Abstract

The random neural network (RNN) is a mathematical model for an “integrate and fire” spiking network that closely resembles the stochastic behaviour of neurons in mammalian brains. Since its proposal in 1989, there have been numerous investigations into the RNN’s applications and learning algorithms. Deep learning (DL) has achieved great success in machine learning, but there has been no research into the properties of the RNN for DL to combine their power. This thesis intends to bridge the gap between RNNs and DL, in order to provide powerful DL tools that are faster, and that can potentially be used with less energy expenditure than existing methods.

Based on the RNN function approximator proposed by Gelenbe in 1999, the approximation capability of the RNN is investigated and an efficient classifier is developed.

By combining the RNN, DL and non-negative matrix factorisation, new shallow and multi-layer non-negative autoencoders are developed. The autoencoders are tested on typical image datasets and real-world datasets from different domains, and the test results yield the desired high learning accuracy.

The concept of dense nuclei/clusters is examined, using RNN theory as a basis. In dense nuclei, neurons may interconnect via soma-to-soma interactions and conventional synaptic connections. A mathematical model of the dense nuclei is proposed and the transfer function can be deduced. A multi-layer architecture of the dense nuclei is constructed for DL, whose value is demonstrated by experiments on multi-channel datasets and server-state classification in cloud servers.

A theoretical study into the multi-layer architecture of the standard RNN (MLRNN) for DL is presented. Based on the layer-output analyses, the MLRNN is shown to be a universal function approximator. The effects of the layer number on the learning capability and high-level representation extraction are analysed. A hypothesis for transforming the DL problem into a moment-learning problem is also presented.

The power of the standard RNN for DL is investigated. The ability of the RNN with only positive parameters to conduct image convolution operations is demonstrated. The MLRNN equipped with the developed training algorithm achieves comparable or better classification at a lower computation cost than conventional DL methods.
Declaration of Originality

I herewith certify that the work presented in this thesis is my own. All material in the thesis which is not my own work has been properly referenced and acknowledged.

Copyright Declaration

The copyright of this thesis rests with the author and is made available under a Creative Commons Attribution Non-Commercial No Derivatives licence. Researchers are free to copy, distribute or transmit the thesis on the condition that they attribute it, that they do not use it for commercial purposes and that they do not alter, transform or build upon it. For any reuse or redistribution, researchers must make clear to others the licence terms of this work.
Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Professor Erol Gelenbe, for his persistent care and efforts to address my needs and problems. I am grateful for the opportunities he provided me to travel and broaden my horizon and exchange ideas with researchers around the world. It has been a real privilege and an honour for me to share his tremendous scientific knowledge and deep insights. He has been a great source of new ideas and knowledge to me throughout my research. His guidances have taught me well beyond the academic world, and significantly contributed to shaping my future life.

My thanks go to my colleagues in the research group: Omer, Olivier, Gokce, Huibo, Lan, Mihajlo, Yasin, Elif, Yuancheng, Olumide and Yunxiao, for all the wonderful memories. I am also very grateful to all the colleagues from the Intelligent Systems and Networks group, which has been a source of inspiration, advice, knowledge, collaboration and friendships; and my thanks also go to the group administrators: Patrick and Joan.

I would also like to acknowledge some precious friends for the happy moments we have shared together: Huibo, Lan, Shanxin, Qi, Qianqian, Yixing, Yuancheng, Junlin, Yang, Yunxiao, Shuanglong, Zhaoxu, Jiabao, Yajie, Tingfa, Pang, Xue, Luis and Ye. Particularly, I wish to express my deep gratitude to Xiaoyan Lin for her patience and constant support. I cannot express how grateful I am as she has been with me throughout many obstacles.

Last but not least, I would also like to express my heartfelt thanks to my parents and sisters for their sustained support and love throughout my years of study.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>i</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>v</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivations and Objectives</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1 Random Neural Network and Deep Learning</td>
<td>1</td>
</tr>
<tr>
<td>1.1.2 Challenges in Data Tasks with Neural Networks</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Thesis Contributions</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Thesis Outline</td>
<td>5</td>
</tr>
<tr>
<td>1.4 Published Work</td>
<td>6</td>
</tr>
<tr>
<td>2 Background</td>
<td>8</td>
</tr>
<tr>
<td>2.1 Neural Networks</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Spiking Neural Network and Neuromorphic Computing</td>
<td>10</td>
</tr>
<tr>
<td>2.3 Deep learning</td>
<td>12</td>
</tr>
<tr>
<td>2.4 Random neural network</td>
<td>14</td>
</tr>
</tbody>
</table>
3 A Classifier based on Random Neural Network Function Approximator

3.1 Introduction

3.2 RNN Function Approximator

3.2.1 Lemmas

3.2.2 Theorem

3.3 Learning in RNN Function Approximator

3.3.1 Learning Problem Description

3.3.2 Learning Procedure Design

3.3.3 Computational-Complexity Analysis and Comparison

3.4 Application to Pattern Classification

3.4.1 Experiment Settings

3.4.2 Numerical Results and Comparisons

3.5 Conclusions

4 Non-negative Autoencoders with Simplified Random Neural Network

4.1 Introduction

4.1.1 Technical Background
CONTENTS

4.1.2 Method Overview .................................................. 40

4.2 Random Neural Network in Quasi-Linear Structure ................. 41
  4.2.1 Recurrent RNN .................................................. 41
  4.2.2 Quasi-Linear RNN .............................................. 42

4.3 Shallow Non-negative Autoencoder ................................... 43
  4.3.1 Model Description ............................................... 43
  4.3.2 Problem Formulation .......................................... 45
  4.3.3 Solution .......................................................... 45

4.4 Deep Non-negative Autoencoder ..................................... 48
  4.4.1 Model Description ............................................... 48
  4.4.2 Problem Formulation .......................................... 50
  4.4.3 Solutions .......................................................... 51

4.5 Experimental Evaluations and Comparisons .......................... 53
  4.5.1 Datasets ............................................................. 53
  4.5.2 Convergence Performance in Training .......................... 55
  4.5.3 Dimensionality Reduction for Pattern Classification .......... 58

4.6 Conclusions ............................................................ 62

5 Deep Learning with Dense Random Neural Network .................. 64
  5.1 Introduction .......................................................... 64
    5.1.1 Related Work .................................................. 66
  5.2 Dense Random Neural Network ..................................... 67
5.2.1 Model Description ........................................ 67
5.2.2 Neural Excitation Probabilities ......................... 68
5.3 Modelling Dense Nuclei with Dense Random Neural Network ......................... 69
  5.3.1 Clusters of Identical and Densely Connected Cells ......................... 70
  5.3.2 Condition Analyses for Parameter Setting in Dense Nuclei ................ 73
5.4 Multi-Layer Architectures of Dense Nuclei ................................. 74
  5.4.1 Model Description ........................................ 75
  5.4.2 Learning Problem Formulation .............................. 76
  5.4.3 Efficient Learning Procedure .............................. 76
5.5 MLDRNN for Multi-Channel Datasets .................................. 78
  5.5.1 Modifications to the MCMLDRNN .................................. 79
5.6 Numerical Result Comparisons ...................................... 80
  5.6.1 Application 1: Recognizing 3D Objects ......................... 80
  5.6.2 Application 2: Distinguishing Chemical Gases ....................... 83
  5.6.3 Application 3: Detecting Human Activities ....................... 87
5.7 An Interesting Case Where the Number of Cells in a Nucleus is Sufficiently Large 91
5.8 Conclusions ................................................. 94

6 A Theoretical View into Deep Random Neural Network 95
  6.1 Deep Learning Problem in Multi-Layer RNN ............................ 95
    6.1.1 Problem Transformation ..................................... 97
  6.2 Analysis .................................................. 99
<table>
<thead>
<tr>
<th>6.2.1</th>
<th>The expression of $E[q_{i,1}(X)]$</th>
<th>99</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2.2</td>
<td>The expression of $E[q_{i,1}^2]$</td>
<td>100</td>
</tr>
<tr>
<td>6.2.3</td>
<td>The expression of $E[q_{i,1}^3]$</td>
<td>100</td>
</tr>
<tr>
<td>6.2.4</td>
<td>The expression of $E[q_{i,2}(W_1^+, W_1^-, X)]$</td>
<td>100</td>
</tr>
<tr>
<td>6.2.5</td>
<td>Another form of the expression of $E[q_{i,2}]$</td>
<td>101</td>
</tr>
<tr>
<td>6.2.6</td>
<td>The expression of $E[q_{i,2}^2]$</td>
<td>102</td>
</tr>
<tr>
<td>6.2.7</td>
<td>The expression of $E[q_{i,2}^3]$</td>
<td>104</td>
</tr>
<tr>
<td>6.2.8</td>
<td>The expression of $E[q_{i,3}]$</td>
<td>105</td>
</tr>
<tr>
<td>6.2.9</td>
<td>The expression of $E[q_{i,3}^2]$</td>
<td>106</td>
</tr>
<tr>
<td>6.2.10</td>
<td>The expression of $E[q_{i,L}]$</td>
<td>106</td>
</tr>
<tr>
<td>6.2.11</td>
<td>The expression of $E[q_{i,L}y_i]$</td>
<td>107</td>
</tr>
<tr>
<td>6.2.12</td>
<td>The expression of $E[q_{i,L}^2]$ and $Var[q_{i,L}]$</td>
<td>107</td>
</tr>
<tr>
<td>6.3</td>
<td>Theorem and Hypothesis</td>
<td>108</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Universal Approximation Property</td>
<td>108</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Learning Efficiency in a Deep Architecture</td>
<td>110</td>
</tr>
<tr>
<td>6.3.3</td>
<td>High-Level Representation Extraction</td>
<td>112</td>
</tr>
<tr>
<td>6.3.4</td>
<td>Moment Learning Hypothesis</td>
<td>113</td>
</tr>
<tr>
<td>6.4</td>
<td>Conclusions</td>
<td>115</td>
</tr>
<tr>
<td>7</td>
<td>Deep Learning with Standard Random Neural Network</td>
<td>116</td>
</tr>
<tr>
<td>7.1</td>
<td>Introduction</td>
<td>116</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Related Work and Discussion</td>
<td>117</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>7.2</td>
<td>Convolution Operation with RNN: Two Approaches</td>
<td>118</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Single-Cell Approach</td>
<td>118</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Twin-Cell Approach</td>
<td>120</td>
</tr>
<tr>
<td>7.3</td>
<td>Convolution with RNN: a Cluster Approach</td>
<td>121</td>
</tr>
<tr>
<td>7.3.1</td>
<td>First-Order Approximation of the RNN Equation</td>
<td>121</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Relationship between RNN and ReLU</td>
<td>123</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Cluster-Based Convolution</td>
<td>125</td>
</tr>
<tr>
<td>7.4</td>
<td>Numerical Verification of RNN Convolution</td>
<td>126</td>
</tr>
<tr>
<td>7.5</td>
<td>Individual-Cell Based Multi-Layer RNN</td>
<td>127</td>
</tr>
<tr>
<td>7.5.1</td>
<td>Mathematical Model Formulation in Scalar-Form</td>
<td>127</td>
</tr>
<tr>
<td>7.5.2</td>
<td>Mathematical Model Formulation in Matrix-Form</td>
<td>130</td>
</tr>
<tr>
<td>7.5.3</td>
<td>Learning Problem Description</td>
<td>131</td>
</tr>
<tr>
<td>7.5.4</td>
<td>Training Procedure for MLRNN</td>
<td>131</td>
</tr>
<tr>
<td>7.5.5</td>
<td>MCMLRNN: Mathematical Model and Training Procedure</td>
<td>136</td>
</tr>
<tr>
<td>7.6</td>
<td>Numerical Results</td>
<td>137</td>
</tr>
<tr>
<td>7.7</td>
<td>Conclusion</td>
<td>140</td>
</tr>
<tr>
<td>8</td>
<td>Conclusions and Future Work</td>
<td>141</td>
</tr>
<tr>
<td>8.1</td>
<td>Summary of Achievements</td>
<td>141</td>
</tr>
<tr>
<td>8.2</td>
<td>Future Work</td>
<td>143</td>
</tr>
</tbody>
</table>

Bibliography | 144
Appendices

A  Proof of Lemmas 3.3 to 3.5

B  UCI Dataset Descriptions

C  Deduction of Λ⁺ₙ and Λ⁻ₙ in (5.4)

D  Applications of Deep Learning With Random Neural Networks

D.1  Server-State Classification in Cloud Servers

D.1.1  Introduction

D.1.2  Classification Dataset Construction

D.1.3  Classification Accuracy Comparison

D.1.4  Conclusion

D.2  Toxicity Prediction of Chemical Compounds

D.2.1  Introduction

D.2.2  Formalisation

D.2.3  Learning Procedure

D.2.4  Classification Results

D.2.5  Conclusions
List of Tables

3.1 Minimum numbers of operations required to compute the RNN function approximator, SOCPNN and MLP with \( N \) inputs, \( K \) hidden units and one output once ......................................................... 30

3.2 Attribute, instance and class numbers of UCI real-world datasets from different areas ................................................. 32

3.3 Comparison of testing accuracy rates of RNN, SOCPNN, MOCPNN, ELM and MLP as well as regularized RBFNN and SVM for pattern classification ........................................ 33

3.4 Comparison of testing time of RNN, SOCPNN, MOCPNN, ELM and MLP as well as regularized RBFNN and SVM for pattern classification ........................................ 34

3.5 Comparison of numbers of hidden units (or basis-function units)/support vectors of RNN, SOCPNN, MOCPNN, ELM and MLP (with numbers of hidden units determined by cross validation) as well as regularized RBFNN and SVM for pattern classification ........................................ 36

4.1 Testing accuracies (%) using different encoding algorithms for the MNIST, Yale face, CIFAR-10 and UCI datastes in classification ........................................ 61

5.1 Training and testing accuracies (%) and time (s) of different methods on NORB dataset for recognizing 3D objects ........................................ 82

5.2 Structure settings of different methods to handle the TGSA dataset ........................................ 85
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3</td>
<td>Training and testing accuracies (%) and time (s) of different methods on Task 1 of TGSA dataset for distinguishing chemical gases.</td>
</tr>
<tr>
<td>5.4</td>
<td>Training and testing accuracies (%) and time (s) of different methods on Task 2 of TGSA dataset for distinguishing chemical gases.</td>
</tr>
<tr>
<td>5.5</td>
<td>Confusion matrix of MCMLDRNN to classify the testing subdataset in Task 1 of TGSA dataset for distinguishing chemical gases.</td>
</tr>
<tr>
<td>5.6</td>
<td>Confusion matrix of MCMLDRNN to classify the testing subdataset in Task 2 of TGSA dataset for distinguishing chemical gases.</td>
</tr>
<tr>
<td>5.7</td>
<td>Training and testing accuracies (%) and time (s) of MCMLDRNN on Tasks 3 to 6 (Units 2 to 5) of TGSA dataset for distinguishing chemical gases.</td>
</tr>
<tr>
<td>5.8</td>
<td>Training and testing accuracies (%) and time (s) of different methods on DSA dataset for detecting human activities.</td>
</tr>
<tr>
<td>5.9</td>
<td>Confusion matrix of MCMLDRNN to classify the DSA testing subdataset for detecting human activities.</td>
</tr>
<tr>
<td>5.10</td>
<td>Results by using dense nuclei (5.13) and very dense nuclei (5.20) in 100 trials to classify DSA dataset.</td>
</tr>
<tr>
<td>7.1</td>
<td>Testing accuracies (%), training and testing time (s) of different methods for NORB and DAS datasets.</td>
</tr>
<tr>
<td>7.2</td>
<td>Testing accuracies (%), training and testing time (s) of different methods for TGSA dataset.</td>
</tr>
<tr>
<td>D.1</td>
<td>Features of classification datasets collected from different servers.</td>
</tr>
<tr>
<td>D.2</td>
<td>Testing accuracies (%) and training time (s) of different methods for state detection of Server 1 with Task 1.</td>
</tr>
</tbody>
</table>
D.3 Testing accuracies (%) and training time (s) of different methods for state detection of Server 2 with Task 2. ................................. 184

D.4 Testing accuracies (%) and training time (s) of different methods for state detection of Server 3 with Task 2. ................................. 185
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Schematic representation of a spiking RNN system.</td>
<td>15</td>
</tr>
<tr>
<td>3.1</td>
<td>Schematic representation of the RNN function approximator</td>
<td>24</td>
</tr>
<tr>
<td>4.1</td>
<td>Structure of a quasi-linear RNN.</td>
<td>43</td>
</tr>
<tr>
<td>4.2</td>
<td>Structure of a shallow non-negative LRNN autoencoder.</td>
<td>44</td>
</tr>
<tr>
<td>4.3</td>
<td>Structure of the deep non-negative LRNN autoencoder.</td>
<td>50</td>
</tr>
<tr>
<td>4.4</td>
<td>Examples of 10 classes in the MNIST dataset.</td>
<td>54</td>
</tr>
<tr>
<td>4.5</td>
<td>Examples of 10 of 15 classes in the Yale face dataset.</td>
<td>54</td>
</tr>
<tr>
<td>4.6</td>
<td>Examples of 10 classes in the CIFAR-10 dataset. The first row includes airplane, automobile, bird, cat, deer: the second row includes dog, frog, horse, ship, truck.</td>
<td>55</td>
</tr>
<tr>
<td>4.7</td>
<td>Reconstruction error (Y-axis) versus iteration number (X-axis) in training shallow and multi-layer LRNN autoencoders for the MNIST dataset.</td>
<td>56</td>
</tr>
<tr>
<td>4.8</td>
<td>Reconstruction error (Y-axis) versus iteration number (X-axis) in training shallow and multi-layer LRNN autoencoders for the Yale face dataset.</td>
<td>57</td>
</tr>
<tr>
<td>4.9</td>
<td>Reconstruction error (Y-axis) versus iteration number (X-axis) in training shallow and multi-layer LRNN autoencoders for the CIFAR-10 dataset.</td>
<td>57</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>4.10</td>
<td>Reconstruction error (Y-axis) versus iteration number (X-axis) in training LRNN autoencoders for UCI real-world datasets: IRIS, TAE, LD, Seeds, PID.</td>
<td></td>
</tr>
<tr>
<td>4.11</td>
<td>Reconstruction error (Y-axis) versus iteration number (X-axis) in training LRNN autoencoders for UCI real-world datasets: Glass, Wine, Zoo, Parkinsons, WFRN, Ionosphere.</td>
<td></td>
</tr>
<tr>
<td>4.12</td>
<td>Reconstruction error (Y-axis) versus iteration number (X-axis) in training LRNN autoencoders for UCI real-world datasets: SL, FOTP, Sonar, CA.</td>
<td></td>
</tr>
<tr>
<td>4.13</td>
<td>Testing accuracies (%) versus input-dimension numbers of datasets using different encoding algorithms for the MNIST, Yale face, CIFAR-10 and UCI datasets in classification.</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>Cluster activation $\zeta(x)$ versus input $x$ under different parameters for a twenty-cell cluster ($N = 20$) with $p = 0.05$.</td>
<td></td>
</tr>
<tr>
<td>5.2</td>
<td>Schematic representation of the MLDRNN.</td>
<td></td>
</tr>
<tr>
<td>5.3</td>
<td>Schematic representation of the MCMLDRNN.</td>
<td></td>
</tr>
<tr>
<td>5.4</td>
<td>Schematic representation of the MCMLDRNN1.</td>
<td></td>
</tr>
<tr>
<td>5.5</td>
<td>Examples of two images of each class in the NORB dataset: four-legged animals, human figures, airplanes, trucks and cars.</td>
<td></td>
</tr>
<tr>
<td>5.6</td>
<td>Examples of 8 30-second segments of each class in the TGSA dataset: Carbon Monoxide, Ethanol, Ethylene, Methane (from left to right).</td>
<td></td>
</tr>
</tbody>
</table>
5.7 Examples of 45 5-seconds segments of each class in the DSA dataset: sitting, standing, lying on back, lying on right side, ascending stairs (A1-5); descending stairs, standing in an elevator still, moving around in an elevator, walking in a parking lot, walking on a treadmill with a speed of 4 km/h in flat position (A6-10); walking on a treadmill with a speed of 4 km/h in 15 deg inclined position, running on a treadmill with a speed of 8 km/h, exercising on a stepper, exercising on a cross trainer, cycling on an exercise bike in horizontal position (A11-15); cycling on an exercise bike in vertical position, rowing, jumping, playing basketball (A16-19).

5.8 Training and testing results (Y-axis) versus different trial index (X-axis) by using dense nuclei (5.13) and very dense nuclei (5.20) to classify DSA dataset.

7.1 An RNN convolution operation with the single-cell approach.

7.2 An RNN convolution operation with the twin-cell approach.

7.3 A single unit with ReLU activation.

7.4 A cluster of the LRNN-I and LRNN-E cells to approximate the ReLU activation.

7.5 An RNN convolution operation with the cluster approach.

7.6 The original image.

7.7 Gray-inverted output of a standard convolution operation with ReLU activation.

7.8 Output of an RNN convolution operation with the single-cell approach.

7.9 Output of an RNN convolution operation with the twin-cell approach.

7.10 Output of an RNN convolution operation with the cluster approach.

7.11 Schematic representation of the MLRNN.

7.12 Schematic representation of the MCMLRNN.
D.1 CPU utilization percentage and execution time collected in Server 1 using Task 1 under stochastic background loads. ......................................................... 179

D.2 CPU utilization percentage and execution time collected in Server 2 using Task 2 under stochastic background loads. ......................................................... 180

D.3 CPU utilization percentage and execution time collected in Server 3 using Task 2 under stochastic background loads. ......................................................... 181

D.4 Testing accuracies of all 100 trials of H-ELM and MLDRNN for state detection of Server 1 with Task 1. ................................................................. 182

D.5 Testing accuracies of all 100 trials of H-ELM and MLDRNN for state detection of Server 2 with Task 2. ................................................................. 183

D.6 Testing accuracies of all 100 trials of H-ELM and MLDRNN for state detection of Server 3 with Task 2. ................................................................. 183

D.7 Training and testing mean-value results (Y-axis) versus different assays (X-axis) when the MLP, CNN, DenseRNN, MLRNN, XGBoost, RNN, RBFN and SVM are used to classify the toxicity dataset. ......................................... 192

D.8 Training and testing mean-value results (Y-axis) versus different assays (X-axis) when the MLP, CNN, DenseRNN, MLRNN, XGBoost, RNN, RBFN and SVM are used to classify the toxicity dataset with data augmentation. ....................... 192

D.9 The highest testing results (Y-axis) versus different assay index (X-axis) of the MLP, CNN, DenseRNN, MLRNN, XGBoost, RNN, RBFN and SVM to classify the toxicity dataset before and after data augmentation. ................. 193
Chapter 1

Introduction

1.1 Motivations and Objectives

1.1.1 Random Neural Network and Deep Learning

Originally developed to mimic the behaviour of the brain’s biological neurons, the random neural network (RNN) proposed by Professor Gelenbe functions as a stochastic integer state “integrate-and-fire” system [Gel89, Gel90, Gel93]. In an RNN, an arbitrarily large set of neurons interact with each other via excitatory and inhibitory spikes that modify each neuron’s action potential in continuous time. It has been proved that, in a steady state, the stochastic spiking behaviours of the RNN have a remarkable property called “product form”, with the state probability distribution given by an easily-solvable system of nonlinear equations. Compared to many other neural network models, e.g., the multi-layer perceptron (MLP) [RHW86], the radial basis function (RBF) neural network [PS93], extreme learning machines (ELM) [CHK+13] and orthogonal polynomial networks (OPN) [ZYG+14a], the RNN has the following advantages:

1) The model is closer to the biological neuronal network and can better represent how signals are transmitted through the human brain, whereby the complex stochastic behaviours among neurons can be modelled and analysed more easily in a mathematical form.

2) In general, it has a better predictive capability because of its non-negativity and probability
3) Owing to its highly-distributed nature, this can potentially be deployed in either energy-efficient customised or neuromorphic hardware.

4) Its outputs can be more interpretable. Via linearisation and approximation, the expressions of the RNN outputs are decomposable. This facilitates the analysis of its approximation capability and the impact of the multi-layer architecture on the learning and high-level representation extraction from a theoretical point of view.

Inspired by the structure and function of the brain, deep learning functions as a machine-learning approach [HS06, LBH15]. Most computational models in deep learning exploit a feed-forward neural-network architecture composed of multi-processing layers; this architecture allows the model to extract high-level representations from the raw data. In this manner, essential and useful information can be extracted from this raw data so that the problem of building complex mapping from the raw inputs to the desired outputs becomes more solvable. Deep learning has achieved major advances in solving many historic issues in machine learning.

Indeed, since its proposal in 1989, there have been numerous investigations into the RNN’s applications and learning algorithms [BK00, Tim10, GLK11a]; however, there has not been any research regarding the property of the RNN for deep learning. Therefore, connecting the RNN with deep learning is the main objective of this thesis, which is novel in the following two aspects: 1) the research could provide powerful and deep-learning tools that not only achieve state-of-the-art performance but can potentially be used with less energy expenditure than existing methods; 2) the research may provide a better understanding of the relationship between deep learning and the brain.

1.1.2 Challenges in Data Tasks with Neural Networks

Neural networks, e.g., the MLP [RHW86], RBF-based neural networks [PS93], ELM [CHK+13], OPN [ZYG+14a] and RNN [Ge89, Ge90, Ge93], are powerful tools with which to solve data-related tasks, such as dimension reduction and pattern classification. However, developing an efficient and effective neural-network learning tool that is capable of handling diversified data
types obtained from various systems presents various challenges, such as the over-fitting effect, diversified attribute characteristics, input noise, a fast-changing environment, and speed requirements, some of which are detailed as the following.

1) In many practical scenarios, the collection and labelling of samples could be expensive (e.g., in the biomedical and chemical fields), because they may require long-term observation, measurements from large-scale equipment, experiments based on certain environments, expert opinions, and so forth. The size of these related datasets available for learning could be relatively small, compared to large publicly-available image datasets for object-recognition tasks. Therefore, an effective learning tool is required to learn from a small dataset whilst not over-fitting the data.

2) The data may be collected using different equipment and can be measured in various forms, leading to diversified attribute characteristics. Thus, regarding the input attribute types, the requirement for the learning tools performance is that it be robust.

3) The input dimension of the data could be high, while some of the input data might be redundant or some may be noisy. In this case, an efficient learning tool should be able to conduct dimension reduction so as to extract the most valuable information that contributes to the final learning tasks.

4) There could be a dynamic change to the environment under which data is collected or the learning system is executed. Or, in a large system constituted by multiple subsystems - e.g., a network system - each subsystem requires a customised learning system based on its own characteristics. Therefore, an effective learning system should have a fast learning speed with stable performance, thereby enabling rapid deployment of multiple learning systems.

5) Decision delays in the testing phase of neural networks may harm the overall performance of the systems, such as task allocation in network systems [WG15, YWG17]. In addition, the computers - or, more generally, the hardware systems - in which the trained learning systems are deployed, may have limited computational resources. Consequently, a desirable learning system should have low computational complexity in its testing/inference phase.

It is extremely demanding to develop learning tools with the capability of addressing the challenges outlined above. Therefore, another objective of this thesis is to investigate and utilise
Chapter 1. Introduction

the power of the RNN to develop broadly efficient and effective neural-network learning tools.

1.2 Thesis Contributions

The contributions of this thesis can be summarised into the following aspects:

- An efficient classifier has been proposed based on the RNN function approximator, with universal approximation capability and low computational complexity.

- New non-negative autoencoders have been proposed for dimension reduction, and are robust and effective in handling various types of data.

- There has been a mathematical investigation into the idea of dense nuclei in the human brain. In these dense nuclei, neurons may interconnect via soma-to-soma interactions, in addition to conventional synaptic connections. Furthermore, the mathematical model has been developed and a simple transfer function has been deduced.

- An efficient, deep-learning tool based on the dense nuclei has been developed, and which can be more accurate than the state-of-the-art methods with high learning efficiency in many cases.

- Three approaches for applying the RNN to conducting image convolution operations have been presented.

- A deep-learning tool has been proposed based on the RNN. This tool has lower computational complexity than the conventional deep-learning tools and is capable of achieving the highest testing accuracy in many cases.

- A approach to transforming the inference of the states of servers in the Cloud system into a classification problem has been presented.

- Incorporating the learning capability and multi-layer architecture’s impact, the problem of deep-learning was investigated from both the theoretical and statistical perspectives.
These investigations focused on the case of the multi-layer RNN, with the potential to
generalise other forms of deep neural networks.

1.3 Thesis Outline

This thesis is organised into the following chapters:

- The backgrounds and techniques of neural networks and deep learning are referenced in
  Chapter 2. Literature pertaining to the RNN is reviewed in terms of its proposal, variants,
  approximation property, learning algorithms, and applications.

- Chapter 3 constructs a classifier based on the RNN function approximator in [GML99a,
  GML99b, GML04], develops an efficient learning algorithm for the classifier, and then
tests it on various real-world datasets with diversified attribute characteristics from various
perspectives. Theoretical analysis confirms its low computational complexity and
numerical results verify that the proposed classifier is a generally efficient classification
tool that is able to achieve better classification performance than five other classifiers in
many cases, and often with less computational time.

- Exploited as a feed-forward quasi-linear structure, the RNN is constructed as non-negative
  autoencoders in Chapter 4. The proposed autoencoders are evaluated on both real-world
datasets and image datasets for dimension reduction in order to improve classification
performance. The numerical results have demonstrated that the RNN autoencoders are
both robust and effective in handling various data types.

- Based on the RNN theory, Chapter 5 pursues the idea that the human brain contains
  important areas composed of dense clusters of cells, and develops a mathematical model of
dense clusters/nuclei, named the “dense RNN”. Subsequently, a multi-layer architecture of
dense nuclei is constructed with an efficient learning procedure developed. Subsequently,
the proposed dense RNN learning tool is applied for recognising 3D objects, distinguishing
different chemical gases, and detecting humans’ activities. The corresponding numerical
results indicate that it is more accurate than the state-of-the-art methods in most cases, and possesses a high learning efficiency.

- In Chapter 6, a theoretical study into the multi-layer RNN (MLRNN) for deep learning is presented. Ultimately transformed into a series of moments, mathematical formulation of the deep-learning problem of the MLRNN was undertaken. Subsequently, theoretical analyses of the moments of the layer outputs of the MLRNN are presented. Based on these analyses, the MLRNN is shown to be a universal function approximator. The effect of the number of layers in the MLRNN on the learning capability and high-level representations extraction is also analysed. A hypothesis for transforming the deep-learning problem into a moment-learning problem is also presented, thus laying the foundation for further work.

- Chapter 7 investigates the original, simpler structure of the RNN and demonstrates the power of single standard RNN cells for deep learning in two aspects. In the first, we show that - with only positive parameters - the RNN is capable of implementing image convolution operations similar to convolutional neural networks. In the second, we build a multi-layer architecture composed of standard RNN cells (i.e., the MLRNN) with low computational complexity. Moreover, numerical results regarding multi-channel classification datasets show that the MLRNN is effective and is arguably the most efficient of the five different deep-learning approaches.

1.4 Published Work

1. Ingrid Grenet, Yonghua Yin, Jean-Paul Comet and Erol Gelenbe. “Machine Learning to Predict Toxicity of Compounds.” submitted to The 27th International Conference on Artificial Neural Networks (ICANN), 2018.


Chapter 2

Background

2.1 Neural Networks

In 1943, McCulloch and Pitts [MP43] built mathematical models of neural networks to describe how the neurons in the brain might work, where the neural behaviours are treated by means of propositional logic. This is the first computational model of neuron activity. In [RHHD56], Rochester conducted two sets of simulation experiments representing neural networks with 69 and 512 neurons, respectively, using digital computers, in order to test theories on how the brain works. The theory in [Ros58] was developed for a hypothetical nervous system, called a perceptron, which detailed the organisation of a perceptron in the terms of impulse types, connection patterns and so on, and the mathematical analysis of learning in the perceptron.

In 1989, [HSW89] proved that a multi-layer feed-forward neural network, in other terms, multi-layer perceptron (MLP), with as few as one hidden layer can approximate any continuous function uniformly, provided that the hidden-layer neurons use non-decreasing squashing functions $\mathbb{R} \rightarrow [0, 1]$, e.g., sigmoid. The hidden-layer output is of the form $\sigma(\sum_{i=1}^{N}(\beta_i x_i + b_i))$, where $x_i$, $N$, $\beta_i$, $b_i$ and $\sigma(\cdot)$ are respectively the input, input dimension, connection weight, threshold and activation function. This form is called the ridge function (RF) [ST98]. [SW89] showed that sigmoid functions are not necessary for universal approximation. Further work by [Hor91] investigated the approximation capability of the MLP with bounded and nonconstant...
activation functions. A generalised result from [Hor91] in [LLPS93] relaxes the condition, i.e., an MLP with non-polynomial activation functions is a universal approximator.

In addition to the RF-based neural networks, there are other types. The radial basis function (RBF) based neural networks are also widely used, which have the hidden-output form \( \sigma(||X - C||/a) \), where \( X \in \mathbb{R}^{N \times 1} \) is the input, \( C \in \mathbb{R}^{N \times 1} \) and \( a \) are respectively the centre and impact factor of the hidden node. [PS91, PS93] proved the universal approximation property of the RBF-based neural networks. Another type of neural network uses a product-based orthogonal-polynomial (POP) activation function in [ZYG14a, ZYG14b]. The hidden-output form is \( \prod_{i=1}^{N} p_i(x_i) \), where \( p_i(\cdot) \) is an orthogonal polynomial. The POP neural networks are also universal approximators. It is also proved that the POP-based single-hidden-layer neural network (SLNN) in [ZYG14a] has lower computational complexity than the RF-based one.

With recent improvements, back propagation (BP) has become a standard method to train neural networks. For example, the ReLU units make neural networks easier to train using BP [GBB11]. Weight-sharing strategies, e.g., the type used in convolutional neural networks, reduce the number of parameters needed to be trained, which facilitates the training of multi-layer neural networks. Training a neural network with BP well can be time-consuming and require huge computation load because of the possible slow convergence and potential instability inherent in the training process. This issue has provided the rationale for recent research focusing on the subject of neural networks that adopted the perspectives of both the approximation property and training efficiency.

Huang [HZS06] investigated RF-based neural networks and pointed out that most results on their universal approximation properties are based on the assumption that all weights and biases (including \( \beta_i \) and \( b_i \)) need to be adjusted. As a result, the training of these networks must proceed at a slower pace. Huang [HZS06] proved that, using randomly-generated \( \beta_i \) and \( b_i \), a RF-based single-hidden-layer feed-forward neural network (SLFN) with \( N \) hidden nodes can learn \( N \) distinct samples exactly. Therefore, the SLFN can be described as a linear system whose output weights can be analytically determined. This is the extreme learning machine (ELM) concept. Learning systems based on the ELM can be trained more than a
hundred times faster than the BP-based ones without compromising accuracy in many cases \[ \text{T} \text{DH}16 \]. Zhang \[ \text{ZYG}^+14a, \text{ZYG}^+14b \] focused on the POP neural network. Theoretical results in \[ \text{ZYG}^+14a, \text{ZYG}^+14b \] present its universal approximation property. Similar to the ELM concept, the POP neural network can then be described as a linear system with the output weights analytically determined, which is the weights-direct-determination (WDD) concept.

### 2.2 Spiking Neural Network and Neuromorphic Computing

Von Neumann architecture \[ \text{BGvN}46 \] has provided the basis for many major advances in modern computing \[ \text{CWS}^+16, \text{WWW}^+16 \], allowing for the rapid decrease in transistor size, and improved performance without increasing costs. However, the rate of progress experienced by this type of architecture has recently slowed down, and with various seemingly insurmountable obstacles preventing further significant gains, the end of Moores Law is in sight \[ \text{Pav}15 \]. One major obstacle is that, due to quantum-mechanical effects, transistors smaller than 65 nanometres can no longer switch between “on” and “off” reliably, which means that a digital computer cannot distinguish one from zero. The frequency wall is also a major obstacle, which means that the hardware could melt down because of excess heat if the CPU frequency exceeds four billion times per second. Therefore, a new machine that goes beyond the boundaries imposed by von Neumann architecture needs to be developed, and the areas of neuromorphic computing and spiking neural network (SNN) are starting to attract significant attention from researchers in both academic and commercial institutions.

A non-von Neumann architecture is the TrueNorth cognitive computing system, which has been developed by IBM and is inspired by how organic brains handle complex tasks easily while being extremely energy-efficient \[ \text{EAA}^+13, \text{PWD}^+12, \text{DMJFRN}^+16 \]. The TrueNorth chip has received a lot of attention from wider society since it is the first large-scale commercial spiking-based neuromorphic computing platform \[ \text{CWS}^+16 \]. A first major step occurred when IBM researchers presented an architectural simulator, called Compass, for large-scale network
2.2. Spiking Neural Network and Neuromorphic Computing

models of TrueNorth cores [PWD+12]. Since the Compass simulator is functionally equivalent to TrueNorth and runs on a von Neumann computer, it facilitates research and development for TrueNorth at this stage. Based on this simulator, in [EAA+13], applications - including speaker recognition, digital recognition and so on - are integrated to demonstrate the feasibility of ten different algorithms, such as convolution networks, liquid state machines and so on, to be run in TrueNorth. The conventional neural network algorithms require high precision while the synaptic weights and spikes in TrueNorth have low quantisation resolution, which is the issue investigated in [WWW+16, CWS+16]. Considering that TrueNorth is spiking-based, naturally, mapping models of spiking neural networks into TrueNorth could be a reasonable choice [WWW+16, CWS+16]. In [WWW+16] and [CWS+16], a single-hidden-layer neural network was deployed into TrueNorth with the learning algorithm presented, where the nonlinearity is brought by the cumulative probability function. The research in [DPC+16] implemented a sentiment-analysis algorithm on both the spiking-neural-network simulator and TrueNorth system. First, a fully-connected single-hidden-layer neural network (FCNN) with rectified linear units was trained using the stochastic gradient descent. Then, this FCNN was converted into a SNN model with a discussion on the related issues needed to be addressed, e.g., the precision and the negative inputs. There is a performance drop in this conversion. Finally, the SNN was mapped into TrueNorth. It was mentioned that performance loss in this step is more significant than that through the FCNN-SNN conversion.

The spiking neural network architecture (SpiNNaker) project is another spike-based neuromorphic computing platform [FGTP14, KVM+16, DMJFRN+16]. The SpiNNaker aims to model a large-scale spiking neural network in biological real time. The paper [FGTP14], published in 2014, presented an overview of the SpiNNaker architecture and hardware implementation, introduced its system software and available API and described typical applications. A research project on SpiNNaker in [KVM+16] presented an application for audio classification by mapping a trained one-hidden-layer leaky integrate-and-fire SNN into the SpiNNaker, utilising the PyNN library [DBK+09] that is a common interface for different SNN simulators including NEURON [CH06], NEST [GD07] and Brian [GB08].

Recently, SNN-based neuromorphic computing has made rapid progress by combining its own
inherent benefits and deep learning [EMA+16]. By enforcing connectivity constraints, the work in [EMA+16] applied the deep convolutional network structure into neuromorphic systems. Conventional convolutional networks require high precision while neuromorphic design uses only low-precision one-bit spikes. However, it was demonstrated that, by introducing constraints and approximation for the neurons and synapses, it is possible to adapt the widely-used backpropagation training technique. This enables the creation of a SNN that is directly implementable into a neuromorphic system with low-precision synapses and neurons [EAM+15]. Comparisons based on 8 typical datasets demonstrated that this neuromorphic system achieves comparable or higher accuracies compared with start-of-the-art deep learning methods. High energy efficiency is maintained throughout this process.

2.3 Deep learning

Deep learning [HS06, LBH15] has achieved great success in machine learning and is currently contributing to huge advances in solving complex practical problems compared with traditional methods. For example, based on deep convolutional neural networks (CNN), which are powerful in handling computer vision tasks, deep learning systems can be designed to realise the dream of self-driving cars [BDD+16]. Another example is the game of Go, invented in ancient China more than 2,500 years ago, which was one of the most challenging classic games for artificial intelligence since the number of possible moves is greater than the total number of atoms in the visible universe [Go]. However, a major step was recently taken when a computer system based on deep neural networks, a product of the Google DeepMind laboratories, defeated some of the world’s top human players [SHM+16].

Many computational models used in deep learning exploit a feed-forward neural-network architecture that is composed of multi-processing layers, which allows the model to extract high-level representations from raw data. The feed-forward fully-connected multi-layer neural network, also known as multi-layer perceptron (MLP), has already been investigated for decades [Ros61]. However, progress on the MLP was slow since researchers found difficulties in training MLPs
with more than three layers [GB10]. Furthermore, adding more layers did not seem to help improve the performance. Pre-training the MLP layer by layer is a great advance [HS06, HOT06], and this breakthrough demonstrates that it is possible to train the MLP and adding more layers can actually improve the network performance. This method is still very useful due to its wide adaptability. Recent literature shows that utilising the rectified linear unit (ReLU) reduces the difficulty in the training of deep neural networks [GBB11]. The typical training procedure called stochastic gradient descent (SGD) provides a practical choice for handling large datasets [BB08].

A considerable body of successful work has been amassed thanks to LeCun’s inspirational work on CNN [LBBH98]. The weight-sharing property in the convolutional structure significantly reduces the number of weights to be learned and the local connectivity pattern allows good feature extraction, which facilitates the training of a deep CNN. Recent work on deep residual learning has made it possible to train extremely deep neural networks with thousands of layers [HZRS16].

The task of supervised learning in the DNN can be handled using the greedy layer-wise unsupervised learning, convolution structures and residual learning. Given the ease with which humans are able to learn in an unsupervised manner, and the ready availability of raw datasets - which is not the case with labelled datasets - unsupervised learning has become a key issue upon which the deep learning community has begun to focus. A popular topic in unsupervised learning is the generative adversarial nets (GAN) system [GPAM+14], which includes a discriminative and generative models. The generative model that takes noise as input tries to generate fake samples to cheat the discriminative model such that it cannot distinguish between fake and true samples. The GAN can be used to model natural images and handle image generation tasks [RMC15]. The adversarial autoencoders (AAE) are proposed based on the GAN [MSJG15]. The work in [MSJG15] shows how the AEE can be used in different applications, e.g., semi-supervised classification, unsupervised clustering, dimensionality reduction and data visualisation.
2.4 Random neural network

The random neural network (RNN) proposed by Professor Gelenbe in 1989 [Gel89, Gel90, Gel93] is a stochastic integer-state “integrate and fire” system and was developed to mimic the behaviour of biological neurons in the brain. The RNN is also a recurrent neural network. Note that, in this thesis, to prevent any potential confusion, the acronym “RNN” refers to “random neural network” while the acronym “RecurrentNN” refers to “recurrent neural network”. In an RNN, an arbitrarily large set of neurons interact with each other via excitatory and inhibitory spikes which modify each neuron’s action potential in continuous time. The power of the RNN is derived from the fact that, in steady state, the stochastic spiking behaviors of the network have a remarkable property called “product form” and that the state probability distribution is given by an easily solvable system of non-linear equations. In addition, the RNN models the biological neuronal network with greater accuracy, and offers an improved representation of signal transmission in the human brain, when compared to alternative neural network models, e.g., the MLP [RHW85], ELM [HZS06] and OPN [ZYG14a, ZYG14b].

2.4.1 Model Description

Within the RNN system with $L$ neurons, all neurons communicate with each other with stochastic unit amplitude spikes while receiving external spikes. The potential of the $l$-th neuron, denoted by $k(t) \geq 0$, is dynamically changing in continuous time, and the $l$-th neuron is said to be excited if its potential $k(t)$ is larger than zero. An excitatory spike arrival to the $l$-th neuron increases its potential by 1, denoted by $k(t^+) \leftarrow k(t) + 1$; while an inhibitory spike arrival decreases its potential by 1 if it is larger than zero, denoted by $k(t^+) \leftarrow \max(k(t) - 1, 0)$, where $\max(a, b)$ produces the larger element between $a$ and $b$.

For better illustration, Figure 2.1 presents the schematic representation of the RNN system. Excitatory spikes arrive at the $l$-th neuron from the outside world according to Poisson processes of rate $\Lambda_l$, which means that the probability that there are $g$ excitatory spike arrivals in time interval $\Delta t$ is [Hee00]: $\text{Prob}(g \text{ spike arrivals in interval } \Delta t) = (\Lambda_l \Delta t)^g e^{-\Lambda_t \Delta t} / g!$, where $g$ is a
2.4. Random neural network

non-negative integer. In addition, the rate of inhibitory Poisson spike arrivals from the outside world to the \( l \)-th neuron is \( \lambda_l \). When the \( l \)-th neuron is excited, it may fire excitatory or inhibitory spikes towards other neurons with the inter-firing interval \( \bar{\vartheta} \) being exponentially distributed, which means that the probability density function of \( \bar{\vartheta} \) is \( \text{Prob}(\bar{\vartheta} = \Delta t) = r_l e^{-r_l \Delta t} \), where \( r_l \) is the firing rate of the \( l \)-th neuron. When the \( l \)-th neuron fires a spike, its potential is decreased by 1. The fired spike heads for the \( \hat{l} \)-th neuron as an excitatory spike with probability \( p^{+}_{l,\hat{l}} \) or as an inhibitory spike with probability \( p^{-}_{l,\hat{l}} \), or it departs from the network/system with probability \( \nu_l \). The summation of these probabilities is 1:

\[
\sum_{i=1}^{L} (p^{+}_{l,i} + p^{-}_{l,i}) + \nu_l = 1.
\]

Evidently, the potentials of the \( L \) neurons in the system are dynamically changing over time due to the stochastic spikes and firing events. Let \( \text{Prob}(k_l(t) > 0) \) denote the probability that the \( l \)-th neuron is excited at time \( t \). Accordingly, let \( q_l = \lim_{t \to \infty} \text{Prob}(k_l(t) > 0) \) denote stationary excitation probability of the \( l \)-th neuron. Due to the stochastic and distributed nature of the behaviours of the whole spiking neural system, it is difficult to obtain the value of \( q_l \). For a system with fixed configurations and inputs, a straightforward method is to estimate the value of \( q_l \) by using the Monte Carlo method. However, this method may not enable us to obtain a good estimation of \( q_l \) or be applicable when the number of neurons becomes very large.

In [Gel89], Gelenbe presented important results on the excitation probabilities of the neurons of this RNN system. It is proven in [Gel89, Gel90, Gel93] that \( q_l = \lim_{t \to \infty} \text{Prob}(k_l(t) > 0) \) can

Figure 2.1: Schematic representation of a spiking RNN system.
be directly calculated by the following system of equations:

\[ q_l = \min\left(\frac{\lambda_l^+}{r_l + \lambda_l^-}, 1\right), \] (2.1)

where \( \lambda_l^+ = \Lambda_l + \sum_{i=1}^{N} q_i w_{l,i}^+ \), \( \lambda_l^- = \lambda_l + \sum_{i=1}^{N} q_i w_{l,i}^- \), \( w_{l,i}^+ = r_i p_{l,i}^+ \), \( w_{l,i}^- = r_i p_{l,i}^- \) and \( l = 1, \ldots, L \). Here \( \Lambda_l \) and \( \lambda_l \) are respectively the arrival rates of external excitatory and inhibitory spikes and \( r_l \) is the firing rate of the \( l \)-th neuron. In addition, \( \sum_{l=1}^{L} (p_{l,i}^+ + p_{l,i}^-) + \nu_l = 1 \). Operation \( \min(a, b) \) produces the smaller one between \( a \) and \( b \). In [Gel93], it was shown that the system of \( N \) non-linear equations (2.1) have a solution which is unique. Therefore, the states of the RNN can be efficiently obtained by solving a system of equations without requiring the Monte Carlo method [MU49].

### 2.4.2 Learning in Random Neural Network

Solving supervised learning problems is a key issue in machine learning and artificial intelligence [AB09, Zhu05], as this is a requirement for the majority of practical applications of neural networks. In general, supervised learning of a system requires it to learn mapping from the input data (integers or real values) to the output data (labels or real values) in a given training dataset. Investigations have been carried out to facilitate the development of the RNN learning methods since 1993 [Gel93], with the goal of solving the RNN’s learning problems. In the work of [Gel93], Gelenbe developed a gradient-descent learning algorithm for the RNN, which seeks non-negative weights in terms of the gradient descent of a quadratic cost function and is the most widely-used learning algorithm for the RNN. Likas [LS00] utilises quasi-Newton methods and the derivative results of gradient descent in [Gel93] for the RNN learning. The proposed algorithm was tested on the RNN with two different architectures, where the fully recurrent architecture yielded poor results while the feed-forward architecture yielded smaller errors with fewer steps than the pure gradient-descent algorithm. It is also reported that the proposed algorithm has higher computational complexity and cannot be applied to online learning.

Based on [Gel93], Basterrech [BMRS09] exploited the Levenberg-Marquardt optimisation pro-
procedure and adaptive momentum. Its performance was evaluated when applied to several problems. Compared to the gradient-descent algorithm, it achieved faster convergence speed in almost all cases but proved less robust in solving the XOR problem. The RPROP learning algorithm introduced in [RB93], which performs a direct adaptation of the weight step based on the local gradient information, is combined with the gradient-descent algorithm [Gel93] for the RNN learning [BMRS09]. Georgiopoulos gave a brief description and some critical comments on this RPROP algorithm in [GLK11b]. It is reported in [BMRS09] that, in the learning of the geometrical images, this RPROP algorithm outperformed the gradient-descent algorithm. The gradient-descent-based learning algorithms may have some inherent weaknesses, such as slow convergence and being trapped into poor local minima. Instead of using the gradient descent, Georgiopoulos applied the adaptive inertia weight particle swarm optimization (AIW-PSO) [QYSW06] and differential evolution (DE) approach [SP97] for the RNN learning in [GLK11b]. Then, the performances of the gradient-descent algorithm, RPROP algorithm, AIW-PSO and DE approach for classifying 12 datasets were compared, and there is no clear overall winner among them. In addition, Timotheou tackled the learning problem of the RNN from a different perspective [Tim09, Tim08]. Specifically, the learning problem was first approximated to obtain a non-negative least-square (NNLS) problem. The next step is to apply the solution to the NNLS problem directly to the RNN learning problem. It is reported in [Tim09, Tim08] that this algorithm achieved better performance than the gradient-descent algorithm in a combinatorial optimisation problem emerging in disaster management.

2.4.3 Applications of Random Neural Network

The RNN has been successfully used in numerous applications [BK00, Tim10, GLK11a, RL11]. In [GS94], the RNN is applied in encoding an image into fewer bits in order to achieve image compression by exploiting a feed-forward architecture to learn the mapping from the image to the same image through a narrow passage with fewer hidden-layer neurons, where the extension to video compression applied similar principles [CGG98, CGB96]. In [BG98], multi RNNs are exploited to learn from training data and recognise shaped objects in clutter by combining the
results of all RNNs. In [GKW04], the feed-forward RNN is applied to learning a mapping from the scanning electron microscopy intensity waveform to the cross-section shape (i.e., the profile) such that the profile can be reconstructed from the intensity waveform and the destruction caused by acquiring a cross-section image can be avoided. In [OL07], the task of Denial of Service (DoS) detection is formulated as a pattern classification problem, and then, with useful input features measured and selected, the RNNs with both the feed-forward and recurrent architectures are exploited to fulfil the task.

Gelenbe proposes the multi-class RNN model in [GF99] as a generalisation following the work of the standard RNN [Gel89]. This multiple-class RNN can be described as a mathematical framework to represent networks that simultaneously process information of various types. Then, by setting the excitatory input rates and the desired outputs of the neuron states as the normalised pixel values, this multiple-class RNN is utilised to learn a given colour texture and generate a synthetic colour texture that imitates the original one [GH02]. The earlier work in [GHA00] that uses the standard RNN for black and white texture generation utilises similar principles. The RNN learning is also applied to image segmentation [LS06], vehicle classification [HM05] and texture classification and retrieval [TA06]. More applications related to utilising the learning capability of the RNN can be seen from the critical review in [GLK11b]. The many other areas in which the RNN has already been applied are detailed in the reviews in [BK00, Tim10].

2.4.4 Implementation of Random Neural Network

There are three main approaches to implementing the RNN [AGEK00]. The most accessible approach is to use computer software tools based on CPU to simulate the RNN spiking behaviours or conduct the RNN equations computations. However, this approach can be of low efficiency because it does not utilise the fact that the RNN can be computed or simulated in a parallel manner. The second approach is still to implement the RNN in computers but in a parallel manner. Specifically, the computations related to the RNN can be conducted simultaneously by a very large number of simple processing elements, where the rapid development of
chip multiprocessor (CMP) and Graphics Processing Unit (GPU) make it more feasible. The third approach is to implement the RNN model in hardware using special-purpose digital or analogue components [AGEK00]. The RNN’s high parallelism can be exploited fully via specially designed hardware, and significant speed increases can be achieved when applying the RNN.

For the first approach, Abdelbaki’s work [Abd99] implemented the RNN model using MATLAB software. To the best of the authors’ knowledge, no investigation has hitherto been undertaken to implement the RNN using the second approach. In terms of hardware, another of Abdelbaki’s studies in [AGEK00] shows that the RNN can be implemented by a simple continuous analogue circuit that contains only the addition and multiplication operations. In addition, the Cerkez’s study [CAH97] proposed a digital realisation for the RNN using discrete logic integrated circuits. Kocak’s work [KST03] implemented the RNN-based routing engine of the cognitive packet networks (CPN) in hardware. This innovation significantly decreased the number of weight terms (from $2n^2$ to $2n$) needing to be stored for each RNN in the engine, where the CPN is an alternative to the IP-based network architectures proposed by Gelenbe [GLX01].
Chapter 3

A Classifier based on Random Neural Network Function Approximator

3.1 Introduction

Neural networks are capable of solving complex learning problems, such as pattern classification, mainly owing to their remarkable approximation capabilities [LBH15, TDH16]. In the case of the random neural network (i.e., the RNN), the work by Gelenbe in [GML99a, GML99b, GML04] focused on investigating the approximation property of the RNN. It is proved that the RNN can be constructed as a function approximator with the universal approximation property (UAP).

In this chapter, we pursue the work of the RNN function approximator [GML99a, GML99b, GML04] and explore its theoretical framework. This is to lay a theoretical basis for the learning capability from data of the RNN and the investigation into the capability of the RNN for deep learning in the following chapters. First, we summarize the theoretical results of the RNN function approximator into five lemmas and a theorem. Then, we present a procedure to configure the parameters in this RNN function approximator such that it learns a given dataset efficiently. We also show that the RNN function approximator has lower computational complexity than the orthogonal-polynomial function approximator in [ZYG+14a] and the one-
hidden-layer MLP, under certain assumptions. The RNN function approximator, equipped with
the proposed configuration/learning procedure, is then applied as a tool for solving pattern
classification problems. Numerical experiments on various datasets demonstrate that the RNN
classifier is the most efficient among the six different types of classifiers.

*Part of this chapter is in preparation as: Yonghua Yin and Erol Gelenbe. “A Classifier Based
on Spiking Random Neural Network Function Approximator.” Preprint available in Reserach-
Gate.net.*

### 3.2 RNN Function Approximator

The work in [GML99a, GML99b, GML04] proposed a function approximator based on the
RNN. This paper summarizes the theoretical framework of the RNN function approximator
into the following five lemmas and a theorem. Specifically, Subsection 3.2.1 presents Lemmas
3.1 to 3.5. Lemmas 3.1 and 3.2 respectively come from Lemmas 4 and 8 in [GML04]. Lemmas
3.3 to 3.5 are developed in this work. Subsection 3.1 presents Theorem 3.1 on the construction
and approximation capability of the RNN function approximator, which comes from Theorem
10 in [GML04]. The proof of this theorem is completed by using Lemmas 3.1 to 3.5.

#### 3.2.1 Lemmas

**Lemma 3.1** Consider a term of the form $\frac{x}{(1 + x)^i}$ for $x \in [0, 1]$ with $i \geq 1$, there exists a
spiking RNN with $i + 1$ neurons such that

$$q_{i+1} = \frac{x}{(1 + x)^i}. \quad (3.1)$$

**Proof.** This lemma comes from Lemma 4 in [GML04].

**Lemma 3.2** Consider a term of the form $\prod_{n=1}^{\hat{N}}\frac{1}{(1 + x_n)^{i_n}}$ with $\hat{N} \geq 2$ and $i_n \geq 1$, there
exists a spiking RNN with \( L = \sum_{n=1}^{\hat{N}} i_n + \hat{N} \) neurons such that

\[
q_L = \prod_{n=1}^{\hat{N}} \frac{1}{(1 + x_n)^{i_n}}.
\] (3.2)

Proof. This lemma comes from Lemma 8 in [GML04]. \[\square\]

Lemma 3.3 Consider a term of the form \(-\prod_{n=1}^{\hat{N}} 1/(1 + x_n)^{i_n}\) with \(1 \leq i_1 \leq i_2 \leq \cdots \leq i_{\hat{N}}\), the following polynomial holds:

\[
-\prod_{n=1}^{\hat{N}} \frac{1}{(1 + x_n)^{i_n}} = \left( \sum_{j_1=1}^{i_1} \frac{x_1}{(1 + x_1)^{j_1}} \right) \frac{1}{(1 + x_2)^{i_2}} \cdots \frac{1}{(1 + x_{\hat{N}})^{i_{\hat{N}}}} + \cdots + \left( \sum_{j_{\hat{N}-1}=1}^{i_{\hat{N}-1}} \frac{x_{\hat{N}-1}}{(1 + x_{\hat{N}-1})^{j_{\hat{N}-1}}} \right) \frac{1}{(1 + x_{\hat{N}})^{i_{\hat{N}}}} + \sum_{j_{\hat{N}}=1}^{i_{\hat{N}}} \frac{x_{\hat{N}}}{(1 + x_{\hat{N}})^{j_{\hat{N}}}} - 1
\] (3.3)

Lemma 3.4 Consider a term of the form \(x_1/(1 + x_1)^{i_1} \prod_{n=2}^{\hat{N}} 1/(1 + x_n)^{i_n}\) with \(\hat{N} \geq 2\) and \(i_n \geq 1\), there exists a spiking RNN with \( L = \sum_{n=1}^{\hat{N}} i_n + \hat{N} \) neurons such that

\[
q_L = \frac{x_1}{(1 + x_1)^{i_1}} \prod_{n=2}^{\hat{N}} \frac{1}{(1 + x_n)^{i_n}}.
\] (3.4)

Lemma 3.5 Consider \(\hat{N}\) RNNs where the \(L_n\)th neuron of the \(n\)th spiking RNN is an output neuron with the departure probability \(\nu_{L_n,n} = 1\) and the state denoted by \(q_{L_n,n}\) (\(n = 1, 2, \cdots, \hat{N}\)), they can be connected as a whole spiking RNN with \( L = \sum_{n=1}^{\hat{N}} L_n + 1 \) neurons such that

\[
q_L = \frac{\sum_{n=1}^{\hat{N}} w_n q_{L_n,n}}{1 + \sum_{n=1}^{\hat{N}} w_n q_{L_n,n}},
\] (3.5)

where \(w_n > 0\) is a given weight for \(q_{L_n,n}\).

The proof of Lemmas 3.3 to 3.5 are given in Appendix A.
3.2.2 Theorem

To lay a basis for further discussion, let us define the following product-based function based on the system of equations (2.1) and the work in \[GML99a, GML99b, GML04\]:

\[
\varphi(X) = \sum_{i_1=0}^{I_1} \cdots \sum_{i_N=0}^{I_N} w_{i_1,\cdots,i_N} \left( \prod_{n=1}^{N} \frac{1}{(1 + x_n)^{i_n}} \right),
\]

(3.6)

where \(X = [x_1 \ x_2 \ \cdots \ x_N] \in [0,1]^{1 \times N}\), \([i_n]\) are non-negative integers and \([w_{i_1,\cdots,i_N}]\) are real values. Then, based on Lemmas 3.1 to 3.5 in Subsection 3.2.1 and Theorem 10 in \[GML04\], we have the following theorem of the RNN function approximator on its construction and approximation capability, with the complete proof given.

**Theorem 3.1** For a continuous real-valued function \(f(X) : [0,1]^{1 \times N} \to \mathbb{R}\) and any \(\epsilon > 0\), there exists a constant \(w_{\text{con}}\) and an RNN with \(L-1\) neurons and a single output neuron (the \(L\)-th neuron), resulting in a function \(\varphi(X) = q_L/(1 - q_L) + w_{\text{con}}\) which approximates \(f(X)\) uniformly on \([0,1]^{1 \times N}\) with error \(\sup_{X \in [0,1]^{1 \times N}} |f(X) - \varphi(X)| < \epsilon\).

**Proof.** Based on the Weierstrass theorem \[PPP^{+}81\] and the work in \[GML99a\], for any given continuous function \(f(X) : [0,1]^{1 \times N} \to \mathbb{R}\) and \(\epsilon > 0\), there exists a function \(\varphi(X)\) such that \(\sup_{X \in [0,1]^{1 \times N}} |f(X) - \varphi(X)| < \epsilon\). The following demonstrates the procedure to construct an RNN system that produces a given \(\varphi(X)\).

Let us handle the terms \(w_{i_1,\cdots,i_N} h(X,i_1,\cdots,i_N) = w_{i_1,\cdots,i_N} \left( \prod_{n=1}^{N} 1/(1 + x_n)^{i_n} \right)\) in any given \(\varphi(X)\), i.e., (3.6), case by case. First, if \(i_1 = \cdots = i_N = 0\), then \(w_{i_1,\cdots,i_N} h(X,i_1,\cdots,i_N) = w_{i_1,\cdots,i_N}\), we simply add \(w_{i_1,\cdots,i_N}\) into the constant \(w_{\text{con}}\) rather than taking it into the construction of the RNN. Note that \(w_{\text{con}}\) is initially set as zero.

For discussion convenience, a symbol \(\varphi^+\) is given and also initially set as zero. For the rest terms where either one of \(i_1,\cdots,i_N\) is not zero, there are three cases.

- If \(w_{i_1,\cdots,i_N} = 0\), there is no need to consider this term.
Chapter 3. A Classifier based on Random Neural Network Function Approximator

Figure 3.1: Schematic representation of the RNN function approximator.

- If \( w_{i_1, \ldots, i_N} > 0 \), an RNN with \( \hat{L} \) neurons is constructed according to Lemma 3.2 such that \( q_L = h(X, i_1, \ldots, i_N) \). We add \( w_{i_1, \ldots, i_N} q_L \) into \( \varphi^+ \).

- If \( w_{i_1, \ldots, i_N} < 0 \), this term can be transformed as \( w_{i_1, \ldots, i_N} h(X, i_1, \ldots, i_N) = |w_{i_1, \ldots, i_N}| \xi - |w_{i_1, \ldots, i_N}| \) according to Lemma 3.3 where \( \xi = \sum_{k=1}^{\hat{K}} \xi_k \) and \( \xi_k \) is either in the similar form of \( x_1/(1 + x_1)^{i_1} \prod_{n=2}^{N} 1/(1 + x_n)^{i_n} \) or \( x/(1 + x)^{i} \). For each term \( \xi_k \), an RNN with \( \hat{L} \) neurons is constructed according to Lemma 3.1 or 3.4 such that \( q_{\hat{L}} = \xi_k \). In addition, we add \( |w_{i_1, \ldots, i_N}| q_{\hat{L}} \) into \( \varphi^+ \) and add \(-|w_{i_1, \ldots, i_N}|\) into the constant \( w_{\text{con}} \).

Until now, a certain number of RNNs have been constructed and the constant \( w_{\text{con}} \) have been obtained. In addition, since all terms in \( \varphi(X) \) have been considered, we evidently have \( \varphi(X) = \varphi^+ + w_{\text{con}} \). According to Lemma 3.5, a new RNN with \( L \) neurons can be constructed by connecting these RNNs together such that

\[
q_L = \frac{\varphi^+}{1 + \varphi^+} = \frac{\varphi(X) - w_{\text{con}}}{1 + \varphi(X) - w_{\text{con}}}. \tag{3.7}
\]

In other form,

\[
\varphi(X) = \frac{q_L}{1 - q_L} + w_{\text{con}}. \tag{3.8}
\]

The proof is thus completed.

\[\blacksquare\]
3.3 Learning in RNN Function Approximator

This section develops the learning procedure for the RNN function approximator to find appropriate values of parameters \( \{i_1, \ldots, i_N, w_{i_1}, \ldots, w_{i_N}|i_1 = 1, \ldots, I_1; \ldots; i_N = 1, \ldots, I_N \} \) in equation (3.6) such that the approximator learns a given dataset. Then, the computational complexity in using the approximator is also analyzed and compared with other neural networks, such as the MLP and OPN in [ZYG+14a].

3.3.1 Learning Problem Description

Suppose there is a dataset \( \{(X_d, y_d) | d = 1, 2, \ldots, D \} \), where the input vector of a sample is \( X_d = [x_{d,1}, x_{d,2}, \ldots, x_{d,N}] \in [0, 1]^{1 \times N} \) and \( y_d \) is the desired output or the groundtrue label. According to Theorem 3.1 and equations (3.6) and (3.8), given parameter pairs \( \{[i_1, \ldots, i_N], w_{i_1}, \ldots, w_{i_N}|i_1 = 1, \ldots, I_1; \ldots; i_N = 1, \ldots, I_N \} \), an RNN function approximator can be constructed following the procedure in the proof of Theorem 3.1 as illustrated in Figure 3.1. When \( X_d \) is imported into the RNN function approximator, the stationary excitation probability of the output neuron in this RNN, denoted by \( q_L(X_d) \), satisfies the equation

\[
\varphi(X_d) = \frac{q_L(X_d)}{1 - q_L(X_d)} + w_{\text{con}} = \sum_{i_1=0}^{I_1} \cdots \sum_{i_N=0}^{I_N} w_{i_1, \ldots, i_N} \left( \prod_{n=1}^{N} \frac{1}{(1 + x_{d,n})^{i_n}} \right). \tag{3.9}
\]

Then, the problem of the RNN function approximator to learn the given dataset \( \{(X_d, y_d) | d = 1, 2, \ldots, D \} \) is to find appropriate values of \( [I_1, \ldots, I_N] \) and \( \{[i_1, \ldots, i_N], w_{i_1}, \ldots, w_{i_N}|i_1 = 1, \ldots, I_1; \ldots; i_N = 1, \ldots, I_N \} \) such that, given any input \( X_d \), the output of the approximator described by \( \varphi(X_d) \) approaches the desired output \( y_d \) as close as possible.

Problem Transformation

Let us assume that the sequence vector \( [i_1, \ldots, i_N] \) in equation (3.9) is arranged by a pre-set order method, e.g., the lexicographic order and graded lexicographic order [DX14]. For better
illustration, let us switch the notation of \([i_1 \cdots i_N]\) into \([i_{1,k} \cdots i_{N,k}]\) by adding one more subscript \(k\); correspondingly, let us switch the notation of \(w_{i_1 \cdots i_N}\) into \(w_{i_{1,k} \cdots i_{N,k}}\). Let us define the \(k\)th sequence vector \([i_{1,k} \cdots i_{N,k}]\) as \(C_k = [i_{1,k} \cdots i_{N,k}]\) and define the corresponding \(k\)th weight \(w_{i_{1,k} \cdots i_{N,k}}\) as \(w_k\), where \(k = 1, \cdots, K\).

Let us define a group of RNN basis functions (RNNFs) \(\{h_k(X, C_k) \mid k = 1, 2, \cdots, K\}\) with \(h_k(X, C_k) = \prod_{n=1}^{N} 1/(1 + x_n)^{i_{n,k}}\). In addition, let \(W = [w_1 \cdots w_K]^T\),

\[
C = [C_1^T \cdots C_K^T] = \begin{bmatrix}
i_{1,1} & i_{1,2} & \cdots & i_{1,K} \\
i_{2,1} & i_{2,2} & \cdots & i_{2,K} \\
\vdots & \vdots & \ddots & \vdots \\
i_{N,1} & i_{N,2} & \cdots & i_{N,K}
\end{bmatrix} \in \mathbb{Z}_{\geq 0}^{N \times K}
\]

and \(H(X, C) = [h_1(X, C_1) \cdots h_K(X, C_K)]^T\). In the matrix \(C\), the number of rows \(N\) is the dimension of the input data, the number of the columns \(K\) is the number of the RNNFs used in the RNN function approximator and the integer \(i_{i,k}\) is the order of \(1/(1 + x_i)\) with respect to the \(i\)th input in the \(k\)th RNNFs. Through these denotations, equation (3.6) can be rewritten in the following form:

\[
\varphi(X, C, W) = \sum_{k=1}^{K} w_k \left( \prod_{n=1}^{N} \frac{1}{(1 + x_n)^{i_{n,k}}} \right) = \sum_{k=1}^{K} w_k h_k(X, C_k) = H(X, C)W. \quad (3.10)
\]

Given \(C\) and \(W\), an RNN function approximator can be constructed following the procedure in the proof of Theorem 3.1. When \(X_d\) is imported into the RNN function approximator, the stationary excitation probability of the output neuron in this RNN, denoted by \(q_L(X_d, C, W)\), satisfies the equation

\[
\varphi(X_d, C, W) = \frac{q_L(X_d, C, W)}{1 - q_L(X_d, C, W)} + w_{\text{con}} = \sum_{k=1}^{K} w_k h_k(X_d, C_k) = H(X_d, C)W, \quad (3.11)
\]

which is another form of equation (3.9). Then, the problem of the RNN function approximator to learn the given dataset \(\{(X_d, y_d) \mid d = 1, 2, \cdots, D\}\) is to find appropriate values of \(C\) and \(W\) such that, given any input \(X_d\), the output of the approximator described by \(\varphi(X_d, C, W)\)
approaches the desired output \(y_d\) as close as possible. Mathematically, we want to solve the following optimization problem:

\[
\min_{C,W} \sum_{d=1}^{D} (\varphi(X_d, C, W) - y_d)^2.
\] (3.12)

### 3.3.2 Learning Procedure Design

The procedure of finding appropriate values of \(C\) and \(W\) for the RNN function approximator to learn a given dataset \(\{(X_d, y_d) \mid d = 1, 2, \cdots, D\}\) is developed in this subsection. Specifically, the learning procedure is constituted by two main steps:

- the values of \(C\) are determined utilizing the ordering method; then
- the values of \(W\) are determined by solving a convex optimization problem.

#### First Step: Determining \(C\) Independent of Data

The relationship between \(k\) and \(C_k\) could be determined by different ordering methods [DXI14]. In this work, we choose the graded lexicographic order (GLO) as inspired by [ZYG+14a]. The definition of the GLO is given as the following.

**Definition of graded lexicographic order.** Suppose there are two different sequence vectors \(C_k = [i_{1,k} \ i_{2,k} \ \cdots \ i_{N,k}]\) and \(C_{\hat{k}} = [\hat{i}_{1,k} \ \hat{i}_{2,k} \ \cdots \ \hat{i}_{N,\hat{k}}]\) ordered by the GLO, where positive integers \(k\) and \(\hat{k}\) satisfy \(k \neq \hat{k}\). In addition, let \(|C_k| = i_{1,k} + i_{2,k} + \cdots + i_{N,k}\) and \(|C_{\hat{k}}| = \hat{i}_{1,k} + \hat{i}_{2,k} + \cdots + \hat{i}_{N,\hat{k}}\). Then, we know that \(k > \hat{k}\) if either of the following two conditions is satisfied:

- condition i: \(|C_k| > |C_{\hat{k}}|\);
- condition ii: \(|C_k| = |C_{\hat{k}}|\) and the first nonzero element of \(C_k - C_{\hat{k}} = [i_{1,k} - \hat{i}_{1,k} \ i_{2,k} - \hat{i}_{2,k} \ \cdots \ i_{N,k} - \hat{i}_{N,\hat{k}}]\) is positive.

Given any non-negative integer number \(K\) and the number of input dimensions \(N\), the matrices \(C\) can then be determined accordingly by the GLO. For example, the matrixes of \(C\) with
\( K = 7, N = 2 \) and \( K = 7, N = 3 \) obtained by the GLO are given as the following:

\[
\begin{bmatrix}
0 & 0 \\
0 & 1 \\
1 & 0 \\
0 & 2 \\
1 & 1 \\
2 & 0 \\
0 & 3 \\
\end{bmatrix} \quad \text{when } K = 7, N = 2; \\
\begin{bmatrix}
1 & 0 \\
0 & 2 \\
0 & 1 \\
1 & 0 \\
0 & 3 \\
\end{bmatrix} \quad \text{when } K = 7, N = 3.
\]

**Second Step: Determining \( W \) from Data**

The problem of determining \( W \) is first formulated into an optimization problem and then solved via the pseudoinverse method \([ZYG^{14a}, ZYG^{14b}, HZS06]\).

Given the dataset \( \{(X_d, y_d) \mid d = 1, 2, \ldots, D\} \), the input matrix \( X \) and desired-output vector \( Y \) are expressed as

\[
X = \begin{bmatrix}
X_1 \\
X_2 \\
\vdots \\
X_D \\
\end{bmatrix} = \begin{bmatrix}
x_{1,1} & x_{1,2} & \cdots & x_{1,N} \\
x_{2,1} & x_{2,2} & \cdots & x_{2,N} \\
\vdots & \vdots & \ddots & \vdots \\
x_{D,1} & x_{D,2} & \cdots & x_{D,N} \\
\end{bmatrix} \in [0, 1]^{D \times N},
\]

and \( Y = [y_1 \cdots y_D]^T \in \mathbb{R}^{D \times 1} \). Let us defined an input-activation matrix \( H \) as

\[
H = [H_1(X_1, C) \ H_2(X_2, C) \ \cdots \ H_D(X_D, C)]^T \\
= \begin{bmatrix}
h_1(X_1, C_1) & h_2(X_1, C_2) & \cdots & h_K(X_1, C_K) \\
h_1(X_2, C_1) & h_2(X_2, C_2) & \cdots & h_K(X_2, C_K) \\
\vdots & \vdots & \ddots & \vdots \\
h_1(X_D, C_1) & h_2(X_D, C_2) & \cdots & h_K(X_D, C_K) \\
\end{bmatrix} \in [0, 1]^{D \times K}.
\]
Based on (3.10), an output matrix can be obtained as

$$\Phi(X, C, W) = HW = [\varphi(X_1, C, W) \cdots \varphi(X_D, C, W)]^T.$$ 

Then, based on the optimization problem (3.12), the problem of determine $W$ can be formulated into the following optimization problem:

$$\min_W ||\Phi(X, C, W) - Y||^2.$$ (3.13)

The Moore-Penrose pseudoinverse can be used to obtain solutions to this optimization problem directly [ZYG+14a, ZYG+14b, HZS06]. Then, the values of $W$ can be determined as

$$W = \text{pinv}(H)Y,$$ (3.14)

where $\text{pinv}(H)$ denotes the Moore-Penrose pseudoinverse of $H$ and can be evaluated via the MATLAB routine “pinv”.

Discussions

The classic gradient-descent algorithm has been widely used in training neural networks. However, there are inherent disadvantages, such as slow convergence and the possibilities of being trapped into poor local minimums, since the training is generally conducted by solving a non-convex optimization problem. As described in the proposed learning procedure of the RNN function approximator, the first step fixes the arrangement of the matrix $C$ by the deterministic GLO, which allows the formulation of the learning problem into a convex optimization problem (i.e., the problem (3.13)) that converges much faster than the gradient-descent neural networks. In addition, the solution of this convex problem for the parameter $W$ can be obtained directly via the least-square methods [ZYG+14a, ZYG+14b, HZS06], which avoids the inherent disadvantages of gradient-descent neural networks.
Table 3.1: Minimum numbers of operations required to compute the RNN function approximator, SOCPNN and MLP with $N$ inputs, $K$ hidden units and one output once

<table>
<thead>
<tr>
<th>Operation</th>
<th>RNN</th>
<th>SOCPNN</th>
<th>MLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tansig</td>
<td>0</td>
<td>0</td>
<td>$K$</td>
</tr>
<tr>
<td>Addition</td>
<td>$\zeta_R = N + K - 1$</td>
<td>$\zeta_S = (I^{(\text{avg})} - 1)N + K - 1$</td>
<td>$\zeta_M = NK + K - 1$</td>
</tr>
<tr>
<td>Multiplication</td>
<td>$\chi_R = (I^{(\text{avg})} + K - 1)N$</td>
<td>$\chi_S = (2I^{(\text{avg})} + K - 2)N$</td>
<td>$\chi_M = NK + K$</td>
</tr>
<tr>
<td>Division</td>
<td>$N$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

3.3.3 Computational-Complexity Analysis and Comparison

This subsection analyses the computational complexity of $\varphi(X, C, W) = \sum_{k=1}^{K} w_k h_k(X, C_k)$ in the RNN function approximator with a given $K$ and compare it with the single-output Chebyshev-polynomial neural network (SOCPNN) in [ZYG+14a] and single-hidden-layer MLP.

**Computational-Complexity Analysis**

The computations of $\varphi(X, C, W) = \sum_{k=1}^{K} w_k h_k(X, C_k)$ include the calculation to obtain $\{h_k(X, C_k) \mid k = 1, 2, \cdots, K\}$ and the weighted summation of them. The relationship between $k$ and $C_k = [i_{1,k} \ i_{2,k} \cdots \ i_{N,k}]$ is determined by the GLO. Let us assume that $C_K = [I_1 \ I_2 \cdots \ I_N]$, $I_1, I_2, \cdots, I_N \geq 2$ and $I^{(\text{avg})} = (I_1 + I_2 + \cdots + I_N)/N$. First, the minimum numbers of addition, multiplication and division operations required to construct $\{1/(1+x_n)^{I_n} = (1/(1+x_n))^{I_n} \mid n = 1, 2, \cdots, N\}$ are respectively $N$, $(I_1 + I_2 + \cdots + I_N - N) = (I^{(\text{avg})} - 1)N$ and $N$. Then, at least $(N - 1)K$ more multiplication operations are required to construct $\{h_k(X) \mid k = 1, 2, \cdots, K\}$. The weighted summation in $\sum_{k=1}^{K} w_k h_k(X, C_k)$ requires $(K - 1)$ addition and $K$ multiplication operations. Finally, the minimum numbers of addition, multiplication and division operations required to compute $\varphi(X, C, W)$ once are respectively $N + K - 1$, $(I^{(\text{avg})} + K - 1)N$ and $N$.

**Computational-Complexity Comparison**

The computational complexity of the RNN function approximator is compared with those of the SOCPNN in [ZYG+14a] and the single-hidden-layer MLP. Without loss of generality,
we assumed that the hidden-layer units in the MLP are activated by tan-sigmoidal functions 
\[ \text{tansig}(x) = \frac{2}{1 + \exp(-2x)} - 1 \]. The unit numbers in input and hidden layers are respectively \( N \) and \( K \) for the SOCPNN and MLP. Based on the above analysis in Subsection 3.3.3 and \[ ZYG^{+14a} \], the operation numbers required to compute the RNN function approximator, SOCPNN and MLP once are listed in Table 3.1. Let us define a converting factor \( \beta > 0 \): the computation of a division operation is equivalent to that of \( \beta \) multiplication operations.

**Theorem 3.2** Consider the RNN function approximator, SOCPNN and MLP given in Table 3.1, the RNN function approximator is of the lowest computational complexity among these three methods, provided that \( I_1, I_2, \cdots, I_N \geq 2 \) and \( \beta + 1 < I^{(\text{avg})} \).

**Proof.** It has been proven in \[ ZYG^{+14a} \] that \( \zeta_S = (I^{(\text{avg})} - 1)N + K - 1 < \zeta_M = KN + K - 1 \) and \( \chi_S = (2I^{(\text{avg})} + K - 2)N < \chi_M = NK + K \) when \( N \geq 2 \). Let us first compare the numbers of addition operations in Table 3.1. Since it is assumed that \( I_1, I_2, \cdots, I_N \geq 2 \), then \( I^{(\text{avg})} \geq 2 \). Subsequently, the following can be obtained: \( \zeta_R - \zeta_S = (1 - I^{(\text{avg})})N < 0 \). Then, \( \zeta_R < \zeta_S < \zeta_M \), meaning the computation of the RNN function approximator requires the least addition operations. Then, we compare the numbers of multiplication and division operations in Table 3.1. With converting factor \( \beta \), the computation of \( N \) division operations is equivalent to that of \( \beta N \) multiplication operations, where normally \( \beta \geq 1 \) and its exact value depends on many factors, such as the computation devices and methods used. Subsequently, \( \chi_R + \beta N - \chi_S = (\beta - I^{(\text{avg})} + 1)N \). Then, \( \chi_R + \beta N < \chi_S \) if \( \beta + 1 < I^{(\text{avg})} \). Therefore, \( \chi_R + \beta N < \chi_S < \chi_M \), with a condition \( \beta + 1 < I^{(\text{avg})} \) that is not difficult to satisfy when \( K \) is not small, meaning that the RNN function approximator requires the least operations in multiplications and divisions in this situation. Both the RNN function approximator and SOCPNN do not require computations of tansig functions. We can conclude that the RNN function approximator is of the lowest computational complexity among these three methods under the conditions of \( I_1, I_2, \cdots, I_N \geq 2 \) and \( \beta + 1 < I^{(\text{avg})} \). \( \blacksquare \)
Table 3.2: Attribute, instance and class numbers of UCI real-world datasets from different areas

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Attribute No.</th>
<th>Instance No.</th>
<th>Class No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris</td>
<td>4</td>
<td>150</td>
<td>3</td>
</tr>
<tr>
<td>Teaching Assistant</td>
<td>5</td>
<td>151</td>
<td>2</td>
</tr>
<tr>
<td>Evaluation (TAE)</td>
<td>7</td>
<td>210</td>
<td>3</td>
</tr>
<tr>
<td>Liver Disorders (LD)</td>
<td>5</td>
<td>345</td>
<td>2</td>
</tr>
<tr>
<td>Seeds</td>
<td>8</td>
<td>768</td>
<td>2</td>
</tr>
<tr>
<td>Pima Indians Diabetes (PID)</td>
<td>8</td>
<td>1484</td>
<td>10</td>
</tr>
<tr>
<td>Yeast [HN90]</td>
<td>9</td>
<td>699</td>
<td>2</td>
</tr>
<tr>
<td>Breast Cancer Wisconsin (BC) [MSW90]</td>
<td>9</td>
<td>214</td>
<td>7</td>
</tr>
<tr>
<td>Glass</td>
<td>13</td>
<td>178</td>
<td>3</td>
</tr>
<tr>
<td>Wine</td>
<td>16</td>
<td>100</td>
<td>7</td>
</tr>
<tr>
<td>Parkinsons [LMR + 07]</td>
<td>22</td>
<td>195</td>
<td>2</td>
</tr>
<tr>
<td>Wall-Following</td>
<td>24</td>
<td>5456</td>
<td>4</td>
</tr>
<tr>
<td>Robot Navigation (WFRN) [FBVV09]</td>
<td>34</td>
<td>351</td>
<td>2</td>
</tr>
<tr>
<td>Ionosphere [SWHB89]</td>
<td>35</td>
<td>186</td>
<td>19</td>
</tr>
<tr>
<td>Soybean Large (SL)</td>
<td>51</td>
<td>6118</td>
<td>6</td>
</tr>
<tr>
<td>First-Order</td>
<td>60</td>
<td>208</td>
<td>2</td>
</tr>
<tr>
<td>Theorem Proving (FOTP) [BHP14]</td>
<td>279</td>
<td>452</td>
<td>16</td>
</tr>
</tbody>
</table>

3.4 Application to Pattern Classification

This section presents the application of the RNN function approximator to pattern classification. Experimental results based on various real-world classification datasets demonstrate that the resultant RNN classifier is a comparable or more efficient classifier than both state-of-the-art and conventional classifiers.

3.4.1 Experiment Settings

Brief descriptions on various real-world classification datasets and the settings of these datasets used for the experiments are presented. The settings of the classification methods exploited for the experiments are then presented.
3.4. Application to Pattern Classification

Numerical experiments in this work use real-world classification datasets from different areas, which are the Iris, Teaching Assistant, Evaluation (TAE), Liver Disorders (LD), Seeds, Pima Indians Diabetes (PID), Yeast, Breast Cancer Wisconsin (BC), Wine, Zoo, Parkinsons, Wall-Following Robot Navigation (WFRN), First-Order Theorem Proving (FOTP), Ionosphere, Soybean Large (SL), and Cardiac Arrhythmia (CA). The numbers of input attributes, classes and instances for each dataset are given in Table 3.2. For a better understanding of these datasets and the related classification problems, each dataset is given a brief description in Appendix D.1.2.

Table 3.3: Comparison of testing accuracy rates of RNN, SOCPNN, MOCPNN, ELM and MLP as well as regularized RBFNN and SVM for pattern classification

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Testing Accuracy Rate (%)</th>
<th>Testing-Accuracy Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris</td>
<td>94.67/6 94.67/6 92.00/2 93.64/3 94.15/4 92.00/2</td>
<td><strong>96.00</strong>/7</td>
</tr>
<tr>
<td>TAE</td>
<td>78.57/3 86.67/7 86.67/7 80.98/4 68.39/1 74.67/2</td>
<td>82.67/5</td>
</tr>
<tr>
<td>LD</td>
<td><strong>79.07</strong>/7 76.74/6 76.74/6 74.15/3 64.13/2 55.81/1</td>
<td>76.16/4</td>
</tr>
<tr>
<td>Seeds</td>
<td>88.57/4 77.14/1 89.52/6 85.60/2 <strong>91.83</strong>/7 85.71/3</td>
<td>89.52/6</td>
</tr>
<tr>
<td>PID</td>
<td>74.22/5 74.48/7 74.48/7 63.17/2 67.34/4 60.94/1</td>
<td>65.10/3</td>
</tr>
<tr>
<td>Yeast</td>
<td>59.00/7 70.73/2 58.05/5 58.92/6 56.06/4 52.37/3</td>
<td>39.38/1</td>
</tr>
<tr>
<td>BC</td>
<td><strong>97.42</strong>/7 97.42/7 97.42/7 89.20/1 97.05/4 96.28/3</td>
<td>96.28/3</td>
</tr>
<tr>
<td>Glass</td>
<td>55.24/4 42.86/1 54.29/3 45.08/2 64.07/6 <strong>66.67</strong>/7</td>
<td>60.00/5</td>
</tr>
<tr>
<td>Wine</td>
<td><strong>97.73</strong>/7 94.32/4 <strong>97.73</strong>/7 53.02/2 96.69/5 93.18/3</td>
<td>44.32/1</td>
</tr>
<tr>
<td>Zoo</td>
<td>93.75/5 93.75/5 93.75/5 86.62/1 94.88/6 <strong>95.83</strong>/7</td>
<td>91.67/2</td>
</tr>
<tr>
<td>Parkinsons</td>
<td><strong>86.60</strong>/7 78.35/5 78.35/5 75.40/1 85.15/6 78.35/5</td>
<td>76.29/2</td>
</tr>
<tr>
<td>WFRN</td>
<td>76.97/4 46.57/1 76.38/3 64.81/2 <strong>92.77</strong>/7 85.92/5</td>
<td>88.16/6</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>87.43/6 85.71/4 85.71/4 83.09/1 86.91/5 85.14/2</td>
<td><strong>91.43</strong>/7</td>
</tr>
<tr>
<td>SL</td>
<td>82.02/5 26.97/1 69.66/3 65.44/2 82.15/6 <strong>91.01</strong>/7</td>
<td>79.78/4</td>
</tr>
<tr>
<td>FOTP</td>
<td>47.38/4 13.11/1 46.44/3 48.28/5 49.30/6 44.28/2</td>
<td><strong>52.29</strong>/7</td>
</tr>
<tr>
<td>Sonar</td>
<td>71.84/3 74.76/5 74.76/5 69.85/2 79.43/6 <strong>80.58</strong>/7</td>
<td>56.31/1</td>
</tr>
<tr>
<td>CA</td>
<td>62.80/5 10.14/1 62.32/4 60.60/3 66.57/6 <strong>66.67</strong>/7</td>
<td>58.94/2</td>
</tr>
</tbody>
</table>

| Avg. Rank     | 5.24 3.76 4.82 2.47 5.00 3.94 3.88 |

Dataset Description and Settings

In the experiments, the output pattern of these classification datasets are designed as the following. Let us denote a classification dataset as $S = \{(X_d, Y_d) \mid d = 1, 2, \ldots, D\}$, where $X_d = [x_{d,n}]^{1 \times N}$, $Y_d = [y_{d,m}]^{1 \times M}$. In addition, $N$, $M$ and $D$ are respectively the numbers
Table 3.4: Comparison of testing time of RNN, SOCPNN, MOCPNN, ELM and MLP as well as regularized RBFNN and SVM for pattern classification

<table>
<thead>
<tr>
<th>Dataset</th>
<th>RNN</th>
<th>SOCPNN</th>
<th>MOCPNN</th>
<th>ELM</th>
<th>MLP-LM</th>
<th>RBFNN</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris</td>
<td>0.02/6</td>
<td>0.01/7</td>
<td>0.02/5</td>
<td>0.23/3</td>
<td>0.74/2</td>
<td>0.93/1</td>
<td>0.05/4</td>
</tr>
<tr>
<td>TAE</td>
<td>0.01/7</td>
<td>0.01/6</td>
<td>0.01/5</td>
<td>0.34/3</td>
<td>0.51/1</td>
<td>0.46/2</td>
<td>0.02/4</td>
</tr>
<tr>
<td>LD</td>
<td>0.02/5</td>
<td>0.01/7</td>
<td>0.01/6</td>
<td>0.62/2</td>
<td>0.53/3</td>
<td>2.53/1</td>
<td>0.09/4</td>
</tr>
<tr>
<td>Seeds</td>
<td>0.02/6</td>
<td>0.02/5</td>
<td>0.02/4</td>
<td>0.30/3</td>
<td>0.55/2</td>
<td>1.47/1</td>
<td>0.02/7</td>
</tr>
<tr>
<td>PID</td>
<td>0.03/7</td>
<td>0.03/5</td>
<td>0.03/6</td>
<td>0.14/4</td>
<td>0.71/2</td>
<td>18.4/1</td>
<td>0.55/3</td>
</tr>
<tr>
<td>Yeast</td>
<td>0.05/7</td>
<td>0.05/6</td>
<td>0.11/5</td>
<td>0.27/4</td>
<td>0.74/3</td>
<td>63.97/1</td>
<td>2.22/2</td>
</tr>
<tr>
<td>BC</td>
<td>0.06/7</td>
<td>0.07/6</td>
<td>0.09/5</td>
<td>0.31/3</td>
<td>1.29/2</td>
<td>28.15/1</td>
<td>0.26/4</td>
</tr>
<tr>
<td>Glass</td>
<td>0.03/6</td>
<td>0.02/7</td>
<td>0.03/5</td>
<td>0.09/3</td>
<td>0.67/2</td>
<td>1.00/1</td>
<td>0.06/4</td>
</tr>
<tr>
<td>Wine</td>
<td>0.05/4</td>
<td>0.03/6</td>
<td>0.04/5</td>
<td>0.19/3</td>
<td>0.71/2</td>
<td>0.90/1</td>
<td>0.03/7</td>
</tr>
<tr>
<td>Zoo</td>
<td>0.01/6</td>
<td>0.03/5</td>
<td>0.05/4</td>
<td>0.28/2</td>
<td>0.68/1</td>
<td>0.23/3</td>
<td>0.01/7</td>
</tr>
<tr>
<td>Parkinsons</td>
<td>0.04/5</td>
<td>0.04/6</td>
<td>0.02/7</td>
<td>0.11/3</td>
<td>0.53/2</td>
<td>0.80/1</td>
<td>0.09/4</td>
</tr>
<tr>
<td>WFRN</td>
<td>0.70/7</td>
<td>0.78/5</td>
<td>3.73/3</td>
<td>0.87/4</td>
<td>0.71/6</td>
<td>150.16/1</td>
<td>21.42/2</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>0.04/7</td>
<td>0.10/5</td>
<td>0.07/6</td>
<td>0.19/3</td>
<td>0.75/2</td>
<td>3.73/1</td>
<td>0.14/4</td>
</tr>
<tr>
<td>SL</td>
<td>0.07/7</td>
<td>0.25/4</td>
<td>0.18/5</td>
<td>0.28/3</td>
<td>0.70/2</td>
<td>0.87/1</td>
<td>0.09/6</td>
</tr>
<tr>
<td>FOTP</td>
<td>2.13/6</td>
<td>2.61/4</td>
<td>2.86/3</td>
<td>2.28/5</td>
<td>2.01/7</td>
<td>357.28/1</td>
<td>60.32/2</td>
</tr>
<tr>
<td>Sonar</td>
<td>0.09/5</td>
<td>0.15/3</td>
<td>0.08/6</td>
<td>0.08/7</td>
<td>0.53/2</td>
<td>1.42/1</td>
<td>0.11/4</td>
</tr>
<tr>
<td>CA</td>
<td>0.34/7</td>
<td>0.47/6</td>
<td>0.95/3</td>
<td>0.59/5</td>
<td>0.61/4</td>
<td>9.91/1</td>
<td>2.40/2</td>
</tr>
</tbody>
</table>

Avg. Rank | 6.17 | 5.47 | 4.88 | 3.53 | 2.65 | 1.18 | 4.12 |

of attributes, classes and instances. The desired output of the $d$th instance $Y_d$ is a vector consisting of “1” and “2”, where the position of “2” is the class label. For example, if the $d$th instance belongs to the 3rd class in a 3-class dataset, then $Y_d = [1 \ 1 \ 2]$.

In the experiments, each classification dataset in Table 3.2 is randomly and equally split into two sub-datasets. One of the sub-datasets is used as the training dataset, while the other one is used as the testing dataset.

Classification Approaches Settings

For comparisons, the RNN function approximator, Chebyshev-polynomial neural network (CPN-N) [ZYG+14a], ELM [HZS06], MLP equipped with the Levenberg-Marquardt (LM) algorithm, radial-basis-function neural networks (RBFNN) [WM96, YKN09] and SVM are exploited to classify the datasets in Table 3.2. The numerical experiments of these classification approaches
3.4. Application to Pattern Classification

are conducted in the MATLAB R2014a environment, operated on a personal computer (CPU: Intel i7-4770 3.40 GHz; memory: 8.00 GB). For better understanding, brief introductions and the settings of these classification methods are given in the followings.

- The RNN classifier exploits an ensemble of multi RNN function approximators for handling the multi-output classification problems. The parameters in the RNN function approximator are determined based on the procedure described in Section 3.3. In addition, its structure (i.e., the hidden-unit number) is selected by adapting the 4-fold cross-validation-based algorithm \[AC^{+10}, ZYG^{+14a}\]. In each fold, the training dataset is split into a training subset and a validation subset and then the hidden-unit number gradually increases one by one to search for the one achieving the lowest validation error. The final hidden-unit number is obtained by combining the results of 4 folds. Input values of datasets are linearly normalized into \([0\ 1]\).
- The single-output CPNN (SOCPNN) and multi-output CPNN (MOCPNN) are conducted using the source codes in http://www.yonghuayin.icoc.cc provided by \[ZYG^{+14a}\]. Input values of datasets are also linearly normalized into \([0\ 1]\) for the CPNN.
- Hidden-layer neurons of both the ELM and MLP are activated by \[\text{tansig}(x) = \frac{2}{[1 + \exp(-2x)]} - 1\]. The source code to implement the ELM \[HZS06\] is downloaded from http://www.ntu.edu.sg/home/egbhuang/elm_codes.html. The MLP is trained by the LM algorithm \[HBH07\] using the MATLAB toolbox. The structures of the ELM and MLP are selected by the cross-validation-based algorithm \[ZYG^{+14a}\].
- The RBFNN is a regularization single-hidden-layer feed-forward network (SLFN) \[WM96, YKN09\] and each instance in the training dataset corresponds to a hidden-layer neuron. The weights in the RBFNN are determined by the Moore-Penrose pseudoinverse method.
- The SVM (C-support vector classification with RBF kernel) is conducted using the source code provided by \[CLL1\] (available at http://www.csie.ntu.edu.tw/~cjlin/libsvm).

3.4.2 Numerical Results and Comparisons

The results on testing accuracies, testing time and unit numbers by using the RNN classifier, SOCPNN, MOCPNN, ELM, MLP, RBFNN and SVM to classify the datasets in Table 3.2 are
Table 3.5: Comparison of numbers of hidden units (or basis-function units)/support vectors of RNN, SOCPNN, MOCPNN, ELM and MLP (with numbers of hidden units determined by cross validation) as well as regularized RBFNN and SVM for pattern classification

<table>
<thead>
<tr>
<th>Dataset</th>
<th>RNN</th>
<th>SOCPNN</th>
<th>MOCPNN</th>
<th>ELM</th>
<th>MLP-LM</th>
<th>RBFNN</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris</td>
<td>71</td>
<td>11</td>
<td>51</td>
<td>14</td>
<td>19</td>
<td>75</td>
<td>29</td>
</tr>
<tr>
<td>TAE</td>
<td>20</td>
<td>7</td>
<td>14</td>
<td>4</td>
<td>16</td>
<td>76</td>
<td>67</td>
</tr>
<tr>
<td>LD</td>
<td>16</td>
<td>7</td>
<td>14</td>
<td>14</td>
<td>16</td>
<td>173</td>
<td>173</td>
</tr>
<tr>
<td>Seeds</td>
<td>61</td>
<td>21</td>
<td>59</td>
<td>17</td>
<td>6</td>
<td>105</td>
<td>41</td>
</tr>
<tr>
<td>PID</td>
<td>28</td>
<td>8</td>
<td>16</td>
<td>14</td>
<td>14</td>
<td>384</td>
<td>384</td>
</tr>
<tr>
<td>Yeast</td>
<td>160</td>
<td>16</td>
<td>158</td>
<td>19</td>
<td>11</td>
<td>745</td>
<td>740</td>
</tr>
<tr>
<td>BC</td>
<td>40</td>
<td>19</td>
<td>38</td>
<td>29</td>
<td>15</td>
<td>350</td>
<td>167</td>
</tr>
<tr>
<td>Glass</td>
<td>92</td>
<td>14</td>
<td>80</td>
<td>22</td>
<td>25</td>
<td>109</td>
<td>99</td>
</tr>
<tr>
<td>Wine</td>
<td>46</td>
<td>15</td>
<td>50</td>
<td>24</td>
<td>11</td>
<td>90</td>
<td>90</td>
</tr>
<tr>
<td>Zoo</td>
<td>70</td>
<td>13</td>
<td>63</td>
<td>11</td>
<td>7</td>
<td>52</td>
<td>39</td>
</tr>
<tr>
<td>Parkinsons</td>
<td>38</td>
<td>10</td>
<td>20</td>
<td>14</td>
<td>11</td>
<td>98</td>
<td>98</td>
</tr>
<tr>
<td>WFRN</td>
<td>403</td>
<td>54</td>
<td>464</td>
<td>45</td>
<td>25</td>
<td>2729</td>
<td>1536</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>20</td>
<td>13</td>
<td>26</td>
<td>32</td>
<td>7</td>
<td>176</td>
<td>91</td>
</tr>
<tr>
<td>SL</td>
<td>465</td>
<td>37</td>
<td>453</td>
<td>19</td>
<td>14</td>
<td>97</td>
<td>97</td>
</tr>
<tr>
<td>FOTP</td>
<td>254</td>
<td>54</td>
<td>169</td>
<td>82</td>
<td>29</td>
<td>3060</td>
<td>2482</td>
</tr>
<tr>
<td>Sonar</td>
<td>32</td>
<td>16</td>
<td>32</td>
<td>16</td>
<td>15</td>
<td>105</td>
<td>101</td>
</tr>
<tr>
<td>CA</td>
<td>113</td>
<td>9</td>
<td>98</td>
<td>25</td>
<td>14</td>
<td>214</td>
<td>214</td>
</tr>
</tbody>
</table>

respectively concluded in Tables 3.3, 3.4 and 3.5. To rank the classification performances of these approaches, the average-rank ranking method is adapted from [BS00]. For the results of each dataset in Table 3.3, the approach achieving the highest testing accuracy rate gets the highest rank, i.e., 7, while the one with the lowest testing accuracy rate gets the lowest rank, i.e., 1. After ranking the approaches for all datasets, the average rank for each approach with respect to all datasets is calculated, where a higher average rank indicates a better classification accuracy. Table 3.3 shows that the RNN achieves the highest ranks for five datasets and the highest average rank among all datasets. The RBFNN also achieves the five highest ranks; however, its average rank is much lower than that of the RNN. The average rank of the MLP is only slightly lower; however, it only achieves two of the highest ranks. Therefore, in general, the RNN achieves the best testing accuracy in terms of stability and superiority.

For the testing-time results in Table 3.4, we apply a similar ranking method, where the difference is that the approach achieving the lowest testing time (fastest) gets the highest rank, i.e., 7.
Note that some testing-time results for two approaches may seem to be the same while the ranks are different in Table 3.4 as the results in Tables 3.3 and 3.4 have been rounded but the ranking is conducted using the original unrounded results.

From Table 3.5, we can see that the hidden-unit numbers of the RNN and MOCPNN are generally higher than that of the SOCPNN, corresponding to the results in Tables 3.3 and 3.4. Note that the hidden-unit numbers in Table 3.5 are selected to achieve the nearly-optimal testing accuracies. From Table 3.3, in many cases, e.g., the Yeast, WFRN, SL and CA datasets, the RNN and MOCPNN learn the datasets much better than the SOCPNN. A larger number of hidden units could provide a better learning capability but, at the same time, increases the computational complexity of the network. As seen from Table 3.5, the hidden-unit numbers of the regularized RBFNN for the WFRN and FOTP are so high that the corresponding testing time could be more than one hundred times longer that those of other methods. It is worth pointing out that, though the hidden-unit numbers of the RNN and MOCPNN are similar, the testing time of the RNN is generally lower than that of the MOCPNN, based on the results in Table 3.4. For the LD dataset, the RNN and MLP have the same hidden-unit numbers, but the RNN is significantly faster (i.e., has a much lower testing time) than the MLP. This also verifies that the RNN has a lower computational complexity than the MOCPNN and MLP. For the Yeast and SL datasets, the RNN and MOCPNN have similar hidden-unit numbers, but the RNN has better testing accuracies than the MOCPNN. For many cases, e.g., the PID, Yeast, Wine and Ionosphere datasets, the RNN has more hidden units than the ELM and MLP-LM but has better testing accuracies. These results also demonstrate that the RNN has a good prediction capability in classification.

Therefore, it can be concluded from Tables 3.3 3.4 and 3.5 that the RNN achieves the highest ranks in terms of both the testing accuracy rates and the testing time, demonstrating that the RNN can perform pattern classification with both the highest accuracy and the fastest speed among these seven approaches.
3.5 Conclusions

This work has summarized the theoretical work of the RNN function approximator from [GML99a, GML99b, GML04] into five lemmas and a theorem. The procedure to configure the RNN function approximator to learn data has been proposed. Based on the configuration procedure, the RNN function approximator has been applied to pattern classification. Numerical results on real world datasets have illustrated that the resultant RNN classifier is able to achieve, in most cases, the highest accuracy and the fastest speed among the other compared five different classifiers.

The work in this chapter has laid a theoretical basis for the learning capability from data of the RNN and the investigation into the capability of the RNN for deep learning in the following chapters.
Chapter 4

Non-negative Autoencoders with Simplified Random Neural Network

4.1 Introduction

This chapter combines the knowledge from three machine learning fields, which are: the random neural network (i.e., the RNN), deep learning, and non-negative matrix factorisation (NMF), to develop new multi-layer non-negative autoencoders that have the potential to be implemented in a highly-distributed manner.


4.1.1 Technical Background

The mathematical tool of the RNN has existed since 1989 [Gel89, Gel90, Gel93], but is less well known in the machine-learning community. The RNN is developed to mimic the behaviour of biological neurons in the brain. In the RNN, an arbitrarily large set of neurons interact with
each other via stochastic spikes which modify each neuron’s action potential in continuous time. Though the stochastic spiking behaviors are complex, the state probability distribution of the RNN can be calculated by an easily solvable system of non-linear equations. This powerful property of the RNN provides the feasibility of utilizing the techniques from the other machine learning fields, e.g., the deep learning and NMF described as the following, for the development of new learning tools based on the RNN.

Deep learning has achieved great success in machine learning [HS06, LBH15]. The following network architecture and the training techniques could be combined with the RNN. In deep learning, the feed-forward neural-network architecture, composed of multi-processing layers, allows a model to extract high-level representations from raw data. Pre-training a multi-layer network layer by layer is an effective and widely-adaptable technique to tackle the training difficulty in the network [HS06, HOT06]. In addition, the typical training procedure called stochastic gradient descent (SGD) provides a practical choice for handling large datasets [BB08].

Non-negative matrix factorisation (NMF) is also a popular topic in machine learning [LS99, LYJ10, Hoy02, WZ13, DLPP06], and it provides another perspective of optimizing the parameters of the RNN with the constraints of non-negativity. Lee [LS99] suggested that the perception of the whole in the brain may be based on the part-based representations (based on the physiological evidence [WOP94]) and proposed simple yet effective update rules. Hoyer [Hoy02] combined sparse coding and NMF that allows control over sparseness. Ding investigated the equivalence between the NMF and K-means clustering in [DHS05, DLPP06] and presented simple update rules for orthogonal NMF. Wang [WZ13] provided a comprehensive review on recent processes in the NMF area.

### 4.1.2 Method Overview

This chapter first exploits the structure of the RNN equations as a quasi-linear structure. Using it in the feed-forward case, an RNN-based shallow non-negative autoencoder is constructed. This shallow autoencoder is then stacked into a multi-layer feed-forward autoencoder following the network architecture in the deep learning area [HS06, HOT06, LBH15]. Since connecting
weights in the RNN are products of firing rates and transition probabilities, they are subject to the constraints of non-negativity and that the sum of probabilities is no larger than 1, which are called the RNN constraints in this chapter. In view of this, the conventional gradient descent is not applicable for training such an autoencoder. By adapting the update rules from non-negative graph embedding \cite{LY10} that is closely related to NMF, applicable update rules are developed for the autoencoder, which satisfy the first RNN constraint of non-negativity. For the second RNN constraint, we impose a check-and-adjust procedure into the iterative learning process of the learning algorithms. The training procedure of the SGD is also adapted into the algorithms. The learning efficacy of the non-negative autoencoders equipped with the learning algorithms is well verified via numerical experiments on both typical image datasets including the MNIST \cite{LBBH98}, Yale face \cite{CHH07} and CIFAR-10 \cite{KH09} datasets and 16 real-world datasets in different areas from the UCI machine learning repository \cite{Lic13}, in terms of convergence and reconstruction performance and dimensionality-reduction capability for pattern classification.

4.2 Random Neural Network in Quasi-Linear Structure

4.2.1 Recurrent RNN

Let us recall the model description of the recurrent RNN as the following. An arbitrary neuron in the RNN can receive excitatory or inhibitory spikes from external sources, in which case they arrive according to independent Poisson processes. Excitatory or inhibitory spikes can also arrive from other neurons to a given neuron, in which case they arrive when the sending neuron fires, which happens only if that neuron’s input state is positive (i.e., the neuron is excited) and inter-firing intervals from the same neuron $v$ are exponentially distributed random variables with rate $r_v \geq 0$. Since the firing times depend on the internal state of the sensing neuron, the arrival process of neurons from other cells is not in general Poisson. From the preceding assumptions, it was proved in \cite{Gel93} that, for an arbitrary $N$ neuron RNN, which may or may not be recurrent (i.e. containing feedback loops), the probability in steady-state
that any cell \( h \), located anywhere in the network, is excited is given by the expression:

\[
q_h = \min\left( \frac{\lambda_h^+ + \sum_{v=1}^{N} q_v r_v p_{vh}^+}{r_h + \lambda_h^- + \sum_{v=1}^{N} q_v r_v p_{vh}^-}, 1 \right),
\]

for \( h = 1, \ldots, N \), where \( p_{vh}^+ \), \( p_{vh}^- \) are the probabilities that cell \( v \) may send excitatory or inhibitory spikes to cell \( h \), and \( \lambda_h^+ \), \( \lambda_h^- \) are the external arrival rates of excitatory and inhibitory spikes to neuron \( h \). Note that \( \min(a, b) \) is an element-wise operation whose output is the smaller one between \( a \) and \( b \). In [Gel93], it was shown that the system of \( N \) non-linear equations (4.1) have a solution which is unique.

### 4.2.2 Quasi-Linear RNN

Before adapting the RNN as a non-negative autoencoder (Section 4.3), the recurrent RNN model (4.1) is exploited as the feed-forward structure shown in Figure 4.1. The feed-forward RNN has an input layer and a hidden layer. The \( V \) input neurons receive excitatory spikes from only the outside world and fire excitatory spikes to the \( H \) hidden neurons and the outside world. The \( H \) hidden neurons receive excitatory spikes from only the \( V \) input neurons. Note there is no inhibitory spike in the network. Specifically, neurons in this model interact with each other in the following manner, where \( h = 1, \ldots, H \) and \( v = 1, \ldots, V \). When the \( v \)th input neuron fires, it sends excitatory spikes to the \( h \)th hidden neuron with probability \( p_{vh}^+ \geq 0 \). Evidently, \( \sum_{h=1}^{H} p_{v,h}^+ \leq 1 \). The \( v \)th input neuron receives excitatory spikes from the outside world with rate \( x_v \geq 0 \). When the \( h \)th hidden neuron fires, it sends excitatory spikes outside the network.

Let us denote by \( \hat{q}_v \) the probability that the \( v \)th input neuron \( (v = 1, \ldots, V) \) is excited and \( q_h \) the probability that the \( h \)th hidden neuron \( (h = 1, \ldots, H) \) is excited. According to [Gel89] and RNN equation (4.1), they are obtained by \( \hat{q}_v = \min(\hat{\Lambda}_v^+ / \hat{r}_v, 1) \), and \( q_h = \min(\Lambda_h^+ / r_h, 1) \), where the quantities \( \hat{\Lambda}_v^+ \) and \( \Lambda_h^+ \) represent the total average arrival rates of excitatory spikes, \( \hat{r}_v \) and \( r_h \) represent the firing rates of the neurons. Let us denote \( w_{v,h} = p_{v,h}^+ \hat{r}_v \). For simplicity, let us set the firing rates of all neurons to \( \hat{r}_v = r_h = 1 \) or that \( \sum_{h=1}^{H} w_{v,h} \leq 1 \). Then, \( \hat{\Lambda}_v^+ = x_v \).
4.3. Shallow Non-negative Autoencoder

\[ \hat{r}_v = 1, \quad \Lambda_h^+ = \sum_{v=1}^{V} w_{v,h} \hat{q}_v, \]
and using the fact that \( \hat{q}_v, q_h \) are probabilities, we can write:

\[
\hat{q}_v = \min(x_v, 1), \quad q_h = \min(\sum_{v=1}^{V} w_{v,h} \hat{q}_v, 1),
\] (4.2)

subject to $\sum_{h=1}^{H} w_{v,h} \leq 1$. It can be seen from (4.2) that this simplified RNN is quasi-linear. Therefore, for the network shown in Figure 4.1 we call it a quasi-linear RNN (LRNN).

4.3 Shallow Non-negative Autoencoder

This section proposes the model of the shallow non-negative autoencoder based on the quasi-linear RNN described in Section 4.2 and designs its training procedure.

4.3.1 Model Description

The model of the shallow non-negative autoencoder is presented in both the scalar and matrix forms.
Figure 4.2: Structure of a shallow non-negative LRNN autoencoder.

Scalar-Form Model

An output layer with $O$ neurons is added on top of the hidden layer of the LRNN shown in Figure 4.1 to construct a shallow non-negative LRNN autoencoder, whose structure is given in Figure 4.2. Let $\tilde{q}_o$ denote the probability that the $o$th output neuron is excited, and the $o$th output neurons interact with the LRNN in the following manner, where $o = 1, \cdots, O$.

- When the $h$th hidden neuron fires, it sends excitatory spikes to the $o$th output neuron with probability $\bar{p}_{h,o}^+ \geq 0$. Also, $\sum_{o=1}^{O} \bar{p}_{h,o}^+ \leq 1$.
- The firing rate of the $o$th output neuron $\tau_o = 1$.

Let $\bar{w}_{h,o} = \bar{p}_{h,o}^+ \tau_h = \bar{p}_{h,o}^\dagger$. Then, $\sum_{o=1}^{O} \bar{w}_{h,o} \leq 1$. The shallow non-negative LRNN autoencoder is described by

\[
\begin{align*}
\hat{q}_v &= \min(x_v, 1), \\
q_h &= \min(\sum_{v=1}^{V} w_{v,h} \hat{q}_v, 1), \\
\tilde{q}_o &= \min(\sum_{h=1}^{H} \bar{w}_{h,o} q_h, 1),
\end{align*}
\]

(4.3)

where $O = V$ and the input, hidden and output layers are the visual, encoding and decoding layers. The RNN constraints are $w_{v,h} \geq 0$, $\bar{w}_{h,o} \geq 0$, $\sum_{h=1}^{H} w_{v,h} \leq 1$ and $\sum_{o=1}^{O} \bar{w}_{h,o} \leq 1$. 
4.3. Shallow Non-negative Autoencoder

Matrix-Form Model

Suppose there is a dataset denoted by a non-negative $D \times V$ matrix $X = [x_{d,v}]$, where $D$ is the number of instances, each instance has $V$ attributes and $x_{d,v}$ is the $v$th attribute of the $d$th instance. We import $X$ into the input layer of the LRNN autoencoder. Let $\hat{q}_{d,v}$, $q_{d,h}$ and $\bar{q}_{d,o}$ respectively denote the values of $\hat{q}_v$, $q_d$ and $\bar{q}_o$ for the $d$th instance.

Let a $D \times V$-matrix $\hat{Q} = [\hat{q}_{d,v}]$, a $D \times H$-matrix $Q = [q_{d,h}]$, a $D \times O$-matrix $\bar{Q} = [\bar{q}_{d,o}]$, a $V \times U$-matrix $W = [w_{v,h}]$ and a $H \times O$-matrix $\bar{W} = [\bar{w}_{h,o}]$. Then, (4.3) can be rewritten as the following matrix form:

$$\begin{cases} \hat{Q} = \min(X, 1), Q = \min(\hat{Q}W, 1), \\ \bar{Q} = \min(Q\bar{W}, 1), \end{cases}$$

subject to the RNN constraints $W \geq 0$, $\bar{W} \geq 0$, $\sum_{h=1}^{H} w_{v,h} \leq 1$ and $\sum_{o=1}^{O} \bar{w}_{h,o} \leq 1$.

4.3.2 Problem Formulation

The objective is to adjust parameters of $W$ and $\bar{W}$ such that the LRNN autoencoder (4.4) learns representations from original data for the purpose of encoding or dimensionality reduction. Given the dataset $X = [x_{d,v}]$, the training of the LRNN autoencoder (4.4) can be formulated as the following optimization/reconstruction problem with equality and inequality constraints:

$$\min_{W,\bar{W}} ||X - \bar{Q}||^2 = ||X - \min(\min(X, 1)W, 1)\bar{W}, 1)||^2,$$

s.t. $W \geq 0$, $\bar{W} \geq 0$, $\sum_{h=1}^{H} w_{v,h} \leq 1$, $\sum_{o=1}^{O} \bar{w}_{h,o} \leq 1.$

4.3.3 Solution

Under certain assumptions, optimization problem (4.5) can be simplified. Specifically, let us assume that $X \leq 1$, $\hat{Q}W \leq 1$, $Q\bar{W} \leq 1$ and $\sum_{h=1}^{H} w_{v,h} \leq 1$, $\sum_{o=1}^{O} \bar{w}_{h,o} \leq 1$. Then, optimization
problem (4.5) become the following:

$$\min_{W,W} ||X - XW\textbf{W}||^2, \text{ s.t. } W \geq 0, \textbf{W} \geq 0. \quad (4.6)$$

Then, based on Liu’s work [LYJ10], update rules can be designed for solving the simplified optimization problem in the following subsections.

**Update Rule of $W$ for Given $\textbf{W}$**

Let us denote the objective function as $F(W) = ||X - XW\textbf{W}||^2$. A Lagrange-multiplier $V \times U$-matrix $\Phi = [\phi_{v,h}]$ is defined, in order to integrate the non-negative constraint of $W = [w_{v,h}]$ into the objective function. Then, the Lagrange function is defined as $L(W) = ||X - XW\textbf{W}||^2 + \text{Tr}(\Phi W^T)$, where operation $\text{Tr}(\cdot)$ produces the trace of a square matrix and $W^T$ denotes the transposition of matrix $W$. Taking the partial derivative of $L(W)$ with respect to $W$ and setting it to zero, we have

$$\frac{\partial L(W)}{\partial W} = -2X^T(X - XW\textbf{W})W^T + \Phi = -2X^TX\textbf{W}^T + 2X^TXW\textbf{W}W^T + \Phi = 0. \quad (4.7)$$

Each part in equation (4.7) is a matrix with dimensions $V \times H$. Multiplying the element in the $v$th row and $h$th column of each part in equation (4.7) with the weight $w_{v,h}$, we have

$$-2(X^TX\textbf{W}^T)_{v,h}w_{v,h} + 2(X^TXW\textbf{W}W^T)_{v,h}w_{v,h} + \phi_{v,h}w_{v,h} = 0, \quad (4.8)$$

where the symbol $(\cdot)_{v,h}$ denotes the element in the $v$th row and $h$th column of a matrix. According to the Karush-Kuhn-Tucker (KKT) condition [KT51] [BTW09], we set $\phi_{v,h}w_{v,h} = 0$.

Then, equation (4.8) becomes

$$(X^TX\textbf{W}^T)_{v,h}w_{v,h} = (X^TXW\textbf{W}W^T)_{v,h}w_{v,h}, \quad (4.9)$$
4.3. Shallow Non-negative Autoencoder

which leads to the update rule of $w_{v,h}$:

$$w_{v,h} \leftarrow w_{v,h} \frac{(X^T X W^T)_{v,h}}{(X^T X W W^T)_{v,h}}. \quad (4.10)$$

Let us define the operation $\odot$ as an element-wise product of two matrices. The division between matrices is also an element-wise division operation. Then, the update rule (4.10) can be written in the following matrix form:

$$W \leftarrow W \odot \frac{X^T X W^T}{X^T X W W^T}. \quad (4.11)$$

**Update Rule of $\bar{W}$ for Given $W$**

To integrate the non-negative constraint of $\bar{W} = [\bar{w}_{h,o}]$ into the objective function $F(W) = ||X - X W \bar{W}||^2$, a Lagrange-multiplier $H \times O$-matrix $\Phi = [\phi_{h,o}]$ is defined. Then, the Lagrange function becomes $\bar{L}(\bar{W}) = ||X - X W \bar{W}||^2 + \text{Tr}(\Phi W^T)$. Taking the partial derivative of $\bar{L}(\bar{W})$ with respect to $\bar{W}$ and setting it to zero, we have

$$\frac{\partial \bar{L}(\bar{W})}{\partial \bar{W}} = -2 W^T X^T (X - X W \bar{W}) + \Phi = -2 W^T X^T X + 2 W^T X^T X W \bar{W} + \Phi = 0. \quad (4.12)$$

Multiplying the element in the $h$th row and $o$th column of each part in equation (4.12) with the weight $\bar{w}_{h,o}$, we have

$$-2(W^T X^T X)_{h,o} \bar{w}_{h,o} + 2(W^T X^T X W \bar{W})_{h,o} \bar{w}_{h,o} + \phi_{h,o} \bar{w}_{h,o} = 0. \quad (4.13)$$

According to the Karush-Kuhn-Tucker (KKT) condition [KT51, BTW09], we set $\phi_{h,o} \bar{w}_{h,o} = 0$. Then, equation (4.13) becomes

$$\bar{w}_{h,o} = (W^T X^T X W \bar{W})_{h,o} \bar{w}_{h,o}, \quad (4.14)$$
which leads to the update rule of $\bar{w}_{h,o}$:

$$
\bar{w}_{h,o} \leftarrow \bar{w}_{h,o} \frac{(W^T X^T X)_{h,o}}{(W^T X^T X W)_{h,o}},

(4.15)
$$

The matrix-form update rule of $\bar{W}$ is

$$
\bar{W} \leftarrow \bar{W} \odot \frac{W^T X^T X}{W^T X^T X W},

(4.16)
$$

### Training Procedure with Check-and-Adjust Technique

The procedure to train the shallow LRNN autoencoder (4.4) is given in Algorithm 1, where the operation $\max(\cdot)$ produces the maximal element in a vector/matrix. After each update, adjustments need to be made such that $W$ and $\bar{W}$ satisfy the RNN constraints. The operations of $w_{v,h} \leftarrow w_{v,h} / \sum_{h=1}^{H} w_{v,h}$ and $\bar{w}_{h,o} \leftarrow \bar{w}_{h,o} / \sum_{o=1}^{O} \bar{w}_{h,o}$ guarantee that the weights satisfy the RNN constraints. Alternatively, operations $W \leftarrow W / \max(X W)$ or $\bar{W} \leftarrow \bar{W} / \max(H \bar{W})$ can be used to normalize the weights to reduce the number of neurons that are saturated if $\max(H \bar{W}) > 1$ or $\max(H \bar{W}) > 1$. Note that, to avoid the division-by-zero problem, zero elements in the denominators of (4.11) and (4.16) are replaced with tiny positive values, (e.g., “eps” in MATLAB).

### 4.4 Deep Non-negative Autoencoder

This section proposes the model of the deep non-negative autoencoder based on the shallow non-negative autoencoder in Section 4.3 and designs its training procedures.

#### 4.4.1 Model Description

Multi LRNNs are stacked to build a deep non-negative LRNN autoencoder, which is given in Figure 4.3. The multi-layer autoencoder has a visual layer, $M$ encoding layers and $M$ decoding
Algorithm 1: Procedure for training a shallow LRNN autoencoder (4.4)

Randomly initialize $W$ and $\bar{W}$ that satisfy RNN constraints

while terminal condition is not satisfied do

for each minibatch $\bar{X}$ do

update $W$ with (4.11)

for $v = 1, \cdots, V$ do

if $\sum_{h=1}^{H} w_{v,h} > 1$

$w_{v,h} \leftarrow \frac{w_{v,h}}{\sum_{h=1}^{H} w_{v,h}}$

for $h = 1, \cdots, H$

$W \leftarrow W / \max(\bar{X}W)$

update $\bar{W}$ with (4.16)

for $h = 1, \cdots, H$ do

if $\sum_{o=1}^{O} \bar{w}_{h,o} > 1$

$\bar{w}_{h,o} \leftarrow \frac{\bar{w}_{h,o}}{\sum_{o=1}^{O} \bar{w}_{h,o}}$

for $o = 1, \cdots, O$

$H = \min(\bar{X}W, 1)$

$\bar{W} \leftarrow \bar{W} / \max(HW)$

layer ($M \geq 2$), and they are connected in series with excitatory weights. Let $H_m$ and $O_m$ respectively denote the numbers of neurons in the $m$th encoding layer and decoding layer, where $m = 1, \cdots, M$. For notation ease, let us define $H_0 = V$ and $O_0 = H_M$, where $V$ is the numbers of neurons in the visual layer (or say, the 0th encoding layer). In addition, we know that $H_m = O_{M-m}$ with $m = 1, \cdots, M - 1$.

Let $Q_0$ denote the state of the visual layer, $Q_m$ denote the state of the $m$th encoding layer and $\bar{Q}_m$ denote the state of the $m$th decoding layer. We import a dataset $X$ into the visual layer of the autoencoder. Then, the multi-layer LRNN autoencoder is described by

$$
\begin{align*}
Q_0 &= \min(X, 1), \quad Q_1 = \min(Q_0W_1, 1), \\
&\vdots \\
Q_m &= \min(Q_{m-1}W_m, 1), \cdots, \quad Q_M = \min(Q_{M-1}W_M, 1), \\
&\vdots \\
\bar{Q}_1 &= \min(Q_M\bar{W}_1, 1), \cdots, \quad \bar{Q}_m = \min(\bar{Q}_{m-1}\bar{W}_m, 1), \cdots, \quad \bar{Q}_M = \min(\bar{Q}_{M-1}\bar{W}_M, 1), 
\end{align*}
$$

(4.17)

with $m = 2, \cdots, M$, where $W_m = [w_{h_{m-1},h_m}]$ with $m = 1, \cdots, M$ is a $H_{m-1} \times H_m$ matrix, and $\bar{W}_m = [\bar{w}_{o_{m-1},o_m}]$ with $m = 1, \cdots, M$ is a $O_{m-1} \times O_m$ matrix. The RNN constraints for (4.17)
4.4.2 Problem Formulation

The advantage of building an autoencoder in the multi-layer architecture is to obtain richer multi-level representations from original data than that with a shallow autoencoder. In addition, the quasi-linear activation function of the RNN allows the deep autoencoder to remove redundant information layer by layer. The problem for the deep LRNN autoencoder (4.17) to learn dataset $X$ can be described as the following optimization problem:

$$
\arg \min_{W_m, \overline{W}_m} \|X - \overline{O}_M\|^2,
$$

subject to $W_m \geq 0, \overline{W}_m \geq 0, \sum_{h_m=1}^{H_m} w_{h_{m-1},h_m} \leq 1, \sum_{o_m=1}^{O_m} \overline{w}_{o_{m-1},o_m} \leq 1$ with $m = 1, \cdots, M$, which means the summation of each row in $W_m$ and $\overline{W}_m$ is not larger than 1.
4.4.3 Solutions

Layer-Wise Training Procedure

In the deep learning area, a conventional procedure to solve the optimization problem (4.18) is to use the chain rule and calculate the gradient descent of the objective function $||X - Q_M||^2$ with respect to $W_m$ and $\overline{W}_m$ and then derive the corresponding update rules. However, the training could be unstable using this approach. Therefore, instead of solving the optimization problem (4.18) directly, we apply the layer-wise procedure and break it into a series of sub-problems.

For illustrative convenience, let us define $X_1 = X$. With assumptions of $X_1 \leq 1$, $\sum_{h=1}^{H_1} w_{h_1, h} \leq 1$, $\sum_{o=1}^{O_M} w_{oM-1, oM} \leq 1$, the first sub-problem is

$$\arg \min_{W_1, \overline{W}_M} ||X_1 - X_1 W_1 \overline{W}_M||^2, \text{ s.t. } W_1 \geq 0, \overline{W}_M \geq 0. \quad (4.19)$$

Update rules for $W_1$ and $\overline{W}_M$ can be designed for this sub-problem based on the procedure described in Section 4.3:

$$W_1 \leftarrow W_1 \odot \frac{X_1^T X_1 \overline{W}_M^T}{X_1^T X_1 W_1 \overline{W}_M \overline{W}_M} \quad (4.20)$$

and

$$\overline{W}_M \leftarrow \overline{W}_M \odot \frac{W_1^T X_1^T X_1}{W_1^T X_1^T X_1 W_1 \overline{W}_M}. \quad (4.21)$$

Then, we can define $X_m = \min(X_{m-1} W_{m-1}, 1)$ with $m = 2, \ldots, M$. With assumptions of $\sum_{h=1}^{H_m} w_{h_{m-1}, h_m} \leq 1$, $\sum_{o=1}^{O_{M-m}} w_{oM-m, oM-m+1} \leq 1$, the $m$th sub-problem is

$$\arg \min_{W_m, \overline{W}_{M-m+1}} ||X_m - X_m W_m \overline{W}_{M-m+1}||^2, \text{ s.t. } W_m \geq 0, \overline{W}_{M-m+1} \geq 0. \quad (4.22)$$
The corresponding update rules can be designed as:

\[
W_m \leftarrow W_m \odot \frac{X_m^T X_m \overline{W}_{m-m+1}^T}{X_m^T X_m W_m \overline{W}_{m-m+1}^T \overline{W}_{m-m+1}}, \quad (4.23)
\]

and

\[
\overline{W}_{M-m+1} \leftarrow \overline{W}_{M-m+1} \odot \frac{W_m^T X_m^T X_m}{W_m^T X_m^T W_m \overline{W}_{m-m+1}}. \quad (4.24)
\]

In this manner, the parameters in the deep non-negative LRNN autoencoder \((4.17)\) can be obtained layer by layer. For better illustration, the layer-wise procedure to train the deep non-negative LRNN autoencoder \((4.17)\) is given in Algorithm 2.

**Algorithm 2** Procedure for training a deep LRNN autoencoder \((4.17)\) (layer-wise manner)

\[
X_1 = X
\]

**for** \(m = 1, \ldots, M\) **do**

  **Train** \(W_m\) and \(\overline{W}_{M-m+1}\) with Algorithm 1 that takes \(X_m\) as input dataset

  **if** \(m \neq M\) **do**

    \(X_{m+1} = \min(X_m W_m, 1)\)

**Mini-Batch Training Procedure**

To avoid loading the whole dataset into the computer memory, Algorithm 3, which applies the idea of mini-batch training in the stochastic gradient descent to train the autoencoder is developed. The update rules in Algorithm 3 could be

\[
W_m \leftarrow W_m \odot \frac{Q_m^T Q_{m-1} \overline{W}_{m-m+1}^T}{Q_m^T Q_{m-1} W_m \overline{W}_{m-m+1}^T \overline{W}_{m-m+1}}, \quad (4.25)
\]

\[
\overline{W}_{M-m+1} \leftarrow \overline{W}_{M-m+1} \odot \frac{W_m^T Q_m^T Q_{m-1}}{W_m^T Q_m^T W_{m-1} \overline{W}_{m-m+1}}, \quad (4.26)
\]

with \(m = 1, \ldots, M\) and the operation \(\odot\) denoting the element-wise product of the two matrices.

To avoid the division-by-zero problem, zero elements in denominators of \((4.25)\) and \((4.26)\) are
replaced with tiny positive values. After updating $W_m$, operations

$$w_{h_{m-1}, h_m} \leftarrow \frac{w_{h_{m-1}, h_m}}{\sum_{h_m=1}^m w_{h_{m-1}, h_m}}$$  \hspace{1cm} (4.27)$$

are conducted to satisfy the RNN constraints; similarly, after updating $\overline{W}_m$, operations

$$\overline{w}_{o_{m-1}, o_m} \leftarrow \frac{\overline{w}_{o_{m-1}, o_m}}{\sum_{o_m=1}^m \overline{w}_{o_{m-1}, o_m}}$$  \hspace{1cm} (4.28)$$

are conducted. Alternatively, operations $W_m \leftarrow W_m / \max(Q_{m-1}W_m)$ and $\overline{W} \leftarrow \overline{W} / \max(\overline{Q}_{m-1}\overline{W})$ are used to normalize the weights to reduce the number of neurons that are saturated.

**Algorithm 3** Proceduce for training a deep LRNN autoencoder (4.17) (minibatch manner)

Randomly initialize $W_m$ and $\overline{W}_m$ that satisfy RNN constraints (with $m = 1, \cdots, M$)

while terminal condition is not satisfied do

for each minibatch $\bar{X}$ do

for $m = 1, \cdots, M$ do

update $W_m$ with (4.25)

adjust $W_m$ to satisfy RNN constraints

normalize $W_m$ subject to $\bar{X}$

update $\overline{W}_m$ with (4.26)

adjust $\overline{W}_m$ to satisfy RNN constraints

normalize $\overline{W}_m$ subject to $\bar{X}$


### 4.5 Experimental Evaluations and Comparisons

#### 4.5.1 Datasets

**MNIST:** The MNIST dataset of handwritten digits [LBBH98] contains 60,000 and 10,000 images in the training and test dataset. The number of input attributes is 784 ($28 \times 28$ images), which are in $[0, 1]$. The number of classes is 10. Examples of these classes are given in Figure 4.4.

**Yale face:** This database [http://vision.ucsd.edu/content/yale-face-database](http://vision.ucsd.edu/content/yale-face-database) contains 165 gray scale images of 15 individuals. Here we use the pre-processed dataset from
Chapter 4. Non-negative Autoencoders with Simplified Random Neural Network

Figure 4.4: Examples of 10 classes in the MNIST dataset.

Figure 4.5: Examples of 10 of 15 classes in the Yale face dataset.

where each image is resized as $32 \times 32$ (1024 pixels). The number of classes is 15. Examples of 10 of these classes are given in Figure 4.5. Among 165 samples, 110 of them are used as the training dataset and the rest 55 samples are used as the testing dataset.

CIFAR-10: The CIFAR-10 dataset consists of 60,000 $32 \times 32$ colour images [KH09]. Each image has 3072 attributes. It contains 50,000 and 10,000 images in the training and test dataset. The number of classes is 10. Examples of 10 classes are given in Figure 4.5.

UCI real-world datasets: Besides image datasets, we conduct numerical experiments on different real-world datasets in different areas from the UCI machine learning repository [Lic13]. They are the Iris, Teaching Assistant, Evaluation (TAE), Liver Disorders (LD), Seeds, Pima
4.5. Experimental Evaluations and Comparisons

Figure 4.6: Examples of 10 classes in the CIFAR-10 dataset. The first row includes airplane, automobile, bird, cat, deer: the second row includes dog, frog, horse, ship, truck.

Indians Diabetes (PID), Breast Cancer Wisconsin (BC) [MSW90], Glass, Wine, Zoo, Parkinsons [LMR+07], Wall-Following Robot Navigation (WFRN) [FBVV09], Ionosphere [SWHB89], Soybean Large (SL), First-Order Theorem Proving (FOTP) [BHP14], Sonar [GS88] and Cardiac Arrhythmia (CA) [GADC97] datasets. The numbers of input attributes, classes and instances for each dataset are given in Table 3.2. For a better understanding of these datasets and the related classification problems, each dataset is given a brief description in Appendix D.1.2.

Each classification dataset is randomly and equally split into two sub-datasets. One of the sub-datasets is used as the training dataset, while the other one is used as the testing dataset.

4.5.2 Convergence Performance in Training

Results of MNIST: Let us first test the convergence performance in training the shallow non-negative LRNN autoencoder. Structures of $784 \rightarrow 100$ (for simplicity, we use the encoding part to represent an autoencoder) and $784 \rightarrow 50$ and the MNIST dataset are used for experiments. The training dataset of 60,000 images is used for training. Figures 4.7(a) and 4.7(b) show the curves of the reconstruction error versus the number of iterations, where, in each iteration, a minibatch of size 100 is handled and the reconstruction error is measured by the mean square error. A multi-layer non-negative LRNN autoencoder with structure $784 \rightarrow 1000 \rightarrow 500 \rightarrow 250 \rightarrow 50$ is then used, and the corresponding curve of the reconstruction error versus iterations
Figure 4.7: Reconstruction error (Y-axis) versus iteration number (X-axis) in training shallow and multi-layer LRNN autoencoders for the MNIST dataset.

It can be seen from Figure 4.7 that reconstruction errors using the LRNN autoencoders equipped with the developed algorithms converge well for different structures. In addition, the lowest errors using the shallow and multi-layer autoencoders are respectively 0.0204 and 0.0190. The results show that, for the same encoding dimension, the performances of the shallow and multi-layer structures are similar for this dataset.

Results of Yale face: Attribute values are normalized into $[0, 1]$ (by dividing by 255). The structures for the shallow and multi-layer LRNN autoencoders are respectively $1024 \rightarrow 50$ and $1024 \rightarrow 500 \rightarrow 100 \rightarrow 50$. Both the training and testing datasets (total 165 images) are used for training the autoencoders. The size of a minibatch is 5. Curves of reconstruction errors versus iterations are given in Figure 4.8. For this dataset, the shallow autoencoder seems more stable than the multi-layer one.

Results of CIFAR-10: Attribute values of the dataset are also divided by 255 for norma-
4.5. Experimental Evaluations and Comparisons

Figure 4.8: Reconstruction error (Y-axis) versus iteration number (X-axis) in training shallow and multi-layer LRNN autoencoders for the Yale face dataset.

Figure 4.9: Reconstruction error (Y-axis) versus iteration number (X-axis) in training shallow and multi-layer LRNN autoencoders for the CIFAR-10 dataset.

The size of minibatch is chosen as 100. The results are given in Figure 4.9. We can see that reconstruction errors for both structures converge as the number of iterations increases. In addition, the lowest reconstruction errors when using the shallow and multi-layer autoencoders are the same (0.0082). These results, together with those with the MNIST and Yale face datasets (Figures 4.7 to 4.9), verify the good convergence and reconstruction performance in training both the shallow and multi-layer LRNN autoencoders for handling image datasets.

**Results of UCI real-world datasets:** Let \( N \) denote the attribute number in a dataset. The structures of the LRNN autoencoders used are \( N \rightarrow \text{round}(N/2) \), where the operation \( \text{round}(\cdot) \)
produces the nearest integer number of its input. The attribute values are linear normalized in range \([0, 1]\). Both the training and testing datasets are used for training the autoencoders for all datasets. The size of mini-batches is set as 50 for all datasets. Curves of reconstruction errors versus iterations are given in Figures 4.10 to 4.12. We see that the reconstruction errors generally decrease as the number of iterations increases. These results also demonstrate the efficacy of the non-negative LRNN autoencoders equipped with the training algorithms.

**4.5.3 Dimensionality Reduction for Pattern Classification**

This subsection compares the LRNN autoencoder with other similar encoding algorithms based on accuracies in different classification tasks. Notably, we are not interested in fine tuning parameters (or hyper parameters) in a classifier to achieve state-of-the-art accuracies for specific datasets.

First, the inputs from a classification dataset are imported into an encoding algorithm. The encoded inputs are then used for classification. In the experiments, the MNIST, Yale face,
4.5. Experimental Evaluations and Comparisons

Figure 4.11: Reconstruction error (Y-axis) versus iteration number (X-axis) in training LRNN autoencoders for UCI real-world datasets: Glass, Wine, Zoo, Parkinsons, WFRN, Ionosphere.

CIFAR-10 and different UCI real-world datasets are used; the classifier is chosen as the extreme learning machine (ELM) due to its fast training speed [HZS06]; five encoding algorithms are exploited for comparisons, which are the principal component analysis (PCA), probabilistic PCA (PPCA), multi-layer deep autoencoder (DAE) [HS06], simplified non-negative graph embedding (SNGE) (that is equivalent to a shallow LRNN autoencoder without the RNN probability constraints and the LRNN activation $y = \min(x, 1)$), and shallow and multi-layer LRNN autoencoders (SLRNN and MLRNN). The PCA, PPCA and DAE are implemented by using the MATLAB toolbox developed by [MH08, VDMPVdH09]. The dimensions of encoded inputs of the MNIST, Yale face and CIFAR-10 datasets are 50, 50 and 150, respectively, while those for the UCI datasets are half of the original input dimensions ($N \rightarrow \text{round}(N/2)$). In these encoding experiments, the training dataset of the MNIST dataset is used for training, while both the training and testing datasets are used for training for the rest datasets. Note that the MLRNN is not applied to encoding the UCI datasets, since the dimensions of these datasets are low and the advantage of the MLRNN lays on handling higher dimension datasets, which will be further demonstrated by the following results and discussions.
Figure 4.12: Reconstruction error (Y-axis) versus iteration number (X-axis) in training LRNN autoencoders for UCI real-world datasets: SL, FOTP, Sonar, CA.

In the classification experiments, the number of hidden neurons is set as 2000 in the ELM for the MNIST and CIFAR-10 datasets, while the number is 50 for the Yale face dataset to avoid overfitting since it has a small training dataset (105 instances). For the UCI dataset, 2N hidden neurons are used. At least 10 trials (20 for MNIST and UCI, 100 for Yale face) are conducted for an experiment and the average results are obtained to reduce biases caused by the randomness in the ELM. In these classification experiments, for all datasets, the training and testing datasets are respectively used for training testing, where the separation of them has been described in Subsection 4.5.1. The results of testing accuracies (%) are summarized in Table 4.1, where “Original” in the table means the original dataset without using any encoding algorithm. We analyse these results in the following two cases.

**Results of Image datasets:** For the MNIST dataset, the DAE achieves the highest testing accuracy, while the second and third best ones are the SLRNN and MLRNN and their results are very close. For the Yale face and CIFAR-10 datasets, the MLRNN obtains the highest testing accuracies, slightly higher than those of the SLRNN. The reason may be that introducing multi
Table 4.1: Testing accuracies (%) using different encoding algorithms for the MNIST, Yale face, CIFAR-10 and UCI datastes in classification

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>Dim. No.</th>
<th>Original</th>
<th>PCA</th>
<th>PPCA</th>
<th>DAE</th>
<th>SNGE</th>
<th>SLRNN</th>
<th>MLRNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>784</td>
<td>95.32</td>
<td>95.97</td>
<td>92.63</td>
<td>97.15</td>
<td>95.55</td>
<td>96.67</td>
<td>96.54</td>
</tr>
<tr>
<td>Yale face</td>
<td>1024</td>
<td>32.97</td>
<td>52.59</td>
<td>37.62</td>
<td>33.30</td>
<td>30.48</td>
<td>53.33</td>
<td>55.83</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>3072</td>
<td>44.31</td>
<td>41.53</td>
<td>–</td>
<td>40.46</td>
<td>46.40</td>
<td>48.54</td>
<td>–</td>
</tr>
<tr>
<td>MNIST</td>
<td>784</td>
<td>95.32</td>
<td>95.97</td>
<td>92.63</td>
<td>97.15</td>
<td>95.55</td>
<td>96.67</td>
<td>96.54</td>
</tr>
<tr>
<td>Yale face</td>
<td>1024</td>
<td>32.97</td>
<td>52.59</td>
<td>37.62</td>
<td>33.30</td>
<td>30.48</td>
<td>53.33</td>
<td>55.83</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>3072</td>
<td>44.31</td>
<td>41.53</td>
<td>–</td>
<td>40.46</td>
<td>46.40</td>
<td>48.54</td>
<td>–</td>
</tr>
<tr>
<td>Iris</td>
<td>4</td>
<td>93.07</td>
<td>90.93</td>
<td>91.33</td>
<td>90.87</td>
<td>91.87</td>
<td>91.93</td>
<td>–</td>
</tr>
<tr>
<td>TAE</td>
<td>5</td>
<td>82.53</td>
<td>82.40</td>
<td>81.93</td>
<td>79.60</td>
<td>84.07</td>
<td>82.93</td>
<td>–</td>
</tr>
<tr>
<td>LD</td>
<td>5</td>
<td>77.70</td>
<td>77.35</td>
<td>74.01</td>
<td>77.27</td>
<td>73.75</td>
<td>79.45</td>
<td>–</td>
</tr>
<tr>
<td>Seeds</td>
<td>7</td>
<td>92.48</td>
<td>93.90</td>
<td>88.71</td>
<td>90.05</td>
<td>92.29</td>
<td>92.76</td>
<td>–</td>
</tr>
<tr>
<td>PID</td>
<td>8</td>
<td>74.02</td>
<td>71.46</td>
<td>64.47</td>
<td>71.73</td>
<td>72.85</td>
<td>70.14</td>
<td>–</td>
</tr>
<tr>
<td>BC</td>
<td>9</td>
<td>97.22</td>
<td>97.34</td>
<td>67.81</td>
<td>97.45</td>
<td>96.46</td>
<td>97.26</td>
<td>–</td>
</tr>
<tr>
<td>Glass</td>
<td>9</td>
<td>66.00</td>
<td>61.81</td>
<td>38.90</td>
<td>64.19</td>
<td>44.05</td>
<td>60.71</td>
<td>–</td>
</tr>
<tr>
<td>Wine</td>
<td>13</td>
<td>97.50</td>
<td>94.89</td>
<td>91.08</td>
<td>94.77</td>
<td>95.91</td>
<td>94.03</td>
<td>–</td>
</tr>
<tr>
<td>Zoo</td>
<td>16</td>
<td>89.90</td>
<td>85.52</td>
<td>85.83</td>
<td>88.85</td>
<td>73.44</td>
<td>91.67</td>
<td>–</td>
</tr>
<tr>
<td>Parkinsons</td>
<td>22</td>
<td>81.13</td>
<td>79.90</td>
<td>74.38</td>
<td>75.15</td>
<td>77.27</td>
<td>77.27</td>
<td>–</td>
</tr>
<tr>
<td>WFRN</td>
<td>24</td>
<td>68.99</td>
<td>70.65</td>
<td>42.57</td>
<td>71.67</td>
<td>62.66</td>
<td>69.91</td>
<td>–</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>34</td>
<td>85.26</td>
<td>85.06</td>
<td>83.94</td>
<td>85.09</td>
<td>85.66</td>
<td>82.63</td>
<td>–</td>
</tr>
<tr>
<td>SL</td>
<td>35</td>
<td>80.22</td>
<td>75.28</td>
<td>67.81</td>
<td>81.52</td>
<td>69.44</td>
<td>71.91</td>
<td>–</td>
</tr>
<tr>
<td>FOTP</td>
<td>51</td>
<td>50.14</td>
<td>50.40</td>
<td>47.12</td>
<td>50.86</td>
<td>49.81</td>
<td>50.89</td>
<td>–</td>
</tr>
<tr>
<td>Sonar</td>
<td>60</td>
<td>66.70</td>
<td>66.50</td>
<td>51.31</td>
<td>67.09</td>
<td>62.62</td>
<td>65.53</td>
<td>–</td>
</tr>
<tr>
<td>CA</td>
<td>279</td>
<td>55.53</td>
<td>61.40</td>
<td>47.51</td>
<td>52.00</td>
<td>52.15</td>
<td>56.71</td>
<td>–</td>
</tr>
</tbody>
</table>

layers into the LRNN autoencoder allows it to extract more useful information. For all three image datasets, the SLRNN and MLRNN achieve better results than the SNGE. The reason may be that the RNN constraints bring regularization into training and the LRNN activation \( y = \min(x, 1) \) allows the autoencoder to remove redundant information layer by layer.

Results of UCI real-world datasets: Firstly, we can see that, when the number of input dimensions of the datasets is low (e.g., less than 24), using encoding algorithms may not achieve better accuracies than using the original datasets directly for classification. There is no absolute winner for these encoding algorithms to handle these real-world datasets. There are performance drops when using the PPCA for the glass, WFRN and CA datasets, and similar phenomena occur for the SNGE on the glass dataset. Therefore, we could choose the PCA, DAE and SLRNN for these datasets.

Based on the results in Table 4.1, the curves of the testing accuracies versus the numbers of
Figure 4.13: Testing accuracies (%) versus input-dimension numbers of datasets using different encoding algorithms for the MNIST, Yale face, CIFAR-10 and UCI datasets in classification dataset dimensions (input dimensions) for all encoding algorithms are given in Figure 4.13. As seen from Figure 4.13(a), the performances of the proposed SLRNN and MLRNN become the best when the number of dataset dimensions is larger than 1000. For better illustration, the dimension numbers are presented in the log scale in Figure 4.13(b). From the figure, we can see that, when the dimension number is less than around 700, the PCA, DAE and SLRNN perform generally and similarly well for all tested datasets, while there are some performance drops when using the PPCA and SNGE. Again, we can see that, when the dimension number becomes larger than around 700, the testing accuracies of the SLRNN and MLRNN begin to surpass other compared encoding algorithms. These results demonstrate, to some extent, that the proposed SLRNN and MLRNN could work better for the datasets with higher dimensions.

4.6 Conclusions

In this chapter, new non-negative autoencoders have been proposed based on the RNN model, which adopt the feed-forward multi-layer network architecture in the deep-learning area. To comply the RNN constraints of non-negativity and that the sum of probabilities is no larger than 1, learning algorithms have been developed by adapting weight update rules from the NMF area. Numerical results based on typical image datasets, including the MNIST, Yale face and
CIFAR-10 datasets and 16 real-world datasets from different areas, have well verified the robust convergence and reconstruction performance and high learning accuracy of the autoencoders.
Chapter 5

Deep Learning with Dense Random Neural Network

5.1 Introduction

This chapter explores the idea that the human brain contains many important areas that are composed of dense clusters of cells, such as the basal ganglia and various nuclei. These clusters may be composed of similar or identical cells, or of cells of different varieties; however, due to the density of their arrangement, it may be that such clusters allow for a substantial or at least increased amount of direct communication between somata, in addition to the commonly exploited signalling through dendrites and synapses. Thus, based on recurrent RNN \cite{Gel89}, a mathematical model of dense clusters/nuclei is developed that models both synapses and direct soma-to-soma interactions, creating a dense random neural network (Dense RNN). Each cluster is modelled as a recurrent spiking RNN. Each neuron in each nucleus has a statistically identical interconnection structure with the other cells in the same nucleus; this statistical regularity allows for great individual variability among neurons both with regard to spiking times and interconnection patterns.

The model of dense clusters allows consideration of a multi-layer architecture (MLA) network wherein each layer is composed of a finite number of dense nuclei. Within each nucleus, the cells
communicate with each other in fully-connected recurrent structures that use both synapse and
direct soma-to-soma interaction [GT08a]. The communication structure between the various
layers of nuclei is a conventional multi-layer feedforward structure: the nuclei in the first layer
receive excitation signals from external sources, while each cell in each nucleus creates an
inhibitory projection to the layer up. The MLA of dense nuclei, modelled by the Dense RNN
and named as MLDRNN, is given schematically in Figure 5.2 where the details are explained
in the following sections.

Further, an efficient training procedure is developed for the MLDRNN, which combines unsu-
pervised and supervised learning techniques [HS06, TDH16, GY16a, ZYG+14a, KZH13]. The
multi-channel MLDRNN (MCMLDRNN), extended from the MLDRNN, is developed to handle
multi-channel classification datasets. Two more modifications of the MCMLDRNN are also de-
veloped for comparisons. The MCMLDRNN is applied to recognizing 3D objects, distinguishing
different chemical gases and detecting daily and sports activities of humans. The correspond-
ing numerical results indicate that the MCMLDRNN is more accurate than state-of-the-art
methods in most cases with high training efficiency.

Finally, the case where the number of cells in a nucleus is very large is investigated, by which
the transfer functions of dense nuclei are significantly simplified. Numerical results show that
a dense-enough nucleus produces a more efficient algorithm than a not-that-dense nucleus.

Part of this chapter has been published as:
(Available at https://arxiv.org/abs/1609.07160)
al Joint Conference on Neural Networks (IJCNN), pages 1633-1638, 2016.
5.1.1 Related Work

Specific synaptic connections between cells in the brain are formed via growing their axons and dendrites during the development [MKN+88]. Cells interact with each other via excitatory and inhibitory spikes through these synaptic connections, which modify each cell’s action potential. The synaptic contacts can be modelled by the random neural network (i.e., the RNN), which is developed to mimic the behaviours of biological cells [Gel89]. In an RNN, an arbitrarily large set of cells interact with each other via excitatory and inhibitory spikes which modify each cell’s action potential in continuous time. Mathematical properties of the RNN model have been rigorously established via the follow-up work [Gel90, Gel93].

In addition to the usual excitatory and inhibitory interactions between cells, synchronised firing (SF) has been observed among cultured cortical cells [MKN+88, RKJ+93]. The synchronization means that a large number of cells may possess cooperative dynamical properties, e.g., firing simultaneously and jointly acting upon other cells. The neural oscillations also reflect these synchronization phenomena. It has been suggested that synchronized activity is ubiquitous in the brain [FG06, DH13] and serves a role in information processing of both sensory and motor systems [KE95, GT08c]. It is also suggested that synchronization appears both in homogeneous and clustered neuronal networks [SSBBJ01, SBSBJ03]. Therefore, much effort has been devoted to the study of cell ensembles and clusters and their behaviors.

To provide a theoretical explanation for the cortical and thalamic oscillations, in [GC98c], the RNN was further used to construct a model of the thalamocortical feedback loop. The constructed model has three layers respectively for the thalamus, reticular layer and cortex. Specifically, all cortical cells involved are statistically identical and modelled by a recurrent RNN. In [GT08c, GT08b], Gelenbe discussed an extension of the RNN, which offers the possibility that cells synchronously act together on other cells, triggering successive firing instants in other cells. Specifically, when a cell fires, it may send the usual excitatory and inhibitory spikes towards another cell or trigger another cell to provoke a second-order effect on a third cell. In addition, a gradient-based learning algorithm has been developed to the model.
Based on the RNN theory, this chapter studies the model of dense clusters of cells. Due to the density of the arrangement, the cells within a cluster interact with each other with triggers, in addition to the commonly exploited spikes. When a first cell fires, it may send the spikes or triggers towards a second cell. The trigger may provoke a state change of a third cell and stop at the second cell, or the trigger may move on to the third cell and provoke an effect on a fourth cell. The procedure repeats until the trigger stops at a cell. Under this interaction manner, cascades of triggered firings of multi cells can occur when a cell fires. In addition, this chapter connects these dense clusters and the RNN models with deep learning for the first time.

5.2 Dense Random Neural Network

Based on the random neural network (i.e., the RNN) with excitatory and inhibitory spikes, this section proposes a mathematical model of Dense Random Neural Network (Dense RNN) with excitatory and inhibitory spikes and triggers, in which neurons interconnect via both conventional synaptic connections and soma-to-soma interactions.

5.2.1 Model Description

We consider a Dense RNN model composed of $M \geq 3$ neurons or cells, each of which receives excitatory (positive) and inhibitory (negative) spike trains from external sources, which may be sensory sources or cells. These arrivals occur according to independent Poisson processes of rates $\lambda_m^+$ for the excitatory spike train, and $\lambda_m^-$ for the inhibitory spike train, respectively, to neuron $m \in S = \{1, \ldots, M\}$.

In this model, each neuron is represented at time $t \geq 0$ by its internal state $k_m(t)$ which is a non-negative integer. If $k_m(t) > 0$, then the arrival of a negative spike to neuron $m$ at time $t$ results in the reduction of the internal state by one unit: $k_m(t^+) = k_m(t) - 1$. The arrival of a negative spike to a cell has no effect if $k_m(t) = 0$. On the other hand, the arrival of an excitatory spike always increases the neuron’s internal state by +1.
If $k_m(t) > 0$, then the neuron $m$ is said to be “excited”, and it may “fire” a spike with probability $r_m \Delta t$ in the interval $[t, t + \Delta t]$, where $r_m > 0$ is its “firing rate”, so that $r_m^{-1}$ may be viewed as the average firing delay of the excited $m$-th neuron. In addition, when a neuron fires, its internal state drops by 1.

Neurons in this model can interact in the following manner at time $t \geq 0$. If neuron $i$ is excited $i \in \{1, \ldots, M\}$, i.e. $k_i(t) > 0$, then it may fire. When neuron $i$ fires, one of the following four cases may happen:

1) Neuron $i$ can send a positive or excitatory spike to neuron $j$ with probability $p^+(i, j)$ resulting in $k_i(t^+) = k_i(t) - 1$ and $k_j(t^+) = k_j(t) + 1$.

2) Or neuron $i$ can send a negative or inhibitory spike to neuron $j$ with probability $p^-(i, j)$ so that $k_i(t^+) = k_i(t) - 1$ and $k_j(t^+) = \max(k_j(t) - 1, 0)$.

3) Or neuron $i$ can “trigger” neuron $j$ with probability $p(i, j)$ so that $k_i(t^+) = k_i(t) - 1$ and $k_j(t^+) = \max(k_j(t) - 1, 0)$. If $k_j(t) > 0$, one of the following two cases may also happen:
   (A) With probability $Q(j, m)$, we have $k_m(t^+) = k_m(t) + 1$ so that neurons $i$ and $j$ together have incremented the state of neuron $m$. Thus, we see that a trigger allows two neurons $i$ and $j$ to increase the excitation level of a third neuron $m$ by 1, while neurons $i$ and $j$ are both depleted by 1.
   (B) Or with probability $\rho(j, m)$ the trigger moves on to the neuron $m$ and then the sequence (A) or (B) is repeated among neurons $j$ and $m$ and another neuron $l$.

4) Or the corresponding spike or trigger from neuron $i$ is lost or it leaves the network with probability $d_i$.

Note that $\sum_{j=1}^{M} [p(i, j) + p^-(i, j) + p^+(i, j)] = 1 - d_i$. Also, $\sum_{m=1}^{M} [Q(j, m) + \rho(j, m)] = 1$.

### 5.2.2 Neural Excitation Probabilities

To lay a basis for further discussion, let $z_L(m) = (i_1, \ldots, i_j, \ldots, i_L)$ be any ordered sequence of numbers, where $m \in S = \{1, \ldots, M\}$, $L \geq 2$, and $i_j \in S$ with $j = 1, \ldots, L$. A sequence $z_L(m)$ describes a situation that a trigger starts from the neuron $i_1$, travels through neurons $i_2, \ldots, i_L$ and finally stops at the neuron $m$. Let us denote by $q_m = \lim_{t \to \infty} \text{Prob}[k_m(t) > 0]$
5.3 Modelling Dense Nuclei with Dense Random Neural Network

the probability that the neuron $m$ is excited. Based on the RNN and G-Networks theory \cite{GT08b,Gel00}, $q_m$ is given by the following expression:

$$q_m = \frac{\Lambda^+_m}{r_m + \Lambda^-_m},$$

(5.1)

where the variables in (5.1) are of the form:

$$\Lambda^+_m = \lambda^+_m + \sum_{j=1, j \neq m} r_j q_j p^+(j, m) + \sum_{L=2}^{+\infty} \left( \sum_{z_L(m)} \sum_{i_1} r_{i_1} q_{i_1} p(i_1, i_2) \left( \prod_{j=2, \ldots, L-1} q_{i_j} \rho(i_j, i_{j+1}) \right) q_{i_L} Q(i_L, m) \right),$$

(5.2)

$$\Lambda^-_m = \lambda^-_m + \sum_{j=1, j \neq m} r_j q_j p^-(j, m) + \sum_{i_1=1}^{M-1} r_{i_1} q_{i_1} p(i_1, m) + \sum_{L=2}^{+\infty} \left( \sum_{z_L(m)} \sum_{i_1} r_{i_1} q_{i_1} p(i_1, i_2) \left( \prod_{j=2, \ldots, L-1} q_{i_j} \rho(i_j, i_{j+1}) \right) q_{i_L} \rho(i_L, m) \right).$$

(5.3)

In the sequel, to simplify the notations we will write $w^+_{j,i} = r_j p^+(j, i)$ and $w^-_{j,i} = r_j p^-(j, i)$.

5.3 Modelling Dense Nuclei with Dense Random Neural Network

As already noted, we pursue the idea that the human brain contains important areas composed of dense clusters of cells, such as the basal ganglia and various nuclei. These clusters may be composed of similar or identical cells, or of varieties of cells. When large ensembles of hundreds or thousands of cells are represented, as in dense nuclei, probability models can be more convenient and tractable.
5.3.1 Clusters of Identical and Densely Connected Cells

Using the model of the Dense RNN, let us now consider the construction of special clusters of densely interconnected cells. We first consider a special network, call it $M(N)$, that contains $N$ connected cells that are homogenous and statistically identical. Each cell $n \in M(N)$ has a firing rate $r_n = r$ and external inhibitory and excitatory arrivals of spikes denoted by $\lambda_n^- = \lambda^-$ and $\lambda_n^+ = \lambda^+$, respectively. Since all cells in $M(N)$ are statistical identical, the state of each cell can be denoted by $q$, where the state here means the stationary excitation probability. Each cell in $M(N)$ receives an inhibitory input from the external cell $u$ with the state denoted by $\hat{q}_u$ which does not belong to $M(N)$. Thus, for any internal cell $i \in M(N)$, we have an inhibitory weight $w_u^- \equiv \hat{w}_{u, i} = \hat{r}_u \hat{p}^-(u, i) > 0$ from external cell $u$ to internal cell $i$, where $\hat{r}_u$ is the firing rate of external cell $u$ and $\hat{p}^-(u, i)$ is the probability that, when external cell $u$ fires, the spike goes to internal cell $i$. Evidently, $\hat{p}^-(u, i) = \hat{p}^-(u, j)$ for any $i, j \in M(N)$.

For any $i, j \in M(N)$, we have $p^-(i, j) = 0$ and $w^-_{i, j} = r_i p^-(i, j) = 0$. Whenever one of the cells fires, it triggers the firing of the other cells with $p(i, j) = p/N$ and $Q(i, j) = (1 - p)/N$. Since $\sum_{j=1}^N (Q(i, j) + \rho(i, j)) = 1$, we have $\rho(i, j) = p(i, j) = p/N$. Let us also set $p^+(i, j) = (1 - p)/N$ and $w^+_{i, j} = r_i p^+(i, j) = r(1 - p)/N$, where $\sum_{j=1}^N (p^-(i, j) + p^+(i, j) + p(i, j)) = 1$ satisfies. Based on (5.1), we can obtain the expression of $q_n$ for $n \in M(N)$. To lay a basis for further discussion, we also define $z_L(n) = (i_1, \cdots, i_L)$ with $L \geq 1, 1 \leq i_l \leq N$ and $i_i \neq n$. Then, we have

$$q_n = \frac{\Lambda_n^+}{r_n + \Lambda_n^-}, \quad (5.4)$$

where

$$\Lambda_n^+ = \lambda_n^+ + \sum_{j=1, j \neq n}^N r_j q_j p^+(j, n) + \sum_{all \, z_2(n)} r_{i_1} q_{i_1} p(i_1, i_2) q_{i_2} Q(i_2, n)$$

$$+ \sum_{L=3}^{+\infty} \left( \sum_{all \, z_L(n)} r_{i_1} q_{i_1} p(i_1, i_2) \left( \prod_{j=2, \cdots, L-1} q_{i_j} \rho(i_j, i_{j+1}) \right) q_{i_L} Q(i_L, n) \right), \quad (5.5)$$
\[ \Lambda_n^- = \lambda_n^- + \hat{q}_u \hat{w}_u^- + \sum_{j=1,j \neq n}^N r_{ij} q_{i1} p(i_1, n) + \sum_{\text{all } z_2(n)} r_{i1} q_{i1} p(i_1, i_2) q_{i2} \rho(i_2, n) + \sum_{L=3}^{+\infty} \left( \sum_{\text{all } z_L(n)} r_{i1} q_{i1} p(i_1, i_2) \left( \prod_{j=2,\ldots,L-1} q_{i_j} \rho(i_j, i_{j+1}) \right) q_{i_L} \rho(i_L, n) \right). \] (5.6)

Let us handle \( \Lambda_n^+ \) first. Since \( Q(i, j) = p^+(i, j) = (1 - p)/N \) and \( \rho(i, j) = p(i, j) = p/N \),

\[ \Lambda_n^+ = \Lambda^+ = \lambda^+ + \sum_{j=1,j \neq n}^N r q \left( \frac{1 - p}{N} \right) + \sum_{\text{all } z_2(n)} r q p \left( \frac{1 - p}{N} \right) + \sum_{L=3}^{+\infty} \left( \sum_{\text{all } z_L(n)} r q \left( \prod_{j=2,\ldots,L-1} q p \left( \frac{1 - p}{N} \right) \right) \right) \]

\[ = \lambda^+ + r q (N - 1) \left( \sum_{L=0}^{+\infty} \left( \frac{q p (N - 1)}{N} \right)^L \right) \left( \frac{1 - p}{N} \right). \] (5.7)

Then, let us handle \( \Lambda_n^- \):

\[ \Lambda_n^- = \Lambda^- = \lambda^- + \hat{q}_u \hat{w}_u^- + \sum_{j=1,j \neq n}^N r q \frac{p}{N} + \sum_{\text{all } z_2(n)} r \left( \frac{q p}{N} \right)^2 + \sum_{L=3}^{+\infty} \left( \sum_{\text{all } z_L(n)} r q \left( \prod_{j=2,\ldots,L-1} \frac{q p}{N} \right) \right) \]

\[ = \lambda^- + \hat{q}_u \hat{w}_u^- + r q (N - 1) \left( \sum_{L=0}^{+\infty} \left( \frac{q p (N - 1)}{N} \right)^L \right) \frac{p}{N}. \] (5.8)

The detailed deductions of \( \Lambda_n^- \) and \( \Lambda_n^+ \) are given in Appendix C.

Finally, we have

\[ q_n = \frac{\Lambda_n^+}{r_n + \Lambda_n^-} = q = \frac{\Lambda^+ + r q (N - 1) \left( \sum_{L=0}^{+\infty} \left( \frac{q p (N - 1)}{N} \right)^L \right) \left( \frac{1 - p}{N} \right)}{r + \lambda^- + \hat{q}_u \hat{w}_u^- + r q (N - 1) \left( \sum_{L=0}^{+\infty} \left( \frac{q p (N - 1)}{N} \right)^L \right) \frac{p}{N}}. \] (5.9)

Since \( q p (N - 1)/N < 1 \),

\[ \sum_{L=0}^{+\infty} \left( \frac{q p (N - 1)}{N} \right)^L = \lim_{L \to +\infty} \frac{1 - \left( \frac{q p (N - 1)}{N} \right)^L}{1 - \frac{q p (N - 1)}{N}} = \frac{N}{N - q p (N - 1)}. \] (5.10)
We can simplify (5.9) as

$$q = \frac{\lambda^+ + \frac{rq(N-1)(1-p)}{N-qp(N-1)}}{r + \lambda^- + \hat{q}_u \hat{w}_u - \frac{rpq(N-1)}{N-qp(N-1)}},$$

which is a second-degree polynomial in $q$:

$$q^2 p(N-1)(\lambda^- + \hat{q}_u \hat{w}_u) + q(N-1) \left( r(1-p) - \lambda^+ p \right) - qN \left( r + \lambda^- + \hat{q}_u \hat{w}_u - \lambda^+ N \right) = 0.$$  (5.12)

The positive root of (5.12), which is less than one, is computed, denoted by $q(\hat{q}_u \hat{w}_u) = \zeta(\hat{q}_u \hat{w}_u)$, since the value of $q$ that we seek is a probability. For ease of illustration, let $x = \hat{q}_u \hat{w}_u$. Then,

$$q(\hat{q}_u \hat{w}_u) = q(x) = \zeta(x) = \frac{-(\Upsilon - Nx) - \sqrt{(\Upsilon - Nx)^2 - 4p(N-1)(\lambda^- + x)N\lambda^+}}{2p(N-1)(\lambda^- + x)},$$

where $\Upsilon = \lambda^+ p + rp - \lambda^- N - r - \lambda^+ pN - Npr$. The conditions to guarantee the existence of a positive root that is less than 1 are given in the following subsection.
5.3.2 Condition Analyses for Parameter Setting in Dense Nuclei

In the nucleus activation function $q(x) = \zeta(x)$ derived in (5.13), there are five parameters inside a nucleus needed to be determined, which are the number of cells $N$, the probability for repeated firing $p$, the firing rate $r$, the external excitatory input $\lambda^+$ and external inhibitory input $\lambda^-$. First, we analyse the constraints for these parameters so as to set them appropriately (while maintaining the stability of the nuclei). From (5.12), we have

$$q^2(pN\lambda^- + pN\hat{q}_u\hat{w}_u^- - p\lambda^- - p\hat{q}_u\hat{w}_u^-) + q(\lambda^+p + rp - \lambda^-N - r - n\hat{q}_u\hat{w}_u^- - \lambda^+p - Npr) + \lambda^+N = 0.$$  

(5.14)

For illustrative convenience, let $y(q) = aq^2 + bq + c$, where $a = pN\lambda^- + pN\hat{q}_u\hat{w}_u^- - p\lambda^- - p\hat{q}_u\hat{w}_u^-$, $b = \lambda^+p + rp - \lambda^-N - r - N\hat{q}_u\hat{w}_u^- - \lambda^+p - Npr$, $c = \lambda^+N$. Then, let us deduce the conditions needed for $y(q) = 0$ to have two positive roots with one of them being less than 1 as follows:

- If the conditions $N \geq 2$ and $\lambda^- > 0$ holds, then $pN\lambda^- - p\lambda^- > 0$ and $pN\hat{q}_u\hat{w}_u^- - p\hat{q}_u\hat{w}_u^- > 0$. Then, $a > 0$.

- Since $p$ is a probability, then condition $0 \leq p \leq 1$ is required. If $0 \leq p \leq 1$, then $rp - r \leq 0$. In addition, since $\lambda^+ \geq 0$ and $N \geq 2$, we have $\lambda^+p - \lambda^+pN < 0$. Then, $b = (\lambda^+p - \lambda^+pN) + (rp - r) - \lambda^-N - N\hat{q}_u\hat{w}_u^- - Npr < 0$. Then, $-b/2a > 0$.

- If the condition $\lambda^+ > 0$ holds, then we have $y(q = 0) = c = \lambda^+N > 0$. Until here, we can guarantee that there are two positive roots for $y(q) = 0$.

- To guarantee that one of the roots is less than 1, let us analyse $y(q = 1) = a + b + c = (Np - p - N)(\lambda^- + \hat{q}_u\hat{w}_u^- - \lambda^+) + (p - Np - 1)r$. It is clear that $Np - p - N < 0$ and $p - Np - 1 < 0$. If the condition $r > 0$ holds, we have $(p - Np - 1)r < 0$. Then, if $\lambda^- + \hat{q}_u\hat{w}_u^- \geq \lambda^+$ holds, then $y(q = 1) < 0$. Since $\hat{q}_u\hat{w}_u^- \geq 0$ is the input from outside the cluster, we relax $\lambda^- + \hat{q}_u\hat{w}_u^- \geq \lambda^+$ as the condition $\lambda^- \geq \lambda^+$.

Until now, $y(q = 0) > 0$ and $y(q = 1) < 0$, and the second degree parabola $y(q)$ has two positive roots.
roots with one of them being less than 1. It can be concluded that the conditions/constraints for setting the parameters are \( N \geq 2, 0 \leq p \leq 1, \lambda^+ > 0, \lambda^- > 0, r > 0 \) and \( \lambda^- \geq \lambda^+ \).

For a more intuitive illustration of the dense-nuclei activation \( q(x) = \zeta(x) \), in Figure 5.1, we present curves of nucleus (cluster) activation \( \zeta(x) \) versus the cluster input \( x \) under different settings of parameters of \( r, \lambda^+, \lambda^- \) that satisfy their corresponding constraints/conditions, with the parameters \( n = 20 \) and \( p = 0.05 \). From the figures, we can see that the mathematical prediction of dense-nuclei activation \( q(x) = \zeta(x) \) is in the valid range of \([0, 1]\) provided that the conditions \( N \geq 2, 0 \leq p \leq 1, \lambda^+ > 0, \lambda^- > 0, r > 0 \) and \( \lambda^- \geq \lambda^+ \) are satisfied.

### 5.4 Multi-Layer Architectures of Dense Nuclei

We consider a network composed of a multi-layer architecture. Each layer in the network is composed of a finite number of dense nuclei. Within each dense nucleus, the cells communicate with each other in a fully-connected recurrent structure that can use both synapses and direct soma-to-soma interactions, which has been modelled by the Dense RNN described in Sections 5.2 and 5.3. The communication structure between different layers of nuclei is a conventional multi-layer feedforward structure. The cells in the first layer receive excitation signals from external sources, while each cell in each nucleus has an inhibitory projection to the next higher layer. This multi-layer architecture of the Dense RNN (MLDRNN) is shown schematically in
5.4.1 Model Description

In the MLDRNN system given in Figure 5.2, the first (input) layer is made up of $H_1$ cells that receive excitatory spike trains from external sources, resulting in a quasi-linear cell activation $q(x) = \min(x, 1)$ that has been detailed in Chapter 4. The successive $L$ layers ($L \geq 2$) are hidden layers composed of dense nuclei (clusters) that receive inhibitory spike trains from cells in the previous layer, with a resultant activation function $q(x) = \zeta(x)$. Specifically, the $l$th hidden layer has $H_{l+1}$ dense nuclei; and the connecting weight between the $h_{l+1}$th dense nucleus in the $l$th hidden layer and the $h_l$th cell/nucleus in the previous layer is $w_{h_l, h_{l+1}} \geq 0$, with $l = 1, 2, \cdots, L$.

Let us denote the connecting weight matrices between layers of the $L$-hidden-layer MLDRNN by $W_l = [w_{h_l, h_{l+1}}] \in \mathbb{R}^{H_l \times H_{l+1}}$ with $l = 1, 2, \cdots, L$. In addition, let us define a readout matrix as $W_{L+1} \in \mathbb{R}^{H_{L+1} \times H_{L+2}}$, where $H_{L+2}$ is the dimension of the outputs. Then, we have the matrix-form description of the MLDRNN. Given an input matrix $X = [x_{d,h}] \in \mathbb{R}^{D \times H_1}$ with $D$ being the number of instances, a forward pass of $X$ in the MLDRNN can be described as:

$$
\begin{align*}
Q_1 &= \min(X, 1), \\
Q_l &= \zeta(Q_{l-1}W_{l-1}), \text{ for } l = 2, \cdots, L + 1, \\
O &= Q_{L+1}W_{L+1},
\end{align*}
$$

where $Q_1 \in \mathbb{R}^{D \times H_1}$ is the 1st layer output matrix, $Q_l \in \mathbb{R}^{D \times H_l}$ is the $l$th layer output ($l = 2, \cdots, L + 1$) and $O \in \mathbb{R}^{D \times H_{L+2}}$ (or written as $O(X)$) is the final MLDRNN output matrix. The element-wise operation $\min(x, y)$ produces the smaller value between $x$ and $y$. For notation ease, we will use $\zeta(\cdot)$ as a term-by-term function for vectors and matrices.
Algorithm 4 Improved training procedure for the MLDRNN

Get data matrix \(X\) and label matrix \(Y\)

\(X_1 \leftarrow X\)

for \(l = 1, \cdots, L - 1\) do

solve Problem (5.15) for \(W_l\) with input \(X_l\)

\(W_l \leftarrow W_l / \max(X_l W_l) / 10\)

\(X_{l+1} \leftarrow \zeta(X_l W_l)\)

randomly generate \(W_{L+1}\) in range \([0, 1]\)

\(W_{L+1} \leftarrow \text{pinv}(\zeta(X_L W_L)) Y\)

5.4.2 Learning Problem Formulation

For illustration, a dataset with \(D\) instances \(\{(X,Y)\}\) is given, where \(X = [x_{d,h1}] \in \mathbb{R}_{\geq 0}^{D \times H_1}\) is the input-attribute matrix and \(Y = [y_{d,hL+2}] \in \mathbb{R}_{\geq 0}^{D \times H_{L+2}}\) is the desired-output matrix. For the \(d\)th instance \((x_d,y_d)\), the input-attribute vector is denoted by \(x_d = [x_{d,1} x_{d,2} \cdots x_{d,H_1}] \in \mathbb{R}_{\geq 0}^{1 \times H_1}\) and the desired-output vector is denoted by \(y_d = [y_{d,1} y_{d,2} \cdots y_{d,H_{L+2}}] \in \mathbb{R}_{\geq 0}^{1 \times H_{L+2}}\).

Ideally, the learning problem of the dataset \(\{(X,Y)\}\) is that, for any given input \(x_d\), we would like to have a reliable estimate of the desired output \(y_d\), i.e., to discover some function \(f : \mathbb{R}_{\geq 0}^{1 \times H_1} \to \mathbb{R}_{\geq 0}^{1 \times H_{L+2}}\) to best match the input-output mapping from the dataset [GH02].

Specifically, the problem of the MLDRNN to learn the dataset \(\{(X,Y)\}\) is to select appropriate connecting weight matrices \(W_1, \cdots, W_L \geq 0\) and the readout matrix \(W_{L+1}\) such that, given input \(x_d\), the output of the MLDRNN \(O(x_d) : \mathbb{R}_{\geq 0}^{1 \times H_1} \to \mathbb{R}_{\geq 0}^{1 \times H_{L+2}}\) is a meaningful estimate of the desired output \(y_d\).

5.4.3 Efficient Learning Procedure

Our work in [GY16a] proposed a learning procedure for the MLDRNN, which is denoted as the original learning technique in this chapter. Further work in [YGI16b, GY17] improved the learning procedure. The following presents the improved learning procedure for the MLDRNN, which combines unsupervised learning and supervised learning techniques.

First, the unsupervised learning techniques [HS06, TDH16] are utilized and adapted to obtain the connecting weights from the input layer to the \((L - 1)\)th hidden layer. Specifically, the
Algorithm 5 Training procedure for the MCMLDRNN

Get data matrices \( X_c \) (\( c = 1, \cdots, C \)) and label matrix \( Y \)

\[
X_{c,1} \leftarrow X_c \text{ with } c = 1, \cdots, C
\]

for \( l = 1, \cdots, L - 1 \) do

for \( c = 1, \cdots, C \) do

solve Problem (5.16) for \( W_{c,l} \) with input \( X_{c,l} \)

\[
W_{c,l} \leftarrow W_{c,l} / \max(X_{c,l} W_{c,l}) / 10
\]

\[
X_{c,l+1} \leftarrow \zeta(X_{c,l} W_{c,l})
\]

\[
\bar{X} \leftarrow [X_{1,L} \cdots X_{C,L}]
\]

randomly generate \( W_L \) in range \([0 1]\)

\[
W_{L+1} \leftarrow \text{pinv}(\zeta(\bar{X} W_L)) Y
\]

weights \( W_l \) (\( l = 1, \cdots, L - 1 \)) are determined by solving a series of reconstruction problems:

\[
\min_{W_l} \|X_l - \sigma(\zeta(X_l \bar{W})) W_l\|^2 + \|W_l\|_{\ell_1}, \text{ s.t. } W_l \geq 0, \quad (5.15)
\]

where \( \bar{W} \geq 0 \) is a randomly-generated matrix with appropriate dimensions. The operation \( \sigma(\cdot) \) first maps each column of its input \( H \) into \([0 1]\) linearly, then uses the “zcore” MATLAB operation and finally adds a positive constant to remove negativity. For \( W_1 \), the input \( X_1 \) is obtained via \( X_1 \leftarrow X \); while for \( W_l \) with \( l = 2, \cdots, L - 1 \), the input \( X_l \) is obtained via \( X_l \leftarrow \zeta(X_{l-1} W_{l-1}) \). The fast iterative shrinkage-thresholding algorithm (FISTA) in [BT09] is used to solve Problem (5.15) for \( W_l \) with the modification of setting negative elements in the solution to zero in each iteration.

The supervised learning technique is then utilized to determine the connecting weights \( W_L \) and \( W_{L+1} \). Based on the ELM concept, weight matrix \( W_L \) is randomly generated in range \([0 1]\), while \( W_{L+1} \) is determined by the Moore-Penrose pseudo-inverse [GY16a, ZYG+14a, YZ12, ZYY+12, KZH13, TDH16] (denoted by “pinv”). The improved training procedure for the MLDRNN is shown in Algorithm [4] where operation \( \max(\cdot) \) produces the maximal element of its input.

Note that, following the conditions deduced in Subsection 5.3.2 parameters \( n, p, \) and \( r \) for the clusters are selected as \( N = 20, p = 0.05 \) and \( r = 0.001 \). In addition, numerous numerical tests show that 0.01, 0.005 are generally good choices for \( \lambda = \lambda^+ = \lambda^- \).
5.5 MLDRNN for Multi-Channel Datasets

The MLDRNN is adapted to handle multi-channel classification datasets, called the MCML-DRNN, where a C-channel classification dataset is denoted by \( \{(X_1, X_2, \cdots, X_C, Y)\} \) with \( X_c \) being the input data from the \( c \)th channel \((c = 1, 2, \cdots, C)\). For ease of illustration, an example of a 2-channel \( L \)-hidden-layer MCMLDRNN for a 2-channel dataset is shown schematically in Figure 5.3. Let us denote the connecting weights between layers for only Channel-\( c \) by \( W_{c,1}, \cdots, W_{c,L-1} \geq 0 \), those between the \( L - 1 \) and \( L \) hidden layers by \( W_L \geq 0 \) and output weights by \( W_{L+1} \). The weights \( W_{c,l} \geq 0 \) \((c = 1, 2, \cdots, C; l = 1, \cdots, L - 1)\) are determined by solving a series of reconstruction problems using the modified FISTA (described in Section 5.4.3):

\[
\min_{W_{c,l}} ||X_{c,l} - \text{adj}(\zeta(X_{c,l}W))W_{c,l}||^2 + ||W_{c,l}||_{\ell_1}, \text{ s.t. } W_{c,l} \geq 0,
\]

(5.16)

where \( X_{c,l} \) is either the raw data from Channel-\( c \) or its layer encodings. Taking the \( c \)th channel for example, for the weights \( W_{c,1} \), the input \( X_{c,1} \) is obtained via \( X_{c,1} \leftarrow X_1 \); while for \( W_{c,l} \) \((l = 2, \cdots, L - 1)\), \( X_{c,l} \) is obtained via \( X_{c,l} \leftarrow \zeta(X_{c,l-1}W_{l-1}) \). The training procedure of a \( C \)-channel \( L \)-hidden-layer MCMLDRNN is shown in Algorithm 5. The superiority of the MCMLDRNN equipped with Algorithm 5 is demonstrated by results on both the multi-channel...
5.5. MLDRNN for Multi-Channel Datasets

For comparison, a modification is made to the MCMLDRNN, called the MCMLDRNN1, where the schematic representation of a C-channel L-hidden-layer B-branch MCMLDRNN1 is shown in Figure 5.4. Let us denote the connecting weights to the \(l\)th hidden layer for Channel-\(c\) of Branch-\(b\) by \(W_{c,l,b} \geq 0\) (\(c = 1, \ldots, C; l = 1, \ldots, L - 1; b = 1, \ldots, B\)), those for all channels between the \(L - 1\) and \(L\) hidden layers by \(W_L \geq 0\) and output weights by \(W_{L+1}\). The training procedure of the MCMLDRNN1 is detailed in Algorithm 6.

Algorithm 6: Training procedure for the MCMLDRNN1

| Get data matrices \(X_c\) and let \(X_{c,1,b} \leftarrow X_c\) for \(b = 1, \ldots, B\) (\(c = 1, \ldots, C\)), and get label matrix \(Y\); |
| for \(l = 1, \ldots, L - 1\) do |
| for \(c = 1, \ldots, C\) do |
| for \(b = 1, \ldots, B\) do |
| solve a similar reconstruction problem to (5.16) for \(W_{c,l,b}\) with input \(X_{c,l,b}\) |
| \(W_{c,l,b} \leftarrow W_{c,l,b}/\max(X_{c,l,b}W_{c,l,b})/10\) |
| \(X_{c,b} \leftarrow \zeta(X_{c,l,b}W_{c,l,b})\) |
| \(X \leftarrow [X_{c,L,b}]\) for \(b = 1, \ldots, B\) and \(c = 1, \ldots, C\) |
| randomly generate \(W_L\) in range \([0 1]\) |
| \(W_{L+1} \leftarrow \text{pinv}((XW_L))Y\) |

The second modification denoted by MCMLDRNN2 is a simplified MCMLDRNN1, obtained by removing the last hidden layer of the MCMLDRNN1 that produces random mappings via random connections \(W_L\). The schematic representation and training procedure are omitted.
because they are similar to the previous ones of the MCMLDRNN1.

5.6 Numerical Result Comparisons

Numerical tests are conducted with three multi-channel classification datasets: an image dataset and two real-world time-series datasets. In the numerical experiments, we use the MCMLDRNN with Algorithm 5, MCMLDRNN1 with Algorithm 6, MCMLDRNN2, MLDRNN with Algorithm 4, MLDRNN with the algorithm reported in [GY16a], the multi-layer perception (MLP), the convolutional neural network (CNN) and hierarchical extreme learning machine (H-ELM) [TDH16]. The experiments are run in a personal computer with Intel i7-6700K CPU (4.00GHz), 16GB memory and GeForce GTX 1080 GPU. The MCMLDRNN, MCMLDRNN1, MCMLDRNN2, MLDRNN and H-ELM are implemented using MATLAB with only CPU. The MLP and CNN are implemented using Keras [Cho15] with either Theano [The16] or Tensorflow [AAB+15] as the backend: when the backend is Theano, only CPU is used; while Keras with Tensorflow uses both CPU and GPU.

5.6.1 Application 1: Recognizing 3D Objects

NORB Dataset

The small NORB dataset [LHB04] is intended for experiments in 3D object recognition from shape. The objects belong to 5 generic categories: four-legged animals, human figures, airplanes, trucks and cars. The instance numbers for both training and testing are 24300. There are two $96 \times 96$ images in each instance which are downsampled into $32 \times 32$. Examples of the two images of each class are given in Figure 5.5. All images are whitened using the code provided by [TDH16]. The objective is to recognize the generic category of a given instance using the corresponding two images. The number of channels is 2.
5.6. Numerical Result Comparisons

Figure 5.5: Examples of two images of each class in the NORB dataset: four-legged animals, human figures, airplanes, trucks and cars.

Experiment Settings

The structures for the MLDRNN, MCMLDRNN, MCMLDRNN1, MRCMLDRNN2, H-ELM and MLP (Tanh activation) are respectively 2048-500-2000-5, 2x1024-2x500-2000-5, 2x1024-80x2x100-10x160x10-8000-5, 2x1024-2x2x500-2x4x500-5, 2048-1000-1000-12000-5 and 2048-500-500-5. The CNN with Tanh activation is constituted by two convolution layers, a pooling layer, a fully-connected layer and an output layer (denoted as “input-conv-conv-pool-fc-output”) and each convolution layer has 30 filters with kernel size 2x2. The dropout technique developed to prevent from overfitting [SHK+14] is exploited for the MLP and CNN. The Adadelta optimizer [Zei12] is used to train the MLP and CNN; the training method of the H-ELM is specified in [TDH16]. For each method, five trials are conducted. For the MLP and CNN, there are 50 epochs in each trial and both Theano and Tensorflow are used; in the cases of early stopping, the number of epochs is 4.

Experiment Results

The results of different methods on the training and testing accuracies and the training and testing time are summarised in Table 5.1. The proposed MCMLDRNN obtains the second highest testing accuracy (91.19%), which is very close to the one reported in [TDH16]. With the
Table 5.1: Training and testing accuracies (%) and time (s) of different methods on NORB dataset for recognizing 3D objects.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracies (%)</th>
<th>Times (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training</td>
<td>Testing</td>
</tr>
<tr>
<td>MLP</td>
<td>89.43 ±10.49</td>
<td>75.70 ±9.07</td>
</tr>
<tr>
<td>MLP+dropout</td>
<td>82.19 ±17.54</td>
<td>64.61 ±19.08</td>
</tr>
<tr>
<td>CNN</td>
<td>100.00 ±0</td>
<td>89.59 ±0.24</td>
</tr>
<tr>
<td>CNN+dropout</td>
<td>100.00 ±0</td>
<td>90.61 ±2.06</td>
</tr>
<tr>
<td>MLP*</td>
<td>99.40 ±0.52</td>
<td>83.29 ±3.24</td>
</tr>
<tr>
<td>MLP+dropout*</td>
<td>99.41 ±0.38</td>
<td>83.28 ±1.08</td>
</tr>
<tr>
<td>CNN*</td>
<td>100.00 ±0</td>
<td>90.31 ±0.35</td>
</tr>
<tr>
<td>CNN+dropout*</td>
<td>100.00 ±0</td>
<td>91.09 ±0.49</td>
</tr>
<tr>
<td>CNN†</td>
<td>99.55 ±0.39</td>
<td>89.07 ±1.57</td>
</tr>
<tr>
<td>CNN+dropout†</td>
<td>99.64 ±0.22</td>
<td>88.18 ±1.15</td>
</tr>
<tr>
<td>H-ELM</td>
<td>99.80 ±0.10</td>
<td>87.36 ±0.85</td>
</tr>
<tr>
<td>Original MLDRNN</td>
<td>99.94 ±0.03</td>
<td>87.11 ±0.44</td>
</tr>
<tr>
<td>Improved MLDRNN</td>
<td>99.99 ±0</td>
<td>90.46 ±0.60</td>
</tr>
<tr>
<td>MCMLDRNN</td>
<td>99.99 ±0.01</td>
<td>91.19 ±0.63</td>
</tr>
<tr>
<td>MCMLDRNN1</td>
<td>99.82 ±0.06</td>
<td>90.64 ±0.58</td>
</tr>
<tr>
<td>MCMLDRNN2</td>
<td>99.51 ±0.11</td>
<td>90.83 ±0.34</td>
</tr>
<tr>
<td>H-ELM*</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

*These experiments are conducted using the GPU.
†Early-stopping is used to reduce over-fitting.
*This data is obtained directly from [TDH16].

aid of the dropout technique, the CNN achieves a comparable testing accuracy (91.09%) with the MCMLDRNN. However, the computational complexity is much higher. Specifically, using only the CPU, the proposed MCMLDRNN can be trained and applied (tested) respectively around 100 and 10 times faster than the CNN. Aided by the computational power of the GPU, the training of the CNN is still more than 15 times slower than that of the MCMLDRNN. The testing accuracy of the improved MLDRNN is slightly lower than that of the MCMLDRNN, as are the training and testing times. Therefore, both the improved MLDRNN and MCMLDRNN developed in this chapter are good classifiers for the NORB dataset with high accuracy and training efficiency.
5.6. Numerical Result Comparisons

5.6.2 Application 2: Distinguishing Chemical Gases

Twin Gas Sensor Arrays (TGSA) Dataset

The TGSA dataset includes 640 recordings of 5 twin 8-sensor detection units (Units 1 to 5) being exposed to 4 different gases (Carbon Monoxide (CO), Ethanol, Ethylene and Methane) \cite{FFGG+16}. The duration of each recording is 600 seconds (100Hz sampling frequency) producing 480,000 (8x600x100) features. We use 30-second segments, such that each instance has 24,000 (8x3000) attributes. Examples of these 30-second segments of each class are given in Figure 5.6. The objective is to distinguish the gas types using the 30-second segments. The number of channels is 8. The following tasks are considered, in which two thirds of instances are used for training while the rest for testing:

- Task 1 (3,029 instances): build a specific classifier for Unit 1 to fulfill the objective.

- Task 2 (12,089 instances): build one classifier for all units to fulfill the objective.

- Tasks 3 to 6: build specific classifiers for Units 2 to 5 to fulfill the objective.
Chapter 5. Deep Learning with Dense Random Neural Network

Figure 5.7: Examples of 45 5-seconds segments of each class in the DSA dataset: sitting, standing, lying on back, lying on right side, ascending stairs (A1-5); descending stairs, standing in an elevator still, moving around in an elevator, walking in a parking lot, walking on a treadmill with a speed of 4 km/h in flat position (A6-10); walking on a treadmill with a speed of 4 km/h in 15 deg inclined position, running on a treadmill with a speed of 8 km/h, exercising on a stepper, exercising on a cross trainer, cycling on an exercise bike in horizontal position (A11-15); cycling on an exercise bike in vertical position, rowing, jumping, playing basketball (A16-19).

Motivations

Due to the inherent variability of chemical gas sensors, a classification model built on a specific sensor-array system may not be applicable to another system [FFGG16, MGG12]. For example, a classifier built for Unit 1 cannot be used for Units 2 to 5. Therefore, a fast and robust method to build accurate classifiers for sensor-array systems is required.

Experiment Settings

The structure settings of different methods to handle Tasks 1 and 2 of the TGSA dataset are given in Table 5.2. To fit the time-series data into the CNN, we reshape it from 8x3000 to
Table 5.2: Structure settings of different methods to handle the TGSA dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Input layer</th>
<th>Hidden layers Task 1</th>
<th>Hidden layers Task 2</th>
<th>Output layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>24000</td>
<td>500-500</td>
<td>500-500</td>
<td>4</td>
</tr>
<tr>
<td>CNN</td>
<td>8x50x60</td>
<td>conv-conv-pool-fc</td>
<td>conv-conv-pool-fc</td>
<td>4</td>
</tr>
<tr>
<td>H-ELM</td>
<td>24000</td>
<td>500-500-8000</td>
<td>1000-1000-12000</td>
<td>4</td>
</tr>
<tr>
<td>Ori. DenseRNN</td>
<td>24000</td>
<td>100-2000</td>
<td>50-5000</td>
<td>4</td>
</tr>
<tr>
<td>Improved DenseRNN</td>
<td>24000</td>
<td>100-3000</td>
<td>50-5000</td>
<td>4</td>
</tr>
<tr>
<td>MCMLDRNN</td>
<td>8x3000</td>
<td>8x200-8x100-2000</td>
<td>8x300-8x50-5000</td>
<td>4</td>
</tr>
<tr>
<td>MCMLDRNN1</td>
<td>8x3000</td>
<td>2x8x200-2x16x100-2000</td>
<td>2x8x200-2x16x100-5000</td>
<td>4</td>
</tr>
<tr>
<td>MCMLDRNN2</td>
<td>8x3000</td>
<td>2x8x200-2x16x100</td>
<td>2x8x200-2x16x100</td>
<td>4</td>
</tr>
</tbody>
</table>

8x50x60. The training methods for these networks are the same as those used in Section 5.6.1. When Task 1 is handled, the trial number is 5 for the MLP and CNN (80 epoches in each trial). The activation function of them is the Tanh function. For the rest networks, the trial number is 20. When Task 2 is handled, 5 trials are conducted for all methods, except the cases of the MLP and CNN trained by only the CPU, in which the training is slow and one trial is conducted. The settings for Tasks 3 to 6 are the same as the ones for Task 1.

**Experiment Results**

Table 5.3 presents the results of different methods to handle Task 1. The accuracies of the CNN and MLP are not satisfactory and the training time is long. The MCMLDRNN is the most accurate classifier for Unit 1 (98.27% testing accuracy). Its training and testing times are lower than the MCMLDRNN1, which achieves 98.12%. The training times of the H-ELM and the MLDRNN are low, but the testing accuracies are not satisfactory. Taking both accuracy and computational complexity into account, the MCMLDRNN is the best choice for Task 1.

For Task 2, with its results given in Table 5.4, the highest testing accuracy is 93.66%, achieved by the MCMLDRNN. For the H-ELM, adding more neurons into the last hidden layer may increase its accuracy to handle Task 2. However, its testing time increases as well, where the current one is 2.60s, which is already higher than the one of the MCMLDRNN (2.29s). Larger network sizes for the CNN may also help it to achieve higher accuracies for Tasks 1 and
Table 5.3: Training and testing accuracies (%) and time (s) of different methods on Task 1 of TGSA dataset for distinguishing chemical gases.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracies (%) Training</th>
<th>Accuracies (%) Testing</th>
<th>Times (s) Training</th>
<th>Times (s) Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>24.96 ± 0.10</td>
<td>24.95 ± 0.10</td>
<td>569.39 ± 10.67</td>
<td>0.29 ± 0.01</td>
</tr>
<tr>
<td>MLP+dropout</td>
<td>24.96 ± 0.10</td>
<td>24.95 ± 0.10</td>
<td>571.67 ± 8.34</td>
<td>0.32 ± 0.01</td>
</tr>
<tr>
<td>CNN</td>
<td>47.57 ± 22.51</td>
<td>47.97 ± 22.92</td>
<td>5428.14 ± 125.44</td>
<td>6.90 ± 0.17</td>
</tr>
<tr>
<td>CNN+dropout</td>
<td>52.45 ± 27.59</td>
<td>51.98 ± 27.13</td>
<td>5604.41 ± 122.86</td>
<td>7.36 ± 0.17</td>
</tr>
<tr>
<td>MLP*</td>
<td>24.96 ± 0.10</td>
<td>24.95 ± 0.10</td>
<td>81.96 ± 0.64</td>
<td>0.15 ± 0.00</td>
</tr>
<tr>
<td>MLP+dropout*</td>
<td>24.96 ± 0.10</td>
<td>24.95 ± 0.10</td>
<td>81.37 ± 0.24</td>
<td>0.16 ± 0.00</td>
</tr>
<tr>
<td>CNN*</td>
<td>42.79 ± 17.93</td>
<td>42.38 ± 17.52</td>
<td>98.80 ± 0.33</td>
<td>0.22 ± 0.01</td>
</tr>
<tr>
<td>CNN+dropout*</td>
<td>71.5 ± 5.87</td>
<td>71.63 ± 6.78</td>
<td>101.25 ± 0.18</td>
<td>0.24 ± 0.02</td>
</tr>
<tr>
<td>H-ELM</td>
<td>62.51 ± 3.27</td>
<td>61.68 ± 3.66</td>
<td>10.21 ± 0.75</td>
<td>0.59 ± 0.08</td>
</tr>
<tr>
<td>Original MLDRNN</td>
<td>99.85 ± 0.15</td>
<td>92.72 ± 1.63</td>
<td>4.70 ± 0.12</td>
<td>0.22 ± 0.01</td>
</tr>
<tr>
<td>Improved MLDRNN</td>
<td>99.73 ± 0.27</td>
<td>96.78 ± 1.73</td>
<td>6.41 ± 0.17</td>
<td>0.28 ± 0.01</td>
</tr>
<tr>
<td>MCMLDRNN</td>
<td>99.85 ± 0.10</td>
<td>98.27 ± 0.74</td>
<td>9.27 ± 0.43</td>
<td>0.37 ± 0.03</td>
</tr>
<tr>
<td>MCMLDRNN1</td>
<td>99.78 ± 0.12</td>
<td>98.12 ± 0.40</td>
<td>17.73 ± 1.33</td>
<td>0.70 ± 0.08</td>
</tr>
<tr>
<td>MCMLDRNN2</td>
<td>97.72 ± 0.35</td>
<td>95.30 ± 0.64</td>
<td>16.73 ± 0.4</td>
<td>0.54 ± 0.09</td>
</tr>
</tbody>
</table>

*These experiments are conducted using the GPU.

2. However, the current accuracy gap between the CNN and MCMLDRNN is large and the testing time (using only the CPU) of the CNN is more than ten times larger than that of the MCMLDRNN.

The confusion matrices of the MCMLDRNN to handle Tasks 1 and 2 in the testing phase are given in Tables 5.5 and 5.6, respectively. For Task 1, Methane has the highest misclassification rate; while, for Task 2, it is Ethylene.

The MCMLDRNN is exploited to handle Tasks 3 to 5, with results given in Table 5.7. For all units, the proposed MCMLDRNN produces high classification rates (from 94.5% to as high as 98.99%) and the training is highly efficient, which can be completed in less than 10 seconds.

Therefore, it can be concluded from these results that the proposed MCMLDRNN is the most efficient method among the compared ones to distinguish chemical gases based on the TGSA dataset.
Table 5.4: Training and testing accuracies (%) and time (s) of different methods on Task 2 of TGSA dataset for distinguishing chemical gases.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracies (%)</th>
<th>Times (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training</td>
<td>Testing</td>
</tr>
<tr>
<td>MLP</td>
<td>25.04</td>
<td>25.06</td>
</tr>
<tr>
<td></td>
<td>3098.65</td>
<td>1.42</td>
</tr>
<tr>
<td>MLP+dropout</td>
<td>24.86</td>
<td>24.83</td>
</tr>
<tr>
<td></td>
<td>2741.69</td>
<td>1.41</td>
</tr>
<tr>
<td>CNN</td>
<td>69.29</td>
<td>68.69</td>
</tr>
<tr>
<td></td>
<td>21577.89</td>
<td>28.16</td>
</tr>
<tr>
<td>CNN+dropout</td>
<td>25.04</td>
<td>25.06</td>
</tr>
<tr>
<td></td>
<td>23130.27</td>
<td>29.38</td>
</tr>
<tr>
<td>MLP*</td>
<td>24.98 ±0.12</td>
<td>24.96 ±0.12</td>
</tr>
<tr>
<td></td>
<td>322.76 ±0.55</td>
<td>0.64 ±0.02</td>
</tr>
<tr>
<td>MLP+dropout*</td>
<td>25.02 ±0.02</td>
<td>25.04 ±0.01</td>
</tr>
<tr>
<td></td>
<td>326.62 ±1.37</td>
<td>0.65 ±0.01</td>
</tr>
<tr>
<td>CNN*</td>
<td>55.47 ±30.37</td>
<td>55.16 ±30.08</td>
</tr>
<tr>
<td></td>
<td>400.17 ±4.03</td>
<td>0.88 ±0.03</td>
</tr>
<tr>
<td>CNN+dropout*</td>
<td>77.13 ±12.70</td>
<td>75.69 ±12.97</td>
</tr>
<tr>
<td></td>
<td>417.62 ±11.77</td>
<td>1.06 ±0.11</td>
</tr>
<tr>
<td>H-ELM</td>
<td>62.02 ±2.38</td>
<td>61.90 ±2.60</td>
</tr>
<tr>
<td></td>
<td>33.98 ±0.66</td>
<td>2.60 ±0.19</td>
</tr>
<tr>
<td>Original MLDRNN</td>
<td>78.97 ±1.48</td>
<td>26.52 ±2.80</td>
</tr>
<tr>
<td></td>
<td>48.86 ±0.32</td>
<td>1.60 ±0.08</td>
</tr>
<tr>
<td>Improved MLDRNN</td>
<td>88.07 ±0.53</td>
<td>87.63 ±0.83</td>
</tr>
<tr>
<td></td>
<td>48.36 ±0.10</td>
<td>1.67 ±0.09</td>
</tr>
<tr>
<td>MCMLDRNN</td>
<td>93.60 ±0.32</td>
<td>93.66 ±0.48</td>
</tr>
<tr>
<td></td>
<td>53.59 ±1.44</td>
<td>2.29 ±0.16</td>
</tr>
<tr>
<td>MCMLDRNN1</td>
<td>90.97 ±0.18</td>
<td>90.46 ±0.51</td>
</tr>
<tr>
<td></td>
<td>62.18 ±0.74</td>
<td>3.90 ±0.18</td>
</tr>
<tr>
<td>MCMLDRNN2</td>
<td>79.83 ±0.23</td>
<td>79.21 ±0.27</td>
</tr>
<tr>
<td></td>
<td>32.94 ±1.15</td>
<td>1.96 ±0.20</td>
</tr>
</tbody>
</table>

*These experiments are conducted using the GPU.

Table 5.5: Confusion matrix of MCMLDRNN to classify the testing subdataset in Task 1 of TGSA dataset for distinguishing chemical gases.

<table>
<thead>
<tr>
<th>True Class</th>
<th>Classified Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CO</td>
</tr>
<tr>
<td>CO</td>
<td>251</td>
</tr>
<tr>
<td>Ethanol</td>
<td>0</td>
</tr>
<tr>
<td>Ethylene</td>
<td>0</td>
</tr>
<tr>
<td>Methane</td>
<td>6</td>
</tr>
</tbody>
</table>

5.6.3 Application 3: Detecting Human Activities

Daily and Sports Activities (DSA) Dataset

The DSA dataset \[ABT10\] \[BY14\] \[AB10\] comprises time-series data of 19 daily and sports activities performed by 8 subjects (humans) recorded by 45 motion sensors (25 Hz sampling frequency) worn on five different body locations (torso, right arm, left arm, right leg and left leg). The attribute number is 5,625 \((45 \times 5 \times 25)\) since 5-second segments are used, while the class number is 19. Examples of these 5-second segments of each class are given in Figure 5.7. Two
Table 5.6: Confusion matrix of MCMLDRNN to classify the testing subdataset in Task 2 of TGSA dataset for distinguishing chemical gases.

<table>
<thead>
<tr>
<th>True</th>
<th>Classified</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CO</td>
</tr>
<tr>
<td>CO</td>
<td>965</td>
</tr>
<tr>
<td>Ethanol</td>
<td>20</td>
</tr>
<tr>
<td>Ethylene</td>
<td>76</td>
</tr>
<tr>
<td>Methane</td>
<td>26</td>
</tr>
</tbody>
</table>

Table 5.7: Training and testing accuracies (%) and time (s) of MCMLDRNN on Tasks 3 to 6 (Units 2 to 5) of TGSA dataset for distinguishing chemical gases.

<table>
<thead>
<tr>
<th>Unit</th>
<th>Accuracies (%)</th>
<th>Times (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training</td>
<td>Testing</td>
</tr>
<tr>
<td>2</td>
<td>99.36 ± 0.25</td>
<td>97.28 ± 0.94</td>
</tr>
<tr>
<td>3</td>
<td>99.85 ± 0.10</td>
<td>96.77 ± 0.84</td>
</tr>
<tr>
<td>4</td>
<td>99.85 ± 0.05</td>
<td>94.87 ± 1.38</td>
</tr>
<tr>
<td>5</td>
<td>100.00 ± 0</td>
<td>98.99 ± 0.80</td>
</tr>
</tbody>
</table>

thirds of 9120 instances are used for training, while the rest are used for testing. The objective is to detect human activities using the 5-second segments. The number of channels is 45.

Motivations and Related Work

Effective human-activity detection classifiers are of interest due to their potential applications in ambient-assisted living [vKK07, SCSE10], smart hospital [STF08], sports and abnormal-activity detection [YYP08, HZY+09], which can be either vision-based or sensor-based. We focus on the sensor-based classifiers in this section. Based on body-worn sensors, [ABT10] collected time-series data (the DSA dataset) from humans. In total, 1170 features are extracted from the raw time-series data, e.g., the mean, minimum and maximum values and frequency values after the discrete Fourier transform, which are then reduced into 30 features through the principal component analysis (PCA). Different types of methods are then exploited to build classifiers based on the 30 features. The work in [BY14] on the DSA dataset also utilized the same data pre-processing procedure (30 features). The recognition architecture developed in [LPLH15] extracted the statistical, frequency, structure and coordinate
features etc from the raw data. A recent work [PMPMV16] on the DSA dataset extracted 810 features in the time and frequency domains by referring to various literature, where the feature number was reduced using the PCA.

In contrast, this work investigates the feasibility in building effective classifiers directly on the raw time-series data using multi-layer neural networks, where useful features are automatically extracted in the training of the networks rather than hand-picked.

**Experiment Settings**

The hidden-layer structure settings for different networks are the same as the ones for Task 1 of the TGSA dataset illustrated in Table 5.2. The main differences lays on the input and output layers, e.g., the structure of the MCMLDRNN, MCMLDRNN1 and MCMLDRNN2 are repetitively 45x125-2x45x200-2x90x100-2000-19 and 45x125-2x45x200-2x90x100-19. To facilitate the training, the activation functions of the MLP and CNN are selected as the ReLU function [GBB11]. The input data is reshaped from 45x125 to 45x5x25 to fit the CNN. For each method, 20 trials are conducted. The epoch number is 80 for the MLP and CNN. The trial number is 10 in the cases of the MLP and CNN trained with only the CPU. Note that the human-activity detection problem here is considered as a pattern-classification problem solved by feed-forward neural networks. However, considering that the dataset is also a time-series dataset, a recurrent network that is capable of extracting time dependencies from the data may also be adapted for solving the problem in the further work.

**Experiment Results**

The results on the training and testing accuracies and the training and testing time are summarised in Table 5.8. It is worth pointing out here that the differences between the training times of the MCMLDRNN1 and MCMLDRNN2 may show opposite directions in Tables 5.3 and 5.8 as their training times also depend on the channel numbers of the datasets and the neuron numbers of the last hidden layer. It can be seen that most methods achieve high activity-
Table 5.8: Training and testing accuracies (%) and time (s) of different methods on DSA dataset for detecting human activities.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracies (%)</th>
<th>Times (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training</td>
<td>Testing</td>
</tr>
<tr>
<td>MLP</td>
<td>99.61 ±0.35</td>
<td>97.19 ±0.58</td>
</tr>
<tr>
<td>MLP+dropout</td>
<td>99.56 ±0.37</td>
<td>97.47 ±0.92</td>
</tr>
<tr>
<td>CNN</td>
<td>99.84 ±0.13</td>
<td>98.50 ±0.38</td>
</tr>
<tr>
<td>CNN+dropout</td>
<td>99.84 ±0.12</td>
<td>99.11 ±0.20</td>
</tr>
<tr>
<td>MLP*</td>
<td>99.00 ±0.95</td>
<td>96.68 ±1.32</td>
</tr>
<tr>
<td>MLP+dropout*</td>
<td>99.69 ±0.26</td>
<td>97.62 ±0.67</td>
</tr>
<tr>
<td>CNN*</td>
<td>99.78 ±0.21</td>
<td>98.54 ±0.44</td>
</tr>
<tr>
<td>CNN+dropout*</td>
<td>99.87 ±0.1</td>
<td>98.96 ±0.25</td>
</tr>
<tr>
<td>H-ELM</td>
<td>98.08 ±0.65</td>
<td>96.83 ±0.61</td>
</tr>
<tr>
<td>Original MLDRNN</td>
<td>98.41 ±0.60</td>
<td>91.43 ±1.60</td>
</tr>
<tr>
<td>Improved MLDRNN</td>
<td>94.68 ±0.65</td>
<td>91.74 ±0.46</td>
</tr>
<tr>
<td>MCMLDRNN</td>
<td>99.79 ±0.03</td>
<td>99.13 ±0.18</td>
</tr>
<tr>
<td>MCMLDRNN1</td>
<td>99.59 ±0.05</td>
<td>98.88 ±0.13</td>
</tr>
<tr>
<td>MCMLDRNN2</td>
<td>98.51 ±0.14</td>
<td>94.41 ±0.39</td>
</tr>
</tbody>
</table>

*These experiments are conducted using the GPU.
† The data is 65.31±7.17(s) if the experiment with 900.28s (unknown error) is removed.

detection accuracies and the CNN, MCMLDRNN and MCMLDRNN1 are capable of achieving more than 99% accuracy. This demonstrates the effectiveness of using raw time-series data directly for detecting human activities. The MCMLDRNN obtains the best testing accuracy, as high as 99.13%. The MCMLDRNN can be trained efficiently and is 50 times faster than the CNN, when both methods use only the CPU. Moreover, the MCMLDRNN can still be trained three times faster even when the CNN utilizes both the CPU and GPU.

The confusion matrix of the MCMLDRNN to classify the testing sub-dataset is given in Table 5.9. As noted in [ABT10], with six confusion matrices given, A7 (standing in an elevator still) and A8 (moving around in an elevator) are the most confusing activities with each other, since they are similar. In contrast, as shown in Table 5.9, the MCMLDRNN may misclassify A8 as A7, however it does not misclassify A7 as A8.

In conclusion, these results (Tables 5.8 and 5.9) have well demonstrated the superiority of the MCMLDRNN for detecting human activities.
An Interesting Case Where the Number of Cells in a Nucleus is Sufficiently Large

In Sections 5.4.3 and 5.5, the number of cells in a nucleus, i.e., the parameter $N$ in (5.13), is set as 20, which is a reasonable value considering the fact that the MLDRNN and MCMLDRNN are not just algorithms but they also mathematical models of a physical system (more precisely, a biological “integrate and fire” system).

In this section, we investigate the case where the number of cells in a nucleus is very large. We find that the results are interesting since this case leads to a simpler mathematical form of nuclei activation despite the fact that the corresponding physical system is much more complicated.
Table 5.9: Confusion matrix of MCMLDRNN to classify the DSA testing subdataset for detecting human activities.

| True Class | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 | A17 | A18 | A19 |
|------------|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| A1         | 159| 0  | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| A2         | 0  | 160| 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| A3         | 0  | 0  | 160| 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| A4         | 0  | 0  | 0  | 160| 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| A5         | 0  | 0  | 0  | 0  | 160| 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| A6         | 0  | 0  | 0  | 0  | 0  | 160| 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| A7         | 0  | 0  | 0  | 0  | 0  | 0  | 160| 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| A8         | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 143| 13 | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| A9         | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 160| 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| A10        | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 160| 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| A11        | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 159| 0   | 0   | 0   | 0   | 0   | 0   |
| A12        | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 160| 0   | 0   | 0   | 0   | 0   |
| A13        | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 160| 0   | 0   | 0   | 0   |
| A14        | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0  | 160| 0   | 0   | 0   |
| A15        | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0  | 0  | 160| 0   | 0   |
| A16        | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 160| 0   |
| A17        | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 160|
| A18        | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| A19        | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 156|

Let us recall \( \hat{q}_u \hat{w}_u \) substituted by \( x \):

\[
q^2 p(N - 1)(\lambda^- + x) + q(N - 1) (r(1 - p) - \lambda^+ p) - qN(r + \lambda^- + x) + \lambda^+ N = 0. \tag{5.17}
\]

When the cell number in a nucleus, i.e., \( N \), is sufficiently large (i.e., \( N >> 1 \)), the expression \( (5.17) \) simplifies to:

\[
q^2 p(\lambda^- + x) + q (r(1 - p) - \lambda^+ p) - q(r + \lambda^- + x) + \lambda^+ = 0.
\]

Further,

\[
q^2 p(\lambda^- + x) + q(-rp - \lambda^+ p - \lambda^- - x) + \lambda^+ = 0. \tag{5.18}
\]

The positive root of \( (5.18) \) which is less than one is computed:

\[
q(x) = \zeta(x) = \frac{rp + \lambda^+ p + \lambda^- + x}{2p(\lambda^- + x)} - \frac{\sqrt{(rp + \lambda^+ p + \lambda^- + x)^2 - 4p[\lambda^- + x] \lambda^+}}{2p(\lambda^- + x)}. \tag{5.19}
\]
Table 5.10: Results by using dense nuclei (5.13) and very dense nuclei (5.20) in 100 trials to classify DSA dataset.

<table>
<thead>
<tr>
<th></th>
<th>Nuclei (5.13)</th>
<th>Nuclei (5.20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training accuracy (%)</td>
<td>99.79±0.05</td>
<td>99.78±0.04</td>
</tr>
<tr>
<td>Testing accuracy (%)</td>
<td>99.13±0.18</td>
<td>99.16±0.15</td>
</tr>
<tr>
<td>Training time (s)</td>
<td>21.38±0.45</td>
<td>13.10±0.35</td>
</tr>
<tr>
<td>Testing time (s)</td>
<td>3.39±0.15</td>
<td>1.14±0.08</td>
</tr>
</tbody>
</table>

We can simplify (5.19) as

\[ \zeta(x) = e - \sqrt{e^2 - f}, \quad (5.20) \]

where \( e = (r + \lambda^+)/[2(\lambda^- + x)] + 1/(2p) \) and \( f = \lambda^+/[p(\lambda^- + x)] \).

Evidently, this case, where a nucleus has a sufficiently large number of cells (i.e., is sufficiently dense), produces a much simpler activation function (5.20) than (5.13) in the sense of computation. In the following, numerical experiments are conducted to compare them, mainly on the computational time. For both cases, the MCMLDRNN architecture with structure 45x126-45x200-45x100-2000-19 are used to classify the DSA dataset, where parameters \( p = 0.05, r = 0.001, \lambda = \lambda^+ = \lambda^- = 0.005 \). For the case of (5.13), \( N = 20 \). For each case, 100 trials are conducted. The results on the training accuracy and time, and the testing accuracy and time, are comparatively given in Figure 5.8. It can be seen from Figures 5.8(a) and 5.8(b) that the training and testing accuracies by using dense nuclei (5.13) with \( N = 20 \) and dense nuclei (5.20) are very close, around 99.00%. Figures 5.8(c) and 5.8(d) demonstrate that both the training and testing times are significantly reduced by using very dense nuclei (5.20). We also summarize the results of these 100 trials in Table 5.10, from which we can see that, by using (5.20), the training and testing times decrease from around 21.38s and 3.39s to 13.10s and 1.14s, respectively.

These results show that a sufficiently-dense nucleus produces a more efficient algorithm than a “not-so-dense” nucleus in the sense of mathematical form and computation.
5.8 Conclusions

In this chapter, based on the RNN theory, we have explored the idea that the human brain contains important areas composed of dense clusters of cells, such as various nuclei. A mathematical model of the dense random neural network (Dense RNN) has been developed for networks with both synapses and direct soma-to-soma interactions. Aided by the Dense RNN, a mathematical model of dense nuclei that contains statistically identical connected cells has been developed, with the parameter conditions within dense nuclei deduced. Neural networks with multi-layer architectures, where each layer is composed of a finite number of dense nuclei, have then been constructed and formulated for deep learning, where efficient training procedures have been designed.

Comparative numerical experiments have been conducted based on both images and real-world data for multi-channel datasets, compared with several state-of-the-art deep-learning methods. The numerical results show that the proposed deep learning tool based on the Dense RNN provides useful improvements as compared to other methods, in terms of both classification accuracy and training efficiency.
Chapter 6

A Theoretical View into Deep Random Neural Network

This chapter studies the multi-layer RNN (i.e., the MLRNN) for deep learning from the theoretical and statistical points of view. The organization is as follows. Section I presents the mathematical formulation of the deep-learning problem of the MLRNN. Section II presents the analysis. Section III presents the theoretical results based on the analysis in Section II.

6.1 Deep Learning Problem in Multi-Layer RNN

Note that the multi-layer RNN has a feed-forward architecture of \( L \geq 3 \) layers.

The formulation begins with a given input-output pair \( \{(X, Y)\} \), where \( X = [x_1 x_2 \cdots x_{N_1}] \in \mathbb{R}^{2 \times N_1} \) with \( x_1 = [\lambda_{1,1}^+, \lambda_{1,1}^-] \), \( x_2 = [\lambda_{2,1}^+, \lambda_{2,1}^-] \), \( \cdots \) and \( x_{N_1} = [\lambda_{N_1,1}^+, \lambda_{N_1,1}^-] \). The values in \( X \) and \( Y \) are nonnegative real numbers, i.e., \( X \in \mathbb{R}^{2 \times N_1} \) and \( Y = [y_1 y_2 \cdots y_{N_L}] \in \mathbb{R}^{N_L} \). The mapping from \( X \) to \( Y \) is \( f : \mathbb{R}^{2 \times N_1} \rightarrow \mathbb{R}^{N_L} \) and \( Y = f(X) \).

Inputs \( X = [x_1 x_2 \cdots x_{N_1}] \) are given as the pairs of excitatory and inhibitory spike flow rates entering the neurons of the first layer (or say, the input layer) of the MLRNN. The probability that any neuron \( i \) located in the first layer is excited in the steady state, is given by the following
expression based on the RNN theory \cite{Gel89, Gel90, Gel93}:

\[ q_{i,1} = \frac{\lambda_{i,1}^+}{r_{i,1} + \lambda_{i,1}^-} \]  \hspace{1cm} (6.1)

for \( i = 1, \ldots, N_1 \), where \( w_{i,j,1}^+ = r_{i,1}p_{i,j,1}^+ \), \( w_{i,j,1}^- = r_{i,1}p_{i,j,1}^- \), \( p_{j,i,1}^+ \), \( p_{j,i,1}^- \) are the probabilities that neuron \( j \) may send excitatory or inhibitory spikes to neuron \( i \), \( r_{i,1} \) is the firing rate of neuron \( i \) in the first layer and \( \lambda_{i,1}^+ \), \( \lambda_{i,1}^- \) are the external arrival rates of excitatory and inhibitory spikes to neuron \( i \) in the first layer.

In succession, the probability that, in the steady state, any neuron \( i \), located in the \( l + 1 \) layer, is excited, is given by the expression \cite{Gel89, Gel90, Gel93}:

\[ q_{i,l+1} = \frac{\lambda_{i,l+1}^+ + \sum_{j=1}^{N_l} q_{j,l}w_{j,i,l}^+}{r_{i,l+1} + \lambda_{i,l+1}^- + \sum_{j=1}^{N_l} q_{j,l}w_{j,i,l}^-} \]  \hspace{1cm} (6.2)

for \( i = 1, \ldots, N_{l+1} \), where \( r_{i,l+1} \) is the firing rate of neuron \( i \) in the \( l \)th layer and \( \lambda_{i,l+1}^+ \), \( \lambda_{i,l+1}^- \) are the external arrival rates of excitatory and inhibitory spikes to neuron \( i \) in the \((l + 1)\)th layer.

In addition, \( w_{j,i,l}^+ = r_{j,l}p_{j,i,l}^+ \), \( w_{j,i,l}^- = r_{j,l}p_{j,i,l}^- \), and \( p_{j,i,l}^+ \), \( p_{j,i,l}^- \) are the probabilities that neuron \( j \) in the \( l \)th layer may send excitatory or inhibitory spikes to neuron \( i \) in the \((l + 1)\)th layer, \( r_{j,l} \) is the firing rate of neuron \( j \) in the \( l \)th layer.

The probability that, in the steady state, any neuron \( i \), located in in the last/output/Lth layer, is excited, is given by the expression \cite{Gel89, Gel90, Gel93}:

\[ q_{i,L} = \frac{\lambda_{i,L}^+ + \sum_{j=1}^{N_{L-1}} q_{j,L-1}w_{j,i,L-1}^+}{r_{i,L} + \lambda_{i,L}^- + \sum_{j=1}^{N_{L-1}} q_{j,L-1}w_{j,i,L-1}^-} \]  \hspace{1cm} (6.3)

The weights in the multi-layer RNN \( w_{i,j,l}^+ \) and \( w_{i,j,l}^- \) are nonnegative real numbers, i.e., \( w_{i,j,l}^+ \in \mathbb{R}_{\geq 0} \) and \( w_{i,j,l}^- \in \mathbb{R}_{\geq 0} \). Here, \( i, j = 1, 2, \ldots, N \). Let \( W_l^+ = [w_{i,j,l}^+] \in \mathbb{R}_{\geq 0}^{N_l \times N_{l+1}} \) and \( W_l^- = [w_{i,j,l}^-] \in \mathbb{R}_{\geq 0}^{N_l \times N_{l+1}} \). Then, \( q_{i,l+1} \) can be denoted as \( q_{i,l+1}(W_1^+, W_2^+, \ldots, W_l^+, W_1^-, W_2^-, \ldots, W_l^-, X) \in \mathbb{R}_{\geq 0} \).

Let \( W^+ = \{W_l^+| l = 1, \ldots, L - 1\} \) and \( W^- = \{W_l^-| l = 1, \ldots, L - 1\} \). The objective is to use the multi-layer RNN to identity (or say, to learn) the mapping \( f \). This could be achieved by minimizing a pre-defined cost function
\[ \psi(q_{1,L}(W^+, W^-, X), q_{2,L}(W^+, W^-, X), \ldots, q_{N_L,L}(W^+, W^-, X), Y). \]

Let us assume that the inputs \( \lambda^+_{n_1,1} \) in \( x_{n_1} \) with \( n_1 = 1, 2, \ldots, N_1 \) are random variables with independent distributions \( \theta^+_{n_1,1} \), i.e., \( \lambda^+_{n_1,1} \sim \theta^+_{n_1,1} \), while the inputs \( \lambda^-_{n_1,1} \) in \( x_{n_1} \) are fixed. The sample space of \( \theta^+_{n_1,1} \) is the nonnegative number.

By sampling from the distributions \( \theta^+_{n_1,1} \) for \( D \) times and exploiting the mapping \( f \), an available dataset \( S = \{(X_d, Y_d)|d = 1, 2, \ldots, D\} \) is obtained, where \( X_d = [x_{1,d}, x_{2,d}, \ldots, x_{N_1,d}] \) with \( x_{n_1,d} = [\lambda^+_{n_1,1,d}, \lambda^-_{n_1,1,d}, \lambda_{n_1,1,d} \sim \theta^+_{n_1,1}, Y_d = f(X_d) \) and \( Y_d = [y_{1,d}, y_{2,d}, \ldots, y_{N_L,d}] \). When the weights \( W^+ \) and \( W^- \) are unknown, the deep learning in the MLRNN of mapping \( f \) could be achieved by solving the minimization problem over the available dataset \( S \):

\[
\min_{W^+, W^-} \frac{1}{D} \sum_{d=1}^{D} \sum_{n_L=1}^{N_L} (q_{n_L,L,d}(W^+, W^-, X_d) - y_{n_L,d})^2, \tag{6.4}
\]

\[
\text{s.t. } W^+ \geq 0, W^- \geq 0, \sum_{j=1}^{N_L} (w^+_{i,j,l} + w^-_{i,j,l}) \leq r_{i,l} \text{ with } l = 1, 2, \ldots, L - 1.
\]

### 6.1.1 Problem Transformation

Alternatively, when the weights \( W^+ \) and \( W^- \) are unknown, the deep learning in the MLRNN of mapping \( f \) could be achieved by solving the following more general minimization problem:

\[
\min_{W^+, W^-} \mathbb{E} \left[ \sum_{n_L=1}^{N_L} (q_{n_L,L}(W^+, W^-, X) - y_{n_L})^2 \right], \tag{6.5}
\]

\[
\text{s.t. } W^+ \geq 0, W^- \geq 0, \sum_{j=1}^{N_L} (w^+_{i,j,l} + w^-_{i,j,l}) \leq r_{i,l} \text{ with } l = 1, 2, \ldots, L - 1,
\]
where the symbol $\mathbb{E}[x]$ denote the expectation of $x$. The cost function can be rewritten as:

$$
\mathbb{E} \left[ \sum_{n_L=1}^{N_L} \left( q_{n_L,L}(W^+, W^-, X) - y_{n_L} \right)^2 \right]
$$

$$
= \mathbb{E} \left[ \sum_{n_L=1}^{N_L} \left( q_{n_L,L}(W^+, W^-, X) - 2q_{n_L,L}(W^+, W^-, X)y_{n_L} + y_{n_L}^2 \right) \right] 
$$

$$
= \sum_{n_L=1}^{N_L} \left( \mathbb{E} \left[ q_{n_L,L}(W^+, W^-, X) \right] - 2\mathbb{E} \left[ q_{n_L,L}(W^+, W^-, X)y_{n_L} \right] + \mathbb{E} \left[ y_{n_L}^2 \right] \right). 
$$

Then, the deep-learning problem of the MLRNN can be reformulated as:

$$
\min_{W^+, W^-} \sum_{i=1}^{N_i} \left( \mathbb{E} \left[ q_{i,L}(W^+, W^-, X) \right] - 2\mathbb{E} \left[ q_{i,L}(W^+, W^-, X)y_i \right] + \mathbb{E} \left[ y_i^2 \right] \right), 
$$

s.t. $W^+ \geq 0, W^- \geq 0, \sum_{j=1}^{N_i} (w_{i,j,l}^+ + w_{i,j,l}^-) \leq r_{i,l}$ with $l = 1, 2, \ldots, L - 1.$

**Another Form**

The cost function could also be rewritten as:

$$
\mathbb{E} \left[ \sum_{n_L=1}^{N_L} \left( q_{n_L,L}(W^+, W^-, X) - y_{n_L} \right)^2 \right]
$$

$$
= \sum_{n_L=1}^{N_L} \left( \mathbb{E} \left[ y_{n_L,L}^2(W^+, W^-, X) \right] - \left( \mathbb{E} \left[ q_{n_L,L}(W^+, W^-, X) \right] \right)^2 \right.

+ \left( \mathbb{E} \left[ q_{n_L,L}(W^+, W^-, X) \right] \right)^2 - 2\mathbb{E} \left[ q_{n_L,L}(W^+, W^-, X) \right] \mathbb{E} \left[ y_{n_L,L} \right] + \left( \mathbb{E} \left[ y_{n_L,L} \right] \right)^2

+ \mathbb{E} \left[ y_{n_L,L}^2 \right] - \left( \mathbb{E} \left[ y_{n_L,L} \right] \right)^2 + 2\mathbb{E} \left[ q_{n_L,L}(W^+, W^-, X) \right] \mathbb{E} \left[ y_{n_L,L} \right] - 2\mathbb{E} \left[ q_{n_L,L}(W^+, W^-, X)y_{n_L,L} \right] \right) 
$$

$$
= \sum_{n_L=1}^{N_L} \left( \text{Var} \left[ q_{n_L,L}(W^+, W^-, X) \right] + \text{Var} \left[ y_{n_L,L} \right] - \text{Cov} \left[ q_{n_L,L}(W^+, W^-, X), y_{n_L,L} \right] \right.

+ \left( \mathbb{E} \left[ q_{n_L,L}(W^+, W^-, X) \right] \right)^2 - \left( \mathbb{E} \left[ y_{n_L,L} \right] \right)^2 \right),
$$

where $\text{Var} \left[ x \right]$ denote the variance of $x$, $\text{Cov} \left[ x_1, x_2 \right]$ denote the covariance between $x_1$ and $x_2$. For ease of notation, we also write $q_{n_L,L}(W^+, W^-, X)$ as $q_{n_L,L}$. In another form, the deep-learning
6.2. Analysis

problem could be written as

\[
\min_{W^+, W^-} \sum_{i=1}^{N_L} \left( \text{Var} \left[ q_{i,L} \right] + \text{Var} \left[ y_i \right] - \text{Cov} \left[ q_{i,L}, y_i \right] + (\mathbb{E} \left[ q_{i,L} \right] - \mathbb{E} \left[ y_i \right])^2 \right),
\]

s.t. \( W^+ \geq 0, W^- \geq 0, \sum_{j=1}^{N_L} (w^+_{i,j,l} + w^-_{i,j,l}) \leq r_{i,l} \) with \( l = 1, 2, \cdots, L - 1 \).

(6.9)

It can be seen that, the term \((\mathbb{E} \left[ q_{i,L} \right] - \mathbb{E} \left[ y_i \right])^2\) is to minimize the bias of the MLRNN outputs to the desired outputs; the term \(\text{Var} \left[ q_{i,L} \right]\) is to minimize the variance of the MLRNN outputs that helps to improve the stability of the MLRNN; and the term \(-\text{Cov} \left[ q_{i,L}, y_i \right]\) is to maximize the covariance between the MLRNN outputs and the desired outputs that helps to make them show the similar behaviors.

6.2 Analysis

To analyze the deep-learning problem (6.7) of the MLRNN, the expressions of \(q_{i,L}(W^+, W^-, X)\), \(q_{i,L}^2(W^+, W^-, X)\), \(\mathbb{E} \left[ q_{i,L}(W^+, W^-, X) \right]\) and \(\mathbb{E} \left[ q_{i,L}^2(W^+, W^-, X) \right]\) need to be obtained, while \(\mathbb{E} \left[ y_i \right]\) and \(\mathbb{E} \left[ y_i^2 \right]\) could be estimated from available data.

Let us analyze the expression of \(\mathbb{E} \left[ q_{i,L}(W^+, W^-, X) \right]\) first. To this end, the analysis needs to start from the expression of \(\mathbb{E} \left[ q_{i,1}(X) \right]\) of the first layer. Then, the expected outputs of the 2nd, 3rd, \(\cdots\), \(L\)-th layers, i.e., \(\mathbb{E} \left[ q_{i,2} \right]\), \(\mathbb{E} \left[ q_{i,3} \right]\), \(\cdots\), \(\mathbb{E} \left[ q_{i,L} \right]\), are analyzed successively. The expressions of \(\mathbb{E} \left[ q_{i,L}^2 \right]\) and \(\text{Var} \left[ q_{i,L} \right]\) are also obtained. Based on the analysis in this section, Section 6.3 presents the theoretical results on the MLRNN for deep learning.

6.2.1 The expression of \(\mathbb{E} \left[ q_{i,1}(X) \right]\)

From (6.1),

\[
\mathbb{E} \left[ q_{i,1}(X) \right] = \mathbb{E} \left[ \frac{\lambda_{i,1}^+}{r_{i,1} + \lambda_{i,1}^+} \right] = \frac{\mathbb{E} \left[ \lambda_{i,1}^+ \right]}{r_{i,1} + \lambda_{i,1}^+},
\]

(6.10)

For ease of notation, we also write \(q_{i,1}(X)\) as \(q_{i,1}\).
6.2.2 The expression of $\mathbb{E}[q_{i,1}^2]$

From (6.1),

$$\mathbb{E}[q_{i,1}^2] = \mathbb{E} \left[ \left( \frac{\lambda_{i,1}^+}{r_{i,1} + \lambda_{i,1}^-} \right)^2 \right] = \mathbb{E} \left[ \left( \frac{\blue{\lambda_{i,1}^+}}{r_{i,1} + \lambda_{i,1}^-} \right)^2 \right].$$  \hspace{1cm} (6.11)

6.2.3 The expression of $\mathbb{E}[q_{i,1}^n]$

From (6.1),

$$\mathbb{E}[q_{i,1}^n] = \mathbb{E} \left[ \left( \frac{\lambda_{i,1}^+}{r_{i,1} + \lambda_{i,1}^-} \right)^n \right] = \mathbb{E} \left[ \left( \frac{\blue{\lambda_{i,1}^+}}{r_{i,1} + \lambda_{i,1}^-} \right)^n \right].$$  \hspace{1cm} (6.12)

6.2.4 The expression of $\mathbb{E}[q_{i,2}(W_1^+, W_1^-, X)]$

From (6.2), the expression of $\mathbb{E}[q_{i,2}(W_1^+, W_1^-, X)]$ is obtained as

$$\mathbb{E}[q_{i,2}(W_1^+, W_1^-, X)] = \mathbb{E} \left[ \frac{\lambda_{i,2}^+ + \sum_{j=1}^{N_1} q_{j,1} w_{j,1}^+}{r_{i,2} + \lambda_{i,2}^- + \sum_{j=1}^{N_1} q_{j,1} w_{j,1}^-} \right].$$  \hspace{1cm} (6.13)

However, the expression seems not to be decomposable.

To the end of a decomposable $\mathbb{E}[q_{i,2}(W_1^+, W_1^-, X)]$, it is possible to linearize the formula via approximation. For ease of analysis, the following first-order approximation could be obtained:

$$q_{i,2} = \frac{\lambda_{i,2}^+ + \sum_{j=1}^{N_1} q_{j,1} w_{j,1}^+}{r_{i,2} + \lambda_{i,2}^- + \sum_{j=1}^{N_1} q_{j,1} w_{j,1}^-} \approx \frac{\lambda_{i,2}^+ + \sum_{j=1}^{N_1} q_{j,1} w_{j,1}^+}{r_{i,2} + \lambda_{i,2}^- + \sum_{j=1}^{N_1} q_{j,1} w_{j,1}^-} \left(1 - \sum_{j=1}^{N_1} q_{j,1} w_{j,1}^- \right)$$

$$= \frac{\lambda_{i,2}^+}{r_{i,2} + \lambda_{i,2}^-} + \frac{\sum_{j=1}^{N_1} q_{j,1} w_{j,1}^+}{r_{i,2} + \lambda_{i,2}^-} - \frac{\lambda_{i,2}^+ \sum_{j=1}^{N_1} q_{j,1} w_{j,1}^-}{(r_{i,2} + \lambda_{i,2}^-)^2} \left(1 - \sum_{j=1}^{N_1} q_{j,1} w_{j,1}^- \right)$$

Note that, in the second layer, both $\lambda_{i,2}^+$ and $\lambda_{i,2}^-$ are given, rather than random variables. Using
equation (6.14), the expression of $E[q_{i,2}(W_1^+, W_1^-, X)]$ becomes:

$$
E[q_{i,2}(W_1^+, W_1^-, X)] = E \left[ \frac{\lambda_{i,2}^+ + \sum_{j=1}^{N_1} q_{j,1} w_{j,i,1}^+}{r_{i,2} + \lambda_{i,2}^- + \sum_{j=1}^{N_1} q_{j,1} w_{j,i,1}^-} \right] 
$$

$$
= E \left[ \frac{\lambda_{i,2}^+}{r_{i,2} + \lambda_{i,2}^-} \right] + E \left[ \frac{\sum_{j=1}^{N_1} q_{j,1} w_{j,i,1}^+}{(r_{i,2} + \lambda_{i,2}^-)^2} \right] - E \left[ \frac{\lambda_{i,2}^+ \sum_{j=1}^{N_1} q_{j,1} w_{j,i,1}^-}{(r_{i,2} + \lambda_{i,2}^-)^2} \right].
$$

The last term can be decomposed as

$$
\frac{\sum_{j=1}^{N_1} \sum_{k=1}^{N_1} E[q_{j,1} q_{k,1}] w_{j,i,1}^+ w_{k,i,1}^-}{(r_{i,2} + \lambda_{i,2}^-)^2} = \sum_{j=1}^{N_1} \sum_{k=1(k \neq j)}^{N_1} E[q_{j,1}] E[q_{k,1}] w_{j,i,1}^+ w_{k,i,1}^- 
$$

$$
+ \frac{\sum_{j=1}^{N_1} E[q_{j,1}^2] w_{j,i,1}^+ w_{j,i,1}^-}{(r_{i,2} + \lambda_{i,2}^-)^2}.
$$

Then,

$$
E[q_{i,2}] \approx \frac{\lambda_{i,2}^+}{r_{i,2} + \lambda_{i,2}^-} + \frac{\sum_{j=1}^{N_1} E[q_{j,1}] w_{j,i,1}^+}{r_{i,2} + \lambda_{i,2}^-} - \frac{\lambda_{i,2}^+ \sum_{j=1}^{N_1} E[q_{j,1}] w_{j,i,1}^-}{(r_{i,2} + \lambda_{i,2}^-)^2} 
$$

$$
- \sum_{j=1}^{N_1} \sum_{k=1(k \neq j)}^{N_1} E[q_{j,1}] E[q_{k,1}] w_{j,i,1}^+ w_{k,i,1}^- 
$$

$$
+ \frac{\sum_{j=1}^{N_1} E[q_{j,1}^2] w_{j,i,1}^+ w_{j,i,1}^-}{(r_{i,2} + \lambda_{i,2}^-)^2}.
$$

(6.15)

6.2.5 Another form of the expression of $E[q_{i,2}]$

The expression of $q_{i,2}$ could also be written in another form as:

$$
q_{i,2} \approx \frac{\lambda_{i,2}^+}{r_{i,2} + \lambda_{i,2}^-} + \frac{\sum_{j=1}^{N_1} q_{j,1} w_{j,i,1}^+}{r_{i,2} + \lambda_{i,2}^-} - \frac{\lambda_{i,2}^+ \sum_{j=1}^{N_1} q_{j,1} w_{j,i,1}^-}{(r_{i,2} + \lambda_{i,2}^-)^2} 
$$

$$
- \sum_{j=1}^{N_1} \sum_{k=1}^{N_1} w_{j,k} q_{j,1} q_{k,1}.
$$

(6.17)
\[ w^{(1)} + \sum_{j=1}^{N} w^{(2)}_{j} q_{j,1}^{(2)} + \sum_{j=1}^{N} w^{(3)}_{j} q_{j,1}^{(3)} + \sum_{j=1}^{N} \sum_{k=1}^{N} w^{(4)}_{j,k} q_{j,1} q_{k,1} \mathsf{E} \left[ q_{j,1} \mathsf{E} \left[ q_{k,1} \right] \right]_{j \neq k}, \]  

(6.19)

where \( \hat{w}^{(1)}, \hat{w}^{(2)}, \hat{w}^{(3)}, w^{(1)}, w^{(2)}, w^{(3)} \) and \( w^{(4)} \) are the corresponding weights from the given parameters, rather than random variables. The symbol \( \sum_{j=1}^{N} \sum_{k=1}^{N} x_{j} x_{k} \) produces the sum of \( x_{j} x_{k} \) satisfying the condition \( j \neq k \).

In the second layer, both \( \lambda_{i,2}^{+} \) and \( \lambda_{i,2}^{-} \) are given. Then, the expression of \( \mathbb{E}[q_{i,2}(W_{1}^{+}, W_{1}^{-}, X)] \) could also be written as

\[ \mathbb{E}[q_{i,2}] \cong \mathbb{E} \left[ w^{(1)} + \sum_{j=1}^{N} w^{(2)}_{j} \mathbb{E} \left[ q_{j,1} \right] + \sum_{j=1}^{N} w^{(3)}_{j} \mathbb{E} \left[ q_{j,1}^{2} \right] + \sum_{j=1}^{N} \sum_{k=1}^{N} w^{(4)}_{j,k} \mathbb{E} \left[ q_{j,1} \mathbb{E} \left[ q_{k,1} \right] \right] \right]_{j \neq k}. \]  

(6.20)

Therefore, the expression of \( \mathbb{E}[q_{i,2}] \) is a polynomial of \( \mathbb{E}[q_{j,1}] \) and \( \mathbb{E} \left[ q_{j,1}^{2} \right] \) with \( j = 1, 2, \ldots, N_{1} \), i.e., \( \mathbb{E}[q_{i,2}] = \mathbb{P} \left[ \mathbb{E} \left[ q_{1,1} \right], \ldots, \mathbb{E} \left[ q_{N_{1},1} \right] \right] \), where the symbol \( \mathbb{P}[x_{1}, x_{2}, \ldots, x_{n}] \) denotes a polynomial of \( x_{1}, x_{2}, \ldots, x_{n} \).

### 6.2.6 The expression of \( \mathbb{E}[q_{i,2}^{2}] \)

From (6.14), we have

\[ q_{i,2}^{2} \cong \left( \frac{\lambda_{i,2}^{+}}{r_{i,2} + \lambda_{i,2}^{-}} \right)^{2} + \left( \frac{\lambda_{i,2}^{-}}{r_{i,2} + \lambda_{i,2}^{+}} \right)^{2} + \sum_{j=1}^{N_{1}} q_{j,1} w_{j,i,1}^{+} \left( r_{i,2} + \lambda_{i,2}^{+} \right) \left( r_{i,2} + \lambda_{i,2}^{-} \right) - \sum_{j=1}^{N_{1}} \sum_{k=1}^{N_{1}} q_{j,1} q_{k,1} w_{j,i,1}^{+} w_{k,i,1}^{-} \right)^{2}, \]  

(6.21)
Then, the expression of $q_{i,2}^2$ could also be written in the following form:

$$q_{i,2}^2 \approx \hat{w}^{(1)} + \sum_{j=1}^{N_1} \hat{w}^{(2)}_{j} q_{j,1} + \sum_{j=1}^{N_1} \sum_{k=1}^{N_1} \hat{w}^{(3)}_{j,k} q_{j,1} q_{k,1} + \sum_{j=1}^{N_1} \sum_{k=1}^{N_1} \sum_{j=1}^{N_1} \hat{w}^{(4)}_{j,k,j,k} q_{j,1} q_{k,1} q_{j,1} q_{k,1}$$

(6.22)

$$+ \sum_{j=1}^{N_1} \sum_{k=1}^{N_1} \sum_{j=1}^{N_1} \sum_{k=1}^{N_1} \hat{w}^{(5)}_{j,k,j,k} q_{j,1} q_{k,1} q_{j,1} q_{k,1}.$$  (6.22)

Let us handle the elements in (6.22) by considering the following expressions first:

$$E \left[ \sum_{j=1}^{N_1} q_{j,1} \right] = \sum_{j=1}^{N_1} E \left[ q_{j,1} \right];$$  (6.23)

$$E \left[ \sum_{j=1}^{N_1} \sum_{k=1}^{N_1} q_{j,1} q_{k,1} \right] = \sum_{j=1}^{N_1} E \left[ q_{j,1}^2 \right] + \sum_{j=1}^{N_1} \sum_{k=1}^{N_1} w_{j,k}^{(1)} E \left[ q_{j,1} \right] E \left[ q_{k,1} \right].$$  (6.24)

For notation ease, the symbol $j, k, \hat{j}$ denotes that $j, k, \hat{j}$ are distinct numbers. Then,

$$E \left[ \sum_{j=1}^{N_1} \sum_{k=1}^{N_1} \sum_{j=1}^{N_1} q_{j,1} q_{k,1} q_{j,1} q_{k,1} \right] = \sum_{j=1}^{N_1} E \left[ q_{j,1}^3 \right] + \sum_{j=1}^{N_1} \sum_{k=1}^{N_1} w_{j,k}^{(1)} E \left[ q_{j,1}^2 \right] E \left[ q_{k,1} \right] E \left[ q_{j,1} \right]$$

(6.25)

$$+ \sum_{j=1}^{N_1} \sum_{k=1}^{N_1} \sum_{j=1}^{N_1} w_{j,k,j}^{(2)} E \left[ q_{j,1} \right] E \left[ q_{k,1} \right] E \left[ q_{j,1} \right] E \left[ q_{k,1} \right].$$

Similarly, the symbol $j, k, \hat{j}, \hat{k}$ denotes that $j, k, \hat{j}, \hat{k}$ are distinct numbers. Then,

$$E \left[ \sum_{j=1}^{N_1} \sum_{k=1}^{N_1} \sum_{j=1}^{N_1} q_{j,1} q_{k,1} q_{j,1} q_{k,1} q_{j,1} q_{k,1} \right] = \sum_{j=1}^{N_1} E \left[ q_{j,1}^4 \right] + \sum_{j=1}^{N_1} \sum_{k=1}^{N_1} \sum_{j=1}^{N_1} w_{j,k}^{(1)} E \left[ q_{j,1}^3 \right] E \left[ q_{j,1} \right] E \left[ q_{k,1} \right]$$

(6.26)

$$+ \sum_{j=1}^{N_1} \sum_{k=1}^{N_1} \sum_{j=1}^{N_1} \sum_{k=1}^{N_1} \sum_{j=1}^{N_1} \sum_{k=1}^{N_1} \sum_{j=1}^{N_1} \sum_{k=1}^{N_1} w_{j,k,j,k}^{(3)} E \left[ q_{j,1} \right] E \left[ q_{k,1} \right] E \left[ q_{j,1} \right] E \left[ q_{k,1} \right] E \left[ q_{j,1} \right] E \left[ q_{k,1} \right].$$
Therefore, the expression of $\mathbb{E}[q_{i,2}^2]$ could be written as

$$\mathbb{E}[q_{i,2}^2] \simeq \hat{w}^{(1)} + \mathbb{E}\left[ \sum_{j=1}^{N_i} \tilde{w}_{j,2}^{(2)} q_{j,1} \right] + \mathbb{E}\left[ \sum_{j=1}^{N_i} \sum_{k=1}^{N_i} \tilde{w}_{j,k}^{(3)} q_{j,1} q_{k,1} \right] + \mathbb{E}\left[ \sum_{j=1}^{N_i} \sum_{k=1}^{N_i} \sum_{j'=1}^{N_i} \tilde{w}_{j,k,j'}^{(4)} q_{j,1} q_{k,1} q_{j',1} \right]$$

$$+ \mathbb{E}\left[ \sum_{j=1}^{N_i} \sum_{k=1}^{N_i} \sum_{j=1}^{N_i} \sum_{k=1}^{N_i} \tilde{w}_{j,k,j,k}^{(5)} q_{j,1} q_{k,1} q_{j',1} q_{k',1} \right]$$

$$= \hat{w}^{(1)} + \sum_{j=1}^{N_i} \tilde{w}_{j,2}^{(2)} \mathbb{E}[q_{j,1}] + \sum_{j=1}^{N_i} \tilde{w}_{j,2}^{(3)} \mathbb{E}[q_{j,1}^2] + \sum_{j=1}^{N_i} \sum_{k=1}^{N_i} \tilde{w}_{j,k}^{(4)} \mathbb{E}[q_{j,1}] \mathbb{E}[q_{k,1}]$$

$$+ \sum_{j=1}^{N_i} \tilde{w}_{j,2}^{(5)} \mathbb{E}[q_{j,1}^2] + \sum_{j=1}^{N_i} \sum_{k=1}^{N_i} \tilde{w}_{j,k}^{(6)} \mathbb{E}[q_{j,1}] \mathbb{E}[q_{k,1}] + \sum_{j=1}^{N_i} \sum_{k=1}^{N_i} \tilde{w}_{j,k}^{(7)} \mathbb{E}[q_{j,1}^2] \mathbb{E}[q_{k,1}]$$

$$+ \sum_{j=1}^{N_i} \sum_{k=1}^{N_i} \sum_{j=1}^{N_i} \sum_{k=1}^{N_i} \tilde{w}_{j,k,j,k}^{(8)} \mathbb{E}[q_{j,1}] \mathbb{E}[q_{k,1}] \mathbb{E}[q_{j,1}] \mathbb{E}[q_{k,1}]$$

$$+ \sum_{j=1}^{N_i} \sum_{k=1}^{N_i} \sum_{j=1}^{N_i} \sum_{k=1}^{N_i} \sum_{j'=1}^{N_i} \sum_{k'=1}^{N_i} \tilde{w}_{j,k,j,k}^{(9)} \mathbb{E}[q_{j,1}] \mathbb{E}[q_{k,1}] \mathbb{E}[q_{j,1}] \mathbb{E}[q_{k,1}]$$

$$+ \sum_{j=1}^{N_i} \sum_{k=1}^{N_i} \sum_{j=1}^{N_i} \sum_{k=1}^{N_i} \sum_{j'=1}^{N_i} \sum_{k'=1}^{N_i} \sum_{j''=1}^{N_i} \tilde{w}_{j,k,j,k,j''}^{(10)} \mathbb{E}[q_{j,1}] \mathbb{E}[q_{k,1}] \mathbb{E}[q_{j,1}] \mathbb{E}[q_{k,1}]$$

$$+ \sum_{j=1}^{N_i} \sum_{k=1}^{N_i} \sum_{j=1}^{N_i} \sum_{k=1}^{N_i} \sum_{j'=1}^{N_i} \sum_{k'=1}^{N_i} \sum_{j''=1}^{N_i} \sum_{j'''=1}^{N_i} \tilde{w}_{j,k,j,k,j''}^{(11)} \mathbb{E}[q_{j,1}] \mathbb{E}[q_{k,1}] \mathbb{E}[q_{j,1}] \mathbb{E}[q_{k,1}]$$

$$\cdot \mathbb{E}[q_{j,1}] \mathbb{E}[q_{k,1}]$$

(6.27)

### 6.2.7 The expression of $\mathbb{E}[q_{i,2}^n]$

Based on the expression of $q_{i,2}$ in (6.18), the expression of $q_{i,2}^3$ could be written as

$$q_{i,2}^3 \simeq \hat{w}^{(1)} + \sum_{j=1}^{N_i} \tilde{w}_{j,3}^{(2)} q_{j,1} q_{j,1} q_{j,1} + \sum_{j=1}^{N_i} \sum_{j'=1}^{N_i} \tilde{w}_{j,j',3}^{(3)} q_{j,1} q_{j',1} q_{j,1}$$

$$+ \sum_{j=1}^{N_i} \sum_{j'=1}^{N_i} \sum_{j''=1}^{N_i} \sum_{j'''=1}^{N_i} \tilde{w}_{j,j',j'',j'''}^{(4)} q_{j,1} q_{j',1} q_{j',1} q_{j'',1}$$

$$+ \sum_{j=1}^{N_i} \sum_{j'=1}^{N_i} \sum_{j''=1}^{N_i} \sum_{j'''=1}^{N_i} \sum_{j''''=1}^{N_i} \sum_{j'''''=1}^{N_i} \tilde{w}_{j,j',j'',j''''}^{(5)} q_{j,1} q_{j',1} q_{j',1} q_{j'',1} q_{j'',1}$$

$$= \hat{w}^{(1)} + \sum_{k=1}^{6} \sum_{j=1}^{N_i} \sum_{j'=1}^{N_i} \tilde{w}_{j,j',\ldots,j,k}^{(k+1)} \prod_{k=1}^{k} q_{j,k,1}$$

(6.28)
6.2. Analysis

Similarly, the expression of \( q_{i,2}^{n} \) could be written as

\[
q_{i,2}^{n} \cong \tilde{w}^{(1)} + \sum_{k=1}^{2n} \left( \sum_{j_1=1}^{N_1} \cdots \sum_{j_k=1}^{N_1} w_{j_1,\cdots,j_k}^{(k+1)} \prod_{k=1}^{k} q_{j_k,1} \right)
\]  \hspace{1cm} (6.29)

Based on the expressions from (6.23) to (6.26), we have

\[
\mathbb{E} \left[ \sum_{j_1=1}^{N_1} \cdots \sum_{j_k=1}^{N_1} w_{j_1,\cdots,j_k}^{(k+1)} \prod_{k=1}^{k} q_{j_k,1} \right]
\]  \hspace{1cm} (6.30)

\[
= \mathbb{P} \left[ \mathbb{E} [q_{1,1}], \cdots, \mathbb{E} [q_{N_1,1}], \mathbb{E} [q_{1,1}^2], \cdots, \mathbb{E} [q_{N_1,1}^2], \cdots, \mathbb{E} [q_{1,1}^{2n}], \cdots, \mathbb{E} [q_{N_1,1}^{2n}] \right].
\]

Then, we have

\[
\mathbb{E} [q_{i,2}^{n}] \cong \mathbb{P} \left[ \mathbb{E} [q_{1,1}], \cdots, \mathbb{E} [q_{N_1,1}], \mathbb{E} [q_{1,1}^2], \cdots, \mathbb{E} [q_{N_1,1}^2], \cdots, \mathbb{E} [q_{1,1}^{2n}], \cdots, \mathbb{E} [q_{N_1,1}^{2n}] \right],
\]  \hspace{1cm} (6.31)

i.e., the expression of \( \mathbb{E} [q_{i,2}^{n}] \) is a polynomial of \( \mathbb{E} [q_{1,1}], \cdots, \mathbb{E} [q_{N_1,1}], \mathbb{E} [q_{1,1}^2], \cdots, \mathbb{E} [q_{N_1,1}^2], \cdots, \mathbb{E} [q_{1,1}^{2n}], \cdots, \mathbb{E} [q_{N_1,1}^{2n}] \), where all these elements of expectations can be estimated or evaluated over the available dataset using equation (6.12).

6.2.8 The expression of \( \mathbb{E} [q_{i,3}] \)

Based on the expression of \( q_{i,2} \) in (6.18), the expression of \( q_{i,3} \) could be written as

\[
q_{i,3} \cong \frac{\lambda_{i,3}^+}{r_{i,3} + \lambda_{i,3}^-} + \frac{\sum_{j=1}^{N_2} q_{j,2} w_{j,2}^+}{r_{i,3} + \lambda_{i,3}^-} + \frac{\lambda_{i,3}^+}{r_{i,3} + \lambda_{i,3}^-} \left( \sum_{j=1}^{N_2} q_{j,2} w_{j,2}^- \right) - \frac{\sum_{j=1}^{N_2} \sum_{k=1}^{N_2} q_{j,2} q_{k,2} \sum_{j=1}^{N_2} q_{j,2} \sum_{k=1}^{N_2} q_{k,2} \sum_{j=1}^{N_2} q_{j,2} \sum_{k=1}^{N_2} q_{k,2}}{r_{i,3} + \lambda_{i,3}^-} \right)
\]  \hspace{1cm} (6.32)

\[
= \tilde{w}^{(1)} + \sum_{j=1}^{N_2} \tilde{w}_{j,2}^{(2)} q_{j,2} + \sum_{j=1}^{N_2} \sum_{k=1}^{N_2} \tilde{w}_{j,k}^{(3)} q_{j,2} q_{k,2}
\]  \hspace{1cm} (6.32)

\[
= \tilde{w}^{(1)} + \sum_{k=1}^{4} \left( \sum_{j_1=1}^{N_1} \cdots \sum_{j_k=1}^{N_1} w_{j_1,\cdots,j_k}^{(k+1)} \prod_{k=1}^{k} q_{j_k,1} \right).
\]
Based on the analysis on (6.30) and (6.31), the expression of $E[q_{i,3}]$ can be written as

$$E[q_{i,3}] \cong P \left[ E[q_{1,1}], \ldots, E[q_{N,1}], E[q_{1,1}^2], \ldots, E[q_{N,1}^2], \ldots, E[q_{1,1}^3], \ldots, E[q_{N,1}^3] \right]. \quad (6.33)$$

### 6.2.9 The expression of $E[q_{i,3}^2]$

Based on the expressions of $q_{i,2}$ in (6.18) and $q_{i,3}$ in (6.32), the expression of $q_{i,3}^2$ could be written as

$$q_{i,3}^2 \cong \tilde{w}^{(1)} + \sum_{k=1}^{8} \left( \sum_{j_1=1}^{N_1} \ldots \sum_{j_k=1}^{N_1} \tilde{w}_{j_1,\ldots,j_k}^{(k+1)} \prod_{k=1}^{k} q_{j_k,1} \right). \quad (6.34)$$

Based on the analysis on (6.30) and (6.31), the expression of $E[q_{i,3}^2]$ could be written as

$$E[q_{i,3}] \cong P \left[ E[q_{1,1}], \ldots, E[q_{N,1}], E[q_{1,1}^2], \ldots, E[q_{N,1}^2], \ldots, E[q_{1,1}^3], \ldots, E[q_{N,1}^3] \right]. \quad (6.35)$$

### 6.2.10 The expression of $E[q_{i,L}]$

Finally, the expression of $q_{i,L}$ could be written as

$$q_{i,L} \cong \tilde{w}^{(1)} + \sum_{k=1}^{2^{L-1}} \left( \sum_{j_1=1}^{N_1} \ldots \sum_{j_k=1}^{N_1} \tilde{w}_{j_1,\ldots,j_k}^{(k+1)} \prod_{k=1}^{k} q_{j_k,1} \right). \quad (6.36)$$

Based on the analysis on (6.30) and (6.31), the expression of $E[q_{i,L}]$ could be finally written as

$$E[q_{i,L}] \cong P \left[ E[q_{1,1}], \ldots, E[q_{N,1}], E[q_{1,1}^2], \ldots, E[q_{N,1}^2], \ldots, E[q_{1,1}^{2^{L-1}}], \ldots, E[q_{N,1}^{2^{L-1}}] \right]. \quad (6.37)$$

That is, the $L$-th layer expected output of the MLRNN is a polynomial of $E[q_{1,1}], \ldots, E[q_{N,1}], E[q_{1,1}^2], \ldots, E[q_{N,1}^2], \ldots, E[q_{1,1}^{2^{L-1}}], \ldots, E[q_{N,1}^{2^{L-1}}]$, where all these elements of expectations can be estimated or evaluated over the available dataset using equation (6.12).
6.2.11 The expression of $\mathbb{E}[q_{i,L} y_i]$

Based on (6.36), the expression of $q_{i,L} y_i$ could be written as

$$q_{i,L} y_i \cong \tilde{w}^{(1)} y_i + \sum_{k=1}^{2L-1} \left( \sum_{j_1=1}^{N_1} \cdots \sum_{j_k=1}^{N_1} \tilde{w}_{j_1, \ldots, j_k}^{(k+1)} \prod_{k=1}^{k} q_{j_k,1} \right). \quad (6.38)$$

Based on the analysis on (6.30) and (6.31), the expression of $\mathbb{E}[q_{i,L} y_i]$ could be finally written as

$$\mathbb{E}[q_{i,L} y_i] \cong \mathbb{P} \left[ \mathbb{E}[y_i], \mathbb{E}[q_{1,1} y_i], \cdots, \mathbb{E}[q_{N_1,1} y_i], \mathbb{E}[q_{1,1}^2], \cdots, \mathbb{E}[q_{N_1,1}^2] \right], \quad (6.39)$$

6.2.12 The expression of $\mathbb{E}[q_{i,L}^2]$ and $\text{Var}[q_{i,L}]$

Then, the expression of $q_{i,L}^2$ could be written as

$$q_{i,L}^2 \cong \tilde{w}^{(1)} + \sum_{k=1}^{2L} \left( \sum_{j_1=1}^{N_1} \cdots \sum_{j_k=1}^{N_1} \tilde{w}_{j_1, \ldots, j_k}^{(k+1)} \prod_{k=1}^{k} q_{j_k,1} \right). \quad (6.40)$$

Based on the analysis on (6.30) and (6.31), the expression of $\mathbb{E}[q_{i,L}^2]$ could be finally written as

$$\mathbb{E}[q_{i,L}^2] \cong \mathbb{P} \left[ \mathbb{E}[q_{1,1}], \cdots, \mathbb{E}[q_{N_1,1}], \mathbb{E}[q_{1,1}^2], \cdots, \mathbb{E}[q_{N_1,1}^2], \mathbb{E}[q_{1,1}^{2L}], \cdots, \mathbb{E}[q_{N_1,1}^{2L}] \right]. \quad (6.41)$$

From (6.37), we know that

$$\left( \mathbb{E}[q_{i,L}] \right)^2 \cong \mathbb{P} \left[ \left( \mathbb{E}[q_{1,1}] \right)^2, \cdots, \left( \mathbb{E}[q_{N_1,1}] \right)^2, \left( \mathbb{E}[q_{1,1}^2] \right)^2, \cdots, \left( \mathbb{E}[q_{N_1,1}^2] \right)^2, \cdots, \left( \mathbb{E}[q_{1,1}^{2L}] \right)^2, \cdots, \left( \mathbb{E}[q_{N_1,1}^{2L}] \right)^2 \right]. \quad (6.42)$$
Then, the variance of $q_{i,L}$ could be written as

\[
\text{Var}[q_{i,L}] = \mathbb{E} [q_{i,L}^2] - (\mathbb{E} [q_{i,L}])^2 \\
\approx \mathbb{P} \left[ (\mathbb{E} [q_{1,1}])^2, \cdots, (\mathbb{E} [q_{N,1}])^2, (\mathbb{E} [q_{1,1}^2])^2, \cdots, (\mathbb{E} [q_{N,1}^2])^2, \cdots, (\mathbb{E} [q_{1,1}^{2L-1}])^2, \cdots, (\mathbb{E} [q_{N,1}^{2L-1}])^2 \right] \\
\approx \mathbb{P} \left[ \mathbb{E} [q_{1,1}], \cdots, \mathbb{E} [q_{N,1}], \mathbb{E} [q_{1,1}^2], \cdots, \mathbb{E} [q_{N,1}^2], \cdots, \mathbb{E} [q_{1,1}^{2L}], \cdots, \mathbb{E} [q_{N,1}^{2L}] \right].
\]

(6.43)

### 6.3 Theorem and Hypothesis

Based on the theoretical analysis in Section 6.2, this section presents the theoretical results on the approximation property of the MLRNN, the learning efficiency of a multi-layer architecture and the statistic representations extracted from the data through multi layers.

#### 6.3.1 Universal Approximation Property

The MLRNN is a universal function approximator.

To lay a basis, the following lemma about the multivariate Bernstein polynomial is given from [ZYG+14a, ZYG+14b, D06, SS04].

**Lemma 6.1** For a continuous function $f(X) : [0,1]^{1 \times N} \rightarrow \mathbb{R}$, the following multivariate Bernstein polynomial $B_G(X)$ of $f(X)$, uniformly converges to $f(X)$:

\[
B_G(X) = \sum_{j_1=1}^{G} \cdots \sum_{j_N=1}^{G} \tilde{w}_{j_1,\cdots,j_N} \prod_{k=1}^{N} p_{G,j_k}(x_k),
\]

(6.44)

where $\tilde{w}_{j_1,\cdots,j_N} = f(j_1/G, \cdots, j_N/G)$ denotes the weight for $\prod_{k=1}^{G} p_{G,j_k}(x_k)$ and $p_{G,j_k}(x_k) = \binom{N}{j_k} x_k^{j_k} (1-x_k)^{N-j_k}$. That is, $B_G(X) \rightarrow f(X)$, as $G \rightarrow +\infty$. 


Let $Q_1 = [q_{1,1}, q_{2,1}, \cdots, q_{N_1,1}] \in [0, 1]^{1 \times N_1}$. Then, the following theorem on the universal approximation property of the MLRNN could be obtained.

**Theorem 6.1** For a continuous function $f(Q_1) : [0, 1]^{1 \times N} \rightarrow [0, 1]$, there exists a multi-layer RNN with $L$ layers and a $L$-th layer output $q_{1,L}(W^+, W^-, Q_1)$ such that $q_{1,L}(W^+, W^-, Q_1)$ approximates $f(Q_1)$ uniformly, i.e., $q_{1,L}(W^+, W^-, Q_1) \rightarrow f(Q_1)$, as $L \rightarrow +\infty$.

**Proof.** Based on the analysis in Section 6.2, we have the expression of $q_{1,L}(W^+, W^-, Q_1)$ in (6.26):

$$q_{1,L} \equiv \tilde{w}^{(1)} + \sum_{k=1}^{2L-1} \left( \sum_{j_1=1}^{N_1} \cdots \sum_{j_k=1}^{N_1} \tilde{w}_{j_1,\cdots,j_k}^{(k+1)} \prod_{k=1}^{k} q_{j_k,1} \right).$$  \hspace{1cm} (6.45)

In the expression of $q_{1,L}$, the degrees of the terms with respect to $q_{i,1}$ vary from 0 to $2^{L-1}$. The expression of $q_{1,L}$ can also be written in another form as:

$$q_{1,L} \equiv \sum_{j_1=0}^{2^{L-1}} \cdots \sum_{j_{N_1}=0}^{2^{L-1}} \tilde{w}_{j_1,\cdots,j_{N_1}}^{(1)} \prod_{k=1}^{N_1} q_{j_k,1}.$$  \hspace{1cm} (6.46)

Based on the Weierstrass theorem [PPP+81] and the work on the RNN function approximator in [GML99a, GML99b, GML04] and Chapter 3, we know that, for any given continuous function $f(Q_1) : [0, 1]^{1 \times N} \rightarrow [0, 1]$, there exists a function with the form $\tilde{\varphi}(Q_1) = \sum_{j_1=0}^{2^{L-1}} \cdots \sum_{j_{N_1}=0}^{2^{L-1}} \tilde{w}_{j_1,\cdots,j_{N_1}}^{(1)} \prod_{k=1}^{N_1} q_{j_k,1}^{q_{k,1}}$ such that $\tilde{\varphi}(Q_1) \rightarrow f(Q_1)$, as $2^{L-1} \rightarrow +\infty$.

The following presents a proof based on the multivariate Bernstein polynomial. Based on Lemma 6.1, the multivariate Bernstein polynomial $B_G(Q_1)$ of $f(Q_1)$ can be constructed as

$$B_G(Q_1) = \sum_{j_1=1}^{G} \cdots \sum_{j_{N_1}=1}^{G} \tilde{w}_{j_1,\cdots,j_{N_1}}^{(1)} \prod_{k=1}^{N_1} p_{G,j_k}(q_{k,1}),$$  \hspace{1cm} (6.47)

where $p_{G,j_k}(q_{k,1}) = \binom{G}{j_k} q_{k,1}^{j_k} (1-q_{k,1})^{G-j_k}$. In the expression of $B_G(Q_1)$, the degrees of the terms with respect to $q_{i,1}$ vary from 0 to $G$. Let $G = 2^{L-1}$. Then, the expression of $B_G(Q_1)$ can also
be rewritten in the following form:

$$B_{2^{L-1}}(Q_1) = \sum_{j_1=0}^{2^{L-1}} \cdots \sum_{j_{N_1}=0}^{2^{L-1}} \tilde{w}^{(2)}_{j_1,\cdots,j_{N_1}} \prod_{k=1}^{N_1} q_{k,1}^k.$$

(6.48)

Then, there exists a set of coefficients $\tilde{w}^{(1)}_{j_1,\cdots,j_{N_1}}$ such that

$$q_{1,L} \cong B_{2^{L-1}}(Q_1).$$

(6.49)

From Lemma 6.1 we know that $B_{2^{L-1}}(Q_1) \to f(Q_1)$, as $2^{L-1} \to +\infty$. Then, we know that there exists a MLRNN with $L$ layers such that its output $q_{1,L}(W^+,W^-,Q_1)$ approximates $f(Q_1)$ uniformly, i.e., $q_{1,L}(W^+,W^-,Q_1) \to f(Q_1)$, as $L \to +\infty$.

The proof is thus completed. 

\[\square\]

### 6.3.2 Learning Efficiency in a Deep Architecture

Given a target continuous function, the approximation error of the MLRNN could converge towards the desired degree of approximation error exponentially as the number of layers increases.

**Definition 6.1** Let $\Xi = [G_1 \ G_2 \ \cdots \ G_N] \in \mathbb{Z}_{\geq 0}^{1 \times N}$ with $\mathbb{Z}_{\geq 0} = \mathbb{Z}_+ \cup \{0\}$. A $N$-variable polynomial

$$P_\Xi(X) = \sum_{j_1=0}^{G_1} \cdots \sum_{j_N=0}^{G_N} \tilde{w}_{j_1,\cdots,j_N} \prod_{k=1}^{N} x_k^{j_k}$$

(6.50)

is defined as a polynomial with degree $[G_1 \ G_2 \ \cdots \ G_N]$.

**Definition 6.2** When $G_1 = G_2 = \cdots = G_N = G$, a $N$-variable polynomial with degree $[G_1 \ G_2 \ \cdots \ G_N]$ is denoted as a polynomial with full degree $G$:

$$\tilde{P}_G(X) = \sum_{j_1=0}^{G} \cdots \sum_{j_N=0}^{G} \tilde{w}_{j_1,\cdots,j_N} \prod_{k=1}^{N} x_k^{j_k}.$$

(6.51)
Lemma 6.2 Let $\bar{P}_{G_1}(X)$ and $\bar{P}_{G_2}(X)$ denote two polynomials with full degrees $G_1$ and $G_2$. For a continuous function $f(X) : [0,1]^{1 \times N} \rightarrow [0,1]$, there exist a polynomial $\bar{P}_{G_1}(X)$ that approximates better than the polynomial $\bar{P}_{G_2}(X)$ such that the approximation errors $|e_1| = |f(X) - \bar{P}_{G_1}(X)| < |e_2| = |f(X) - \bar{P}_{G_2}(X)|$, provided that $G_1 > G_2$ and $|e_2| \neq 0$.

Proof. Since $G_1 > G_2$, then the difference between $\bar{P}_{G_1}(X)$ and $\bar{P}_{G_2}(X)$ is

$$\bar{P}_{G_1,G_2} = \bar{P}_{G_1}(X) - \bar{P}_{G_2}(X) = \sum_{j_1=G_2}^{G_1} \cdots \sum_{j_N=G_2}^{G_1} \bar{w}_{j_1,\ldots,j_N} \prod_{k=1}^{N} x_k^{j_k}. \quad (6.52)$$

Since the values of $X$ belong to closed intervals, then there exist a $\varepsilon > 0$ such that $-\varepsilon \leq \bar{P}_{G_1}(X) - \bar{P}_{G_2}(X) \leq \varepsilon$. There are three cases of $e_2$, which are $e_2 = 0$, $e_2 > 0$ and $e_2 < 0$. The case of $e_2$ is not considered here.

If $e_2 > 0$, there exists a $\bar{P}_{G_1,G_2}$ such that $0 < \bar{P}_{G_1,G_2} < e_2$. Then, $e_1 = f(X) - \bar{P}_{G_1}(X) = f(X) - \bar{P}_{G_2}(X) - \bar{P}_{G_1,G_2} = e_2 - \bar{P}_{G_1,G_2}$. We then have $0 < e_1 < e_2$ and $|e_1| < |e_2|$.

If $e_2 < 0$, there exists a $\bar{P}_{G_1,G_2}$ such that $-e_2 < \bar{P}_{G_1,G_2} < 0$. Then, $e_1 = f(X) - \bar{P}_{G_1}(X) = f(X) - \bar{P}_{G_2}(X) - \bar{P}_{G_1,G_2} = e_2 - \bar{P}_{G_1,G_2}$. We then have $e_2 < e_1 < 0$ and $|e_1| < |e_2|$.

Therefore, there exist a polynomial $\bar{P}_{G_1}(X)$ that approximates better than the polynomial $\bar{P}_{G_2}(X)$ such that the approximation errors $|e_1| < |e_2|$, provided that $G_1 > G_2$ and $|e_2| \neq 0$.

The proof is thus completed.

Based on Lemma 6.2 as $G$ gets larger, a polynomials with full degree $G$ can approximate a given target function better. Therefore, the minimal approximation error achieved by a polynomials with full degree $G$ is also defined as the approximation error with degree $G$.

Theorem 6.2 For a continuous function $f(Q_1) : [0,1]^{1 \times N} \rightarrow [0,1]$ and the desired approximation error with degree $G$, the approximation error of the multi-layer RNN can converge towards the desired degree exponentially with respect to the number of layers $L$. 

Proof. Based on the analysis in Section 6.2 and (6.46), let us recall the output of the MLRNN $q_{1,L}(W^+, W^-, Q_1)$:

$$q_{1,L} \approx \sum_{j_1=0}^{2^{L-1}} \cdots \sum_{j_{N_1}=0}^{2^{L-1}} \bar{w}^{(1)}_{j_1, \cdots, j_{N_1}} \prod_{k=1}^{N_1} q_{k,1}^k.$$  \hfill (6.53)

This is a polynomial with full degree $2^{L-1}$. As the MLRNN goes deeper, i.e., the number of layers $L$ increases linearly, the full degree $2^{L-1}$ of the output increases exponentially. Therefore, the approximation error of the MLRNN can converge towards the desired approximation error with degree $G$ exponentially.

The proof is thus completed.  \hfill \blacksquare

6.3.3 High-Level Representation Extraction

By solving the deep-learning problem of the MLRNN described by (6.7), a deeper layer in the MLRNN is capable of extracting the higher-level representations from the raw input data and the higher-level relationship between the inputs and desired outputs.

The cost function in (6.7) contains the terms of the 2nd raw moments of the output of the MLRNN, i.e., $E[q_{i,L}^2]$ and the 1st moment of $q_{i,L}y_i$, i.e., $E[q_{i,L}y_i]$. Let us recall the expression of $E[q_{i,L}y_i]$ (6.39) in Section 6.2:

$$E[q_{i,L}y_i] \approx P \left[ E[y_i], E[q_{1,1}y_i], \cdots, E[q_{N_1,1}y_i], E[q_{1,1}^2], \cdots, E[q_{N_1,1}^2] \right],$$  \hfill (6.54)

and the expression of $E[q_{i,L}^2]$ in (6.41)

$$E[q_{i,L}^2] \approx P \left[ E[q_{1,1}], \cdots, E[q_{N_1,1}], E[q_{1,1}^2], \cdots, E[q_{N_1,1}^2] \right].$$  \hfill (6.55)

These are polynomials of the raw moments of different orders with respected to the inputs and
6.3. Theorem and Hypothesis

In the $l$-th layer output of the MLRNN, the highest order of the raw moments extracted from the inputs is $2^l$. As the number of layer increases to $l + 1$, the higher-level representations, which are the $(2^l + 1)$-th, $2^{l+2}$-th, \ldots, $2^{l+1}$-th raw moments of the inputs, are extracted and exploited in the solving process of the deep-learning problem. In addition, the raw moments of $\mathbb{E}[q_{1,1}y_i], \ldots, \mathbb{E}[q_{N,1}y_i], \mathbb{E}[q_{1,1}^2y_i], \ldots, \mathbb{E}[q_{N,1}^2y_i]$ describing the relation between the inputs and desired outputs are also extracted and exploited in the solving process.

Therefore, the MLRNN is capable of extracting higher-level representations from the raw input data layer by layer, which are the raw moments of different orders of the inputs. As the MLRNN goes deeper, the highest order of raw moments extracted increases exponentially; and the higher-lever relationship between the inputs and desired outputs can be exploited in the solving process of the deep learning problem (6.7).

6.3.4 Moment Learning Hypothesis

Based on the analysis in Section 6.2 and (6.39), we may have

$$
\mathbb{E}[q_{i,L}(W^+, W^-, Q_1)y_i] \cong \mathbb{P}[W^+, W^-, \mathbb{E}[q_{1,1}], \ldots, \mathbb{E}[q_{N,1}], \mathbb{E}[q_{1,1}^2], \ldots, \mathbb{E}[q_{N,1}^2]],
$$

\begin{equation}
\cdots, \mathbb{E}[q_{N,1}^2y_i], \cdots, \mathbb{E}[q_{1,1}^{2L-1}y_i], \cdots, \mathbb{E}[q_{N,1}^{2L-1}y_i].
\end{equation}

(6.56)

Similarly, based on the analysis in Section 6.2 and (6.41), we may have

$$
\mathbb{E}[q_{i,L}^2(W^+, W^-, Q_1)] \cong \mathbb{P}[W^+, W^-, \mathbb{E}[q_{1,1}], \cdots, \mathbb{E}[q_{N,1}], \mathbb{E}[q_{1,1}^2], \cdots, \mathbb{E}[q_{N,1}^2],
$$

\begin{equation}
\cdots, \mathbb{E}[q_{1,1}^{2L}], \cdots, \mathbb{E}[q_{N,1}^{2L}]].
\end{equation}

(6.57)
Then, the cost function in (6.7) could be written as

$$
\sum_{i=1}^{N_l} (\mathbb{E} \left[ q_i^2(W^+, W^-, X) \right] - 2 \mathbb{E} \left[ q_i(W^+, W^-, X) y_i \right] + \mathbb{E} \left[ y_i^2 \right])
$$

$$
\cong \mathbb{P} \left[ W^+, W^-, \mathbb{E} \left[ y_1 \right], \ldots, \mathbb{E} \left[ y_{N_1} \right], \mathbb{E} \left[ y_1^2 \right], \ldots, \mathbb{E} \left[ y_{N_1}^2 \right], \mathbb{E} \left[ q_{1,1} \right], \ldots, \mathbb{E} \left[ q_{N_1,1} \right], \mathbb{E} \left[ q_1^2 \right], \ldots, \mathbb{E} \left[ q_{N_1,1}^2 \right], \mathbb{E} \left[ q_{1,1}^2 \right], \ldots, \mathbb{E} \left[ q_{N_1,1}^2 \right], \ldots, \mathbb{E} \left[ q_{1,1}^{2L-1} y_i \right], \ldots, \mathbb{E} \left[ q_{N_1,1}^{2L-1} y_i \right] \right].
$$

(6.58)

Therefore, the deep learning problem of the MLRNN described by (6.7) may be transformed into the following optimization problem:

$$
\min_{W^+, W^-} \mathbb{P} \left[ W^+, W^-, \mathbb{E} \left[ y_1 \right], \ldots, \mathbb{E} \left[ y_{N_1} \right], \mathbb{E} \left[ y_1^2 \right], \ldots, \mathbb{E} \left[ y_{N_1}^2 \right], \mathbb{E} \left[ q_{1,1} \right], \ldots, \mathbb{E} \left[ q_{N_1,1} \right], \mathbb{E} \left[ q_1^2 \right], \ldots, \mathbb{E} \left[ q_{N_1,1}^2 \right], \mathbb{E} \left[ q_{1,1}^2 \right], \ldots, \mathbb{E} \left[ q_{N_1,1}^2 \right], \ldots, \mathbb{E} \left[ q_{1,1}^{2L-1} y_i \right], \ldots, \mathbb{E} \left[ q_{N_1,1}^{2L-1} y_i \right] \right],
$$

(6.59)

s.t. \( W^+ \geq 0, W^- \geq 0, \sum_{j=1}^{N_i} (w_{i,j,l}^+ + w_{i,j,l}^-) \leq r_{i,l}, \) with \( l = 1, 2, \ldots, L - 1, \)

in which the elements \( \mathbb{E} \left[ y_i \right], \mathbb{E} \left[ y_i^2 \right], \mathbb{E} \left[ q_{1,1} \right], \mathbb{E} \left[ q_1^2 \right], \ldots, \mathbb{E} \left[ q_{1,1}^{2L-1} y_i \right], \mathbb{E} \left[ q_{1,1}^{2L-1} y_i \right], \ldots, \mathbb{E} \left[ q_{N_1,1}^{2L-1} y_i \right] \) with \( i = 1, 2, \ldots, N_1 \) can be estimated over the available dataset \( S \) before solving the optimization problem. Specifically, in the cases of handling a large dataset, the whole dataset needs to be stored when solving the deep learning problem described by (6.7), which requires a lot of memory space and is challenging to utilize distributed computer sources. In contrast, in the transformed optimization problem (6.59), the elements of moments can be pre-computed by either a centralized computer or multiple computers and are the only data required to solve the problem, rather than storing the whole dataset. Therefore, the transformation into the moment learning problem can potentially reduce the complexity of the original problem, which can be a future direction.
6.4 Conclusions

In this chapter, a theoretical study into the MLRNN for deep learning has been presented. The deep-learning problem of the MLRNN has been mathematically formulated and transformed into a form of moments to facilitate the analysis from the theoretical point of view. Theoretical analyses of the moments of the layer outputs of the MLRNN have been presented. Based on these analyses, the MLRNN has been shown to be a universal function approximator. The effect of the number of layers in the MLRNN on the learning capability has been investigated, showing that the approximation error of the MLRNN can converge towards the desired degree of approximation error exponentially. Further analyses have shown that the MLRNN is capable of extracting high-level representations from the raw input data layer by layer, which are the raw moments of different orders of the inputs. A hypothesis of transforming the deep-learning problem of the MLRNN into a moment-learning problem has been also presented. This hypothesis illustrates that it is possible to solve the deep-learning problem of the MLRNN with only the statistical estimations from the dataset rather than the whole dataset (whose size could be very big), thereby laying a basis for the future work.
Chapter 7

Deep Learning with Standard Random Neural Network

7.1 Introduction

In Chapter 4 the random neural network (i.e., the RNN) was simplified into a quasi-linear feed-forward structure and proposed a deep non-negative autoencoder for unsupervised learning; in Chapter 5 the RNN has been used to model the dense nuclei that can be found in the brain and a simple transfer function can be derived. Based on the dense nuclei, a deep multi-layer architecture (i.e., the MLDRNN) has been proposed for solving supervised-learning problems. This chapter goes back to the original simpler structure of the RNN and investigates the power of single standard RNN cells for deep learning in the following two aspects.

In the first part, we show that, with only positive parameters, the RNN implements convolution operations similar to convolutional neural networks [LBBH98, LBH15, SDBR14]. Our work examines single-cell, twin-cell and cluster approaches that model the positive and negative parts of a convolution kernel as the arrival rates of excitatory and inhibitory spikes to receptive cells. The relationship between the RNN and ReLU activation [GBB11] is also investigated and an approximate ReLU-activated convolution operation is presented.
In the second part, a multi-layer architecture of the standard RNN (MLRNN) is built. The computational complexity of the MLRNN is lower than the MLDRNN model developed in Chapter 5 [GY16a, YG16b], and it can be generalized to handle multi-channel datasets. Numerical results regarding multi-channel classification datasets show that the MLRNN is effective and that it is arguably the most efficient among five different deep-learning approaches.

*Part of this chapter has been published as: Yonghua Yin and Erol Gelenbe. “Single-cell based random neural network for deep learning.” International Joint Conference on Neural Networks (IJCNN), pages 86-93, 2017.*

### 7.1.1 Related Work and Discussion

Since the proposal of the RNN in 1989 [Gel89], there have been numerous investigations into the RNN’s learning algorithms [BK00, Tim10, GLK11a]. In [Gel93], Gelenbe developed a gradient-descent learning algorithm for the RNN, which is the most widely-used RNN learning algorithm. There are various variants based on the gradient-descent RNN learning algorithm. For example, Likas [LS00] utilises quasi-Newton methods and the RNN gradient descent, while Basterrech [BMRS09] exploited the Levenberg-Marquardt optimisation procedure and adaptive momentum. The RPROP learning algorithm is also combined with the gradient-descent RNN learning algorithm [BMRS09]. In addition to the gradient-descent learning algorithm, Georgioupolous [GLK11b] applied the adaptive inertia weight particle swarm optimization (AIW-PSO) [QYSW06] and differential evolution (DE) approach [SP97] for the RNN. Timoth-eou tackled the learning problem of the RNN by approximating it as a non-negative least-square (NNLS) problem [Tim09, Tim08]. Javed [JLAE17] proposed to combine the artificial bee colony (ABC)/particle swarm optimization (PSO) with the sequential quadratic programming (SQP) optimization algorithm to train the RNN. However, there has not been research regarding the property of the RNN for deep learning. The work in Chapter 5 [GY16a, YG16b] connected the dense RNN with deep learning. The work in this chapter focuses on investigating the properties of the standard RNN for deep learning and develop effective and efficient deep-learning tools. Most of the learning algorithms for the RNN depend on the gradient-descent algorithms, where
there are inherent disadvantages of slow convergence and the possibilities of being trapped into poor local minimums. To achieve the speedup in training and stable performance, this work converts the training of a deep RNN system into multiple convex optimization problems that can be solved in a fast speed with good solutions.

7.2 Convolution Operation with RNN: Two Approaches

In this and the next sections, we show that, though with only positive parameters, the RNN is capable of conducting convolution operators that produce similar effects to those in conventional convolutional neural networks [LBH15, SDBR14], by three approaches. The convolutional operation is denoted as \( O = \text{conv}(I, W) \) with \( O, I \) and \( W \) being the convolution outputs, convolution inputs and convolution kernels, where the convolution inputs are generally images.

Let us configure the standard RNN cells and present the procedure for conducting convolution operations using these cells. Suppose a single RNN cell receives excitatory and inhibitory Poisson spike trains from other cells with rates \( x^+ \) and \( x^- \), respectively, and it also receives an excitatory Poisson spike train from the outside world with rate \( \lambda^+ \). The firing rate of the cell is \( r \). Then, based on the RNN theory [Gel89, Gel90, Gel93] and RNN equation (2.1), the probability in the steady state that this cell is excited can be calculated by

\[
q = \min(\frac{\lambda^+ + x^+}{r + x^-}, 1).
\] (7.1)

For notational ease, we define \( \phi(x^+, x^-)|_{\lambda, r} = \min((\lambda + x^+)/(r + x^-), 1) \) and use \( \phi(\cdot) \) as a term-by-term function for vectors and matrices.

7.2.1 Single-Cell Approach

To lay a basis for further illustration, the convolution operation of this approach with single cells is shown schematically in Figure 7.1.
First, as shown in Figure 7.1, the quasi-linear RNN cells (called the LRNN-E cell) presented in Chapter 4 are utilized as input cells for the convolution inputs \( I \). For these input cells, the firing rates are set as 1, the rates of the external inhibitory spikes are set as 0 and the rates of the external excitatory spikes are set as \( I \). Let us assume that \( 0 \leq I \leq 1 \). Then, the stationary excitation probabilities of these input cells are \( I \).

Second, we normalize the convolution kernel to satisfy the RNN probability constraint via \( W \leftarrow W/(\text{sum}(|W|)) \), where the operation \( \text{sum}(\cdot) \) produces the summation of all input elements. The kernel weight matrix is split as \( W^+ = \max(W,0) \geq 0 \) and \( W^- = \max(-W,0) \geq 0 \) to avoid negativity (that is another RNN probability constraint), where term-by-term operation \( \max(a,b) \) produces the larger element between \( a \) and \( b \). As shown in Figure 7.1, the input cells are connected to a set of RNN cells (7.1) in a local connectivity pattern (each RNN cell (7.1) is connected to only a small region of the input cells) and may fire excitatory and inhibitory spikes towards them. The weights \( W^+ \) are utilized as the rates of excitatory spikes from the input cells to the RNN cells (7.1), while the weights \( W^- \) are utilized as the rates of inhibitory spikes. In this case, the stationary excitation probabilities of the RNN cells (7.1) denoted by a matrix \( O \) can be obtained as

\[
O = \phi(\text{conv}(I,W^+),\text{conv}(I,W^-)),
\]

where \( \phi(X,W) \) denotes a standard image convolution operation of input \( X \) with the convolutional kernel \( W \). Or, in the other case, when \( W^- \) and \( W^+ \) are utilized as excitatory and
inhibitory spike rates, the expression of $O$ is

$$O = \phi(\text{conv}(I, W^-), \text{conv}(I, W^+)). \quad (7.3)$$

### 7.2.2 Twin-Cell Approach

The convolution operation of this approach with twin cells is shown schematically in Figure 7.2. The input cells receive external excitatory spikes with rates $I$ and produce $I$. The input cells are then connected with twin arrays constituted of RNN cells (7.1). These two arrays have the same number of cells (7.1): for the upper array, $\lambda^+ = 1, r = 1$; while for the other array, $\lambda^+ = 0, r = 1$. As seen from Figure 7.2, inputs $I$ pass through the twin arrays and produce

$$I_1 = \phi(0, I)|_{\lambda^+ = 1, r = 1} = \frac{1}{1 + I} = 1 - \frac{I}{1 + I}, \quad (7.4)$$

$$I_2 = \phi(I, I)|_{\lambda^+ = 0, r = 1} = \frac{I}{1 + I} = 1 - \frac{1}{1 + I}, \quad (7.5)$$

where $\mathbf{1}$ is an all-one matrix with appropriate dimensions and the division between matrices is element-wise.

Since the probabilities are non-negative and less than 1, the convolution kernel needs to be adjusted via $W \leftarrow W / \max(|\text{conv}(I_1, W)|)$ if $\max(|\text{conv}(I_1, W)|) > 1$. To satisfy the RNN...
probability constraints, the convolution kernel is normalized via $W ← W/\text{sum}(|W|)$ and split into $W^+ = \max(W, 0) \geq 0$ and $W^- = \max(-W, 0) \geq 0$. As seen from Figure 7.2, the cells in the twin arrays are then connected to a set of receptive cells, which are quasi-linear cells presented in Chapter 4. For these receptive cells, the firing rates are set as 1 and they receive excitatory spikes from the outside world with rate $1 - \text{sum}(W^-)$. The stationary excitation probabilities of the receptive cells are then obtained as

$$O = \min(\text{conv}(I_1, W^+) + \text{conv}(I_2, W^-) + 1 - \text{sum}(W^-), 1)$$

$$= \min(\text{conv}(I_1, W^+) + \text{conv}(1, W^-) - \text{conv}(I_1, W^-) + 1 - \text{sum}(W^-), 1) \quad (7.6)$$

$$= \min(\text{conv}(I_1, W) + 1, 1).$$

7.3 Convolution with RNN: a Cluster Approach

This approach is based on a type of multi-cell cluster that approximates the rectified linear unit (ReLU), where the ReLU unit has been widely-used in the deep-learning area [LBH15 GBB11]. The cluster is constituted by the LRNN-E cell [YG16a] and another different quasi-linear RNN cell that is deduced from the first-order approximation of the RNN equation (4.1).

7.3.1 First-Order Approximation of the RNN Equation

The RNN formula (4.1) lends itself to various approximations such as the first-order approximation. Specifically, according to (4.1),

$$q_h = \frac{\lambda_h^+ + \sum_{v=1}^{N} q_v r_v p_{vh}^+}{r_h + \lambda_h^- + \sum_{v=1}^{N} q_v r_v p_{vh}^-}. $$

Let $\Gamma = \lambda_h^- + \sum_{v=1}^{N} q_v r_v p_{vh}^-$. Then,

$$q_h = \frac{\lambda_h^+ + \sum_{v=1}^{N} q_v r_v p_{vh}^+}{r_h + \Gamma}. $$
The partial derivative of $q_h$ with respect to $\Gamma$ is

$$\frac{\partial q_h}{\partial \Gamma} = -\frac{\lambda_h^+ + \sum_{v=1}^{N} q_v r_v p_{vh}^+}{(r_h + \Gamma)^2}.$$

Based on Taylor’s theorem [Wik18b], $q_h$ can be approximated in a neighbourhood $\Gamma = a$ as:

$$q_h \approx q_h \bigg|_{\Gamma = a} + \frac{\partial q_h}{\partial \Gamma} \bigg|_{\Gamma = a} (\Gamma - a) = \frac{\lambda_h^+ + \sum_{v=1}^{N} q_v r_v p_{vh}^+}{r_h + a} - \frac{\lambda_h^- + \sum_{v=1}^{N} q_v r_v p_{vh}^-}{(r_h + a)^2}.$$

Suppose the value of $a$ satisfies $r_h >> a$, then

$$q_h \approx \frac{\lambda_h^+ + \sum_{v=1}^{N} q_v r_v p_{vh}^+}{r_h} - \frac{\lambda_h^- + \sum_{v=1}^{N} q_v r_v p_{vh}^-}{r_h^2}(\Gamma - a) \approx \frac{\lambda_h^+ + \sum_{v=1}^{N} q_v r_v p_{vh}^+}{r_h} (1 - \frac{\Gamma}{r_h} + \frac{a}{r_h})$$

Then, the first order approximation of formula (4.1) is obtained as:

$$q_h \approx \frac{\lambda_h^+ + \sum_{v=1}^{N} q_v r_v p_{vh}^+}{r_h} (1 - \frac{\lambda_h^- + \sum_{v=1}^{N} q_v r_v p_{vh}^-}{r_h}).$$

(7.7)

**Quasi-Linear RNN cell receiving inhibitory spikes**

Let $w_{vh}^+ = r_v p_{vh}^+ = 0$, $w_{vh}^- = r_v p_{vh}^-$, $\lambda_h^+ = 1$, $\lambda_h^- = 0$, $r_h = 1$ and $r_v = 1$. Note that these settings do not offend the condition $r_h >> \lambda_h^- + \sum_{v=1}^{N} q_v r_v p_{vh}^-$, which becomes $1 >> \sum_{v=1}^{N} w_{vh}^- q_v$. Since $r_v = 1$, then $\sum_{h=1}^{N} w_{vh}^- = \sum_{h=1}^{N} p_{vh}^- \leq 1$. The first-order approximation (7.7) is rewritten as:

$$q_h = \frac{1}{1 + \sum_{v=1}^{N} w_{vh}^- q_v} \approx \min(1 - \sum_{v=1}^{N} w_{vh}^- q_v, 1),$$

(7.8)

subjecting to $\sum_{h=1}^{N} w_{vh}^- \leq 1$ and $1 >> \sum_{v=1}^{N} w_{vh}^- q_v$. Since $q_v \leq 1$, we have $\sum_{v=1}^{N} w_{vh}^- q_v \leq \sum_{v=1}^{N} w_{vh}^-$, and we could then relax the condition $1 >> \sum_{v=1}^{N} w_{vh}^- q_v$ to $1 >> \sum_{v=1}^{N} w_{vh}^-$, which is independent of the cell states. This simplified RNN cell receiving inhibitory spikes is quasi-linear, and thus we could call it a LRNN-I cell.
7.3. Convolution with RNN: a Cluster Approach

Figure 7.3: A single unit with ReLU activation.

Figure 7.4: A cluster of the LRNN-I and LRNN-E cells to approximate the ReLU activation.

7.3.2 Relationship between RNN and ReLU

Based on the LRNN-E cell [YG16a] and LRNN-I cell (7.8), we investigate the relationship between the RNN and the ReLU activation function [LBH15, GBB11] and show that a cluster of the LRNN-I and LRNN-E cells produces approximately the ReLU activation.

ReLU Activation

We consider a single unit with ReLU activation. Suppose the input to the unit is a non-negative vector $X = [x_{1:v}] \in \mathbb{R}^{1 \times V}$ in the range $[0, 1]$, and the connecting weights is a vector $W = [w_{v,1}] \in \mathbb{R}^{V \times 1}$ whose elements can be both positive and negative. Then, the output of this unit is described by $\text{ReLU}(XW) = \max(0, XW)$. Figure 7.3 illustrates the corresponding schematic representation of the ReLU.

A Cluster of LRNN-I and LRNN-E Cells

A cluster of the LRNN-I and LRNN-E cells is capable of producing approximately the ReLU activation, whose schematic representation is given in Figure 7.4. For illustration, let $W^+ =$
\[ [w_{v,1}]^+ = \max(0, W) \text{ and } W^- = [w_{v,1}]^- = -\min(0, W). \] It is evident that \( W^+ \geq 0, W^- \geq 0, \)
\( W = W^+ - W^- \) and \( \text{ReLU}(XW) = \max(0, XW^+ - XW^-). \)

First, the input \( X \) is imported into a LRNN-E cell (the 1st cell) with connecting weights \( W^- \).
Then the stationary excitation probability of the 1st cell is
\[ q_1 = \min(XW^-, 1). \]
Let us assume \( XW^- \leq 1 \). Then, \( q_1 = XW^- \).

The input \( X \) is also imported into a LRNN-I cell (the 2nd cell) with \( W^+ \), and its cell excitation probability is
\[ q_2 = \frac{1}{1 + XW^+} \approx \min(1 - XW^+, 1). \]
Based on (7.8), the condition of this approximation is \( \sum_{v=1}^V w_{v,1}^+ << 1 \). Suppose this condition holds, we also have \( q_2 \approx 1 - XW^+ \).

Second, the 1st and 2nd cells of \( q_1 \) and \( q_2 \) are connected to a LRNN-E cell (i.e., the 3rd cell or the output cell) with connecting weight being 1. The stationary excitation probability of the 3rd/output cell is
\[ q_3 = \min(q_2 + q_1, 1) = \min(\frac{1}{1 + XW^+} + XW^-, 1) \approx \min(1 - XW^+ + XW^-, 1). \]
In \( q_3 \), the information where \( 1 - XW^+ + XW^-= 1-XW \, \text{or} \, XW > 1 \) (i.e., \( XW < 0 \)) is removed by the LRNN-E cell. We have
\[ q_3 = \varphi(XW^+, XW^-) \approx 1 - \text{ReLU}(XW), \quad (7.9) \]
where, for notation ease, we define the activation of this cluster as \( \varphi(x^+, x^-) = \min(1/(1 + x^+) + x^-, 1) \) and use \( \varphi(\cdot) \) as a term-by-term function for vectors and matrices. The conditions for the approximation in (7.9) are \( XW^- \leq 1 \) (that can be loosed as \( \sum_{v=1}^V w_{v,1}^- = \text{sum}(W^-) \leq 1 \) if \( X \leq 1 \)) and \( \sum_{v=1}^V w_{v,1}^+ = \text{sum}(W^+) << 1 \).
7.3. Convolution with RNN: a Cluster Approach

7.3.3 Cluster-Based Convolution

The convolution operation with clusters (7.9) is shown schematically in Figure 7.5. As seen from the figure, the input cells receive excitatory spikes with rates $I$ and produce stationary excitation probabilities $I$. Then, input cells are connected to a set of receptive clusters (7.9) with local connectivity. To satisfy the conditions for the approximation in (7.9), the convolution kernel is normalized via $W \leftarrow W/\text{sum}(|W|)/10$. In addition, let us split the kernel as $W^+ = \max(W, 0) \geq 0$ and $W^- = \max(-W, 0) \geq 0$. By assuming that $0.1 << 1$, it is evident that $\text{sum}(W^+) \leq 0.1 << 1$ and $\text{sum}(W^-) \leq 0.1 << 1$, which satisfy the approximation conditions. Then, the weights $W^+$ and $W^-$ are utilized respectively as the rates of the excitatory and inhibitory spikes from the input cells to the clusters. The stationary excitation probabilities of the output cells of the receptive clusters are obtained as

$$O = \varphi(\text{conv}(I, W^+), \text{conv}(I, W^-)) \approx 1 - \text{ReLU}(\text{conv}(I, W)). \quad (7.10)$$
7.4 Numerical Verification of RNN Convolution

This section conducts numerical experiments on images (implemented in Theano [The16]) to verify the three approaches in Sections 7.2 and 7.3 for adapting the convolution structure into the RNN. The convolution kernels used in the experiments are obtained from the first convolution layer of the pre-trained GoogLeNet [MOD] provided by [VL15], while the image is obtained from [RDS+15], shown in Figure 7.6. The gray-inverted output of a standard convolution operation with the ReLU activation is given in Figure 7.7.

**Single-cell approach:** The output of the convolution is given in Figure 7.8 from which we can see that it is similar to that in Figure 7.7.

**Twin-cell approach:** The output of the convolution is given in Figure 7.9. This approach also produces the edge-detection effect similar to the standard convolution.

**Cluster approach:** The output of the convolution is given in Figure 7.10 which is more similar to those in Figure 7.7 than that in Figure 7.8.

These results demonstrate well the feasibility of the RNN for convolution operations via these approaches.
7.5 Individual-Cell Based Multi-Layer RNN

Chapter 5 (published in [GY16a, YG16b]) presented a mathematical model of dense clusters with dense soma-to-soma interactions, based on which multi-layer architectures (i.e., the ML-DRNN and MCMLDRNN) are built up that was shown to be more efficient than conventional deep-learning tools on both image and time-series datasets. Striving into efficiency, this section presents a multi-layer architecture of individual RNN cells described in (7.1) (called the MLRNN). The MLRNN is then adapted to handle multi-channel datasets (MCMLRNN). The proposed MLRNN and MCMLRNN achieve similar performance as the MLDRNN and MCML-DRNN presented in Chapter 5 but cost less computation time.

7.5.1 Mathematical Model Formulation in Scalar-Form

In this chapter, we present a multi-layer architecture constituted of individual RNN cells (MLRNN), where the schematic representation is given in Figure 7.11. The MLRNN has $L + 2$ layers, where the $l$th ($1 \leq l \leq L + 2$) layer has $N_l$ cells.

1) The first layer is the external-source layer (or say, the input layer), where each cell receives an excitatory spike train from an external input source. In addition, the cells may fire excitatory and inhibitory spikes towards the cells in the next layer.
2) The successive $L$ layers are hidden layers composed of individual RNN cells that receive both excitatory and inhibitory spike trains from the outside world and cells in the previous layer. Correspondingly, the cells may fire spikes towards the external world and the cells in the next layer.

3) The last layer is the output layer, where the RNN cells receive spikes from the outside world and cells in the previous layer and may fire spikes towards the outside world.

Let us define the following notations for the MLRNN.

1) Let $x_{n_1}$ denote the rate of the excitatory spike train from the $n_1$th ($1 \leq n_1 \leq N_1$) external input source.

2) Let $q_{n_l}$ ($1 \leq n_l \leq N_l$) denote the stationary excitation probability of the $n_l$th cell in the $l$th layer ($1 \leq l \leq L + 2$).

3) Let $r_{n_l}$ ($1 \leq n_l \leq N_l$) denote the firing rate of the $n_l$th cell in the $l$th ($1 \leq l \leq L + 2$) layer. For simplicity, set $r_{n_{L+2}} = 0$ in the output layer.

4) Let $w^+_{n_l,n_{l+1}} = p^+_{n_l,n_{l+1}} r_{n_l}$ and $w^-_{n_l,n_{l+1}} = p^-_{n_l,n_{l+1}} r_{n_l}$ denote excitatory and inhibitory connecting weights between the $n_l$th cell in the $l$th layer and $n_{l+1}$th cell in the $(l+1)$th layer with $l = 1, \cdots, L + 1$, where $p^+_{n_l,n_{l+1}}$ and $p^-_{n_l,n_{l+1}}$ denote respectively the probabilities of excitatory and inhibitory spikes when the $n_l$th cell in the $l$th layer fires. In addition, let $d_{n_l}$
7.5. Individual-Cell Based Multi-Layer RNN

Figure 7.10: Output of an RNN convolution operation with the cluster approach.

Figure 7.11: Schematic representation of the MLRNN.

denote the escape probability of a firing spike. Then, \( \sum_{n_{l+1}=1}^{N_{l+1}} (p_{n_l,n_{l+1}}^+ + p_{n_l,n_{l+1}}^-) + d_{n_l} = 1 \) and \( \sum_{n_{l+1}=1}^{N_{l+1}} (w_{n_l,n_{l+1}}^+ + w_{n_l,n_{l+1}}^-) + d_{n_l}r_{n_l} = r_{n_l} \). Further, \( \sum_{n_{l+1}=1}^{N_{l+1}} (w_{n_l,n_{l+1}}^+ + w_{n_l,n_{l+1}}^-) \leq r_{n_l} \).

5) Let \( \lambda_{n_l}^+ \) and \( \lambda_{n_l}^- \) denote the rates of excitatory and inhibitory spikes from the outside world to the \( n_l \)th cell in the \( l \)th-layer (1 \( \leq l \leq L + 2 \)). For the input layer, set \( \lambda_{n_1}^+ = \lambda_{n_1}^- = 0 \).

Based on the spiking RNN theory [Gel89, Gel93], the stationary excitation probabilities of the
MLRNN can be obtained as:

\[
q_{n_l} = \min \left( \frac{x_{n_l}}{r_{n_l}}, 1 \right), \\
q_{n_l} = \min \left( \frac{\lambda_{n_l}^+ + \sum_{n_{l-1}=1}^{N_{l-1}} q_{n_{l-1}} w_{n_{l-1}, n_l}^+}{r_{n_l} + \lambda_{n_l}^- + \sum_{n_{l-1}=1}^{N_{l-1}} q_{n_{l-1}} w_{n_{l-1}, n_l}^-}, 1 \right),
\]

(7.11)

where \( l = 2, \cdots, L + 2 \). The learning parameters (weights and bias) in the MLRNN could be \( r_{n_l}, \lambda_{n_l}^+, \lambda_{n_l}^-, w_{n_l,n_{l+1}}^+ \) and \( w_{n_l,n_{l+1}}^- \), where the constraints are \( r_{n_l}, \lambda_{n_l}^+, \lambda_{n_l}^-, w_{n_l,n_{l+1}}^+, w_{n_l,n_{l+1}}^- \geq 0 \) and \( \sum_{n_{l+1}=1}^{N_{l+1}} (w_{n_l,n_{l+1}}^+ + w_{n_l,n_{l+1}}^-) \leq r_{n_l} \).

### 7.5.2 Mathematical Model Formulation in Matrix-Form

Suppose there is a dataset represented by a non-negative matrix \( X = [x_{d,n_l}] \in \mathbb{R}_{\geq 0}^{D \times N_1} \), where \( D \) is the number of instances, each instance has \( N_1 \) attributes and \( x_{d,n_l} \) is the \( n_l \)th attribute of the \( d \)th instance. By import \( X \) into the MLRNN, a matrix-form description of the MLRNN (7.11) can be obtained.

1) Let \( q_{d,n_l} \) denote the value of \( q_{n_l} \) for the \( d \)th instance. Let a matrix \( Q_l = [q_{d,n_l}] \in \mathbb{R}_{\geq 0}^{D \times N_1} \) (\( 1 \leq l \leq L + 2 \)) denote the value matrix of \( q_{d,n_l} \).

2) Let a matrix \( R_l = [r_{d,n_l}] \in \mathbb{R}_{\geq 0}^{D \times N_1} \) denote the firing-rate matrix for cells in the \( l \)th layer (\( 1 \leq l \leq L + 2 \)). Note that, the fire rate of a cell is the same for all instances, i.e., \( r_{d_1,n_l} = r_{d_2,n_l} \) for \( 1 \leq d_1, d_2 \leq D \).

3) Let two matrices \( W_l^+ = [w_{n_l,n_{l+1}}^+] \in \mathbb{R}_{\geq 0}^{N_l \times N_{l+1}} \) and \( W_l^- = [w_{n_l,n_{l+1}}^-] \in \mathbb{R}_{\geq 0}^{N_l \times N_{l+2}} \) denote excitatory and inhibitory connecting weight matrices between the \( l \)th and \((l + 1)\)th layers for \( l = 1, \cdots, L + 1 \).

4) Let two matrices \( \Lambda_l^+ = [\lambda_{d,n_l}^+] \in \mathbb{R}_{\geq 0}^{D \times N_1} \) and \( \Lambda_l^- = [\lambda_{d,n_l}^-] \in \mathbb{R}_{\geq 0}^{D \times N_1} \) denote the external arrival rate matrices of excitatory and inhibitory spikes for the \( l \)th layer (\( 1 \leq l \leq L + 2 \)). Note also that \( \lambda_{d_1,n_l}^+ = \lambda_{d_2,n_l}^+ \) and \( \lambda_{d_1,n_l}^- = \lambda_{d_2,n_l}^- \) for \( 1 \leq d_1, d_2 \leq D \).

We may write \( r_{d,n_l}, \lambda_{d,n_l}^+, \lambda_{d,n_l}^- \) respectively as \( r_{n_l}, \lambda_{n_l}^+, \lambda_{n_l}^- \) in the rest of the chapter. In the
matrix form, the excitation probability matrix of the MLRNN can be described as:

\[ Q_1 = \min \left( \frac{X}{R_1}, 1 \right) \]

\[ Q_l = \min \left( \frac{\Lambda_l^+ + Q_{l-1}W_{l-1}^+}{R_l + \Lambda_l^- + Q_{l-1}W_{l-1}^-}, 1 \right) \]

where \( l = 2, \cdots, L + 2 \) and the division operation between matrices is element-wise. In addition, \( Q_1 \in [0, 1]^{D \times N_1} \) is the 1st layer output matrix, \( Q_l \in [0, 1]^{D \times N_l} \) is the \( l \)th layer output (\( l = 2, \cdots, L + 1 \)) and \( Q_{L+2} \in [0, 1]^{D \times N_{L+2}} \) (or written as \( Q_{L+2}(X) \)) is the final MLRNN output matrix. The parameters in the MLRNN are required to satisfy the RNN constraints: \( R_1, \cdots, R_{L+2} \geq 0 \); \( \Lambda_2^+, \cdots, \Lambda_{L+2}^+ \geq 0 \); \( \Lambda_2^-, \cdots, \Lambda_{L+2}^- \geq 0 \); \( W_1^+, \cdots, W_{L+1}^+ \geq 0 \); \( W_1^-, \cdots, W_{L+1}^- \geq 0 \) and \( \sum_{n_{l+1}=1}^{N_{l+1}} (w_{n_l,n_{l+1}}^+ + w_{n_l,n_{l+1}}^-) \leq r_{n_l} \) with \( l = 1, \cdots, L + 1 \).

### 7.5.3 Learning Problem Description

Suppose there is a labelled dataset with \( D \) instances \( \{(X, Y)\} \), where \( X = [x_{d,n_1}] \in \mathbb{R}_{\geq 0}^{D \times N_1} \) is the input-attribute matrix and \( Y = [y_{d,n_{L+2}}] \in [0, 1]^{D \times N_{L+2}} \) is the desired-output matrix. For the \( d \)th instance \( (x_d, y_d) \), the input-attribute vector is denoted by \( x_d = [x_{d,1} \ x_{d,2} \ \cdots \ x_{d,N_1}] \in \mathbb{R}^{1 \times N_1} \) and the desired-output vector is denoted by \( y_d = [y_{d,1} \ y_{d,2} \ \cdots \ y_{d,N_{L+2}}] \in [0, 1]^{1 \times N_{L+2}} \). The mapping from \( X \) to \( Y \) is \( f : \mathbb{R}_{\geq 0}^{2 \times N_1} \to [0, 1]^{N_{L+2}} \) and \( Y = f(X) \).

The objective is to use the MLRNN to learn the input-output mapping \( f \). In other term, given input \( x_d \), the output of the MLRNN \( Q_{L+2}(x_d) : \mathbb{R}_{\geq 0}^{1 \times N_1} \to [0, 1]^{1 \times N_{L+2}} \) should be a reliable and meaningful estimate of the desired output \( y_d \). This can be achieved by selecting appropriate firing rates \( R_1, \cdots, R_{L+2} \geq 0 \), excitatory-spike arrival rates \( \Lambda_2^+, \cdots, \Lambda_{L+2}^+ \geq 0 \), inhibitory-spike arrival rates \( \Lambda_2^-, \cdots, \Lambda_{L+2}^- \geq 0 \) and connecting weight matrices \( W_1, \cdots, W_{L+1} \geq 0 \).

### 7.5.4 Training Procedure for MLRNN

The MLRNN considered here has an input layer, \( L \) hidden layers and an output layer (\( L + 2 \) layers in total). Note that the number of hidden cells (i.e., \( N_{L+1} \)) in the (\( L + 1 \))th lay-
er needs to be a multiple of 2, which will be explained later in the following subsections. The training/configuration procedure in this subsection is to find appropriate values for parameters $R_1$ and $R_l, \Lambda_l^+, \Lambda_l^-, W_{l-1}^+, W_{l-1}^-$ with $l = 2, \ldots, L + 2$ that satisfy the RNN constraints, so that the MLRNN learns the given dataset \{(X,Y)\}. It is worth pointing out here that the parameters found by the training procedure are required to satisfy the RNN constraints:

\[
R_1, \ldots, R_{L+2} \geq 0; \quad \Lambda_1^+, \ldots, \Lambda_{L+2}^+ \geq 0; \quad \Lambda_1^-, \ldots, \Lambda_{L+2}^- \geq 0; \quad W_1^+, \ldots, W_{L+1}^+ \geq 0; \quad W_1^-, \ldots, W_{L+1}^- \geq 0; \quad \sum_{n_{l+1}}^{N_{l+1}} (w_{n_{l+1}}^+ + w_{n_{l+1}}^-) \leq r_{d,n_l} \text{ with } l = 1, \ldots, L + 1.
\]

The configuration procedure is presented in the following steps:

**Step 1.** Configure $\Lambda_l^-$ with $l = 2, \ldots, L + 2, W_{L+1}^-, R_1$.

**Step 2.** Configure $W_l^+, W_l^-, \Lambda_{l+1}^+, R_{l+1}$ with $l = 1, \ldots, L - 1$.

**Step 3.** Configure $W_L^+, W_L^-, \Lambda_{L+1}^+, R_{L+1}, W_{L+1}^+, \Lambda_{L+2}^+, R_{L+2}$.

**Step 1. Configure $\Lambda_l^-$ with $l = 2, \ldots, L + 2, W_{L+1}^-$, $R_1$**

Let us set $\Lambda_l^- \leftarrow 0$ for $l = 2, \ldots, L + 2, W_{L+1}^- \leftarrow 0$ and $R_1 \leftarrow 1$. Since the firing rates of the cells in the input layer are 1 and there is no inhibitory spike from the outside world, then the cell activation (stationary excitation probability) in the input layer is quasi-linear (as illustrated in Chapter 4), which means $Q_1 = \min(X, 1)$ based on (7.12). For notation ease, let us define the individual-RNN-cell activation $\phi(x^+, x^-)|_{\lambda,r} = \min((\lambda + x^+)/(r + x^-), 1)$ and use $\phi(\cdot)$ as a term-by-term function for vectors and matrices. Then, the system of equations (7.12) of the MLRNN can be rewritten as

\[
Q_1 = \min(X, 1), \quad (7.14)
\]

\[
Q_l = \phi(Q_{l-1}W_{l-1}^+, Q_{l-1}W_{l-1}^-)|_{\Lambda_l^+, R_l} \text{ with } 2 \leq l \leq L + 1, \quad (7.15)
\]

\[
Q_{L+2} = \phi(Q_{L+1}W_{L+1}^+, 0)|_{\Lambda_{L+2}^+, R_{L+2}}. \quad (7.16)
\]

To sum up, in this step, the configurations are conducted via $\Lambda_l^- \leftarrow 0$ for $l = 2, \ldots, L + 2$, $W_{L+1}^- \leftarrow 0$ and $R_1 \leftarrow 1$. 
7.5. Individual-Cell Based Multi-Layer RNN

Step 2. Configure $W_i^+, W_i^-, \Lambda_{i+1}^+, R_{i+1}$ with $l = 1, \ldots, L - 1$

First, let us set $W_i^+ \leftarrow 0$ with $l = 1, \ldots, L - 1$. Then, a series of reconstruction problems related to the cell activations of the current layer and $W_i^-$ are constructed and solved for $W_i^-$ with $l = 1, \ldots, L - 1$. Specifically, the reconstruction problem for the weight matrix $W_i^-$ is constructed as

$$\min_{W_i^-} ||X_1 - \sigma(\phi(0, X_1 \bar{W}))_{\max(X_1 \bar{W}), \max(X_1 \bar{W})} W_i^- ||^2 + ||W_i^-||_{\ell_1}, \text{ s.t. } W_i^- \geq 0,$$

(7.17)

where $\bar{W} \geq 0$ is a randomly-generated matrix with appropriate dimensions that satisfies the RNN constraints and operation max($\cdot$) produces the maximal element of its input. Besides, the operation $\sigma(H)$ first maps each column of its input $H$ into $[0, 1]$ linearly, then uses the “zcore” MATLAB operation and finally adds a positive constant to remove negativity. Note that $X_1$ is obtained via $X_1 \leftarrow X$. In addition, the fast iterative shrinkage-thresholding algorithm (FISTA) in Chapter 5, which has been adapted from [BT09] with the modification of setting negative elements in the solution to zero in each iteration, is exploited to solve the reconstruction problem (7.17). After solving the problem, we obtain $W_i^- \geq 0$ and then adjust its values to satisfy the RNN constraints $\sum_{n_2=1}^{N_2} (w_{n_1,n_2}^+ + w_{n_1,n_2}^-) = \sum_{n_2=1}^{N_2} w_{n_1,n_2}^- \leq 1$. If $\sum_{n_2=1}^{N_2} w_{n_1,n_2}^- > 1$, the operation $w_{n_1,n_2}^- \leftarrow w_{n_1,n_2}^- / \left( \sum_{n_2=2}^{N_2} w_{n_1,n_2}^- \right)$ with $n_2 = 1, \ldots, N_2$ can be used to guarantee that the weights satisfy the RNN constraints. Then, we set the external arrival rates and firing rates of the cells in the next layer via $\Lambda_{i+1}^+ \leftarrow \max(X_1 W_i^-)/5$ and $R_{i+1} \leftarrow \max(X_1 W_i^-)/5$.

In sequence, the following reconstruction problem is solved for $W_i^-$ with $l = 2, \ldots, L - 1$ using the modified FISTA:

$$\min_{W_i^-} ||X_l - \sigma(\phi(0, X_l \bar{W}))_{\max(X_l \bar{W}), \max(X_l \bar{W})} W_i^- ||^2 + ||W_i^-||_{\ell_1}, \text{ s.t. } W_i^- \geq 0,$$

(7.18)

where $X_l$ is its layer encodings obtained via $X_l \leftarrow \phi(0, X_{l-1} W_{l-1})_{\Lambda_{l-1}, R_l}$ and $\bar{W} \geq 0$ is randomly generated and adjusted to satisfy the RNN constraints. After solving the problem, the matrix $W_i^- \geq 0$ is obtained; and then its values are adjusted to satisfy the RNN constraints $\sum_{n_{l+1}=1}^{N_{l+1}} (w_{n_l,n_{l+1}}^+ + w_{n_l,n_{l+1}}^-) = \sum_{n_{l+1}=1}^{N_{l+1}} w_{n_l,n_{l+1}}^- \leq r_{n_l}$ by using the operation $w_{n_l,n_{l+1}}^- \leftarrow \cdots$
Then, the output weights \( \bar{w}_{n_l,n_{l+1}}/\left(\sum_{n_{l+1}=1}^{N_{l+1}} w_{n_l,n_{l+1}}/r_{n_l}\right) \) with \( n_{l+1} = 1, \cdots, N_{l+1} \). Then, the external arrival rates and firing rates of the cells in the next layer are set via \( \Lambda_{l+1}^+ \leftarrow \max(X_lW_l^-)/5 \) and \( R_{l+1} \leftarrow \max(X_lW_l^-)/5 \) with \( l = 2, \cdots, L - 1 \).

To sum up, in this step, the configurations are conducted as follows: 1) \( W_l^+ \leftarrow 0 \) with \( l = 1, \cdots, L - 1 \); 2) \( W_l^- \) with \( l = 1, \cdots, L - 1 \) are configured by solving reconstruction problems (7.17) and (7.18); 3) \( \Lambda_{l+1}^+ \), \( R_{l+1} \) are configured via \( \Lambda_{l+1}^+ \leftarrow \max(X_lW_l^-)/5 \) and \( R_{l+1} \leftarrow \max(X_lW_l^-)/5 \) with \( l = 1, \cdots, L - 1 \).

**Step 3. Configure** \( W_L^+, W_L^-, \Lambda_{L+1}^+, R_{L+1}, W_{L+1}^+, \Lambda_{L+2}^+, R_{L+2} \)

A single-hidden-layer RNN-based artificial neural network (SLANN) is first constructed using the ELM concept [KZH13, TDH16] (with \( N_{L+1}/2 \) hidden units and \( N_{L+2} \) output units); and then its weights are utilized to configure \( W_L^+, W_L^-, \Lambda_{L+1}^+, R_{L+1}, W_{L+1}^+, \Lambda_{L+2}^+, R_{L+2} \).

For this SLANN whose activation function is \( \tilde{\phi}(x)|_\alpha = \alpha/(\alpha + x) \) with parameter \( \alpha > 0 \) to be determined, the input and output weights are denoted by \( \bar{W}_1 = [\bar{w}_{n_L, \bar{n}_1}] \in \mathbb{R}^{N_L \times (N_{L+1}/2)} \) and \( \bar{W}_2 = [\bar{w}_{\bar{n}_1, n_{L+2}}] \in \mathbb{R}^{(N_{L+1}/2) \times N_{L+2}} \). Let \( X_L \) denote the \( L \)th-layer output of the MLRNN and \( Y \) is the desired output (or say, labels corresponding to the training dataset). Based on \( W_{L-1}^- \) configured in Step 2, we have \( X_L \leftarrow \phi(0, X_{L-1}W_{L-1})|_{\Lambda_{L+1}^+, R_{L+1}} \). With \( X_L \) imported into the SLANN, a forward pass can be described as \( \bar{O}(X_L) = \tilde{\phi}(X_L\bar{W}_1)|_{\alpha}\bar{W}_2 : \mathbb{R}^{D \times N_L} \rightarrow \mathbb{R}^{D \times N_{L+2}} \).

The input weights \( \bar{W}_1 \) are randomly generated in range \([0, 1]\) and then normalized to satisfy the RNN constraints \( 2 \sum_{\bar{n}_1=1}^{N_{L+1}/2} \bar{w}_{n_L, \bar{n}_1} \leq r_{n_L} \) by using the operation \( \bar{w}_{n_L, \bar{n}_1} \leftarrow \bar{w}_{n_L, \bar{n}_1}/\left(2 \sum_{\bar{n}_1=1}^{N_{L+1}/2} \bar{w}_{n_L, \bar{n}_1}\right) \) with \( n_L = 1, \cdots, N_L \). The parameter \( \alpha \) in the activation function is set by \( \alpha \leftarrow \max(X_L\bar{W}_1)/5 \). Then, the output weights \( \bar{W}_2 \) are determined using the Moore-Penrose pseudo-inverse [GY16a, ZYG+14a, YZ12, ZYY+12, KZH13, TDH16] (denoted by “pinv”) as:

\[
\bar{W}_2 \leftarrow \text{pinv}\left(\phi(X_L\bar{W}_1)|_{\alpha}\right)Y.
\] (7.19)

Then, we adjust \( \bar{W}_2 \) via \( \bar{W}_2 \leftarrow \bar{W}_2/\text{sum}(|\bar{W}_2|) \) to make the summation of all elements in \( |\bar{W}_2| \) no larger than 1, where \( \text{sum}(\cdot) \) produces the summations of all elements of its input.
The following shows how to configure $W_L^+, W_L^-$, $\Lambda_{L+1}^+$, $R_{L+1}$, $W_{L+1}^+$, $\Lambda_{L+2}^-$, $R_{L+2}$ in the MLRNN by using $\hat{W}_1$, $\alpha = \max(X_l\hat{W}_1)/5$ and $\tilde{W}_2$ in the SLANN. For illustration ease, let us define $\hat{\phi}(x) = x/(\alpha + x)$. It is evident that $\hat{\phi}(x) = 1 - \hat{\phi}(x)$. Let $\hat{W}_2^+ = \max(\tilde{W}_2, 0) \geq 0$ and $\hat{W}_2^- = \max(-\tilde{W}_2, 0) \geq 0$. Since the output matrix of the SLANN is $\hat{O}(X_L) = [\hat{o}_{d,n_{L+2}}] = \hat{\phi}(X_l\hat{W}_1)|_\alpha \hat{W}_2 \in \mathbb{R}^{D \times N_{L+2}}$, then the $n_{L+2}$th $(n_{L+2} = 1, \cdots, N_{L+2})$ output of this SLANN for the $d$th instance is

$$
\tilde{o}_{d,n_{L+2}} = \sum_{\hat{n}_1 = 1}^{N_{L+1}/2} \left( (\hat{\phi}(X\hat{W}_1)|_\alpha)_{d,\hat{n}_1}(\hat{W}_2^+\hat{n}_1,n_{L+2}) \right) - \sum_{\hat{n}_1 = 1}^{N_{L+1}/2} (\hat{\phi}(X\hat{W}_1)|_\alpha)_{d,\hat{n}_1}(\hat{W}_2^-\hat{n}_1,n_{L+2})
$$

$$
= \sum_{\hat{n}_1 = 1}^{N_{L+1}/2} (\hat{\phi}(X\hat{W}_1)|_\alpha)_{d,\hat{n}_1}(\hat{W}_2^+\hat{n}_1,n_{L+2}) + \sum_{\hat{n}_1 = 1}^{N_{L+1}/2} (\hat{\phi}(X\hat{W}_1)|_\alpha)_{d,\hat{n}_1}(\hat{W}_2^-\hat{n}_1,n_{L+2}) - \sum_{\hat{n}_1 = 1}^{N_{L+1}/2} (\hat{W}_2^-\hat{n}_1,n_{L+2}).
$$

Since $\hat{\phi}(X\hat{W}_1)|_\alpha = \phi(0,X\hat{W}_1)|_{\alpha,\alpha}$ and $\hat{\phi}(X\hat{W}_1)|_\alpha = \phi(X\hat{W}_1,X\hat{W}_1)|_{0,\alpha}$, then

$$
\tilde{o}_{d,n_{L+2}} = \sum_{\hat{n}_1 = 1}^{N_{L+1}/2} (\phi(0,X\hat{W}_1)|_{\alpha,\alpha})_{d,\hat{n}_1}(\hat{W}_2^+\hat{n}_1,n_{L+2})
$$

$$
+ \sum_{\hat{n}_1 = 1}^{N_{L+1}/2} (\phi(X\hat{W}_1,X\hat{W}_1)|_{0,\alpha})_{d,\hat{n}_1}(\hat{W}_2^-\hat{n}_1,n_{L+2}) - \sum_{\hat{n}_1 = 1}^{N_{L+1}/2} (\hat{W}_2^-\hat{n}_1,n_{L+2}).
$$

For the MLRNN, let us configure $\Lambda_{L+2}^+ = [\lambda_{d,n_{L+2}}^+] \in \mathbb{R}^{D \times N_{L+2}}$ in the $(L + 2)$th layer via $\lambda_{d,n_{L+2}}^+ \leftarrow \max(\text{sum}(\hat{W}_2^-, 1)) - \sum_{\hat{n}_1 = 1}^{N_{L+1}/2}(\hat{W}_2^-)\hat{n}_1,n_{L+2}$ for $d = 1, \cdots, D$, where operation $\text{sum}(\hat{W}_2^-, 1)$ produces the summation vector of each column in matrix $\hat{W}_2^-$. The configurations of $\Lambda_{L+1}^+ = [\lambda_{d,n_{L+1}}^+] \in \mathbb{R}^{D \times N_{L+1}}$ and $R_{L+1} = [r_{d,n_{L+1}}] \in \mathbb{R}^{D \times N_{L+1}}$ are divided into two parts. From the 1st to $(N_{L+1}/2)$th hidden cells in the $(L + 1)$th layer, $\lambda_{d,n_{L+1}}^+ \leftarrow \alpha$ and $r_{d,n_{L+1}} \leftarrow \alpha$ with $n_{L+1} = 1, \cdots, N_{L+1}/2$; while, for the $(N_{L+1}/2 + 1)$th to $N_{L+1}$th hidden cells, $\lambda_{d,n_{L+1}}^+ \leftarrow 0$ and $r_{d,n_{L+1}} \leftarrow \alpha$ with $n_{L+1} = N_{L+1}/2 + 1, \cdots, N_{L+1}$, where $d = 1, \cdots, D$.

The connecting weights $W_L^+ = [w_{n_{L+1},n_{L+2}}^+] \in \mathbb{R}^{N_{L+1} \times N_{L+2}}$, $W_L^- = [w_{n_{L+1},n_{L+2}}^-] \in \mathbb{R}^{N_{L+1} \times N_{L+2}}$ and $W_{L+1}^+ = [w_{n_{L+1},n_{L+2}}^+] \in \mathbb{R}^{N_{L+1} \times N_{L+2}}$ are configured based on $\hat{W}_1$, $\hat{W}_2^+$ and $\hat{W}_2^-$, which are also divided into two parts.
Algorithm 7 Training procedure for the MLRNN

Get data matrix $X$ and label matrix $Y$

$r_1 \leftarrow 1$

for $l = 1, \ldots, L - 1$ do

$W_l^+ \leftarrow 0$

solve Problem (7.17) or (7.18) for $W_l^-$ with input $X$

$e \leftarrow \max(\text{sum}(W_l^-, 2))$

if $e > r_l$

$W_l^- \leftarrow W_l^-/(e/r_l)$

$l_{l+1} \leftarrow \max(XW_l^-)/5$

$r_{l+1} \leftarrow l_{l+1}$

$X \leftarrow \phi(XW_l^+, XW_l^-)|_{l_{l+1}, r_{l+1}}$

determine $\bar{W}_1, \bar{W}_2$ of a SLANN with input $X$, label $Y$

map $\bar{W}_1, \bar{W}_2$ to $W_L^+, W_L^-, W_{L+1}^+, \Lambda_{L+1}^+, \Lambda_{L+2}^+, R_{L+1}, R_{L+2}$

Determine $\bar{W}_1, \bar{W}_2$ of a SLANN with input $X$, label $Y$

map $\bar{W}_1, \bar{W}_2$ to $W_L^+, W_L^-, W_{L+1}^+, \Lambda_{L+1}^+, \Lambda_{L+2}^+, R_{L+1}, R_{L+2}$

First, let us set $w_{nL, n_1}^+ \leftarrow 0$, $w_{nL, n_1}^- \leftarrow \bar{w}_{nL, n_1}$ and $w_{n_1, nL+2}^+ \leftarrow \bar{w}_{n_1, nL+2}$ with $n_L = 1, \ldots, N_L$, $n_1 = 1, \ldots, N_{L+1}/2$ and $n_{L+2} = 1, \ldots, N_{L+2}$.

Then, let us set $w_{nL, n_1}^+ \leftarrow \bar{w}_{nL, n_1}$, $w_{nL, n_1}^- \leftarrow \bar{w}_{nL, n_1}$ and $w_{n_1, nL+2}^+ \leftarrow \bar{w}_{n_1, nL+2}$ with $n_L = 1, \ldots, N_L$, $n_1 = N_{L+1}/2 + 1, \ldots, N_{L+1}$ and $n_{L+2} = 1, \ldots, N_{L+2}$.

Simply put, $W_L^+ \leftarrow [0; \bar{W}_1]$, $W_L^- \leftarrow [\bar{W}_1; \bar{W}_1]$ and $W_{L+1}^+ \leftarrow [\bar{W}_2^+; \bar{W}_2^-]$, where $0$ is an all-zero matrix with appropriate dimensions.

After the configurations, the final output $Q_{L+2} = [q_{d, nL+2}] \in [0 1]^{D \times N_{L+2}}$ of the MLRNN can be obtained as

$$q_{d, nL+2} = \bar{o}_{d, nL+2} + \max(\text{sum}(W_2^-, 1)),$$  \hspace{1cm} (7.20)

with $d = 1, \ldots, D$ and $n_{L+2} = 1, \ldots, N_{L+2}$. The procedure to train the MLRNN is also briefly summarized in Algorithm 7, where operation $\text{sum}(W, 2)$ produces the summation vector of each row in matrix $W$.

### 7.5.5 MCMLRNN: Mathematical Model and Training Procedure

The MLRNN can be adapted to handle multi-channel datasets (denoted as the MCMLRNN), shown in Figure 7.12, where the connecting weights between layers for only Channel-$c$ ($c = 1, \ldots, C$) are $W^+_c, W^-_c \geq 0$ ($l = 1, \ldots, L - 1$), those between the $(L - 1)$th and $L$th hidden layers are $W^+_L, W^-_L \geq 0$ and output weights are $W^+_{L+1} \geq 0$. Besides, a vector $\Lambda^+$ denotes the...
Algorithm 8 Training procedure for the MCMLRNN

Get data matrices $X_c$ ($c = 1, \cdots, C$) and label matrix $Y$

for $l = 1, \cdots, L - 1$ do

for $c = 1, \cdots, C$ do

$W^+_l \leftarrow 0$

solve Problem (7.17) or (7.18) for $W^-_{c,l}$ with input $X_c$

$e \leftarrow \max(\text{sum}(W^-_{c,l}, 2))$

if $e > r_{c,l}$

$W^-_{c,l} \leftarrow W^-_{c,l}/(e/r_l)$

$\lambda^+_{c,l+1} \leftarrow \max(X_cW^-_{c,l})/5$

$r_{c,l+1} \leftarrow \lambda^+_{c,l+1}$

$X_c \leftarrow \phi(XW^+_c, XW^-_c)\lambda^+_{c,l+1,r_{c,l+1}}$

$determine \bar{W}_1, \bar{W}_2$ of a SLANN with input $X$, label $Y$

map $\bar{W}_1, \bar{W}_2$ to $W^+_L, W^-_L, W^+_{L+1}, W^-_{L+1}, \Lambda^+_L, \Lambda^-_L, R_{L+1}, R_{L+2}$

external arrival rates of excitatory spikes for the cells in the $(L+2)$th layer, which is the output layer. The training procedure of the MLRNN is generalized for the MCMLRNN, which is given in Algorithm 8

7.6 Numerical Results

In this section, we conduct numerical tests for the MLRNN and MCMLRNN that use three multi-channel classification datasets: an image dataset and two real-world time-series datasets. The detailed descriptions of these datasets and the related classification problem have been given in Section 5.6 of Chapter 5. The following provides brief introductions of these datasets.

NORB Dataset: The small NORB dataset [LHB04] is intended for 3D object recognition from shape. The sizes of training and testing sets are both 24300. Each instance contains two $96 \times 96$ images, which are downsampled into $32 \times 32$. All images are whitened using the code provided by [TDH16]. The number of classes and channels are 5 and 2.

Daily and Sports Activities (DSA) Dataset: The DSA dataset [ABT10, BY14, AB10] comprises time-series data of 19 daily and sports activities performed by 8 subjects recorded by 45 motion sensors (25 Hz sampling frequency). The attribute number is 5,625 (45x5x25)
Chapter 7. Deep Learning with Standard Random Neural Network

since 5-second segments are used. Two thirds of 9120 instances are for training while the rest for testing. The dataset has 19 classes and 45 channels.

Twin Gas Sensor Arrays (TGSA) Dataset: The TGSA dataset includes 640 recordings of 5 twin 8-sensor detection units exposing to 4 different gases \( \text{FFGG}^+16 \). The duration of each recording is 600 seconds (100Hz sampling frequency) producing 480,000 (8x600x100) features. We use 30-second segments, and then each instance has 24,000 (8x3000) attributes. The objective is to classify gas types using recording features. The number of classes and channels are 4 and 8. Two tasks are conducted, in both of which two thirds of instances are used for training while the rest for testing:

- Task 1: (3,029 instances): build a specific classifier for Unit 1 to fulfill the objective.
- Task 2: (21,169 instances): build one classifier for all units to fulfill the objective.

In the numerical experiments, we compare the proposed MCMLRNN with the multi-layer dense RNN (i.e., the MLDRNN) presented in \[YG16b, GY16a\] and Chapter 5, the multilayer perception (i.e., the MLP) \[Cho15\], the convolutional neural network (i.e., the CNN)
Table 7.1: Testing accuracies (%), training and testing time (s) of different methods for NORB and DAS datasets.

<table>
<thead>
<tr>
<th>Method</th>
<th>Testing accuracy</th>
<th>Training time</th>
<th>Testing time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NORB</td>
<td>DAS</td>
<td>NORB</td>
</tr>
<tr>
<td>MCMLRNN</td>
<td>92.44</td>
<td>99.21</td>
<td>13.38</td>
</tr>
<tr>
<td>MCMLDRNN [YG16b]</td>
<td>92.10</td>
<td>99.21</td>
<td>28.80</td>
</tr>
<tr>
<td>MCMLDRNN1 [YG16b]</td>
<td>91.21</td>
<td>98.98</td>
<td>1750.85</td>
</tr>
<tr>
<td>MCMLDRNN2 [YG16b]</td>
<td>91.72</td>
<td>94.67</td>
<td>1169.61</td>
</tr>
<tr>
<td>Improved MLDRNN [YG16b]</td>
<td>90.96</td>
<td>92.17</td>
<td>20.63</td>
</tr>
<tr>
<td>Original MLDRNN [GY16a]</td>
<td>88.51</td>
<td>92.83</td>
<td>18.80</td>
</tr>
<tr>
<td>MLP+dropout [Cho15]</td>
<td>67.12</td>
<td>91.94</td>
<td>2563.27</td>
</tr>
<tr>
<td>CNN [Cho15]</td>
<td>90.80</td>
<td>98.52</td>
<td>1223.93</td>
</tr>
<tr>
<td>CNN+dropout [Cho15]</td>
<td>90.76</td>
<td>99.05</td>
<td>1282.99</td>
</tr>
<tr>
<td>H-ELM [TDH16]</td>
<td>87.56</td>
<td>96.58</td>
<td>125.86</td>
</tr>
<tr>
<td>H-ELM * [TDH16]</td>
<td>91.28</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

*This data is obtained directly from [TDH16].

CHo15, SHK+14] and hierarchical ELM (i.e., the H-ELM) [TDH16]. All numerical experiments are conducted in the same personal computer.

The results based on the above three datasets are given in Tables 7.1 and 7.2 including the testing accuracy, training time and testing time. The proposed MCMLRNN achieves the highest testing accuracies in three of the four classification tasks, while for Task 2 of the TGSA dataset, the testing accuracy of the MCMLRNN is 96.73% that is very close to the highest 96.91%. Compared with the conventional CNN and MLP, the MCMLRNN is more than one hundred times faster in training and achieves better accuracies. For the DAS dataset, the accuracies of the MCMLRNN in this chapter and MCMLDRNN in the previous work [YG16b] and Chapter 5 are the same, however, the MCMLRNN is more than two times faster in training and more than five times faster in testing, where they are in the same structures of 45x125-45x200-45x100-2000-19. The MCMLRNN costs the least testing time among all tools in three of the four tasks. For the DAS dataset, where the MCMLRNN achieved the highest accuracy, its testing time is also less than 1 second, the same as that the lowest one (the CNN). These numerical results well demonstrate that the MCMLRNN is effective and the most efficient among the compared deep-learning tools, as well as the value of individual RNN cells for deep learning.
Table 7.2: Testing accuracies (%), training and testing time (s) of different methods for TGSA dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 1</th>
<th>Task 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCMLRNN</td>
<td>99.31</td>
<td>96.73</td>
<td>16.52</td>
<td>52.49</td>
<td>0.27</td>
<td>1.03</td>
</tr>
<tr>
<td>MCMLDRNN[YG16b]</td>
<td>98.32</td>
<td>96.91</td>
<td>29.56</td>
<td>65.56</td>
<td>1.60</td>
<td>5.45</td>
</tr>
<tr>
<td>MCMLDRNN1[YG16b]</td>
<td>98.61</td>
<td>90.11</td>
<td>55.08</td>
<td>143.76</td>
<td>3.00</td>
<td>13.63</td>
</tr>
<tr>
<td>MCMLDRNN2[YG16b]</td>
<td>94.75</td>
<td>79.52</td>
<td>51.96</td>
<td>75.20</td>
<td>1.72</td>
<td>8.44</td>
</tr>
<tr>
<td>Improved MLDRNN[YG16b]</td>
<td>97.03</td>
<td>87.06</td>
<td>16.78</td>
<td>149.47</td>
<td>0.76</td>
<td>8.67</td>
</tr>
<tr>
<td>Original MLDRNN [GY16a]</td>
<td>85.64</td>
<td>80.29</td>
<td>29.94</td>
<td>154.00</td>
<td>0.89</td>
<td>13.99</td>
</tr>
<tr>
<td>MLP+dropout [Cho15]</td>
<td>25.05</td>
<td>24.86</td>
<td>3327.52</td>
<td>9005.39</td>
<td>0.84</td>
<td>4.03</td>
</tr>
<tr>
<td>CNN [Cho15]</td>
<td>61.78</td>
<td>72.13</td>
<td>1842.38</td>
<td>13593.06</td>
<td>0.83</td>
<td>6.09</td>
</tr>
<tr>
<td>CNN+dropout [Cho15]</td>
<td>69.11</td>
<td>87.00</td>
<td>2484.18</td>
<td>15545.18</td>
<td>0.83</td>
<td>5.95</td>
</tr>
<tr>
<td>H-ELM [TDH16]</td>
<td>61.98</td>
<td>55.94</td>
<td>14.21</td>
<td>122.02</td>
<td>0.71</td>
<td>13.62</td>
</tr>
</tbody>
</table>

### 7.7 Conclusion

This chapter has demonstrated the power of individual RNN cells in multi-layer architectures for deep learning. First, this chapter has presented three approaches, including a single-RNN-cell approach, a twin-RNN-cell approach and a RNN-cluster approach, that allow the RNN to conduct approximate ReLU-activated convolution operations, producing similar edge-detection effects. A multi-layer architecture based on the standard RNN has then been proposed for deep learning, i.e., the MLRNN and its generalized multi-channel version (the MCMLRNN). Numerical results, based on multi-channel classification datasets, have well demonstrated that the MLRNN is effective and that it is arguably the most efficient among the compared deep-learning tools.
Chapter 8

Conclusions and Future Work

8.1 Summary of Achievements

The achievements can be summarised into the following aspects:

1) This thesis has investigated the properties of the random neural network (i.e., the RNN) from different aspects, such as its approximation properties, various types of structures, and numerous learning strategies.

2) Based on these RNN-property investigations, this thesis has successfully connected the RNN with deep learning and investigated the capabilities of the RNN as a deep-learning tool.

3) Efficient and effective neural-network learning tools based on the RNN have been developed for handling various data challenges arising from various systems, such as the over-fitting effect, diversified attribute characteristics, speed requirements, and so forth.

4) The deep-learning problem of the MLRNN has been investigated from both the theoretical and statistical perspectives, including the learning capability and the impact of the multi-layer architecture. The investigations focused on the case of the MLRNN, with the potential to generalise other types of deep neural networks.

Further to this, the achievements in each chapter can be described as follows:

- Chapter investigated the universal approximation property of the RNN and constructed an efficient classifier based on the RNN function approximator in [GML99a, GML99b, GML04].
whose learning can be conducted by solving a convex optimization problem fast. The classifier equipped with the proposed learning procedure has been tested on various real-world datasets and demonstrated to generate better classification performance than five other classifiers in many cases, with less computational time.

- In Chapter 4, the quasi-linear structure of the RNN was investigated and, correspondingly, new non-negative autoencoders were proposed. The proposed autoencoders were tested on both real-world and image datasets for dimension reduction. Furthermore, the robustness and efficacy of the autoencoders are demonstrated by the numerical results.

- Chapter 5 utilised the RNN to model dense nuclei found within important areas of the human brain. Then, a multi-layer architecture of the dense nuclei was constructed alongside the development of an efficient learning procedure. The proposed deep-learning tool was proved to offer greater learning efficiency than the state-of-the-art methods in the application of recognising 3D objects, distinguishing different chemical gases, and detecting human activities.

- In order to conduct theoretical studies into deep learning with the MLRNN, Chapter 6 introduces an analytical method via moments and linearisation. From a theoretical point of view, the MLRNN has been shown to be a universal function approximator. The MLRNN is capable of extracting raw moments of higher orders as the number of layers increases. A hypothesis for transforming the deep-learning problem of the MLRNN into a moment-learning problem has been also presented. This hypothesis illustrates that it is possible to solve the deep-learning problem of the MLRNN using only the statistic estimations from the dataset, rather than the whole dataset - whose size could be very large - thereby laying a foundation for future work.

- There are two aspects to the investigation of the power of individual standard RNN cells for deep learning in Chapter 7. In the first part, the RNN demonstrated its capability of implementing image convolution operations with only positive parameters. In the second part, the deep-learning capability was investigated within the MLRNN and the learning of the MLRNN is conducted by ultimately solving a series of convex optimization problems, which is much faster than solving a non-convex optimization problem using the classic gradient-descent approach. The MLRNN equipped with the learning algorithm is shown to be the most efficient among the five different deep-learning approaches across three applications.
8.2 Future Work

Future work could focus on the following aspects:

1. The capability of the deep-learning tools based on the RNN and dense RNN could be further evaluated in more practical applications when solving real-world problems. Note that, in addition to the applications of recognising 3D objects, distinguishing different chemical gases, and detecting human activities in Chapters 5 and 7, the work in Appendix D has also applied the dense RNN to inferring the states of servers within the Cloud system and applied both the dense RNN and MLRNN for toxicity prediction of Chemical compounds.

2. Extensive comparisons of the proposed RNN tools with other related machine-learning tools could be further investigated.

3. The training and testing phases of the multi-layer RNN could be integrated into the existing deep-learning platforms, such as Tensorflow \[^{15}\].

4. The feasibility of applying the RNN theory to modelling the existing neuromorphic computing hardware platforms may be worthy of investigation.

5. The approach to utilising linearisation facilitates the analysis of the deep neural networks from a theoretical perspective, and as such this may be worth investigating further in order to better understand the deep neural networks and develop improved deep-learning tools.

6. The size of dataset in the issues of deep learning can be very large. Transforming the deep-learning problem into a moment-learning problem using the statistical estimations of raw moments from the dataset could significantly reduce the problem’s complexity. Thus, this could be a promising direction for future work on deep learning.
Bibliography


[CHH+07] Deng Cai, Xiaofei He, Yuxiao Hu, Jiawei Han, and Thomas Huang. Learning a spatially smooth subspace for face recognition. In 2007 IEEE Conference on Computer Vision and Pattern Recognition, pages 1–7. IEEE, 2007.


[MOD] INCEPTION MODULE. Googlenet: Going deeper with convolutions.


Appendices
Appendix A

Proof of Lemmas 3.3 to 3.5

Proof of Lemma 3.3. Notice that

\[
- \frac{1}{(1 + x_n)^{i_n}} = 1 - \frac{1}{(1 + x_n)^{i_n}} - 1 = \sum_{j_n=1}^{i_n} \frac{x_n}{(1 + x_n)^{j_n}} - 1. \tag{A.1}
\]

Then, it is easy to see that

\[
- \prod_{n=1}^{\hat{N}} \frac{1}{(1 + x_n)^{i_n}} = \left( \sum_{j_1=1}^{i_1} \frac{x_1}{(1 + x_1)^{j_1}} - 1 \right) \prod_{n=2}^{\hat{N}} \frac{1}{(1 + x_n)^{i_n}}
= \left( \sum_{j_1=1}^{i_1} \frac{x_1}{(1 + x_1)^{j_1}} \right) \prod_{n=2}^{\hat{N}} \frac{1}{(1 + x_n)^{i_n}} - \prod_{n=2}^{\hat{N}} \frac{1}{(1 + x_n)^{i_n}}. \tag{A.2}
\]

The remaining derivation follows the similar scheme and polynomial (3.3) can be obtained. The proof is thus completed.

Proof of Lemma 3.4. Without loss of generality, consider the case of \( i_1 \leq i_2 \leq \cdots \leq i_{\hat{N}} \). We separate the RNN to be constructed into \( \hat{N} \) layers.

Now set layer \( n \) with \( n = 1, 2, \cdots, \hat{N} \) as:

- \( r_{\hat{l}} = 1/i_n \) with \( \hat{l} = \hat{L} + 1, \hat{L} + 2, \cdots, \hat{L} + i_n + 1 \);
- \( \Lambda_{\hat{L}+1} = x_n \);
Chapter A. Proof of Lemmas 3.3 to 3.5

• $\Lambda_{L+2} = x_n/i_n$ if $n = 1$;
• $w^+_{L,L+1} = 1/i_n$ with $\hat{L} = \hat{L} + 2, \cdots, \hat{L} + i_n$;
• $w^-_{L+1,i} = 1/i_n$ with $\hat{L} = \hat{L} + 2, \cdots, \hat{L} + i_n + 1$;
• $\nu_{L+i_n+1} = 1 - i_{n-1}/i_n$ if $n < N$;
• $\nu_{L+i_n+1} = 1$ if $n = N$;

where $\hat{L} = (\sum_{\hat{n}=1}^{n-1} i_{\hat{n}} + n - 1)\text{sign}(n-1)$.

Then set the connection between layer $n - 1$ to $n$ with $n = 2, \cdots, \hat{N}$ as:

• $w^+_{L,L+2} = 1/i_n$,

where $\hat{L} = (\sum_{\hat{n}=1}^{n-1} i_{\hat{n}} + n - 1)\text{sign}(n-1)$.

The rest parameters in the RNN are set as zero. It can be seen that $q_{i+1} = x_1/(1 + x_1)^{i_1}$, $q_{L+i_n+1} = x_1/(1 + x_1)^{i_1} \prod_{n=2}^{\hat{N}} 1/(1 + x_n)^{i_n}$ with $n = 2, \cdots, \hat{N}$. Then, $q_L = x_1/(1 + x_1)^{i_1} \prod_{n=2}^{\hat{N}} 1/(1 + x_n)^{i_n}$. The proof is thus completed.

Proof of Lemma 3.5. Let $r_{L,n}$ denote the fire rate of the $L_n$th neuron in the $n$th RNN. Besides, let $w_{\text{max}}$ be the largest weight in $\{w_n \mid n = 1, 2, \cdots, \hat{N}\}$ and $r_{\text{min}}$ be the smallest rate in $\{r_{L,n} \mid n = 1, 2, \cdots, \hat{N}\}$. Let $\phi^+ = \sum_{n=1}^{\hat{N}} w_n q_{L,n}$, $\hat{w}_n = w_n r_{\text{min}}/w_{\text{max}}$ and $\hat{\phi}^+ = \sum_{n=1}^{\hat{N}} \hat{w}_n q_{L,n} = \phi^+ r_{\text{min}}/w_{\text{max}}$.

The construction of the new RNN is by connecting the $\hat{N}$ RNNs to a new output neuron. The new RNN has $\hat{N} + 1$ layers, where the $n$th RNN constitutes the $n$th layer and the $L$th neuron is the last layer. Then, the $L_n$th neuron in the $n$th layer becomes the $(\sum_{\hat{n}=1}^{n} L_n)$th in the new RNN. Now set or reset parameters in the new RNN as:

• $\nu_L = 1$;
• $\nu_{L_n} = 1 - \hat{w}_n/r_{L,n}$;
\[ \Lambda_L = 0; \]
\[ \lambda_L = 0; \]
\[ w^+_{L_n, L} = \hat{w}_n/2; \]
\[ w^-_{L_n, L} = \hat{w}_n/2; \]
\[ r_L = r_{\text{min}}/(2w_{\text{max}}); \]

where \( \hat{L}_n = \sum_{n=1}^{N} L_n \) and \( n = 1, 2, \ldots, \hat{N} \). Then,

\[
q_L = \frac{\phi^+}{r_{\text{min}}/2w_{\text{max}} + \phi^+} = \frac{\sum_{n=1}^{N} w_n q_{L_n, n}}{1 + \sum_{n=1}^{N} w_n q_{L_n, n}}. \tag{A.3}
\]

The proof is thus completed.
Appendix B

UCI Dataset Descriptions

This appendix presents brief descriptions of datasets in Table 3.2 [Lic13], which are the Iris, Teaching Assistant, Evaluation (TAE), Liver Disorders (LD), Seeds, Pima Indians Diabetes (PID), Yeast [HN96], Breast Cancer Wisconsin (BC) [MSW90], Glass, Wine, Zoo, Parkinsons [LMR+07], Wall-Following Robot Navigation (WFRN) [FBVV09], Ionosphere [SWHB89], Soybean Large (SL), First-Order Theorem Proving (FOTP) [BHPT4], Sonar [GSS8] and Cardiac Arrhythmia (CA) [GADC97].

Iris dataset. All 4 attributes (including sepal length and width and petal length and width) are real numbers and the task is to recognize which class of Iris plants (Iris Setosa, Iris Versicolour or Iris Virginica) this instance belongs to.

TAE dataset. Each instance has 5 attributes, where 3 are categorical (Course instructor, Course and evaluation scores), 1 is numerical (Class size) and 1 is binary (Summer or regular semester). The task is to recognize whether this teaching assistant is a native English speaker or not.

LD dataset. Each instances has 5 attributes, which are 5 variables (mcv: mean corpuscular volume, alkphos: alkaline phosphotase, sgpt: alamine aminotransferase, sgot: aspartate aminotransferase, gammagt: gamma-glutamyl transpeptidase) obtained from blood tests of a single male individual. The task is to recognize whether the number of half-pint equivalents of
alcoholic beverages drunk per day of the male individual is large than 5 or not.

**Seeds dataset.** Each instance contains 7 geometric parameters of wheat kernels: the area, perimeter, compactness, length of kernel, width of kernel, asymmetry coefficient and length of kernel groove. They are all real and continuous values. Using these parameters, the task is to recognize which variety of wheat kernels (Kama, Rosa or Canadian) this instance belongs to. The data is constructed thanks to the work of the Institute of Agrophysics of the Polish Academy of Sciences in Lublin.

**PID dataset.** Each instance contains 8 numerical attributes, which are information and test results of a female at least 21 years old of Pima Indian heritage. These attributes include the number of pregnancies, the oral glucose tolerance test result, diastolic blood pressure, triceps skin fold thickness, 2-hour serum insulin, body mass index, diabetes pedigree function and age. The task is recognize whether this female has diabetes or not.

**Yeast dataset.** This dataset is about the localization of yeast proteins. Each instance contains 8 features (real and continuous numbers) calculated from the amino acid sequences of the yeast protein, where the detailed descriptions of these features can be in page 111 of [HN96]. The task is to use these features to recognize the location site of the yeast protein with a cell. There are 10 location sites in this dataset, detailed in page 110 of [HN96].

**BC dataset.** More specifically, this is the Breast Cancer Wisconsin (Original) dataset in [Lic13]. Each instance is described by 9 numerical attributes [WM90, MSW90, BM92], where all of them range from 1 to 10. The attributes include the clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli and mitoses. This breast cancer databases was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg. The task is to recognize the class of the beast cancer (benign or malignant).

**Glass dataset.** Each instance has 9 continuous attributes, including the refractive index and the unit measurements of sodium, magnesium, aluminum, silicon, potassium, calcium, barium and iron. There are in total 7 types of glass, while there are instances of only 6 types of glass in
the dataset. The task is to using the 9 attributes to recognize which type of glass this instance belongs to.

Wine dataset. The dataset (donated by Riccardo Leardi, riclea@anchem.unige.it) is about wines derived from three different cultivars in Italy. Each instance includes the quantities of 13 constituents (alcohol, malic acid, ash, alkalinity of ash, magnesium, total phenols, flavanoids, nonflavanoid phenols, proanthocyanins, color intensity, hue, OD280/OD315 of diluted wines and proline) in each type of wines, which are continuous. The task is to recognize which type of wines this instance belongs to.

Zoo dataset. Each instance, corresponding to one type of animals, contains 1 numerical-valued attribute (number of legs) and 15 boolean-valued attributes (hair, feathers, eggs, milk, airborne, aquatic, predator, toothed, backbone, breathes, venomous, fins, tail, domestic). There are 101 types of animals, which can be classified into 7 sets. The task is to use these attributes to recognize which set this instance belongs to.

Parkinsons dataset. The Parkinsons dataset is composed of biomedical voice measurements from 31 people [LMR+07], 23 of which have Parkinson’s disease (PD). Each instance in the dataset contains 22 different voice measurements for an individual, which are real values. These measurement include the average, maximum and minimum vocal fundamental frequency, 5 measures of variation in fundamental frequency, 6 measures of variation in amplitude, 2 measures of ratio of noise to tonal components in the voice, 2 nonlinear dynamical complexity measures, the signal fractal scaling exponent and 3 nonlinear measures of fundamental frequency variation. The task is to recognize whether this instance comes from a healthy individual or from a PD individual.

WFRN dataset. The full title of this dataset is “Wall-Following navigation task with mobile robot SCITOS-G5” [FBVV09]. The robot utilizes the real-valued results measured from 24 ultrasound sensors to navigate in a room. Each instance contains 24 attributes corresponding to the 24 sensors. The task is to utilize these attributes to decide which direction the robot should go, where there are four choices: move forward, slight right turn, sharp right turn and slight left turn.
Ionosphere dataset. Each instance contains 34 continuous attributes, which are radar data collected by a system in Goose Bay, Labrador [SWHB89]. The task is to recognize whether the radar data in the instance are good radar returns or not.

SL dataset. Each instance in this large-soybean dataset contains 35 attributes that are numerical values. By using these attributes, the task is to conduct soybean disease diagnosis for the instance, where there are 19 different diseases. Since values of attributes in some instances are missing, 186 instances are selected from the whole dataset with 307 instances.

FOTP dataset. This dataset is denoted by James P Bridge, Sean B Holden and Lawrence C Paulson from University of Cambridge and studied in [BHP14]. Each instance in the dataset is about a theorem needed to be proved, containing 13 static and 38 dynamic features derived from this theorems, where 5 of them are integer-valued and all other features are continuous. Given these features of a theorem, the task is to predict which of 5 heuristics will give the fast proof when used by a first-order prover or to decline the attempt in proving the theorem by recognizing the theorem being too difficult, meaning that there are 6 classes in total.

Sonar dataset. This the Sonar, Mines vs. Rocks dataset used in [GS88] as the study of the classification of sonar signals. Each instance contains 60 continuous attributes in the range of 0 to 1, which are sonar signals bounced off either a metal cylinder or a roughly cylindrical rock. The task is to utilize these attributes to discriminate between sonar signals bounced off a mine (metal cylinder) and those bounced off a rock.

CA dataset. This a dataset about the diagnosis of cardiac arrhythmia that has been studies in [GADC97]. Each instance is about an individual and contains 279 attributes, which include Electrocardiogram (ECG) signals and some other information such as sex, age and weight. According to the decision of an expert cardiologist, these instances are classified into 16 classes: class 1 refers to 'normal' ECG classes (meaning the individual has no arrhythmia), classes 2 to 15 refer to different classes of arrhythmia and class 16 refers to the rest of unclassified ones. The task is to utilize the 279 attributes of a instance to distinguish the presence and absence of cardiac arrhythmia and to classify it in one of the 16 groups.
Appendix C

Deduction of $\Lambda_n^+$ and $\Lambda_n^-$ in (5.4)

The deduction of $\Lambda_n^+$ in (5.4) is given as the following.

$$\Lambda_n^+ = \lambda^+ + \sum_{j=1,j\neq n}^{N} rq\frac{(1-p)}{N} + \sum_{all \; z_2(n)} r\frac{qp}{N} q\frac{(1-p)}{N} + \sum_{L=3}^{+\infty} \left( \sum_{all \; z_L(n)} \frac{qp}{N} \left( \prod_{j=2,...,L-1} q\frac{p}{N} \right) q\frac{(1-p)}{N} \right)$$

$$= \lambda^+ + (N-1)rq\frac{(1-p)}{N} + \sum_{L=2}^{+\infty} \left( \sum_{all \; z_L(n)} r \left( \prod_{j=1,...,L-1} q\frac{p}{N} \right) q\frac{(1-p)}{N} \right)$$

$$= \lambda^+ + (N-1)rq\frac{(1-p)}{N} + \sum_{L=2}^{+\infty} \left( \sum_{all \; z_L(n)} \left( \frac{qp}{N} \right)^{L-1} q\frac{(1-p)}{N} \right)$$

$$= \lambda^+ + (N-1)q\frac{(1-p)}{N} + \sum_{L=2}^{+\infty} \left( r(N-1)^L \left( \frac{qp}{N} \right)^{L-1} q\frac{(1-p)}{N} \right)$$

$$= \lambda^+ + r(N-1)q\frac{(1-p)}{N} + \sum_{L=2}^{+\infty} \left( r(N-1) \left( \frac{qp(N-1)}{N} \right)^{L-1} q\frac{(1-p)}{N} \right)$$

$$= \lambda^+ + r(N-1)q\frac{(1-p)}{N} + \sum_{L=1}^{+\infty} \left( r(N-1) \left( \frac{qp(N-1)}{N} \right)^L q\frac{(1-p)}{N} \right)$$

$$= \lambda^+ + \sum_{L=0}^{+\infty} \left( r(N-1) \left( \frac{qp(N-1)}{N} \right)^L \frac{(1-p)}{N} \right)$$

$$= \lambda^+ + rq(N-1) \left( \sum_{L=0}^{+\infty} \left( \frac{qp(N-1)}{N} \right)^L \right) \frac{(1-p)}{N}. $$
The deduction of $\Lambda_n^-$ in (5.4) is given as the following.

$$\Lambda_n^- = \Lambda^- = \lambda^- + \hat{q}_u \hat{w}_u^- + \sum_{j=1, j\neq n}^N r \frac{qp}{N} + \sum_{\text{all } z_2(n)} r \left( \frac{qp}{N} \right)^2 + \sum_{L=3}^{+\infty} \left( \sum_{\text{all } z_L(m)} r \frac{qp}{N} \left( \prod_{j=2, \ldots, L-1} \frac{qp}{N} \right) \right)$$

$$= \lambda^- + \hat{q}_u \hat{w}_u^- + \sum_{j=1, j\neq n}^N r \frac{qp(N-1)}{N} + \sum_{L=2}^{+\infty} \left( \sum_{\text{all } z_L(m)} r \left( \frac{qp}{N} \right)^L \right)$$

$$= \lambda^- + \hat{q}_u \hat{w}_u^- + r \frac{qp(N-1)}{N} + \sum_{L=2}^{+\infty} \left( r \left( \frac{qp(N-1)}{N} \right)^L \right)$$

$$= \lambda^- + \hat{q}_u \hat{w}_u^- + \sum_{L=1}^{+\infty} \left( r \left( \frac{qp(N-1)}{N} \right)^L \right)$$

$$= \lambda^- + \hat{q}_u \hat{w}_u^- + \sum_{L=1}^{+\infty} \left( r q(N-1) \left( \frac{qp(N-1)}{N} \right)^{L-1} \frac{p}{N} \right)$$

$$= \lambda^- + \hat{q}_u \hat{w}_u^- + \sum_{L=0}^{+\infty} \left( r q(N-1) \left( \frac{qp(N-1)}{N} \right)^{L} \frac{p}{N} \right)$$

$$= \lambda^- + \hat{q}_u \hat{w}_u^- + r q(N-1) \left( \sum_{L=0}^{+\infty} \left( \frac{qp(N-1)}{N} \right)^{L} \right) \frac{p}{N}.$$
Appendix D

Applications of Deep Learning With Random Neural Networks

There are two applications of the RNN-based learning tools that are presented in this chapter. The first investigates the feasibility of the multi-layer dense RNN (i.e., the MLDRNN in Chapter 5) to infer the states of servers within the Cloud system, while the second investigates whether a machine-learning approach that begins with a compound’s physico-chemical properties can be used to learn whether it may be toxic in normal usage. This involves the RNN classifier in Chapter 3, the MLDRNN in Chapter 5 and the MLRNN in Chapter 7.

D.1 Server-State Classification in Cloud Servers

D.1.1 Introduction

The performance of a set of specific servers in large computing infrastructures - such as the Cloud [DABP14, SPA15] - can be difficult to predict, since they are affected by the servers’ configuration (number and speed of the processors, memory size, and so forth), as well as by workload characteristics, such as compute or I/O bound, execution times, IO transfer sizes and times. Additionally, there are the dynamic and statistical behaviours of the set of tasks being
run, including variations in memory usage, disk usage, and so on, and which are caused by random numbers of users with varying demands on computing resources. On the one hand, end-users anticipate that Cloud services will respect Service Level Agreements (SLA); thus, an accurate prediction of server performance and Quality of Service (QoS) is of great importance. Therefore, machine-learning techniques have been utilised in several contexts [WG15], and the work in [SA09] conducted feature selection from the data collected from servers, before applying support vector machines (SVM) to detecting anomalies. It was also shown by [SPA15] that machine learning can be used to predict the occurrence of unexpected events based on system-feature measurements.

On the other hand, previous work [WG15, BWG16] exploited reinforcement learning with the RNN [Gel89, Gel93] for dynamic workload allocation in both the Cloud and networked systems. Initially developed to represent biological neurons [GC98b], the RNN has also been utilised when handling a variety of tasks [GC98a, GK00, CG00, GKP93, GK14, GW12] via its learning capability.

While conventional deep-learning tools, such as the MLP and CNN, have achieved great successes in machine-learning in recent years [HS06, LBH15, RMN09, CMGS10]. The ELM, for example, is a one-hidden-layer neural network that has also been generalised into multi-layer architectures, while the hierarchical extreme learning-machine (H-ELM) has been shown to be an efficient tool [KZH13, TDH16].

The previous work in [GY16a, GY16b, YG16b, YG17] connected the RNN to both deep learning and to the ELM concept. Furthermore, a new RNN training procedure with a multi-layer architecture of dense RNN incorporating soma-to-soma interactions has been proposed, thus providing a fast and good generalisation performance.

In Chapter 5, the multi-layer architecture of dense RNN, i.e., the MLDRNN, were proposed for solving supervised learning problems effectively. In this work, we investigate the feasibility of the MLDRNN to infer the states of servers in the Cloud system. Comparisons are conducted between different multi-layer neural networks, including the MLDRNN, H-ELM, MLP, and CNN. By collecting time-varying system data from servers with different Cloud configurations, a
D.1. Server-State Classification in Cloud Servers

A dataset for supervised learning is generally composed of two parts - the input-attribute matrix $X$ and the label $Y$ that can either be a vector or a matrix depending on the types of learning
problems being encountered. In this section, we construct $X$ by collecting time-varying system information in a server while, for the label information $Y$, we introduce baseline tasks and measure the execution time required for the server to finish the task.

**Input-Attribute Data Collection**

Time-varying information from the server concerning its CPU, memory, swap memory, and disk usage are collected, with the number of dimensions denoted as $n$ that can be different for different servers. By simulating dynamic stochastic occupation of computing resources through stochastic users, the background load can be periodically changed during the collection.

**Label Information Collection**

We conduct baseline tasks on the server with the execution time recorded, in which the baseline task includes the operations of matrix multiplication and pseudoinverse, and is used to assess the servers state, for example, how quickly the server can handle the task. Two baseline tasks are used for label information collection, whereby Task 1 is simpler as it requires less computational time than Task 2.
D.1. Server-State Classification in Cloud Servers

The execution time (s) required to complete a baseline task is denoted by $t_{exe}$. Rather than solving a regression problem with continuous execution-time labels, we have assigned discontinuous class labels $y$ to the instances according to $t_{exe}$, utilising a preset interval vector $\alpha$, therefore transforming the problem into a classification problem, e.g., $\alpha = [a, b, c]$ means: $y = 1$ if $0 \leq t_{exe} < a$; $y = 2$ if $a \leq t_{exe} < b$; $y = 3$ if $b \leq t_{exe} < c$; $y = 4$ if $t_{exe} \geq c$.

Dataset Construction and Visualisation

After the data collection, a classification dataset is constructed in the following manner. Three collections of the time-varying information immediately before conducting a baseline task are used as input attributes for an instance. Subsequently, $3n$ represents the dimension of input attributes. The labels are the values of $y$ described above in Subsection D.1.2.

Three servers with different configurations are created using the Google Cloud platform: Server 1 has 4 CPUs and 3.6 GB memory; Server 2 has 2 CPUs and 7.5 GB memory; Server 3 has 1 CPU and 0.6 GB memory. For better illustration, we visualise several attributes in the datasets, (i.e., CPU utilization percentages) and the corresponding labelling information (i.e., the execution time). Figure D.1 visualizes the dataset collected in Server 1 using Task 1, while Figures D.2 and D.3 are for Server 2 using Task 2 and Server 3 using Task 2, respectively.
Table D.1: Features of classification datasets collected from different servers

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Attribute No</th>
<th>Instance No.</th>
<th>Class No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server1-T1</td>
<td>72</td>
<td>10196</td>
<td>5</td>
</tr>
<tr>
<td>Server2-T2</td>
<td>63</td>
<td>2358</td>
<td>3</td>
</tr>
<tr>
<td>Server3-T2</td>
<td>63</td>
<td>927</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure D.4: Testing accuracies of all 100 trials of H-ELM and MLDRNN for state detection of Server 1 with Task 1.

From these figures, we can observe the dynamic changes of background loads in the servers. Moreover, for readers’ convenience, the names, attribute numbers, and instance numbers of these classification datasets are listed in Table D.1.

D.1.3 Classification Accuracy Comparison

In this section, we compare the classification performances of four types of multi-layer neural networks in order to handle the classification problems related to the classification datasets, as collected in the servers presented in Subsection D.1.2. Each classification dataset is randomly but equally separated into a training and a testing dataset. Notably, we do not conduct any preprocessing on the datasets input attributes.

Let us first consider the Server1-T1 dataset for Server 1 shown in Table D.1. In these numerical experiments, we use the MLDRNN \cite{YG16a, GY16b}, and the MLP \cite{Cho15} with the dropout technique \cite{SHK14}, the CNN also with dropout \cite{Cho15, SHK14} and H-ELM \cite{TDH16}. The structures of the MLDRNN, MLP and H-ELM are 72-500-1000-5. For the CNN,
Figure D.5: Testing accuracies of all 100 trials of H-ELM and MLDRNN for state detection of Server 2 with Task 2.

Figure D.6: Testing accuracies of all 100 trials of H-ELM and MLDRNN for state detection of Server 3 with Task 2.

As seen in the table, the MLP and CNN with ReLU activation barely converge, and using Tanh activation is much better than using ReLU activation in this case. The reason for this may be that the nonlinearity of the ReLU is insufficient for learning the complex mapping of the Server1-T1 dataset, while that of the Tanh is sufficient. Of the deep-learning tools that were compared, for state detection of Server 1 the highest testing accuracy was generated by the MLDRNN. Moreover, the training time of the MLDRNN and H-ELM is significantly less than
Table D.2: Testing accuracies (%) and training time (s) of different methods for state detection of Server 1 with Task 1.

<table>
<thead>
<tr>
<th>Method</th>
<th>Testing accuracy</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP (ReLU) [Cho15]</td>
<td>6.79</td>
<td>938.14</td>
</tr>
<tr>
<td>MLP (Tanh) [Cho15]</td>
<td>73.64</td>
<td>527.88</td>
</tr>
<tr>
<td>CNN (ReLU) [Cho15]</td>
<td>1.18</td>
<td>220.28</td>
</tr>
<tr>
<td>CNN (Tanh) [Cho15]</td>
<td>77.17</td>
<td>586.36</td>
</tr>
<tr>
<td>H-ELM [TDH16]</td>
<td>79.80</td>
<td>0.28</td>
</tr>
<tr>
<td>MLDRNN</td>
<td><strong>80.35</strong></td>
<td>1.59</td>
</tr>
</tbody>
</table>

Table D.3: Testing accuracies (%) and training time (s) of different methods for state detection of Server 2 with Task 2.

<table>
<thead>
<tr>
<th>Method</th>
<th>Testing accuracy</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP (ReLU) [Cho15]</td>
<td>33.33</td>
<td>229.17</td>
</tr>
<tr>
<td>MLP (Tanh) [Cho15]</td>
<td>81.09</td>
<td>129.93</td>
</tr>
<tr>
<td>CNN (ReLU) [Cho15]</td>
<td>33.16</td>
<td>39.65</td>
</tr>
<tr>
<td>CNN (Tanh) [Cho15]</td>
<td>81.51</td>
<td>142.78</td>
</tr>
<tr>
<td>H-ELM [TDH16]</td>
<td><strong>87.36</strong></td>
<td>0.05</td>
</tr>
<tr>
<td>MLDRNN</td>
<td>86.60</td>
<td>0.31</td>
</tr>
</tbody>
</table>

that of the conventional MLP and CNN. From Figure D.4 we can see that the majority of the MLDRNN trials achieved higher testing accuracies than the highest one of the H-ELM among all the trials.

Subsequently, we considered the Server2-T2 and Server3-T2 datasets for Servers 2 and 3 shown in Table D.1. In these experiments, we also used the same neural-network tools with slightly different structures. For the MLDRNN and H-ELM, the structures are 63-50-10-1000-3, while for the MLP and CNN, the same structures as those in the Server1-T1 dataset are used here. The results are presented in Tables D.3 and D.4 and Figures D.5 and D.6. Of the four tools, the most efficient is the H-ELM, as evidenced by the two datasets.

These results verify the feasibility of the multi-layer neural networks for server-state detection in the Cloud, whereby the MLDRNN and H-ELM generate higher accuracies and are more efficient than either the MLP or CNN when handling the related tasks.
Table D.4: Testing accuracies (%) and training time (s) of different methods for state detection of Server 3 with Task 2.

<table>
<thead>
<tr>
<th>Method</th>
<th>Testing accuracy</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP (ReLU) [Cho15]</td>
<td>33.05</td>
<td>103.65</td>
</tr>
<tr>
<td>MLP (Tanh) [Cho15]</td>
<td>69.76</td>
<td>66.98</td>
</tr>
<tr>
<td>CNN (ReLU) [Cho15]</td>
<td>32.18</td>
<td>16.79</td>
</tr>
<tr>
<td>CNN (Tanh) [Cho15]</td>
<td>85.10</td>
<td>47.81</td>
</tr>
<tr>
<td>H-ELM [TDH16]</td>
<td><strong>88.55</strong></td>
<td>0.03</td>
</tr>
<tr>
<td>MLDRNN</td>
<td>83.80</td>
<td>0.11</td>
</tr>
</tbody>
</table>

D.1.4 Conclusion

This chapter focused on an application of the QoS-oriented server-state classification in Cloud servers, with a procedure presented to construct classification datasets related to the state detection of servers in the Cloud. The purpose was to transform the inference problem into a supervised classification problem. As such, four types of deep neural networks were applied to solve these classification problems, including the MLDRNN that was proposed in the previous chapter and based on the ideas of dense clusters, the H-ELM, and the conventional deep-learning tools of MLP and CNN. Through using experimental data gathered from three different servers, a comparison of the neural networks was able to take place. The results demonstrated that the classification problems related to server-state detection can be solved with high accuracies through using these neural-network tools, while the MLDRNN and H-ELM present with very high efficiencies compared with the two other deep-learning tools.

D.2 Toxicity Prediction of Chemical Compounds

D.2.1 Introduction

Many naturally-occurring or synthenetical chemical compounds can harm living organisms, and this harm is described either through the acute or chronic toxicity that depends on how the living organism is exposed to the chemical. The acute and chronic toxicities of the same chemical can
vary and cannot always be inferred from each other. Acute toxicity is commonly both easier
to observe and understand, and may result from delivery of a high dose. It is often evaluated
through *in vivo* experiments on animals, sometimes via past accidental events involving humans,
or at other times on living cells in cultures. Health effects resulting from acute toxicity may
be temporary or permanent, and are often observed after a period of minutes to several days,
while chronic toxicity is often understood through animal experiments conducted over lengthy
periods of time. Thus, both from an ethical perspective and in terms of the cost involved for
the experiments, it would be ideal if the acute or chronic toxicity of chemical compounds could
be determined directly through physical, mathematical, and chemical means and processes, and
perhaps even being complemented by experimentation on cell cultures, but without the use of
animal experiments. It would certainly be a significant step forward if animal experimentation
could be reduced by using other scientific approaches.

Therefore, in this work, we adopted a machine-learning (ML) approach for the toxicity assess-
ment of various chemical compounds. Our purpose was to investigate whether a ML approach
that begins with a compound’s physico-chemical properties can be used to learn whether it
may be toxic in normal usage. Consequently, if ML can be shown to be effective in this respect,
it would assist in screening compounds and help eliminate those that are obviously toxic. In
which case, *in vivo* testing would only be pursued with the smaller set of compounds that ML
has indicated as not being toxic, and which are thus worth certifying via *in vivo* assessment.

*Part of this work has been submitted as: Ingrid Grenet, Yonghua Yin, Jean-Paul Comet and
Erol Gelenbe. “Machine Learning to Predict Toxicity of Compounds.” submitted to The 27th
International Conference on Artificial Neural Networks (ICANN), 2018. Please note that, in
this work, the dataset is prepared by Ingrid Grenet, a PhD student in computational toxicology
within Bayer (Sophia Antipolis) and the I3S academic laboratory (University of Nice Sophia
Antipolis).*
D.2.2 Formalisation

This approach commences with the available data regarding a set of compounds that we call $C$, where $c \in C$ denotes a given compound. The practical and useful knowledge that we need for these compounds is characterised by the function $F : C \rightarrow V$ which reproduces the empirical evidence regarding the *in vivo* effects that are known for each of the compounds, as represented by the set $V$. A value $v(c) \in V$ may indicate either the lack of toxicity, the toxicity and its effect and severity, its effect on some organs, or on cognitive capabilities and so forth, as well as any other relevant observed data. However, if we regard each $v \in V$ as a vector, the data available concerning $F$ may be incomplete since certain elements of the vector $v = F(c)$ may be either unknown or unspecified.

In addition, we have a set of “assays” or assessments of toxicity that were obtained by *in vitro* experimentation or prior knowledge without animal experiments, as represented by the mapping $G : C \rightarrow A$, where the set of assays is denoted as $A$. For each compound, we will have a mapping $G(c) \in A$, which also may be incomplete depending on $c$.

Similarly, we would have a function $E : C \rightarrow P$ that, for each compound, provides the physico-chemical characteristics for each compound, as represented by the set $P$. Thus, the data will also include the physico-chemical characteristics $E(c) \in P$ of each compound. Notably, $E(c)$ will also be a vector of numbers, where some of the numbers for some of the compounds may be unspecified.

Consequently, although they may be incomplete, the $F$, $E$ and $G$ mappings can be regarded as the available toxicity data.

Ideally, for any given compound $c$, we would wish to have a reliable estimate of toxicity $V(c)$ that can be directly estimated from the physico-chemical characteristics of $p(c)$. In other words, we wish to discover some “unknown” function $f : P \rightarrow V$, which completely removes the need for *in vivo* experimentation. The function $f$ could be applied to the physico-chemical characteristics of any given compound, including those that have never been tested for toxicity, and could provide a reliable estimate of its toxicity. Likewise, we would like to have another
function $g : P \rightarrow A$ that provides the assays directly from the physico-chemical characteristics, even for hitherto un-assessed compounds.

The $f$ and $g$ functions remain unknown, while - based on detailed knowledge of the compounds molecular structures - the function $E$ can be shown. Therefore, the problem is the identification of the functions $f$ and $g$ based on data that represents $F$ and $G$, together with a known representation of $E$. In this work, we focused on the identification of the function $g$ from the physico-chemical characteristics of compounds to \textit{in vitro} assays, by applying different ML methods, including methods developed in the previous chapters (for example, the RNN classifier in Chapter 3, the MLDRNN in Chapter 5 and the MLRNN in Chapter 7). If the approach is shown to be effective, it would serve to help the identification of $f$ for future \textit{in vivo} studies.

\section*{D.2.3 Learning Procedure}

In this subsection, the dataset adopted for the identification of the function $g$ is described. Then, different ML methods - including methods developed in the previous chapters, such as the RNN classifier in Chapter 3, the MLDRNN in Chapter 5 and the MLRNN in Chapter 7 - are briefly introduced. The metrics for measuring the performance of the ML methods are also presented.

\subsection*{Dataset Description}

To lay a basis for future \textit{in vivo} studies, the dataset considered in this study is a subset of compounds that are available both in the ToxCast database on \textit{in vitro} assay results \cite{DHM06} and the Toxicity Reference database (ToxRefDB) on \textit{in vivo} results \cite{MJR09}. Through a careful selection, the final dataset obtained contains 404 chemical compounds. For each compound, there are 805 structural descriptors (physico-chemical properties and binary fingerprints based on the presence or absence of a chemical sub-structure in a compound \cite{RH10}) computed from the compound’s structure described in Structured Data Files, and there are 37 \textit{in vitro} assays. The 805 structural descriptors are used as the input attributes and the 37 \textit{in vitro} assay values...
are the desired outputs in a binarised form.

Learning Methods

The multi-layer perceptron. The MLP [LLPS93] is one of the most commonly used tools in deep learning [LBH15], which generally means a multi-layer fully-connected neural network. Different types of activation functions have been developed for the neurons in the MLP, e.g., the sigmoid function, the Tanh function and the Rectified Linear Unit (ReLU) [GBB11].

The convolutional neural network. The CNN is a deep-learning tool widely used for handling various types of data [LBH15, YWG17, GHV17]. The specific structure used in this work is “input-convolutional-convolutional-pooling-fully*connected-output”, implemented by Keras [Cho15].

Boosted Trees. Boosted Trees is a popular tree ensemble method. The library XGBoost [CG16] provides an easy-to-use tool for implementing boosted tree, more specifically, gradient boosting [Fri01] with regression trees.

The Extreme Learning Machine. The ELM is a one-hidden-layer neural network that has random input layer to hidden layer connections [HZS06]. A cross-validation approach in [ZYG+14a] is used to determine the structure and the weights for the ELM.

The Radial Basis Function Network. The RBFN is a one-hidden-layer neural network [WM96, YKN09]. Each instance \(d\) in the training dataset corresponds to a hidden-layer unit which receives an input vector \(X_d\) and produces an activation in the form of a Gaussian function

\[
g(X_d) = \exp(-\|X_d-X_{\tilde{d}}\|^2/(2a^2))\]

where \(a = L_{\text{max}}/\sqrt{2D}\), \(L_{\text{max}}\) is the “maximum norm” distance among all instances and \(D\) is the number of instances. Its weights are determined by the Moore-Penrose pseudoinverse method.

The Support Vector Machine. The SVM uses a RBF kernel. In this work, we use the source code provided described in [CL11] and available at [http://www.csie.ntu.edu.tw/~cjlin/libsvm](http://www.csie.ntu.edu.tw/~cjlin/libsvm).
Chapter D. Applications of Deep Learning With Random Neural Networks

The Random Neural Network Classifier. The RNN classifier is based on the RNN function approximator, whose efficacy in pattern classification has been tested on various real-world datasets in Chapter 3. In this work, the structure of the RNN classifier is determined using the cross-validation approach developed in ZYG+14a.

The Dense Random Neural Network. In Chapter 5, by exploring the idea of dense clusters of cells found with the human brain, a transfer function of dense nuclei is deduced based on the RNN theory and a multi-layer architecture of the dense RNN is developed, which is the MLDRNN (denoted as DenseRNN here). The structure of the DenseRNN used in this work is fixed to have two hidden layers with 20 and 100 intermediate nodes.

The Multi-Layer Random Neural Network. In contrast to the DenseRNN, Chapter 7 exploits the original simpler structure of the RNN and investigates its power for deep learning. Therefore, a multi-layer architecture of RNN (i.e., the MLRNN) is constructed, and has been shown to achieve a comparable or better classification performance at a much lower computation cost than the conventional deep-learning methods in certain applications. The structure of the MLRNN used here has two hidden layers with 20 and 100 intermediate nodes.

Classification Settings and Performance Metrics

For each of 37 assay, we randomly subdivide the corresponding dataset $D$ into a training set $D_T$ and a testing set $D_t$ with $D = D_T \cup D_t$. We create 50 pairs of $D_T$ and $D_t$ for experiments. The results presented below are the average of these 50 pairs of datasets. Since the in vitro assay value is either 0 or 1, this is a binary classification problem.

Let $TP$, $FP$, $TN$ and $FN$, denote the number of true positives, false positives, true negatives and false negatives, respectively. Then, the performance metrics that we use to evaluate the results
are the “balanced accuracy”, “sensitivity” and “specificity”, which are defined as follows:

\[
\text{Sensitivity} = \frac{TP}{TP + FN}, \\
\text{Specificity} = \frac{TN}{TN + FP}, \\
\text{Balanced Accuracy} = \frac{\text{Sensitivity} + \text{Specificity}}{2}.
\] (D.1)

D.2.4 Classification Results

Results of Unbalanced Datasets

In this section, the MLP, CNN, DenseRNN, MLRNN, XGBoost, RNN, RBFN and SVM are exploited to classify the 50 × 37 pairs of training and testing datasets. The mean-value results over 50 of all 37 assays using these 8 methods are summarized into Figure D.7.

As seen by the balanced accuracy metrics in Figures D.7(a) and D.7(d), the training and testing datasets were classified well by the MLP, CNN, DenseRNN, MLRNN, XGBoost and RBFN; while the RNN and SVM failed to do so. The reason may be that the classification thresholds for the methods trained with unbalanced training datasets may need to be adjusted carefully and accordingly, rather than using the standard one (for example, 0.5). The thresholds could be chosen based on the proportion of positive instances. However, it cannot be guaranteed that the proportion of positive instances in the training dataset is similar to that in the testing dataset.

Results of Balanced Datasets

In this section, we balance the 50 × 37 training datasets with data augmentation, while the corresponding testing datasets remain unchanged. The python toolbox (imbalanced-learn) developed by [LNA17] is exploited to conduct the data augmentation, where the regular SMOTE method [CBHK02] is chosen. The MLP, CNN, DenseRNN, MLRNN, XGBoost, RNN, RBFN and SVM are exploited for experiments. The mean-value results over 50 of all 37 assays using
classify the toxicity dataset with data augmentation.

Figure D.8: Training and testing mean-value results (Y-axis) versus different assays (X-axis) when the MLP, CNN, DenseRNN, MLRNN, XGBoost, RNN, RBFN and SVM are used to classify the toxicity dataset.

Figure D.7: Training and testing mean-value results (Y-axis) versus different assays (X-axis) when the MLP, CNN, DenseRNN, MLRNN, XGBoost, RNN, RBFN and SVM are used to classify the toxicity dataset.

(a) Assay vs training balanced accuracy
(b) Assay vs training sensitivity
(c) Assay vs training specificity
(d) Assay vs testing balanced accuracy
(e) Assay vs testing sensitivity
(f) Assay vs testing specificity

(a) Assay vs training balanced accuracy
(b) Assay vs training sensitivity
(c) Assay vs training specificity
(d) Assay vs testing balanced accuracy
(e) Assay vs testing sensitivity
(f) Assay vs testing specificity

Figure D.8: Training and testing mean-value results (Y-axis) versus different assays (X-axis) when the MLP, CNN, DenseRNN, MLRNN, XGBoost, RNN, RBFN and SVM are used to classify the toxicity dataset with data augmentation.
D.2. Toxicity Prediction of Chemical Compounds

Figure D.9: The highest testing results (Y-axis) versus different assay index (X-axis) of the MLP, CNN, DenseRNN, MLRNN, XGBoost, RNN, RBFN and SVM to classify the toxicity dataset before and after data augmentation.

these 8 methods with balanced training datasets are summarized into Figure D.8. Using only standard thresholds, such as 0.5, the lowest training balanced accuracies of the RNN and SVM increased from 50% (Figure D.7(a)) to more than 70% (Figure D.8(a)), while the highest increased from around 67% to around 97%. Comparing Figures D.7(d) (based on unbalanced training datasets) with Figure D.8(d) (based on balanced training datasets), it is clear that the balanced accuracies of certain assays on the testing datasets increase with data augmentation.

Figure D.9 presents the comparisons between the results based on the unbalanced and balanced training datasets, where the highest testing balanced accuracy, highest testing sensitivity and highest testing specificity achieved by these classification tools are extracted from Figures D.7 and D.8. From Figure D.9 we can see that the highest testing balanced accuracies and testing sensitivities for most assays are increased with data augmentation.

D.2.5 Conclusions

From the results, we can draw several conclusions and raise further interesting issues:

1) The machine-learning methods can correctly predict toxicity assays correctly from the physicochemical descriptors of compounds. It is worth investigating whether more data can be utilised to improve their performance.

2) Of the machine-learning methods, no option clearly emerged as the best. Therefore, we could develop information fusion or majority approaches that use multiple methods simultaneously.
3) The assays themselves tend to dictate the methods performance. Therefore, we could select assays that are more predictable and build classification approaches that target such assays.

4) Data augmentation techniques can play an important role in the classification performance of the unbalanced datasets for toxicity prediction. As such, other types of data-augmentation techniques could be tested so as to identify the most appropriate options.