
<tr>
<td>2.3. A Typical Error Curve</td>
<td>16</td>
</tr>
<tr>
<td>2.4. Examples of Kernel Functions</td>
<td>23</td>
</tr>
<tr>
<td>2.5. Neural Network</td>
<td>29</td>
</tr>
<tr>
<td>2.6. Perceptron and Adaline</td>
<td>33</td>
</tr>
<tr>
<td>2.7. XOR (eXclusive OR)</td>
<td>33</td>
</tr>
<tr>
<td>2.8. Perceptron and Adaline with Hidden Layer</td>
<td>34</td>
</tr>
<tr>
<td>2.9. Hopfield Network</td>
<td>36</td>
</tr>
<tr>
<td>2.10. Multi-layer Perceptron Network</td>
<td>38</td>
</tr>
<tr>
<td>2.11. Processing Units</td>
<td>40</td>
</tr>
<tr>
<td>2.12. Three Different Kinds of Units</td>
<td>41</td>
</tr>
<tr>
<td>2.13. Lapedes and Farber's Network</td>
<td>54</td>
</tr>
<tr>
<td>2.14. Moody and Darken's Network</td>
<td>56</td>
</tr>
<tr>
<td>3.1. Logistic Function, $\theta=1$, $w=1$</td>
<td>63</td>
</tr>
<tr>
<td>3.2. $\theta_1/w_1 &lt; \theta_2/w_2$</td>
<td>64</td>
</tr>
<tr>
<td>3.3. $\theta_1/w_1 &gt; \theta_2/w_2$</td>
<td>64</td>
</tr>
<tr>
<td>3.4. $w_1&lt;0$, $w_2&gt;0$ or $w_1&gt;0$, $w_2&lt;0$</td>
<td>65</td>
</tr>
<tr>
<td>3.5. $w_1, w_2&lt;0$</td>
<td>66</td>
</tr>
<tr>
<td>3.6. Impulse function away from origin</td>
<td>67</td>
</tr>
<tr>
<td>3.7. Linear Node</td>
<td>68</td>
</tr>
<tr>
<td>FIGURE</td>
<td>PAGE</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>3.8. Nonlinear Node</td>
<td>69</td>
</tr>
<tr>
<td>3.9. Basic Unit</td>
<td>70</td>
</tr>
<tr>
<td>3.10. Univariate (SISO) Model</td>
<td>71</td>
</tr>
<tr>
<td>3.11. $y = \cos(x)$, Index of Accuracy vs. Trials</td>
<td>94</td>
</tr>
<tr>
<td>3.12. $y = \cos(x)$ at trial 5</td>
<td>95</td>
</tr>
<tr>
<td>3.13. $y = \cos(x)$ at trial 10</td>
<td>96</td>
</tr>
<tr>
<td>3.14. $y = \cos(x)$ at trial 50</td>
<td>97</td>
</tr>
<tr>
<td>3.15. $y = \cos(x)$ at trial 100</td>
<td>98</td>
</tr>
<tr>
<td>3.16. $y = \sin(x) - \cos(x^2)$, Index of Accuracy vs. Trials</td>
<td>99</td>
</tr>
<tr>
<td>3.17. $y = \sin(x) - \cos(x^2)$ at trial 5</td>
<td>100</td>
</tr>
<tr>
<td>3.18. $y = \sin(x) - \cos(x^2)$ at trial 10</td>
<td>102</td>
</tr>
<tr>
<td>3.19. $y = \sin(x) - \cos(x^2)$ at trial 50</td>
<td>103</td>
</tr>
<tr>
<td>3.20. $y = \sin(x) - \cos(x^2)$ at trial 100</td>
<td>104</td>
</tr>
<tr>
<td>3.21. $y = 0.8x + 1.5$, Index of Accuracy vs. Trials</td>
<td>105</td>
</tr>
<tr>
<td>3.22. $y = 0.8x + 1.5$ at trial 5</td>
<td>106</td>
</tr>
<tr>
<td>3.23. $y = 0.8x + 1.5$ at trial 10</td>
<td>107</td>
</tr>
<tr>
<td>3.24. $y = 0.8x + 1.5$ at trial 50</td>
<td>108</td>
</tr>
<tr>
<td>3.25. $y = 0.8x + 1.5$ at trial 100</td>
<td>109</td>
</tr>
<tr>
<td>3.26. Ramp function, Index of Accuracy vs. Trials</td>
<td>110</td>
</tr>
<tr>
<td>3.27. Ramp function at trial 5</td>
<td>111</td>
</tr>
<tr>
<td>3.28. Ramp function at trial 10</td>
<td>112</td>
</tr>
<tr>
<td>3.29. Ramp function at trial 50</td>
<td>113</td>
</tr>
<tr>
<td>3.30. Ramp function at trial 100</td>
<td>114</td>
</tr>
<tr>
<td>3.31. Multi-ramp function, Index of Accuracy vs. Trials</td>
<td>115</td>
</tr>
<tr>
<td>3.32. Multi-ramp function at trial 5</td>
<td>116</td>
</tr>
<tr>
<td>3.33. Multi-ramp function at trial 10</td>
<td>117</td>
</tr>
<tr>
<td>3.34. Multi-ramp function at trial 50</td>
<td>118</td>
</tr>
<tr>
<td>3.35. Multi-ramp function at trial 100</td>
<td>119</td>
</tr>
<tr>
<td>3.36. Triangle function, Index of Accuracy vs. Trials</td>
<td>120</td>
</tr>
<tr>
<td>FIGURE</td>
<td>PAGE</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>3.37. Triangle function at trial 5</td>
<td>121</td>
</tr>
<tr>
<td>3.38. Triangle function at trial 10</td>
<td>122</td>
</tr>
<tr>
<td>3.39. Triangle function at trial 50</td>
<td>123</td>
</tr>
<tr>
<td>3.40. Triangle function at trial 100</td>
<td>124</td>
</tr>
<tr>
<td>3.41. Step function, Index of Accuracy vs. Trials</td>
<td>125</td>
</tr>
<tr>
<td>3.42. Step function at trial 5</td>
<td>126</td>
</tr>
<tr>
<td>3.43. Step function at trial 10</td>
<td>127</td>
</tr>
<tr>
<td>3.44. Step function at trial 50</td>
<td>128</td>
</tr>
<tr>
<td>3.45. Step function at trial 100</td>
<td>129</td>
</tr>
<tr>
<td>3.46. Chaotic function, Index of Accuracy vs. Trials</td>
<td>130</td>
</tr>
<tr>
<td>3.47. Chaotic function at trial 5</td>
<td>131</td>
</tr>
<tr>
<td>3.48. Chaotic function at trial 100</td>
<td>135</td>
</tr>
<tr>
<td>3.49. $y = \cos(x)$, Index of Accuracy vs. Training points</td>
<td>139</td>
</tr>
<tr>
<td>3.50. $y = \cos(x)$ (a)after 200 points, (b)after 500 points, (c)after 1000 points, (d)after 5000 points</td>
<td>140</td>
</tr>
<tr>
<td>3.51. $y = \sin(x) - \cos(x^2)$, Index of Accuracy vs. Training points</td>
<td>142</td>
</tr>
<tr>
<td>3.52. $y = \sin(x) - \cos(x^2)$ (a)after 200 points, (b)after 500 points, (c)after 1000 points, (d)after 5000 points</td>
<td>143</td>
</tr>
<tr>
<td>3.53. $y = 0.8x + 1.5$, Index of Accuracy vs. Training points</td>
<td>145</td>
</tr>
<tr>
<td>3.54. $y = 0.8x + 1.5$ (a)after 200 points, (b)after 500 points, (c)after 1000 points, (d)after 5000 points</td>
<td>146</td>
</tr>
<tr>
<td>3.55. Ramp function Index of Accuracy vs. Training points</td>
<td>148</td>
</tr>
<tr>
<td>3.56. Ramp function (a)after 200 points, (b)after 500 points, (c)after 1000 points, (d)after 5000 points</td>
<td>149</td>
</tr>
<tr>
<td>4.1. Kolmogorov's Theorem</td>
<td>155</td>
</tr>
<tr>
<td>4.2. Lorentz's Modification</td>
<td>157</td>
</tr>
<tr>
<td>4.3. Sprenger's Modification</td>
<td>159</td>
</tr>
<tr>
<td>4.4. External Function</td>
<td>161</td>
</tr>
<tr>
<td>4.5. A Multivariate Model</td>
<td>163</td>
</tr>
<tr>
<td>FIGURE</td>
<td>PAGE</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>4.6. A Modified Multivariate Model</td>
<td>164</td>
</tr>
<tr>
<td>4.7. Weights Definition</td>
<td>165</td>
</tr>
<tr>
<td>4.8. Quadratic on Rectangle, Index of Accuracy vs. Trials</td>
<td>170</td>
</tr>
<tr>
<td>4.9. Quadratic on Rectangle, (a) actual function, (b) after 5 trials, (c) after 50 trials, (d) after 100 trials</td>
<td>171</td>
</tr>
<tr>
<td>4.10. $z=0.2e^x + \sin(y^2)$, Index of Accuracy vs. Trials</td>
<td>173</td>
</tr>
<tr>
<td>4.11. $z=0.2e^x + \sin(y^2)$, (a) actual function, (b) after 5 trials, (c) after 50 trials, (d) after 100 trials</td>
<td>174</td>
</tr>
<tr>
<td>4.12. $z=0.2x + 1.2\sin(y^2) + 0.3e^0.7x + 1.2\tanh(5y)$, Index vs. Trials</td>
<td>176</td>
</tr>
<tr>
<td>4.13. $z=0.2x + 1.2\sin(y^2) + 0.3e^0.7x + 1.2\tanh(5y)$, (a) actual function, (b) after 5 trials, (c) after 50 trials, (d) after 100 trials</td>
<td>177</td>
</tr>
<tr>
<td>4.14. $z=\sin(x) + 1.2\cos(y^2)$, Index of Accuracy vs. Trials</td>
<td>179</td>
</tr>
<tr>
<td>4.15. $z=\sin(x) + 1.2\cos(y^2)$, (a) actual function, (b) after 5 trials, (c) after 50 trials, (d) after 100 trials</td>
<td>180</td>
</tr>
<tr>
<td>4.16. $z=0.8x - 0.4y$, Index of Accuracy vs. Trials</td>
<td>182</td>
</tr>
<tr>
<td>4.17. $z=0.8x - 0.4y$, (a) actual function, (b) after 5 trials, (c) after 50 trials, (d) after 100 trials</td>
<td>183</td>
</tr>
<tr>
<td>4.18. Mapping Relationship Among $f$, $G_q$'s and $\Phi_{pq}$'s</td>
<td>186</td>
</tr>
<tr>
<td>4.19. $A_r^q$ Segments</td>
<td>189</td>
</tr>
<tr>
<td>4.20. $z=0.2xy + 1.2\sin(x+y)$, The Index of Accuracy vs. Trials of Approach 1</td>
<td>196</td>
</tr>
<tr>
<td>4.21. $z=\sin(x) + 1.2\cos(y^2)$, The Index of Accuracy vs. Trials of Approach 1</td>
<td>197</td>
</tr>
<tr>
<td>4.22. $\Phi_{pq}$'s Networks via Approach 2</td>
<td>199</td>
</tr>
<tr>
<td>4.23. $z=0.2xy + 1.2\sin(x+y)$, The Index of Accuracy vs. Trials of Approach 2</td>
<td>203</td>
</tr>
<tr>
<td>4.24. $z=\sin(x) + 1.2\cos(y^2)$, The Index of Accuracy vs. Trials of Approach 2</td>
<td>204</td>
</tr>
<tr>
<td>FIGURE</td>
<td>PAGE</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>4.25. Φpq's Networks via Approach 3</td>
<td>206</td>
</tr>
<tr>
<td>4.26. $z=0.2xy+1.2\sin(x+y)$, The Index of Accuracy vs. Trials of Approach 3</td>
<td>209</td>
</tr>
<tr>
<td>4.27. $z=\sin(x)+1.2\cos(y^2)$, The Index of Accuracy vs. Trials of Approach 3</td>
<td>210</td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION

1.1. PROBLEM DEFINITION

Modern control theory has had tremendous success in areas where a system is well understood and defined. However, it fails to cope with many complex processes. Some of the fundamental difficulties are rooted in the structural complexity of the controlled system and the complicated interaction between the controlled system and the environment. In order to make a control algorithm work effectively, a good model is crucial. One of the most important criteria in deciding whether a model is "good" or "bad" is the precision with which the model reproduces the behavior of the real system.

Generally speaking, there are four steps involved in the modelling process. The first step is to determine the important variables such as inputs, outputs, and control variables. The second step is to analyze the dimensionality of the model according to the relationship among the important variables and the system's behavior. The third step is to determine the structure of the model. The last step is to determine the value of each parameter in order to complete the modelling process.
Most of the time, the criteria for selecting the input and output variables are quite subjective. The selection is usually based on the designer's experience with the target system. Recently, many efforts have been made to analyze the dimensionality of the system's representation (Crutchfield, et al., 1987; Froehling, et al., 1981, Wolf, et al., 1985). Basically, dimensionality analysis focuses on determining the amount of historical data which should be considered to construct the model for reproducing the system's behavior. Some methodologies have been developed to resolve this problem, but most of them require a large amount of historical data which makes the computation very inefficient. Hence, real time control becomes even more difficult. In this research, we focus on the development of a new method to release the constraints which are associated with the stated third and the fourth steps. In another words, we intend to develop a method which can construct a model to reproduce the system's behavior without requiring prior information about the system's structure. The traditional function approximation assumption is adopted in this research, i.e. a static and deterministic system is assumed.

Most traditional data analysis methods are based on a given model which contains the structure information of the system of interest. This information is given at the very beginning of the analysis. Thus, the original modelling problem becomes finding the values of the parameters which are associated with the given model. Some very powerful methods, such as the least-squares method, have been developed to solve
this problem. If the structure of the system is known, this is a very reasonable and powerful way to find a mathematical representation of the real system's behavior. However, if the structure of the real system is not known \textit{a priori}, it will be unreasonable and dangerous to assume a particular model structure since this may cause the problem of "ill-formedness". It means that no matter how good the analysis method is, it will never construct an acceptable mathematical representation of the real system's behavior. Therefore, the development of a method which does not require the prior structure assumption is a very important research question.

The problem defined above is not new to systems engineers. The black box problem has been with us for hundreds of years.

![Fig. 1.1. Black Box.](image.png)

In this research, we are looking at a certain class of black boxes, the static systems. The input is $I = (i_1, i_2, \ldots, i_n)$, which is defined in a set of closed intervals $\{ [a_i, b_i] \mid a_i \in \mathbb{R}, b_i \in \mathbb{R}, i = 1, 2, \ldots, n \}$ and the output is $O$, which is defined in $\mathbb{R}$. The relationship between input $I$ and output $O$ could then be stated as $O = f(I)$. Hence, the problem now is to find an $f$ from the behavior data without having some prior structural information about the observed system.
1.2. THE APPROACHES TO THE PROBLEM

Barto (Barto, 1987) has depicted two different methods for constructing a model. The first one is the computational method which is preferred by the orthodox control theorists. The causal relations among the inputs, the states, and the outputs are described by using a mathematical representation form, the so-called global representation scheme or the "paper and pencil" form.

\[ O = F(I) \]
\[ O = al^2 + bi + c \]

Fig. 1.2. Global Representation Scheme.

This method has advantages, such as the generalizability and economy in memory space (in terms of computer implementation). It also has some disadvantages, which are listed below.

(1) It is difficult to adapt to changes.

Since the behavior of the controller is constructed in a rigid form, a lot of effort is required when changes occur. This is mainly due to the design of the behavior of the controller which is based on all of the available behavior data of the controlled system. Therefore, any change of the form of the model has to accommodate all available data.

(2) It takes longer computation time than the table-lookup method, which will be described next.
Given the input value, some computation is needed to calculate the corresponding output value.

(3) It depends highly on the prior information about the system, i.e. this method is always applied to a well-defined problem. In this conventional method, the explicit structure information about the controlled system is required, but this information is hard to get in the real world. Most of the time, the available information is a set of behavior data rather than some highly compiled structural information.

The second method is the table-lookup method. The relation between the inputs and the outputs are described explicitly in the form of a table or production rules.

As long as the inputs are defined in a table entry or they meet the condition of the production rules, the corresponding outputs can be obtained immediately. This method has the advantages of saving time and being easily modifiable. Since every entry is independent of the others, we can easily change the value of one entry without affecting the others. Thus, some very complex relations, such as those nonlinear ones,
can be described by this method. However, the table-lookup method suffers from the following disadvantages.

(1) It does not have the capability of generalization.
   Every relation has to be described explicitly in this method.

(2) It takes a lot of memory space.
   Because every relation has to be described explicitly, it requires a lot of computer memory space.

(3) It depends highly on the design stage.
   In order to design a controller with acceptable performance, the input/output relation table has to be constructed in the design stage. Sufficient information about the system must be provided in this stage.

According to the above discussion, it seems that these two methods can compensate for each other's shortcomings. This is true except for the situation that both of them are highly dependent on the design stage. The performance of these two methods will be constant after the design is completed.

However, there is another method which could be used to construct a model. It is called the function approximation method. This method uses some known functions to approximate the unknown ones, and sometimes is referred to as the nonparametric method (Eubank, 1988). No prior structure information about the approximated function is required.
Both linear and nonlinear approaches are used in the function approximation method. In the linear approach, the parameters are the coefficients of the basic functions, $\Phi_i(I)$'s. Given $L(A, I)$ is a linear parametric family in $A$, it can be written as

$$L(A, I) = \sum_{i=1}^{n} a_i \Phi_i(I) \tag{1.2.1}$$

where $A = (a_1, a_2, \ldots, a_n)$ is a parameter vector. Polynomial approximation and Fourier series are linear approaches, since their mathematical representations are infinite sums of some basic functions.

On the other hand, in the nonlinear approach, the parameters include the coefficients of the basic functions $\Phi_i$'s and a parameter vector embedded in the $\Phi_i$'s. Given $NL(A, B, I)$ is a parametric family in $A$ and $B$, it can be written as

$$NL(A, B, I) = \sum_{i=1}^{n} a_i \Phi_i(B, I) \tag{1.2.2}$$

where $A$ and $B$ are parameter vectors. Exponential approximation,

$$F(A, T, x) = \left\{ \sum_{i=1}^{n} a_i e^{t_i x_i} |a_i| < \infty, |t_i| < \infty \right\},$$

is a typical example.
It is well-known that the polynomial approximation can be constructed as the best approximation (Rice, 1964) in the sense of uniform norm. Furthermore, in the finite domain case, we can construct an interpolation instead of an approximation, i.e., the approximating function coincides with the approximated function at the given finite points. However, polynomial approximation has some problems. One of them is that the behavior in any small region determines the behavior everywhere. There are some possible solutions to this problem.

One solution is to generalize the class of polynomial approximating functions by letting the functions be piecewise polynomial of a predetermined degree (n) and blending them together with some required properties, such as $C^{n-1}$ (denotes the existence of the $(n-1)^{th}$ derivative) at each connecting knot. These functions are splines. In a linear version of this method we have to determine the knots in advance. This requires prior identification of the location of these knots before we can construct the approximation. Most of the time, it is not likely that we will have this information a priori. Currently, some free knots methods are under development. These methods are nonlinear.

Some other methods have been developed based on a similar principle. One of them is the kernel approximation method. Many types of kernels, such as the Dirichlet kernel and the Fejer kernel, for example, have been used as the approximating functions. All of them are symmetric and positive (except for the Dirichlet kernel, which is
symmetric but not positive everywhere). Furthermore, most of the kernel approximation methods are linear methods.

Theoretically speaking, we need an infinite number of terms in both the linear and the nonlinear approximation methods, \( n \to \infty \) in (1.2.1) and (1.2.2), to approximate the actual function within any specified error bound. This is infeasible in practice consideration. Therefore, we have to fix the number of the terms and try to find a best approximation given this constraint. It is expected that a richer kernel, i.e., not necessarily symmetric or positive, will be more flexible in approximating the unknown function. That is, a richer kernel will use fewer terms to obtain a better approximation or will reach the same quality of approximation faster. In this research, we construct a richer kernel which is shown to be very effective in performing the approximation task and we proved that this class of kernel is dense on \([a,b]\), \(a < b, a, b \in \mathbb{R}\). Hence, every continuous function defined on \([a,b]\) could be approximated by using this particular kernel.

However, most nonlinear nonparametric approximation methods are difficult to implement computationally, especially for the multivariate cases. This is so because the solutions are hard to derive analytically, because the methods themselves are very difficult to adapt to the multivariate situation, or because it is time consuming to compute the solutions. One solution to the first problem is to develop a search method. One solution to the second problem is to develop a more general method which is less constrained by the numbers of variables.
One solution to the third problem is to take advantage of the current massively parallel computing architecture.

A new method which adopts the neural network/parallel distributed processing (PDP) (Rumelhart et al, 1986; McClelland et al, 1986) approach and kernel approximation will be developed in this research. Because of the flexibility of the relation among units in the network structure, it is possible to "train" a network to have a behavior similar to the real system, i.e. to perform as a model of the real system. Since the designed network structure follows the distributed representation scheme, that is, one entity is represented by many units and one unit has contribution to many entities, the developed method does have the capabilities of generalization. Since the network can tune itself via the observed information, it possesses learning capability. That is to say, the performance of the system is a function of time instead of being a constant which is determined in the design stage.

1.3. RESEARCH OBJECTIVE AND THE ORGANIZATION

The objective of this research is to develop an on-line nonparametric method which can tune a neural network to perform as a nonlinear function approximator.

This research can be divided into three parts. The first part is an examination of the theory behind nonparametric function approximation, especially kernel approximation, and to examine the relationship between the kernel function approach and neural networks.
This task is described in Chapter II. The second part is the construction of a neural network which can perform as a univariate function approximator. This task will be described in Chapter III. In the last part of this research, we look into the mathematical theory of the representation of functions of several variables by functions of one variable which extends the method constructed in the previous chapters to multivariate case. The description and the experimental results of a multivariate model will be described in Chapter IV. Conclusions and recommendations for future research will be described in the final chapter.
CHAPTER II
BACKGROUND AND LITERATURE REVIEW

As stated in the previous chapter, one possible way to represent a system's input/output relation or to model a system's behavior is to use the function approximation approach. That is, the relationship between input and output can be represented by the following form:

\[ O = \sum_{j=1}^{n} g_j(1) \]

This approach is referred as nonparametric function approximation (NPFA) in which it is not necessary to have the system's structural information in advance in order to construct the representation. We can identify parameters of some known functions, which belong to a well-defined parametric family, to approximate the unknown function or the input/output behavior of the target system. Since nonparametric function approximation has a very long and interesting history, we will go over some fundamental results of this field, including polynomial approximation, in the first section. The shortcomings of polynomial approximation will be discussed briefly in the first section, and the reason why kernel approximation is a good alternative approach will also be discussed.
Generally speaking, there are some difficulties in applying theoretical results to describe the behavior of real systems. This is also true in kernel approximation research. And it is even more difficult if we want to construct a kernel function approximation under a finite memory constraint and in a real time environment. The neural network approach, however, offers a possible solution to these problems. In the second section of this chapter, we review the historical background of neural networks research and the error backward propagation method which will be used in this research. The connection between kernel approximation and neural networks is given at the end of this chapter.

2.1. NONPARAMETRIC FUNCTION APPROXIMATION (NPFA)

The subject of this section is to approximate a real continuous function \( f(x) \) by an approximating function \( F(A, x) \) with a fixed finite number of parameters, where \( A \) represents the parameters of the approximating function \( F \). Generally, there are two important parts of this procedure. The first part is determining the type of approximating function that is used. The second part is determining how one measures the "goodness" (Rice, 1964) or the "closeness" (Davis, 1975) of an approximation. In this section, we emphasize two different types of approximating functions: polynomial functions and kernel functions. Before we discuss them, it is appropriate to give some definitions which are relevant to the following discussion and to the formal definition of the approximation problem.
There are certain generalities about nonparametric function approximation theory (hereafter referred to as function approximation). A natural description of these general results is in terms of a normed linear space. The definition of the normed linear space is as follows.

**Definition 2.1.1 (Normed Linear Space)** A linear space $X$ is called a normed linear space if for each element $x$ of the space there is defined a real number designated by $\|x\|$ with the following properties:

1. $\|x\| \geq 0$ (positivity)
2. $\|x\| = 0$ if and only if $x=0$ (definiteness)
3. $\|\alpha x\| = |\alpha| \|x\|$ for every scalar $\alpha$ (homogeneity)
4. $\|x+y\| \leq \|x\| + \|y\|$ (triangle inequality)

The quantity $\|x\|$ is known as the norm of $x$.

The norm gives us a notion of distance in $X$, i.e. if $x,y \in X$, then the distance from $x$ to $y$ (or $y$ to $x$) is $\|x-y\|$.

Based on the above definition, the approximation problem could be defined as follows.

**Approximation Problem** (Rice 1964)

Let $f(x)$ be a given real-valued continuous function defined on a set $X$, and let $F(A,x)$ be a real-valued approximating function depending continuously on $x \in X$ and on $n$ parameters $A$. Given the distance function or norm $\rho$, determine the parameters $A^* \in P$ such that

$$\rho[F(A^*,x),f(x)] \leq \rho[F(A,x),f(x)] \quad \text{for all } A \in P$$
A solution to this problem is called as a best approximation, corresponding to norm $p$. The problem of finding a best approximation can be pictured geometrically. Given a linear space $X$, and a set of linearly independent elements $\{w_1, w_2, \ldots, w_n\}$ which span a subspace $W$, we are to find a $w \in W$ whose distance from $x$ is least; that is, find $w^* \in W$ such that $||x-w||$ is least for $w = w^*$ (Fig. 2.1). Such a $w^*$ is called a best approximation to $x$ in $W$.

![Fig. 2.1. Geometric View of A Best Approximation](image)

The first step to approximating a function $f(x)$ is to translate the practical problem into a mathematical form. This means that one must choose the approximating function and the distance function. Generally, the distance function can be defined in the following sense.

**Definition 2.1.2 (L$_p$-norm)** The L$_p$-norm of the function $f(x)$ is denoted by $L_p(f)$, and is defined by

$$L_p(f) = \left[ \int |f(x)|^p \, dx \right]^{1/p}, \quad p > 0$$
There are several cases to be distinguished, including (a) $p < 1$, (b) $p = 1$, (c) $p = 2$, and (d) $p = \infty$. The function $x^p$ for different values of $p$ is shown in Fig. 2.2. All of these functions are monotonic, that is, if $|x| \leq |y|$, then $|x|^p \leq |y|^p$.

Given a function $f(x)$ and an approximating function $F(A, x)$ to $f(x)$, Fig. 2.3 (Rice, 1964) shows a typical error curve $F(A, x) - f(x)$.

**Fig. 2.2.** The Behavior of $x^p$ for $p = 0.5$, $p = 1$, $p = 2$, and $p = 500$.

**Fig. 2.3.** A Typical Error Curve
If $F(A, x)$ is not a best approximation we can vary $A$ to decrease $L_p[F(A, x) - f(x)]$.

This could be seen heuristically by considering the effect of a small variation $\delta(x)$ in the error curve. We have

$$\int_0^1 |F(A, x) - f(x) + \delta(x)|^p \, dx = \int_0^1 |F(A, x) - f(x)|^p \, dx + p \int_0^1 |F(A, x) - f(x)|^{p-1} \delta(x) \, dx \quad (2.1.1)$$

Now, let us state the different values of $p$ of $L_p$-norm in the following:

(a) $p < 1$

As we can see from (2.1.1), the integrand $|F(A, x) - f(x)|^{p-1} \delta(x)$ becomes infinite at the zeros of the error curve. Thus, we have a situation that a best approximation is the one which makes as much of the error curve very small for as large a range of $x$ as possible, without much weight being given to the values of the error curve elsewhere. Because of this property, $L_p$-norms for $p < 1$ are not commonly used in practical situations.

(b) $p = 1$ ($L_1$-norm)

In this case, all changes in the error curve are weighted equally, regardless of whether they occur near the zeros or near the extremes of the error curve.
(c) $p = 2$ (L$_2$-norm or least squares norm)

This is a very commonly selected norm. Since
\[ |F(A, x) - f(x)|^2 = [F(A, x) - f(x)]^2 , \]
it has computational advantages. And, the change of $\delta(x)$ depends mildly on its distance to the zeros or to the extremes of the error curve.

(d) $p = \infty$ (L$_\infty$-norm or Tchebycheff norm)

Equivalently, this norm is defined as follows,
\[ \|g(x)\| = \sup_{0 \leq x \leq 1} |g(x)|. \]

Thus, a best Tchebycheff approximation to $f(x)$ minimizes the maximum value of $|F(A, x) - f(x)|$. This effect can be seen in Fig. 2.2, and the largest error has the only nonzero weight.

The L$_p$-norms can be generalized by the notion of weight function $w(x)$.

\[ L_p(w, F - f) = \left[ \int_{0}^{1} |F(A, x) - f(x)|^p w(x) \, dx \right]^{1/p} . \]

The $p$th root associated with $L_p(w, F - f)$ may be omitted from time to time since the minimal $L_p(w, F - f)$ will give the same answer as the one with the $p$th root.

Although the only requirement is that $w(x)$ is integrable, most researchers consider only weight functions which have the following properties:
\[ \int_{0}^{1} w(x) \, dx = 1 \]
\[ w(x) > 0, \quad 0 \leq x \leq 1. \]

The reason why we consider $x \in [0,1]$ is because that there is no general difference between the L$_p$-norm on [0,1] and the L$_p$-norm on [a,b] where $a, b \in \mathbb{R}$. 

\[ 0 \leq x \leq 1. \]
Now, let us consider the question of selecting the approximating function. Although there is no rigorous mathematical analysis of this question (Rice, 1964), there are some types of approximating functions that are most popular among researchers.

The key to efficient approximation is to find an approximating function which can take on the same nature or behavior as \( f(x) \). However, this requires some prior knowledge about the approximated function, the target system. It is not likely that this information is available in practice. Therefore, the question becomes selecting a flexible approximating function which can accommodate the variation in the structure of the approximated functions. The most commonly used are polynomial functions (including pointwise and piecewise polynomials), and kernel functions.

2.1.1. POLYNOMIAL APPROXIMATION (PA)

One of the most fundamental theorems of approximation theory is the Weierstrass approximation theorem of 1885.

**Theorem 2.1.1 (Weierstrass Approximation Theorem)** Let \( f(x) \in C[a,b] \).

Given an \( \varepsilon > 0 \), we can find a polynomial \( P_n(x) \) (of sufficiently high degree) for which

\[
\| f(x) - P_n(x) \| \leq \varepsilon, \quad a \leq x \leq b.
\]

where \( \| \cdot \| \) is the uniform norm over the interval \([a,b]\).
Basically, Weierstrass' theorem asserts the possibility of uniform approximation by a polynomial to continuous functions over a closed interval, say C[a,b].

It is quite obvious that the Weierstrass approximation theorem (WAT) can be applied if \( f(x) \) is an analytic function on the interval \([a,b]\). However, for functions that are not analytic, i.e., there is no expansion in power series, WAT assures us that we can uniformly approximate functions which are merely continuous.

WAT gives us a notion of existence of a polynomial which can approximate arbitrarily closely any continuous function which is defined on a compact set. It goes without saying that in order to achieve more and more accuracy in the approximation, the polynomial will have to be of higher and higher degree. It is also important for both theory and numerical practice to accomplish as much as possible with polynomials of a fixed degree. In order to investigate this question, the notion of closeness of approximation (or performance measure of the approximation) must be defined. Frequently, we measure the closeness of approximation over the interval by taking \( L_\infty \)-norm (Tchebycheff norm) or \( L_2 \)-norm (least squares norm). Once a criterion of closeness of approximation has been decided upon, we may look into the problem of whether, among the elements of \( P_n(x) \), there is one whose closeness to a given function \( f(x) \) is not exceeded by any other element of \( P_n(x) \). If there is, this particular \( P_n(x) \) is known as a best approximation to \( f(x) \).
Much work has been conducted to develop the best approximation theory of polynomial approximation. (For detailed information about polynomial approximation refer to Rice (Rice, 1964) and Davis (Davis, 1975)).

Polynomial approximation has some very good properties such as uniqueness and existence. However, it suffers from computational difficulty. In most cases, polynomial approximation is very sensitive to the information which is available when the approximation is constructed. In practice, it is very unlikely that all of the information will be available at the moment of constructing the model. Therefore, it is highly desired to have an iterative approximating procedure. Unfortunately, polynomial approximation does not have this property. Rice (1964) gave one of the reasons why polynomial approximation is not a good candidate in achieving this particular requirement.

"Functions which express physical relationships are frequently of a disjointed or disassociated nature. That is to say that their behavior in one region may be totally unrelated to their behavior in another region. Polynomials along with most other mathematical functions, have just the opposite property."

To solve this problem, a more flexible representation has been developed: the kernel approximation.
2.1.2. KERNEL APPROXIMATION (KA)

The approximation of functions by means of a kernel, a function that is independent of the design, can be easily described. If one denotes the kernel by $K_{\lambda}(x,s)$, then an approximation $A_{\lambda}(f,x)$, constructed by a single kernel function, to $f(x)$ is obtained by:

$$f(x) \sim A_{\lambda}(f,x) = \int f(s)K_{\lambda}(x,s)ds$$

or

$$f(x) \sim A_{\lambda}(f,x) = \sum_{i=1}^{n} y_i K_{\lambda}(x,x_i)$$

where $\{(x_i,y_i), i = 1, 2, ..., n\}$ is a set of behavior data, $\lambda$ is the characteristic parameter of the kernel and $y_i = f(x_i)$.

In a multiple kernel functions case, say $m$ functions, the above formula can be written as:

$$f(x) \sim A(f,x) = \sum_{j=1}^{m} \int f(s)K_{\lambda_j}(x,s)ds$$

or

$$f(x) \sim A(f,x) = \sum_{j=1}^{m} \sum_{i=1}^{n} y_i K_{\lambda_j}(x,x_i)$$

It is obvious to see that the quality of the approximation depends on the nature of the kernel function $K_{\lambda}(x,s)$ and the number of kernels. Some examples of the kernel functions are given in Fig. 2.4.
Fig. 2.4. Examples of Kernel Functions.
It is not difficult to see the reason why a kernel function can perform as an approximating function. One of the extreme cases is to define the "delta" function \( D(x-s) \) as the kernel; i.e.

\[
\int_{-\infty}^{\infty} D(x-s) \, ds = 1
\]

where \( D(x-s) = \begin{cases} 0 & x \neq s \\ 1 & x = s \end{cases} \).

Then

\[
f(x) = \int D(x-s)f(s) \, ds
\]

This formula only serves as an explanation of why kernel approximation will work. It has no value in constructing any useful approximation. In kernel approximation research, most of the efforts have been made using some other class of kernels. The properties of these kernels are listed as following.

(1) They have finite support, \([-1,1]\) or \([a,b]\).

Define \( I_{\delta_1, \delta_2}(x) = \{s | -1 \leq s \leq 1, s \notin [x-\delta_1, x+\delta_2]\} \)

\[
\lim_{\lambda \to \infty} \int_{-\delta_1}^{\delta_2} K_\lambda(x,s) \, ds = 0, \quad \delta_1, \delta_2 > 0
\]

\[
\lim_{\lambda \to \infty} \int_{x-\delta_1}^{x+\delta_2} K_\lambda(x,s) \, ds = 1
\]

(2) They are nonnegative.

\( K_\lambda(x,s) \geq 0 \)
(3) They satisfy the following moment conditions.

a. \( \int_{-1}^{1} K_{\lambda}(x, s)ds = c \), \( c \) is a constant, and without lost of generality \( c = 1 \).

b. \( \int_{-1}^{1} s \cdot K_{\lambda}(x, s)ds = 0 \)

Since \( K_{\lambda}(x, s) \) is defined as nonnegative, this condition implies symmetry of the kernel function.

c. \( \int_{-1}^{1} s^2 \cdot K_{\lambda}(x, s)ds = \alpha \neq 0 \)

This condition makes the kernel function much richer than the impulse function. This guarantees the kernel function has tails.

d. \( \int_{-1}^{1} K_{\lambda}^2(x, s)ds < \infty \)

These conditions, however, are only sufficient conditions for kernels to perform as a second order approximating functions. For instance, in Fig. 2.4, Dirichlet kernel is not a nonnegative kernel. But, if the kernels have all the above properties, it is easy to prove that

\[ \lim_{\lambda \to \infty} A_{\lambda}(f, x) = f(x), \]  

i.e. the sequence of constructed approximations will converge to the approximated function \( f(x) \) (Rice, 1964).

Even though the polynomial function approximation has been shown to be the best approximation method (Davis, 1975), it is very difficult to implement in practice. Kernel function approximation, however, emphasizes the implementation perspective. In the extreme case, i.e. \( \lambda \to \infty, \delta \to 0 \), all kernels perform like the "delta" function which
is known to be able to approximate any arbitrary function \( f(x) \in C[a,b] \). Due to these characteristics of the kernel approximation, a more interesting question is to search for a more flexible kernel which can form a better approximation under the finite and fixed number of kernels. It is desired that this kind of kernel can form a better approximation to the approximated function \( f(x) \in C[a,b] \) for a fixed \( m \) if we define the approximation as

\[
\sum_{j=1}^{m} \sum_{i=1}^{n} y_i K_j(x, x_i)
\]

As we have stated before, the properties (1), (2), (3), and (4) are not necessary conditions. It is believed that a more flexible kernel might have a better approximating capability. This capability becomes crucial in practice, and especially for finite memory implementation.

It is obvious that the nonnegativity requirement can be dropped since Dirichlet kernel has been shown to be a kernel function with nice properties. Another condition that can also be dropped is the symmetry requirement. Consider the more realistic situation of a finite points approximation, \( x \in X_m \) where \( X_m = \{ a \leq x_1 \leq x_2 \leq \ldots \leq x_m \leq b \} \) and for some \( f(x_j) > 0 \) and \( f(x_{i+1}) < 0 \). This case needs at least two nonnegative kernels to cover these two points. Instead, it could be represented by a single and more flexible kernel which it is not limited to the family of nonnegative and symmetric kernels.

Unavoidably, there is a problem associated with using a more flexible kernel. It is the computational difficulty introduced by the
nonlinearity of a more flexible kernel. This is one of the main reasons why most kernel approximation research has concentrated on linear approximation theory. In this research, we introduce a much richer kernel function which does not belong to the nonnegative and symmetric family. It is expected that this kernel function is more suitable for actual implementation. And, the computational difficulty can be resolved by implementation with neural networks.

Let us go back to WAT again. WAT shows the richness of the polynomial approximation. It is quite reasonable to ask whether we can isolate the property of the polynomials which makes the WAT possible for other kind of functions. Fortunately, there is an answer to this question. There is a generalized version of WAT, the so-called the Stone-Weierstrass Theorem (SWT). Before start to look into the SWT, the following definitions (Rudin, 1964) are necessary.

**Definition 2.1.3.** A family of real functions defined on a set $K$ is said to be an algebra if

(i) $f + g \in \mathcal{A}$

(ii) $f \cdot g \in \mathcal{A}$

(iii) $c \cdot f \in \mathcal{A}$

$\forall f \in \mathcal{A}, \ g \in \mathcal{A}, \ c \in \mathbb{R},$ i.e. $\mathcal{A}$ is closed under addition, multiplication, and scalar multiplication.

If $\mathcal{A}$ has the property that $f \in \mathcal{A}$ whenever $f_n \in \mathcal{A}$ $(n = 1, 2, \ldots)$ and $f_n \to f$ uniformly on $K$, then $\mathcal{A}$ is said to be uniformly closed.
Let $\mathcal{B}$ be the set of all functions which are limits of uniformly convergent sequences of members of $\mathcal{A}$. Then $\mathcal{B}$ is called the uniform closure of $\mathcal{A}$.

**Theorem 2.1.2** Let $\mathcal{B}$ be the uniform closure of an algebra $\mathcal{A}$ of bounded functions. Then $\mathcal{B}$ is a uniformly closed algebra.

**Definition 2.1.4** Let $\mathcal{A}$ be a family of functions on a set $K$. Then $\mathcal{A}$ is said to **separate points on $K$** if to every pair of distinct points $x_1, x_2 \in K$ there is a function $f \in \mathcal{A}$ such that $f(x_1) \neq f(x_2)$, i.e. $x_1, x_2 \in K, x_1 \neq x_2$, then $\exists f \in \mathcal{A}$ such that $f(x_1) \neq f(x_2)$.

**Definition 2.1.5.** If to each $x \in K$ there corresponds a function $g \in \mathcal{A}$ such that $g(x) \neq 0$, we say that $\mathcal{A}$ **vanishes at no point of $K$**. i.e. $\forall x \in K, \exists g \in \mathcal{A}$, such that $g(x) \neq 0$.

Now, the Stone-Weierstrass theorem can be stated as follows.

**Theorem 2.1.3 (Stone-Weierstrass Theorem).** Let $\mathcal{A}$ be an algebra of real continuous functions on a compact set $K$ (e.g. closed interval). If $\mathcal{A}$ separates points on $K$ and if $\mathcal{A}$ vanishes at no point of $K$, then the uniform closure $\mathcal{B}$ of $\mathcal{A}$ consists of all real continuous functions on $K$ (i.e. $\mathcal{A}$ is $\rho_K$-dense in the space of real continuous functions on $K$).

This is very useful theorem in a sense that we will use it to show the denseness of the family of our kernel in the next chapter.
2.2. NEURAL NETWORKS (NN)

A revolutionary idea has been proposed to develop a very flexible computational technique using parallel architecture. This idea is sometimes called 'neural network', sometimes 'parallel distributed processing', or sometimes 'connectionism'. A neural network (Fig. 2.5) uses a large number of simple processing elements or units, each connected to some number of other units in the system (the mathematical description of the neural networks will be given in the next section). The units have little information stored internally, typically only a single scalar activity level \( a_j \), that is, a sort of short-term working memory. The long-term storage of information is accomplished by altering the pattern of inter-connections among the units, or by modifying a quantity called the weight \( w_{ij} \) associated with each connection.

Fig. 2.5. Neural Network.

---

1 Jerry Feldman coined the term 'connectionism' to refer to the study of a certain class of massively parallel architectures for artificial intelligence. (Fahlman and Hinton, 1987).
In daily life, many tasks require that a number of different pieces of information must be processed simultaneous to satisfying different constraints. For instance, picking up a pencil, we need not only note the position of the pencil but also the position of the fingers and the speed of the moving hand and the shape of the hand. Intuitively, these tasks seem to require mechanisms in which each aspect of the information can act on other aspects, simultaneously influencing other aspects and being influencing by them. This is one of the reasons why the models for dealing with these aspects are called neural network or PDP (parallel distributed processing) models.

First, a brief history of neural network research and related neural network oriented data processing work are given. Then, some basic principles of the PDP approach are provided as the background information.

2.2.1. BRIEF HISTORY OF NEURAL NETWORK RESEARCH

Neural network research can be described as having two distinct stages. In this dissertation, we call them the old stage and the current stage.

Old Stage

The neural network is not a totally new idea. The first concepts originated from McCulloch and Pitts's ingenious paper (McCulloch and Pitts, 1943). The structure which they introduced is today called a
'McCulloch-Pitts net'. The net is an extremely simplified representation. For instance, switching occurs only at regular, discrete intervals. Thus, formal neurons are just simple logical switches, quite unlike real neurons. Despite these simplifications, McCulloch-Pitts nets are important in that they can embody any operations and processes that can be described in logical terms.

McCulloch and Pitts proved that formal neural nets, if given infinitely large memory spaces, are equivalent to a class of computing machines that Alan M. Turing has shown to be computationally universal. But what happens if such nets malfunction from time to time or are damaged? (This concern does have physiological and practical interest). This problem attracted one of the leading mathematicians of this century, John von Neumann. He solved this problem by introducing redundancy, using many neurons to do the job of one. In such nets, one bit of information is signaled by the synchronous activation of many neurons rather than by the all-or-nothing activation of one formal neuron. Von Neumann proved that redundant McCulloch-Pitts nets operating in such a fashion can be designed to carry out arithmetical calculations with high reliability (Cowan and Sharp, 1988). In order to make these nets solve any problem, hundreds of neurons must be connected together. This complexity makes the design problem intractable. The simplest solution to this problem is to make the network program itself. Such a network is called an adaptive neural net.
In 1949 Hebb (Hebb, 1949) proposed that the connectivity of the brain is continually changing as an organism learns different functional tasks and that cell assemblies are created by such changes. He postulated that the repeated activation of one neuron by another through a particular contact, or synapse, increases its conductance, so that groups of weakly connected cells, if synchronously activated, tend to organize into more strongly connected assemblies. Although Hebb's proposal lacks evidence to prove that the brain really operates this way, this concept has been widely adopted in adaptive neural nets research, especially in pattern classification and pattern recognition applications.

Some ten years after the publication of McCulloch and Pitt's paper, a major approach to the pattern-recognition problem was introduced by Frank Rosenblatt, who showed how McCulloch-Pitts nets with modifiable connections could be "trained" to classify certain sets of patterns as similar or distinct (Rosenblatt, 1958). He called such nets perceptrons. A perceptron contains of a set of "sensory" units connected, through a single layer of McCulloch-Pitts neurons, to a set of "motor" units. Rosenblatt followed Hebb's idea to develop a method to adjust the weights among each units. It has been shown that after only a finite number of presentations of stimulus-response patterns, the weights converge to a set of values representing whatever representation or classification is embodied in these patterns (Novikoff, 1963). Shortly after Rosenblatt's first publication there appeared a closely related variant of the perceptron invented by Bernard Widrow and M. E. Hoff. They called it the adaline, for adaptive linear neuron (Widrow and Hoff, 1960). Basically,
perceptrons and adalines use the same structure (Fig. 2.6), many linear units. The only difference between them lies in the training procedure.

\[
\begin{align*}
\text{Input Units} & \quad w_{io} \quad \text{Output Unit} \\
\text{Input Units} & \quad w_{ip} \\
\end{align*}
\]

Fig. 2.6. Perceptron and Adaline.

In 1969, Seymour A. Papert and Marvin L. Minsky wrote a book, Perceptrons, which showed that elementary perceptrons cannot distinguish between such simple patterns as T and C (Papert and Minsky, 1969). And worse, they showed that all linear neural nets such as perceptrons cannot solve the XOR(Exclusive OR) problem (Fig. 2.7).

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig. 2.7. XOR (eXclusive OR).
This problem arises because a linear net cannot construct its own internal representation. One of the possible structures that solves the XOR problem is to introduce a hidden layer (Fig. 2.8).

![Diagram of Perceptron and Adaline with Hidden Layer](image)

**Fig. 2.8. Perceptron and Adaline with Hidden Layer.**

But, Papert and Minsky claimed that no algorithm could be constructed to determine the weights associated with the links in the hidden layer. Papert and Minsky's effort successfully decreased interest in neural net research. After that, most of the artificial intelligence related projects used the symbolic approach (Papert, 1988). This marked the end of the old stage of neural net research.

**Current stage**

It turns out that some of the limitations of simple perceptrons and adalines can be overcome. While some partial solutions had been
suggested from time to time, the major contribution to this problem came after 1985. Between 1969 and 1985, there were a lot of important research results which have proved to have very important influence on the current neural net research, such as "Non-holographic Associative Memory" (Willshaw et al., 1969), "Theory of Cerebellar Cortex" (Marr, 1969), "Associative Memory" and "Content-Addressable Memory" (Kohonen, 1977), and "Brain State in the Box" (Anderson et al, 1977). According to Cowan and Sharp (Cowan and Sharp, 1988), the current stage could considered as starting from Hopfield's work (Hopfield, 1982, 1984).

Hopfield recognized the formal analogy between a net of neuronlike elements with random symmetric connection weights and a spin glass. Using Hebb's rule for synaptic weight modification, he showed that the weights can be modified to stabilize net activity. Given such weights, any initial configuration of active and inactive elements will evolve toward a stable configuration (Fig. 2.9).
Thus, the stable configurations can be used to store information in a reliable fashion. Although Hopfield nets suffered from inefficiency\(^2\), the principle they embody, storing information in dynamically stable configurations, is profound. The Hopfield net has been applied to the "traveling salesman problem" which is recognized as a very difficult optimization problem. Although the Hopfield net does not find the optimal solution, i.e. the shortest path, it does find one that compares quite favorably with the nonoptimal solution found through the Kernighan-Lin procedure (Hopfield and Tank, 1985).

Michael Cohen and Stephen Grossberg (Cohen and Grossberg, 1983) developed the fundamental stability theory for this kind of network.

\(^2\)In Hopfield's 1982 paper, he mentioned, "...about 0.15N (N is the number of units) states can be simultaneously remembered before error in recall is severe".
Besides the inefficiency problem, Hopfield nets also suffer from one defect in their ability to find the best solution in constrained optimization problems. They can get "trapped" in locally optimal configurations. To find the global optimum, the net must generate some "noise" to help itself to get out of the local optima from time to time. This is the essential idea been applied by a well-known version of a Monte Carlo procedure (Metropolis et al., 1953). In this procedure the change is produced by a random resource. If the new configuration is more stable, it is retained. Otherwise the configuration is rejected. Such a procedure, although slow, will eventually find the most stable configurations. Ackley, Hinton and Sejnowski accordingly used this procedure to find stable configurations in Hopfield nets, the resulting net and computational procedure is called Boltzmann machine (Ackley et al, 1985).

The Boltzmann machine provides a solution to the credit assignment problem for hidden units by use of adaptive Hopfield nets. A Boltzmann machine can form a representation that eventually reproduces relations between classes of events in its input environment. More generally, it provides a way in which distributed representations of abstract symbols can be formed and therefore permits the investigation of symbolic reasoning by means of adaptive neural nets (Hinton, 1984). The Boltzmann machine represents a considerable advance in unsupervised machine learning. However, because the machine uses a Monte Carlo procedure to find stable configurations, the learning is very slow. Furthermore, the Boltzmann machine is a Hopfield net with only symmetric connections. These limitations have been overcome by D.E.
Rumelhart, G. Hinton and R.J. Williams. They introduced a multiple layered perceptron structure network with hidden units (Fig. 2.10) and developed a method to solve the weight adjustment problem which is called the "error back-propagation method" (Rumelhart et al, 1986).

![Multi-layer Perceptron Network](image)

Fig. 2.10. Multi-layer Perceptron Network.

In this research, the error back-propagation method is adopted as the basic rule to modify the interconnected weights. A detailed description of this method is presented later. A promising feature of the back-propagation method is that it is able to solve the T/C problem and the XOR problem. The inability of perceptrons and adalines to solve these problems resulted in a lack of continued interest in neural net methods at the end of the "old stage" of research. Largely because of this development, neural net research has been able to accelerate and has been
considered as a promising approach to attack some very difficult problems in science and in engineering.

2.2.2. THE STRUCTURE OF THE MULTI-LAYER PERCEPTRON MODEL AND THE ERROR BACK-PROPAGATION METHOD

According to Rumelhart and McClelland (Rumelhart et al, 1986), there are eight major components of a multi-layer perceptron (or PDP) model:

* A set of processing units.

* A state of activation for each unit.

* An output function for each unit.

* A propagation rule for propagating patterns of activities through the network of connectivities.

* An activation rule for combining the inputs impinging on a unit with the current state of that unit to produce a new level of activation for the unit.

* A learning rule whereby patterns of connectivity are modified by experience.

* An environment within which the system must be operated.

Fig. 2.11 illustrates the basic aspects graphically.
Fig. 2.11. Processing Units.

where

\[ a_i(t) = F_i(a_i(t-1), Net_i(t)) : \text{activation value of unit } i \text{ at time } t. \]

\[ F_i(a_i(t-1), Net_i(t)) : \text{activation function (rule) of unit } i. \]

\[ O_i(t) = f_i(a_i(t)) : \text{output value of unit } i \text{ at time } t. \]

\[ f_i(a_i(t)) : \text{output function of unit } i. \]

\[ w_{ij}(t) : \text{weighing strength, the amount of effect that unit } i \text{ has} \]
\[ \text{on unit } j. \text{ There is no cycle allowed in this network.} \]

\[ Net_j(t) : \text{the input strength received by unit } j \text{ at time } t. \]

\[ Net_j(t) = \sum_i w_{ij}(t)O_i(t) \]

These systems are viewed as being plastic in the sense that the pattern of interconnections is not fixed for all time; rather, the weights can undergo modification as a function of experience. In this way the system can evolve. In the rest of the section, we develop an explicit notation for each of these major components.
1. Processing units:

A unit's job is simply to receive input from its neighbors and, as a function of the inputs it receives, to compute an output value which it sends to its neighbors. The system is inherently parallel in that many units can carry out their computations at the same time. The units could be grouped into three different kinds: input, hidden, and output (Fig. 2.12).

![Diagram of three kinds of units](image)

**Fig. 2.12. Three Different Kind of Units.**

2. The state of activation:

We need a representation of the state of the system at time $t$. This is primarily specified by a vector of $N$ elements, i.e., we have $N$ units in the network. Real values, $a_i(t)$, represent the pattern of activation over the set of processing units. Each element of the state vector stands for the
activation of one of the units at time $t$. The activation of unit $U_i$ at time $t$ is designated $a_i(t)$. The pattern of activation over the set of units captures what the system is representing at any time. Activations may have the following types:

(i) Discrete

(1) binary, e.g. $\{-1, 1\}$, $\{-, +\}$, or $\{0, 1\}$.

(2) finite set, e.g. $\{1, 2, \ldots, 10\}$

(ii) Continuous

(1) bounded, e.g. $[0, 1]$.

(2) unbounded, e.g. $\mathbb{R}$

Each of these assignments leads to a model with slightly different characteristics. The assignment should be determined before starting to build the model.

3. Output of the units:

The degree to which each unit will affect its neighbors is determined by their degree of activation and the weighting strength between them. The observable part of the activation is represented by the value of output which is

$$O_i(t) = f_i(a_i(t))$$

Actually, $f_i(\cdot)$ could be performed as a linear function or a nonlinear function. Usually, $f_i(\cdot)$ is defined as both bounded from above and below, i.e. the output of $f_i(\cdot)$ is constrained to be within a closed interval.
Sometimes $f_i(\cdot)$ is assumed to be a stochastic function in which the output of the unit depends on its activation values in a probabilistic fashion.

4. The pattern of connectivity:

The pattern of connectivity constitutes the system's long-term memory and determines how it will respond to an external input. A positive weight represents an excitatory connection and a negative weight represents an inhibitory connection. It is convenient to represent such a pattern of connectivity by a weight matrix $W$ in which the entry $W_{ij}$ represents the strength of the connection from unit $U_i$ to unit $U_j$.

- If $w_{ij} > 0$, that means unit $i$ excites unit $j$.
- If $w_{ij} = 0$, that means unit $i$ has no direct connection to unit $j$.
- If $w_{ij} < 0$, that means unit $i$ inhibits unit $j$.

5. The rule of propagation:

We need a rule which takes the output vector, $O(t)$, representing the output values of the units of the previous layer and combines it with the connectivity matrices to produce a net input into the unit. We let $Net_j(t)$ mean the net input of unit $U_j$ at time $t$.

$$Net_j(t) = \sum_i w_{ij}(t)O_i(t)$$
6. Activation rule:

We also need a rule whereby the net input of a particular unit are combined with the current activation state of the unit to produce a new activation state. This is represented as a function $F_i(\cdot,\cdot)$:

$$a_i(t) = F_i(a_i(t-1), \text{Net}_i(t))$$

$F_i(\cdot,\cdot)$ can have the following forms:

(i) Linear function,

$$a_i(t) = a_i(t-1) + \text{Net}_i(t).$$

(ii) Threshold function (hard limiter or hard-threshold function),

$$a_i(t) = \begin{cases} 
1 & \text{if } (a_i(t-1) + \text{Net}_i(t)) \geq T \\
0 & \text{if } (a_i(t-1) + \text{Net}_i(t)) < T 
\end{cases}$$

where $T$ is a predetermined threshold value.

(iii) Stochastic,

(iv) Decaying with time. In this case, $F_i(\cdot,\cdot)$ should be written as $F_i(\cdot,\cdot,t)$,

(v) Quasi-linear (or soft-threshold function),

This is the most common class of activation function. In this case $F_i(\cdot,\cdot)$ is a nondecreasing function.
(vi) Semi-linear,

In this case, \( F_i(\cdot) \) is quasi-linear and differentiable. One example of this type of activation function is logistic function.

7. Learning rules:

Changing the processing or knowledge structure in a PDP model involves modifying the patterns of interconnectivity. Basically, there are three kinds of modification:

(i) The development of a new connection,

(ii) The loss of an existing connection,

(iii) The modification of the strength of a connection that already exists.

Whether the third kind of modification will include the first and the second kind is a matter of allowing every two units or only selected unit pairs to have pathways in the network. Most of the PDP models have constraints in constructing the connections among nodes.

Virtually all learning rules can be considered as a variant of the Hebbian rule which suggests that when a cell A repeatedly and persistently takes part in firing another cell B, then A's efficiency of firing B is increased (Hebb, 1949). Translated to PDP's terminology, it should be stated as "If a unit, \( U_j \), receives an input from another unit, \( U_i \) at time \( t \); then, if both are highly active, the weight, \( W_{ij}(t+1) \), from i to j should be strengthened." According to Rumelhart and McClelland's assumption
(Rumelhart and McClelland, 1986), the change of weight can be considered as the product of two functions, g and h.

\[ \Delta w_{ij}(t) = g(O_j(t), D_j(t)) \cdot h(I_i(t), w_{ij}(t)) \]

\[ w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t) \]

where \( D_j(t) \) is a teaching input to \( U_j \), i.e. desired output\(^3\) of \( U_j \) at time \( t \). \( I_i(t) \) is the output signal from node \( i \) at time \( t \). If \( U_j \) is not an input unit, \( I_i(t) = O_i(t) \). If \( U_j \) is an input unit, \( I_i(t) \) is the \( i \)th input value of the network at time \( t \).

(i) The simplest rule is

\[ \Delta w_{ij}(t) = \eta \cdot I_i(t) \cdot O_j(t) \]

where \( \eta \) could be viewed as learning rate which is greater than 0.

(ii) Widrow-Hoff or delta rule:

If we let \( h(I_i(t), w_{ij}(t)) = I_i(t) \) and \( g(O_j(t), D_j(t)) = \eta \cdot (D_j(t) - O_j(t)) \)

then we can rewrite \( \Delta w_{ij}(t) \) as follows.

\[ \Delta w_{ij}(t) = \eta \cdot (D_j(t) - O_j(t)) \cdot I_i(t) \]

This is called delta rule because the change in strength is proportional to the difference between the actual activation achieved and the target activation provided by a 'teacher'. Basically, the delta rule is only adequate in two-layer associative networks (or multi-layer networks with linear activation functions and linear output functions). Although Minsky and Papert (Minsky and Papert, 1969) have provided a very clear

\(^3\) In PDP approach, a "teacher" is assumed to exist in the environment.
analysis for this kind of network (i.e. without hidden units), the perceptron convergence procedure, their inability to solve certain kinds of problems has already been mentioned. A classic example is the exclusive-or (XOR) problem. Because of this, the network with hidden units is necessary in terms of generalization. The task of developing a new learning rule in this kind of network is a very challenging task. There have been a couple of independent developed learning rules have been developed in the last five years (Rumelhart et al, 1986), (le Cun, 1987). Although they were developed under different disciplines, they ended up with the similar result which is displayed in the following.

(iii) Error backward propagation method (generalized delta rule):

(Rumelhart, Hinton and McClelland, 1986)

Without hidden units, the error surface is shaped like a bowl with one minimum, so gradient descent is guaranteed to find the best set of weights. In the case with hidden units this is not true. In the case of linear units, hidden units provide no advantage because the multi-layer system with linear units could be represented as a two-layer system with linear units. Therefore, we describe the back propagation method by assuming each unit is a semi-linear unit (unit with semi-linear activation function). Let

$$\text{Net}_j(t) = \sum_{i} w_{ij}(t) O_i(t)$$  \hspace{1cm} (2.2.4)

where $O_i(t) = X_i(t)$ for input units. and $X_i(t)$ is the $i^{th}$ input to the network at time $t$. 

Then, one semi-linear activation function is

\[ a_j(t) = F_j(a_j(t-1), Net_j(t)) = Net_j(t) \]  \hspace{1cm} (2.2.5)

The output function is given as follows.

\[ O_j(t) = f_j(a_j(t)) = f_j(Net_j(t)) \]  \hspace{1cm} (2.2.6)

where \( f_j(\cdot) \) is differentiable and nondecreasing.

Let \( E \) be the sum-squared error function and assume a network has \( K \) input units, \( M \) output units and \((N-K-M)\) hidden units

\[ E = \frac{1}{2} \sum_{m=1}^{M} (O_m(t) - D_m(t))^2 \]

then

\[ \frac{\partial E}{\partial w_{ij}} = \left( \frac{\partial E}{\partial Net_i} \right) \left( \frac{\partial Net_i}{\partial w_{ij}} \right) \]  \hspace{1cm} (2.2.7)

by (2.2.4) we obtain

\[ \frac{\partial Net_j}{\partial w_{ij}} = O_i \]  \hspace{1cm} (2.2.8)

Let

\[ \frac{\partial E}{\partial Net_j} = -\delta_j \]  \hspace{1cm} (2.2.9)

and by (2.2.7), (2.2.8) and (2.2.9), we have

\[ \frac{\partial E}{\partial w_{ij}} = -\delta_j O_i \]  \hspace{1cm} (2.2.10)

Usually, the information that is available is the network inputs \( I_k = X_k(t) \) for \( k=1,2, ..., K \) and the network outputs \( O_m = D_m(t) \) for \( m=1,2, ..., M \) where \( \{(X(t),D(t)) \mid t \geq 0\} \) is the set of input and output data. It is unlikely that we will have the information of the desired output value for each hidden unit. Therefore, the key issue here is to determine what \( \delta_j \) is for each unit \( U_j \), especially the hidden units, in the network. Here, we can derive
a simple recursive computation of these \( \delta \)'s which can be implemented by propagating error signals backward through the networks.

According to (2.2.9)

\[
\delta_j = - \left( \frac{\partial E}{\partial \text{Net}_j} \right) = - \left( \frac{\partial E}{\partial O_j} \right) \left( \frac{\partial O_j}{\partial \text{Net}_j} \right) \quad (2.2.11)
\]

by (2.2.6) we have

\[
\frac{\partial O_j}{\partial \text{Net}_j} = f'_j \quad (2.2.12)
\]

where prime means the first derivative.

We have to consider two different cases to determine \( \frac{\partial E}{\partial O_j} \),

**Case 1.** The unit \( U_m \) is an output unit

\[
\frac{\partial E}{\partial O_m} = -(D_m - O_m) \quad (2.2.13)
\]

According to (2.2.11), (2.2.12), and (2.2.13), we have

\[
\delta_m = (D_m - O_m) f_m \quad (2.2.14)
\]

According to (2.2.10) and (2.2.14)

\[
\frac{\partial E}{\partial w_{jm}} = - \delta_m O_j = -(D_m - O_m) f_m O_j \quad (2.2.15)
\]

**Case 2.** The unit \( U_j \) is a hidden unit.

\[
\frac{\partial E}{\partial O_j} = \sum_m \left( \frac{\partial E}{\partial \text{Net}_m} \right) \left( \frac{\partial \text{Net}_m}{\partial O_j} \right)
\]

\[
= \sum_m \left( \frac{\partial E}{\partial \text{Net}_m} \right) w_{jm}
\]

\[
= - \sum_m \delta_m w_{jm} \quad (2.2.16)
\]

So,

\[
\delta_j = f_j \left( \sum_m \delta_m w_{jm} \right) \quad (2.2.17)
\]

\[
\frac{\partial E}{\partial w_{ij}} = - \delta_j O_i = -(f_j \cdot \sum_m \delta_m \cdot w_{jm}) \cdot O_i \quad (2.2.18)
\]

From above, (2.2.14) and (2.2.17) form a recursive procedure for computing the \( \delta \)'s for all units in the networks. Therefore, the
application of this back propagation method (or generalized delta rule) involves the following two phases:

(1) In the first phase, the input is presented and propagated forward through the network to compute the output value $O_j$ for each unit. This output is then compared with the targets, resulting in an error signal $\delta_j$ for each output.

(2) The second phase involves a backward pass through the network during which the error signal is passed to each unit in the network and the appropriate weight changes are made by using the procedure which mentioned above.

8. Environment

This is concerned with the inputs to the PDP model or the target behavior that the PDP model has to learn. Up to now, much research has assumed that inputs are restricted to the orthogonal or linearly independent (Anderson, 1977; Amari, 1978).

2.3. LOCALIZED DISTRIBUTED REPRESENTATION: THE CONNECTION BETWEEN KERNEL APPROXIMATION AND NEURAL NETWORKS

Bobrow proposed a view to treat representation as "a selective mapping of [interesting] aspects of the world" (Bobrow, 1975). Another perspective of the representation issue is the mechanism of the representation, i.e. how those representations or mappings are
constructed. Most researchers preselect a set of elements. Then, all the "interesting aspects of the world", named as entities, are constructed by those elements. That means, given a set of simple elements, the entities are represented by the combination of those simple elements. Generally speaking, there are three kinds of mechanisms. They are local representation, distributed representation and localized distributed representation.

If we assumed a set with an infinite number of elements, the simplest representation is to use each element to represent each individual entity. This kind of representation is the so-called local representation scheme. It has the property that the representation of each entity is mutually independent to each other. The table look-up method, which has been described in Chapter I, is an example of this kind of scheme. (The advantages and the disadvantages of the local representation can be found within the description of the table look-up method in Chapter I).

Because of the impossibility of having an infinite set, however, a more flexible representation was developed which is called a distributed representation scheme. According to (Hinton et al, 1986), this kind of representation scheme can be defined as follows.

"Each entity is represented by a pattern of activity distributed over many elements, and each element is involved in representing many different entities."
This is the most common representation scheme applied in neural networks research.

It is possible to identify a connection between distributed representation schemes and polynomial function approximation. Since each entity is represented by many elements and every element contributes to many entities, distributed representations suffer from the syndrome that "behavior in a small region determines the behavior everywhere". This is the same problem exhibited by the polynomial function approximation.

One possible solution to this problem is to constrain the influence of each element to only an "adequate portion" of the entities, i.e. each element is involved in representing an adequate amount of entities and each entity is represented by an adequate portion of the elements. If the "adequate portion" is defined as one entity, it can be considered as a local representation scheme. If the "adequate portion" is defined as the whole set of entities, it is a fully distributed representation scheme.

Giving the previous description of the kernel function approximation, it is not difficult to determine the relation between localized distributed representation scheme and the kernel function approximation. Every kernel function corresponds only to a small subset of the input space as shown by the limited support of the kernel function. Every element in the input space will activate a certain number of kernel functions, i.e. some kernel functions overlap with the other kernel functions.
As we have mentioned in the last section, in the old stage, most neural network models were based on McCulloch and Pitts (MP) neuron which is known as having hard threshold computational units. In the current stage, however, a refinement of MP neurons which have sigmoidal (soft threshold) computational units is widely used. But, no matter what kind of computational units are used, most neural network research emphasized approximating boolean functions instead of real-valued functions. In another words, most research has concerned assigning input patterns into one of some finite set of categories. The famous examples are the Perceptron, the Brain-State-in-a-Box (BSB) model and the Boltzmann machine. However, in system modelling and system control perspectives, real-valued function approximation will be more important and interesting.

To the author's knowledge, the first neural network research concerned with approximating real-valued functions was performed by A. Lapedes and R. Farber (Lapedes, Farber; 1988). They constructed a neural network to predict the behavior of the Glass-Mackey equation\(^4\). The architecture of their network is based on multi-layer perceptron nets. Actually, they specifically designed a three-layer (two hidden layers) perceptron-like network which is shown in Fig. 2.13.

\[ \dot{x}(t) = \frac{a \cdot x(t - \tau)}{1 + x^{10}(t - \tau)} - b \cdot x(t) \]  
This is a nonlinear differential, delay equation with an initial condition specified by an initial function defined over a strip of width \( t \).
This design is consistent with Lippmann's analysis (Lippmann, 1987) that no more than three layers (two hidden layers) are required in perceptron-like feed-forward nets to perform as a pattern recognizer. This result is due to the fact that a three-layer net can generate arbitrarily complex decision regions and the complexity of the regions is determined by the number of the units in the network. Although this analysis is valid in theoretical sense, using perceptron-like nets for real-valued function approximation is inefficient.

Generally speaking, the performance of multi-layer perceptron networks should be affected by the structure of the network and the algorithm of updating the weights of the network. The typical human cortex has six layers, and, counting fine subdivisions, perhaps as many as eight or nine in some areas (Churchland, 1986). However, it is still an open question whether increasing the number of layers in a multi-layer perceptron network will improve the performance of the network. This is not an easy question to answer, because the performance of a network is...
not only based on the structure of the network but also on the weights updating algorithm.

Currently, the most popular weights updating algorithm is the error back propagation method (which has been applied in Lapedes and Farber's work as well) which was described in the last section. Although error back propagation is quite successful in many applications, it suffers from extreme inefficiency. In Lapedes and Farber's work, it took 1 hour of Cray X/MP time at 90 MFlops (Million Floating operations per second) to reach 0.05 prediction accuracy\(^5\) (Moody and Darken, 1988). Although they didn't mention the approximate number of learning trials that the network needed to go through, it is believed that this number must be very large. Even though Lapedes and Farber's work suffered from inefficiency, they demonstrated one possible way of applying neural networks in real-valued function approximation.

One possible reason that Lapedes and Farber's network suffered from inefficiency is the distributed representation scheme they applied. As we have mentioned above, this representation scheme has the same problem as the polynomial function approximation. That is, a small region determines the behavior everywhere. One possible solution is to construct a network which has the same property as the kernel function which is known as an alternative method of function approximation. After the author started this research, a similar idea was proposed by Moody and Darken in the summer of 1988 (Moody and Darken, 1988).

\(^5\) prediction accuracy = rms prediction error/\text{std of the observations}.\
They adopted the data structure of overlapping receptive fields which has been studied in many biologically related fields. The architecture of their network is shown in Fig. 2.14.

![Moody and Darken's Network](image)

**Fig. 2.14. Moody and Darken's Network**

The activation (or output) function of each unit in their network is taken to be gaussian. In each gaussian, three parameters need to be determined. They are the location parameter, the spreading parameter and the amplitude parameter. In Moody and Darken's work, they applied two different computational procedures to determine these three parameters. The location and the spreading parameters are determined in a self-organizing procedure. The amplitude parameter was determined based on the back propagation procedure.

Although Moody and Darken's work is similar to the author's idea of improving the efficiency of the error back propagation method by constructing a localized distributed network, this research can be distinguished from their work in the following ways.
In Moody and Darken's network, gaussians are used as the activation (or output) function of each unit. It is known that gaussians are symmetric and positive functions. In this research, we apply a much more flexible output function which is not limited to the category of positive and symmetric functions. As we discussed in the section of kernel functions, it is expected that the efficiency and the performance of the network will be improved by having more flexible units.

In the previous mentioned work, neither group pinpointed the difference between the univariate model and the multivariable model. However, it is well-known that univariate problems and multivariate problems are quite different. Another possible reason why their work suffers from inefficiency is the assumption of the same feature of the univariate problem and the multivariate problem. In this research, we discuss the univariate and the multivariate problems separately. In the multivariate part, a well-known theory of the representation of a multivariate function by the superposition of univariate functions is chosen to be the fundamental theory in developing the multivariate model. It is believed that a detailed analysis of the constructed network is going to be very useful in understanding the behavior of the neural networks.

In this research, the error back propagation method is used throughout the network. No special self-organization procedure is adopted in this research. However, according to the experiments which have been conducted in this research, the kernel function does change its
location and spreading factor in order to accommodate the "environment". However, the network designed in this research shares a common property with regular feed-forward networks. Specifically, the units do not affect the other units in the same layer. The influence between units is limited to the connected layers.

It is difficult to compare the current research with the work of Moody and Darken since somewhat different problems have been addressed and the different facilities are available for each work. It is believed that both research efforts make independent contributions to neural network and system modelling research.
CHAPTER III

UNIVARIATE MODEL

Given a set of two-element vector data, i.e. input value and output value, it is difficult to construct a representation to mimic the relationship between these two elements. It is even more difficult if these variables have a nonlinear relationship and no prior structural information is given. One possible solution to this problem is to use a linear superposition of many small "elements". This idea has been depicted in the previous chapter. However, many issues must be addressed in order to implement this method. Before describing these issues, we should examine the assumptions upon which most of the nonparametric function approximation methods are based.

Most of the current nonparametric methods require the "elements" to be continuous, nonnegative, and symmetric (see Chapter II). Because of these restrictions, they require more "elements" to eliminate unnecessary influence by the other "elements". This is not a severe problem from the theoretical point of view since we can have infinite number of such "elements" to lump together. However, a more interesting and challenging issue is to determine a practical implementation since we cannot have an infinite number of "elements".
The "cancel out" phenomenon means a waste of resources. It is conceivable that a more flexible "element", i.e. without the constraints of nonnegativity and symmetry, will be a better candidate to perform as an approximating function. Besides the construction of more flexible "elements", i.e. approximating functions, a computational procedure associated with these "elements" must be developed as well.

The structure of the proposed element is given in the first section of this chapter. In the second section, the structure of a univariate function approximator and a computational procedure are given. The associated learning rate problem will be defined and discussed in the third section. A dynamic way of determining the learning rate will also be included in the third section. Some experimental results and a discussion of the implementation of the proposed structure are given in the last section.

### 3.1. BASIC UNITS

The candidate element has to have the following properties:

1. It is a continuous function which, at least, belongs to $C^1$, that is, the 1st derivative exists. This is due to adopting the neural network structure (see section 2.2).

2. It is a unimodal function, e.g. a bump function.

   This implies that the function has influence only on a certain range of input. As we have stated in section 2.4, we are interested in the localized distributed representation scheme. A bump
function will be activated only in a small range of the input interval.

(3) The shape of the unimodal function can be controlled by some adjustable parameters.

Since this research is focused on developing a system which can "tune" itself over the lifetime of the system, the adjustability of each element is required.

These properties can be achieved by designing a function, say \( g(p_1, p_2; x) \), which is a linear combination of two monotone increasing functions, say \( g_1(p_1; x) \) and \( g_2(p_2; x) \) respectively,

\[
g(p_1, p_2; x) = a_1 g_1(p_1; x) - a_2 g_2(p_2; x), \quad a_1, a_2 \in \mathbb{R}
\]

where \( a_1, a_2 \in \mathbb{R}, p_i \) is the parameter vector of function \( g_i \), and \( x \) is the input variable.

There are some conditions that \( g_1 \) and \( g_2 \) should meet to fulfill the required properties of \( g \).

(1) \( g_1 \) and \( g_2 \) have to be, at least, \( C^1 \) functions.

This requirement will guarantee the property of continuity and the existence of the first derivative of the desired \( g \) function.

(2) \( g_1 \) and \( g_2 \)'s ranges are bounded.

This property will guarantee that the resulting \( g \) function is bounded.
(3) Each $g_i$ ($i=1,2$) can be characterized by its own parameter vector $p_i$ ($i = 1, 2$). And, in this research, we assume that $g_1$ and $g_2$ are in the same parametric family.

(4) Since both $g_1$ and $g_2$ are continuous and monotone increasing functions, we can set $a_1 = a_2 = \alpha$. This will guarantee the resulting $g$ function is an unimodal function.

Many kinds of $g_i$ functions have the above properties. The one most commonly used in neural networks research is the logistic function. There is no special reason to believe that it is the best choice. But the logistic function is chosen in this research for its popularity and, the most importantly, for the nice properties it has.

As we have stated in section 2.2, the output function has the following form.

$$O(w, \theta; x) = \frac{1}{1 + e^{\theta - wx}}$$

(3.1.1)

where $w$ denotes the length of the transition range, and $\theta/w$ denotes the midpoint of the transition range.
The \( g \) function can be written as the following:

\[
g(x) = \alpha \cdot [g_1(w_1, \theta_1; x) - g_2(w_2, \theta_2; x)]
\]  

(3.1.2)

Without loss of generality, we assume \( \alpha = 1 \). Therefore, we can rewrite (3.1.2) as follows:

\[
g(x) = g_1(w_1, \theta_1; x) - g_2(w_2, \theta_2; x)
\]

(3.1.3)

The behavior of \( g(x) \) will be discussed separately according to the different values of \( w_1 \) and \( w_2 \):

(1) \( w_1, w_2 > 0 \)

(a) If \( \theta_1/w_1 < \theta_2/w_2 \), an example of \( g \) is
Fig 3.2. \( \frac{\theta_1}{w_1} < \frac{\theta_2}{w_2} \).
Dashed Line \( g_1: \theta_1=-1.5, w_1=0.5 \), Dash-dotted Line, \( g_2: \theta_2=1.0, w_2=2.0 \),
Solid Line, \( g: g = g_1 - g_2 \).

(b) If \( \frac{\theta_1}{w_1} > \frac{\theta_2}{w_2} \), an example of \( g \) is

Fig 3.3. \( \frac{\theta_1}{w_1} > \frac{\theta_2}{w_2} \).
Dashed Line \( g_1: \theta_1=1.0, w_1=2.0 \), Dash-dotted Line, \( g_2: \theta_2=-1.5, w_2=0.5 \),
Solid Line, \( g: g = g_1 - g_2 \).
(2) $w_1 < 0, w_2 > 0$ or $w_1 > 0, w_2 < 0$

An example of these types of functions is shown in Fig. 3.4.

![Graph showing different types of functions with annotations](image)

Fig 3.4. $w_1 < 0, w_2 > 0$ or $w_1 > 0, w_2 < 0$,
Dashed Line $g_1$: $\theta_1=-4.0, w_1=-0.8$, Dash-dotted Line, $g_2$: $\theta_2=2.0, w_2=1.0$,
Solid line, $g$: $g = g_1 - g_2$.

Notice, these types of function cannot perform as bump functions since they all have influence from $-\infty$ to $+\infty$. 
(3) $w_1, w_2 < 0$

An example of this type of function is illustrated in Fig. 3.5.

![Graph](image)

Fig 3.5. $w_1, w_2 < 0$,
Dashed Line $g_1: \theta_1=-1.5, w_1=-0.5$, Dash-dotted Line, $g_2: \theta_2=1.0, w_2=-2.0$, Solid Line, $g: g = g_1 - g_2$.

This type of function has the similar property as (1), $w_1 > 0$ and $w_2 > 0$.

In order to avoid (2), which does not have local influence property, we require $w_i$ to be positive. This limitation will not degrade the capability of the $g$ function to perform as the desired approximating function.

Although the $g$ function can fulfill the required properties, it has an inherent problem. It cannot approximate an impulse function whose peak is away from the origin.
If the approximated function is an impulse function, the transition range of each $g_i$ would be very short which results in very large $w_i$'s ($= \infty$). In addition, a large $w_i$ will move the center of the transition range toward the origin. Hence, the resulting $g$ will not be very useful in this research. Yet, this problem can be resolved by making the determination of the center of the transition range independent of the $w_i$. The following reparametized logistic function has the desired property.

$$g_i(x) = \frac{1}{1 + e^{w_i(\theta_i - x)}} \quad (3.1.4)$$

where $w_i$ determines the length of the transition range, and $\theta_i$ determines the center of the transition range. Then the resulting $g$ function is as follows:

$$g(x) = \alpha\left[\frac{1}{1 + e^{w_1(\theta_1 - x)}} - \frac{1}{1 + e^{w_2(\theta_2 - x)}}\right] \quad (3.1.5)$$

For the same reason as stated before, $w_1 \geq 0$
One of the objectives of this research is to implement this element using neural network structure. The rest of this section is thus devoted to this attempt.

It turns out that this kind of element can easily be implemented in the neural network structure. Within each single unit (neuron) of the neural network, there are two different nodes which are described as follows:

(1) Linear node:

\[ k \]

\[
\begin{align*}
    Net_k(x) &= \sum_{i=1}^{I} w_{ik} x_i \\
    a_k(t) &= F_k(Net_k(x)) = Net_k(x)
\end{align*}
\]
The output function $O_k$ is defined as the identity function of the activation value $a_k$.

$$O_k(a_k) = a_k \quad (3.1.8)$$

(2) Nonlinear node:

Fig. 3.8. Nonlinear Node.

The input strength function $Net_j(x)$ is defined as that in the linear node.

$$Net_j(x_1, x_2, ..., x_i) = \sum_{i=1}^{i} w_{ij} x_i$$

The activation function $a_j(Net_j(x))$ is defined as a logistic function

$$a_j(Net_j(x)) = F_j(Net_j(x)) = \frac{1}{1+e^{-Net_j(x)}} \quad (3.1.9)$$

The output function $O_j$ is the identity function of the activation value $a_j(Net_j(x))$,

$$O_j(a_j(Net_j(x))) = a_j(Net_j(x))$$
This structure is different from the PDP model in the local memory of each unit. In this research, we assume each node does not have any local memory, i.e. it only responds to the input strength, and there is no memory of the previous activated state of the unit. Based on the above basic nodes, we can construct the basic unit of the network as follows:

This unit actually performs the required mathematical property.

\[ y_j = \alpha \cdot [g_1 - g_2] \]

\[ = \alpha \cdot \left( \frac{1}{1 + e^{w_1(\theta_1 - x)}} - \frac{1}{1 + e^{w_2(\theta_2 - x)}} \right) \]

(3.1.10)

Based on this kind of basic unit, we can construct a single input single output (SISO) model (or univariate model) and develop a computational procedure to update all the weights. This will be given in the next section.
3.2. THE STRUCTURE OF THE UNIVARIATE NETWORK AND THE ASSOCIATED COMPUTATIONAL PROCEDURE

A possible structure is to connect a certain number of units in parallel to form a network. It is shown in Fig. 3.10.

![Diagram of Univariate (SISO) Model](image)

**Fig. 3.10. Univariate (SISO) Model.**

The design concept of this network is based on the localized distributed representation scheme which has been described in the previous chapter. Only some of the units in the network will be activated for each input, and more than one unit will be activated by each input.

There are two important tasks needed to be accomplished here. One task is to investigate the denseness of the family of the constructed kernel functions which the neural network is represented about.
Another task is to develop a computational procedure to modify the weights of the network.

**The Denseness of The Constructed Kernel Function on a Compacta**

Before start to prove the denseness of the family of the constructed kernel functions, some definitions and lemmas should be described in the following:

**Definition.** For any continuous function $G(\cdot)$ mapping $\mathbb{R}$ to $\mathbb{R}$ and $r \in \mathbb{N}$ let $\Sigma(G)$ be the class of functions

$$
\left\{ f: \mathbb{R} \rightarrow \mathbb{R} : f(x) = \sum_{j=1}^{q} \beta_j \cdot G(A_j(x)), x \in \mathbb{R}, \beta_j \in \mathbb{R}, A_j \in A, q = 1, 2, \ldots \right\}
$$

where $A$ is the set of linear functions, and $A_j(x) = a_jx + b_j$.

**Lemma 1.** (Hormik et al, 1989) $\Sigma(\cos)$ is an algebra.

**Proof.**

$$
\{ f: \mathbb{R} \rightarrow \mathbb{R} : f(x) = \sum_{j=1}^{q} \beta_j \cdot \cos(A_j(x)), x \in \mathbb{R}, \beta_j \in \mathbb{R}, A_j \text{ is a set of affine functions} \}
$$

It is obvious that

$$
f_1(x) + f_2(x) \in f
$$

$$
c \cdot f(x) \in f
$$

$$
f_1(x) \cdot f_2(x) = \left[ \sum_{j=1}^{q_1} \beta_j \cos(A_j(x)) \right] \cdot \left[ \sum_{i=1}^{q_2} \beta_i \cos(A_i(x)) \right]
$$

$$
= \sum_{j=1}^{q_1} \sum_{i=1}^{q_2} \beta_j \beta_i \cos(A_j(x)) \cdot \cos(A_i(x))
$$

(Since $\cos(a) \cdot \cos(b) = (1/2) \cos(a+b) + (1/2) \cos(a-b)$)
$= \sum_{j=1}^{q_1} \sum_{i=1}^{q_2} \beta_{i,j} \cos(A_i(x)) \cdot \cos(A_j(x))$

$= \sum_{k=1}^{2q_1q_2} \beta_k \cos(A_k(x)) \in \mathcal{F}$

**Lemma 2.** (Hornik et al, 1989) $\Sigma(\cos)$ is uniformly dense on compacta in $\mathcal{C}$, where $\mathcal{C}$ is the set of continuous functions for $\mathcal{R} \to \mathcal{R}$.

(i) If $x, y \in \mathcal{K}, x \neq y$, then there is an $A$ where $A(x) \neq A(y) + 2n\pi$, $n = 0, 1, 2, \ldots$, then $\cos(A(x)) \neq \cos(A(y))$, i.e. $\Sigma(\cos)$ separates point on $\mathcal{K}$.

(ii) Since $A(x) = ax + b$, let us set $a = 0$ and pick $b \neq \pi/2 + 2n\pi$, $n = \ldots, -2, -1, 0, 1, 2, \ldots$. For all $x \in \mathcal{K}$, $G(A(x)) = G(b) \neq 0$. This ensures that $\Sigma \cos(A(x))$ vanishes at no point of $\mathcal{K}$.

By (i), (ii) and lemma 1 $\Sigma \cos(A(x))$ is uniformly dense on compacta in $\mathcal{C}$.

**Theorem** Let $A = \{ \sum_{j=1}^{q_1} \beta_{j,1} \left[ \frac{1}{(\theta_j - x)\alpha_j} - \frac{1}{(\theta_j - x)\alpha_j^2} \right] \}$,

$\beta_j, \theta_j, \theta_j^2 \in \mathcal{R}, \mathcal{R}^+ = \{ a \mid a \in \mathcal{R}, a > 0 \}$

$\alpha_j, \alpha_j^2 \in \mathcal{R}^+, x \in [L,U], L, U \in \mathcal{R}, q = 1, 2, \ldots \}

$A$ is uniformly dense on $[L,U]$ in $\mathcal{C}$.

**Proof.** Since $\cos(x)$ is a periodic function, we need only to study a period, say $x \in [-\pi/2, 3\pi/2]$, instead of the whole real line.

Let us define $g(x) = \sum_{j=1}^{n} [g_{j,1}(x) - g_{j,2}(x)]$
where \( g_{jl}(x) = \frac{\beta_j}{1 + e^{-\alpha_{j1}(x - \theta_{j1})}} \)

and \( g_{ip}(x) = \frac{\beta_j}{1 + e^{-\alpha_{ip}(x - \theta_{ip})}} \)

Now, let us look at \( x \in [-\pi/2, 0] \). Given \( \varepsilon > 0 \), pick \( n \) such that \( (1/n) < (\varepsilon/2) \).

Let \( j \in \{1, 2, \ldots, n\} \), and \( \beta_j = 1/n \). Since \( g_{jl} \) is a logistic function, we can find \( M_1, M_2 \in \mathbb{R} \), \( M_1 < M_2 \) and \( g_{jl}(M_1) < \varepsilon/2n \), \( g_{jl}(M_2) > 1 - \varepsilon/2n \).

Let us set

\[
\tau_j = \sup \{ x : \cos(x) = j/n \}
\]

Since \( \cos(x) \) is continuous and monotonic on \([-\pi/2, 0]\), such \( \tau_j \)'s exist. Let us take \( x \in [\tau_j, \tau_{j+1}] \)
We can find a \( g_{j1}(x) \) such as

\[
\sup_{x \in [r_j, r_{j+1})} \left| \cos(x) - g_{j1}(x) \right| < \frac{\epsilon}{2} + \frac{ne}{2n} = \epsilon
\]

That means we can always find a finite number (constrained by the given \( \epsilon \)) of \( g_{j1}(x) \) which has the uniform approximation with \( \epsilon \) on \([-\pi/2, 0]\).

It is obvious that the same type of proof will apply on \([0, \pi/2]\) when we are using \(-g_{j2}(x)\) instead of \(g_{j2}(x)\) This is because that \( \cos(x) \) is monotone decreasing in \([0, \pi/2]\). This extends the result that we can find a finite number of \( g_{j1}(x) - g_{j2}(x) \) which can approximate \( \cos(x) \) with \( x \in [-\pi/2, \pi/2] \) by given \( \epsilon > 0 \).

Since \( \cos(x): x \in [-\pi/2, \pi/2] = -\cos(x): x \in [\pi/2, 3\pi/2] \). The result will apply. That means, there exists a finite number of \( -g_{j1}(x) + g_{j2}(x) \) which can approximate \( \cos(x): x \in [\pi/2, 3\pi/2] \) within the given \( \epsilon \).

As we stated before, \( \cos(x) \) is a periodic function. It is obvious that this result can apply on the \( \cos(x), x \in K \) where \( K \) is a compact set. That means we can find a finite sum of

\[
\sum_{j=1}^{n} \beta_j \left[ \frac{1}{1 + e^{-\alpha_{j1}(x - \theta_{j1})}} - \frac{1}{1 + e^{-\alpha_{j2}(x - \theta_{j2})}} \right]
\]

which can approximate \( \cos(x): x \in [a,b] \) as closely as we want.

Since \( \sum \cos(x) \) is dense, this concludes that

\[
\sum_{j=1}^{n} \beta_j \left[ \frac{1}{1 + e^{-\alpha_{j1}(x - \theta_{j1})}} - \frac{1}{1 + e^{-\alpha_{j2}(x - \theta_{j2})}} \right]
\]

is a universal approximator. Q.E.D.
As stated above, another important task is to develop a computational procedure to modify the weights of the network due to the error which is generated by the network. The procedure for the current design is described as follows.

Computational Procedure for Calculating the Updated Weights

Given this structure, we must develop a procedure to modify the weights in order to minimize the mean squared difference between the desired output, $D$, and actual network output, $O_0$. The equations associated with each node of the network can be written as the following:

The objective function: \[ E = (D - O_0)^2 / 2 \]

The output node:
\[
O_0 = \text{Net}(o) \quad \text{Net}(o) = \sum w(o)_j O(h)_j
\]
where \(h\) means hidden layer,
\(o\) means output layer,
\(j\) means the \(j^{th}\) node in the hidden layer

The hidden layer nodes:
\[
O(h)_j = O(h)_{j1} - O(h)_{j2}
\]
\[
O(h)_{j1} = 1/(1 + \exp(-\text{Net}(h)_{j1}))
\]
\[
O(h)_{j2} = 1/(1 + \exp(-\text{Net}(h)_{j2}))
\]
\[
\text{Net}(h)_{j1} = w(h)_{j1} \cdot (x - \theta_{j1})
\]
\[
\text{Net}(h)_{j2} = w(h)_{j2} \cdot (x - \theta_{j2})
\]
where 1 means the \(g_1\) subnode, 2 means the \(g_2\) subnode.
We want to adjust the weights in order to minimize the objective function. The modification rules are as follows.

\[
\frac{\partial E}{\partial w(o)_j} = \frac{\partial E}{\partial O_o} \left( \frac{\partial O_o}{\partial \text{Net}(o)} \right) \left( \frac{\partial \text{Net}(o)}{\partial w(o)_j} \right) \\
= -(D - O_o) \cdot 1 \cdot O(h)_j \\
= -\delta \cdot O(h)_j
\]

where \( \delta = (D - O_o) \)

To get the correct generalization of the delta rule, Rumelhart (Rumelhart, et al, 1986) set \( \Delta w(o)_j = -\frac{\partial E}{\partial w(o)_j} \)

Thus, \( \Delta w(o)_j = \eta(o) \delta O(h)_j \)

\[
\frac{\partial E}{\partial w(h)_j} = \frac{\partial E}{\partial O_o} \left( \frac{\partial O_o}{\partial \text{Net}(o)} \right) \left( \frac{\partial \text{Net}(o)}{\partial w(h)_j} \right) \\
\left( \frac{\partial O(h)_j}{\partial \text{Net}(h)_j} \right) \left( \frac{\partial \text{Net}(h)_j}{\partial w(h)_j} \right) \\
= -\delta \cdot 1 \cdot w(o)_j \cdot 1 \cdot O(h)_j \cdot (1 - O(h)_j) \cdot (x - \theta_{j1}) \\
= -\delta \cdot w(o)_j \cdot O(h)_j \cdot (1 - O(h)_j) \cdot (x - \theta_{j1}) \\
= -\delta(h)_j \cdot (x - \theta_{j1}) \\
\]

where \( \delta(h)_j = \delta \cdot w(o)_j \cdot O(h)_j \cdot (1 - O(h)_j) \)

Thus, \( \Delta w(h)_j = \eta(h)_j \delta(h)_j (x - \theta_{j1}) \)

For the same reason,

\[
\frac{\partial E}{\partial w(h)_2} = \delta(h)_2 (x - \theta_{j2}) \quad \text{and} \quad \Delta w(h)_2 = -\eta(h)_2 \delta(h)_2 (x - \theta_{j2}) \\
\frac{\partial E}{\partial \theta_{j1}} = \delta(h)_j w(h)_j \quad \text{and} \quad \Delta \theta_{j1} = -\eta(h)_j \delta(h)_j w(h)_j \\
\frac{\partial E}{\partial \theta_{j2}} = -\delta(h)_2 w(h)_2 \quad \text{and} \quad \Delta \theta_{j2} = \eta(h)_2 \delta(h)_2 w(h)_2 \\
\]

where \( \eta(o), \eta(h)_j, \eta(h)_2 \) are referred as learning rates.
3.3. DYNAMIC LEARNING RATE

It is still an open question whether the back-propagation method will converge to a fixed set of weights which can "closely" satisfy the training set. One of the crucial factors affecting the property of the convergence is the set of learning rates $\eta(o), \eta(h)_1, \eta(h)_2$ (the rate of convergence is also affected by them). Currently, most researchers assign a fixed value for the learning rate (Fahlman, 1988). It is believed that some information from the status and the performance of the network can be extracted to help to determine the value of the learning rate. In this research, this technique is referred to as the dynamic learning rate to distinguish from the traditional constant learning rates.

Generally speaking, the error back-propagation method is closely related to the gradient descent methods of optimization research. It is different from successive approximation, such as Newton's method, in the sense that it decreases the cost functional continuously from one step to the next. Assume that we want to find a mx1 vector $x_i$ (m=1 is the univariate case) which will minimize the functional $f$ when an initial point $x_1$ is given. The iterations are constructed according to an equation of the form

$$x_{n+1} = x_n + \alpha_n d_n$$

where $\alpha_n$ is a scalar and $d_n$ is a mx1 direction vector. The procedure for selecting the vector $d_n$ varies from technique to technique. But, ideally, once it is chosen the scalar $\alpha_n$ is selected to minimize $f(x_n + \alpha_n d_n)$, where $f(x_n + \alpha_n d_n)$ is regarded as a function of the scalar $\alpha_n$. Generally, things
are arranged so that \( f(x_n + \alpha_n d_n) < f(x_n) \) for small positive \( \alpha_n \). In practice, it is rarely possible to find the minimizing value of \( \alpha_n \) exactly. Therefore, some iterative methods are developed to "search" for the solution instead. However, the new difficulties introduced by the searching methods are those of insuring that the solution has been found, \( f_0 \), is, in fact, the minimum of \( f \) and, the most difficult, that convergence is rapid enough to make the whole scheme practical (Luenberger, 1969).

As we have mentioned earlier, various descent methods exist based on the different ways of determining \( d_n \) vector. The most widely used is the method of steepest descent. This method is applicable to functionals defined on Hilbert space (due to the requirement for an inner product operation). In this method the direction vector \( d^*_n \) at a given point \( x_n \) is chosen to be the negative of the gradient of \( f \) at \( x_n \). If we review the computational procedure of updating the weights of the network, the connection between back-propagation and steepest descent methods can easily be seen. Another commonly used descent method is the conjugate direction method. Generally, the problem of minimizing a quadratic functional on a Hilbert space can be formulated as a Hilbert space minimum norm problem. This approach underlies conjugate direction methods. Unlike the steepest descent method, the conjugate direction method defines the direction vector \( d^\#_n \) as a sequence of orthogonal vectors which span a subspace \( \{ d^1, d^2, \ldots, d^\#_n \} \), where \( n \leq m \). To summarize, the methods of steepest descent and the conjugate direction can be described as follows.

\textbf{Given a quadratic functional}
\[ f(x) = (x | Qx) - 2 (b | x) \]

where \((x | Qx)\) denotes the inner product of \(x\) and \(Qx\) and \(Q\) is an \(m \times m\) self-adjoint positive-definite operator on the Hilbert space \(X\), \(f(x)\) is minimized by the unique vector \(x_0\) satisfying the equation

\[ Qx_0 = b \]

Let us define \(y_n\) as the \(n^{th}\) residual of the approximation where

\[ y_n = b - Qx_n \]

In the method of steepest descent, we set \(d_n^* = y_n\) and \(f\) is of the form

\[ f(x_{n+1}) = f(x_n + \alpha_n y_n) \]

where \(\alpha_n\) is chosen to minimize \(f(x_{n+1})\), where \(f(x_{n+1})\) is

\[ f(x_{n+1}) = ((x_n + \alpha y_n) | Q(x_n + \alpha y_n)) - 2 ((x_n + \alpha y_n) | b) \]
\[ = \alpha^2 (y_n | Qy_n) - 2 \alpha (y_n | y_n) + (x_n | Qx_n) - 2 (x_n | b) \]

Through some straightforward calculation, we can obtain \(\alpha_n\)

\[ \alpha_n = \frac{(y_n | y_n)}{(y_n | Qy_n)} \]

On the other hand, the method of conjugate direction takes the similar form

\[ x_{n+1} = x_n + \alpha_n d_n^* \]

where \(d_1^*, d_2^*, \ldots, d_n^*\) is a sequence of vectors that are orthogonal to each other, i.e. the inner products of all \(n(n-1)/2\) pairs are zeros. Such a sequence is said to be a sequence of conjugate directions. The vector \(x_0\) (which minimize \(f\)) can be expanded in a Fourier series with respect to this sequence. If the \(n^{th}\) partial sum of such an expansion is denoted \(x_n\), then \(\|x_n - x_0\|\) \((= (x_n - x_0)Q(x_n - x_0))\) is minimized over the subspace \([d_1^*, d_2^*, \ldots, d_n^*]\). Therefore, as \(n\) increases, the value of
and the value of $f$ decrease, as with the steepest descent method. Following the same procedure as in the steepest descent method, we can obtain the $\alpha_n$ which minimizes $f(x_{n+1})$ and $\alpha_n$ is

$$\alpha_n = \frac{\langle d_n, y_n \rangle}{\langle d_n, Q d_n \rangle}$$

It is quite clear that the major difference between the steepest descent method and the conjugate direction method is the choice of direction vectors. Both methods follow the same procedure to determine the step size $\alpha$. As with the network computational procedure described above, the step size $\alpha$ and the learning rate $\eta$ play the same important role in convergence behavior.

One of the characteristics of neural networks is that it is implemented in a parallel fashion. That is, it is not desired to include the global information into each local weight computation. Therefore, it is highly desired to evaluate the learning rate by using local information only. In this research, we assume that the learning rate is "evaluated" by each unit, i.e., every link connecting to the same unit shares the same learning rate at each particular time. Thus, we can define $\eta(h)_{j,k}$ as the learning rate of the $k$th node ($k=1,2$) in the $j$th unit of the hidden layer, and $\eta(o)$ is defined as the learning rate of the output layer. Let us rewrite the results of the last section as follows.

$$\Delta w(o)_j = \eta(o) \delta o(h)_j$$
$$\Delta w(h)_{j1} = \eta(h)_{j1} \delta(h)_{j1} (x - \theta_{j1})$$
\[ \Delta \theta_j = -\eta(h)_j \delta(h)_j w(h)_j \]

where \( \delta(h)_j = \delta \cdot w(o)_j \cdot O(h)_j \cdot (1 - O(h)_j) \)

\[ \Delta w(h)_j = -\eta(h)_j \delta(h)_j (x - \theta_j) \]

\[ \Delta \theta_j = \eta(h)_j \delta(h)_j w(h)_j \]

where \( \delta(h)_j = \delta \cdot w(o)_j \cdot O(h)_j \cdot (1 - O(h)_j) \), and

\[ O(h)_j = O(h)_j - O(h)_j \]

**Evaluation of \( \eta(o) \)**

If no modification has been made in the hidden layer, the error is expected to be compensated as the following:

\[ \sum_{j=1}^{m} O(h)_j \cdot \Delta w(o)_j = \delta \]

Since \( \Delta w(o)_j = \eta(o) \delta O(h)_j \)

Thus,

\[ \sum_{j=1}^{m} O(h)_j \cdot \eta(o) \cdot \delta \cdot O(h)_j = \delta \]

\[ \eta(o) \cdot \delta \cdot \sum_{j=1}^{m} O(h)_j^2 = \delta \]

\[ \eta(o) = \frac{1}{\sum_{j=1}^{m} O(h)_j^2} \] (3.3.1)

Since the output node is a linear node, it is possible to determine a \( \eta(o) \) which can compensate the associated error based on the current input.

We are interested not only in compensating for the error based on each input but also in looking for a set of fixed weights which can satisfy the whole set of input/output relations instead of each single I/O pair at each time. It is desired that the weights in the hidden layer should be modified
based on the error. That is, error does not only trigger the modification of
the amplitude, \( w(o) \), of each unit but also updates the structure of that
unit, i.e. \( w(h)_{jk} \) and \( \theta_{jk} \), \( k = 1, 2 \). The easiest way to accomplish this
process is to allow the output layer to compensate for only part of the
error. That is,

\[
\eta(o) = \frac{r}{m} \sum_{j=1}^{m} O(h)_{j}^2 \quad \text{where } 0 < r < 1. \quad (3.3.2)
\]

And, \((1-r)\delta\) will "leak" to the hidden layer. It is not only very difficult
but also impractical to evaluate \( r \) analytically. This is due to the need for
global information in determining the optimal value of \( r \) at each
instance. As stated before, this requirement conflicts with the philosophy
of parallel processing. In this research, we take \( r \) to be a constant, say 0.25.

**Evaluation of \( \eta(h)_{jk}, k=1,2 \)**

It is easy to calculate \( \Delta Net(h)_{jk} \), \( k = 1, 2 \), as follows

\[
\frac{\partial E}{\partial Net(h)_{j1}} = (\frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial Net(o)}) \cdot (\frac{\partial Net(o)}{\partial O(h)_{j1}} \cdot \frac{\partial O(h)_{j1}}{\partial O(h)_{j1}})
\]

\[
= - \delta \cdot 1 \cdot w(o)_{j} \cdot 1 \cdot O(h)_{j1} \cdot (1 - O(h)_{j1})
\]

Thus,

\[
\Delta Net(h)_{j1} = \eta \cdot \delta(h)_{j1}/2 \quad (3.3.3)
\]

As we know

\[
\Delta w(h)_{j1} = \eta(h)_{j1} \delta(h)_{j1} \cdot (x - \theta_{j1}) \quad (3.3.4)
\]

\[
\Delta \theta_{j1} = -\eta(h)_{j1} \delta(h)_{j1} w(h)_{j1} \quad (3.3.5)
\]
And,
\[ \text{Net}(h)_{j1} = w(h)_{j1} (x - \theta_{j1}) \]

Therefore,
\[ \Delta \text{Net}(h)_{j1} = (w(h)_{j1} + \Delta w(h)_{j1}) [x - (\theta_{j1} + \Delta \theta_{j1})] - w(h)_{j1} (x - \theta_{j1}) \]
\[ = \Delta w(h)_{j1} (x - \theta_{j1}) - (w(h)_{j1} + \Delta w(h)_{j1}) \Delta \theta_{j1} \] (3.3.6)

By (3.3.3), (3.3.4), (3.3.5), (3.3.6), and since the feedback error in the jth unit is shared by nodes \( g_1 \) and \( g_2 \)

\[ \eta^* \delta(h)_{j1}/2 = \eta(h)_{j1} \delta(h)_{j1} (x - \theta_{j1})^2 \]
\[ + \left[ w(h)_{j1} + \eta(h)_{j1} \delta(h)_{j1} (x - \theta_{j1}) \right] \eta(h)_{j1} \delta(h)_{j1} w(h)_{j1} \]

If \( \delta(h)_{j1} \neq 0 \),
\[ \eta^*/2 = \eta(h)_{j1}^2 \delta(h)_{j1} w(h)_{j1} (x - \theta_{j1}) + \eta(h)_{j1} [(x - \theta_{j1})^2 + w(h)_{j1}^2] \] (3.3.7)

In this equation, the values of \( \eta(h)_{j1} \) and \( \eta^* \) must be determined.

However, if we assign \( \eta^* \) a fixed value then,
\[ \eta(h)_{j1} = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \] (3.3.8)

where
\[ A = \delta(h)_{j1} w(h)_{j1} (x - \theta_{j1}) \]
\[ B = (x - \theta_{j1})^2 + w(h)_{j1}^2 \]
\[ C = -\eta^*/2 \]

Since \( (x - \theta_{j1})^2 + w(h)_{j1}^2 \geq 0 \) and \( \eta^* > 0 \), it is clear that there are three possible cases for these two roots.

(1) Both roots are positive.

It is clear that we should pick the smaller one as our \( \eta(h)_{j1} \).
(2) Roots have opposite sign.
Following the principle of determining the learning rate, \( \eta(h)_{11} \) should be the one with the positive sign.

(3) Both roots are complex.
This is a highly undesirable case. It makes no sense to have a complex learning rate. However, this problem can be resolved by adjusting the value of \( \eta^* \). If (3.3.7) has complex roots, then

\[
B^2 - 4AC < 0
\]

It is known that \( C = -\eta^*/2 \).
Then,

\[
\frac{\eta^*}{2} > \frac{B^2}{4A} \geq 0.
\]

Thus,

\[
\eta^* = \frac{B^2}{2A}.
\]

will result only one positive root. There is another case associated with these roots. If it is known that \( |A| << B \), a more suitable solution to the learning rate is

\[
\eta(h)_{11} = \frac{\eta^*}{2B} \quad (3.3.9)
\]

The same discussion applies to the determination of \( \eta(h)_{p2} \).
3.4. EXPERIMENTAL RESULTS AND DISCUSSION

Initialization

Since the structural information concerning the observed system is not given in advance, one approach to initialization is to evenly distribute the location of the units within the given input range. As stated before, localized distributed representation is one of the important concepts in this research. This notion can be achieved by overlapping the cover range of each unit a certain amount. In the following experiments, we uniformly divide the input interval, say \([L,U]\), into \(m\) connected small intervals, where \(m\) is the number of units in the network. Then, the two biases of each unit, \((\theta_{j1}, \theta_{j2})\), are taken as \((L+(j-2)(U-L)/m)\) and \((L+(j+2)(U-L)/m)\) respectively. The units in each pair of neighbors take values from the two intervals \((L+(j-3)(U-L)/m, L+(j+1)(U-L)/m)\), and \((L+(j-1)(U-L)/m, L+(j+3)(U-L)/m)\). The \(w(h)_{j1}'s\), \(w(h)_{j2}'s\), and \(w(o)'s\) are randomly generated from an inverse c.d.f of the continuous uniform distribution sampled between \(a\) and \(b\). Since the \(w(h)_{j1}'s\), and \(w(h)_{j2}'s\) are constrained to have positive values, and because of the localized property of each unit, we take \(a = 3\), and \(b = 6\). However, the \(w(o)'s\) do not have this positive constraint. We thus take \(a = -2\), and \(b = 2\). Experience from the experiments suggested that the performance of the network will not be strongly affected by variation in the values of \(a\) and \(b\). However, extremal values (either very large or very small) of \(a\) and \(b\) do cause problems. It is easy to see the reason for this. The localized distributed representation notion can only be achieved when the \(w(h)_{jk}'s\) (\(k = 1, 2\)) are moderate. Small values of the \(w(h)_{jk}'s\) form a distributed
representation scheme while large values of the $w(h)_{jk}$'s form a local representation scheme.

Index of Accuracy

The index of accuracy is defined to be the root mean square error (RMSE) divided by a constant scale factor, which defined as the standard deviation of the training data. It is necessary to remove the scale dependence of the data. Dividing RMSE by the standard derivation of the training data provides a measure to use. Thus, the resulting "index of accuracy" is insensitive to the dynamic range of the input data.

Convergence Test

It is known that convergence of the error back-propagation weights updating procedure is not guaranteed. Although the convergence of the network developed in this research cannot be guaranteed either, eight different experiments were conducted to test the convergence property. The experiments were designed to assess the convergence behavior of the network. Given a predetermined function, we take a number of equally spaced points, say 400, from the allowable input range and calculate the associated output values. Thus, we have 400 training pairs. The set of these 400 training pairs is called one training trial. In this part of experiment, we construct both 50 and 200 unit networks based on the structure which has been described previously. Since each unit has 5 parameters, a 50 unit network has 250 adjustable parameters which is smaller than the number of training pairs. However, a 200 unit network has 1000 adjustable parameters. We display the results of these two
experiments together to show the convergence behavior of the network. For each training trial, we feed the 400 training pairs to the network in random order. Each experiment has two stages. In the first stage, the network compares the desired output with the actual output and then applies the back propagation procedure to modify the weights after each training pair. After the 400 training pairs have been fed into the network, we start the second stage. In the second stage, we are only interested in the performance of the network. Therefore, no weight modification is conducted in this stage. The accuracy index and the root mean squared error are calculated as the performance index of the network at this stage. In the following experiments, we display the following information: (1) the plot of accuracy index vs. training trial of each network; (2) the tables of accuracy index and RMSE at trial 5, 10, 25, 50, 75 and 100 for each network; and (3) the performance of the networks after 5, 10, 50, and 100 trials. A description of each experiment follows.

The input is bounded on [-10,10] unless otherwise specified.

**Case 1: y = cos(x) (Fig. 3.11. - Fig. 3.15.)**

Since our network has been proved that it can approximate cosine function as closely as desired, this experiment demonstrates convergence behavior under the constraint that only a finite number of units are available.
**Case 2:** \( y = \sin(x) - \cos(x^2), \ x \in [-1,4] \) (Fig. 3.16. - Fig. 3.20.)

It is known that this case has highly nonlinear behavior around the origin. This experiment displays the ability of our network to approximate this highly nonlinear function.

**Case 3:** \( y = 0.8 \times + 1.5 \) (Fig. 3.21. - Fig. 3.25.)

Although this research is aimed at developing a method to approximate nonlinear functions by an iterative learning procedure, it is highly desirable that the method should be rich enough to approximate linear functions, as well. In this case, we demonstrate that our network has the capability to perform as a linear approximator.

\[
\begin{cases}
-2 & \text{if } -10 \leq x < -5 \\
\left(\frac{10}{9}\right)(x + 5) - 2 & \text{if } -5 \leq x \leq 4 \\
8 & \text{if } 4 < x \leq 10
\end{cases}
\]

**Case 4:** \( y = \begin{cases}
-2 & \text{if } -10 \leq x < -8 \\
x + 6 & \text{if } -8 \leq x < -2 \\
4 & \text{if } -2 \leq x < 4 \\
\left(\frac{6}{5}\right)(x - 4) + 4 & \text{if } 4 \leq x \leq 9 \\
10 & \text{if } 9 \leq x \leq 10
\end{cases} \) (Fig. 3.26. - Fig. 3.30.)

Since the network unit is constructed by the family of transcendental functions, known as \( C^\infty \) functions, it is interesting to see the capability of this network to approximate the functions which is belong to \( C^0 \), say, the ramp function in this experiment.

The performance of the network is very good in this experiment.
This is a multi-level ramp function. It is also interesting to see how two break points can influence the behavior of the network.

\[
\begin{align*}
\text{Case 6: } y &= \begin{cases} 
0 & \text{if } -10 \leq x < -5 \\
\frac{(3x + 15)}{7} & \text{if } -5 \leq x < 2 \\
4 - \frac{(x)}{2} & \text{if } 2 \leq x < 8 \\
0 & \text{if } 8 \leq x \leq 10
\end{cases}
\end{align*}
\]

(Fig. 3.36. - Fig. 3.40.)

This is a triangle function. The particular interest in this kind of function is due to the peak that it generates. It is expected that this kind of function cannot be represented by a linear combination of extremly smooth functions. However, the function approximator, which is developed in this research can still achieve the accuracy of 0.015.

\[
\begin{align*}
\text{Case 7: } y &= \begin{cases} 
-1 & \text{if } -10 \leq x < 2 \\
3 & \text{if } 2 \leq x \leq 10
\end{cases}
\end{align*}
\]

(Fig. 3.41. - Fig. 3.45.)

This is known as a step function which is discontinuous in nature. We can see that there is a relative large error around the discontinuous point. However, the network can still approximate this function within a reasonable index of accuracy 0.09.

\[y = 3.95x(1-x), \quad x \in [0,1].\] (Fig. 3.46. - Fig. 3.48.)

This is a function which has chaotic behavior. Currently, a great deal of research has involved the study of a deterministic equation with similarly random behavior. Chaos theory suggests that some of the randomness which we experience in the real world is generated from a deterministic structure. It is not the purpose of this research to investigate along this direction. However, it is
quite interesting to see the capability of our network to approximate chaotic behavior. In this experiment, we only display the behavior of the networks after 5 trials and 100 trials respectively. Notice that the networks performed quite well in approximating this chaotic function.

Dynamic Learning

The objective of this research was to construct a neural network which can iteratively tune its weights in order to approximate the behavior of the observed system. One of the characteristics of this approach is that the network will learn the behavior of the system based on the information that it received from the system. In the convergence test, we assumed that we have a set of uniformly distributed data on a predefined interval. This is unlikely to be the case in practice. It is quite important to test the performance of the network without the assumption of uniformly distributed data.

In this set of experiments, we use a 200 unit network to approximate a predetermined function. We use a random number generator to generate the input data and calculate the corresponding output value from the predetermined function. This input data and the corresponding output data form a training pair. Then, we feed this training pair to the network and the difference between the desired output and the actual output is used to update the weights of the
network. It is expected that the performance of the network will improve as more training pairs are processed.

Since there is no control over the training set, the monotone behavior of the accuracy index as a function of numbers of training pairs is not expected. However, a generally downward trend should be expected to be observed in the plot.

In this set of experiments, we tested four cases. They are: (1) cosine function, \( y = \cos(x) \) (Fig. 3.49 - Fig. 3.50); (2) nonlinear function, \( y = \sin(x) - \cos(x^2) \) (Fig. 3.51 - Fig. 3.52); (3) linear function, \( y = 0.8x + 1.5 \) (Fig. 3.53 - Fig. 3.54); and (4) ramp function (Fig. 3.55 - Fig. 3.56).

The reason for the fluctuation in the training pairs versus accuracy index plot is due to the nature of the distributed representation. As we have described before, the network is designed based on the localized distributed representation scheme. Every individual input will affect some of the corresponding units' weights. In another words, the existing memory of the network will be affected due to the new inputs. However, this changes the behavior of the network to the old inputs to a certain extent.

These experimental results of the convergence test and the dynamic learning show the relationship between the learning performance and the training environment. This relationship is quite well-known in the field of human learning research. Although it is
impossible to draw any conclusion at this point, it may suggest a very vague similarity between the brain and the artificial neural networks.
Fig. 3.11. $y=\cos(x)$, Index of Accuracy vs. Trials.
Dash-dotted Line is the 50 Unit Network.
Solid Line is the 200 Unit Network.

Table 3.1. The Index of Accuracy (I.A.) and RMSE of $y=\cos(x)$.

<table>
<thead>
<tr>
<th># of trial</th>
<th>50 units</th>
<th>200 units</th>
</tr>
</thead>
<tbody>
<tr>
<td># of trial</td>
<td>Index of Accuracy</td>
<td>RMSE</td>
</tr>
<tr>
<td>5</td>
<td>.1523</td>
<td>.1099</td>
</tr>
<tr>
<td>10</td>
<td>.1043</td>
<td>.0753</td>
</tr>
<tr>
<td>25</td>
<td>.0528</td>
<td>.0381</td>
</tr>
<tr>
<td>50</td>
<td>.0323</td>
<td>.0233</td>
</tr>
<tr>
<td>75</td>
<td>.0259</td>
<td>.0187</td>
</tr>
<tr>
<td>100</td>
<td>.0225</td>
<td>.0162</td>
</tr>
</tbody>
</table>
Fig. 3.12. $y = \cos(x)$ at Trial 5.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.13. $y = \cos(x)$ at Trial 10.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.14. $y = \cos(x)$ at Trial 50.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.15. $y = \cos(x)$ at Trial 100.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.16. $y = \sin(x) - \cos(x^2)$, Index of Accuracy vs. Trials.
Dash-dotted Line is the 50 Unit Network.
Solid Line is the 200 Unit Network.

Table 3.2. The Index of Accuracy (I.A.) and RMSE of $y = \sin(x) - \cos(x^2)$.

<table>
<thead>
<tr>
<th># of trial</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
<th># of trial</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.0727</td>
<td>0.0772</td>
<td>5</td>
<td>0.0576</td>
<td>0.0613</td>
</tr>
<tr>
<td>10</td>
<td>0.0633</td>
<td>0.0673</td>
<td>10</td>
<td>0.0479</td>
<td>0.0509</td>
</tr>
<tr>
<td>25</td>
<td>0.0506</td>
<td>0.0537</td>
<td>25</td>
<td>0.0360</td>
<td>0.0383</td>
</tr>
<tr>
<td>50</td>
<td>0.0462</td>
<td>0.0491</td>
<td>50</td>
<td>0.0272</td>
<td>0.0289</td>
</tr>
<tr>
<td>75</td>
<td>0.0459</td>
<td>0.0488</td>
<td>75</td>
<td>0.0219</td>
<td>0.0232</td>
</tr>
<tr>
<td>100</td>
<td>0.0447</td>
<td>0.0475</td>
<td>100</td>
<td>0.0184</td>
<td>0.0195</td>
</tr>
</tbody>
</table>
Fig. 3.17. $y = \sin(x) - \cos(x^2)$ at Trial 5.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.18. $y = \sin(x) - \cos(x^2)$ at Trial 10.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.19. $y = \sin(x) - \cos(x^2)$ at Trial 50.

(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.20. $y = \sin(x) - \cos(x^2)$ at Trial 100.
(a) 50 Unit Network,  (b) 200 Unit Network.
Fig. 3.21. $y=0.8(x)+1.5$, Index of Accuracy vs. Trials.
Dash-dotted Line is the 50 Unit Network.
Solid Line is the 200 Unit Network.

Table 3.3. The Index of Accuracy (I.A.) and RMSE of $y=0.8(x)+1.5$.

<table>
<thead>
<tr>
<th></th>
<th>50 units</th>
<th>200 units</th>
</tr>
</thead>
<tbody>
<tr>
<td># of trial</td>
<td>Index of Accuracy</td>
<td>RMSE</td>
</tr>
<tr>
<td>5</td>
<td>.0251</td>
<td>.1160</td>
</tr>
<tr>
<td>10</td>
<td>.0183</td>
<td>.0845</td>
</tr>
<tr>
<td>25</td>
<td>.0121</td>
<td>.0559</td>
</tr>
<tr>
<td>50</td>
<td>.0088</td>
<td>.0409</td>
</tr>
<tr>
<td>75</td>
<td>.0063</td>
<td>.0292</td>
</tr>
<tr>
<td>100</td>
<td>.0046</td>
<td>.0213</td>
</tr>
</tbody>
</table>
Fig. 3.22. $y = 0.8x + 1.5$ at Trial 5.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.23. $y = 0.8x + 1.5$ at Trial 10.
(a) 50 Unit Network,  (b) 200 Unit Network.
Fig. 3.24. $y = 0.8x + 1.5$ at Trial 50.
(a) 50 Unit Network,  (b) 200 Unit Network.
Fig. 3.25. \( y = 0.8x + 1.5 \) at Trial 100.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.26. Ramp Function, Index of Accuracy vs. Trials.  
Dash-dotted Line is the 50 Unit Network.  
Solid Line is the 200 Unit Network.

Table 3.4. The Index of Accuracy (I.A.) and RMSE of Ramp Function

<table>
<thead>
<tr>
<th># of trial</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>.0243</td>
<td>.1015</td>
</tr>
<tr>
<td>10</td>
<td>.0185</td>
<td>.0772</td>
</tr>
<tr>
<td>25</td>
<td>.0126</td>
<td>.0526</td>
</tr>
<tr>
<td>50</td>
<td>.0094</td>
<td>.0395</td>
</tr>
<tr>
<td>75</td>
<td>.0071</td>
<td>.0299</td>
</tr>
<tr>
<td>100</td>
<td>.0053</td>
<td>.0221</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># of trial</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>.0160</td>
<td>.0669</td>
</tr>
<tr>
<td>10</td>
<td>.0109</td>
<td>.0454</td>
</tr>
<tr>
<td>25</td>
<td>.0066</td>
<td>.0278</td>
</tr>
<tr>
<td>50</td>
<td>.0046</td>
<td>.0190</td>
</tr>
<tr>
<td>75</td>
<td>.0036</td>
<td>.0152</td>
</tr>
<tr>
<td>100</td>
<td>.0031</td>
<td>.0129</td>
</tr>
</tbody>
</table>
Fig. 3.27. Ramp Function at Trial 5.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.28. Ramp Function at Trial 10.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.29. Ramp Function at Trial 50.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.30. Ramp Function at Trial 100.
(a) 50 Unit Network,  (b) 200 Unit Network.
Fig. 3.31. Multi-ramp Function, Index of Accuracy vs. Trials.
Dash-dotted Line is the 50 Unit Network.
Solid Line is the 200 Unit Network.

Table 3.5. The Index of Accuracy (I.A.) and RMSE of Multi-ramp Function

<table>
<thead>
<tr>
<th># of trial</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
<th># of trial</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>.0284</td>
<td>.0977</td>
<td>5</td>
<td>.0212</td>
<td>.0730</td>
</tr>
<tr>
<td>10</td>
<td>.0196</td>
<td>.0674</td>
<td>10</td>
<td>.0151</td>
<td>.0518</td>
</tr>
<tr>
<td>25</td>
<td>.0119</td>
<td>.0410</td>
<td>25</td>
<td>.0096</td>
<td>.0329</td>
</tr>
<tr>
<td>50</td>
<td>.0077</td>
<td>.0263</td>
<td>50</td>
<td>.0064</td>
<td>.0219</td>
</tr>
<tr>
<td>75</td>
<td>.0059</td>
<td>.0204</td>
<td>75</td>
<td>.0051</td>
<td>.0175</td>
</tr>
<tr>
<td>100</td>
<td>.0051</td>
<td>.0176</td>
<td>100</td>
<td>.0044</td>
<td>.0151</td>
</tr>
</tbody>
</table>
Fig. 3.32. Multi-ramp Function at Trial 5.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.33. Multi-ramp Function at Trial 10.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.34. Multi-ramp Function at Trial 50.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.35. Multi-ramp Function at Trial 100.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.36. Triangle Function, Index of Accuracy vs. Trials.
Dash-dotted Line is the 50 Unit Network.
Solid Line is the 200 Unit Network.

Table 3.6. The Index of Accuracy (I.A.) and RMSE of Triangle Function.

<table>
<thead>
<tr>
<th># of trial</th>
<th>50 units</th>
<th>200 units</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Index of Accuracy</td>
<td>RMSE</td>
</tr>
<tr>
<td>5</td>
<td>.1069</td>
<td>.1070</td>
</tr>
<tr>
<td>10</td>
<td>.0696</td>
<td>.0696</td>
</tr>
<tr>
<td>25</td>
<td>.0329</td>
<td>.0329</td>
</tr>
<tr>
<td>50</td>
<td>.0191</td>
<td>.0191</td>
</tr>
<tr>
<td>75</td>
<td>.0153</td>
<td>.0154</td>
</tr>
<tr>
<td>100</td>
<td>.0135</td>
<td>.0135</td>
</tr>
</tbody>
</table>
Fig. 3.37. Triangle Function at Trial 5.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.38. Triangle Function at Trial 10.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.39. Triangle Function at Trial 50.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.40. Triangle Function at Trial 100.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.41. Step Function, Index of Accuracy vs. Trials.
Dash-dotted Line is the 50 Unit Network.
Solid Line is the 200 Unit Network.

Table 3.7. The Index of Accuracy (I.A.) and RMSE of Step Function.

<table>
<thead>
<tr>
<th># of trial</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
<th># of trial</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>.1306</td>
<td>.2562</td>
<td>5</td>
<td>.0975</td>
<td>.1912</td>
</tr>
<tr>
<td>10</td>
<td>.1191</td>
<td>.2336</td>
<td>10</td>
<td>.0931</td>
<td>.1825</td>
</tr>
<tr>
<td>25</td>
<td>.1066</td>
<td>.2090</td>
<td>25</td>
<td>.0905</td>
<td>.1774</td>
</tr>
<tr>
<td>50</td>
<td>.0995</td>
<td>.1951</td>
<td>50</td>
<td>.0897</td>
<td>.1756</td>
</tr>
<tr>
<td>75</td>
<td>.0963</td>
<td>.01889</td>
<td>75</td>
<td>.0893</td>
<td>.1751</td>
</tr>
<tr>
<td>100</td>
<td>.0944</td>
<td>.01851</td>
<td>100</td>
<td>.0889</td>
<td>.1744</td>
</tr>
</tbody>
</table>
Fig. 3.42. Step Function at Trial 5.
(a) 50 Unit Network,  (b) 200 Unit Network.
Fig. 3.43. Step Function at Trial 10.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.44. Step Function at Trial 50.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.45. Step Function at Trial 100.
(a) 50 Unit Network, (b) 200 Unit Network.
Fig. 3.46. Chaotic Function, Index of Accuracy vs. Trials.
Dash-dotted Line is the 50 Unit Network.
Solid Line is the 200 Unit Network.

Table 3.8. The Index of Accuracy (I.A.) and RMSE of Chaotic Function.

<table>
<thead>
<tr>
<th># of trial</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>.0526</td>
<td>.0164</td>
</tr>
<tr>
<td>10</td>
<td>.0341</td>
<td>.0106</td>
</tr>
<tr>
<td>25</td>
<td>.0223</td>
<td>.0069</td>
</tr>
<tr>
<td>50</td>
<td>.0149</td>
<td>.0046</td>
</tr>
<tr>
<td>75</td>
<td>.0123</td>
<td>.0038</td>
</tr>
<tr>
<td>100</td>
<td>.0112</td>
<td>.0035</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># of trial</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>.0302</td>
<td>.0094</td>
</tr>
<tr>
<td>10</td>
<td>.0207</td>
<td>.0064</td>
</tr>
<tr>
<td>25</td>
<td>.0155</td>
<td>.0084</td>
</tr>
<tr>
<td>50</td>
<td>.0117</td>
<td>.0036</td>
</tr>
<tr>
<td>75</td>
<td>.0099</td>
<td>.0031</td>
</tr>
<tr>
<td>100</td>
<td>.0089</td>
<td>.0028</td>
</tr>
</tbody>
</table>
Fig. 3.47. Chaotic Function at Trial 5.
(a) 50 Unit Network, (b) 200 Unit Network.
Input is the Output at Time (t-1).
Fig. 3.47. (Continued).
(a) 50 Unit Network, (b) 200 Unit Network.
Input is the Output at Time (t-1).
Fig. 3.47. (Continued).
(a) 50 Unit Network,  (b) 200 Unit Network.
Input is the Output at Time (t-1).
Fig. 3.47. (Continued).
(a) 50 Unit Network, (b) 200 Unit Network.
Input is the Output at Time (t-1).
Fig. 3.48. Chaotic Function at Trial 100.
(a) 50 Unit Network,  (b) 200 Unit Network.
Input is the Output at Time (t-1).
Fig. 3.48. (Continued).
(a) 50 Unit Network, (b) 200 Unit Network.
Input is the Output at Time (t-1).
Fig. 3.48. (Continued).
(a) 50 Unit Network,  (b) 200 Unit Network.
Input is the Output at Time (t-1).
Fig. 3.48. (Continued).
(a) 50 Unit Network, (b) 200 Unit Network.
Input is the Output at Time (t-1).
Fig. 3.49. \( y = \cos(x) \), Index of Accuracy vs. Training Points.

Table 3.9. The Index of Accuracy and RMSE of \( y = \cos(x) \) via Dynamic Learning

<table>
<thead>
<tr>
<th># of points</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>.2967</td>
<td>.2143</td>
</tr>
<tr>
<td>10</td>
<td>.1710</td>
<td>.1235</td>
</tr>
<tr>
<td>25</td>
<td>.1146</td>
<td>.0828</td>
</tr>
<tr>
<td>50</td>
<td>.0688</td>
<td>.0497</td>
</tr>
<tr>
<td>75</td>
<td>.0565</td>
<td>.0408</td>
</tr>
<tr>
<td>100</td>
<td>.0427</td>
<td>.0308</td>
</tr>
</tbody>
</table>

200 units
Fig. 3.50. \( y = \cos(x) \).
(a) After 200 Points, (b) After 500 Points.
Fig. 3.50. (Continued).
(c) After 1000 Points, (d) After 5000 Points.
Fig. 3.51. $y = \sin(x) - \cos(x^2)$, Index of Accuracy vs. Training Points.

Table 3.10. The Index of Accuracy and RMSE of $y = \sin(x) - \cos(x^2)$ via Dynamic Learning.

<table>
<thead>
<tr>
<th># of points</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>.0903</td>
<td>.0697</td>
</tr>
<tr>
<td>10</td>
<td>.0571</td>
<td>.0440</td>
</tr>
<tr>
<td>25</td>
<td>.0439</td>
<td>.0339</td>
</tr>
<tr>
<td>50</td>
<td>.0312</td>
<td>.0241</td>
</tr>
<tr>
<td>75</td>
<td>.0180</td>
<td>.0139</td>
</tr>
<tr>
<td>100</td>
<td>.0167</td>
<td>.0129</td>
</tr>
</tbody>
</table>
Fig. 3.52. $y = \sin(x) - \cos(x^2)$.
(a) After 200 Points, (b) After 500 Points.
Fig. 3.52. (Continued).
(c) After 1000 Points, (d) After 5000 Points.
Fig. 3.53. $y=0.8x+1.5$, Index of Accuracy vs. Training Points.

Table 3.11. The Index of Accuracy and RMSE of $y=0.8x+1.5$
via Dynamic Learning.

<table>
<thead>
<tr>
<th># of points</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>.0571</td>
<td>.2657</td>
</tr>
<tr>
<td>10</td>
<td>.0360</td>
<td>.1675</td>
</tr>
<tr>
<td>25</td>
<td>.0261</td>
<td>.1215</td>
</tr>
<tr>
<td>50</td>
<td>.0216</td>
<td>.0926</td>
</tr>
<tr>
<td>75</td>
<td>.0216</td>
<td>.1006</td>
</tr>
<tr>
<td>100</td>
<td>.0142</td>
<td>.0659</td>
</tr>
</tbody>
</table>
Fig. 3.54. $y = 0.8x + 1.5$.
(a) After 200 Points, (b) After 500 Points.
Fig. 3.54. (Continued).
(c) After 1000 Points, (d) After 5000 Points.
Fig. 3.55. Ramp Function, Index of Accuracy vs. Training Points.

Table 3.12. The Index of Accuracy and RMSE of Ramp Function via Dynamic Learning.

<table>
<thead>
<tr>
<th># of points x50</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>.0637</td>
<td>.2664</td>
</tr>
<tr>
<td>10</td>
<td>.0291</td>
<td>.1216</td>
</tr>
<tr>
<td>25</td>
<td>.0173</td>
<td>.0725</td>
</tr>
<tr>
<td>50</td>
<td>.0111</td>
<td>.0463</td>
</tr>
<tr>
<td>75</td>
<td>.0093</td>
<td>.0390</td>
</tr>
<tr>
<td>100</td>
<td>.0077</td>
<td>.0322</td>
</tr>
</tbody>
</table>
Fig. 3.56. Ramp Function.
(a) After 200 Points, (b) After 500 Points.
Fig. 3.56. (Continued).
(c) After 1000 Points, (d) After 5000 Points.
CHAPTER IV
MULTIVARIATE MODEL

In real life, most interesting problems are multivariate rather than univariate. It is well-known that multivariate problems are usually much harder than univariate ones. Nowadays, most multivariate problems are solved under the assumption that the structural information (or the functional form) is known. This will reduce the problem to that of estimating a set of parameters, and will have the same disadvantages as have been mentioned in Chapter I. To the extent that the model is correct, such an approach can be successful. Unfortunately, structural correctness is difficult to predict and to verify in practice. And, incorrect structural assumptions can yield misleading results.

Furthermore, it is known that multivariate cases suffer from "the curse of dimensionality" (Bellman, 1961). That is, the amount of information has to be increased in order to achieve the same amount of "knowledge" about the observed system. For example, if \( y = f(x) \) is specified over \([0,1]\) by its values at the points \( k(0.01), k = 0, 1, 2, \ldots, 99 \), we must tabulate 100 values. If \( y = f(x_1, x_2) \) is specified over \( 0 \leq x_1, x_2 \leq 1 \) by a grid of similar type in \( x_1, x_2 \), we now require \( 100 \times 100 = 10^4 \) values. It is easy to see that we need \( 10^6 \) values to determine a function with three
variables. Although, in theoretical development, we can use vector representation to derive similar results for both univariate and multivariate cases, there are still tremendous difficulties in implementing the multivariate approximations. This is exactly what Bellman has said "a malediction that has plagued the scientist from earliest days." (Bellman, 1961).

One possible way to escape from the "curse of dimensionality" is to accomplish multivariate function approximation by combining functions with fewer variables. This is equivalent to asking whether a multivariate function can be represented as a superposition of functions of fewer variables. The research on the representability of several variables functions in terms of prescribed superpositions of functions of fewer variables was initiated by Hilbert. A description of Hilbert's 13th problem and a remarkable answer to this problem by Kolmogorov are provided in the first section. The discussion of Kolmogorov's theorem and its implementation by neural networks structure are stated in the first section. A discussion of the work that has been done and the related future research are given at the end of this chapter.

4.1. THE THEORY OF REPRESENTATION FUNCTIONS OF SEVERAL VARIABLES BY FUNCTIONS OF ONE VARIABLE

Given a set of behavior data, say
\[ D = \{(x_{i1}, x_{i2}, \ldots, x_{in}, y) \}, \text{i,n}=1,2,\ldots \], how can we find the relation associated with the x's (the inputs) and y (the output)? This is not a new
question. For a long time, scientists and engineers have tried to develop methods to resolve this problem. Along with these long term attempts, there have been a few principles pointed out by mathematicians. Unfortunately, these principles do not offer enough information for current technology to implement them.

**Hilbert's 13th Problem**

The idea of representing functions of $n$ variables as superpositions of functions of $m < n$ variables for the purpose of studying the structure of function classes is due to Hilbert. The thirteenth of his celebrated twenty three problems (Lorentz, 1976) is the following conjecture:

"prove that the equation of the seventh degree $x^7 + ax^3 + bx^2 + cx + 1 = 0$ is not solvable with the help of any continuous functions of only two variables."

This conjecture is algebraic in origin. It emerged out of the attempts to eliminate the largest possible number of coefficients from polynomial equations $\sum_{k=0}^{n} a_k x^k = 0$, thereby expressing their roots, regarded as functions of $n+1$ coefficients, as functions of fewer coefficients. However, this problem could be interpreted as a more general one, such as (Lorentz, 1976)

"prove that there are continuous function of three variables, not representable by continuous functions of two variables."

Basically, Hilbert asked if there are genuine multivariate functions. Of course, $x + y$, for example, is a function of two variables but $xy = e^{\log(x)+\log(y)}$ is a linear superposition of single variable functions and
addition. Fifty seven years after Hilbert proposed his famous 13th problem, Kolmogorov (Kolmogorov, 1957) and Arnol'd (Arnol'd, 1956) resolved this problem. They proved that Hilbert's conjecture is wrong.

**Kolmogorov's Theorem**

Let $E^n$ be the n-fold cartesian product $E \times E \times \cdots \times E$ of the unit intervals $E = [0,1]$, we let $C(E^n)$ be the Banach space of real valued continuous functions defined on $E^n$. Then, the remarkable Kolmogorov Theorem can be stated as follows.

**Theorem: (Kolmogorov Representation Theorem)**

There exists a fixed collection of continuous increasing functions $\Phi_{pq}(x)$, which is independent to the approximated function, on $E = [0,1]$, so that each continuous function $f \in E^n$ can be written in the form

$$
 f(x_1, x_2, \ldots, x_n) = \sum_{q=1}^{2n+1} G_q( \sum_{p=1}^{n} \Phi_{pq}(x_p))
$$

where $G_q$ are properly chosen continuous functions of one variable.

The schematic interpretation of this theorem is illustrated in Fig. 4.1.

In order to obtain a geometric interpretation of Kolmogorov's theorem, let us assume that the mapping from n-dimensional space to the $(2n+1)$-dimensional space given by

$$
 Z_q(x_1, x_2, \ldots, x_n) = \Phi_{1q}(x_1) + \Phi_{2q}(x_2) + \cdots + \Phi_{nq}(x_n), \quad q = 1, 2, \ldots, 2n+1
$$
Fig. 4.1. Kolmogorov's Theorem.
This mapping is continuous and one-to-one. (Otherwise, there would exist two points in n-dimensional space which are not distinguishable in the (2n+1)-dimensional space through the mapping $Z_q$. Then, all functions that can be represented by (4.1.2) will coincide at these two points, and the representation (4.1.1) will be impossible for some function $f \in C(E^n)$.

Since the domain $N$ (n-dimensional space) is compact, its image $M$ ((2n+1)-dimensional space) under (4.1.2) is compact and (4.1.2) is a homeomorphism between $N$ and $M$. Therefore, there is one-to-one correspondence between all continuous functions $f(x_1, x_2, \ldots, x_n)$ on $N$ and all continuous functions $F(Z_1, Z_2, \ldots, Z_{2n+1})$ on $M$. This generalized result has been shown by Tihomirov (Vitushkin, 1978): for any compact $K$ of dimension $n$ there exists a homeomorphic embedding, $Z(x) = (Z_1(x), Z_2(x), \ldots, Z_{2n+1}(x)), x \in K$ into (2n+1)-dimensional euclidean space such that any continuous function $f(x)$ on $K$ can be represented in the form

$$f(x) = \sum_{q=1}^{2n+1} G_q(Z_q(x))$$

where $G_q$'s are continuous functions of one variable.

Some researchers have improved upon Kolmogorov's result. Lorentz (Lorentz, 1965) is the first one to observe that one can replace the $G_q$ by one single function $G$. That is,

$$f(x) = \sum_{q=1}^{2n+1} G(Z_q(x))$$

(4.1.3)

The schematic interpretation is shown in Fig. 4.2.
Fig. 4.2. Lorentz's Modification.
Furthermore, Sprencher (Sprencher, 1966) has shown that Kolmogorov's theorem can be proved with a function \( \Phi_q \) multiplied by constants \( \lambda_q \). Specifically, for each integer \( n \geq 2 \) there exists a monotonic increasing function \( \Phi \in \mathcal{C}(E) \) such that each \( f \in \mathcal{C}(E^n) \) has a representation of the form

\[
f(x_1, x_2, \ldots, x_n) = \sum_{q=1}^{2n+1} \sum_{p=1}^{n} a_{p,q} \Phi_q(x_p)
\]

where the constants \( \lambda_1, \lambda_2, \ldots, \lambda_n \) are linearly independent over the field of rationals. The schematic representation of this theorem is illustrated in Fig. 4.3.

**4.2. THE CONSTRUCTION OF A MULTIVARIATE MODEL BASED ON KOLMOGOROV THEOREM AND EXPERIMENTAL RESULTS**

In this section we present a detailed description of the Kolmogorov representation theorem, especially relating to the number of summands, the external functions \( G_q \) and the internal functions \( \Phi_{pq} \).

**4.2.1. The Number of Summands**

Before we start to discuss the details about the \( G_q \) (so-called external functions) and \( \Phi_{pq} \) (so-called internal functions), it is necessary to examine the number of summands that Kolmogorov has assigned in his representation theorem. Kolmogorov specified this number as \( 2n+1 \). The reason for his assigning this value has a long history. Whitney (Whitney, 1935) proved that a smooth \( n \)-dimensional manifold, which is
Fig. 4.3. Sprecher's Modification.
compact, may be embedded in $\mathbb{R}^{2n+1}$. This result has been frequently implemented in dissipative dynamic system analysis. However, it is very difficult to determine $n$ in those problems. The dimensionality of the attractor is the solution to be found rather than known a priori. However, Kolmogorov inverted Whitney's finding in his theorem by saying that given the relation among the input variables and the output variable is limited to $n$ dimensions, that is to say the dimension of the attractor is given as $n$, this relation must be embedded in $\mathbb{R}^{2n+1}$. Whether $2n+1$ is the minimal number remains an open question.

4.2.2. The External Functions, $G_q$'s

In Kolmogorov's theorem (4.1.1), the $G_q$'s are specified as "properly chosen continuous functions of one variable". It is known (Lorentz, 1962) that the $\Phi_{pq}$'s are independent of the actual function $f$, upon which the $G_q$'s depend. This means that there exists a set of universal change of variables $\Phi_{pq}$'s, which construct an one-to-one and continuous mapping from $n$-dimensional space into $(2n+1)$-dimensional space. And, there exists a set of external functions, $G_q$'s, which can be used to construct the representation of the $f$ function in $(2n+1)$-dimensional space with addition. We will postpone the discussion of the $\Phi_{pq}$ functions to a later part of this section. Here, we assume that we have the $\Phi_{pq}$ functions already. Then, we can construct a mapping as in (4.1.2). Therefore, the task remaining is to construct a mapping from $\mathbb{R}^{2n+1}$ to $\mathbb{R}^1$ with addition. That is,
From (4.2.1) we can see that this problem is to find a linear superposition of single variable functions.

![Diagram](image)

**Fig. 4.4. External Function.**

It is important to notice here that, in this section, we constrain the approximated function in the class of functions which can be written as the summation of one variable functions. For example, 

\[ z = 0.2e^x + \sin(y^2) \]

and 

\[ z = 0.2x + 1.2\sin(y^2) + 0.3e^{0.7x+1.2}\text{Tanh}(5y) \]

are members of this class. However, 

\[ z = 0.2xy + 1.2x \sin(y^2) \]

is not in this class. The reason for this constraint is that we only investigate the ability of the developed method in approximating the external functions, the \( G_{pq} \)'s. It is important to select the task which can highlight this ability. In the superposition
function case, the internal functions are trivial, i.e. identity function.

Under this consideration, (4.2.1) can be rewritten as

\[ f(x) = \sum_{i=1}^{n} G_i(x_i) \quad x = (x_1, x_2, \ldots, x_n) \quad (4.2.2) \]

A more general case, however, will be investigated within the discussion of constructing internal functions, the \( \Phi_{\text{pg}} \)'s.

The Construction of External Functions

One of the alternative ways to find the \( G_i \)'s is to generalize the method that we have developed in the previous chapter. We have shown the ability of the proposed neural network structure to approximate a univariate function in the previous chapter. The structure in Fig. 4.4 can be considered as a large network which has \((2n+1)\) univariate function approximators in parallel. A simple model of this concept is shown in Fig. 4.5.
Fig. 4.5. A Multivariate Model.

Since the $\Sigma$ nodes are linear nodes, Fig. 4.5 can be modified as Fig. 4.6.
Fig. 4.6. A Modified Multivariate Model.

(Please refer to Chapter III for the detailed description of the network.)
The computational procedure of the multivariate model is very similar to the univariate model since the multivariate model is a straightforward extension of the univariate one. We display the weight updating formula in the following. (Refer to Chapter III for the detailed derivation of these formulae). A sample portion of the multivariate model is shown in Fig. 4.7.

\[ w(h)_{q,i,j} = \text{the weight of the link associated with the } g_1 \text{ node in the } j^{\text{th}} \text{ hidden unit of the } i^{\text{th}} \text{ input}, \]

\[ w(h)_{q,i,j+1} = \text{the weight of the link associated with the } g_2 \text{ node in the } j^{\text{th}} \text{ hidden unit of the } i^{\text{th}} \text{ input}, \]

\[ \theta_{i,j,1} = \text{the bias of the } g_1 \text{ node in the } j^{\text{th}} \text{ hidden unit of the } i^{\text{th}} \text{ input}, \]

Fig. 4.7. Weight Definitions
\[ \theta_{i,j,2} = \text{the bias of the g}_2 \text{ node in the j}^{\text{th}} \text{ hidden unit of the i}^{\text{th}} \text{ input}, \]

\[ w(o)_{i,j} = \text{the weight of the link associated with j}^{\text{th}} \text{ hidden unit of the i}^{\text{th}} \text{ input to the output node}, \]

\[ O(h)_{i,j} = \text{the output of the j}^{\text{th}} \text{ hidden unit of the i}^{\text{th}} \text{ input}, \]

\[ D = \text{the desired output associated with the given input}, \]

\[ O_o = \text{the actual output of the network associated with the given input}, \]

\[ \eta(h)_{i,j,1} = \text{the learning rate associated with the g}_1 \text{ node in the j}^{\text{th}} \text{ hidden unit of the i}^{\text{th}} \text{ input}, \]

\[ \eta(h)_{i,j,2} = \text{the learning rate associated with the g}_2 \text{ node in the j}^{\text{th}} \text{ hidden unit of the i}^{\text{th}} \text{ input}, \]

\[ \eta(o) = \text{the learning rate associated with the output node}. \]

The following are the weight updating formulae:

\[ \Delta w(o)_{i,j} = \eta(o) \delta O(h)_{i,j} \]  \hspace{1cm} (4.2.3)

where \( \delta = D - O \), and

\[ \eta(o) = \frac{1}{\left[ \sum_i \sum_j (O^2(h)_{i,j}) \right]} \]  \hspace{1cm} (4.2.4)

\[ \Delta w(h)_{i,j,1} = \eta(h)_{i,j,1} \delta(h)_{i,j,1} (Z_i - \theta_{i,j,1}) \]  \hspace{1cm} (4.2.5)

\[ \Delta \theta_{i,j,1} = -\eta(h)_{i,j,1} \delta(h)_{i,j,1} w(h)_{i,j,1} \]  \hspace{1cm} (4.2.6)

where \( \delta(h)_{i,j,1} = \delta w(o)_{i,j} O(h)_{i,j} (1 - O(h)_{i,j}) \)

\[ \eta(h)_{i,j,1} = 0.5 / [w(h)^2_{i,j,1} + (Z_i - \theta_{i,j,1})^2]^2 \]  \hspace{1cm} (4.2.7)

\[ \eta(h)_{i,j,1} = 0.5 / [w(h)^2_{i,j,1} + (Z_i - \theta_{i,j,1})^2] \]  \hspace{1cm} (4.2.8)

\[ \Delta w(h)_{i,j,2} = \eta(h)_{i,j,2} \delta(h)_{i,j,2} (Z_i - \theta_{i,j,2}) \]  \hspace{1cm} (4.2.9)
\[ \Delta \theta_{i,j,2} = -\eta(h)_{i,j,2} \delta(h)_{i,j,2} w(h)_{i,j,2} \]  
\[ \text{where} \quad \delta(h)_{i,j,2} = -\delta w(o)_{i,j} O(h)_{i,j} (1 - O(h)_{i,j}) \]  
\[ \eta(h)_{i,j,2} = 0.5 / [w(h)^2_{i,j,2} + (Z_i - \theta_{i,j,2})^2] \]

**Experimental Results**

Since the multivariate model can be considered as parallel connection of univariate models, the initialization procedure is very similar to the one which has been described in the univariate experiments. However, the biases of each unit have to be determined based upon the corresponding input range.

We still apply the index of accuracy as the performance measure in this part of experiments. The index of accuracy is, as defined in Chapter III, the root mean square error divided by the standard deviation of the training output data.

**Convergence Test**

In this part of the experiments, we construct a multivariate model with 150 units associated with each input. The experiments were designed in the same way as for the univariate case described in Chapter III.

**Case 1: Quadratic Function on Rectangle Domain**

The bivariate normal density function is
In this experiment, we take a special case, say $x_1$ and $x_2$ are independent, $\mu_1 = \mu_2 = 0$ and $\sigma_1 = \sigma_2 = 1$. Then, the above equations can be rewritten as

$$y = \frac{1}{2\pi \sigma_1 \sigma_2 \sqrt{1 - \rho^2}} \exp\left\{ -\frac{1}{2(1 - \rho^2)} \left[ \left( \frac{x_1 - \mu_1}{\sigma_1} \right)^2 \right. \right.$$

$$\left. - 2\rho \left( \frac{x_1 - \mu_1}{\sigma_1} \right) \left( \frac{x_2 - \mu_2}{\sigma_2} \right) + \left( \frac{x_2 - \mu_2}{\sigma_2} \right)^2 \right]\right\}$$

It is equivalent to the following form

$$\ln(y) = \ln(1) - \ln(2\pi) - \frac{x_1^2 + x_2^2}{2}$$

If we define $\ln(y) = y'$, then we can write down

$$y' = \ln(1) - \ln(2\pi) - \frac{x_1^2 + x_2^2}{2}$$

This is a form of (4.2.1). In this case, we take $x_1$ and $x_2$ as inputs, where $x_1, x_2 \in [-2, 2]$, and $y'$ as the output. For the clarity of presentation, we multiply the function by 20. Thus, the actual function that the multivariate model learns is

$$y' = \ln(20) - \ln(2\pi) - \frac{x_1^2 + x_2^2}{2}$$

The plots of accuracy index versus number of trials and RMSE versus number of trials are shown in Fig. 4.8. The behavior of the multivariate model at different trials are shown in Fig. 4.9. The index of accuracy reaches 1.41% after 100 trials.

Some other types of functions have also been used to show the convergence behavior and capability of our network to capture the $G_q$'s. The functions tested and their results are listed as follows.
Case 2. $z = 0.2 e^x + \sin(y^2)$ (Fig. 4.10. - Fig. 4.11.)

Case 3. $z = 0.2x + 1.2 \sin(y^2) + 0.3 e^{0.7x} + 1.2 \tanh(5y)$ (Fig. 4.12. - Fig. 4.13.)

Case 4. $z = \sin(x) + 1.2 \cos(y^2)$ (Fig. 4.14. - Fig. 4.15.)

Case 5: $z = 0.8x - 0.4y$ (Fig. 4.16. - Fig. 4.17.)

The index of accuracy and RMSE after 100 trials are listed in table 4.6.

Table 4.6. The Index of Accuracy and RMSE After 100 Trials.

<table>
<thead>
<tr>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
<th>Case 4</th>
<th>Case 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index of Accuracy</td>
<td>1.41%</td>
<td>0.86%</td>
<td>2.85%</td>
<td>0.97%</td>
</tr>
<tr>
<td>RMSE</td>
<td>1.31%</td>
<td>1.04%</td>
<td>4.34%</td>
<td>1.00%</td>
</tr>
</tbody>
</table>

According to the results of these experiments, the ability of our network in approximating a multivariate function of the form (4.2.1) is very good. Again, according to Kolmogorov's theorem, there exists a set of "universal change of variables" $\Phi_{pq}$'s. That means, $G_q$'s (not $\Phi_{pq}$'s) are the functions which are related to the approximated function. The results of these experiments suggest that it may be possible to construct a multivariate network which can approximate all kinds of real continuous functions given that we can find $\Phi_{pq}$'s. Therefore, the effort involved in searching for $\Phi_{pq}$'s become very valuable.
Fig. 4.8. Quadratic on Rectangle, Index of Accuracy vs. Trials.

Table 4.1. The Index of Accuracy and RMSE of Case 1.

<table>
<thead>
<tr>
<th># of trial</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>.0906</td>
<td>.0847</td>
</tr>
<tr>
<td>10</td>
<td>.0624</td>
<td>.0583</td>
</tr>
<tr>
<td>25</td>
<td>.0321</td>
<td>.0300</td>
</tr>
<tr>
<td>50</td>
<td>.0202</td>
<td>.0189</td>
</tr>
<tr>
<td>75</td>
<td>.0160</td>
<td>.0150</td>
</tr>
<tr>
<td>100</td>
<td>.0141</td>
<td>.0132</td>
</tr>
</tbody>
</table>
Fig. 4.9. Quadratic on Rectangle.
(a) Actual Function,  (b) After 5 Trials.
Fig. 4.9. (Continued).
(c) After 50 Trials, (d) After 100 Trials.
Fig. 4.10. $z=0.2(e^x)+\sin(y^2)$, Index of Accuracy vs. Trials.

Table 4.2. The Index of Accuracy and RMSE of Case 2.

<table>
<thead>
<tr>
<th># of trial</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>.0574</td>
<td>.0696</td>
</tr>
<tr>
<td>10</td>
<td>.0431</td>
<td>.0523</td>
</tr>
<tr>
<td>25</td>
<td>.0219</td>
<td>.0265</td>
</tr>
<tr>
<td>50</td>
<td>.0112</td>
<td>.0136</td>
</tr>
<tr>
<td>75</td>
<td>.0093</td>
<td>.0112</td>
</tr>
<tr>
<td>100</td>
<td>.0086</td>
<td>.0104</td>
</tr>
</tbody>
</table>
Fig. 4.11. $z=0.2(e^{x})+\sin(y^{2})$.
(a) Actual Function,  (b) After 5 Trials.
Fig. 4.11. (Continued).
(c) After 50 Trials, (d) After 100 Trials.
Fig. 4.12. $z=0.2x + 1.2\sin(y^2) + 0.3e^{0.7x} + 1.2\tanh(5y)$, Index of Accuracy vs. Trials.

Table 4.3 The Index of Accuracy and RMSE of Case 3.

150 units

<table>
<thead>
<tr>
<th># of trial</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>.1130</td>
<td>.1719</td>
</tr>
<tr>
<td>10</td>
<td>.0831</td>
<td>.1265</td>
</tr>
<tr>
<td>25</td>
<td>.0552</td>
<td>.0840</td>
</tr>
<tr>
<td>50</td>
<td>.0393</td>
<td>.0598</td>
</tr>
<tr>
<td>75</td>
<td>.0323</td>
<td>.0492</td>
</tr>
<tr>
<td>100</td>
<td>.0285</td>
<td>.0434</td>
</tr>
</tbody>
</table>
Fig. 4.13. $z = 0.2x + 1.2\sin(y^2) + 0.3e^{0.7x} + 1.2\tanh(5y)$.
(a) Actual Function, (b) After 5 Trials.
Fig. 4.13. (Continued).
(c) After 50 Trials, (d) After 100 Trials.
Fig. 4.14. $z = \sin(x) + 1.2\cos(y^2)$, Index of Accuracy vs. Trials.

Table 4.4. The Index of Accuracy and RMSE of Case 4.

<table>
<thead>
<tr>
<th># of trial</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.0503</td>
<td>0.0516</td>
</tr>
<tr>
<td>10</td>
<td>0.0341</td>
<td>0.0350</td>
</tr>
<tr>
<td>25</td>
<td>0.0249</td>
<td>0.0255</td>
</tr>
<tr>
<td>50</td>
<td>0.0169</td>
<td>0.0174</td>
</tr>
<tr>
<td>75</td>
<td>0.0125</td>
<td>0.0128</td>
</tr>
<tr>
<td>100</td>
<td>0.0097</td>
<td>0.0100</td>
</tr>
</tbody>
</table>
Fig. 4.15. $z = \sin(x) + 1.2\cos(x^2)$.
(a) Actual Function,  (b) After 5 Trials.
Fig. 4.15. (Continued).
(c) After 50 Trials, (d) After 100 Trials.
Fig. 4.16. $z=0.8x-0.4y$, Index of Accuracy vs. Trials.

Table 4.5. The Index of Accuracy and RMSE of Case 5.

<table>
<thead>
<tr>
<th># of trial</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>.0662</td>
<td>.0720</td>
</tr>
<tr>
<td>10</td>
<td>.0391</td>
<td>.0425</td>
</tr>
<tr>
<td>25</td>
<td>.0155</td>
<td>.0169</td>
</tr>
<tr>
<td>50</td>
<td>.0084</td>
<td>.0091</td>
</tr>
<tr>
<td>75</td>
<td>.0069</td>
<td>.0075</td>
</tr>
<tr>
<td>100</td>
<td>.0062</td>
<td>.0068</td>
</tr>
</tbody>
</table>
Fig. 4.17. $z = 0.8x - 0.4y$.
(a) Actual Function, (b) After 5 Trials.
Fig. 4.17. (Continued).
(c) After 50 Trials, (d) After 100 Trials.
4.2.3. The Internal Functions, $\Phi_{pq}$

As we have shown in Fig. 4.4., the multivariate network with the external functions $G_q$ is designed to approximate any real continuous function which can be represented as

$$f(Z) = \sum_{q=1}^{2n+1} G_q(Z_q), \quad Z = (Z_1, Z_2, \ldots, Z_{2n+1}) \quad (4.2.1)$$

The functions of the form (4.2.1) are a subset of the space of all continuous functions. But, theoretically, this does not limit the applicability of this multivariate model. One of the most important contributions that Kolmogorov made with his representation theorem is that there exists a set of "universal change of variables" which can map any n-dimensional continuous multivariate function into the (2n+1)-dimensional space, and the image function is of form $f(Z) = \sum_{q=1}^{2n+1} G_q(Z_q)$. That means, any continuous function can be written in the form of (4.2.1). Therefore, the multivariate model can be used to approximate all continuous functions as long as we select the correct "universal change of variables". In Kolmogorov's theorem, he even pointed out that the "universal change of variables" or the internal functions, the $\Phi_{pq}$'s, are independent of the approximated function. On the other hand, we can preselect the internal functions before we begin the approximating process. Although constructing $\Phi_{pq}$'s is not a part of the learning procedure of the multivariate model, it is very important to extend the applicability of the proposed multivariate model.
What are the precise forms of the internal functions, $\Phi_{pq}$'s? This is still an unsolved question. However, there are some necessary conditions that the internal functions should fulfill. These conditions are:

1. $\Phi_{pq}$'s are continuous monotone functions,
2. $\Phi_{pq}$'s are nonsmooth functions.

The descriptions of these conditions are as follows:

1. $\Phi_{pq}$'s are continuous monotone functions

   As we have stated in the early part of this chapter, the mapping which is constructed by the internal functions has to be continuous and one-to-one. The mapping relation in Kolmogorov's theorem is as follows:

   ![Diagram of mapping relationship among $f$, $G_q$'s, and $\Phi_{pq}$'s.]

   Since $f$ and the $G_q$'s are continuous functions, the $\Phi_{pq}$'s have to be continuous also. And, the $\Phi_{pq}$'s are one-to-one mappings, otherwise they will have two points in n-dimensional space which cannot be distinguished by the family of functions

   $$\sum_{p=1}^{n} \Phi_{pq}(x_p) = q = 1, 2, \ldots, 2n + 1$$
(2) \( \Phi_{pq} \)'s are nonsmooth functions

Let us return to Hilbert's conjecture. Despite its disproof, it originated from a sound idea: that "bad" functions in general cannot be represented by superpositions of "good" functions. The goodness of a function is defined by a characteristic \( \mathcal{X} = n/(p+\alpha) \) (Kolmogorov, 1959). That means a function of \( n \) variables, \( f(x_1, x_2, \ldots, x_n) \), defined on an \( n \)-dimensional unit cube have all partial derivatives of orders not exceeding \( p \), with the derivatives of order \( i \) satisfying a Lipschitz\(^1\) condition with exponent \( \alpha \) (\( i = 0,1, \ldots, 0 < \alpha \leq 1 \)). Lorentz (Lorentz, 1962) proved the following theorem to state the relation between this characteristic and the representability.

**Theorem.** (Lorentz) Not all continuous functions with a given characteristic \( \mathcal{X}_0 \) can be represented by superpositions of functions of characteristics \( \mathcal{X} = n/(p+\alpha) < \mathcal{X}_0 \) and with \( p+\alpha \geq 1 \).

It is straightforward to see why the \( \Phi_{pq} \)'s functions should be nonsmooth functions in order to construct the exact representation (not an approximation). In this research, we use a particular form to represent the external functions of the Kolmogorov form:

---

\(^1\) \( f(x) \) belongs to class \( \text{Lip}[\alpha] \) if there are constants \( c \) and \( \alpha \), \( 0 < \alpha \leq 1 \), for which 
\[ |f(x) - f(y)| \leq c|x - y|^{\alpha} \] for all points, \( x \) and \( y \), in the domain of \( f \).
In order to have the equality in Kolmogorov's representation, 
\[ \sum \Phi_{pq}(x_p) \] has to be a nonsmooth function, as do the \( \Phi_{pq} \) functions.

**Construction of the Internal Functions, \( \Phi_{pq} \)'s**

There are two different kinds of constructions of the \( \Phi_{pq} \)'s. One is due to Kolmogorov's work (Kolmogorov, 1957). Another is based on Sprencher's modification (Sprencher, 1966).

**Kolmogorov's Version of \( \Phi_{pq} \)'s**

In the whole construction of Kolmogorov's \( \Phi_{pq} \)'s the indices \( p, q, r, \) and \( i \) will always take on integer values where \( 1 \leq p \leq n \), \( 1 \leq q \leq 2n+1 \), \( r = 1, 2, \ldots \), and \( 1 \leq i \leq m_r = (9n)^r+1 \). In the following context, we refer to \( r \) as the rank of each interval.

Kolmogorov defined the segment \( A^q_{r, i} \) as

\[
A_{r, i}^q = \left[ \frac{1}{(9n)^r(i-1 - \frac{q}{3n})}, \frac{1}{(9n)^r(i - \frac{1}{3n} - \frac{q}{3n})} \right]
\]

For fixed \( r \) and \( q \), one of them is obtained from the other through the replacement of \( i \) by \( (i+1) \). These segments are not only nonoverlapping
but are separated by distance $1/(3n(9n)^r)$. Fig. 4.19 illustrates the definitions of $A_{r,i}^q$ and its neighbors.

Fig. 4.19. $A_{r,i}^q$ Segments.

Except for these gaps of $1/(3n(9n)^r)$, these segments cover the unit segment $E = [0,1]$. Similarly, the cubes

$$S_{r,i_1,i_2,...,i_n}^q = \prod_{p=1}^{n} A_{r,i_p}^q$$

will cover the cube $E^n$ except for the separating bands of width $1/(3n(9n)^r)$. Therefore, Kolmogorov established the following lemmas, which were stated without proof in his 1957 paper, for the purpose of constructing the $\Phi_{pq}$'s.

**Lemma 1.** (Kolmogorov) The system of all cubes $S_{r,i_1,i_2,...,i_n}^q$ with a constant $r$, variables $q$, and $i_1, i_2, \ldots, i_n$, cover the unit cube $E^n$ so that every point of $E^n$ is covered at least $n+1$ times.

**Lemma 2.** (Kolmogorov) One can select constants $\lambda_{r,i}^{p,q}$ and $\epsilon_r$ so that the following conditions will be satisfied

(1) $\lambda_{r,i}^{p,q} < \lambda_{r,i}^{p,q} + 1 \leq \lambda_{r,i}^{p,q} + \frac{1}{2^{r}}$
(2) \( \lambda_{r,1}^{p,q} < \lambda_{r+1,1}^{p,q} \leq \lambda_{r,1}^{p,q} + \varepsilon_r - \varepsilon_{r+1} \), if the \( A_{r,i}^q \) and \( A_{r+1,i}^q \) intersect,

(3) The segments \( \Delta_{r,i_1,\ldots,i_n}^q = \left[ \sum_{p} \lambda_{r,i_p}^{p,q} \right] - \varepsilon r \), for fixed \( r \) and \( q \), do not intersect.

According to (1) and (3), it can be derived that

(4) \( \varepsilon_r \leq 1/2^r \)

On the basis of the above mentioned properties of the \( A_{r,i}^q \) and properties (1), (2), and (4) of the constants \( \lambda_{r,i}^{p,q} \) and \( \varepsilon_r \), Kolmogorov established the following lemma.

**Lemma 3.** (Kolmogorov) If \( p \) and \( q \) are kept fixed then the requirement

(5) \( \lambda_{r,i}^{p,q} < \Phi_{pq}(x) \leq \lambda_{r,i}^{p,q} + \varepsilon_r \), where \( x \in A_{r,i}^q \), determines uniquely the continuous function \( \Phi_{pq} \) on \( \mathbb{E}^1 \), and,

(6) \( \sum_{p} \Phi_{pq}(x_p) \in \Delta_{r,i_1,\ldots,i_n}^q \), where \( (x_1,\ldots,x_n) \in S_{r,i_1,i_2,\ldots,i_n}^q \).

**Sprenger's Version of \( \Phi_{pq} \)**

Sprenger observed that the Kolmogorov's representation could be simplified as in Fig. 4.3. For the constants, \( \lambda_p \)'s, of this representation, one can take arbitrary rationally independent numbers, i.e. irrational numbers. This is because any number \( \lambda_p \) for which a relation \( r_1\lambda_1 + r_2\lambda_2 + \ldots + r_p\lambda_p = 0 \) with rational \( r_p \)'s is possible is when \( r_1 = r_2 = \ldots = r_p = 0 \) only. Although this is theoretically valid, it cannot be implemented on computers. This is due to the fact that computers cannot represent
irrationals. For this reason, we will not look into Sprecher's version of \( \Phi_{pq} \)'s in this research.

In the rest of the section, we describe the ideas and the experiments that we have for constructing \( \Phi_{pq} \)'s. Although Lorentz and Sprecher have done some modifications on Kolmogorov's theorem, we still apply the original Kolmogorov's formula in this research. The reasons can be summarized as follows. In Lorentz's modification, \( G_q \)'s can be substituted by a single \( G \). This is a tremendous achievement analytically, however, it makes parallel processing become very difficult. As described in the previous section, \( G_q \)'s were constructed via neural networks. There was an individual network for each \( G_q \). This construction makes the task of learning \( G_q \) easy since every network is independent of the others. This is one of the reasons for the success of that particular task. Having only one network construct the relationship between each input and output becomes very difficult. And, the associated weight updating procedure will be much more complicated than what we have now. Although the concept of having a single \( G \) may work, the author cannot forsee any advantage that we might gain by using the neural network structure. As mentioned before (see Fig. 4.3.), Sprecher modified Kolmogorov's theorem by substituting \( (2n+1) \Phi_q \)'s for \( n(2n+1) \Phi_{pq} \)'s. Again, this is a very important achievement theoretically. However, Sprecher's modification is valid under the assumption that the \( \lambda_i \) are rationally independent. But the fact that computers cannot process irrationals rules out the possibility of implementing Sprecher's modification.
The author has tried several approaches of constructing $\Phi_{pq}$'s. The first approach is based on Kolmogorov's construction. The second approach is to design a neural network to construct a continuous, monotone increasing and nonsmooth function. The third approach is to design a neural network to construct a continuous, monotone increasing and smooth function. The approaches can be described in detail as follows.

**Approach 1**

We can construct $\Phi_{pq}$'s based on Kolmogorov's suggestion. It is worthwhile to notice that $\Phi_{pq}$'s are independent of the approximated function $f(x)$, i.e. $\Phi_{pq}$'s will be fixed functions in the learning process.

Since, in the learning process, the networks are dedicated to constructing the $G_q$ functions, the weight updating procedure remains the same (from (4.2.3) to (4.2.12)). The most crucial part here is to explicitly construct $\Phi_{pq}$'s which have the continuous, monotone increasing and nonsmooth properties that we have described before.

According to Kolmogorov's construction, rank $r$ can be any positive number. In order to implement his construction on a computer, we have to determine an upper bound for $r$, say $r$. From Fig. 4.19, we can see that for fixed values of $n$, $q$, and $r$, the gap between the $i^{th}$ and the $(i+1)^{th}$ interval is $1/(3n(9n)^r)$. Given $h$ is the precision value that one desires, $r$ can be determined as follows.
\[ h \geq 1/(3n(9n)r), \text{ and } n \geq 2. \]

\[ \Rightarrow r \geq -\ln(3nh)/\ln(9n). \]

Since \( r \) is an integer, we thus take \( r \) as the smallest integer which is greater than or equal to \(-\ln(3nh)/\ln(9n)\). Since

\[ \varepsilon_r \leq 1/2^r, \]

\[ \lambda^{p,q}_{r,r} < \Phi_{pq}(x) \leq \lambda^{p,q}_{r,i_p} + \varepsilon_r, \text{ where } x \in A^{q}_{r,i_p} \]

and the length of each interval at rank \( r \) is \( \frac{3n-1}{3n(9n)^r} \), we approximate \( \Phi_{pq}(x), x \in A^{q}_{r,i_p} \) to be of the following form.

\[
\Phi_{pq,i_p,r}(x) = \begin{cases} 
0 & \text{if } x < \frac{1}{(9n)^r}(i_p - 1 - \frac{q}{3n}), \\
\frac{3n(9n)^r}{2^r(3n-1)}(x - \frac{1}{(9n)^r}(i_p - 1 - \frac{q}{3n})) & \text{if } x \in A^{q}_{r,i_p'}, \\
\frac{1}{2^r} & \text{if } x > \frac{1}{(9n)^r}(i_p - 1 - \frac{1}{3n} - \frac{q}{3n}).
\end{cases}
\]

\[ \Phi_{pq}(x) = \sum_{i_p=1}^{r} \sum_{r=1}^{r} \Phi_{pq,i_p,r}(x) \]

In this experiment, we focus in the class of functions which have two variables only, i.e. \( n = 2 \) (bivariate functions). We let \( h = 0.001 \).

Therefore, the parameters of the experiment are determined as follows.
We have tried to apply this model with the external network, $G_q$'s, which was described in the last section, to approximate two different functions. They are (1) $z=0.2 \ xy + 1.2 \ \sin(x+y)$ and (2) $z=\sin(x)+1.2 \ \cos(y^2)$. In case (1), it is impossible to approximate this functions simply by $G_q$'s since its variables are nonseparable. The result of the convergence test are shown in Fig. 4.20 and Table 4.7. The results are not as good as expected.

However, we were quite successful in approximating function (2) simply by $G_q$'s. Please refer to the experimental results of the last section. It is surprising that the experimental results of applying the combination of $\Phi_{pq}$'s and $G_q$'s were quite poor. The results are shown in Fig. 4.21 and Table 4.8. Some possible reasons for these unsatisfactory results are the following.

1. Kolmogorov's result applies to infinite $r$. This will guarantee the one-to-one and continuous mapping. However, in computer application, it is impossible to allow $r$ to be infinite. That means giving any upper bound for $r$ will not guarantee the mapping as is one-to-one.
2. By looking into the mapping between n-dimensional space and 
(2n+1)-dimensional space, we found out that most of the n-
dimensional points have been mapped into a small range, i.e., a 
diagonal tube, in (2n+1)-dimensional space. That is, given (x₁,x₂),
the resulting point (y₁,y₂,...,y₅) obtained by Φpq transformation has
the property that the yᵢ's are very close to each other. This 
phenomenon limited the performance of the Gq's approximation. It
is known that Gq's performance will decay in an environment 
characterized by sharp or discontinuous change as was seen in the 
approximation of the step function by Gq's.

There are some possible approaches to resolve this problem. One is
to study the behavior of Φpq mappings in a more detailed fashion.
Kolmogorov's construction did not explicitly describe the relations 
among the Φpq's. Theoretically, it can be proved that the one-to-one 
mapping holds under Kolmogorov's construction. However, it is not
good enough for computer implementation. One of the explanations is
that x and y are theoretically counted as different if x-y=10⁻⁸. However, in 
computer implementation, x and y are the same in most cases. We need
to understand the behavior of Φpq in more detail. This approach is
related to traditional functional analysis.
Fig. 4.20. $z=0.2xy+1.2\sin(x+y)$, Index of Accuracy vs. Trials of Approach 1.

Table 4.7. The Index of Accuracy and RMSE of Approach 1.

<table>
<thead>
<tr>
<th># of trial</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>.4783</td>
<td>.3879</td>
</tr>
<tr>
<td>10</td>
<td>.3498</td>
<td>.2837</td>
</tr>
<tr>
<td>25</td>
<td>.2614</td>
<td>.2120</td>
</tr>
<tr>
<td>50</td>
<td>.2288</td>
<td>.1855</td>
</tr>
<tr>
<td>75</td>
<td>.2054</td>
<td>.1666</td>
</tr>
<tr>
<td>100</td>
<td>.1889</td>
<td>.1532</td>
</tr>
</tbody>
</table>
Fig. 4.21. \( z = \sin(x) + 1.2 \cos(y^2) \), Index of Accuracy vs. Trials of Approach 1.

Table 4.8. The Index of Accuracy and RMSE of Approach 1.

<table>
<thead>
<tr>
<th># of trial</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.5138</td>
<td>0.6844</td>
</tr>
<tr>
<td>10</td>
<td>0.4394</td>
<td>0.5853</td>
</tr>
<tr>
<td>25</td>
<td>0.3617</td>
<td>0.4818</td>
</tr>
<tr>
<td>50</td>
<td>0.3460</td>
<td>0.4609</td>
</tr>
<tr>
<td>75</td>
<td>0.3475</td>
<td>0.4626</td>
</tr>
<tr>
<td>100</td>
<td>0.3463</td>
<td>0.4613</td>
</tr>
</tbody>
</table>
Approach 2

It is very difficult to construct numerically the set of $\Phi_{pq}$'s that are ideal in Kolmogorov's sense. However, is it possible to include $\Phi_{pq}$ in the learning process? In other words, can the network learn the $\Phi_{pq}$'s as well? In this subsection, we consider this approach.

As discussed before, the $\Phi_{pq}$'s have to be continuous, monotone increasing and nonsmooth. It is important that we need to design a network which can "learn" the representation of the observed system and fulfill those requirements as well. One possible construction for $\Phi_{pq}$'s is shown in Fig. 4.22.
Fig. 4.22. $\Phi_{pq}$'s Networks via Approach 2.
Every hidden unit in $\Phi_{pq}$ nets is represented by indices $p$, $q$, $m$ ($1 \leq p \leq n$, $1 \leq q \leq 2n+1$, $1 \leq m \leq M$, where $n$ is the number of inputs, $M$ is the number of units associated with each $\Phi_{pq}$ net). The output function of each unit is defined as follows.

$$O_{pqm} = \max \{ \min \left[ 1, w_{pqm} (x_p - \sum_{u=1}^{m-1} \frac{1}{w_{pqu}} ) \right], 0 \}$$

where $w_{pqm} > 0$, and the output of $\Phi_{pq}$ net is defined as the following.

$$O_{pq} = \sum_{m=1}^{M} O_{pqm} = \sum_{m=1}^{M} \max \{ \min \left[ 1, w_{pqm} (x_p - \sum_{u=1}^{m-1} \frac{1}{w_{pqu}} ) \right], 0 \}$$

It is conceivable that $\Phi_{pq}$'s nets perform as a set of continuous, monotone increasing and nonsmooth functions.

Another important issue is to develop a weight updating procedure. The $G_q$'s nets are considered to be the same as that have been described previously. Although these $\Phi_{pq}$'s nets fulfill the desired requirements, the weight updating procedure is very difficult to develop. This is due to the Max and Min functions which are included in the computational representation of the $\Phi_{pq}$'s nets. The author has developed a heuristic weight updating procedure which is described in the following.

Given the error feedback from the $G_q$ net is $\delta_q$, which can be calculated via the backward propagation method, we assume

$$\Delta w_{pqm} = \eta \cdot w_{pqm} \text{ and}$$
We have applied this model with the external networks, $G_q's$, to approximate two different functions. They are (1) $z=0.2xy+1.2\sin(x+y)$, and (2) $z=\sin(x)+1.2\cos(y^2)$. The results of the convergence test are shown in Fig. 4.23., Table 4.9, Fig. 4.24., and Table 4.10.

This approach has its own inherent problem. The $G_p's$ networks construct the representation between $(2n+1)$-dimensional space and $\mathcal{F}$. But, if we allow $\Phi_{pq}'s$ to construct the representation from $n$-dimensional space to $(2n+1)$-dimensional space along with the $G_q's$, it will cause a lot of "confusion" for the $G_q's$. The nonmonotonic decreasing behavior in
Fig. 4.23 and Fig. 24 are expected in this approach. However, whether the \( \Phi_{pq} \)'s and \( G_q \)'s will settle down and construct a reasonable representation is still an open question. In these experiments, the performance of this approach is an improvement compared to the previous approach. However, the performance is still poor when compared to the experiments with the \( G_q \)'s functions only. One of the possible reasons for the poor performance is the weight updating procedure. The weight updating procedure which is used in this approach simplified the nonlinearity of the constructions. No \( \text{Max}(\cdot, \cdot) \) and \( \text{Min}(\cdot, \cdot) \) functions are considered in the weight updating procedure. This may decrease the efficiency of the networks to approximating the observed function. However, this approach shows an alternative method to implementing the neural networks as a multivariate function approximator.
Fig. 4.23. $z=0.2xy+1.2\sin(x+y)$, Index of Accuracy vs. Trials of Approach 2.

Table 4.9. The Index of Accuracy and RMSE of Approach 2.

<table>
<thead>
<tr>
<th># of trial</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>.7939</td>
<td>.6438</td>
</tr>
<tr>
<td>10</td>
<td>.3301</td>
<td>.2677</td>
</tr>
<tr>
<td>25</td>
<td>.2498</td>
<td>.2026</td>
</tr>
<tr>
<td>50</td>
<td>.1840</td>
<td>.1493</td>
</tr>
<tr>
<td>75</td>
<td>.1435</td>
<td>.1164</td>
</tr>
<tr>
<td>100</td>
<td>.1136</td>
<td>.0921</td>
</tr>
</tbody>
</table>
Fig. 4.24. $z = \sin(x) + 1.2\cos(y^2)$, Index of Accuracy vs. Trials of Approach 2.

Table 4.10. The Index of Accuracy and RMSE of Approach 2.

<table>
<thead>
<tr>
<th># of trial</th>
<th>Index of Accuracy</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>.6181</td>
<td>.8234</td>
</tr>
<tr>
<td>10</td>
<td>.2874</td>
<td>.3828</td>
</tr>
<tr>
<td>25</td>
<td>.2781</td>
<td>.3704</td>
</tr>
<tr>
<td>50</td>
<td>.1499</td>
<td>.1997</td>
</tr>
<tr>
<td>75</td>
<td>.1749</td>
<td>.2330</td>
</tr>
<tr>
<td>100</td>
<td>.1452</td>
<td>.1934</td>
</tr>
</tbody>
</table>
Approach 3

As described in the last subsection, the poor performance of the networks, which is constructed via approach 2, is largely due to the weight updating procedure. One alternative solution is to construct the $\Phi_{pq}$'s nets via logistic nodes, i.e. each node performs as a logistic function. The unit in the $\Phi_{pq}$'s nets is identified by the indices $p, q, m$ ($1 \leq p \leq n, 1 \leq q \leq 2n+1, 1 \leq m \leq M$, where $n$ is the number of inputs, $M$ is the number of units associated with each $\Phi_{pq}$ net). The output function of each unit is defined as follows.

$$O_{pqm} = \frac{1}{1 + e^{-Net_{pqm}}}$$

And

$$Net_{pqm} = w_{pqm} \cdot (x_p - O_{pqm}), \text{ where } w_{pqm} > 0.$$

The output of the $\Phi_{pq}$ net is

$$O_{pq} = \sum_{v=1}^{M} P_{pqv}.$$

The input of the $G_q$ net is

$$O_q = \sum_{p=1}^{n} O_{pq}.$$

The schematic diagram of $\Phi_{pq}$'s nets is illustrated in Fig. 4.25.
Fig. 4.25. $\Phi_{pq}$'s Nets via Approach 3.
It is conceivable that the continuous and monotone increasing requirements of the internal functions, $\Phi_{pq}$'s can be fulfilled in this construction. However, the nonsmooth requirement no longer holds in this construction. This is because that the logistic function are $C^\infty$ functions. It is obvious that the sum of $C^\infty$ functions is still a $C^\infty$ function. However, the weight updating procedure is easier than the one in approach 2. The weight updating procedure can be described as follows.

Given the error feedback from the $G_q$ net is $\delta_q$ (this can be calculated via error backward propagation method),

$$
\frac{\partial E}{\partial w_{pqm}} = \frac{\partial E}{\partial O_q} \cdot \frac{\partial O_q}{\partial O_{pq}} \cdot \frac{\partial O_{pq}}{\partial O_{pqm}} \cdot \frac{\partial O_{pqm}}{\partial N_{pqm}} \cdot \frac{\partial N_{pqm}}{\partial w_{pqm}}
$$

$$=-\delta_q \cdot 1 \cdot 1 \cdot O_{pqm} (1 - O_{pqm}) \cdot (x_p - \theta_{pqm})$$

Therefore,

$$\Delta w_{pqm} = \eta_q \cdot \delta_q \cdot O_{pqm} (1 - O_{pqm}) \cdot (x_p - \theta_{pqm})$$

And,

$$
\frac{\partial E}{\partial \theta_{pqm}} = \frac{\partial E}{\partial O_q} \cdot \frac{\partial O_q}{\partial O_{pq}} \cdot \frac{\partial O_{pq}}{\partial O_{pqm}} \cdot \frac{\partial O_{pqm}}{\partial N_{pqm}} \cdot \frac{\partial N_{pqm}}{\partial \theta_{pqm}}
$$

$$=-\delta_q \cdot 1 \cdot 1 \cdot O_{pqm} (1 - O_{pqm}) \cdot -w_{pqm}$$

Thus,

$$\Delta \theta_{pqm} = -\eta_q \cdot \delta_q \cdot O_{pqm} (1 - O_{pqm}) \cdot w_{pqm}.$$
The $\eta_q$'s are determined via the discussion of the dynamic learning rate (section 3.3). We have applied this model with the external networks, $G_q$'s, to approximate two different functions. They are (1) $z=0.2xy+1.2\sin(x+y)$, and (2) $z=\sin(x)+1.2\cos(y^2)$. The results of the convergence test are shown in Fig. 4.26., Table 4.11, Fig. 4.27., and Table 4.12.

This approach has the same inherent problem as in the approach 2. It is because that two networks are concatenated together and updated at the same moment. This causes confusion to the $G_q$'s networks. The nonmonotonic behavior of accuracy index vs. trials is conceivable. However, the logistic approach (or approach 3) has another problem. The linear combination of logistic functions is theoretically monotonic increasing. However, in certain input range, the resulting function, $\Phi_{pq}$'s, can be very flat. This behavior of the $\Phi_{pq}$'s networks will increase the difficulty of the $G_q$'s networks to approximate the external functions. It is expected that some type of self-organization algorithm need to be applied to improve the performance of the networks by eliminating the flat range.
Fig. 4.26. $z=0.2xy+1.2\sin(x+y)$, Index of Accuracy vs. Trials of Approach 3.

Table 4.11. The Index of Accuracy and RMSE of Approach 3.

<table>
<thead>
<tr>
<th>$\Phi_{pq}$ 30 units</th>
<th>$G_q$ 120 units</th>
</tr>
</thead>
<tbody>
<tr>
<td># of trial</td>
<td>Index of Accuracy</td>
</tr>
<tr>
<td>5</td>
<td>.4948</td>
</tr>
<tr>
<td>10</td>
<td>.4543</td>
</tr>
<tr>
<td>25</td>
<td>.4016</td>
</tr>
<tr>
<td>50</td>
<td>.3651</td>
</tr>
<tr>
<td>75</td>
<td>.3468</td>
</tr>
<tr>
<td>100</td>
<td>.3355</td>
</tr>
</tbody>
</table>
Fig. 4.27. $z = \sin(x) + 1.2\cos(y^2)$, Index of Accuracy vs. Trials of Approach 3.

Table 4.12. The Index of Accuracy and RMSE of Approach 3.

<table>
<thead>
<tr>
<th>$\Phi_{pq}$ 30 units</th>
<th>$G_q$ 120 units</th>
</tr>
</thead>
<tbody>
<tr>
<td># of trial</td>
<td>Index of Accuracy</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
</tr>
<tr>
<td>5</td>
<td>.6172</td>
</tr>
<tr>
<td>10</td>
<td>.5830</td>
</tr>
<tr>
<td>25</td>
<td>.5367</td>
</tr>
<tr>
<td>50</td>
<td>.4957</td>
</tr>
<tr>
<td>75</td>
<td>.4696</td>
</tr>
<tr>
<td>100</td>
<td>.4521</td>
</tr>
</tbody>
</table>
The tasks which have been accomplished in this chapter can be summarized as follows.

(1) Constructing the external functions via neural networks.

In this task, we focused on constructing the external functions, $G_q$'s. In order to eliminate the effects of the internal functions, $\Phi_{pq}$'s, the experiments were conducted on a specific class of functions. These functions are represented as a summation of univariate functions. In this class of functions, the internal functions, $\Phi_{pq}$'s, are considered as identity functions. This is because that there is no need to "transform" the form of the approximated function into the Kolmogorov's form. Some experiments which have been conducted showed a promising ability of the constructed networks in terms of simulating the behavior of the observed system.

(2) Constructing the internal functions via Kolmogorov's construction or neural networks.

It is obvious that there are many functions that are not linear superposition of single variable functions. However, due to Kolmogorov's representation theorem, those functions can be represented in linear superposition form in a higher dimensional space by using a set of "universal change of variables". More specifically, the higher dimension is $2n+1$ given that the original function is in a $n$-dimensional space. There were three different approaches developed for constructing the internal functions, $\Phi_{pq}$'s. The first approach was to implement the original Kolmogorov's
construction for $\Phi_{pq}$'s. Although Kolmogorov's construction is theoretically valid, it is very difficult to implement on computers. The second approach was to construct a set of networks to perform as the internal functions. The networks can tune themselves via the feedback error from the environment and the external functions. The most significant difference between the first and the second approaches is the adjustability of the $\Phi_{pq}$'s. In the first approach, $\Phi_{pq}$'s are fixed functions which have been determined in the design stage. In the second approach, $\Phi_{pq}$'s are part of a large multi-layer networks. This approach offered a possibility to "learn" the representation of $\Phi_{pq}$'s while in the implementation stage. In this stage, a particular network structure has been suggested which can construct functions with the desired requirements. However, due to the nonlinearity of the structure, the weight updating procedure is very difficult to develop. A heuristic weight updating procedure, suggested by the author, showed an interesting behavior in approximating some functions. In the third approach, the same concept of "learning" the representation of the internal function is applied. Due to the difficulty that caused by the structure of the networks in the second approach, a simpler structure was suggested. Although it is easier to develop the weight updating procedure under this structure, the non-smooth property no longer held. The results of these approaches are not as good as we expected. Some possible reasons for this were included in the discussion of each approach.
Constructing a model of a system without having the systems structural information is difficult. It is even more difficult if the system's behavior is highly nonlinear. Currently, most of the methods either assume the structure of the observed system is known or use a linear model to approximate the behavior of the system. The former approach suffers from an unrealistic assumption. The latter approach has had very limited success in reconstructing the behavior of highly nonlinear systems. In this research, a method which unifies the concepts of nonparametric approximation and the implementation of parallel processing is introduced and developed. Three major tasks have been studied in this research. They are (1) the localized distributed representation scheme and the associated neural network structure, (2) a neural-network-based nonparametric univariate function approximator, and (3) the Kolmogorov representation theorem and the multivariate function approximator.

The major contributions of this research are as follows.

1. This research demonstrates a fairly flexible and powerful method to reconstruct the behavior of an observed system. It is flexible because no
prior structural information is required. It is powerful because of the on line and iterative nature of the method. Few parallel processing based methods have been developed for system modelling. This work demonstrates a promising research direction of applying the neural network approach in solving a difficult modelling problem.

2. Neural network research, especially the error back propagation method, has suffered from the slow speed and inefficiency of the learning process. This research suggests a new structure for neural networks which has efficient learning performance by using the error back propagation method. This research introduces a new direction of research to investigate the initial structure of the networks. However, structure is, at least, as important as the weight updating method. In addition, a dynamic method of determining the learning rates associated with each unit in the neural networks is developed in this research. This method suggests an alternative way to determining the learning rates along with the system's lifetime. It is different from the traditional method of assigning a fixed value in the design stage.

3. It is always a criticism that purely theoretical research results have not been very useful in real world application. Kolmogorov's representation theorem is among those criticized. Even Lorentz, a very successful mathematician, doubted its usefulness. (Lorentz, 1976). Although we did not resolve the multivariate function approximation
problem completely, the usefulness of Kolmogorov's theorem is demonstrated.

4. This research explains connections between the new developing neural networks approach and the traditional modeling process. This research suggests a new alternative solution to the nonlinear modelling problem and also shows a new field of application for the current neural network research.

FUTURE RESEARCH

The following are recommendations for future research extending to this dissertation.

1. Searching for the "universal change of variables".
   This is a major unsolved problem in this research. Although this issue is more closely related to mathematically oriented research than the neural network or system oriented areas, the importance of this issue is described clearly in the content of this dissertation. There may be, though, an alternative answer to this question. Instead of looking for an isomorphism for all the events in the input space, it is easier to search for a homeomorphism. One of the possible advantages of this approach is to filter out the undesired noise. Some methods have been developed along this direction. One example is the projection pursuit method (Friedman and Stuetzle, 1981; Diaconis and Shahshahani, 1984) which looks for the relatively important affine transformation of
the variables. It would be quite interesting to apply this concept to neural network design.

2. Determining the relationship between the expected approximation error and the number of hidden units in the neural networks.

There was no attempt to optimize the structure of the neural network for solving a particular problem in this research. We constructed one feasible design of the neural networks instead of the best one. It should be possible to investigate the nearly best structure or even the best structure of the neural network for solving a particular problem. One of the most obvious questions is to determine the required number of hidden layer units for achieving some specified approximation error. Some research has been conducted in this direction (Hornik et al, 1989). One of the suggested ways to resolve this problem is to apply the $\varepsilon$-covering and $\varepsilon$-capacity theories that were developed by Kolomogorov.


In this research, we have constructed a neural-network-based nonparametric function approximator. However, the neural network can only perform as one function. This may not be enough for performing a complicated task. Due to the noise and the dynamic nature in the real world, there is no functional behavior, in a static sense, shows in the observed system. This is especially true when the temporal relations within the environment or the inputs are important for determining the behavior of the observed system. In another words, a dynamic function approximator is more desirable. In
the author's opinion, a multi-function approximator will be an appropriate step for future research. However, a dynamic function approximator is the one that we should always looking forward to.
LIST OF REFERENCES


218


Crutchfield, James P. and Bruce S. McNamara, Equations of Motion from a Data Series, *Complex Systems*, Vol.1, 1987, pp. 417-452

Davis, Philip J., Interpolation and Approximation, Dover Publications Inc. New York, 1975

Eubank, Randall L., Spline Smoothing and Nonparametric Regression, Marcel Dekker, Inc. 1988


Hopfield, John, Neural Networks and Physical Systems with Emergent Collective Computational Abilities, Proceedings of the National Academy of Sciences Vol. 79, 1982

Hopfield, John, Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons, Proceedings of the National Academy of Sciences Vol. 81, 1984

Hopfield, John and David Tank, "Neural" Computation and Constraint Satisfaction Problems and the Traveling Salesman, Biological Cybernetics, Vol. 55, 1985

Hornik K., M. Stinchcombe and H. White, Multi-Layer Feedforward Networks are Universal Approximators, Technical Report, Department of Economics, UCSD, June 1988

Kohonen, Teuvo, Self-Organization and Associative Memory, Springer-Verlag, New York, 1988


Metropolis, Nicholas and et al., Equations of State Calculations by Fast Computing Machines, *Journal of Chemical Physics*, Vol. 21, 1953


Minsky, Marvin and Seymour Papert, Perceptrons : An Introduction to Computational Geometry, The MIT Press, 1969


