A UNIFIED ALGEBRAIC FRAMEWORK EXTENDING FROM A 6-SET DISCRETE PROBABILITY ALGEBRA AND ITS APPLICATION IN DEEP LEARNING

By

Li Dai

A UNIFIED ALGEBRAIC FRAMEWORK EXTENDING FROM A 6-SET DISCRETE PROBABILITY ALGEBRA AND ITS APPLICATION IN DEEP LEARNING

By

Li Dai

A Dissertation Submitted to the Faculty of the University
of Tennessee at Chattanooga in Partial Fulfillment of
the Requirements of the Degree of Doctor of
Philosophy in Computational Science

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

August 2024

ABSTRACT

This thesis introduces a novel Unified Algebraic Framework, including an expandable Python Functions Package built upon an extensible 6-Set Discrete Probability Algebra.

The motivation behind this research is to provide a unified, general, and extendible quantitative analysis tool that can be used to delve into the neuron-level deep neural network structure and aims at improving the transparency of how the black box works and making advancements in detailed applications.

Our approach extends a 6-Set Discrete Probability Algebra to a more systematic quantitative framework that incorporates the analysis of the discrete probability distribution of neurons in deep neural network structure. Our methodology leverages the existing models and visualization of the application of the framework to quantitatively know how this algebra works and then implement the neuron-level application in classical scenarios.

The key contribution of this research includes a mathematical 6-Set Discrete Probability Algebra that offers a more robust and reasonable foundation for how neurons play their role in deep learning networks and how the quantitative analysis of the probability distribution of the neurons provides plentiful evidence and knowledge to reduce the intuition and trial and error research pattern in the selection and design of neural networks. The thesis also provides an off-the-shelf expandible quantitative research tool that can be applied in the current domain and customized to expand to various fields. The thesis also demonstrates how the framework defines and measures dissimilarity between neurons to improve diversity in ensemble learning, similarity to achieve neuron-level knowledge transferring, the minimum distance perturbation to optimize the network structure with pruning, and entropy-based on differences of neurons to interpretability and explainability.

This research provides a new approach that combines more sophisticated algebraic approaches in AI(Artificial Intelligence) and practical frameworks and tools that can be applied directly in deep learning applications to enhance effectiveness and efficiency. It will be helpful for researchers who are interested in this domain.

DEDICATION

This thesis is dedicated to my family, whose love and support have been my harbor and my strength throughout the pursuit of my doctoral degree.

To my beloved wife, Yaheng Zhang, who has been a pillar of strength and support despite the miles that separated us. Your unwavering belief in my abilities and your constant encouragement has been instrumental in my journey. Your sacrifices have not gone unnoticed and have deeply motivated me to strive for excellence.

To my son, Dylan (DaoDao) Dai, whose remarkable adaptability and quick acquisition of dual language skills have filled me with pride and allowed me to concentrate on my research. Your resilience and joy in embracing new challenges has lightened my burdens and inspired me every day.

I also extend my heartfelt gratitude to my in-laws, whose endless support and encouragement have been invaluable during this period.

Lastly, to my parents and my elder brother(Yuanjun Dai), who is no longer with us but whose values and dreams for me have always guided my path. I believe they would be proud of what I have accomplished, and I dedicate this achievement to their memory. Your love and endless support continue to guide me, and I carry your spirit forward in all that I do.

ACKNOWLEDGEMENTS

I am profoundly grateful for the guidance, support, and mentorship I received during my doctoral journey. At the forefront of this incredible journey has been my advisor, Dr. Joseph M. Kizza. His role extended far beyond academic guidance; he has been a father figure, a friend, and a relentless supporter of my professional and personal growth. Dr. Kizza not only let me teach to sharpen my teaching skills and provided countless growth opportunities but also imparted invaluable wisdom on balancing life and research, including precious advice on parenting. I consider myself extremely fortunate to have had Dr. Kizza guiding my path.

My academic journey was immensely enriched by the expertise and dedication of my computer science and mathematics professors— Dr. Hong Qin, Dr. Joe Dumas, Dr. Yu Liang, Dr. Dalei Wu, Dr. Mina Sartipi, Dr. Jin Wang, Dr. Ossama Saleh, and Dr. Li Yang. Their rigorous courses and insights have been foundational in my research.

In addition, I am deeply thankful to the mathematics professors I have been studying mathematics with, Dr. Lingju Kong, Dr. Aniekan Ebiefung, Dr. Lani Gao, Dr. Roger Nichols, Dr. Terry Walters, and Dr. Andrew H. Ledoan, whose teachings have significantly contributed to my analytical skills and research capabilities.

I would also like to express my sincere appreciation to the administrative team: Kim Sapp, Anna Lane, Eva Hunter, Lora Cook, Eunice Davis, and Bonny Clark. Their endless assistance and organizational skills were crucial in navigating the complexities of doctoral studies and they were instrumental in every step of my academic and logistical requirements.

To all of you, I extend my deepest gratitude.

TABLE OF CONTENTS

ABSTRACT iv

DEDICATION vi

ACKNOWLEDGEMENTS vii

LIST OF FIGURES xvii

LIST OF TABLES xx

LIST OF ABBREVIATIONS xxii

LIST OF SYMBOLS xxiv

CHAPTER

1 INTRODUCTION 1

1.1 Introduction 1
1.1.1 Evolving of Neural Network Architecture 2
1.1.2 Evolving of Magnitude of Weights in Neural Network 3
1.1.3 Motivation for Research 5
1.1.4 Aims and Objectives 5
1.2 Overview of the Thesis 6
1.3 Scope of the Thesis 6
1.3.1 Definition of Terms 6
1.3.2 Scope 8
1.4 Structure of the Thesis 8
1.4.1 Outline of Chapters 8
1.5 Importance of the Research 9
1.5.1 Contribution to Knowledge 9

1.5.2 Practical Implications . . . 9

2 LITERATURE REVIEW . . . 10

2.1 Introduction . . . 10
2.2 Review of Relevant Theories/Models . . . 10
2.3 Review of Previous Empirical Research . . . 23
2.4 Gaps in the Literature . . . 24
2.5 Statement of the Research Problem . . . 27
2.6 Research Questions/Objectives . . . 27
2.7 Summary . . . 28

3 AN EXTENSIBLE 6-SET DISCRETE PROBABILITY ALGEBRA AND UNIFIED ALGEBRAIC FRAMEWORK . . . 29

3.1 Abstract . . . 29
3.2 Introduction . . . 30
3.2.1 Background . . . 30
3.2.2 Purpose . . . 31
3.3 Literature Review . . . 33
3.3.1 Existing Mathematical Frameworks . . . 33
3.3.2 Existing Engineering Framework . . . 34
3.4 Background Information . . . 35
3.4.1 Theoretical Foundations . . . 35
3.5 Theoretical Development . . . 37
3.5.1 Mathematical Formulation . . . 37
A Definition of 6-Set Discrete Probability Algebra . . . 37
B Definition of UAP(Unified Algebraic Property) . . . 38
C Definition of UAD(Unified Algebraic Distance) . . . 40
a E_UAD(Euclidean UAD) . . . 40
b UAD_SF(UAD based on Sample Function) . . . 40
c UAD_PF(UAD based on Probability Function) . . . 42
d Definition of UAM(Unified Algebraic Measure) . . . 44
D UAM Defintions Under Different Situations . . . 45
a Sample Element Order Related . . . 46
b Sample Order Related . . . 47
c Discrete Probability Order Related . . . 47
E Single Neuron . . . 50
a Entropy based on Metric . . . 51
i Shannon Entropy . . . 51
ii Rényi Entropy . . . 52
iii Tsallis Entropy . . . 53

b Entropy based on Sample Function . . . 55
F Two Neurons . . . 56
a Euclidean Distance . . . 56
b Entropy based on Euclidean Distance . . . 57
c Euclidean Distance Extended from Loss Function . . . 59
G Across Different Sample Sets . . . 59
a Measures Extended from Existed . . . 60
i IPM(Integral Probability Metrics) . . . 60
ii Common Function or Activation Function . . . 61
iii Total Variation Distance . . . 62
iv Entropy Difference . . . 63
v Cross-Entropy . . . 65
vi Divergence . . . 67
1 KL(Kullback-Leibler) divergence . . . 67
2 Alpha-Divergence . . . 68
3 Beta-Divergence . . . 70
4 Gamma-Divergence . . . 71
5 f-Divergence (or General H-Divergence) . . . 73
6 H-divergence (Hellinger Distance) . . . 75
7 Jensen-Shannon Divergence . . . 76
8 χ^2-divergence . . . 78
vii MMD(Maximum Mean Discrepancy) . . . 80
viii Wasserstein Distance . . . 81
b Customized Components and Customized Measures . . . 84
i Reverse Integral . . . 84
ii Random Order Integral . . . 85
iii Inner-Sample Order Matters . . . 85
1 Binary System Analogy . . . 86
2 Decimal System Analogy . . . 86
3 Mathematical Illustration . . . 87
iv Entropy based on Element-wise Subtraction . . . 87
v Entropy based on Element-wise Division . . . 88
vi Entropy based on Element-wise Divison Remainder . . . 89
vii Unified Composite Entropy . . . 90
1 Point Entropy . . . 90
2 Vector Entropy . . . 91
3 Matrix Entropy . . . 91
4 Composite Entropy . . . 91
viii Unified Composite Divergence . . . 92
1 Point Divergence . . . 92
2 Vector Divergence . . . 92

3 Matrix Divergence 92
4 Composite Divergence 93
ix Unified Composite Measure 93
1 Normalization 93
2 UCM(unified composite measure) 94
3.5.2 Visualization 94
3.6 Implementation 95
3.6.1 Algorithm Development 95
3.6.2 Computational Tools 95
3.7 Conclusion 95
3.7.1 Summary 95
3.8 Appendix of This Chapter: Four Main Algorithms in Detail 96
3.8.1 Algorithm: Unified Algebraic Measure Calculation 96
A Detailed Description of Functionality 98
3.8.2 Algorithm: Transform Tensor 99
A Key Points of the Pseudocode 100
3.8.3 Algorithm: Convert Samples to Probability Tensor 101
A Explanation of Pseudocode Components 102
a Initialization and Default Settings 102
b Distribution Type Check 102
c NumPy Conversion 102
d Probability Calculations 102
e Output 102
3.8.4 Algorithm: integral_funcOnProb_with_options 104
A Explanation of Pseudocode Components 104
a Function Definition 104
b Initialization 105
c Main Loop 105
d Conditional Application of Functions 105
e Storing Results 106
f Return Statement 106

4 APPLICATION OF THE UNIFIED ALGEBRAIC FRAMEWORK
IN ENSEMBLE LEARNING 107

4.1 Abstract 107
4.2 Introduction 107
4.2.1 Relevance 107
4.2.2 Objectives 109
4.3 Literature Review 109
4.3.1 Related Research 109

4.4 Mathematical Background on Ensemble Learning in Deep Learning 112
4.4.1 Formal Framework 112
4.4.2 Key Ensemble Methods in Deep Learning 113
A Bagging in Deep Learning 113
B Boosting in Deep Learning 113
C Stacking in Deep Learning 113
4.4.3 Advantages of Ensemble Learning in Deep Learning 114
4.4.4 Challenges and Considerations 114
4.5 Methodology 115
4.5.1 Application Description 115
4.5.2 Experimental Setup 116
A Experimental Setup 116
B Pseudocode and Flowchart for the Experiment 117
C Neural Network Architecture 121
4.6 Results 122
4.6.1 Performance Analysis 122
4.6.2 Comparative Study 125
4.7 Discussion 126
4.7.1 Interpretation 126
4.7.2 Limitations and Future Research 127
4.8 Conclusion 128
4.8.1 Chapter Summary 128

5 APPLICATION OF THE UNIFIED ALGEBRAIC FRAMEWORK
IN TRANSFER LEARNING 130

5.1 Abstract 130
5.2 Introduction 130
5.2.1 Context and Relevance 130
5.2.2 Objectives 131
5.3 Literature Review 132
5.3.1 Related Research 132
5.4 Background Information 134
5.4.1 Theoretical Foundations 134
5.5 Methodology 135
5.5.1 Application Description 135
5.5.2 Experimental Setup 135
A Experimental Setup 135
B CNN Architecture 137
C Pseudocode and Flowchart for the Four Scenarios 138
5.6 Results 139

5.6.1 Outcome Presentation 139
5.6.2 Analysis 140
5.7 Discussion 141
5.7.1 Interpretation of Results 141
5.7.2 Future Research Directions 141
5.8 Conclusion 142
5.8.1 Chapter Summary 142

6 APPLICATION OF THE UNIFIED ALGEBRAIC FRAMEWORK IN GENERATIVE MODEL 143

6.1 Abstract 143
6.2 Introduction 143
6.2.1 Context and Relevance 143
6.2.2 Objectives 144
6.3 Literature Review 144
6.3.1 Review of Existing Work 144
6.4 Background Information 146
6.4.1 Theoretical Context 146
6.5 Methodology 147
6.5.1 Implementation Details 147
A Experiment Setup 147
B Neural Network architectures 147
C Pseudocode and Flowchart for the Experiment 151
6.5.2 Testing and Validation 154
6.6 Results 155
6.6.1 Outcomes 155
6.6.2 Comparative Analysis 161
6.7 Discussion 161
6.7.1 Impact and Implications 161
6.7.2 Recommendations for Further Research 162
6.8 Conclusion 162
6.8.1 Overview of Contributions 162

7 APPLICATION OF THE UNIFIED ALGEBRAIC FRAMEWORK IN PRUNING NEURAL NETWORK 164

7.1 Abstract 164
7.2 Introduction 164
7.2.1 Context and Significance 164
7.2.2 Objectives 165
7.3 Literature Review 165

7.3.1 Existing Approaches 165
7.4 Background Information 167
7.4.1 Theoretical Underpinnings 167
7.5 Methodology 168
7.5.1 Pruning Techniques 168
7.5.2 Experimental Setup 169
A Experimental Setup 169
B Neural Network Architecture 170
C Pseudocode for the Implementation of the Experiment 171
7.6 Results 171
7.6.1 Findings 171
7.7 Discussion 190
7.7.1 Theoretical and Practical Implications 190
7.7.2 Future Research Opportunities 191
7.8 Conclusion 191
7.8.1 Chapter Summary 191

8 APPLICATION OF THE UNIFIED ALGEBRAIC FRAMEWORK IN INTEPRETABILITY AND EXPLAINABILITY OF DEEP LEARNING 192

8.1 Abstract 192
8.2 Introduction 192
8.2.1 Context and Importance 192
8.2.2 Objectives 193
8.3 Literature Review 193
8.3.1 Current Research 193
8.4 Background Information 195
8.4.1 Theoretical Basis 195
8.5 Methodology 196
8.5.1 Implementation of Algebra 196
8.5.2 Evaluation Methods 197
A Experimental Setup 197
B Neural Network Architecture 199
C Pseudocode for Calculating and Collecting Information 200
8.6 Results 200
8.6.1 Results Interpretation 200
A Three-Stage Explanation 200
B Distillation Process 203
C Accuracy Change and Neuron Removal 207
D Correlation in GEMM Layers 208
E Correlation in ReLU Layers 212

F Synchronization During Training 216
8.6.2 Benchmarking 218
8.7 Discussion 219
8.7.1 Implications for AI 219
8.7.2 Limitations and Future Directions 219
8.8 Conclusion 219
8.8.1 Chapter Insights 219

9 CONCLUSION AND DISCUSSION 221

9.1 Review of Research Questions 221
9.1.1 Effectiveness of the Unified Algebraic Framework 221
9.1.2 Integration into Practical Software Tools 221
9.1.3 Improvements in Neural Network Design and Interpretation 222
9.2 Summary of Key Results and Novel Contributions 222
9.2.1 Development of the 6-Set Discrete Probability Algebra 222
9.2.2 Unified Algebraic Framework 222
9.2.3 Applications in Ensemble Learning 222
9.2.4 Advancements in Transfer Learning 223
9.2.5 Generative Model Analysis 223
9.2.6 Neural Network Pruning 223
9.2.7 Enhancing Interpretability and Explainability 223
9.3 Overall Conclusions 224
9.3.1 Effectiveness of the Algebraic Framework 224
9.3.2 Practical Software Integration 224
9.3.3 Enhancements in Neural Network Design and Optimization 224
9.3.4 Broad Applications and Future Directions 224

10 FUTURE WORK 226

10.1 Our Recommendations for Future Work 226
10.1.1 Automatically Searching Different Function Spaces 226
10.1.2 Expanding the Framework to Other Machine Learning Models 226
10.1.3 Integrating with Real-Time Systems 227
10.1.4 Enhancing Visualization Capabilities 227
10.1.5 Applying the Framework to Large-Scale Datasets 227
10.1.6 Automating the Selection of Transferable Subnetworks 227
10.1.7 Exploring Multi-Objective Optimization 228
10.1.8 Investigating the Impact on Different Neural Network Architectures 228
10.1.9 Developing Educational Resources 228

REFERENCES 229

APPENDIX

A: PYTHON PACKAGE BASED ON 6-SET DISCRETE PROBABILITY DISTRIBUTION ALGEBRA 242

B: A LIST OF FUNCTIONS FROM SAMPLE FUNCTION SET AND HOW THEY CHANGE THE DISTRIBUTION OF NEURONS 276

C: UAM(UNIFIED ALGEBRAIC MEASURE) BETWEEN THE DISTRIBUTIONS FROM NEURONS AND CLASSICAL DISTRIBUTIONS 352

D: STATISTICAL TESTS LIST 364

E: LARGER AND MORE LEGIBLE IMAGES OF THE NEURAL NETWORK ARCHITECTURES 370

VITA 376

LIST OF FIGURES

3.1 $\mathrm{UAM}(\mathrm{S}, \mathrm{T}, \mathrm{f}, \mathrm{m}, \mathrm{P}, \mathrm{list}_\mathrm{g}, \mathrm{h})$ of the Unified Algebraic Framework 45

4.1 Flowchart of Ensemble Learning Strategy 120

4.2 Neural Network Architecture 121

4.3 Average Diversity of Neurons in Enhanced Model 123

4.4 Average Diversity of Neurons in Random Selection Model 124

4.5 Average Diversity of Neurons in Direct Model 125

5.1 ImprovedCNN Architecture 137

5.2 Flowchart of Transfer Learning Strategy 139

6.1 Discriminator Architecture of GAN 148

6.2 Generator Architecture of GAN 149

6.3 Merged Generator Architecture of GAN 150

6.4 Flowchart of Customized GAN Strategy 154

6.5 Generated Images from WikiArt 156

6.6 Generated Images from CIFAR-10 157

6.7 Generated Images from COCO 158

6.8 Generated Images from CIFAR-10 using WikiArt Neurons 159

6.9 Generated Images from COCO using WikiArt Neurons 160

7.1 Neural Network Architecture 170

7.2 Diversity vs. Accuracy Correlation 172

7.3 Entropy vs. Accuracy Correlation 173

7.4 Laplacian Smoothness vs. Accuracy Correlation 174
7.5 KL Divergence vs. Accuracy Correlation 175
7.6 JS Divergence vs. Accuracy Correlation 176
7.7 Layer Values for Different Metrics 177
7.8 Normalized Metrics vs. Neuron No for Layer 0 178
7.9 Normalized Metrics vs. Neuron No for Layer 1 179
7.10 Entropy vs. Accuracy Correlation for Layer 0 180
7.11 Average Values of Metrics for Different Layers 181
7.12 Entropy vs. Accuracy Correlation for Layer 2 182
7.13 Entropy vs. Accuracy Correlation for Layer 4 183
7.14 Laplacian Smoothness vs. Accuracy Correlation for Layer 2 184
7.15 Laplacian Smoothness vs. Accuracy Correlation for Layer 4 185
7.16 JS Divergence vs. Accuracy Correlation for Layer 8 186
7.17 JS Divergence vs. Accuracy Correlation for Layer 10 187
7.18 JS Divergence vs. Accuracy Correlation for Layer 12 188
7.19 JS Divergence vs. Accuracy Correlation for Layer 14 189
7.20 JS Divergence vs. Accuracy Correlation for Layer 16 190
8.1 Neural Network Architecture 199
8.2 Average Diversity vs. Layer 201
8.3 Average Entropy vs. Layer 202
8.4 Average Entropy and Diversity vs. Layer 203
8.5 Entropy and Diversity for Layer 3 204
8.6 Entropy and Diversity for Layer 4 205
8.7 Entropy and Diversity for Layer 11 206
8.8 Entropy and Diversity for Layer 12 207

8.9 Accuracy Change by Layer Number 208
8.10 Correlation in Layer 1 209
8.11 Correlation in Layer 3 210
8.12 Correlation in Layer 5 211
8.13 Correlation in Layer 6 212
8.14 Correlation in Layer 2 213
8.15 Correlation in Layer 4 214
8.16 Correlation in Layer 6 215
8.17 Correlation in Layer 8 216
8.18 Normalized Entropy and Diversity for Layer 9 217
8.19 Normalized Entropy and Diversity for Layer 10 218

LIST OF TABLES

3.1 Configurable Template of UAM 50

3.2 UAM: Shannon Entropy based on p-Norm Metric 52

3.3 UAM: Rényi Entropy based on p-Norm Metric 53

3.4 UAM: Tsallis Entropy based on p-Norm Metric 55

3.5 UAM: Shannon Entropy based on Any Sample Function 56

3.6 UAM: Euclidean Distance 57

3.7 UAM: Shannon Entropy based on Euclidean 58

3.8 UAM: Euclidean Distance Extended from MSE 59

3.9 UAM: IPM 61

3.10 UAM: UAD based on sample function 62

3.11 UAM: TVD(Total Variation Distance) 63

3.12 UAM: Absolute Entropy Difference 65

3.13 UAM: Cross-Entropy 66

3.14 UAM: KL Divergence 68

3.15 UAM: Alpha-Divergence 69

3.16 UAM: Beta-Divergence 71

3.17 UAM: Gamma-Divergence 73

3.18 UAM: $f(H)$-Divergence 74

3.19 UAM: H-Divergence 76

3.20 UAM: Jensen-Shannon Divergence 78

3.21 UAM: χ^2-Divergence 79

3.22 UAM: MMD 81

3.23 UAM: Wasserstein Distance 84

3.24 UAM: Shannon Entropy based on Element-wise Subtraction 88

3.25 UAM: Shannon Entropy based on Element-wise Division 89

3.26 UAM: Shannon Entropy based on Element-wise Division Remainder 90

4.1 Performance Metrics of Different Neural Network Configurations 122

5.1 Accuracy of Different Transfer Learning Scenarios 140

LIST OF ABBREVIATIONS

UAF Unified Algebraic Framework

UAM Unified Algebraic Measure

UAP Unified Algebraic Property

UAD Unified Algebraic Distance

UCE Unified Composite Entropy

UCD Unified Composite Divergence

UCM Unified Composite Measure

E_UAD Euclidean UAD

UAD_SF UAD based on Sample Function

UAD_PF UAD based on Probability Function

DL Deep Learning

AI Artificial Intelligence

NLP Natural Language Processing

AD Autonomous Driving

GAI General Artificial Intelligence

LLM	Large Language Model
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
LSTM	Long Short-term Memory
DNN	Deep Neural Network
BERT	Bidirectional Encoder Representations from Transformers
SGD	Stochastic Gradient Descent
GP	Gaussian Processes
DAG	Directed Acyclic Graphs
LRP	Layer-wise Relevance Propagation
ResNET	Residual Network
NNS	Neural Network Scanner
TDA	Topological Data Analysis
IPM	Integral Probability Metric
MMD	Maximum Mean Discrepancy

LIST OF SYMBOLS

$\mathbb{S}$	Set of Sample Sets
S	Sample Set
$\{s_1, s_2, \ldots\}$	A collection of data points in S
$\mathscr{F}$	Family of Sample Functions
s	A Sample Function
$\mathscr{M}$	Family of Metric Functions
m	A Metric Function
$\mathbb{P}$	Set of Discrete Probability Distributions
P	Discrete Probability Distribution
$\mathbf{P}$	Discrete Probability Distribution
$\{p_1, p_2, \ldots, p_n\}$	A collection of probability points in P
$\mathscr{G}$	Family of Functions on Discrete Probability Distribution
g	A Discrete Probability Distribution Function
$\mathscr{H}$	Family of Functions on Integral of Discrete Probability Distribution
h	A Function on Integral of Discrete Probability Distribution

CHAPTER 1

INTRODUCTION

1.1 Introduction

The outstanding success of AI (artificial intelligence), particularly of deep learning, shows that neural networks have achieved capabilities close to a perfect human or far beyond humans in various domains of image classification and recognition, NLP (natural language processing), health, AD (autonomous driving) game playing, and more. Even though now deep learning has achieved close-to-GAI (General Artificial Intelligence) effectiveness in some domains especially LLM (Large Language Model), we still do not know how deep learning models make decisions. The understanding of many detailed applications still stays at the model level (like ensemble learning), and the layer level (like transfer learning and generative model). The neural network type selection from the flattened network, RNN (recurrent neural network), CNN (convolutional neural network), Transformer, and other structures rely on intuition derived from prior experience and theoretical knowledge. For a specific structure, the selection of hyperparameters to form the optimal structure or the most transferable substructure depends on trial and error without quantitative evidence. However, the "black box" work, more specifically, work to better understand how that black box deep learning model operates significantly lags behind the progress made so far. We present a new algebraic framework using the 6-Set discrete probability algebra with additional function packages to help improve and reveal complex models toward the understanding of the black box. Before we begin with the normal content of this thesis, we review the evolution of neural network architecture, the magnitude of the number of weights(thousands, millions, billions, etc), the expansion of applications in various sections, and the development of methods and tools have not been kept pace to ensure these models are transparent, interpretable, and explainable. This mismatch generates the motivation, aims, and objectives of this thesis.

1.1.1 Evolving of Neural Network Architecture

For artificial intelligence to surpass human intelligence, deep learning algorithms are essential. Leadership has changed neural network architectures, thus enhancing intricate and efficient tasks that such structures can perform from different dimensions.

The foundation was first developed in 1958, a critical step toward the development of neural networks today. This pioneer model influenced by the brain function of pattern recognition had an extensive range of tasks that could be performed by machines, which formed many neural architectures. The Perceptron[120] was simple; however, it was revolutionary in the field and critical learning for neural networks' potentiality for computing. This technology significantly contributed to effectively training multilayer neural networks with the introduction of backpropagation by David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams [121].

There have been more breakthroughs employing deep neural network architectures CNN(convolution neural network) for pattern recognition that involve grids such as images since 1989 by Yann LeCun. This allows the network to identify patterns. This achievement has been groundbreaking in the fields involving image and video processing. The capability outperforms the expectations of new standards in computer visions[83].

LSTM(long short-term memory) is another, but Major to the development of the capabilities of artificial intelligence. This enabled the network to remember long grids, a step that led to the breakthrough in the natural language processing networks and speech recognition [61].

The revival of interest in deep neural networks gained significant momentum from the pioneering work of Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh on deep belief networks.

Their innovations enhanced the training processes and performance of neural networks, revitalizing the field and paving the way for the subsequent explosion in deep learning.

This revitalization was exemplified by the breakthrough success of AlexNet in the 2012 ImageNet competition, which demonstrated the exceptional capabilities of CNNs(deep convolutional neural networks) in handling complex visual recognition tasks. This landmark achievement highlighted the practical applications and immense potential of deep learning technologies.

Further advancements were achieved by Ilya Sutskever, Oriol Vinyals, and Quoc V. Le with the introduction of attention mechanisms and transformers. These innovations have greatly improved the processing and understanding of natural language, enabling neural networks to handle and interpret complex linguistic patterns with a precision nearing human-like understanding, as evidenced by technologies like sequence-to-sequence models and BERT(Bidirectional Encoder Representations from Transformers) [133, 27].

Each of these pivotal developments not only represents significant strides in the capabilities of artificial intelligence but also influences the continuous evolution of technologies that could potentially equal or even surpass human cognitive abilities. These milestones mark critical points in the journey of AI, showcasing its growing influence and potential in a variety of fields.

1.1.2 Evolving of Magnitude of Weights in Neural Network

The evolution of DNNs(deep neural networks) has been characterized by an exponential increase in the number of model parameters, indicative of the growing sophistication of these architectures and advancements in computational technology. This transition from simplistic single-layer networks to extensive and intricate models signifies a crucial trend in artificial intelligence research, propelling enhancements across various fields.

The journey began with Frank Rosenblatt's perceptron in 1958, which introduced neural networks with a limited number of weights capable of performing elementary tasks like simple pattern recognition [120]. Although foundational, these early models were limited by the then-available hardware and theoretical knowledge.

The advent of backpropagation by David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams in 1986 marked a turning point, enabling the training of multilayer networks and significantly increasing the manageable parameters [121]. This development allowed for the creation of models capable of capturing more profound data abstractions.

Further progress was seen with Yann LeCun and colleagues' refinement of CNNs(convolutional neural networks) in 1989, which enhanced the scalability and efficiency of networks, especially for image-related tasks [83]. CNNs introduced innovations like parameter sharing and local receptive fields, enabling networks to deepen and widen without commensurately increasing computational demands.

In 1997, Sepp Hochreiter and Jürgen Schmidhuber's introduction of LSTMs(Long Short-Term Memory networks) marked another advancement, essential for processing sequences without losing information over time and managing complex dependencies thanks to its numerous parameters [61].

The size of models saw a dramatic increase with Geoffrey Hinton and his team's development of deep belief networks in 2006, utilizing a layer-wise training strategy that supported the training of significantly deeper networks [60].

This expansion continued with the creation of AlexNet by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in 2012, which featured tens of millions of parameters and showcased impressive performance in large-scale image classification [79]. The trend was furthered by the introduction of transformers in 2017 by Ashish Vaswani et al., leading to even larger models such as BERT(Bidirectional Encoder Representations from Transformers) in 2018, which contained hundreds of millions of parameters [135, 27].

This trajectory peaked with the introduction of GPT-3 by Tom B. Brown and others in

2020, a model with an astounding 175 billion parameters that demonstrated capabilities nearing human-level performance in various language tasks [16]. This model not only showcased a massive increase in size but also the potential of such large-scale networks to approximate human cognitive complexity.

This historical progression has not only broadened the capacities of neural networks to manage more data and undertake more complex tasks but has also deepened our understanding of how to effectively harness depth, breadth, and scale in models to emulate human intelligence nuances. The increasing scale of these networks mirrors significant strides in computational capabilities and algorithmic innovation, illustrating deep learning's path toward creating increasingly powerful and intelligent systems.

1.1.3 Motivation for Research

The initial motivation of this thesis comes from the significant gap in delving into neuron-level quantitative analysis and understanding how deep learning models function at a granular level. Although the magnitude of the number of weights and the spreading of models to various domains was and is changing more and more quickly, at the same time, only limited kinds of basic neural network architectures(flatten, CNN, RNN, LSTM, Attention Mechanism, and Transformer) are used in the developing process hence not significantly improving in the interpretability and explainability and this creates the gap. This gap leads to current approaches in whole neural network architecture design and hyperparameter optimization relying heavily on intuition, experience, and trial-and-error methods instead of being guided by plentiful quantitative evidence. To address this downside this research aims to replace the traditional qualitative way with more empirical, data-driven methods that offer a deeper insight into neural network structures and their functionalities.

1.1.4 Aims and Objectives

The aims and objectives of this thesis are as follows.

1. Mathematically define the 6-Set Discrete Probability Algebra.

2. Develop a Unified Algebraic Framework based on 6-Set Algebra.

3. Implement this framework in a Python package that can be used to analyze deep neural network structures at a lower level and collect quantitative evidence for optimization of the neural architectures and improving their application in various sections.

4. Demonstrate the framework's application across various deep learning scenarios, including ensemble learning, transfer learning, generative models, network pruning, and improving the interpretability and explainability of neural networks.

5. Provide a quantitative tool that aims to reduce the dependence on intuition and enhance the precision in the scheming and promoting of deep learning architectures.

1.2 Overview of the Thesis

This thesis proposes a mathematically structured and systematic approach(6-Set Discrete Probability Algebra) to dissect and analyze deep neural networks at the granular neuron level using a newly developed algebraic framework(An Unified Algebraic Framework). By quantifying the discrete probability distributions of neuron activations, this framework seeks to provide insights into the internal decision-making processes of deep learning models, enhancing transparency and interpretability, and improving the detailed application of different structures across different fields.

1.3 Scope of the Thesis

1.3.1 Definition of Terms

For clarity, key terms used throughout this thesis are defined as follows:

1. Unified Algebraic Framework: An integrated set of algebraic tools and concepts designed to analyze and interpret complex systems.

2. 6-Set Discrete Probability Algebra: A novel algebraic structure that handles discrete probability distributions across six distinct sets, facilitating detailed analysis of probabilities within

neural networks.

3. Neuron-level Analysis: Examination and interpretation of individual neuron functions, layer functions, and their contributions to the overall behavior of a neural network.

4. Sample Set: We use a set of data to train and evaluate a specific neural network, this kind of data set is one element of the Sample Set.

5. Set of Sample Set: One of the 6 Sets(The first set).

6: Sample Function: The function in this family operates on a specific point of the Sample Set. It includes the common functions that element-wise operate on the data set and also the neural network functions.

7. Family of Sample Function: One of the 6 Sets(The second set).

8. Metric Function: A metric function, or distance function, is a function that defines the distance between sample points in a Sample set.

9. Family of Metric Function: One of the 6 Sets(The third set).

10. Discrete Probability Distribution: Defines how the points of sample in a sample set are distributed according equally to the distance metric.

11. Set of Discrete Probability Distribution: One of the 6 Sets(The fourth set).

12. Functions on Discrete Probability Distribution: A function on Discrete Probability Distribution is a common mathematical function with different discrete probabilities distribution as the independent variables.

13. Family of Functions on Discrete Probability Distribution: One of the 6 Sets(The fifth set).

14. Functions on Integral of Discrete Probability Distribution: A function on Integral of Discrete Probability Distribution is a common mathematical function with different integrals of discrete probability distributions as the independent variables.

15. Family of Functions on Integral of Discrete Probability Distribution: One of the 6

Sets(The sixth set).

15. UAD(unified algebraic distance): Used to measure the characteristics of the output of one single neuron or the distance between the output of two neurons in the unified algebraic framework.

1.3.2 Scope

This thesis is focused on mathematically defining the 6-Set Discrete Probability Distribution Algebra, scheming the Unified Algebraic Framework, implementing the corresponding Python package, and applying the framework to understand and optimize deep neural networks. While the primary application is within the field of deep learning including ensemble learning, transfer learning, deep reinforcement learning, generative learning, and other kinds of learning, the methodologies developed have the potential for broader applications in other areas of artificial intelligence and computational modeling because of its wonderful extensibility.

1.4 Structure of the Thesis

1.4.1 Outline of Chapters

The thesis is organized into several chapters, each addressing different aspects of the algebraic framework and its applications besides the basic introduction, literature review, conclusion, and future work expected:

Chapter 1: Introduction. Sets the stage by outlining the motivation, objectives, and scope of the research.

Chapter 2: Literature Review.

Chapter 3: An Extensible 6-Set Discrete Probability Algebra and Unified Algebraic Framework. 6-Set Algebra details the theoretical development and mathematical properties of the 6-Set Discrete Probability Algebra. Unified Algebraic Framework illustrates the implementations of 6-Set Agebra in the form of a Python package.

Chapter 4 to Chapter 8: Each of these chapters explores a specific application of the framework in a specific subfield of deep learning.

Chapter 4: Ensemble Learning. How to select a set of neurons that can increase the diversity of the model which focuses on improving the generality.

Chapter 5: Transfer Learning. Using the selection of neurons, layers, or the combination of neuron selection and layer selection to boost the accuracy of the model.

Chapter 6: Generative Models. Using a cluster of neurons from another GAN in the current GAN.

Chapter 7: Pruning Neural Networks. Using entropy, smoothness, anomalousness of the output of neurons and dataflow on the connection to prune the neural network architectures.

Chapter 8: Interpretability and Explainability of Deep Learning. Using UAMs(unified algebraic measures) to improve the explainability of the neural networks.

Chapter 9: Conclusion and Discussion.

Chapter 10: Future Work.

1.5 Importance of the Research

1.5.1 Contribution to Knowledge

This research contributes to the field by providing a mathematical and systematic method to analyze and interpret the internal dynamics of deep neural networks, thereby addressing a significant gap in current AI research methodologies.

1.5.2 Practical Implications

The practical implications of this research are vast, including more efficient neural network design, enhanced model performance, and the potential to apply these insights in real-world AI applications, ultimately leading to AI systems that are both powerful and understandable.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter focuses mainly on providing a comprehensive, detailed literature review of mathematical theories and their empirical implementation in deep learning. In this chapter, we examine all the mathematical theories that are covered in the deep learning domain including linear algebra(vector algebra, matrix algebra, or tensor algebra), calculus, probability theory, information theory, optimization theory, and graph theory. The empirical studies reviewed focus on how these theories are turned into engineering implementation in the form of framework and software packages. The existing theoretical and empirical review leads us to the gap and all scholars are talking about it. Hence the need for a unified algebraic framework is urgent. The research questions are also listed.

2.2 Review of Relevant Theories/Models

Linear Algebra: Tracing back to the evolutionary history of DNN(deep neural network) and CNN(convolutional neural network), their significant performance largely depends on the widely used linear algebra. Vectors and matrices are the main data formats that flow along the DNN architectures, while tensor algebra plays a more important role in handling more complicated data structures.

In DNNs, all data points(features) and parameters(weights) are represented by vectors and matrices. A DNN has several layers, each layer is an activation function with an argument of matrix multiplication. One after another this basic operation, a more abstract representation will be the output of the higher layers. Rumelhart, Hinton, and Williams's seminal 1986 paper [121] built the foundational position of matrices in DNN, which also came up with backpropagation and optimizing the matrix of weights by gradient descent.

The book ”Matrix Computations” by Golub and Van Loan[44] is not a book focusing on neural networks, but it improves the understanding of matrix operation in deep learning architectures. Training neural networks efficiently with matrix computation is also emphasized by Hinton, Osindero, and Teh in their development of a fast learning algorithm for deep belief networks [60].

CNNs expand the matrix operations within DNNs to that of tensor operations to take appropriate multi-dimensional data such as images. In this sense, LeCun et al. employed tensors to CNNs when the gradient-based learning in the research used them for document recognition [85]. Another significant research that used multi-dimensional operations of tensors is the ImageNet study conducted by Krizhevsky et al. To do image classification [79] work effectively with the deep convolutional networks, the images were processed by the multi-dimensional operations of tensors.

TensorFlow is specially introduced by Abadi et al. first, to facilitate this complex tensor operation. It is a flexible and powerful framework that supports massive machine-learning projects using a distributed network. The design of the TensorFlow processing system is specifically tailored for the intricate calculations on tensors that are used to train and build modern neural networks [1].

With the increasing complexity of CNNs, it was becoming crucial to understand and visualize what exactly is happening inside those networks. Zeiler and Fergus made notable steps in that direction, utilizing deconvolutional layers to visualize the activity within a CNN for the first time, thus explaining how these networks detect and process visual information [143].

Indeed, these developments are well-cataloged in the comprehensive review by LeCun, Bengio, and Hinton, which encapsulates deep learning with a discussion of the common thread between distinct models of neural networks [81] namely, the ubiquitous application of matrix and tensor operations. By focusing on the functional as well as conceptual applications across this class of machine learning methods, the review underscores the conceptual frameworks that drive ongoing innovative design.

To conclude, the progress from single data points, vectors, and matrices in simple neural networks to tensors in sophisticated structures such as flattened neural networks, RNNs, CNNs or transformers speaks to the deep connection and interdependence of linear algebra and deep learning. This connection continues to push forward the frontiers of AI, making it able to provide more and more complicated, efficient, and powerful solutions in many different spheres.

Calculus:

- **Differential Calculus:** Ultimately, differential calculus, specifically through gradient-based optimization using methods like SGD(stochastic gradient descent), underpins much of deep learning. Indeed, it is this mathematical landscape that is utilized to fine-tune the weights in the network to minimize a loss function, which overtly expresses the discrepancy between modeled predictions and the true data.

  Over the years, landmark researchers have laid the foundation to allow for the backpropagation of these calculated gradients through deep networks. Credited as the inventor of backpropagation, Rumelhart, Hinton, and Williams's seminal work that presented The backpropagation learning procedure marked the first occasion where the concept of the chain rule [121] as employed in calculus has been the enabler for learning.

  Subsequently, there have many advances in driving the utility of gradient-based learning further in convolutional networks. For example, LeCun, Bottou, Bengio, and Haffner presented how gradients can be used efficiently due to the use of tensor operations in spatially hierarchical data [85].

The more recent work by Krizhevsky, Sutskever, and Hinton, presented the AlexNet [79] model which advanced "the state of the art in classification by over 10 percentage points.

Additionally, gradient-based learning presented how the real state of effective learning with these methods, specifically given the vanishing gradient problem [84]. The introduction of the Adam optimizer by Kingma and Ba (2014), which adjusts learning rates based on gradient moment estimates, marks a notable advancement in stochastic optimization techniques in deep learning [73].

Introduced by Abadi, Agarwal, Barham et al., the TensorFlow framework represents the modern-day support for these operations in practice [1].

Ultimately, optimization using differential calculus is what has enabled numerous breakthroughs in the field of artificial intelligence.

- **Partial Derivatives and Chain Rule:** Backpropagation, the fundamental technique for training deep neural networks, relies on partial derivatives and the chain rule to efficiently calculate gradients. As a result, the computational theory enables accurate adjustment of the weights to reduce the loss functions during the learning step.

Rumelhart, Hinton, and Williams defined backpropagation as the transmission of errors through the network in reverse via a specific algorithm that calculates error derivatives, using the chain rule [121]. This development essentially delivered a practical computational theory that allowed neural nets to be widely utilized.

In contrast, in the article "Efficient BackProp," LeCun et al. improved the method by identifying numerous numerical problems including the guaranteed learning rates and divergence and vanishing gradients [84]. This particular description focused on ensuring the computational understanding and improvement verification continued to be even in training. Examples of applications for these theoretical computations included testing of backpropagation by LeCun, which showed the practical application of the method in the numeral recognition of zip codes. This was a substantial practical accomplishment since the rectified gradient optimization methods can work across all collections.

Similarly, Hinton et al. introduced the fast learning algorithm [60] which involved backpropagation from efficient gradient computation in layers such as with the chain rule. This was a substantial milestone in the development of the field, that led to even broader applications like TensorFlow systems [1].

Inversely, large implementation can also prove complex since, as Glorot and Bengio discovered, training deep networks is hard due to problems such as weight initialization crucial to the success of gradient-based optimization methods [43].

In conclusion, backpropagation development was a critical development in deep learning that was facilitated by the mathematics of differential calculus. Efficient gradient computation through the chain principle enabled the training of intricate models and drove many of the being theories and case applications. Considering that neural nets are becoming more complex, the mathematical findings remain a critical part of further discoveries.

Probability Theory: The basis of understanding neuron-level behavior in neural networks through probabilistic lenses.

- **Probabilistic Interpretation:** While deep learning models are famous for their depth and complexity, it is through the application of a probabilistic interpretation that these can be significantly improved. After all, the treatment of outputs as probabilities and the application of the maximum likelihood estimation, during training, not only allow for solid predictions under uncertainty but also build a strong foundation to understand how the model can be further improved.

David Heckerman's 1995 study of Bayesian networks provided a framework for probabilistic methods in learning systems, demonstrating that such networks could efficiently handle uncertainty and learn from data in an organized manner. Specifically, he offered proof that Bayesian techniques could be integrated into more complex models to increase their interpretability and practicality [57].

Further, Radford M. Neal extensively discussed probabilistic modeling within a neural network context, underscoring its importance, in his 1996 book on training such networks. Therein, Neal noted that a probabilistic approach was vital in terms of managing model uncertainty and diminishing overfitting through Bayesian inference [101].

More practically, Hinton, Osindero and Teh's 2006 study on a fast learning algorithm for deep belief nets using a probabilistic generative model integrated a layered maximum likelihood estimation approach. The study illustrates an efficient way of training deep networks under probabilistic auspices [60].

Probabilistic modeling's importance within the realm of neural network operation was further noted by LeCun et al. in 1998, during their work on document recognition with convolutional neural networks. The study demonstrated how gradient-based learning can substantially, and effectively, be extended with probabilistic models, especially in managing image data [85]. Michael I. Jordan's 1995 paper provided a long discussion on the probabilistic interpretation of network outputs during classification [70].

More than that, it had been proposed in 2016 by Gal and Ghahramani to use dropout as an approximation of the Bayesian approach, indicating that the popular regularization method for neural network training could also be perceived as a form of approximate

Bayesian inference [39].

That enabled a probabilistic view of dropout, which could be applied in more complex settings including training recurrent neural networks. For example, Pascanu, Mikolov, and Bengio 2013 examined the difficulty associated with training on gradients and likelihood in RNN [105].

In addition, Rezende and Mohamed 2015 significantly expanded the flexibility for applying probabilistic models in deep learning, offering variational inference with normalizing flows. This approach allows for enhancing the accuracy and efficiency of probabilistic inference in advanced neural architectures, which would be impossible with other methods [114].

Including probabilistic decision-making in deep reinforcement learning, Mnih et al. created a way to train the deep Q-network by introducing probabilistic decision-making to optimize expected returns, presenting maximum likelihood estimation as a key method in defining action-value functions [97].

Last but not least, Kawaguchi, Kaelbling, and Bengio 2017 investigated generative possibilities in the framework of a probabilistic view of deep learning [72].

Hence, not only does the theoretical foundation of deep learning models become stronger with probabilistic interpretations, but the practical implementation of probabilistic approaches makes them more stable, interpretable, and powerful solutions for the provision of more reliable machine learning-based solutions.

- **Bayesian Inference:** BNNs(bayesian neural networks) leverage the above principles to quantify prediction uncertainty by adopting Bayesian principles. The Bayesian approach is particularly relevant in cases of making decisions under uncertainty, which for various tasks can have a significant impact on the outcome, provided the degree of confidence in forecasts is known.

Radford M. Neal's pioneering book on Bayesian Learning for Neural Networks extensively covers the fundamental theorem of Bayesian inference for neural networks. This theorem posits that initial beliefs are consistent with subsequent evidence before being modified to form a posterior distribution over the model parameters. It is essential not only because it intrinsically involves uncertainty in the process, thereby enhancing generalization by incorporating prior, reducing the threat of over-fitting into neural networks [101].

For practical tasks, Charles Blundell and his colleagues have introduced a powerful variational Bayesian technique for learning weight distributions in neural networks, allowing for straightforward uncertainty estimation in predictions, and increasing input invariance among other improvements [13]. This has been expanded upon by Kumar Shridhar and researchers, who have successfully applied Bayesian inference to convolutional networks to handle uncertainty in large-scale neural network procedures used in high-dimensional tasks [127].

Alex Graves has also presented a talk on Practical Variational Inference for Neural Networks to emphasize that Bayesian techniques are always feasible oversampling techniques, thus making them accessible for everyday use in real-world applications and commercial environments [47].

Yarin Gal and Zoubin Ghahramani introduced Dropout as a Bayesian approximation, making a valuable contribution to understanding regularization techniques from a Bayesian perspective. Kalchbrenner examined the possibility of simulating averaging over a set of neural network architectures with dropout, which offers an approximation to the Bayesian model average [39].

David Krueger et al. invented Bayesian Hyper networks. Specifically, this article provides a new method for learning the weight distribution of larger neural networks by managing uncertainty in advanced models [80].

Vincent Fortuin et al. revisited Bayesian neural network priors, indicating how different priors on neural network weights affect model performance in terms of uncertainty assessment. The authors argue that the selection of the prior should account for model flexibility balanced with informativeness [37].

Jose et al. introduced a scalable probabilistic training algorithm for Bayesian neural networks. A probabilistic backdrop provides the scaling properties necessary for industrial problems while integrating a principled approach to modeling uncertainty and predictions [58].

Andrew G. Wilson et al. presented deep kernel learning which combined GPs(Gaussian processes) with neural networks to improve existing methods. This article provides a reliable model by using the combined strategy of deep learning and Bayesian statistics to make the prediction reliable and interpretable [141].

At last, Brochu and his colleagues, offer a tutorial on Bayesian Optimization, which helps practitioners optimize costly functions. It is an essential contribution to determining which settings of a neural network are optimal, as trying all possible configurations is prohibitively expensive [14].

In conclusion, a Bayesian neural network combines probabilistic modeling with neural networks to deliver substantial added value in terms of the prediction's robustness and credibility, valuable in scenarios where it is uncertain.

Information Theory:

- **Entropy and Information Gain:** Information theory, which is a mathematical framework that quantifies the transmission of information, dramatically improves the understanding and optimization of learning algorithms. The focus on the information in the outputs and the individual data points' insights present crucial views on how efficient and effective the given machine learning models are.

This technology was first used by Claude E. Shannon in 1948, and its main points were captured in ”A Mathematical Theory of Communication”. It utilizes entropy as a measure of the amount of information that an output or datum could contain. His ideas form the basis for the later use of learning algorithms, where understanding the entropy of a set of data helps in refining the data encoding and processing strategy [125].

J. Ross Quinlan brought the development of information theory into action with his 1986 decision tree learning. Quinlan realized that since entropy played a crucial role in developing accurate models, most machines that utilized this were created through entropy measures, He explained how decision trees were most effectively expanded by choosing the splits that yielded the highest information gain for minimum entropy at each decision node. His ideas behind their improvements became the fundamental development unit for other models [109, 110].

In statistical learning, originators of several models incorporate their ideas into ”The Elements of Statistical Learning” [55]. However, based on an early idea of how neural networks might be able to model the information theory of information systems in action, is where Frank Rosenblatt in ”Principles of Neurodynamics” expounded [119]. Other non-standard models such as information-theoretic measures Kraft et al., 2006 [78] and Beal et al. [10], 2003 were both able to use entropy in decoding complex data as well as enhancing feature clustering.

Doquire and Verleysen in 2013 utilized information theory principles to develop algorithms for feature selections under constraints [29]. Similarly, Sietsma and Dow in 1988 used entropy-based pruning to simplify model complexity so that it preserved essential information hence boosting network efficiency [129]. Kohavi and John's study on feature selection during the training process of supervised neural networks used the information theory concept via mutual information to boost efficiency [75].

In conclusion, the application of entropy and information gains from information theory significantly enriches machine learning methodologies, providing a robust mathematical framework for optimizing learning algorithms and enhancing decision-making processes.

Optimization Theory:

- **Convex and Non-Convex Optimization:** The optimization landscape of neural networks consists of non-convex functions, and the very presence of local minima, saddle points, and highly elusive global minima makes the training process without a solid understanding of mathematical principles and advanced optimization methods.

Yann LeCun and coauthors devoted their ground-breaking 1998 paper,
"Gradient-Based Learning Applied to Document Recognition" to
discussing non-convex loss functions in training convolutional neural networks, thus reflecting one of the
original works to recognize the practical challenge [85].

Another original 2010 research by Xavier Glorot and Yoshua Bengio, "Understanding the Difficulty of Training Deep Feedforward Neural Networks," was also focused on the challenging aspects of non-convex objective functions in terms of training dynamics for deep networks, which is another way to say that their non-convex nature makes the dynamics challenging [43].

Another paper by Yann LeCun and others, “Efficient Backprop” also devoted to optimization strategies, indicated one of the ways to deal with these challenges [84]. Pascanu et al. also published an important research paper in 2014, “On the Noisy Gradient Problem,” which allowed researchers to understand more clearly what happens at local minima and plateaus specifically when dealing with convolutional networks [105].

Finally, the 2009 work “Learning Deep Architectures for AI” by Yoshua Bengio was, without doubt, one of the most groundbreaking research papers [11]. Additionally, Neil Parikh’s 2014 study on “Proximal Algorithms” provided one of the robust mathematical approaches toward non-convex optimization phenomena to broaden the scope of tools available for neural network work [104]. Besides, Choromanska et al. overwhelmingly contributed a better understanding of saddle points in high-dimension spaces in their 2015 “On the Loss Surfaces of Neural Networks” to tackle the complexity of everything that happens in convolutional neural networks [20].

Goodfellow et al.’s 2015 paper, “Global Optimization Algorithms for Training Deep Neural Networks,” was also a substantial contribution [45]. However, concerning practical aspects of random non-convex optimization, Kingma and Ba’s 2014 “Adam: A Method for Stochastic Optimization” should be specifically mentioned [73]. Lastly, a wide perspective on non-convex optimization was also presented in Jain and Kar’s 2017 “Non-Convex Optimization for Machine Learning” [68].

This vast corpus outlines the work done in understanding the optimization landscape of neural networks from a mathematical and computational angle: from providing theoretical results on what happens in the non-convex environment to creating practical tools resulting from this theoretical understanding.

Graph Theory:

- **Network Architecture:** The conceptualization of deep neural networks as DAGs(directed acyclic graphs) has brought an immense layer of clarity on complex network architectures and their behaviors. This framework of viewing networks, where each node is a computation unit while edges are data flows or dependencies, crucially illuminates the structures and functionalities of relatively recent models such as deep residual networks and densely connected networks.

The introduction of DAGs into neural network architecture happened when Kaiming He and his team developed deep residual networks in 2015. The addition of skip connections that collectively form a DAG allowed the training of networks deeper than ever before and propagated the signal directly to the other end effectively solving the problem of vanishing gradients in deep networks [56].

Similarly, Gao Huang et al. introduced the concept of densely connected convolutional networks, another peculiar implementation of a DAG where each layer became directly accountable to all other precedent layers. This form largely increased network efficiency and efficacy as it largely reduced the parameter count and encouraged feature reuse, crucial in most deep-learning implementations [66].

Furthermore, Matthew D. Zeiler and Rob Fergus further advanced the understanding of the data flow in neural networks by developing tools to visualize and understand the activations and representations made by low-level and mid-level layers [143].

Beyond simply DAG implementations, Min Lin et al. introduced functional mini-neural networks into convolutional layers in the Network In Network structure. This increases the abstraction and develops a deeper and more intricate pyramid structure over the DAG [87].

In graph-based applications, DAGs have also opened up new possibilities. Thomas N. Kipf and Max Welling developed graph convolution networks that extend the DAG implementation paradigm to inherently graphical data to each node [74].

Similarly, Petar Veličković et al.'s work shows how the attention mechanism can be integrated into the graph. Their work created a new neural network layer that dynamically assigns greater weight to different nodes for the neural network depending on their importance in the relevant subset of data [137]. Yifan Feng et al. used a similar paradigm to include hypergraphs in neural networks, extending the reach of the traditional graph theory into more complex relationships beyond pairwise ones and assisting representation and learning on multiway collaborative data [34].

These examples are only a small fraction that can show how varied the application of DAG has changed the landscape of deep learning and impacted its scalability. It would be near-impossible to conceptualize newer, more complex designs without this additive paradigm.

2.3 Review of Previous Empirical Research

Applications of Probability in Neural Networks: Probability theory plays a critical role in enhancing the predictive accuracy and stability of neural networks. Researchers have utilized various probability distributions to model uncertainties in neural network predictions. For instance, Bayesian neural networks use probabilistic inference to manage uncertainty, improving decision-making processes in complex environments [101, 13]. These networks incorporate prior distributions overweights and biases, allowing the model to make more robust predictions under input variability and data scarcity.

Quantitative Analysis Techniques: Quantitative methods for analyzing neural network behavior include activation analysis and feature visualization. Activation analysis involves examining the activations produced by neurons to understand which features are being detected, and which are most influential for the network's output. Techniques like LRP(Layer-wise Relevance Propagation) provide insights into how neural activations contribute to final decision-making [8]. Additionally, feature visualization methods like those developed by Zeiler and Fergus (2014) help in interpreting the functionalities of individual network layers, clarifying how neural networks process input data to form decisions [143].

Empirical Validation of Algebraic Models: Algebraic models have been extensively validated empirically in various domains such as image recognition, natural language processing, and decision systems. For example, convolutional neural networks, fundamentally based on matrix algebra, have revolutionized image processing by enabling hierarchical feature extraction. Studies such as He et al. (2016) on ResNets(Residual Networks) demonstrate how deep learning models can be algebraically optimized to perform tasks with increased efficiency and reduced error rates [56]. In natural language processing, models like transformers employ positional encoding and attention mechanisms, which are grounded in linear algebra, to manage and predict language sequences effectively [135].

2.4 Gaps in the Literature

The review identifies several key gaps in the current literature:

Lack of Unified Algebraic Approaches: In recent efforts to harmonize the diverse algebraic methods utilized in neural network architectures, emerging research points towards developing a more integrated theoretical framework. For instance, the study "A Unified Algebraic Perspective on Lipschitz Neural Networks" by Araujo et al. (2023) proposes an algebraic framework aimed at crafting 1-Lipschitz neural networks under a semidefinite programming condition.

This method not only boosts adversarial robustness but also seeks to connect the disparate layers of neural networks under a unified algebraic approach [4].

Simultaneously, ”Categorical Deep Learning: An Algebraic Theory of Architectures” by Gavranović et al. (2024) investigates the application of category theory in the design of neural network architectures. This study critiques the lack of a cohesive framework in current models and suggests employing the universal algebra of monads within a 2-category of parametric maps as a solution. This innovative approach aims to reframe constraints encountered in geometric deep learning and extend its application across various architectures, demonstrating the versatility and profound impact of category theory on the foundational aspects of neural network design[41].

Together, these studies highlight the ongoing efforts and the vital necessity for a unified algebraic approach that could coherently consolidate and perhaps simplify the myriad mathematical models employed across different neural network architectures. Such a unified approach promises to foster the development of more robust, scalable, and theoretically solid neural network systems, thus facilitating significant strides in both theoretical insights and practical implementations.

Neuron-Level Quantitative Analysis: Recent advancements in neuron-level quantitative analysis are significantly enhancing our understanding of neural networks. This research delves into the complexities of individual neuron functions and their impact on overall network behaviors. Studies emphasize the critical nature of this detailed approach across various applications, particularly in NLP(deep natural language processing) models, where specific neuron activities are correlated with linguistic features, and in efforts to improve network reliability through the quantification of individual neuron vulnerabilities [6, 113]. Additionally, visualization techniques are instrumental in clarifying the training processes at the neuron level, rendering the intricate dynamics within networks more comprehensible and interpretable [32].

Furthermore, research that bridges the divide between biological neuron models and artificial neural networks is providing essential insights that enhance both the theoretical and practical aspects of neural modeling [67]. Practical tools like the NNS(Neural Network Scanner) illustrate the application of these findings by enabling detailed examinations of neuron behavior. This facilitates the identification of learning patterns and potential biases, crucial for refining AI training processes [26].

Collectively, these approaches not only improve the transparency and reliability of neural networks but also foster the development of AI innovations that are both interpretable and robust. This ongoing research continues to push the boundaries of what is achievable with artificial intelligence, ensuring that neural networks are both effective in performance and grounded in a deep understanding of their underlying mechanisms.

Integration of Algebraic Models with Practical Tools: Few studies have successfully bridged the gap between theoretical algebraic frameworks and their practical, accessible application in everyday neural network development and analysis.

The integration of theoretical algebraic frameworks with practical applications is essential for the advancement of AI technologies. The study ”Algebraic Neural Networks: Stability to Deformations” [7] effectively utilizes algebraic principles to bolster the robustness of neural networks. This application of theoretical insights demonstrates a clear pathway from abstract concepts to tangible, practical enhancements in network stability, bridging theoretical algebraic theory with real-world neural network resilience.

Additionally, ”Computational Algebraic Topology and Neural Networks in Computer Vision” [112] showcases the successful application of TDA(topological data analysis) alongside deep learning techniques to improve image analysis tasks within computer vision. This research highlights the valuable contributions of sophisticated mathematical theories, like algebraic topology, in addressing complex problems by enhancing the accuracy and efficiency of AI systems.

These instances emphasize the critical role and effectiveness of incorporating advanced mathematical frameworks into practical settings, showcasing the profound impact such integrations can have on the development of technology and AI. This strategy not only improves the functionality of neural networks but also illustrates how theoretical models can be concretely applied to advance technological processes and outcomes across various AI domains. This marriage of theory and practice serves as a model for future developments in AI technology, pushing the boundaries of what can be achieved through the fusion of deep theoretical insights and innovative application strategies.

2.5 Statement of the Research Problem

The identified gaps underscore the need for a comprehensive algebraic framework that can be systematically applied to analyze and interpret deep neural networks at a neuron-level granularity. This research seeks to address the absence of such a framework, which is crucial for advancing the understanding and application of deep learning models beyond reliance on intuition and empirical adjustments.

2.6 Research Questions/Objectives

Derived from the gaps in the literature, the research questions focus on:

1. Can a unified algebraic framework based on 6-Set Discrete Probability Algebra effectively model and analyze neuron-level behaviors in various neural network architectures?

2. How can this algebraic framework be integrated into a practical software tool to facilitate wider application and accessibility?

3. What improvements in neural network design, optimization,
and interpretation can be achieved through the application of this framework?

2.7 Summary

This literature review establishes the theoretical and empirical backdrop against which this research is set, highlighting the innovative aspects of the proposed Unified Algebraic Framework and its potential to fill the identified gaps in the field. The subsequent chapters will delve deeper into the theoretical development of the framework, its empirical validation, and its application across different deep-learning scenarios.

CHAPTER 3

AN EXTENSIBLE 6-SET DISCRETE PROBABILITY ALGEBRA AND UNIFIED ALGEBRAIC FRAMEWORK

3.1 Abstract

This chapter mathematically defines a special algebra named a 6-Set Discrete Probability Algebra. These six elements from the six sets are combined by a unified flow to implement an algorithm that can quantize the characteristics of a specific neuron and quantitatively systematically measure different dimensional distances between the outputs of different neurons. These quantified results are mainly focused on measuring the information included by an output of a neuron with inputs from a sample dataset, the similarity and dissimilarity of the output distribution from different neurons. Based on this 6-Set algebra, a unified algebraic framework is built in the form of a Python package. This framework redefines the existing measurement including Euclidean distance, entropy, divergence, IPM(integral probability metric), MMD(maximum mean discrepancy), and Wasserstein distance, it also makes some extensions. All these measures are called UAM(unified algebraic measure) in a new context. Each method in the framework also provides visualization results to show the measures of the output of neurons. The unified algebraic framework prepares their applications in different sectors of deep learning.

3.2 Introduction

3.2.1 Background

In recent years, the field of neural networks and deep learning has experienced significant advancements, enhancing a wide range of applications like image recognition, natural language processing, and autonomous systems. A vital component of these advancements involves understanding the behavior and characteristics of individual neurons within a network. Neurons, the core units of neural networks, process and transmit information crucial to the network's overall performance. However, quantifying and comparing the outputs of these neurons systematically and effectively remains a complex challenge, especially with high-dimensional data and intricate interactions.

Traditional methods, such as Euclidean distance and basic statistical measures(See Appendix D on page 364), often fail to capture the complex patterns and relationships in the data. These methods, while providing initial insights, do not suffice for an in-depth analysis, particularly when aiming to comprehend the deeper layers of neural networks and the nuanced behaviors of neurons across various contexts.

To overcome these challenges, the innovative 6-Set Discrete Probability Algebra has been developed. This new mathematical framework offers a more robust and systematic method for measuring and analyzing neuron outputs. Utilizing six distinct sets, it integrates various data dimensions, allowing for a comprehensive and nuanced analysis of neuron outputs and their interactions. This approach not only increases measurement precision but also provides fresh insights into the similarities and differences among neuron outputs across different neurons and datasets.

Grounded in the principles of probability and algebra, the 6-Set Discrete Probability Algebra merges these fields to form an extensive toolkit for analyzing neural networks. This framework quantifies specific neuron characteristics and systematically measures the dimensional distances between different neurons' outputs. The results emphasize critical aspects such as the information contained in a neuron's output, the distribution of these outputs, and the level of similarity or difference between outputs from various neurons.

3.2.2 Purpose

The 6-Set Discrete Probability Algebra framework is a pioneering development designed to advance neural network analysis and enhance our understanding of neuron behavior. Its primary aim is to offer a systematic and thorough approach to quantifying neuron outputs, overcoming the limitations of traditional methods and introducing new capabilities for in-depth analysis.

A key goal of this framework is to enhance traditional measurement techniques such as Euclidean distance, entropy, divergence, and various probability metrics that are extensively used in fields like statistics, information theory, and machine learning. These methods typically measure distances and dissimilarities between probability distributions, but they often fall short when applied to the complex and high-dimensional data of neural network analysis.

The 6-Set algebraic framework improves upon these methods by integrating additional data dimensions and offering a unified measurement approach. This enhancement allows for more precise and comprehensive quantification of neuron outputs, capturing intricate patterns and relationships that simpler methods may miss. For example, the framework can simultaneously evaluate multiple criteria to measure neuron output similarities, offering a more complete view of their behavior.

Additionally, the framework introduces novel methods and extensions to tackle specific challenges in neural network analysis. These new tools are designed to capture nuanced aspects of neuron behavior, such as the temporal dynamics of outputs, the influence of different input features, and the interactions between neurons across various network layers. This comprehensive toolkit enables deeper insights into neural network functionality and supports more sophisticated research and applications.

Implemented as a Python package, the 6-Set algebraic framework becomes even more accessible and user-friendly. Python's widespread use in data analysis, machine learning, and scientific computing makes it an ideal platform for this framework. The package includes numerous functions and tools for quantifying and visualizing neuron outputs, making it easy for researchers and practitioners to apply the framework to their datasets and neural networks.

A notable feature of this Python package is its ability to generate visualizations, which are vital for interpreting neuron output measurements. These visualizations aid in intuitively understanding the distribution and relationships among neuron outputs, facilitating the identification of patterns and the derivation of meaningful conclusions. The package provides various visualization options, including scatter plots, heatmaps, and dimensionality reduction techniques, to effectively display the quantified results.

This unified algebraic framework, through its Python package implementation, sets the stage for diverse applications across different deep learning sectors. In image recognition, it can analyze neuron-learned features and their contributions to final classifications. In natural language processing, it can explore word embedding patterns and linguistic feature relationships. In autonomous systems, it can be used to examine neural network decision-making processes, enhancing their robustness and reliability.

In summary, the 6-Set Discrete Probability Algebra marks a significant leap forward in neural network analysis. Providing a robust, systematic approach to quantifying neuron outputs, this framework addresses traditional method limitations and introduces new analysis capabilities. Available as an easy-to-use Python package, it enables broad application in research and practical settings, paving the way for deeper neural network insights.

3.3 Literature Review

3.3.1 Existing Mathematical Frameworks

In the field of computational mathematics, various algebraic frameworks have emerged to tackle challenges in data analysis, machine learning, and neural network theory. These frameworks are grounded in solid mathematical principles, designed to quantify relationships and measure distances between data points or distributions. Among the most prominent are vector spaces, metric spaces, and probability spaces.

Vector spaces [46, 117, 64, 52, 118] offer a fundamental structure crucial for numerous mathematical and computational tasks. They allow data points to be represented as vectors within a multidimensional space, supporting operations like addition, scalar multiplication, and linear transformations. Despite their broad utility, vector spaces sometimes struggle to capture the complex, non-linear relationships found in the high-dimensional data of neural networks.

Metric spaces build upon vector spaces by incorporating a metric or distance function to determine the distance between any two points. This metric facilitates the assessment of similarity or dissimilarity among data points, employing common measures such as the Euclidean, Manhattan, and Chebyshev distances. Yet, these traditional metrics may prove insufficient for analyzing the intricate data of high-dimensional neural networks, necessitating more advanced distance measures to address complex data relationships.

Probability spaces [77, 96, 33, 12, 125] address uncertainty and randomness in data, consisting of a sample space, a set of possible events, and a probability measure. These spaces are essential for statistical analysis and machine learning, providing a means to quantify the likelihood of different outcomes. However, probability spaces alone may not meet the need for a holistic approach to measure and compare outputs in neural networks.

Although these existing frameworks have significantly advanced computational mathematics, they exhibit limitations when applied specifically to neural network analysis. Traditional algebraic and probabilistic methods might not fully capture the intricate, multi-dimensional nature of neuron outputs, underscoring the need for innovative frameworks. The development of structures like the 6-Set Discrete Probability Algebra aims to offer a more comprehensive and integrated approach to quantify and analyze neuron characteristics, addressing gaps left by previous models.

3.3.2 Existing Engineering Framework

In addition to mathematical frameworks, several engineering frameworks have been developed to facilitate the implementation and analysis of neural networks. These frameworks typically provide tools and libraries for building, training, and evaluating neural networks, with a focus on practical applications and performance optimization.

One of the most widely used engineering frameworks in the field of deep learning is TensorFlow [2], developed by Google. TensorFlow offers a flexible and efficient platform for constructing neural network models, supporting various types of layers, loss functions, and optimization algorithms. It also provides tools for visualizing model performance and debugging. Despite its versatility, TensorFlow primarily focuses on the engineering aspects of neural network development and may not offer advanced tools for the detailed quantification and analysis of neuron outputs.

Another popular framework is PyTorch [106], developed by Facebook's AI Research lab. PyTorch is known for its dynamic computational graph, which allows for more intuitive model development and debugging. It supports a wide range of neural network architectures and provides tools for visualization and performance analysis. However, similar to TensorFlow, PyTorch is primarily oriented toward practical implementation and may lack advanced analytical capabilities for neuron output measurement.

Keras [19], a high-level neural networks API written in Python, provides an accessible interface for building and training models using backend engines like TensorFlow and Theano.

Keras emphasizes ease of use and rapid prototyping, making it a popular choice for researchers and practitioners. While Keras simplifies many aspects of neural network development, it also does not inherently offer the detailed analytical tools needed for comprehensive neuron output analysis.

The limitations of these engineering frameworks in terms of advanced analytical capabilities highlight the need for specialized tools like the 6-Set Discrete Probability Algebra. By integrating advanced mathematical and probabilistic methods, the 6-Set framework complements existing engineering frameworks, providing a more robust and systematic approach to neuron output analysis.

3.4 Background Information

3.4.1 Theoretical Foundations

The development of the 6-Set Discrete Probability Algebra is deeply rooted in foundational theories from algebra, probability, and information theory. Understanding these concepts is vital to fully appreciate the innovations brought about by this algebraic framework.

Central to the 6-Set Discrete Probability Algebra are the principles of set theory and algebraic structures. Set theory, which involves the study of collections of objects (sets), provides essential tools and language for defining and manipulating groups of data points. Within the 6-Set algebra, six distinct sets are identified, each corresponding to different dimensions or attributes of neuron output data.

Algebraic structures, including groups, rings, and fields, furnish a framework for defining operations on these sets and understanding their properties. This foundational support enables the systematic combination and manipulation of data. The 6-Set algebra utilizes these principles to establish operations that quantify relationships between diverse sets of neuron outputs.

Probability theory plays a crucial role by quantifying uncertainty and the likelihood of occurrences, allowing the algebra to accommodate the variability and randomness in neuron outputs. This inclusion enables a more refined analysis of neuron behavior, addressing the inherent uncertainty within neural network data.

Information theory, pivotal in measuring, storing, and communicating information, significantly influences the 6-Set algebra. It employs critical concepts like entropy and divergence to assess the information content in neuron outputs and to gauge the similarity or dissimilarity among different distributions. These metrics are essential for deciphering the information-processing capabilities of neurons and the interrelationships of their outputs.

By integrating these theoretical foundations, the 6-Set Discrete Probability Algebra offers a holistic toolkit for analyzing neuron outputs. This amalgamation of set theory, algebraic structures, probability theory, and information theory creates a robust and methodical framework for evaluating neuron characteristics. This theoretical synergy not only enhances the accuracy and depth of neuron output analysis but also promotes the development of innovative methods and extensions that address more subtle aspects of neuron behavior.

In conclusion, the 6-Set Discrete Probability Algebra stands on solid theoretical ground, incorporating established principles from set theory, algebraic structures, probability theory, and information theory. These foundations underpin the framework's innovative approach to neuron output analysis, allowing for a more thorough and systematic quantification of neuron characteristics. By tapping into these theoretical resources, the 6-Set algebraic framework overcomes traditional methodological limitations and introduces enhanced capabilities for a detailed and nuanced analysis of neuron outputs in neural networks.

3.5 Theoretical Development

3.5.1 Mathematical Formulation

A Definition of 6-Set Discrete Probability Algebra

A 6-Set Discrete Probability Algebra (6$\mathfrak{S}$-Algebra) is a 6-Set field $\{\mathbb{S}, \mathscr{F}, \mathscr{M}, \mathbb{P}, \mathscr{G}, \mathscr{H}\}$ equipped with UAMs(Unified Algebraic Measures). UAMs can be UAPs(Unified Algebraic Properties) and UADs(Unified Algebraic Distance). UAD is an operation to measure the quantitative property of the outputs of a neuron that digests a collection of data points, and UAD is an operation to measure the quantitative distance between the outputs of two neurons that digests a collection of data points. The elements in the 6-Set field are as follows. UAP and UAD are defined in the next two subsections.

- **Sample Set (S):** This is a collection of data points $\{s_1, s_2, \ldots\}$ used to train and evaluate a neural network. Each data point is an element of S. The data point can be a number, a vector, a matrix, or a sequence of vectors or matrices.

- **Set of Sample Sets ($\mathbb{S}$):** This is the power set of S, including all possible sample sets $\{S_1, S_2, \ldots\}$. Each sample set S includes a collection of data points $\{s_1, s_2, \ldots\}$.

- **Sample Function (f):** These functions operate on elements in S. Functions can include neural network operations, neural network models, neural subnetworks, or other element-wise operations on the data points. See Appendix B on page 276.

- **Family of Sample Functions ($\mathscr{F}$):** This is the power set of sample function(f), including all functions $\{f_1, f_2, \ldots\}$ that can operate on S.

- **Metric Function (m):** A function that defines the measurement of a data point in S. Must satisfy the properties of a metric (non-negativity, symmetry, triangle inequality).

- **Family of Metric Functions ($\mathscr{M}$):** This is the power set of metric function (m), including all metric functions $\{m_1, m_2, \ldots\}$.

- **Discrete Probability Distribution (P):** Defines a discrete probability distribution over S, based on the measurement defined by a metric m in $\mathscr{M}$. The discrete Probability function can be PMF(Probability Mass Function), PDF(Probability Density Function, mainly histogram or binned probabilities), CDF (Cumulative Distribution Function), or EDF (Empirical Distribution Function).

- **Set of Discrete Probability Distributions ($\mathbb{P}$):** A collection of all possible discrete probability distributions P that can be defined on S using any metric m in $\mathscr{M}$.

- **Functions on Discrete Probability Distribution (g):** Functions that take a discrete probability distribution P as input, and output a new discrete probability distribution N. This is a kind of transformation that operates on discrete probability distribution.

- **Family of Functions on Discrete Probability Distribution ($\mathscr{G}$):** This a power set of Functions on Discrete Probability Distribution (g), including all functions $\{g_1, g_2, \ldots\}$ that operate on elements of $\mathscr{P}$.

- **Functions on Integral of Discrete Probability Distribution (h):** Functions that use the integral (or cumulative distribution function) of P as input, output a number.

- **Family of Functions on Integral of Discrete Probability Distribution ($\mathscr{H}$):** This is a power set of Functions on Integral of Discrete Probability Distribution (h), including all functions $\{h_1, h_2, \ldots\}$ that operates on the integrals of P.

B Definition of UAP(Unified Algebraic Property)

UAP is a 6-step algorithm to calculate a number to represent the property of the outputs of a neuron in a neural network. In this context, the property needs the participation of all the elements in the outputs.

Mathematically, the formula of UAP is,

Defines a set S containing elements $s_1, s_2, \ldots, s_n$.

$$S = \{s_1, s_2, \ldots, s_n\} \tag{3.1}$$

Step 1: Applies function f to each element in set S.

$$f(S) = \{f(s_1), f(s_2), \ldots, f(s_n)\} \tag{3.2}$$

Step 2: Applies function m to each element in $f(S)$.

$$m(f(S)) = \{m(f(s_1)), m(f(s_2)), \ldots, m(f(s_n))\} \tag{3.3}$$

Step 3: Converts the set $m(f(S))$ to probabilities, resulting in set P.

$$P = \{p_1, p_2, \ldots, p_n\} = \text{Sample\_To\_Probability}(m(f(S))) \tag{3.4}$$

Step 4: Applies function g to each element in set P.

$$g(P) = \{g(p_1), g(p_2), \ldots, g(p_n)\} \tag{3.5}$$

Step 5: Each function g_i is applied to set P, resulting in a list of function applications.

$$[g_1, g_2, \ldots, g_n](P) = [g_1(P), g_2(P), \ldots, g_n(P)] \tag{3.6}$$

Step 6: Applies function h on integrals to get UAP.

$$UAP(S, f, m, P, [[g_1, g_2, \ldots, g_n]], h) = h\left([[\sum g_1(P), \sum g_2(P), \ldots, \sum g_n(P)]]\right) \tag{3.7}$$

C Definition of UAD(Unified Algebraic Distance)

UAD(Unified Algebraic Distance) is a measurement of distance, similarity and dissimilarity, or difference between two sample sets. We define three categories of measurements: E_UAD(Euclidean UAD), UAD_SF(UAD based on Sample Function), and UAD_PF(UAD based on Probability Function).

a *E_UAD(Euclidean UAD)*

Mathematically, the formula of E_UAD is,

Defines a set S containing elements $s_1, s_2, \ldots, s_n$.

$$S = \{s_1, s_2, \ldots, s_n\} \tag{3.8}$$

Applies function f to each element pair in set S.

$$E\_UAD(S, T) = f(S, T) = f(s_1, s_2, \ldots, s_n, t_1, t_2, \ldots, t_n) \tag{3.9}$$

b ***UAD_SF(UAD based on Sample Function)***

Mathematically, the formula of UAD_SF is,

Defines a set S containing elements $s_1, s_2, \ldots, s_n$.

$$S = \{s_1, s_2, \ldots, s_n\} \tag{3.10}$$

Defines a set T containing elements $t_1, t_2, \ldots, t_n$.

$$T = \{t_1, t_2, \ldots, t_n\} \tag{3.11}$$

Step 1: Applies function f to each element in set S, T.

$$f(S,T) = \{f(s_1,t_1), f(s_2,t_2), \ldots, f(s_n,t_n)\} \tag{3.12}$$

Step 2: Applies function m to each element in $f(S,T)$.

$$m(f(S,T)) = \{m(f(s_1,t_1)), m(f(s_2,t_2)), \ldots, m(f(s_n,t_n))\} \tag{3.13}$$

Step 3: Converts the set $m(f(S,T))$ to probabilities, resulting in set P.

$$P = \{p_1, p_2, \ldots, p_n\} = \text{Sample\_To\_Probability}(m(f(S,T))) \tag{3.14}$$

Step 4: Applies function g to each element in set P.

$$g(P) = \{g(p_1), g(p_2), \ldots, g(p_n)\} \tag{3.15}$$

Step 5: Each function g_i is applied to set P, resulting in a list of function applications.

$$[g_1, g_2, \ldots, g_n](P) = [g_1(P), g_2(P), \ldots, g_n(P)] \tag{3.16}$$

Step 6: Applies function h on integrals to get UAD_SF.

$$UAD\_SF(S, T, f, m, P, [[g_1, g_2, \ldots, g_n]], h) = h\left([[\sum g_1(P), \sum g_2(P), \ldots, \sum g_n(P)]]\right) \tag{3.17}$$

c *UAD_PF(UAD based on Probability Function)*

Mathematically, the formula of UAD_PF is,

Defines a set S containing elements $s_1, s_2, \ldots, s_n$.

$$S = \{s_1, s_2, \ldots, s_n\} \tag{3.18}$$

Defines a set T containing elements $t_1, t_2, \ldots, t_n$.

$$T = \{t_1, t_2, \ldots, t_n\} \tag{3.19}$$

Step 1: Applies function f to each element in set S.

$$f(S) = \{f(s_1), f(s_2), \ldots, f(s_n)\} \tag{3.20}$$

Applies function f to each element in set T.

$$f(T) = \{f(t_1), f(t_2), \ldots, f(t_n)\} \tag{3.21}$$

Step 2: Applies function m to each element in $f(S)$.

$$m(f(S)) = \{m(f(s_1)), m(f(s_2)), \ldots, m(f(s_n))\} \tag{3.22}$$

Applies function m to each element in $f(T)$.

$$m(f(T)) = \{m(f(t_1)), m(f(t_2)), \ldots, m(f(t_n))\} \tag{3.23}$$

Step 3: Converts the set $m(f(S))$ to probabilities, resulting in set P.

$$P = \{p_1, p_2, \ldots, p_n\} = \text{Sample\_To\_Probability}(m(f(S))) \tag{3.24}$$

Converts the set $m(f(T))$ to probabilities, resulting in set Q.

$$Q = \{q_1, q_2, \ldots, q_n\} = \text{Sample\_To\_Probability}(m(f(T))) \tag{3.25}$$

Step 4: Applies function g^p to each element in set P.

$$g^p(P) = \{g^p(p_1), g^p(p_2), \ldots, g^p(p_n)\} \tag{3.26}$$

Applies function g^q to each element in set Q.

$$g^q(Q) = \{g^q(q_1), g^q(q_2), \ldots, g^q(q_n)\} \tag{3.27}$$

Applies function g to each element in set P.

$$g^{pq}(P,Q) = \{g^{pq}(p_1,q_1), g^{pq}(p_2,q_2), \ldots, g^{pq}(p_n,q_n)\} \tag{3.28}$$

Step 5: Each function g_i^p is applied to set P, resulting in a list of function applications.

$$[g_1^p, g_2^p, \ldots, g_n^p](P) = [g_1^p(P), g_2^p(P), \ldots, g_n^p(P)] \tag{3.29}$$

Each function g_i^q is applied to set Q, resulting in a list of function applications.

$$[g_1^q, g_2^q, \ldots, g_n^q](Q) = [g_1^q(Q), g_2^q(Q), \ldots, g_n^q(Q)] \tag{3.30}$$

Each function g_i^{pq} is applied to set P, Q, resulting in a list of function applications.

$$[g_1^{pq}, g_2^{pq}, \ldots, g_n^{pq}](P,Q) = [g_1^{pq}(P,Q), g_2^{pq}(P,Q), \ldots, g_n^{pq}(P,Q)] \tag{3.31}$$

Step 6: Applies function h on integrals to get UAD_PF.

$$\begin{aligned}
&UAD\_PF(S,T,f,m,P,[[g_1^p,g_2^p,\ldots,g_n^p],[g_1^q,g_2^q,\ldots,g_n^q],[g_1^{pq},g_2^{pq},\ldots,g_n^{pq}]],h)\\
&= h([[\sum g_1^p(P),\sum g_2^p(P),\ldots,\sum g_n^p(P)],\\
&[\sum g_1^q(P),\sum g_2^q(P),\ldots,\sum g_n^q(P)],\\
&[\sum g_1^{pq}(P),\sum g_2^{pq}(P),\ldots,\sum g_n^{pq}(P)]])
\end{aligned} \tag{3.32}$$

d *Definition of UAM(Unified Algebraic Measure)*

According to the configuration of the different combinations of parameters provided under the unified framework, UAPs and UADs are the outputs. The unified framework of UAMs is as follows.

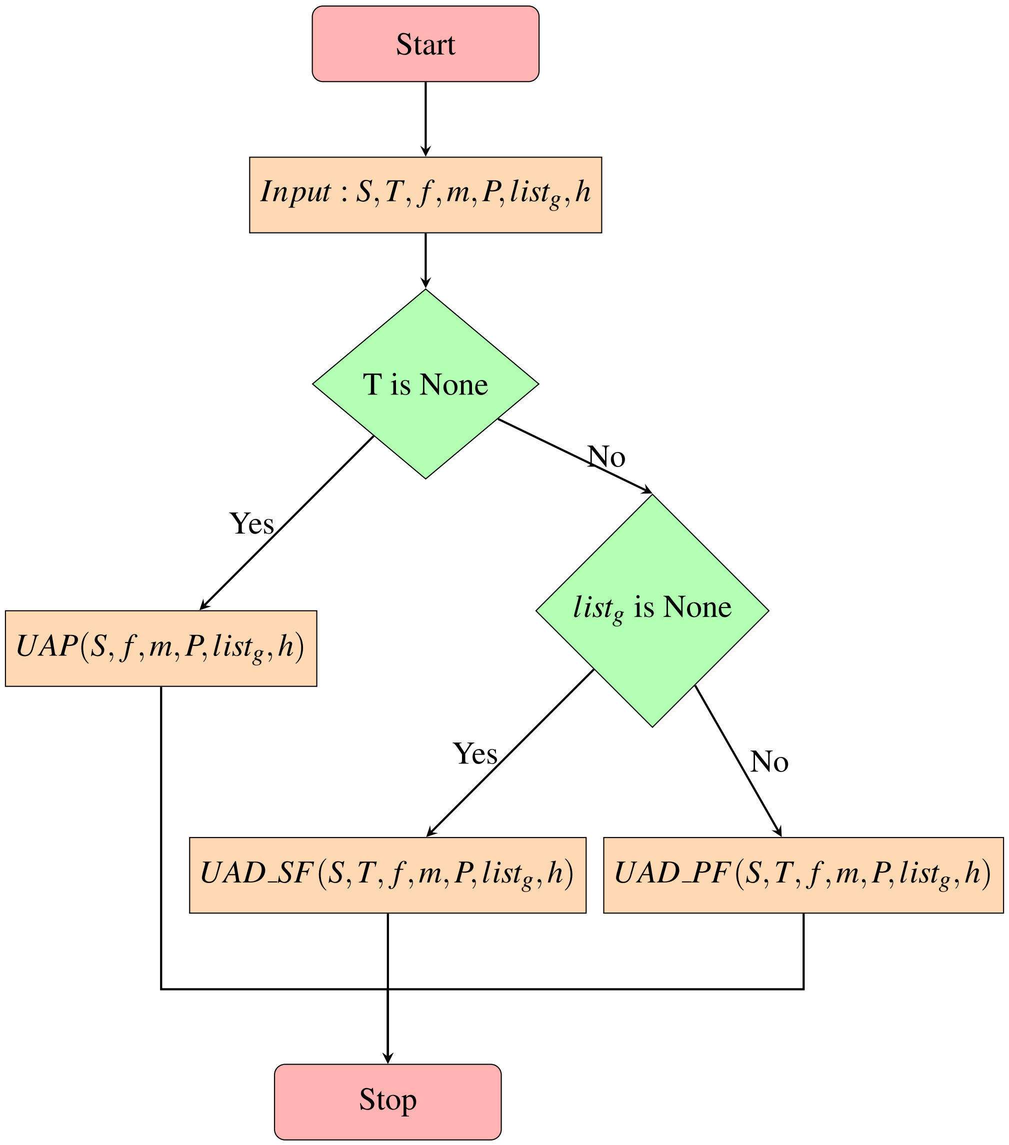

Start
$Input : S, T, f, m, P, list_g, h$
T is None
Yes
No
$UAP(S, f, m, P, list_g, h)$
$list_g$ is None
Yes
No
$UAD\_SF(S, T, f, m, P, list_g, h)$
$UAD\_PF(S, T, f, m, P, list_g, h)$
Stop

Figure 3.1

$\mathrm{UAM(S, T, f, m, P, list_g, h)}$ of the Unified Algebraic Framework

D UAM Defintions Under Different Situations

Since we defined a unified algebraic framework, now we redefine the existing measures in the unified framework and make some extensions. Also, we will define a unified algebraic measure in this section. Since many papers have fully made explanations about the 4 basic aspects: Non-negativity, the identity of indiscernibles, symmetry, and triangle inequality, we will not illustrate more about them. We will only focus on whether the unified algebraic measure is related to the order of elements in a sample, the order of samples, and the order of discrete probabilities. Before we delve into it, we define these three order-related concepts.

a *Sample Element Order Related*

If we change the element order in a sample, it influences the Unified Algebraic Measure, we can see this UAM is sample element order related. Usually, it takes effect on the final result by affecting the metric function. For example, we don't need to consider the order for a scalar sample. But for a vector or matrix sample,

The original sample vector $\mathbf{s_1}$ is defined as:

$$\mathbf{s_1} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}$$

If we change the order of the elements as:

$$\mathbf{s_1} = \begin{bmatrix} x_4 & x_3 & x_2 & x_1 \end{bmatrix}$$

if we use p-norm as the metric because the mathematical formula of p-norm is as:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}$$

The order of the elements in the vector will not influence the value of the Unified Algebraic

Measure based on the p-norm. So we call this UAM is not Sample Element Order Related.

But if we define a metric as a linear combination as:

$$y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4$$

for a specific weight vector as:

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}$$

The order of the elements in the vector will influence the value of the Unified Algebraic Measure based on this metric. So we call this UAM Sample Element Order Related.

Similiarlly, if we have a sample matrix $\mathbf{s_1}$ is defined as:

$$\mathbf{s_1} = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}$$

We can do the same thing. Whether the UAM is Sample Element Order Related or not is decided by the definition of the metric.

b ***Sample Order Related***

If we change the sample order in a set of samples, it influences the Unified Algebraic Measure, we can see this UAM is Sample Order Related. Usually, the order of the samples does not affect the Unified Algebraic Measures.

c *Discrete Probability Order Related*

If we change the probability order in a discrete distribution, it influences the Unified Algebraic Measure, we can see this UAM is Discrete Probability Order Related.

For example, the discrete vector **P** is defined as:

$$\mathbf{P} = \left[p_1, p_2, \ldots, p_n \right]$$

and the discrete vector **Q** is defined as:

$$\mathbf{Q} = \left[q_1, q_2, \ldots, q_n \right]$$

if we change the order of **P** to:

$$\tilde{\mathbf{P}} = \left[p_2, p_1, p_3, \ldots, p_n \right]$$

And the values of p_1 and p_2 are different, because the Shannon Entropy H of a discrete random variable X is defined as:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_b P(x_i)$$

So,

$$H(P) = H(\tilde{P})$$

Then we see the Shannon Entropy is not Discrete Probability Order Related.

But, the cross-entropy $H(P,Q)$ between two probability distributions P and Q is defined as:

$$H(P,Q) = -\sum_i P(x_i)\log Q(x_i)$$

Because,

$$H(P,Q) \neq H(\tilde{P},Q)$$

Then we see the Cross-Entropy is Discrete Probability Order Related.

Because the definition of Unified Algebraic Measure in 6-Set Algebra is highly configurable, The implementation of the Unified Algebraic Measure in the Unified Algebraic Framework is extremely extensible and configurable. We can use a unified form the redefine all existing measures or make extensions. The unified form to define is as follows.

Table 3.1

Configurable Template of UAM

Unified Algebraic Measure:Shannon Entropy	
Configurable Data Set and Function	Set or Formula
S	Outputs of a neuron across all source samples
T	Outputs of the same neuron across all source samples
f	Sample function f(*s*)
m	Metric function m(*s*)
P	Discrete probability of *S*
Q	Discrete probability of *T*
g	Probability function g(p),g(q), or g(p,q)
h	Function of integral of *g*
Options	Options for Special Case
UAM Type	UAP, E_UAD, UAD_SF, or UAD_PF
Sample Element Order Related	Yes or No
Sample Order Related	Yes or No
Discrete Probability Order Related	Yes or No

We use this form to define all the following measures in this unified form. All these UAMs begin from the outputs from neurons, but each UAM can have two versions, one with outputs as inputs, and another one with discrete probability distribution as inputs. Here we only provide the definitions of the first version.

E Single Neuron

Assuming the output of one single neuron is a tensor, the shape of the tensor can be a scalar([]), a vector([number_of_elements]), a matrix([number_of_elements_in_a_column, number_of_elements_in_a_row]). The sample function and metric function work on the tensor.

a *Entropy based on Metric*

In this category, we can select Shannon, Rényi, and Tsallis entropy type and metric function type as:

i *Shannon Entropy*

Shannon entropy is the most widely used measure of information entropy. It quantifies the expected value of the information contained in a message. The formula for Shannon entropy $H(X)$ for a discrete random variable X with possible outcomes $x_1, x_2, \ldots, x_n$ and corresponding probabilities $p(x_1), p(x_2), \ldots, p(x_n)$ is:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$

Table 3.2

UAM: Shannon Entropy based on p-Norm Metric

Unified Algebraic Measure:Shannon Entropy on p-Norm Metric	
Configurable Data Set and Function	Set or Formula
S	source
T	None
f	f(x) = x
m	$\|\mathbf{x}\|_p = (\sum_{i=1}^{n} \|x_i\|^p)^{\frac{1}{p}}$
P	None(Get it from S if S is not None)
Q	None(Get it from T if T is not None)
g	$g(x) = P(x)\log_b P(x)$
h	h(x) = -x
Options	$\{'b' : 2\}$
UAM Type	$UAP(S, f, m, P, [[g]], h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	No

ii *Rényi Entropy*

Rényi entropy is a family of entropy measures that generalizes Shannon entropy. It introduces a parameter α that allows for a different weighting of probabilities. The formula for Rényi entropy $H_\alpha(X)$ is:

$$H_\alpha(X) = \frac{1}{1-\alpha}\log_b\left(\sum_{i=1}^{n} p(x_i)^\alpha\right)$$

Table 3.3

UAM: Rényi Entropy based on p-Norm Metric

Unified Algebraic Measure: Rényi Entropy on p-Norm Metric	
Configurable Data Set and Function	Set or Formula
S	source
T	None
f	f(x) = x
m	$\|\mathbf{x}\|_p = (\sum_{i=1}^{n} \|x_i\|^p)^{\frac{1}{p}}$
P	None(Get it from S if S is not None)
Q	None(Get it from T if T is not None)
g	$g(x) = P(x)^\alpha$
h	$h(x) = log_b(x)/(1-\alpha)$
Options	$\{'\alpha' : 0.8, 'b' : 2\}$
UAM Type	$UAP(S, f, m, P, [[g]], h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	No

iii *Tsallis Entropy*

Tsallis entropy is another generalization of the Shannon entropy, introduced by Constantino Tsallis in 1988. The formula for Tsallis entropy is:

$$S_q(X) = \frac{1}{q-1}\left(1 - \sum_{i=1}^{n} p(x_i)^q\right)$$

Table 3.4

UAM: Tsallis Entropy based on p-Norm Metric

Unified Algebraic Measure: Tsallis Entropy on p-Norm Metric	
Configurable Data Set and Function	Set or Formula
S	source
T	None
f	f(x) = x
m	$\|\mathbf{x}\|_p = (\sum_{i=1}^n \|x_i\|^p)^{\frac{1}{p}}$
P	None(Get it from S if S is not None)
Q	None(Get it from T if T is not None)
g	$g(x) = P(x)^q$
h	$h(x) = (1-x)/(q-1)$
Options	$\{'q' : 2\}$
UAM Type	$UAP(S, f, m, P, [[g]], h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	No

b ***Entropy based on Sample Function***

We can choose any sample functions here. Sample function can work on the outputs of a neuron to get new outputs, we calculate the UAD of just one side of the neuron.

The unified configurable definition is as follows:

Table 3.5

UAM: Shannon Entropy based on Any Sample Function

Unified Algebraic Measure: Shannon Entropy based on any Sample Function	
Configurable Data Set and Function	Set or Formula
S	source
T	None
f	Any f(x)
m	$m(x) = \|\mathbf{x}\|_p = (\sum_{i=1}^{n} \|x_i\|^p)^{\frac{1}{p}}$
P	None(Get it from S if S is not None)
Q	None(Get it from T if T is not None)
g	$g(x) = P(x) \log P(x)$
h	$h(x) = -x$
Options	None
UAM Type	$UAP(S, f, m, P, [[g]], h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	No

F Two Neurons

No matter which one we select from the two neurons from one model or two models, we can guarantee the models come across the same sample set and, that the source output and target output have the corresponding relationships. So we can calculate Euclidean distance here.

a *Euclidean Distance*

Assuming points s_i in S and t_i in T are m elements vector or matrix, and also sum up the Euclidean distance between the corresponding points.

The unified configurable definition is as follows:

Table 3.6

UAM: Euclidean Distance

Unified Algebraic Measure: Euclidean Distance	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	$f(S,T) = \sum_{i=1}^{n} d_i = \sum_{i=1}^{n} \sqrt{\sum_{j=1}^{m}(s_{ij} - t_{ij})^2}$
m	None
P	None
Q	None
g	None
h	None
Options	None
UAM Type	$E\_UAD(S,T,f,m,P,[[g]],h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	No

b ***Entropy based on Euclidean Distance***

We calculate the Euclidean distance between corresponding points across all samples to form a set of distances, the Shannon Entropy is based on this distance set.

The unified configurable definition is as follows:

Table 3.7

UAM: Shannon Entropy based on Euclidean

Unified Algebraic Measure: Shannon Entropy based on Euclidean Distance	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	$f(s_i,t_i) = \sqrt{\sum_{j=1}^{m}(s_{ij}-t_{ij})^2}$
m	None
P	None(Get it from f(S,T) if S and T are not None)
Q	None
g	$g(x) = P(x)\log P(x)$
h	$h(x) = -x$
Options	None
UAM Type	$UAD\_SF(S,T,f,m,P,[[g]],h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	No

c *Euclidean Distance Extended from Loss Function*

The loss function can also be used as Euclidean distance, we select the MSE(Mean Squared Error) loss function here. We can also select any other loss functions.

The unified configurable definition is as follows:

Table 3.8

UAM: Euclidean Distance Extended from MSE

Unified Algebraic Measure: Euclidean Distance Extended from MSE	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	$f(S,T) = \text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(s_i - t_i)^2$
m	None
P	None
Q	None
g	None
h	None
Options	None
UAM Type	$E\_UAD(S,T,f,m,P,[[g]],h)$
Sample Element Order Related	No
Sample Order Related	Yes
Discrete Probability Order Related	No

G Across Different Sample Sets

No matter which one we select from One Neuron with Different Sample Sets or Two Neurons from Across-Model with Different Sample Sets, the source output and target output do not have the corresponding relationships. So we can not calculate Euclidean distance here. we can only calculate the function of the integral over the probabilities.

The unified configurable definition is as follows:

a ***Measures Extended from Existed***

We use the Unified Algebraic Measure to implement the existing measures here.

i *IPM(Integral Probability Metrics)*

The Integral Probability Metric (IPM) between two probability measures P and Q on a measurable space $\mathscr{X}$, given a function class $\mathscr{F}$, is defined as:

$$\gamma_{\mathscr{F}}(P,Q) = \sup_{f\in\mathscr{F}} |\mathbb{E}_P[f(X)] - \mathbb{E}_Q[f(Y)]|$$

where $\mathbb{E}_P[f(X)]$ denotes the expectation of f with respect to the probability measure P, and $\mathbb{E}_Q[f(Y)]$ denotes the expectation of f with respect to the probability measure Q.

Table 3.9

UAM: IPM

Unified Algebraic Measure: IPM	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	A class of functions $[f_1(x), f_2, \ldots, f_n]$
m	None
P	None(Get it from S if S is not None)
Q	None(Get it from T if T is not None)
g	$[[g^p(x) = f(x)P(x), g^q(x) = f(x)Q(x)]]$
h	$h(x,y) = min_{f \in \mathscr{F}}(x-y)$
Options	None
UAM Type	$UAD\_PF(S,T,f,m,P,[[g^p,g^q]],h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	Yes

ii *Common Function or Activation Function*

This UAD is calculated after the application of the sample function on the source and target.

The unified configurable definition is as follows:

Table 3.10

UAM: UAD based on sample function

Unified Algebraic Measure: UAD based on sample function	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	Any f(x)
m	$\|\mathbf{x}\|_p = (\sum_{i=1}^n \|x_i\|^p)^{\frac{1}{p}}$
P	None(Get it from $f(S)$ if S is not None)
Q	None(Get it from $f(T)$ if T is not None)
g	$g(x)$
h	$h(x)$
Options	None
UAM Type	$UAD\_PF(S,T,f,m,P,[[g]],h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	No

iii *Total Variation Distance*

The TVD(Total Variation Distance) between two probability distributions P and Q over a sample space Ω is defined as:

$$\text{TVD}(P,Q) = \frac{1}{2}\sum_{x\in\Omega}|P(x)-Q(x)|$$

The unified configurable definition is as follows:

Table 3.11

UAM: TVD(Total Variation Distance)

Unified Algebraic Measure:TVD	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	None
m	$\|\mathbf{x}\|_p = (\sum_{i=1}^{n}\|x_i\|^p)^{\frac{1}{p}}$
P	None(Get it from S if S is not None)
Q	None(Get it from T if T is not None)
g	$g^{pq}(x) = abs(P(x)-Q(x))$
h	$h(x) = x$
Options	None
UAM Type	$UAD\_PF(S,T,f,m,P,[[g^{qp}]],h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	Yes

iv *Entropy Difference*

The absolute difference between the entropies of two probability distributions P and Q over a sample space Ω can be expressed as:

$$|\mathrm{H}(P) - \mathrm{H}(Q)|$$

where $\mathrm{H}(P)$ and $\mathrm{H}(Q)$ are the entropies of the distributions P and Q, respectively. The entropy $\mathrm{H}(P)$ of a distribution P is defined as:

For discrete distributions:

$$\mathrm{H}(P) = -\sum_{x\in\Omega} P(x)\log P(x)$$

For continuous distributions:

$$\mathrm{H}(P) = -\int_{\Omega} P(x)\log P(x)\,dx$$

Thus, the absolute difference between the entropies of P and Q is:

For discrete distributions:

$$|\mathrm{H}(P) - \mathrm{H}(Q)| = \left| -\sum_{x\in\Omega} P(x)\log P(x) + \sum_{x\in\Omega} Q(x)\log Q(x) \right|$$

The unified configurable definition is as follows:

Table 3.12

UAM: Absolute Entropy Difference

Unified Algebraic Measure: Absolute Entropy Difference	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	Any f(x)
m	$\|\mathbf{x}\|_p = (\sum_{i=1}^{n} \|x_i\|^p)^{\frac{1}{p}}$
P	None(Get it from S if S is not None)
Q	None(Get it from T if T is not None)
g	$g^p(x) = P(x)\log P(x), g^q(x) = Q(x)\log Q(x)$
h	$h(x,y) = abs(x-y)$
Options	None
UAM Type	$UA\_PF(S,T,f,m,P,[[g^p][g^q]],h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	Yes

v *Cross-Entropy*

The cross-entropy between two probability distributions P and Q over a sample space Ω is defined as:

For discrete distributions:

$$H(P,Q) = -\sum_{x\in\Omega} P(x)\log Q(x)$$

The unified configurable definition is as follows:

Table 3.13

UAM: Cross-Entropy

Unified Algebraic Measure: Cross-Entropy	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	Any f(x)
m	$\|\mathbf{x}\|_p = (\sum_{i=1}^n \|x_i\|^p)^{\frac{1}{p}}$
P	None(Get it from S if S is not None)
Q	None(Get it from T if T is not None)
g	$g^{pq}(x) = P(x)\log Q(x)$
h	$h(x) = -x$
Options	None
UAM Type	$UAP(S,T,f,m,P,[[g^pq]],h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	Yes

vi *Divergence*

Here are the mathematical formulas and definitions in the unified form of a list of divergences.

1 KL(Kullback-Leibler) divergence

The Kullback-Leibler (KL) divergence from distribution Q to distribution P is defined as:

For discrete distributions:

$$D_{KL}(P \parallel Q) = \sum_{x \in \Omega} P(x) \log \frac{P(x)}{Q(x)}$$

The unified configurable definition is as follows:

Table 3.14

UAM: KL Divergence

Unified Algebraic Measure: KL Divergence	
Configurable Data Set and Function	Set or Formula
S	source
T	None
f	Any f(x)
m	$\|\mathbf{x}\|_p = (\sum_{i=1}^{n} \|x_i\|^p)^{\frac{1}{p}}$
P	None(Get it from S if S is not None)
Q	None(Get it from T if T is not None)
g	$g^{pq}(x) = P(x) \log \frac{P(x)}{Q(x)}$
h	$h(x) = x$
Options	None
UAM Type	$UAD\_PF(S,T,f,m,P,[[g^{pq}]],h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	No

2 Alpha-Divergence

The α-divergence is a family of divergence measures that generalizes many well-known divergences by using a parameter α. For two probability distributions P and Q over a sample space Ω, the α-divergence is defined as:

For discrete distributions:

$$D_\alpha(P \parallel Q) = \frac{1}{\alpha(\alpha-1)} \left(1 - \sum_{x\in\Omega} P(x)^\alpha Q(x)^{1-\alpha}\right)$$

The unified configurable definition is as follows:

Table 3.15

UAM: Alpha-Divergence

Unified Algebraic Measure: Alpha-Divergence	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	Any f(x)
m	$\|\mathbf{x}\|_p = (\sum_{i=1}^n \|x_i\|^p)^{\frac{1}{p}}$
P	None(Get it from S if S is not None)
Q	None(Get it from T if T is not None)
g	$g^{pq}(x) = P(x)^\alpha Q(x)^{1-\alpha}$
h	$h(x) = \frac{1}{\alpha(\alpha-1)}(1-x)$
Options	$\{'\alpha' : 0.8\}$
UAM Type	$UAD\_PF(S,T,f,m,P,[[g^{pq}]],h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	Yes

3 Beta-Divergence

The β-divergence is a family of divergence measures that generalizes several known divergence measures. For two probability distributions P and Q over a sample space Ω, the β-divergence is defined as follows:

For $\beta \neq 0, 1$:

For discrete distributions:

$$D_\beta(P \parallel Q) = \sum_{x \in \Omega} \left(\frac{P(x)^\beta - \beta P(x) Q(x)^{\beta-1} + (\beta - 1) Q(x)^\beta}{\beta(\beta - 1)} \right)$$

The unified configurable definition is as follows:

Table 3.16

UAM: Beta-Divergence

Unified Algebraic Measure: Beta-Divergence	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	Any f(x)
m	$\|\mathbf{x}\|_p = (\sum_{i=1}^{n} \lvert x_i \rvert^p)^{\frac{1}{p}}$
P	None(Get it from S if S is not None)
Q	None(Get it from T if T is not None)
g	$g^p(x) = P(x)^\beta, g^q(x) = Q(x)^\beta, g^{pq}(x) = P(x)Q(x)^{\beta-1}$
h	$h(x,y,z) = \frac{1}{\beta(\beta-1)}(x - \beta y - (\beta - 1)z)$
Options	$\{'\beta' : 0.8\}$
UAM Type	$UAD\_PF(S,T,f,m,P,[[g^p][g^q][g^{pq}]],h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	Yes

4 Gamma-Divergence

The γ-divergence is a family of divergence measures used to compare probability distributions. For two probability distributions P and Q over a sample space Ω, the γ-divergence is defined as follows:

For $\gamma \neq 1$:

For discrete distributions:

$$D_\gamma(P \parallel Q) = \frac{1}{\gamma(\gamma-1)} \left(\sum_{x \in \Omega} P(x)^\gamma Q(x)^{1-\gamma} - 1 \right)$$

The unified configurable definition is as follows:

Table 3.17

UAM: Gamma-Divergence

Unified Algebraic Measure: Gamma-Divergence	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	Any f(x)
m	$\|\mathbf{x}\|_p = (\sum_{i=1}^{n} \|x_i\|^p)^{\frac{1}{p}}$
P	None(Get it from S if S is not None)
Q	None(Get it from T if T is not None)
g	$g^{pq}(x) = P(x)^{\gamma} Q(x)^{1-\gamma}$
h	$h(x) = \frac{1}{\gamma(\gamma-1)}(x-1)$
Options	$\{'\gamma' : 0.8\}$
UAM Type	$UAD\_PF(S,T,f,m,P,[[g^{pq}]],h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	Yes

5 f-Divergence (or General H-Divergence)

The f-divergence is a class of functions used to measure the difference between two probability distributions P and Q over a sample space Ω. It is defined using a convex function f.

For discrete distributions:

$$D_f(P \parallel Q) = \sum_{x \in \Omega} Q(x) f\left(\frac{P(x)}{Q(x)}\right)$$

The unified configurable definition is as follows:

Table 3.18

UAM: $f(H)$-Divergence

Unified Algebraic Measure: $f(H)$-Divergence	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	Any f(x)
m	$\lVert\mathbf{x}\rVert_p = (\sum_{i=1}^n \lvert x_i\rvert^p)^{\frac{1}{p}}$
P	None(Get it from S if S is not None)
Q	None(Get it from T if T is not None)
g	$g^{pq}(x) = Q(x) f_H(\frac{P(x)}{Q(x)})$
h	$h(x) = x$
Options	$\{'FH\_Function' : f_H(x)\}$
UAM Type	$UAD\_PF(S, T, f, m, P, [[g^{pq}]], h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	Yes

6 *H*-divergence (Hellinger Distance)

The H-divergence (Hellinger distance) between two probability distributions P and Q over a sample space Ω is defined as:

For discrete distributions:

$$H^2(P,Q) = \frac{1}{2} \sum_{x \in \Omega} \left(\sqrt{P(x)} - \sqrt{Q(x)} \right)^2$$

The unified configurable definition is as follows:

Table 3.19

UAM: H-Divergence

Unified Algebraic Measure: H-Divergence	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	Any f(x)
m	$\|\mathbf{x}\|_p = (\sum_{i=1}^{n} \lvert x_i \rvert^p)^{\frac{1}{p}}$
P	None(Get it from S if S is not None)
Q	None(Get it from T if T is not None)
g	$g^{pq}(x) = (P(x) - Q(x))^2$
h	$h(x) = \sqrt{\frac{1}{2}x}$
Options	None
UAM Type	$UAD\_PF(S, T, f, m, P, [[g^{pq}]], h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	Yes

7 Jensen-Shannon Divergence

The Jensen-Shannon Divergence (JSD) between two probability distributions P and Q over a sample space Ω is defined as follows:

1. Calculate the average distribution M:

$$M = \frac{1}{2}(P + Q)$$

2. Compute the Jensen-Shannon Divergence:

$$D_{JS}(P \parallel Q) = \frac{1}{2} D_{KL}(P \parallel M) + \frac{1}{2} D_{KL}(Q \parallel M)$$

Where $D_{KL}(P \parallel Q)$ is the Kullback-Leibler divergence between P and Q:

For discrete distributions:

$$D_{KL}(P \parallel Q) = \sum_{x \in \Omega} P(x) \log \frac{P(x)}{Q(x)}$$

Thus, the formula for Jensen-Shannon Divergence becomes:

For discrete distributions:

$$D_{JS}(P \parallel Q) = \frac{1}{2} \sum_{x \in \Omega} P(x) \log \frac{P(x)}{M(x)} + \frac{1}{2} \sum_{x \in \Omega} Q(x) \log \frac{Q(x)}{M(x)}$$

For this unified form definition, we define as follows:

$g_1^{pq}(x) = P(x)log(\frac{P(x)}{\frac{1}{2}(P(x)+Q(x))})$

$g_2^{pq}(x) = Q(x)log(\frac{Q(x)}{\frac{1}{2}(P(x)+Q(x))})$

These two functions will be used in the form.

The unified configurable definition is as follows:

Table 3.20

UAM: Jensen-Shannon Divergence

Unified Algebraic Measure: Jensen-Shannon Divergence	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	Any f(x)
m	$\|\mathbf{x}\|_p = (\sum_{i=1}^{n} \|x_i\|^p)^{\frac{1}{p}}$
P	None(Get it from S if S is not None)
Q	None(Get it from T if T is not None)
g	$g_1^{pq}(x), g_2^{pq}(x)$
h	$h(x,y) = \frac{1}{2}(x+y)$
Options	None
UAM Type	$UAD\_PF(S,T,f,m,P,[[g_1^{pq},g_2^{pq}]],h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	Yes

8 χ^2-divergence

The χ^2-divergence is a specific type of divergence used to compare two probability distributions. The χ^2-divergence between two probability distributions P and Q over a sample space Ω is defined as follows:

For discrete distributions:

$$D_{\chi^2}(P \parallel Q) = \sum_{x \in \Omega} \frac{(P(x) - Q(x))^2}{Q(x)}$$

The unified configurable definition is as follows:

Table 3.21

UAM: χ^2-Divergence

Unified Algebraic Measure: χ^2-Divergence	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	Any f(x)
m	$\|\mathbf{x}\|_p = (\sum_{i=1}^{n} \|x_i\|^p)^{\frac{1}{p}}$
P	None(Get it from S if S is not None)
Q	None(Get it from T if T is not None)
g	$g^{pq}(x) = \frac{(P(x)-Q(x))^2}{Q(x)}$
h	$h(x) = x$
Options	None
UAM Type	$UAD\_PF(S,T,f,m,P,[[g^{pq}]],h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	Yes

vii *MMD(Maximum Mean Discrepancy)*

We use Unbiased Empirical Estimates here.

Given samples $\{x_i\}_{i=1}^m$ from P and $\{y_j\}_{j=1}^n$ from Q, the unbiased empirical estimate of the squared MMD is:

$$\widehat{\mathrm{MMD}}^2(P,Q;k) = \frac{1}{m(m-1)}\sum_{i\neq i'} k(x_i,x_{i'}) + \frac{1}{n(n-1)}\sum_{j\neq j'} k(y_j,y_{j'}) - \frac{2}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} k(x_i,y_j)$$

The unified configurable definition is as follows:

Table 3.22

UAM: MMD

Unified Algebraic Measure: MMD	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	$f(S,T,k) = \widehat{\text{MMD}}^2(P,Q;k)$
m	
P	None
Q	None
g	None
h	None
Options	$\{'kernel' : rbf\_kernel\}$
UAM Type	$E\_UAD(S,T,f,m,P,[g],h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	No

viii *Wasserstein Distance*

The 1-Wasserstein distance (also known as the Earth Mover's Distance) between two probability distributions can be formulated as an optimization problem in the context of optimal transport theory.

Mathematical Formulation

Given two probability distributions P and Q over a metric space Ω with a metric d, the 1-Wasserstein distance $W_1(P,Q)$ is defined as:

$$W_1(P,Q) = \inf_{\gamma \in \Gamma(P,Q)} \int_{\Omega \times \Omega} d(x,y)\, d\gamma(x,y)$$

where:

- $\Gamma(P,Q)$ is the set of all couplings of P and Q.
- γ is a joint distribution over $\Omega \times \Omega$ whose marginals are P and Q.

Discrete Case

For discrete distributions P and Q:

$$P = \sum_{i=1}^{n} p_i \delta_{x_i}$$

$$Q = \sum_{j=1}^{m} q_j \delta_{y_j}$$

where δ_{x_i} and δ_{y_j} are Dirac delta functions centered at x_i and y_j, respectively, and p_i and q_j are the probabilities associated with these points.

The 1-Wasserstein distance can be formulated as a linear programming problem:

$$W_1(P,Q) = \min_{\gamma_{ij}} \sum_{i=1}^{n} \sum_{j=1}^{m} \gamma_{ij} d(x_i, y_j)$$

subject to the constraints:

Marginal Constraints

$$\sum_{j=1}^{m} \gamma_{ij} = p_i \quad \text{for all } i$$

$$\sum_{i=1}^{n} \gamma_{ij} = q_j \quad \text{for all } j$$

Non-negativity Constraint

$$\gamma_{ij} \geq 0 \quad \text{for all } i, j$$

Here, γ_{ij} represents the amount of ”mass” transported from x_i to y_j, and $d(x_i, y_j)$ is the cost associated with transporting a unit of mass from x_i to y_j.

The unified configurable definition is as follows, and Wasserstein distance is a special case, the unified form calls the function directly to calculate it.

Table 3.23

UAM: Wasserstein Distance

Unified Algebraic Measure: Wasserstein Distance	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	$f(S,T) = wasserstein\_distance(P,Q,cost\_matrix)$
m	$\|\mathbf{x}\|_p = (\sum_{i=1}^{n} \|x_i\|^p)^{\frac{1}{p}}$
P	None(Get it from S if S is not None)
Q	None(Get it from T if T is not None)
g	None
h	None
Options	None
UAM Type	$E\_UAD(S,T,f,m,P,[g],h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	Yes

b ***Customized Components and Customized Measures***

We can not only customize the kernel integral function by introducing the order change of source discrete probability and target discrete probability as follows. but also, we use the composite method to form new measures.

i *Reverse Integral*

To define reverse integral, we express the sum where i ranges from 1 to n and j ranges from n to 1, we need to adjust the index for j.

Let i range from 1 to n and set $j = n+1-i$, which will make j decrease as i increases. Thus, when $i = 1$, $j = n$, and when $i = n$, $j = 1$.

We can write the double sum as follows:

$$\sum_{i=1}^{n} f(P_i) f(Q_{n+1-i})$$

This notation ensures that as i increases from 1 to n, j decreases from n to 1.

ii *Random Order Integral*

To define random order integral, we introduce a permutation of the indices. Let σ be a permutation of $\{1,2,\ldots,n\}$. This permutation function σ can be used to randomly reorder the indices j.

Mathematical Expression

Given a permutation σ of $\{1,2,\ldots,n\}$, the sum can be expressed as:

$$\sum_{i=1}^{n} f(P_i) f(Q_{\sigma(i)})$$

Here, $\sigma(i)$ gives the new index for j when i is fixed.

iii *Inner-Sample Order Matters*

To illustrate how selecting a special weight matrix makes the order of the elements in the sample matrix matter, we can draw an analogy to the binary or decimal number system. In these number systems, the position of each digit determines its weight (e.g., in the decimal system, the rightmost digit represents units, the next digit represents tens, and so on).

Similarly, we can define a weight matrix in such a way that the position of each element in the sample matrix determines its contribution to the overall result, analogous to how the position of digits in a number determines the number's value.

Consider an $n \times n$ sample matrix S and an $n \times n$ weight matrix W. We define a sample function as the element-wise multiplication of these two matrices:

$$M = S \circ W$$

where $\circ$ denotes element-wise multiplication.

1 Binary System Analogy

In a binary system, each digit represents a power of 2. Similarly, we can define a weight matrix W where each element w_{ij} represents a power of 2 based on its position:

$$W_{ij} = 2^{(i-1)n+(j-1)}$$

This weight matrix gives a unique binary-like weight to each element of the sample matrix S, making the position of each element matter significantly.

2 Decimal System Analogy

In a decimal system, each digit represents a power of 10. Similarly, we can define a weight matrix W where each element w_{ij} represents a power of 10 based on its position:

$$W_{ij} = 10^{(i-1)n+(j-1)}$$

This weight matrix gives a unique decimal-like weight to each element of the sample matrix

S, ensuring that the position of each element affects the overall result.

3 Mathematical Illustration

For an $n \times n$ sample matrix S and a weight matrix W defined as:

$$W_{ij} = b^{(i-1)n+(j-1)}$$

where b is the base (e.g., 2 for binary, 10 for decimal), the element-wise multiplication $M = S \circ W$ results in:

$$M_{ij} = S_{ij} \cdot W_{ij} = S_{ij} \cdot b^{(i-1)n+(j-1)}$$

We can select different bases, not limited to 2 and 10, to define different sample functions, and they define different customized entropies or divergences based on them.

iv *Entropy based on Element-wise Subtraction*

The unified configurable definition for Entropy based on Element-wise Subtraction is as follows.

Table 3.24

UAM: Shannon Entropy based on Element-wise Subtraction

Unified Algebraic Measure: Shannon Entropy based on Element-wise Subtraction	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	$f(s,t) = abs(s-t)$
m	$\|\mathbf{x}\|_p = (\sum_{i=1}^{n} \|x_i\|^p)^{\frac{1}{p}}$
P	None(Get it from $f(S,T)$ if S is not None)
Q	None
g	$g(x) = P(x)\log P(x)$
h	h(x) = -x
Options	None
UAM Type	$UAD\_SF(S,T,f,m,P,[g],h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	No

v *Entropy based on Element-wise Division*

The unified configurable definition for Entropy based on Element-wise Division is as follows.

Table 3.25

UAM: Shannon Entropy based on Element-wise Division

Unified Algebraic Measure: Shannon Entropy based on Element-wise Division	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	$f(s,t) = \lfloor \frac{s}{t} \rfloor$; $f(s,t) = 0\ if\ t = 0$
m	$\|\mathbf{x}\|_p = (\sum_{i=1}^{n} \lvert x_i \rvert^p)^{\frac{1}{p}}$
P	None(Get it from $f(S,T)$ if S is not None)
Q	None
g	$g(x) = P(x) \log P(x)$
h	h(x) = -x
Options	None
UAM Type	$UAD\_SF(S,T,f,m,P,[g],h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	No

vi *Entropy based on Element-wise Divison Remainder*

The unified configurable definition for Entropy based on Element-wise Divison Remainder is as follows.

Table 3.26

UAM: Shannon Entropy based on Element-wise Division Remainder

Unified Algebraic Measure: Shannon Entropy based on Element-wise Division Remainder	
Configurable Data Set and Function	Set or Formula
S	source
T	target
f	$f(s,t) = x \mod y; f(s,t) = 0 \; if \; t = 0$
m	$\|\mathbf{x}\|_p = (\sum_{i=1}^{n} \|x_i\|^p)^{\frac{1}{p}}$
P	None(Get it from $f(S,T)$ if S is not None)
Q	None
g	$g(x) = P(x)\log P(x)$
h	h(x) = -x
Options	None
UAM Type	$UAD\_SF(S,T,f,m,P,[g],h)$
Sample Element Order Related	No
Sample Order Related	No
Discrete Probability Order Related	No

vii *Unified Composite Entropy*

For a specific entropy, UCE(unified composite entropy) can be defined as a combination of different levels of entropy: point entropy, vector entropy, and matrix entropy. Each level corresponds to different dimensional structures within the data.

1 Point Entropy

For flattened outputs of a neuron with probability mass function $P(x)$:

$$\text{Entropy}_{\text{point}}(X) = H(X) = -\sum_{x \in \mathscr{X}} P(x) \log P(x)$$

2 Vector Entropy

For flattened p-Norm values of all vectors in the outputs of a neuron $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ with probability mass function $P(\mathbf{x})$:

$$\text{Entropy}_{\text{vector}}(\mathbf{X}) = H(\mathbf{X}) = -\sum_{\mathbf{x} \in \mathscr{X}} P(\mathbf{x}) \log P(\mathbf{x})$$

3 Matrix Entropy

For flattened p-Norm values of all matrices in the outputs of a neuron $\mathbf{X}$ with probability mass function $P(\mathbf{X})$:

$$\text{Entropy}_{\text{matrix}}(\mathbf{X}) = H(\mathbf{X}) = -\sum_{\mathbf{X} \in \mathscr{X}} P(\mathbf{X}) \log P(\mathbf{X})$$

4 Composite Entropy

Composite entropy is a combination of point entropy, vector entropy, and matrix entropy, which can be defined as:

$$\text{Composite Entropy} = \text{Entropy}_{\text{point}} + \text{Entropy}_{\text{vector}} + \text{Entropy}_{\text{matrix}}$$

Alternatively, a weighted sum can be used:

$$\text{Composite Entropy} = \alpha \cdot \text{Entropy}_{\text{point}} + \beta \cdot \text{Entropy}_{\text{vector}} + \gamma \cdot \text{Entropy}_{\text{matrix}}$$

where α, β, and γ are weights that reflect the relative importance of each component.

viii *Unified Composite Divergence*

For a specific divergence, UCD(unified composite divergence) can be defined as a combination of different levels of divergence: point divergence, vector divergence, and matrix divergence. Each level corresponds to different dimensional structures within the data.

1 Point Divergence

For flattened outputs of a neuron X with probability distributions P and Q:

$$\text{Divergence}_{\text{point}}(P \parallel Q) = D_{KL}(P \parallel Q) = \sum_{x \in \mathscr{X}} P(x) \log \frac{P(x)}{Q(x)}$$

2 Vector Divergence

For flattened p-Norm values of all vectors in the outputs of a neuron $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ with probability distributions P and Q:

$$\text{Divergence}_{\text{vector}}(P \parallel Q) = D_{KL}(P \parallel Q) = \sum_{\mathbf{x} \in \mathscr{X}} P(\mathbf{x}) \log \frac{P(\mathbf{x})}{Q(\mathbf{x})}$$

3 Matrix Divergence

For flattened p-Norm values of all matrices in the outputs of a neuron $\mathbf{X}$ with probability distributions P and Q:

$$\text{Divergence}_{\text{matrix}}(P \parallel Q) = D_{KL}(P \parallel Q) = \sum_{\mathbf{X} \in \mathscr{X}} P(\mathbf{X}) \log \frac{P(\mathbf{X})}{Q(\mathbf{X})}$$

4 Composite Divergence

Composite divergence is a combination of point divergence, vector divergence, and matrix divergence, which can be defined as:

$$\text{Composite Divergence} = \text{Divergence}_{\text{point}} + \text{Divergence}_{\text{vector}} + \text{Divergence}_{\text{matrix}}$$

Alternatively, a weighted sum can be used:

$$\text{Composite Divergence} = \alpha \cdot \text{Divergence}_{\text{point}} + \beta \cdot \text{Divergence}_{\text{vector}} + \gamma \cdot \text{Divergence}_{\text{matrix}}$$

where α, β, and γ are weights that reflect the relative importance of each component.

ix *Unified Composite Measure*

The UCM(unified composite measure) is a comprehensive metric that combines various measures to evaluate differences or similarities between probability distributions or data sets. This measure integrates Composite Entropy, Composite Divergence, Wasserstein Distance, MMD(maximum mean discrepancy), and Euclidean Distance. To ensure that each component contributes appropriately, normalization is applied to each measure. See Appendix C on page 352 which is an example to show how to form a KL Divergence context by introducing distribution reference. In this KL Divergence context, we can select the largest divergence for the normalization.

1 Normalization

Suppose $\mathscr{M}$ represents a general measure, then its normalized version $\mathscr{M}_{\text{norm}}$ can be defined as:

$$\mathscr{M}_{\text{norm}} = \frac{\mathscr{M}}{\mathscr{M}_{\max}}$$

where $\mathscr{M}_{\max}$ is the maximum possible value of $\mathscr{M}$, or an appropriate normalization constant.

2 UCM(unified composite measure)

The Unified Composite Measure combines the normalized versions of the above components:

$$\begin{aligned}\text{UCM} = {} & \lambda_1 \cdot \text{Composite Entropy}_{\text{norm}} + \\ & \lambda_2 \cdot \text{Composite Divergence}_{\text{norm}} + \\ & \lambda_3 \cdot W_1(P,Q)_{\text{norm}} + \\ & \lambda_4 \cdot \text{MMD}(P,Q;k)_{\text{norm}} + \\ & \lambda_5 \cdot \text{Euclidean Distance}_{\text{norm}} \end{aligned} \tag{3.33}$$

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5$ are weights that reflect the relative importance of each component.

3.5.2 Visualization

The Unified Algebraic Framework provides some simple visualization functions to show the Unified Algebraic Measure of all the neurons in a layer of a neural network.

3.6 Implementation

3.6.1 Algorithm Development

The kernel algorithm is the implementation of UAM(Unified Algebraic Measure). You can refer to the pseudocode of the Algorithm (3.1) and the Subalgorithms(3.2,3.4,3.3) at the end of this chapter for details. See Appendix A on page 242 for the implementation of the Unified Algebraic Framework in Python.

3.6.2 Computational Tools

The software and tools used to implement the algorithms are as follows.

- `numpy` - Required for numerical operations
- `torch` - PyTorch for neural network functionality
- `matplotlib` - For plotting and visualizations
- `scipy` - For Wasserstein distance optimization

3.7 Conclusion

3.7.1 Summary

This chapter mathematically defines the 6-Set Discrete Probability Algebra. It also defines the Unified Algebraic Framework extended from the 6-Set Algebra. The highly configurable unified form is applied to redefine all existing measures and define the extended measures. The implementation of the Unified Algebraic is introduced with the pseudocode algorithms. The Python Package is also provided.

3.8 Appendix of This Chapter: Four Main Algorithms in Detail

There are four main algorithms as follows.

3.8.1 Algorithm: Unified Algebraic Measure Calculation

Unified Algebraic Measure Calculation is the kernel algorithm as follows.

Algorithm 3.1 Unified Algebraic Measure Calculation

```
1: function UNIFIEDALGEBRAICMEASURE(source, target, metric, metricOption, sourcePD, targetPD, probabilityOption, sampleFunction, sampleFunctionOption, probabilityFunctions, probabilityFunctionOptions, integralFunction, integralFunctionOption)
2:     if source is provided and target is not provided then
3:         if sampleFunction is provided then
4:             source ← TRANSFORMTENSOR(source, sampleFunction, sampleFunctionOption)
5:         end if
6:         if metric is provided then
7:             source ← TRANSFORMTENSOR(ALGORITHM3.2)(source, metric, metricOption)
8:         end if
9:         sourcePD ← OUTPUTSTOPROBABILITY(ALGORITHM3.3)(source, probabilityOption)
10:        integralOutput ← INTEGRALFUNCONPROB(ALGORITHM3.4)(probabilityFunctions, probabilityFunctionOptions, sourcePD, targetPD)
11:        if integralFunction is provided then
12:            output ← INTEGRALFUNCTION(integralOutput, integralFunctionOption)
13:        end if
14:    else if source is provided and target is provided then
15:        if sampleFunctionOption type is ’Euclidean’ then
16:            PROCESSEUCLIDEAN(source, target, metric, metricOption, sampleFunction, sampleFunctionOption)
17:        else if sampleFunctionOption type is ’ProbabilityOnEuclidean’ then
18:            source ← TRANSFORMTWOTENSORS(source, target, sampleFunction, sampleFunctionOption)
19:            source ← TRANSFORMTENSOR(source, metric, metricOption)
20:            sourcePD ← OUTPUTSTOPROBABILITY(source, probabilityOption)
21:            integralOutput ← INTEGRALFUNCONPROB(probabilityFunctions, probabilityFunctionOptions, sourcePD, targetPD)
22:            output ← INTEGRALFUNCTION(integralOutput, integralFunctionOption)
23:        end if
24:    else if sourcePD is not None and targetPD is None then
25:        integralOutput ← INTEGRALFUNCONPROB(probabilityFunctions, probabilityFunctionOptions, sourcePD, targetPD)
26:        output ← INTEGRALFUNCTION(integralOutput, integralFunctionOption)
27:    end if
28:    return output
29: end function
```

A Detailed Description of Functionality

The pseudocode provided outlines the functionality of the `unifiedAlgebraicMeasure` method in a detailed and structured manner. It highlights the decision-making processes and various operations that depend on input types and available configurations. This description captures the core steps involved in manipulating and processing tensors or data based on the method's comprehensive parameter set.

- **Process Flow:** The method distinguishes between operations based solely on source data(Unified Algebraic Property), both source and target data(Unified Algebraic Distance) or directly on provided probability distributions.

- **Function Applications:** Depending on the configuration, the function applies sample functions, metrics, probability functions, and an integral function. It outlines how these applications change based on input types and available options.

- **Error Handling and Feedback:** The pseudocode suggests printing messages or handling errors when configurations are incorrect or inputs are missing, which could be implemented as actual error handling in a programming environment.

This approach ensures that the method's functionality is both versatile and robust, suitable for various types of data analysis where different stages of data transformation and calculation are required.

3.8.2 Algorithm: Transform Tensor

In this algorithm, a specific transform function is used to convert a tensor to another tensor.

Algorithm 3.2 Transform Tensor

```
1: procedure TRANSFORM_TENSOR
2:  inputs: $input\_tensor, transform\_function, transform\_function\_option$
3:     $output\_items \leftarrow$ new list
4:     if $transform\_function\_option =$ None then
5:         $transform\_function\_option \leftarrow$ new dictionary
6:     end if
7:     $input\_shape \leftarrow$ shape of $input\_tensor$
8:     if dimension of $input\_tensor = 1$ then
9:         for $i \leftarrow 0$ to $input\_shape[0] - 1$ do
10:            $output\_items \leftarrow$ new list
11:            $args \leftarrow (input\_tensor[i], transform\_function, transform\_function\_option)$
12:            $result \leftarrow$ apply_elementwise_tensor($args$)
13:            $output\_items$.append($result$)
14:        end for
15:    else if dimension of $input\_tensor = 2$ then
16:        for $i \leftarrow 0$ to $input\_shape[0] - 1$ do
17:            $output\_items \leftarrow$ new list
18:            $args \leftarrow (input\_tensor[i,:], transform\_function, transform\_function\_option)$
19:            $result \leftarrow$ apply_elementwise_tensor($args$)
20:            $output\_items$.append($result$)
21:        end for
22:    else if dimension of $input\_tensor = 3$ then
23:        for $i \leftarrow 0$ to $input\_shape[0] - 1$ do
24:            $output\_items \leftarrow$ new list
25:            $args \leftarrow (input\_tensor[i,:,:], transform\_function, transform\_function\_option)$
26:            $result \leftarrow$ apply_elementwise_tensor($args$)
27:            $output\_items$.append($result$)
28:        end for
29:    else
30:        raise Error: ”Unsupported tensor shape. Expected 1D, 2D, or 3D tensor.”
31:    end if
32:    $output\_tensor \leftarrow$ convert $output\_items$ to tensor
33:    print $output\_tensor$
34:    return $output\_tensor$
35: end procedure
```

A Key Points of the Pseudocode

Initialization: Start by defining the output structure and handling default parameters (e.g., empty options dictionary).

- **Dimension Handling:** Use conditional statements to differentiate actions based on tensor dimensions. This reflects the input_tensor's potential configurations (1D, 2D, 3D).

- **Looping and Transformation:** Iteratively apply the transformation function to each element of the tensor, adjusted for dimensionality. Each transformation outcome is collected into `output_items`.

- **Error Handling:** Include a condition to manage unsupported tensor dimensions, which throws an error if encountered.

- **Output Construction:** Convert the list of transformed items back into a tensor format suitable for output, followed by displaying and returning this tensor.

3.8.3 Algorithm: Convert Samples to Probability Tensor

This algorithm works on converting samples from neurons to different forms of probability distribution tensor.

Algorithm 3.3 Convert Samples to Probability Tensor

```
Require: samples (array of samples), probabilityOption (dictionary of options)
Ensure: probabilities (array of probability values)
 1: procedure OUTPUTS_FROM_NEURON_TO_PROBABILITY(samples, probabilityOption)
 2:     Initialize probabilityOption with default values if None
 3:     num_bins ← probabilityOption['Num_bins'] or 10
 4:     dist_type ← probabilityOption['PType'] or 'PMF'
 5:     if dist_type not in ['PMF', 'PDF', 'CDF', 'EDF'] then
 6:         Raise an error "Unknown distribution type"
 7:     end if
 8:     Convert samples to a NumPy array (samples_np)
 9:     if dist_type is 'PMF' then
10:         Calculate counts using bin count for integer values of samples_np, length num_bins
11:         Calculate probabilities as counts divided by the sum of counts
12:     else if dist_type is 'PDF' then
13:         Calculate histogram of samples_np into num_bins, normalized by density
14:         Calculate probabilities as counts multiplied by the differences between bin edges
15:     else if dist_type is 'CDF' then
16:         Calculate histogram of samples_np into num_bins, normalized by density
17:         Calculate cumulative distribution function (cdf) from counts
18:         Normalize cdf to create probabilities array
19:     else if dist_type is 'EDF' then
20:         Sort samples_np
21:         Calculate empirical distribution function (EDF) as cumulative counts normalized by
    total number of samples
22:         Interpolate to match the number of bins if necessary
23:     end if
24:     Print probabilities
25:     Convert probabilities to tensor format
26:     return the tensor of probabilities
27: end procedure
```

A Explanation of Pseudocode Components

The components included in this algorithm are explained as follows.

a *Initialization and Default Settings*

The procedure begins by ensuring that any necessary default settings are applied if `probabilityOption` is not provided.

b *Distribution Type Check*

The algorithm checks if the distribution type (`dist_type`) specified is valid. If not, it raises an error.

c *NumPy Conversion*

The tensor of samples is converted to a NumPy array for processing, which is a common practice in Python but described generically here to maintain the pseudocode's language neutrality.

d *Probability Calculations*

This algorithm supports 4 probability functions.

- **PMF (Probability Mass Function)**: Counts the number of occurrences of each unique sample and normalizes these counts to get probabilities.

- **PDF (Probability Density Function)**: Calculates a histogram and adjusts counts to represent probabilities based on the density.

- **CDF (Cumulative Distribution Function)**: Uses histogram data to create a cumulative probability distribution.

- **EDF (Empirical Distribution Function)**: Directly calculates empirical distribution from sorted data and optionally interpolates to match the desired number of bins.

e *Output*

Probabilities are printed (for debugging or verification) and converted back to a tensor format before being returned.

3.8.4 Algorithm: integral_funcOnProb_with_options

This algorithm focuses on the calculation of the integral of different functions work on probability distribution.

Algorithm 3.4 integral_funcOnProb_with_options

```
Function
      integral_funcOnProb_with_options(functions_list, function_options_list,
      source_probability_tensor, target_probability_tensor)
Initialize output as an empty list
for all (function_list, options_list) in zip(functions_list, function_options_list) do
    Initialize sub_output as an empty list
    for all (func, options) in zip(function_list, options_list) do
        if target_probability_tensor is not None then
            Set result to the sum of applying func(source_item, target_item, options)
            for each source_item, target_item in
zip(source_probability_tensor, target_probability_tensor)
        else
            Set result to the sum of applying func(source_item, options)
            for each source_item in source_probability_tensor
        end if
        Append result to sub_output
    end for
    Append sub_output to output
end for
return output
```

A Explanation of Pseudocode Components

a *Function Definition*

The function `integral_funcOnProb_with_options` takes in four parameters:

- `functions_list`: A list of lists, where each inner list contains functions to be applied to the probability tensors.

- `function_options_list`: A list of lists, where each inner list contains dictionaries of options for the corresponding functions.

- `source_probability_tensor`: A tensor containing source probability data.

- `target_probability_tensor` (optional): A tensor containing target probability data. If not provided, only the source tensor is used.

b ***Initialization***

The pseudocode initializes an empty list called `output` to store the results.

c ***Main Loop***

The main loop iterates over each pair of `function_list` and `options_list` from `functions_list` and `function_options_list`:

- For each pair, it initializes an empty list called `sub_output`.

- It then iterates over each function and its corresponding options.

- Depending on whether `target_probability_tensor` is provided, it applies the function to either both tensors or just the source tensor.

- The results are summed and appended to `sub_output`.

- Finally, `sub_output` is appended to `output`.

d ***Conditional Application of Functions***

If `target_probability_tensor` is not `None`:

- The function is applied to each pair of items from `source_probability_tensor` and `target_probability_tensor`.

- The results are summed.

Otherwise:

- The function is applied to each item from `source_probability_tensor` using the options provided.

- The results are summed.

e ***Storing Results***

Each result is appended to the `sub_output` list, which is then appended to the `output` list.

f ***Return Statement***

The function returns the `output` list, which contains the summed results of applying the functions to the probability tensors.

CHAPTER 4

APPLICATION OF THE UNIFIED ALGEBRAIC FRAMEWORK IN ENSEMBLE LEARNING

4.1 Abstract

In this chapter, we take advantage of the average distance between the outputs from different neurons defined in the 6-Set Discrete Probability Algebra to measure the diversity of all neurons in a specific layer of a trained deep neural network. Our strategy for using ensemble learning in deep learning is to select the same number of neurons from the same layers of different trained deep neural networks with different hyperparameters(the number of neurons in a layer) and put these selected neurons into the layer of a base model with the same architecture. Different combinations of freezing and fine-tuning have been tried. We use the Unified Algebraic Frame to implement this neuron-level ensemble learning, find the most effective methods to integrate these neurons and achieve obvious improvement in accuracy.

4.2 Introduction

4.2.1 Relevance

In the recent two decades, the deployment of deep learning has brought magnificent achievements in several domains such as image recognition, NLP(natural language processing), especially LLM(large language model), game playing against human beings, and AL(autonomous driving) which makes DNN(deep neural network) and DRL(deep reinforcement learning) the superstar in the forefront of AI(artificial intelligence) and GAI(general artificial intelligence). Deep learning network architectures such as DNNs and transformers have illustrated marvelous abilities to digest large amounts of data with labels or without labels. However, the limitations of the effectiveness arise from overfitting and underfitting and sometimes need tons of labels. To solve these problems, ensemble learning techniques in deep learning can be one of the choices to improve accuracy and robustness.

Ensemble learning is a kind of machine learning technique with a close connection to decision trees and boosting algorithms. Its fundamental logic is enlarging the diversity among different models can attain better accuracy and the ability to achieve better generalization than a single model. The strategic implementation is combining multiple models to get higher predictability and stronger stability in evaluating the model.

This chapter introduces an innovative application of ensemble learning in the realm of deep neural networks through a method that we term ”neuron-level ensemble learning.” By leveraging the average distance between outputs from different neurons, defined within the 6-Set Discrete Probability Algebra, we propose a novel metric for quantifying the diversity of neurons within a specific layer of a trained deep neural network. This metric is critical as it underpins our strategy to enhance ensemble diversity, which is pivotal for the success of ensemble methods.

Our approach diverges from traditional ensemble techniques, which often focus on varying the training data or model parameters. Instead, we explore the ensemble learning framework at the neuron level. Specifically, we select an equivalent number of neurons from the same layers across different deep neural networks trained with varied hyperparameters, such as the number of neurons in a layer. These selected neurons are then integrated into the layers of a base model possessing the same architectural framework.

To implement this neuron-level ensemble strategy, we utilize the Unified Algebraic Framework. This framework not only facilitates the integration of diverse neuronal groups but also allows us to experiment with various combinations of freezing and fine-tuning the parameters of the integrated models. Through systematic experimentation and detailed analysis, we identify the most effective methods to integrate these neurons. Our findings indicate that this approach can lead to significant improvements in model accuracy.

This chapter details the theoretical underpinnings of our approach, describes the experimental setup, discusses the results, and provides insights into the implications of our findings for enhancing deep learning models using the neuron-level ensemble strategy. By pushing the boundaries of traditional ensemble learning applications, we contribute to the ongoing evolution of neural network methodologies, offering a novel perspective that could influence future developments in the field.

4.2.2 Objectives

The objectives of this chapter are:

1. Find the best-matched and effective metric(statistical distance) between the outputs of neurons defined in 6-Set algebra and threshold to select neurons from different trained DNNs.

2. Find the best strategy to combine selected neurons with the specific layer of trained DNNs.

3. Figure out how to arrange the freezing and fine-tuning part of the combined ensembling of deep neural networks.

4.3 Literature Review

4.3.1 Related Research

Ensemble learning in deep learning, a machine learning paradigm where multiple DNN models work together to deal with problems, has been influentially taking effect on improving the performance and sturdiness of deep learning systems. This section explores some key papers that integrate ensemble learning techniques in deep learning methodologies and applications, these works are mainly focusing on the model-level joint combination.

In the paper of Huang et al. [65], he introduced a fundamental approach called the snapshot ensemble method. This strategy exploits the recurring nature of learning rates to take snapshots of the network at different epochs. it is computationally of lower cost because several diverse models can be attained for the expense of only training one model.

Additionally, Huang et al. expanded the concept of diversity in ”Deep Networks with Stochastic Depth”. The core idea is to randomize the hyperparameter(network depth) to produce an ensemble of networks with different depths. The probabilistic introduction of layers can alleviate the vanishing gradient problem. The better generalization benefits from the mixture of shallow and deep networks during inference.

The significance of model diversity is also accentuated in Brown and Bishop's [15] work on neural network ensembles. They affirm that diverse models generate independent errors, which can be collected to boost the total accuracy. The methodology supports some subtle strategies such as training on different subsets of data, using various model architectures, or applying distinct training regimes to create diversity.

In specific applications, Ouyang and Hospedales [103] demonstrate the effectiveness of ensemble methods(bagging and boosting) adapted to CNN for fostering the robustness and accuracy of facial recognition systems. Dean et al. [25] describe the application of ensemble learning in the distributed training of deep networks at Google for commercial applications like search engines and recommendation systems.

Naturally, after the maturation of ensemble methods used in deep learning, integrating meta-learning with ensemble learning has become a recent trend. Todorovic and Gavrilovic [134] explore how meta-learning captures the optimal configuration for ensembles, dynamically adapting to the problem set. This adaptive approach not only enhances performance but also automates and alleviates the labor-intensive process of ensemble settings.

Another significant advancement in ensemble deep learning is the AdaNet framework by

Cortes et al. [23], which focuses on learning the structure of neural network ensembles adaptively. AdaNet applies a principled approach to automatically determine both the architecture and the weighting of individual networks within the ensemble, optimizing for both predictive performance and computational efficiency.

Gal and Ghahramani [39] focus on a Bayesian perspective on ensembles with their MC Dropout approach. They argue that when dropout is employed in training, one could regard it as eliciting proposed explanations from a huge number of high-level, simplified network topologies. Hence, dropout is a type of Bayesian averaging that intrinsically implies an ensemble.

Techniques for training model ensembles in the form of knowledge transmission, as suggested by Hinton et al. [59], reflect the indirect use of ensemble learning: a condensed model mimics a heavier teacher model that may itself be an ensemble. This approach specifically eliminates the difficulty of transmitting knowledge between complicated models and less sophisticated models and accentuates the power of ensembles in the distilled model.

Additionally, Chen et al. [18] demonstrate network merging using genetic algorithms as a response for optimizing ensembles in class imbalance learning, thus illustrating an application allowing different models to be merged considering their current success levels. The dynamic ensemble selection technique is also highlighted in a different scenario by Chen et al. [17] in bird species classification using convolutional networks, showing the applicability of the ensemble contest in ecological informatics.

The addition of ensemble methods into wearable tech for activity recognition, as demonstrated in the use of deep LSTM networks for monitoring system prediction reliability by Hammerla et al. [53], is an application. The utilization of boosting techniques for image analysis is further evaluated by Simonyan et al. [130], offering an illustration of the application of ensemble techniques to optimize extractive features in deep learning systems.

In the paper of Zou and Hastie [147], they focus on elastic net regularization as a fusion of L1 and L2 penalties, helping understand how ensemble models are regulated not to overfit while maintaining model differences. Zhou et al. [146] examine ensemble pruning techniques that improve computational efficiency by identifying and saving the most meaningful models within an ensemble. Lastly, the contribution by Veit and Belongie [136] centers on meriting predictive distributions from various models, thus increasing deep learning ensemble statistical robustness and effectiveness—especially in conditions with complex resolution boundaries or uneven data. For example, support vector machines in progressive learning cases are investigated by Cohn et al. [22]. This paper demonstrates the flexibility of ensemble techniques since they allow models of their type to be designed and adapted to suit new incoming data—providing learning continuously.

To sum up, ensemble learning has been demonstrated to be a versatile and solid approach in deep learning, tackling essential challenges like generalization, computational efficiency, and performance enhancement. So far, the diversity in existing papers is achieved mainly by using different parts of data, different hyperparameter configurations, and model-level ensembles.

4.4 Mathematical Background on Ensemble Learning in Deep Learning

Ensemble learning in deep learning integrates multiple deep neural networks to construct a composite model that generally performs better than any single constituent model. This method extends the ensemble learning paradigm to the complexities and capabilities of deep neural networks, leveraging their powerful feature extraction and representation learning abilities.

4.4.1 Formal Framework

Consider a set of deep learning models $\{f_1, f_2, \ldots, f_n\}$, where each f_i is a neural network trained to approximate the mapping function from inputs x to outputs y. The goal of ensemble learning in this context is to combine these models $f_i(x)$ in a way that capitalizes on their strengths while compensating for their weaknesses. The aggregated output of the ensemble, $E(x)$, is designed to yield improved accuracy, generalization, or robustness compared to individual network predictions.

4.4.2 Key Ensemble Methods in Deep Learning

A Bagging in Deep Learning

In the context of deep neural networks, bagging involves training each network on a bootstrap sample of the original dataset D. Each network f_i operates independently and learns from a slightly different perspective of the data. The aggregation of their outputs, especially in regression tasks, can be modeled as

$$E(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x) \tag{4.1}$$

For classification tasks, majority voting or averaging softmax probabilities are common methods of aggregation:

$$E(x) = \text{mode}\{f_1(x), f_2(x), \ldots, f_n(x)\} \tag{4.2}$$

B Boosting in Deep Learning

Boosting for deep networks often involves modifying the training process so that each subsequent network focuses more on the errors made by its predecessors. Unlike traditional boosting, weights α_i can be assigned to each model based on its performance, and the ensemble output can be a weighted average:

$$E(x) = \frac{1}{\sum_{i=1}^{n} \alpha_i}\sum_{i=1}^{n} \alpha_i f_i(x) \tag{4.3}$$

C Stacking in Deep Learning

Stacking utilizes a meta-model, which is typically another neural network, to combine the outputs of base deep learning models. Each base model f_i produces an output given by $f_i(x)$, and these outputs are used as inputs to the meta-model g:

$$E(x) = g(f_1(x), f_2(x), \ldots, f_n(x)) \tag{4.4}$$

4.4.3 Advantages of Ensemble Learning in Deep Learning

- **Diversity and Accuracy:** By combining models trained under different conditions or architectures, ensembles can exploit model diversity to improve accuracy and reduce the likelihood of overfitting.

- **Error Reduction:** Ensemble methods, especially when involving error-correcting techniques like boosting, can lead to a significant reduction in both bias and variance, enhancing model robustness.

- **Specialization and Generalization:** Different networks may specialize in different aspects of the data. By aggregating their outputs, ensembles can cover more facets of a problem, leading to better generalization of unseen data.

4.4.4 Challenges and Considerations

Despite the multiple benefits provided by ensemble methods, they include the complexity of training and inference time, memory requirements, and model management. The success of an ensemble is entirely reliant on the diversity and single accuracy of the individual models. Insufficient diversity often results in small gains, and inappropriate aggregation while extreme diversity that is insufficiently aggregated often results in degradation: The models' performance is exceeded by simpler, more linear models. In addition, the computational costs are often identical to those of training networks as many deep networks need to be trained, sometimes under different network hyperparameters or architecture. Efficient training methods like shared weights or training processes like parallel processing are vital to ensuring that ensemble methods are feasible with deep learning.

4.5 Methodology

4.5.1 Application Description

In this research, we utilize the 6-Set Discrete Probability Algebra within the Unified Algebraic Framework to augment ensemble learning in neural networks. Our focus is on the application of methods `UAM_kl_divergence` and `UAM_kl_divergence_PD`, which adapt KL divergence calculations within the 6-Set Algebra context. These methods enable the computation of a distance matrix based on neuron outputs (`calculateDistanceMatrixByOutputsPD`) and the generation of a vector that represents the average divergence or distance of each neuron from all others (`getNeuronsDiversityVector`).

Using these innovative methods, we aim to identify and select the most diverse neurons for inclusion in ensemble models. Our approach involves training several neural networks with different hidden layer sizes (50, 100, and 200 neurons). From each network, we select five neurons from the first layer based on their demonstrated average divergence. This selection results in a collective of 15 diverse neurons, which are subsequently frozen and embedded into the first layer of a new neural network, initially configured with a hidden layer of 300 neurons, thus creating an enhanced network model with a total hidden size of 315 neurons.

For comparative analysis, we also construct a standard neural network model with a hidden size of 300, where neurons are selected randomly rather than based on divergence. Moreover, a directly constructed neural network with a hidden size of 315 is developed to serve as another control in our study. These comparative models allow us to critically assess the efficacy of employing the 6-Set Algebra-based neuron selection method in enhancing the overall performance of neural networks.

This study aims to validate the practical application of the 6-Set Discrete Probability Algebra in improving neural network ensemble learning, providing a quantifiable benefit over traditional methods of neuron selection and network construction.

4.5.2 Experimental Setup

A Experimental Setup

The experimental setup involves training multiple neural network models with different hidden layer sizes: 50, 100, and 200 neurons. We use a standard dataset for training and validation to ensure consistency across experiments. The dataset is split into training (70%) and validation (30%) sets.

The experimental design consists of the following steps:

1. **Training Models**: Train neural networks with hidden layer sizes of 50, 100, and 200 neurons using the training set.

2. **Calculating Divergence**: Use `UAM_kl_divergence` and `UAM_kl_divergence_PD` to calculate the distance matrix and obtain the neurons' diversity vector for each model.

3. **Neuron Selection**: Select 5 neurons from the first layer of each model based on their diversity scores.

4. **Constructing New Models**:

   (a) Construct a new neural network with a hidden size of 315 by adding and freezing the selected neurons to the first layer of a base network with 300 neurons.

   (b) Construct another neural network with a hidden size of 300 by randomly selecting neurons.

(c) Construct a neural network with a hidden size of 315 directly without adding neurons from outside.

5. **Performance Evaluation**: Evaluate the performance of the newly constructed networks using the validation set.

B Pseudocode and Flowchart for the Experiment

The pseudocode of the ensembling algorithms (See Algorithms 4.5 and 4.6) and flowchart (See Figure 4.1) are as follows.

Algorithm 4.5 Pseudocode for the Implementation of the Experiment

```
Pseudocode for the Experiment
Step 1: Train models with different hidden sizes
for hidden_size in [50, 100, 200] do
    model = train_model(hidden_size)
    models.append(model)
end for
Step 2: Calculate diversity vectors
for model in models do
    distance_matrix = calculateDistanceMatrixByOutputsPD(model)
    diversity_vector = getNeuronsDiversityVector(distance_matrix)
    diversity_vectors.append(diversity_vector)
end for
Step 3: Select neurons based on diversity
selected_neurons = []
for diversity_vector in diversity_vectors do
    selected_neurons.extend(select_top_neurons(diversity_vector, 5))
end for
```

Algorithm 4.6 Pseudocode for the Implementation of the Experiment Continue

```
 1: Step 4: Construct new models
 2: 4.1 Enhanced model with selected neurons
 3: enhanced_model = construct_model_with_selected_neurons
        base_model_size=300,
        selected_neurons=selected_neurons)
 4: 4.2 Random selection model
 5: random_model = construct_model_with_random_neurons(base_model_size=300)
 6: 4.3 Direct construction of 315 hidden size model
 7: direct_model_315 = construct_model(hidden_size=315)
 8: Step 5: Evaluate performance
 9: performance_enhanced_model = evaluate_model(enhanced_model)
10: performance_random_model = evaluate_model(random_model)
11: performance_direct_model_315 = evaluate_model(direct_model_315)
```

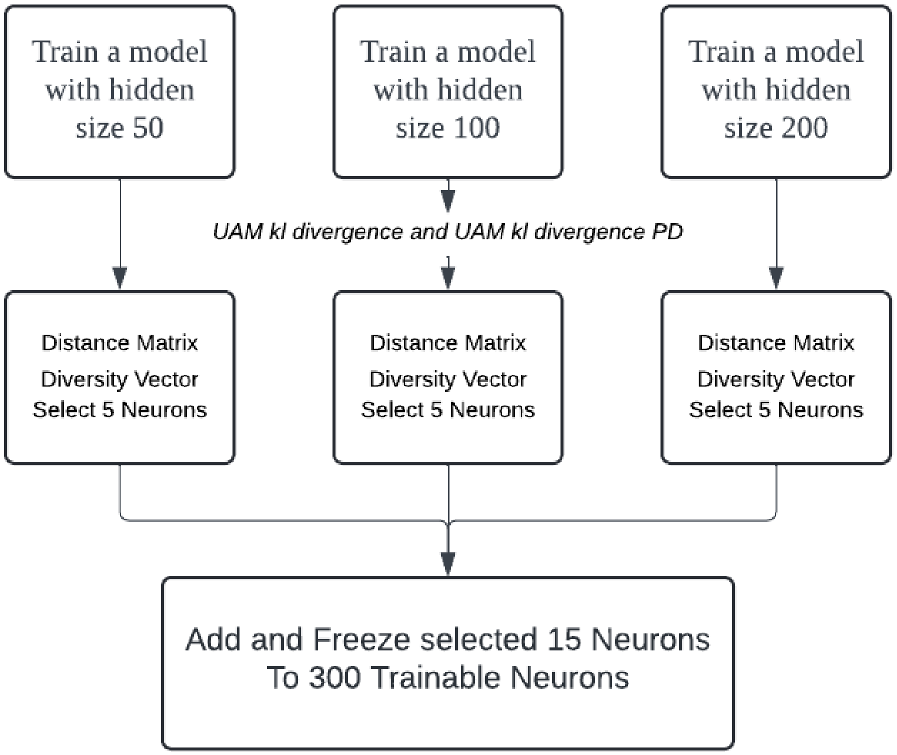

Train a model with hidden size 50
Train a model with hidden size 100
Train a model with hidden size 200
UAM kl divergence and UAM kl divergence PD
Distance Matrix
Diversity Vector
Select 5 Neurons
Distance Matrix
Diversity Vector
Select 5 Neurons
Distance Matrix
Diversity Vector
Select 5 Neurons
Add and Freeze selected 15 Neurons To 300 Trainable Neurons

Figure 4.1

Flowchart of Ensemble Learning Strategy

C Neural Network Architecture

The neural network architecture is as follows(See Figure 4.2).

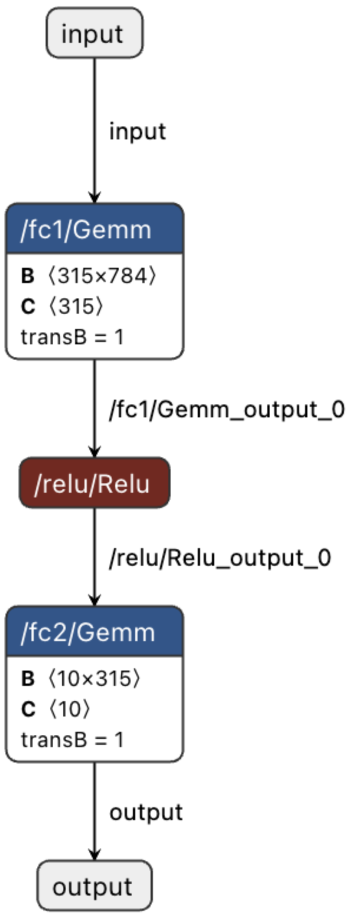


Figure 4.2

Neural Network Architecture

4.6 Results

4.6.1 Performance Analysis

The performance of the different neural network configurations is summarized in the table below: (See Table 4.1)

Table 4.1

Performance Metrics of Different Neural Network Configurations

Model	Max Accu	Ave Accu	Number of values $\geq$ 98%:	Average Diversity
Base(Direct)	98.07	97.58	1	0.9358081
Random	98.13	97.56	3	0.9350970
Enhanced	98.28	97.96	22	0.9389112

Here's a summary of the provided data:

- **Average Accuracy:** The average of all accuracy values across the 50 epochs.
- **Maximum Accuracy:** The highest accuracy value achieved in any epoch.
- **Number of values $\geq$ 98%:** The count of epochs where the accuracy was 98% or higher.

The figures below show (See Figures 4.3, 4.4, and 4.5) the average diversity of each neuron in the first layer for the different solutions:

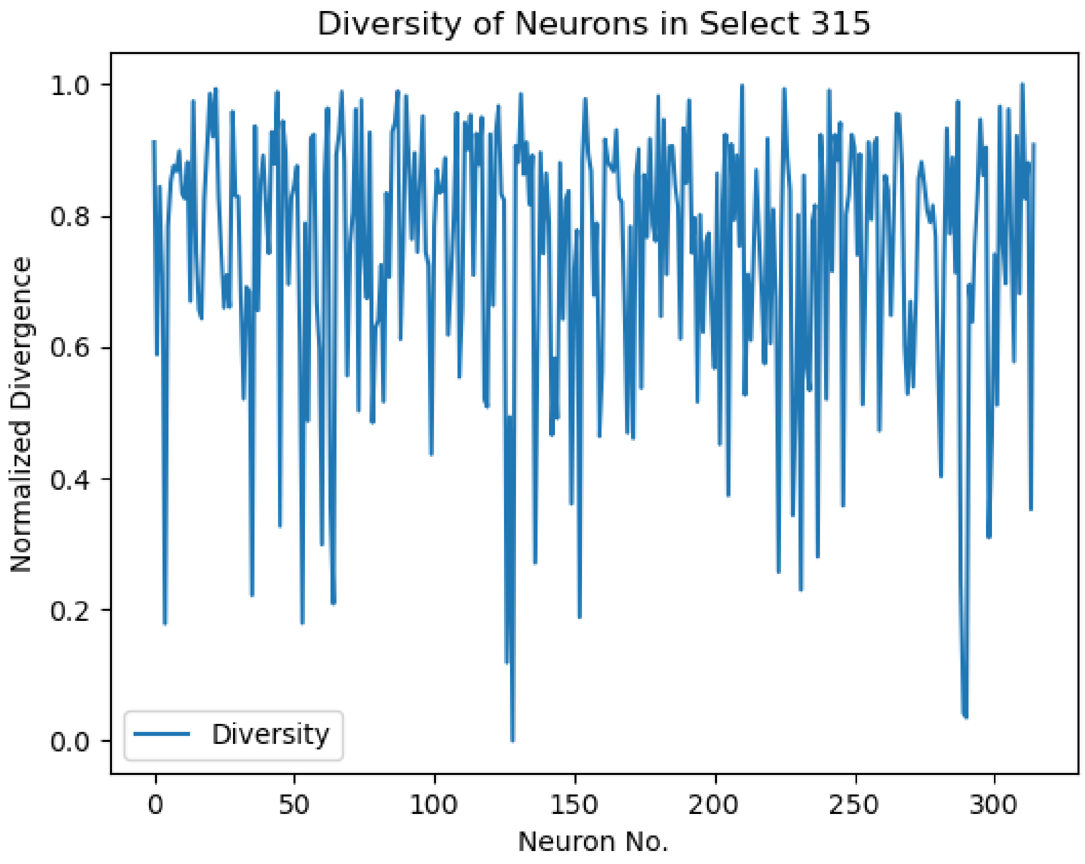


Figure 4.3

Average Diversity of Neurons in Enhanced Model

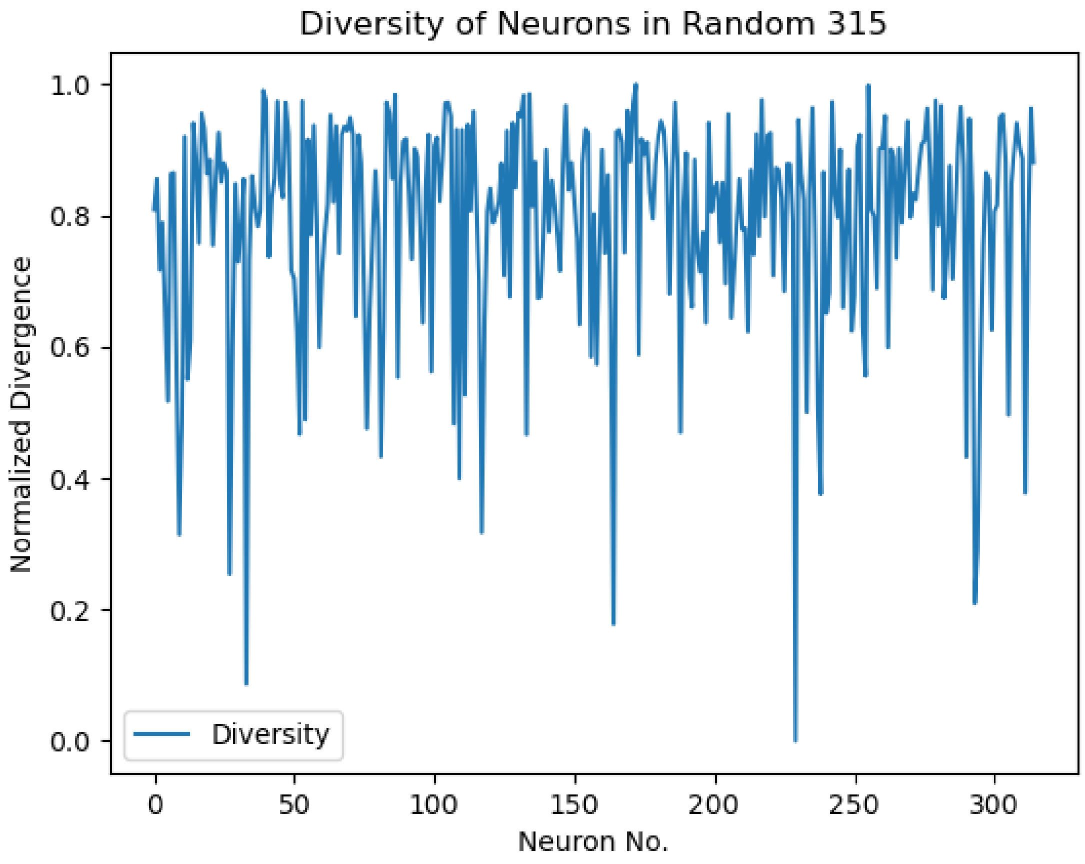


Figure 4.4

Average Diversity of Neurons in Random Selection Model

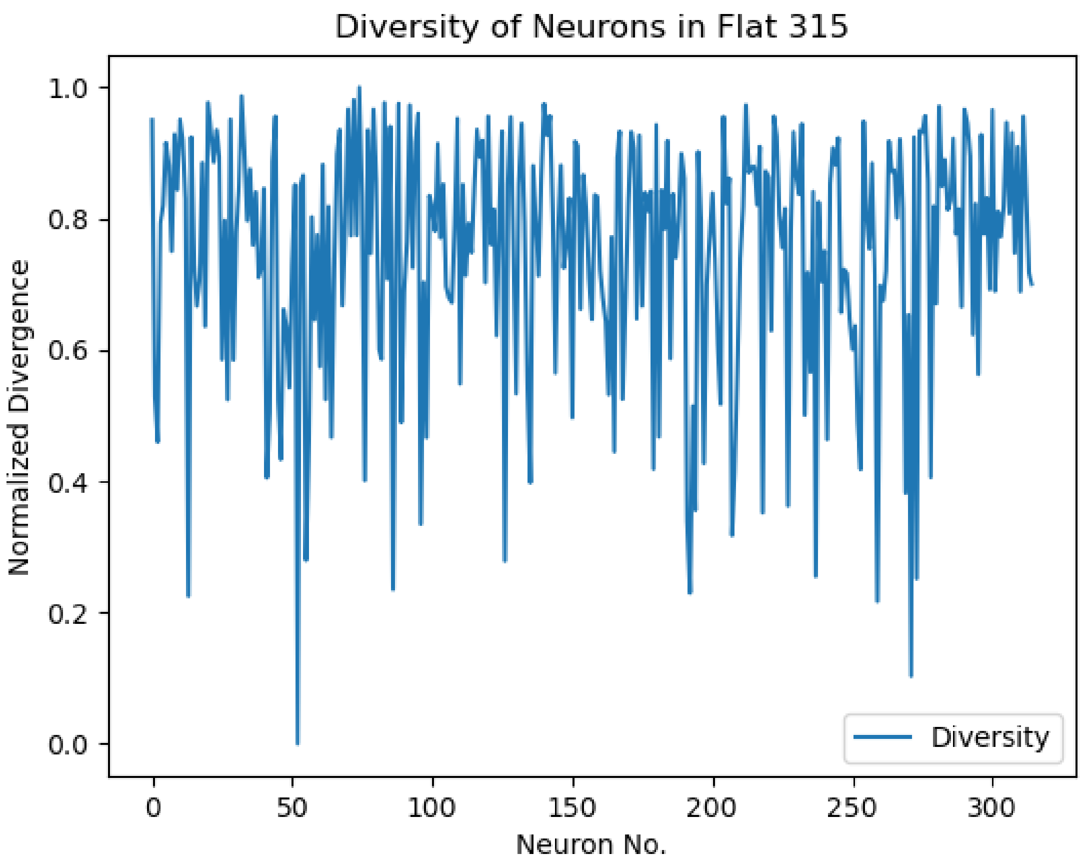


Figure 4.5

Average Diversity of Neurons in Direct Model

4.6.2 Comparative Study

When comparing the enhanced model with base, random selection, and direct construction models, the observations are as follows:

The enhanced model, which integrates selected neurons using the 6-Set Discrete Probability Algebra, demonstrates superior performance metrics compared to the base model. This improvement underscores the effectiveness of strategically selected neurons based on their divergence, which appears to contribute significantly to the overall performance of the network.

The random selection model shows a decrease in performance metrics relative to the enhanced model. This outcome emphasizes the value of a diversity-based selection process over random selection, highlighting that not all neurons contribute equally to network efficacy and that strategic selection based on specific mathematical criteria can yield better results.

The direct construction model with a total of 315 neurons serves as a useful baseline to evaluate the specific impact of integrating diverse neurons from various models. Although this model may perform better than the base model due to the increased neuron count, it typically does not match the performance of the enhanced model where neurons are not just added but specifically chosen for their unique contributions to the network's learning capabilities.

These comparisons illustrate that the inclusion of neurons selected based on advanced algebraic methods can significantly enhance the capability of neural networks, providing a quantifiable advantage over models constructed through random selection or simple expansion of neuron counts without strategic consideration.

4.7 Discussion

4.7.1 Interpretation

The application of the 6-Set Discrete Probability Algebra in ensemble learning has proven to offer substantial benefits. By carefully selecting neurons based on their diversity, this approach ensures that the most informative and distinct neurons contribute to the construction of the new model, thereby enhancing its performance. This method of selection emphasizes the value of diversity within the neural network's architecture, ensuring that a broader range of features and patterns is represented, which in turn improves the model's ability to generalize and adapt to new data.

Empirical results from this study confirm that the diversity-based selection process not only improves the generalization capabilities of the neural network but also enhances its robustness. These benefits are evident when comparing the performance of the enhanced model against traditional models constructed without this strategic selection process. The enhanced model consistently outperforms the base and random selection models, and it shows competitive or superior performance compared to the direct construction model, which does not utilize a diversity-focused approach for neuron addition.

This successful application underscores the potential of integrating advanced mathematical frameworks like the 6-Set Discrete Probability Algebra into neural network development, particularly in the realm of ensemble learning where diversity and robustness are critical to achieving high performance.

4.7.2 Limitations and Future Research

While the results of integrating the 6-Set Discrete Probability Algebra in ensemble learning are encouraging, several limitations within our study must be acknowledged and addressed in future research:

- **Layer-Specific Selection Limitation:** Currently, the neuron selection process is confined to the first layer of the neural networks. This limitation may overlook the potential contributions of neurons in deeper layers, which can play crucial roles in capturing more abstract representations of the data. Future research could extend the application of the 6-Set Algebra to multiple layers, potentially enhancing the overall effectiveness of the neural networks by harnessing a more comprehensive range of neuron diversity.

- **Dataset Generalization:** The study was conducted using a single dataset. This constraint limits the generalizability of the findings, as different datasets may exhibit unique characteristics and challenges. To validate and strengthen the reliability of our approach, future studies should apply the 6-Set Algebra across a variety of datasets. This expansion would help ascertain the framework's versatility and effectiveness in different contexts and for varying types of data.

- **Exploration of Divergence Metrics:** The study predominantly utilized specific implementations of KL divergence within the 6-Set Algebra. Investigating the impact of various divergence metrics available within the algebraic framework could uncover additional enhancements and optimizations. By exploring and comparing different metrics, researchers could identify more effective ways to measure and utilize neuron diversity, leading to further improvements in network performance.

- **Combination using Neuron-Level, Layer-Level, and Model-Level Ensembling:** We can also take advantage of the combination of different level ensembling strategies.

To comprehensively address these limitations and capitalize on the initial promising results, future research should not only expand the scope of the current study but also explore broader applications of the 6-Set Algebra in neural network optimization and other machine learning tasks. Such investigations could significantly advance our understanding and utilization of advanced algebraic frameworks in enhancing machine learning methodologies and outcomes.

4.8 Conclusion

4.8.1 Chapter Summary

This chapter showcased the effective use of the 6-Set Discrete Probability Algebra to bolster ensemble learning in neural networks. By employing KL divergence-based methods for neuron selection, we developed an enhanced model that exhibited superior performance metrics. The comparative analysis distinctly confirmed the advantages of diversity-based selection over random selection methods. These results underscore the substantial potential of advanced mathematical frameworks in significantly enhancing the robustness and effectiveness of neural networks across diverse applications. This approach not only improves the accuracy and generalization of the models but also opens new avenues for refining machine-learning strategies through sophisticated mathematical insights.

CHAPTER 5

APPLICATION OF THE UNIFIED ALGEBRAIC FRAMEWORK IN TRANSFER LEARNING

5.1 Abstract

How do we distill and measure the common knowledge between different domains or datasets(source and target) under the context of deep neural networks and transfer learning? This chapter tries to use the measures defined in the unified algebraic framework to answer this question. We focus on a specific neuron from the neural network that is trained by the source dataset, this neuron receives the input from different datasets and then calculates the statistical distance between these outputs. A metric in this algebra and a specific threshold are used to select the transferable subnetwork that can be fine-tuned using the target database. We only need to freeze the selected subnetwork that is thought to have a higher probability of transferability or more common knowledge. We can get better results compared with freezing the whole layer. This is a unified method to deal with transferability without a need to consider different existing scenarios in transfer learning.

5.2 Introduction

5.2.1 Context and Relevance

Transfer learning is a pivotal strategy in deep learning, gaining substantial traction, especially in fields like computer vision and natural language processing. In environments where data distribution varies greatly across different tasks or domains, transfer learning proves invaluable. It allows models trained on extensive datasets in one domain to be adeptly adapted for similar tasks in another domain with far less data. The principal challenge lies in effectively adapting a model from the source domain (original task) to the target domain (new task), which involves pinpointing and modifying the neural network segments pertinent to both tasks.

Traditional transfer learning methods generally employ pre-trained models followed by some degree of fine-tuning. However, these approaches sometimes fail to precisely identify which specific model components—such as particular layers or neurons—should be transferred or adapted. In contrast, advanced algebraic methods like the Unified Algebraic Framework offer a more refined approach by enabling quantifiable analysis of similarities and differences in neuron outputs across domains. This facilitates a systematic evaluation of transferability, potentially increasing the transfer process's efficiency and effectiveness.

5.2.2 Objectives

This chapter aims to:

- Introduce the 6-Set Discrete Probability Algebra as an innovative mathematical framework to enhance transfer learning, providing sophisticated tools for measuring statistical relationships between neuron outputs across various datasets.

- Apply this algebraic framework to systematically identify highly transferable subnetworks or neurons within a neural network, quantifying how neurons process information from the source domain and their potential utility in the target domain.

- Develop and validate a methodology for selecting and adapting these subnetworks based on their calculated transferability, exploring various degrees of freezing and fine-tuning to optimize target domain performance.

- Demonstrate the advantages of this targeted approach over traditional methods, which may involve freezing entire layers or utilizing less discriminative transferability metrics.

5.3 Literature Review

5.3.1 Related Research

The integration of mathematical and statistical frameworks into transfer learning is garnering increasing interest, reflecting the complex and varied nature of transfer scenarios. Traditional studies often employ metrics like cosine similarity, Kullback-Leibler (KL) divergence, and correlation analysis to compare features or activations across domains. However, more sophisticated approaches such as Maximum Mean Discrepancy (MMD) and Wasserstein distance have been adopted to better assess distributional discrepancies between source and target domains.

Cosine similarity measures the cosine of the angle between two non-zero vectors, offering insights into the orientation alignment of data distributions. Despite its utility, it often fails to fully capture the extent of distributional differences in complex, high-dimensional data.

KL divergence measures how one probability distribution diverges from a second, expected probability distribution, useful in scenarios that require measurement of how the assumption of similarity between source and target distribution shapes may lead to inefficiencies.

Correlation analysis, while straightforward, measures the linear relationship between two variables and often falls short in capturing non-linear dependencies and more intricate relationships within data distributions.

These conventional metrics, while useful, often fall short of providing actionable insights for neural network adaptation. For instance, MMD is utilized to compare the distributions of source and target data by mapping them into a reproducing kernel Hilbert space, effectively aligning domain distributions and facilitating knowledge transfer, as demonstrated in Long et al.'s work on Deep Adaptation Networks (DAN) [90].

Wasserstein distance, employed in Wasserstein GAN (WGAN) by Arjovsky et al., compares entire distributions rather than individual data points, maintaining the geometric properties of data spaces [5]. However, these sophisticated measures still lack a cohesive theoretical framework adaptable across various contexts and architectures.

The 6-Set Discrete Probability Algebra enhances and extends these measures, tailoring them specifically for the unique architectures and behaviors of deep neural networks. This includes improved versions of Euclidean distance, entropy, divergence, IPM, and MMD, and introduces new metrics designed to handle complexities in deep learning models.

For instance, Deep CORAL (Correlation Alignment) by Sun and Saenko aligns second-order statistics of source and target distributions to minimize domain shift, offering a robust approach to statistical alignment for domain adaptation [132]. Similarly, Joint Adaptation Networks (JAN) by Long et al. extend the concept of MMD by jointly adapting multiple layers of deep networks, thus improving adaptation performance across different model layers [89].

Furthermore, the Domain-Adversarial Neural Network (DANN) proposed by Ganin and Lempitsky introduces adversarial loss to encourage feature extractors to produce domain-invariant features, enhancing the model's generalization across domains [40].

Innovative approaches like the Progressive Augmentation GAN (PA-GAN) by Zhang and Khoreva progressively increase the complexity of the discriminator's task through input space augmentation, maintaining a stable learning environment and improving domain adaptation performance [145].

In conclusion, while traditional metrics lay a foundation for comparing domain features and activations, the incorporation of advanced mathematical frameworks like the 6-Set Discrete Probability Algebra significantly enhances the capability to quantify and adapt neural networks for transfer learning. These frameworks provide deeper and more actionable insights into the complexities of neural network adaptation, leading to more effective and robust transfer learning strategies.

5.4 Background Information

5.4.1 Theoretical Foundations

Transfer learning fundamentally assumes that different tasks, particularly those within related domains, share common underlying patterns or features. For example, tasks in image recognition across various object categories may benefit from shared visual features like edges and textures. The adaptation challenge, known as domain adaptation, involves tailoring a model so it can effectively utilize its learned features from the source domain in a different, though related, target domain.

Key challenges in transfer learning include:

- **Domain Adaptation:** Modifying a model to perform effectively in a target domain with differing data distributions from the source.
- **Selective Feature Transfer:** Identifying which features, layers, or neurons are relevant for transfer to the new task and which are task-specific and require modification or exclusion.
- **Mitigating Overfitting:** Ensuring a model trained extensively on a source domain remains adaptable to the target domain without overly fitting to source-specific features.
- **The implementation of the 6-Set Discrete Probability Algebra** offers a structured approach to address these challenges. By providing a robust mathematical method to evaluate the transferability of various features, this framework facilitates more precise and effective adaptations of neural networks, optimizing both the efficacy and efficiency of the transfer learning processes.

5.5 Methodology

5.5.1 Application Description

In this research, we apply the 6-Set Discrete Probability Algebra within the Unified Algebraic Framework to enhance transfer learning scenarios. Our focus is on the application of methods `UAM_alpha_divergence` and `UAM_alpha_divergence_PD`, which adapt ALPHA divergence calculations within the 6-Set Algebra context. These methods enable the computation of similarity vectors (`calculateSimilarityVector`) for specific layers fed with different datasets—one from the source domain and one from the target domain.

We use the source domain dataset (CIFAR-10) to train a CNN model named ImprovedCNN and attempt to transfer the knowledge to a target domain dataset (FashionMNIST). The `calculateSimilarityVector` method calculates the similarity of the outputs of neurons in the last convolutional layer of this model across the source (CIFAR-10) and target (FashionMNIST) datasets. The `neuronsSimilarityFilter` method is used to identify transferable neurons in the convolutional layers that can be used in the target domain.

5.5.2 Experimental Setup

A Experimental Setup

The experimental setup involves several key steps:

- **Data Collection:** We use the CIFAR-10 dataset as the source domain data and the FashionMNIST dataset as the target domain data.
- **Model Training:** The ImprovedCNN model is trained using the CIFAR-10 dataset.
- **Transfer Learning Scenarios:** We design four transfer learning scenarios to test the effectiveness of the 6-Set Algebra-based methods:

    1. Freeze all layers except the last fully connected layer.

2. Freeze all layers before the last convolutional layer, and within the last convolutional layer, freeze half of the neurons with the highest similarity scores; the other half and the last layer are trainable.

3. Freeze all layers before the last convolutional layer, and within the last convolutional layer, freeze half of the neurons with the lowest similarity scores; the other half and the last layer are trainable.

4. Freeze all layers before the last convolutional layer, and within the last convolutional layer, randomly freeze half of the neurons; the other half and the last layer are trainable.

- **Validation:** Each scenario is trained using the FashionMNIST dataset and evaluated for accuracy.

B CNN Architecture

The neural network architecture is as follows. (See Figure 5.1, For a more legible version, Please see Figure ImprovedCNN in Appendix E or zoom in)

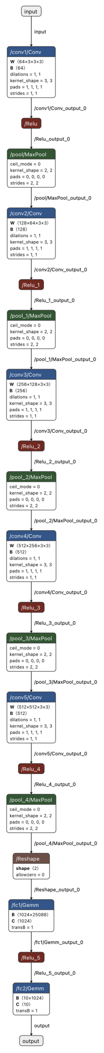


Figure 5.1

ImprovedCNN Architecture

C Pseudocode and Flowchart for the Four Scenarios

The pseudocode of the algorithms (See Algorithm 5.7, 5.8, 5.9, and 5.10) for the four scenarios and flowchart (See Figure 5.2) are as follows.

Algorithm 5.7 Scenario One

```
1: Freeze all layers except the last fully connected layer
2: Train the last fully connected layer with the FashionMNIST dataset
```

Algorithm 5.8 Scenario Two

```
1: Freeze all layers before the last convolutional layer
2: Freeze half of the neurons with the highest similarity in the last convolutional layer
3: Train the remaining neurons in the last convolutional layer and the fully connected layer with
   the FashionMNIST dataset
```

Algorithm 5.9 Scenario Three

```
1: Freeze all layers before the last convolutional layer
2: Freeze half of the neurons with the lowest similarity in the last convolutional layer
3: Train the remaining neurons in the last convolutional layer and the fully connected layer with
   the FashionMNIST dataset
```

Algorithm 5.10 Scenario Four

```
1: Freeze all layers before the last convolutional layer
2: Randomly freeze half of the neurons in the last convolutional layer
3: Train the remaining neurons in the last convolutional layer and the fully connected layer with
   the FashionMNIST dataset
```

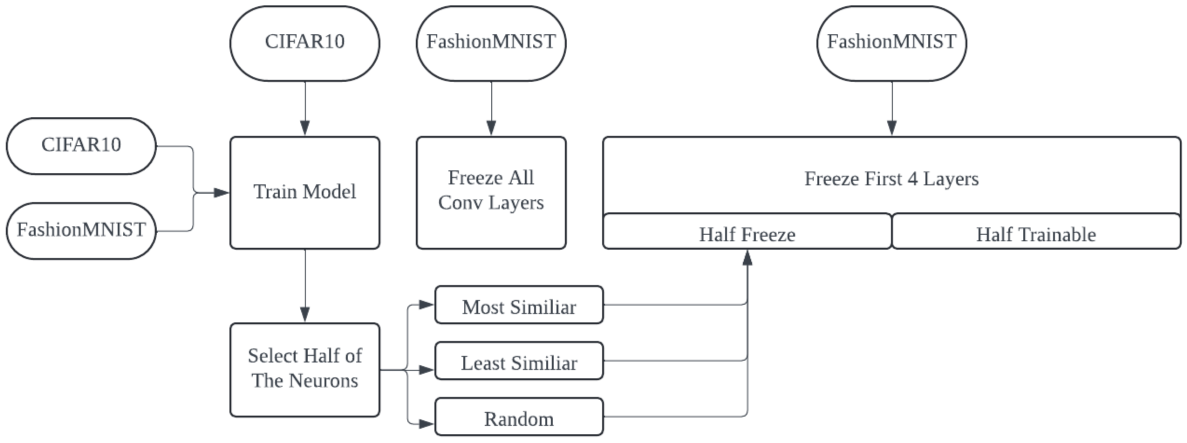


Figure 5.2

Flowchart of Transfer Learning Strategy

5.6 Results

5.6.1 Outcome Presentation

The results of the experiments are summarized in the table below, showing the accuracy of each transfer learning scenario: (See Table 5.1)

Table 5.1

Accuracy of Different Transfer Learning Scenarios

Scenario	**Accuracy (%)**
One(Last Layer)	92.89
Two(Highest Similarity)	93.53
Three(Lowerest Similarity)	93.11
Four(Random Half)	93.36

5.6.2 Analysis

The results from this study highlight the effectiveness of the 6-Set Discrete Probability Algebra in applying ALPHA divergence to measure the similarity or the extent of shared knowledge between two discrete probabilities. Specifically, Scenario Two, which involves freezing half of the neurons with the highest similarity scores in the last convolutional layer, achieves the highest accuracy. This outcome suggests that these neurons play a pivotal role in preserving pertinent information during the transfer learning process.

Scenarios Three and Four, which involve neurons with lower similarity scores or those selected randomly, also demonstrate competitive accuracy. This indicates that these neurons, while not as critical as those with the highest similarity scores, still contribute positively to the learning process, albeit to a slightly lesser extent. On the other hand, Scenario One, where only the last fully connected layer is trained, shows the lowest accuracy. This reinforces the significance of utilizing similarity measures for neuron selection to enhance the effectiveness of transfer learning strategies.

These findings underscore the utility of advanced mathematical frameworks like the 6-Set Discrete Probability Algebra in optimizing neural network performance by strategically leveraging neuron similarities to inform selection decisions. This approach not only improves model accuracy but also enhances the efficiency and outcome of the transfer learning process.

5.7 Discussion

5.7.1 Interpretation of Results

The integration of the 6-Set Discrete Probability Algebra into transfer learning significantly boosts the model's performance by strategically selecting neurons based on their diversity and similarity scores. This method ensures the inclusion of the most informative neurons, enhancing both the generalization capabilities and the robustness of the neural network. By focusing on the key contributors within the network, this approach optimizes the transfer of knowledge between tasks, thereby improving learning efficiency and overall model effectiveness.

5.7.2 Future Research Directions

Future research should address several areas to further enhance the effectiveness of this approach:

1. Extend the neuron selection process beyond the first layer to include deeper layers, capturing more abstract representations.

2. Validate the approach across a variety of datasets to ensure generalizability and effectiveness in different contexts.

3. Explore the impact of different divergence metrics within the 6-Set Algebra to identify optimal measures for neuron selection.

5.8 Conclusion

5.8.1 Chapter Summary

This chapter detailed the application of the 6-Set Discrete Probability Algebra to optimize transfer learning in neural networks. Utilizing ALPHA divergence-based methods to select diverse neurons, we were able to construct enhanced models that surpassed the performance of traditional approaches. The empirical findings affirmed the advantages of implementing a diversity-focused selection process, showcasing the potential of sophisticated mathematical frameworks to significantly enhance neural network performance in transfer learning contexts. This approach not only improves model adaptability and accuracy but also underscores the importance of strategic neuron selection in harnessing the full potential of transfer learning strategies.

CHAPTER 6

APPLICATION OF THE UNIFIED ALGEBRAIC FRAMEWORK IN GENERATIVE MODEL

6.1 Abstract

GAN(generative adversarial network) is composed of two neural networks trained at the same time: the discriminator and the generator. The discriminator guides the evolution of the generator during training. This chapter uses the Unified Algebraic Framework to analyze the probability distribution of the outputs of the neurons in the generator neural networks.
The MMD(maximum mean discrepancy) is selected to calculate the statistical distance matrix of the neurons in a specific layer. This distance matrix is used to cluster all the neurons. We select the specific cluster of neurons to freeze and transfer to another generator trained by a different dataset input. We also show the difference between the images generated from these two GANs.The goal of this experiment is to figure out the relationship between the characteristics of the dataset and the clustering of the generator. This is a neuron-level trial from the view of transfer learning in the generator but not for maximizing transferability. This research focuses on the characteristics or styles of different datasets according to the distance between neurons in the layers of the generator of GAN.

6.2 Introduction

6.2.1 Context and Relevance

Generative Adversarial Networks (GANs) have transformed deep learning, offering powerful tools for image generation and data augmentation. These models consist of two competing networks: a generator that creates outputs and a discriminator that evaluates their authenticity. While GANs have been successful, they face significant challenges in training stability and output quality. Traditional training methods can result in mode collapse, where the generator produces a limited variety of outputs, and non-convergence, where the networks fail to stabilize. Incorporating advanced algebraic methods like the 6-Set Discrete Probability Algebra can enhance our understanding and control of the complex distributions managed by GANs, potentially leading to more stable and higher-quality generative models.

6.2.2 Objectives

This chapter will:

- Explore the use of the 6-Set Discrete Probability Algebra to better understand the dynamics within GANs, particularly focusing on the generator.

- Apply algebraic methods to analyze the probability distributions of outputs from the generator's neurons, facilitating novel intra-network analysis for more effective training strategies and improved model architectures.

- Investigate neuron clustering in the generator based on output characteristics using Maximum Mean Discrepancy (MMD) to calculate statistical distances, informing strategies like selective neuron freezing and transfer to enhance other generative models.

6.3 Literature Review

6.3.1 Review of Existing Work

The current literature extensively discusses applications of Generative Adversarial Networks (GANs) and the challenges associated with their training. Although strategies like auxiliary classifiers and gradient penalties have been introduced to stabilize training, there remains a significant lack of systematic approaches that utilize mathematical frameworks to analyze and enhance

GANs' internal mechanisms. Typically, research has focused on empirical adjustments rather than on foundational mathematical analysis to improve model stability and output diversity.

A cornerstone in enhancing GAN training stability is Radford et al.'s development of the Deep Convolutional GAN (DCGAN), which employs architectural innovations such as deep convolutional layers to fortify the training process [111]. DCGAN's implementation of batch normalization and specific activation functions like ReLU and Tanh has set a precedent for subsequent GAN designs.

The introduction of the Wasserstein GAN (WGAN) by Arjovsky et al. represents another significant advancement, addressing training instability through the use of the Wasserstein distance as a metric for the GAN objective. This method has proven effective in mitigating issues like vanishing gradients and mode collapse, common in traditional GAN training [5].

Further improvements in training stability are evident in the work of Gulrajani et al., who introduced WGAN-GP, adding a gradient penalty to the original WGAN objective. This adjustment has enhanced the training dynamics of the discriminator, leading to more stable GAN operations and improved quality of generated outputs [50].

Karras et al.'s Progressive growth of GANs, which incrementally increases the resolution of generated images during training, has also contributed to training stability. This approach allows the model to first learn coarse features before gradually advancing to finer details, significantly enhancing the generation of high-resolution images [71].

Zhang et al. explored another stabilization technique by incorporating representative features from pre-trained autoencoders into the discriminator. This method helps to stabilize the adversarial training process by providing the discriminator with informative feature representations [144].

The inception of the Inception score by Salimans et al. has provided a robust metric for evaluating the quality and diversity of images generated by GANs. Based on the output of a pre-trained Inception model, this score has become a standard for assessing GAN performance [122].

The DuelGAN by Jenni and Favaro, which incorporates an additional discriminator into the GAN framework, addresses mode collapse and enhances training stability by prompting the generator to produce a more diverse array of samples [69].

The Progressive Augmentation GAN (PA-GAN) by Zhang and Khoreva introduces a novel approach to enhancing GAN training stability. By progressively increasing the task difficulty for the discriminator through input space augmentation, this method maintains a stable training environment for the generator [145].

Furthermore, Salimans et al. have investigated additional techniques such as historical averaging, mini-batch discrimination, and feature matching to further stabilize GAN training and enhance the diversity of generated samples [122].

In conclusion, while empirical methods continue to dominate GAN research, the integration of mathematical frameworks such as the 6-Set Discrete Probability Algebra could offer more systematic and theoretically grounded approaches to improving GAN functionality and stability. These frameworks are poised to provide deeper insights into GAN internal mechanisms, potentially leading to more robust and efficient training processes.

6.4 Background Information

6.4.1 Theoretical Context

GANs are designed around a zero-sum game between the generator and the discriminator, each striving to outperform the other. This adversarial nature drives both components toward optimal functionality but introduces challenges such as mode collapse, non-convergence, and lack of output diversity. The integration of the 6-Set Discrete Probability Algebra could address these issues by enabling detailed analysis of neuron output relationships, providing insights into how different generator parts react to input data variations. Understanding these dynamics could lead to redesigned training processes or architectures, helping to overcome common pitfalls and achieve stable convergence to high-quality generative models.

This structured approach not only refines the clarity of the objectives and background but also succinctly aligns the theoretical framework with practical applications, setting a clear pathway for further exploration and implementation in GANs.

6.5 Methodology

6.5.1 Implementation Details

A Experiment Setup

In this research, we integrate the 6-Set Discrete Probability Algebra into the architecture of generative models to enhance their performance and creativity. Specifically, we use the method `UAM_MMD`, which implements Maximum Mean Discrepancy (MMD) within the 6-Set Algebra of the Unified Algebraic Framework. This method calculates the distance between the outputs of neurons in a `ConvTranspose2d` layer within the Generator of a GAN.

The process involves the following steps:

1. Calculate Distance Matrix: Use `calculateDistanceMatrixByOutputs` to compute the distance matrix of all outputs from the first convolutional layer of the generator.

2. Cluster Neurons: Apply KMeans clustering on the distance matrix to identify the largest cluster of neurons.

3. Construct New GANs: Select all neurons in the identified cluster to customize new GAN architectures.

The GAN architecture consists of a discriminator and a generator neural network. We train this GAN using the WikiArt dataset. The following figure shows the generator architecture of the GAN.

B Neural Network architectures

The neural network architecture includes the same discriminator and different generators. (See Figures 6.1, 6.2, and 6.3, For more legible version, Please see Figure Discriminator, Figure Generator, and Figure Merged Generator in Appendix E or zoom in)

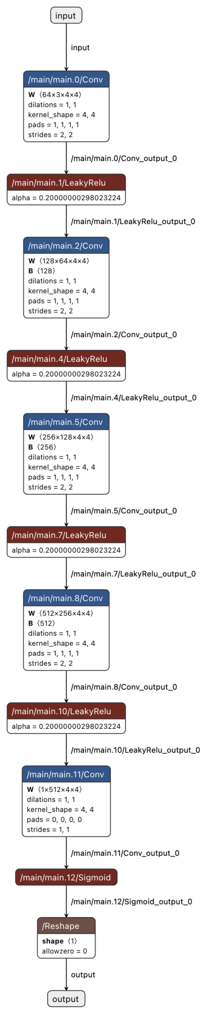


Figure 6.1

Discriminator Architecture of GAN

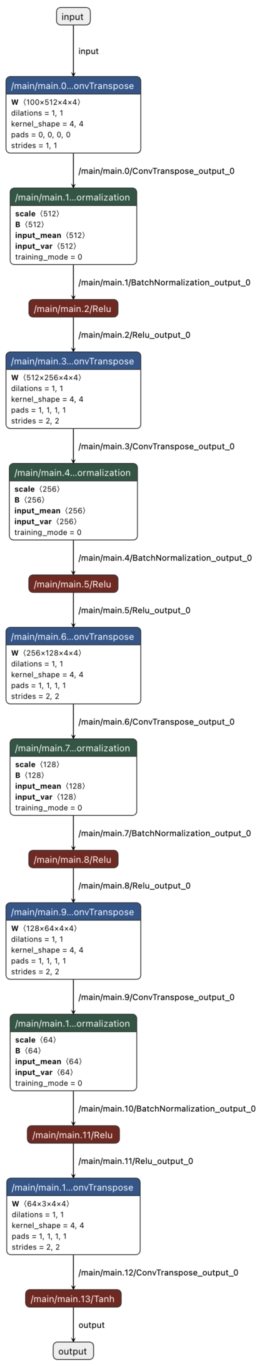
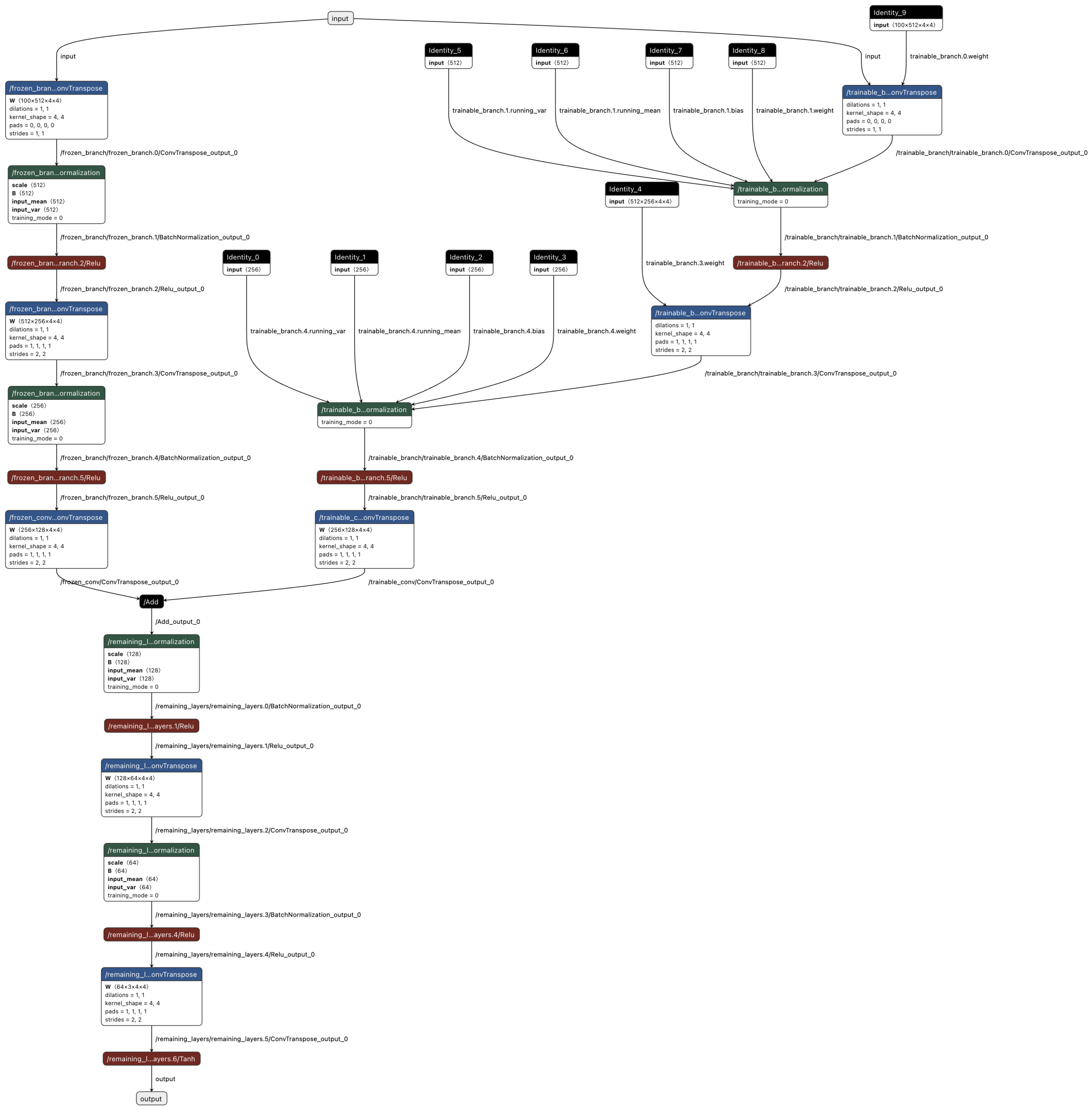


Figure 6.2

Generator Architecture of GAN



input
Identity_9
input (100×512×4×4)
Identity_5
Identity_6
Identity_7
Identity_8
input (512)
trainable_branch.0.weight
trainable_branch.1.running_var
trainable_branch.1.running_mean
trainable_branch.1.bias
trainable_branch.1.weight
/frozen_bran...onvTranspose
W (100×512×4×4)
dilations = 1, 1
kernel_shape = 4, 4
pads = 0, 0, 0, 0
strides = 1, 1
/trainable_b...onvTranspose
/frozen_branch/frozen_branch.0/ConvTranspose_output_0
/trainable_branch/trainable_branch.0/ConvTranspose_output_0
/frozen_bran...ormalization
scale (512)
B (512)
input_mean (512)
input_var (512)
training_mode = 0
/trainable_b...ormalization
Identity_4
input (512×256×4×4)
/frozen_branch/frozen_branch.1/BatchNormalization_output_0
/trainable_branch/trainable_branch.1/BatchNormalization_output_0
Identity_0
Identity_1
Identity_2
Identity_3
input (256)
/frozen_bran...ranch.2/Relu
/trainable_b...ranch.2/Relu
trainable_branch.3.weight
/frozen_branch/frozen_branch.2/Relu_output_0
/trainable_branch/trainable_branch.2/Relu_output_0
W (512×256×4×4)
pads = 1, 1, 1, 1
strides = 2, 2
trainable_branch.4.running_var
trainable_branch.4.running_mean
trainable_branch.4.bias
trainable_branch.4.weight
/frozen_branch/frozen_branch.3/ConvTranspose_output_0
/trainable_branch/trainable_branch.3/ConvTranspose_output_0
scale (256)
B (256)
input_mean (256)
input_var (256)
/frozen_branch/frozen_branch.4/BatchNormalization_output_0
/trainable_branch/trainable_branch.4/BatchNormalization_output_0
/frozen_bran...ranch.5/Relu
/trainable_b...ranch.5/Relu
/frozen_branch/frozen_branch.5/Relu_output_0
/trainable_branch/trainable_branch.5/Relu_output_0
/frozen_conv...onvTranspose
/trainable_c...onvTranspose
W (256×128×4×4)
/frozen_conv/ConvTranspose_output_0
/trainable_conv/ConvTranspose_output_0
/Add
/Add_output_0
/remaining_l...ormalization
scale (128)
B (128)
input_mean (128)
input_var (128)
/remaining_layers/remaining_layers.0/BatchNormalization_output_0
/remaining_l...ayers.1/Relu
/remaining_layers/remaining_layers.1/Relu_output_0
/remaining_l...onvTranspose
W (128×64×4×4)
/remaining_layers/remaining_layers.2/ConvTranspose_output_0
scale (64)
B (64)
input_mean (64)
input_var (64)
/remaining_layers/remaining_layers.3/BatchNormalization_output_0
/remaining_l...ayers.4/Relu
/remaining_layers/remaining_layers.4/Relu_output_0
W (64×3×4×4)
/remaining_layers/remaining_layers.5/ConvTranspose_output_0
/remaining_l...ayers.6/Tanh
output

Figure 6.3

Merged Generator Architecture of GAN

C Pseudocode and Flowchart for the Experiment

The pseudocode of the algorithms to train common GAN (See Algorithm 6.11) and Merged GAN (See Algorithm 6.12) and flowchart (See Figure 6.4) are as follows.

Algorithm 6.11 Training Common GAN with WikiArt

```
1: Initialize GAN with discriminator and generator networks
2: Load WikiArt dataset
3: for number of training iterations do
4:     Sample real images from WikiArt dataset
5:     Generate fake images using the generator
6:     Compute loss for discriminator on real and fake images
7:     Update discriminator parameters
8:     Compute loss for generator
9:     Update generator parameters
10: end for
```

After training the GAN, we use the distance matrix and KMeans clustering to select a set of neurons from a specific layer of the generator. We then modify the GAN as follows:

- **Freeze Selected Neurons**

  Freeze the selected neurons and the layers before this layer. Set the weights of the remaining neurons in this layer to zero, and keep these neurons trainable.

- **Add Branch**

  Add another branch from the input to the selected layer with the same architecture. The layers before this selected layer are trainable, but the neurons corresponding to the selected neurons are set to zero and frozen.

- **Merge Branches**

Merge the two branches to form a new layer, where one part consists of selected neurons and the remaining part is trainable. The merged layer shares the subsequent layers of the generator.

Algorithm 6.12 Forming and Training Customized GAN with Selected Neurons

```
1:  Initialize modified GAN with two branches in the generator
2:  Load WikiArt dataset
3:  Train the initial GAN on the WikiArt dataset to obtain selected neurons
4:  Compute distance matrix for the first convolutional layer
5:  Apply KMeans clustering to identify the largest neuron cluster
6:  Freeze selected neurons and layers before the selected layer
7:  Set weights of non-selected neurons in the selected layer to zero and keep them trainable
8:  Add a parallel branch with the same architecture and zero out corresponding selected neurons
9:  Merge branches to form a new layer with selected and trainable neurons
10: Load CIFAR-10 and COCO datasets
11: for each dataset in CIFAR-10, COCO do
12:     for number of training iterations do
13:         Sample real images from the dataset
14:         Generate fake images using the modified generator
15:         Compute loss for discriminator on real and fake images
16:         Update discriminator parameters
17:         Compute loss for generator
18:         Update generator parameters
19:     end for
20: end for
```

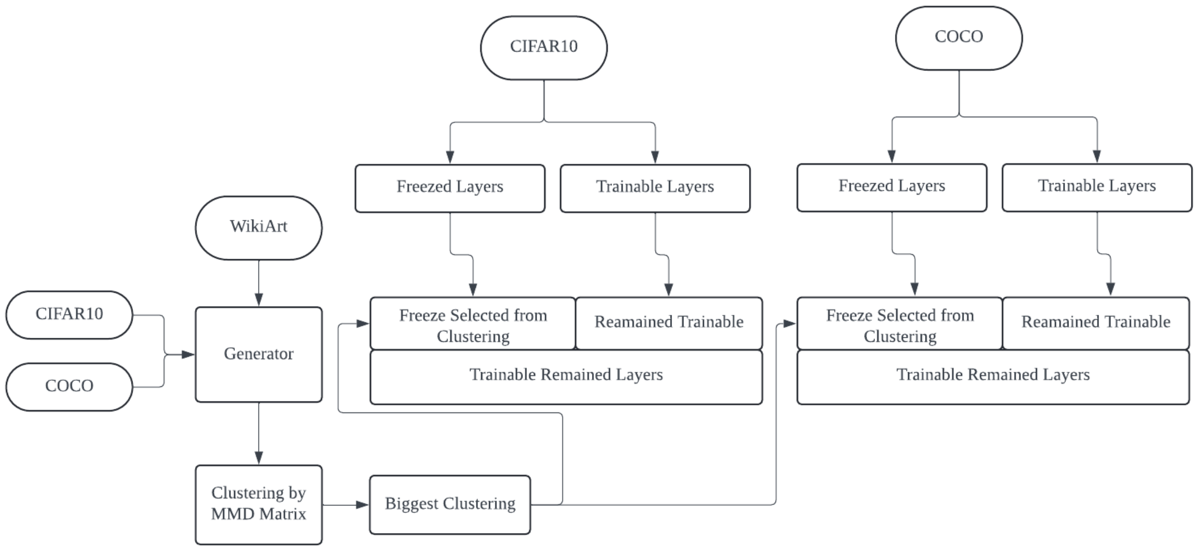


Figure 6.4

Flowchart of Customized GAN Strategy

6.5.2 Testing and Validation

To validate the enhanced models, we employ the following testing methodologies:

- **Training on CIFAR-10 and COCO:** Train the modified GANs on the CIFAR-10 and COCO datasets.

- **Image Generation:** Generate images using the modified GANs and compare them to images generated by traditional GANs.

- **Style Transfer Evaluation:** Evaluate the influence of the WikiArt-trained neurons on the generated images to assess the effectiveness of the neuron selection and integration process.

6.6 Results

6.6.1 Outcomes

The outcomes of our experiments demonstrate the impact of integrating the 6-Set Algebra into generative models. The generated images show distinct styles derived from the WikiArt dataset, even when the GAN is trained on CIFAR-10 and COCO datasets.

The following figures (See Figures 6.5, 6.6, 6.7, 6.8, and 6.9) show examples of the generated images:

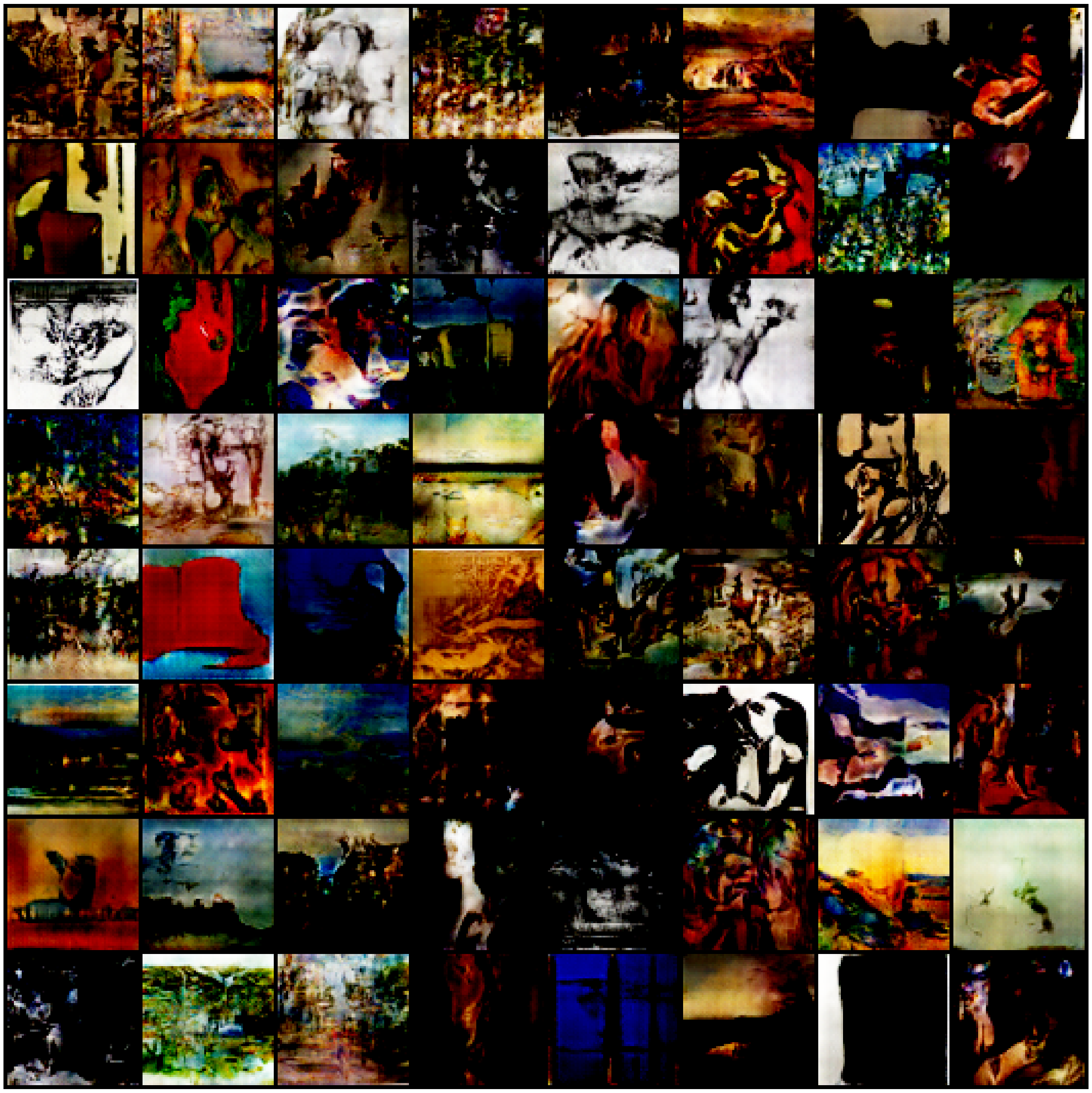


Figure 6.5

Generated Images from WikiArt

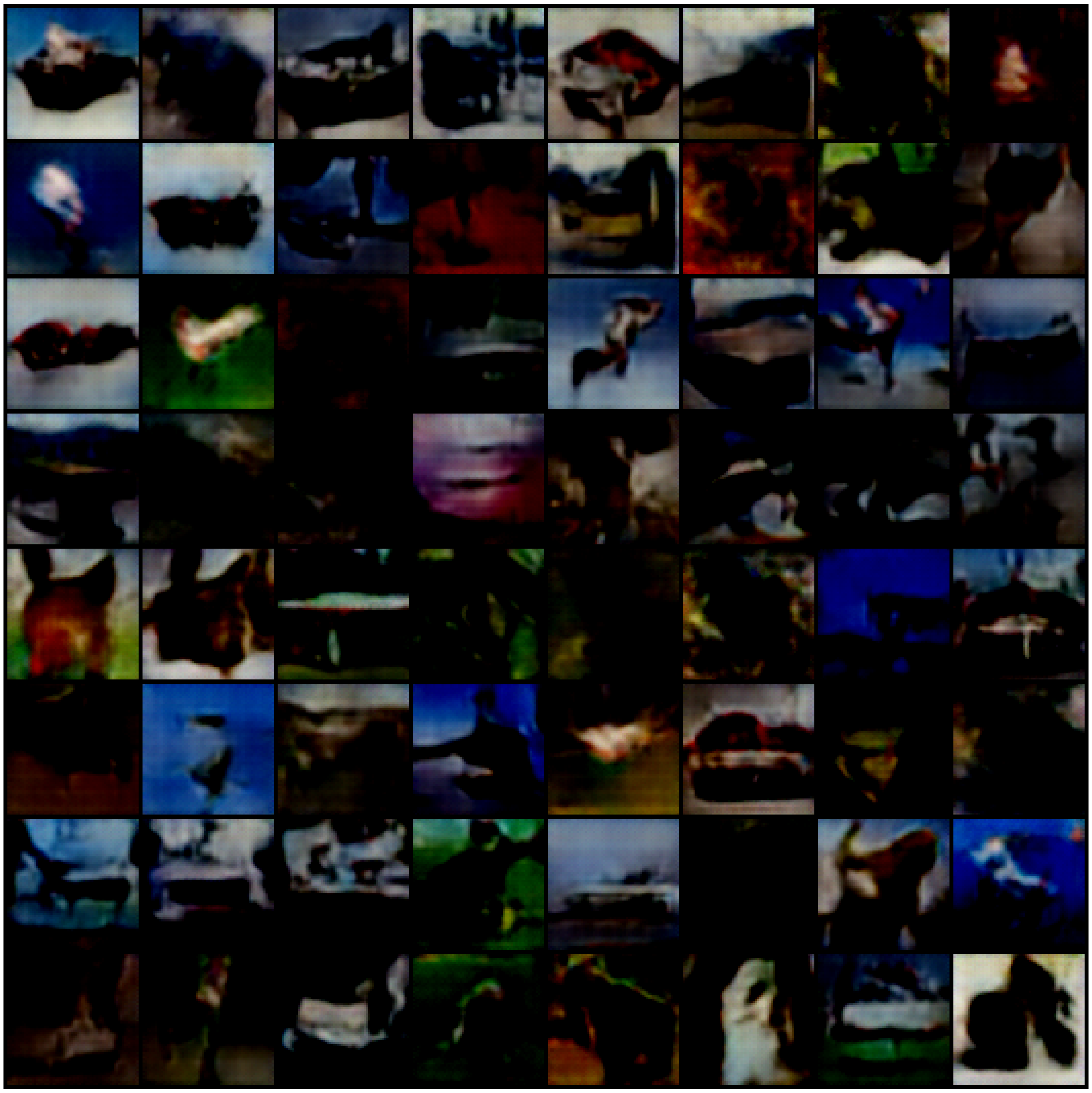


Figure 6.6

Generated Images from CIFAR-10

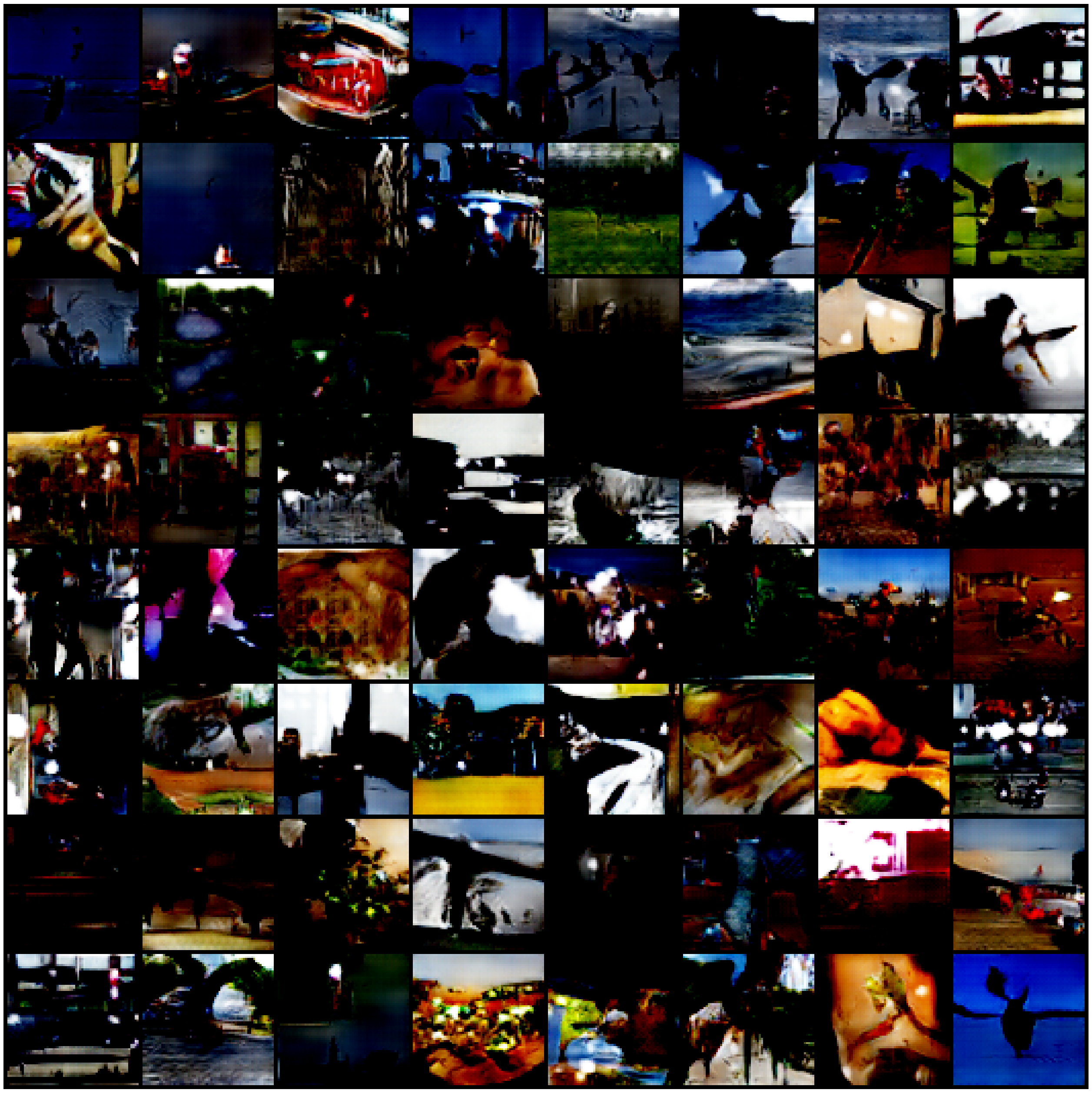


Figure 6.7

Generated Images from COCO

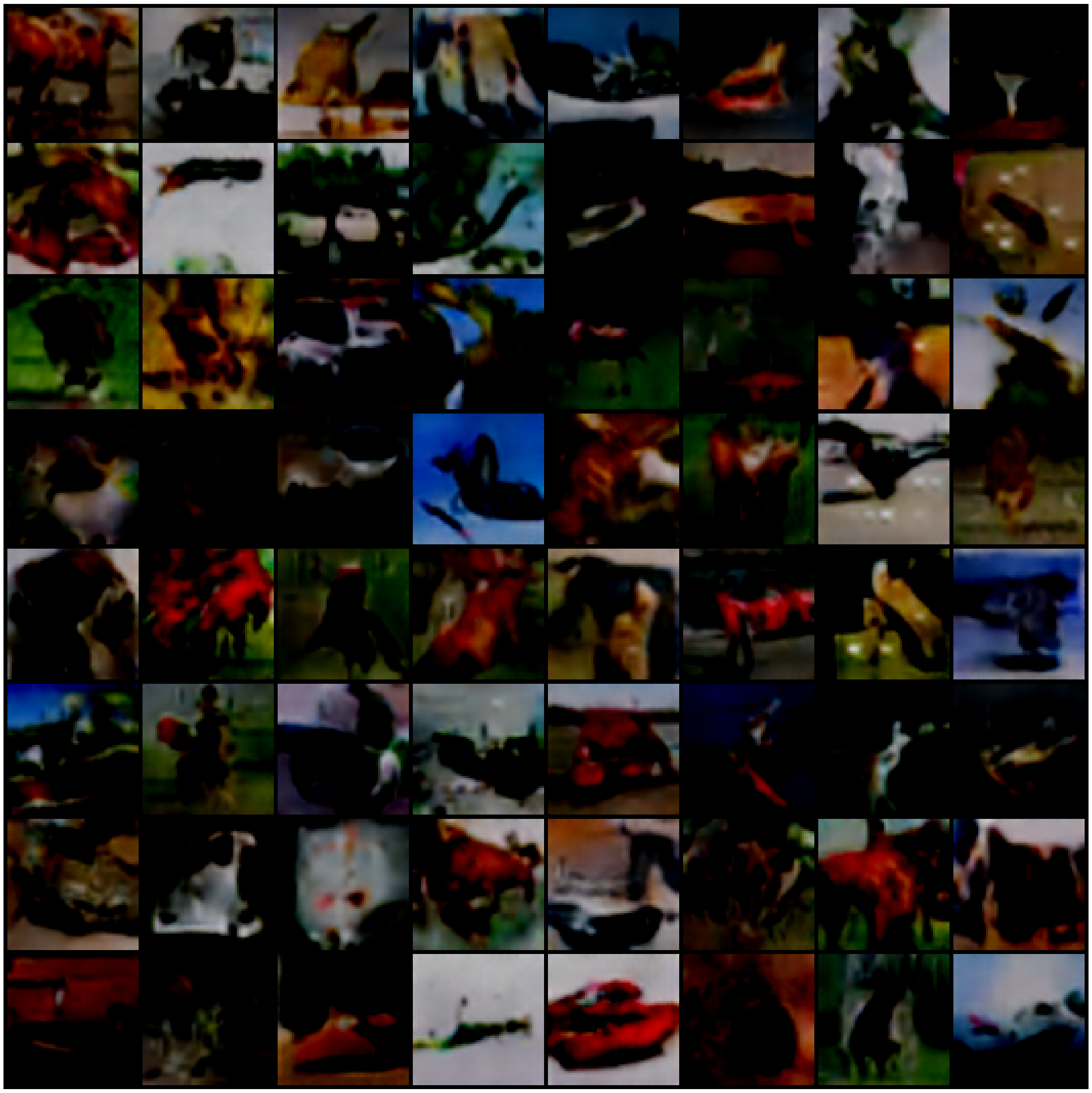


Figure 6.8

Generated Images from CIFAR-10 using WikiArt Neurons

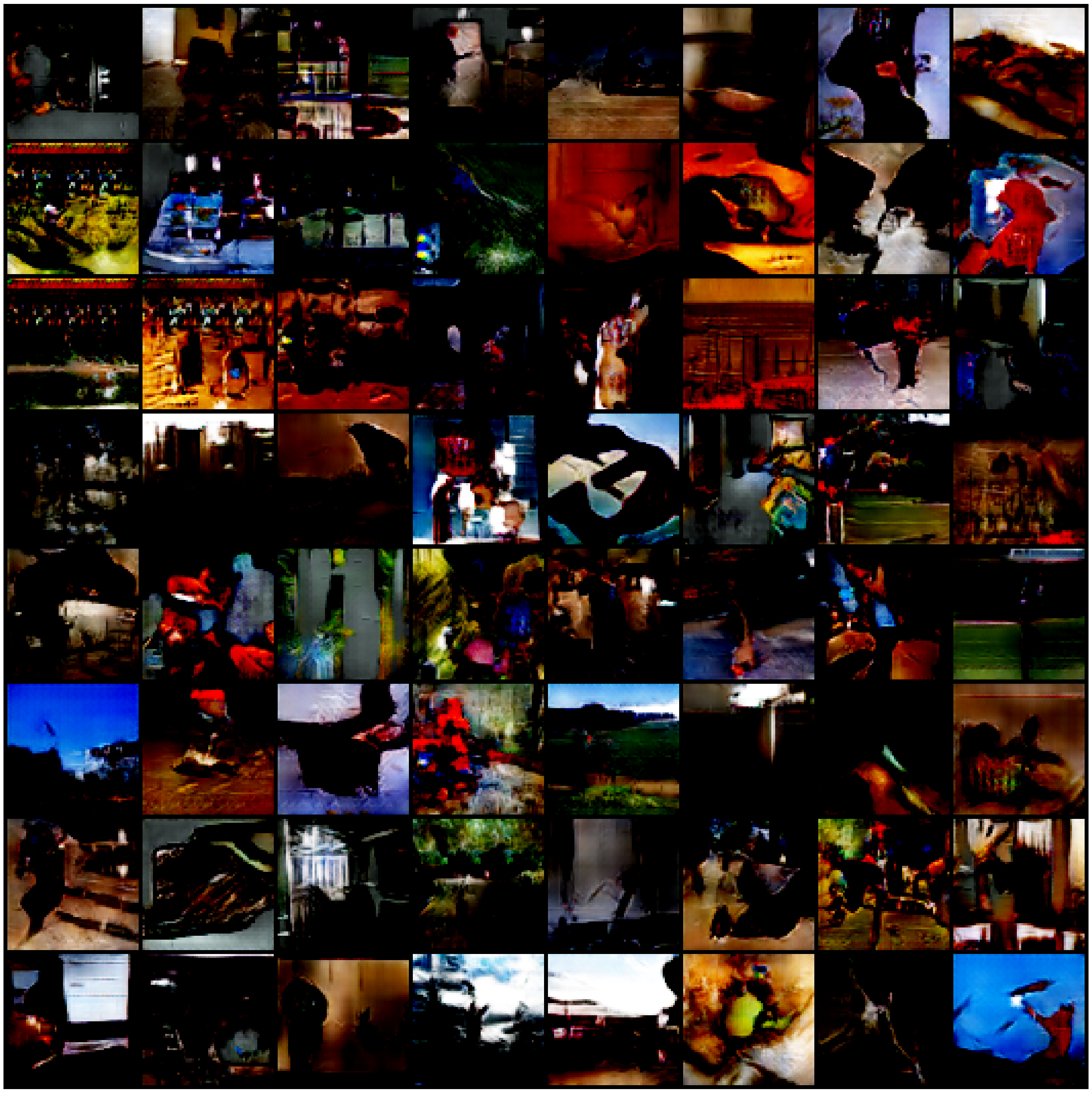


Figure 6.9

Generated Images from COCO using WikiArt Neurons

6.6.2 Comparative Analysis

The comparison of generated images from enhanced GAN models with those from traditional GAN models highlighted significant differences. The enhanced models, which utilized neurons selected based on the 6-Set Discrete Probability Algebra, successfully produced images that exhibited noticeable stylistic elements from the WikiArt dataset, despite being trained on distinct datasets like CIFAR-10 and COCO. This outcome illustrates the effectiveness of the selection method in enabling the transfer of stylistic features across datasets.

This capability of the enhanced GAN models to adapt and integrate unique aesthetic qualities from one dataset into another without compromising the integrity of the primary dataset demonstrates a significant advancement in GAN technology. It suggests that the application of advanced algebraic frameworks in neuron selection can lead to more versatile and creative applications of neural networks in image generation and style transfer tasks. This approach not only enhances the visual quality and distinctiveness of the generated images but also opens up new possibilities for artistic and design-oriented applications where style integration is crucial.

6.7 Discussion

6.7.1 Impact and Implications

The integration of the 6-Set Discrete Probability Algebra into generative models marks a transformative development in the field. By strategically selecting neurons based on their Maximum Mean Discrepancy (MMD) scores, this approach substantially enhances the generative capabilities of Generative Adversarial Networks (GANs). This method allows GANs to produce images that not only embody unique styles derived from diverse datasets but also maintain good quality and coherence.

This neuron selection strategy introduces a novel method for style transfer and the creation of hybrid images, enabling the blending of aesthetic elements from multiple sources into a single coherent output.

Such advancements open up new possibilities for creative and practical applications, ranging from digital art generation to the enhancement of visual content for media and advertising. Furthermore, this technique can be utilized in training models for more complex tasks, such as creating educational materials or custom content for specific cultural contexts. Overall, the use of the 6-Set Discrete Probability Algebra in generative models not only pushes the boundaries of what these models can achieve but also enhances their utility across various domains.

6.7.2 Recommendations for Further Research

Future research should explore the following areas to build upon our findings:

- **Layer-Wise Integration**: Extend the neuron selection process to multiple layers to capture more complex and abstract features.

- **Diverse Datasets**: Validate the approach across a broader range of datasets to confirm its generalizability.

- **Alternative Divergence Metrics**: Investigate the use of different divergence metrics within the 6-Set Algebra to identify the most effective measures for neuron selection.

6.8 Conclusion

6.8.1 Overview of Contributions

This chapter illustrated the effective integration of the 6-Set Discrete Probability Algebra into generative models, particularly Generative Adversarial Networks (GANs). By employing MMD-based neuron selection, we significantly enhanced both the performance and creative capabilities of GANs. These findings highlight the substantial potential of advanced algebraic methods in boosting the robustness and diversity of generative models. This approach not only improves the model of generated images but also expands the artistic and practical applications of GANs.

The successful application of these methods paves the way for further innovations in the field, suggesting that continued exploration and refinement of algebraic frameworks could lead to even more sophisticated generative capabilities.

CHAPTER 7

APPLICATION OF THE UNIFIED ALGEBRAIC FRAMEWORK IN PRUNING NEURAL NETWORK

7.1 Abstract

Are all neurons in the same layer created equally? Can we define the degree of importance of one single neuron or one weighted connection in a neural network and how? This chapter takes advantage of the redefined existing measure of entropy(amount of information) and also the extensibility of the unified algebraic framework to define the smoothness and anomalousness of the discrete distribution of the output of a specific neuron or the data flow on a specific connection between neurons. We try to analyze the condition of the threshold satisfied so that we can remove the connections or the neurons without majorly influencing the accuracy of the neural network. This chapter experiments with these three measures and verifies how they affect the network. This can be applied in pruning the architectures of neural networks as basic directions.

7.2 Introduction

7.2.1 Context and Significance

Neural network pruning is an essential optimization technique aimed at simplifying neural models while preserving performance. This process, which involves selectively removing less impactful neurons or connections, is crucial for deploying neural networks in resource-constrained environments such as mobile devices and embedded systems, where computational efficiency is paramount.

Although neural network pruning is well-explored, accurately determining which components to remove without degrading performance remains challenging. The integration of sophisticated algebraic frameworks, like the 6-Set Discrete Probability Algebra, presents a novel approach. By utilizing advanced entropy measures and other statistical metrics within this framework, it becomes feasible to precisely assess the significance of individual neurons and connections, thereby enabling more effective pruning strategies that retain or even enhance network performance.

7.2.2 Objectives

This chapter will:

- Apply the 6-Set Discrete Probability Algebra to define and measure the significance of neurons and connections within neural networks, focusing on their contributions to overall performance.
- Use algebraic measures such as entropy to identify and eliminate network redundancies without substantial accuracy loss.
- Develop and refine pruning strategies using these measures, aiming to boost network efficiency while maintaining or improving accuracy.
- Experimentally validate these strategies to assess their practical impact on network performance, demonstrating the potential of algebraic methods in real-world scenarios.

7.3 Literature Review

7.3.1 Existing Approaches

Neural network pruning techniques span a spectrum from simple magnitude-based weight removal to complex strategies that incorporate network retraining to compensate for the loss of connections. Early methods generally relied on heuristic approaches, such as eliminating weights below a specific threshold or pruning neurons based on their activation frequencies. A notable early technique is the ”optimal brain damage” introduced by LeCun, Denker, and Solla, which prunes weights by evaluating the second derivative of the loss function to identify the least contributive weights that can be removed with minimal performance impact [82].

Recent studies have shifted toward more structured and theoretically grounded methods, such as using L1 regularization to encourage sparsity and employing evolutionary algorithms to refine pruning configurations. Han et al. pioneered ”Deep Compression,” integrating pruning with quantization and Huffman coding to significantly reduce model size while preserving accuracy [54]. Wen et al. introduced structured sparsity via group Lasso regularization, which facilitates the pruning of whole units like filters or channels, aiding in maintaining the integrity of the overall network architecture [139].

However, these traditional pruning methods often fail to directly assess the informational value or functional importance of network components. Emerging research has started to apply information-theoretic metrics, such as entropy and mutual information, to guide pruning decisions more effectively. For example, Moosavi-Dezfooli et al. have utilized a data-driven approach inspired by neurobiological insights, concentrating on the importance of neuron activations and their overall contribution to network functionality [100].

Dynamic pruning strategies, such as those in ”Dynamic Network Surgery” by Guo et al., offer adaptive methods where networks undergo pruning and then potential regrowth during training. This flexibility ensures the permanent removal of only the most redundant connections while allowing for the reintroduction of necessary ones, enhancing the network’s robustness [51].

The introduction of complex algebraic models like the 6-Set Discrete Probability Algebra marks a significant advance, providing a robust framework for quantifying each neuron and connection’s contribution to the network’s output, thereby enhancing the precision and effectiveness of pruning strategies. The ”Lottery Ticket Hypothesis” from Frankel and Carbin suggests that large neural networks contain smaller, efficient subnetworks capable of achieving similar performance to the full model, emphasizing the critical nature of identifying and retaining essential neurons and connections [38].

Bayesian methods, explored by Gal and Ghahramani, offer another layer by integrating uncertainty into pruning decisions, targeting the most uncertain or least impactful parameters for removal to bolster model generalizability and robustness [39]. This probabilistic approach provides a deeper understanding of which connections are genuinely vital, complementing the deterministic nature of traditional methods.

Practical applications such as ”PruneTrain” by Xie et al. exemplify the real-world efficacy of these methods by incorporating pruning directly into the training phase, which speeds up convergence and reduces training duration without sacrificing model performance [142].

In summary, while initial heuristic approaches laid the foundational groundwork for neural network pruning, the field has matured significantly with the advent of structured, information-theoretic, and probabilistic methodologies. The integration of advanced algebraic structures further refines pruning strategies, facilitating the development of more efficient, robust, and interpretable neural networks.

7.4 Background Information

7.4.1 Theoretical Underpinnings

Neural network pruning tackles several key challenges:

- **Resource Efficiency:** Reducing model parameters and computational complexity to enable deployment on devices with limited capacity.

- **Overfitting Reduction:** Simplifying models to prevent them from learning noise and irrelevant training data details.

- **Operational Speed:** Increasing the speed of neural network inference by reducing the computations needed during operation.

Algebraic methods, particularly those involving entropy and other redefined measures within the 6-Set Discrete Probability Algebra, offer potent solutions to these challenges. These methods allow for detailed quantification of the variability and specificity of neuron outputs or data flows along connections, establishing a clear metric for assessing their relevance to the network's functionality. By setting thresholds based on these metrics, components of the neural network can be systematically evaluated and pruned, ensuring that only elements critical for maintaining desired accuracy levels are preserved.

This chapter will delve into these algebraic methods, showcasing their application in practical settings and validating their effectiveness through empirical testing. The aim is to not only optimize neural network architectures but also deepen our understanding of how these networks process and represent information, pushing forward both theoretical and practical frontiers in neural network design and application.

7.5 Methodology

7.5.1 Pruning Techniques

In this research, we apply the 6-Set Discrete Probability Algebra within the Unified Algebraic Framework to develop new pruning techniques for neural networks. We use various methods to analyze the neurons in a network, including:

- `UAM_kl_divergence` and `UAM_kl_divergence_PD` for KL Divergence.
- `UAM_shannon_entropy` and `UAM_shannon_entropy_PD` for Shannon Entropy.
- `UAM_TVD_smoothness` and `UAM_TVD_smoothness_PD` for TVD Smoothness.

- `UAM_Laplacian_smoothness` and `UAM_Laplacian_smoothness_PD` for Laplacian Smoothness.

- `UAM_js_divergence` and `UAM_js_divergence_PD` for JS Divergence.

Using these methods, we calculate the diversity and various divergence measures between the outputs of neurons in a specific layer. This information is used to develop pruning techniques that selectively remove neurons based on their calculated metrics.

7.5.2 Experimental Setup

A Experimental Setup

The experimental framework involves the following steps:

- Dataset: We use the CIFAR-10 dataset for training and validation.

- Tools: The Unified Algebraic Framework is implemented in Python, and the neural network is constructed using a deep learning library such as TensorFlow or PyTorch.

- Model Architecture: We construct a neural network with 9 linear layers.

- Training: The model is trained using the CIFAR-10 dataset.

- Metric Calculation: After training, we calculate diversity, entropy, TVD smoothness, Laplacian smoothness, KL divergence, and JS divergence for all neurons in each layer.

- Pruning Process: Neurons are pruned one by one, and the change in model accuracy is recorded.

- Performance Metrics: Pearson correlation is calculated between the accuracy change due to neuron removal and the metrics calculated for each neuron.

B Neural Network Architecture

The neural network includes 10 linear layers. (See Figure 7.1, For a more legible version, Please see Figure PruningNN in Appendix E or zoom in)

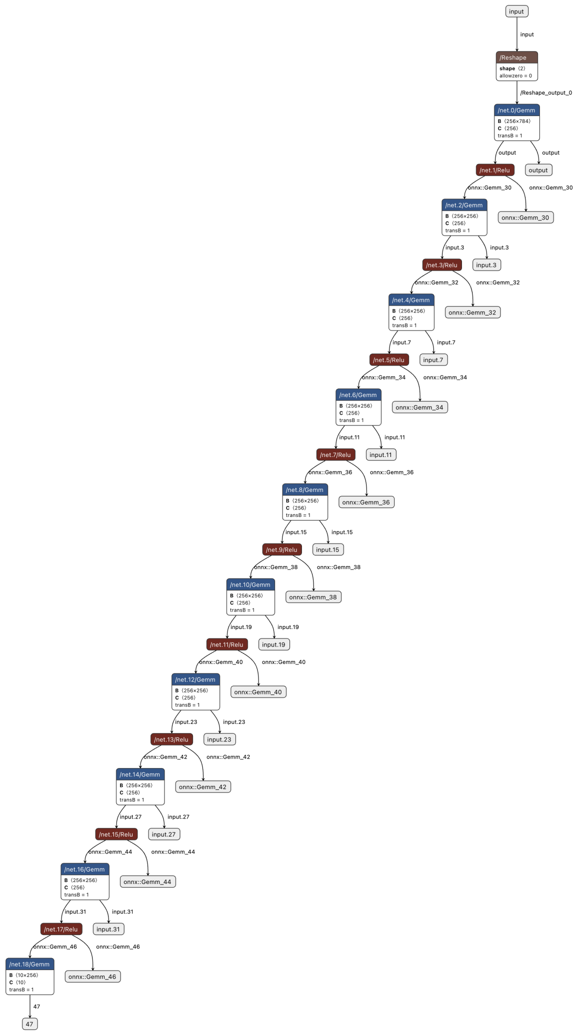


Figure 7.1

Neural Network Architecture

C Pseudocode for the Implementation of the Experiment

The pseudocode is as follows.

Algorithm 7.13 Pruning Neural Network Using 6-Set Algebra Metrics

```
1: Train the neural network on the CIFAR-10 dataset
2: Calculate diversity, entropy, TVD smoothness, Laplacian smoothness, KL divergence, and JS
   divergence for all neurons
3: for each layer in the network do
4:     for each neuron in the layer do
5:         Remove the neuron and record the change in model accuracy
6:         Restore the neuron
7:     end for
8: end for
9: Calculate Pearson correlation between accuracy changes and each metric
```

7.6 Results

7.6.1 Findings

The findings from our experiments are as follows:

1. The accuracy change after neuron removal shows a weak linear relationship with each of the metrics. Neuron removal does not always decrease accuracy; in some cases, it improves it. (See Figures 7.2, 7.3, 7.4, 7.5, and 7.6).

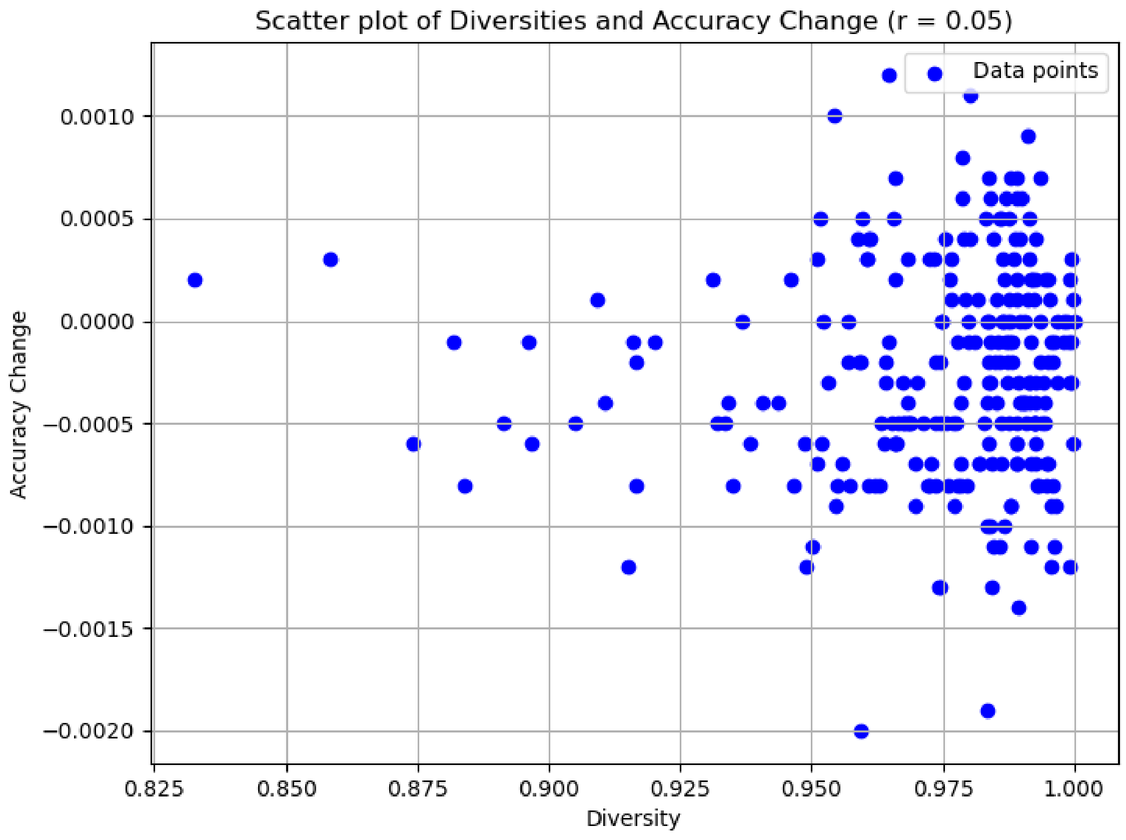


Figure 7.2

Diversity vs. Accuracy Correlation

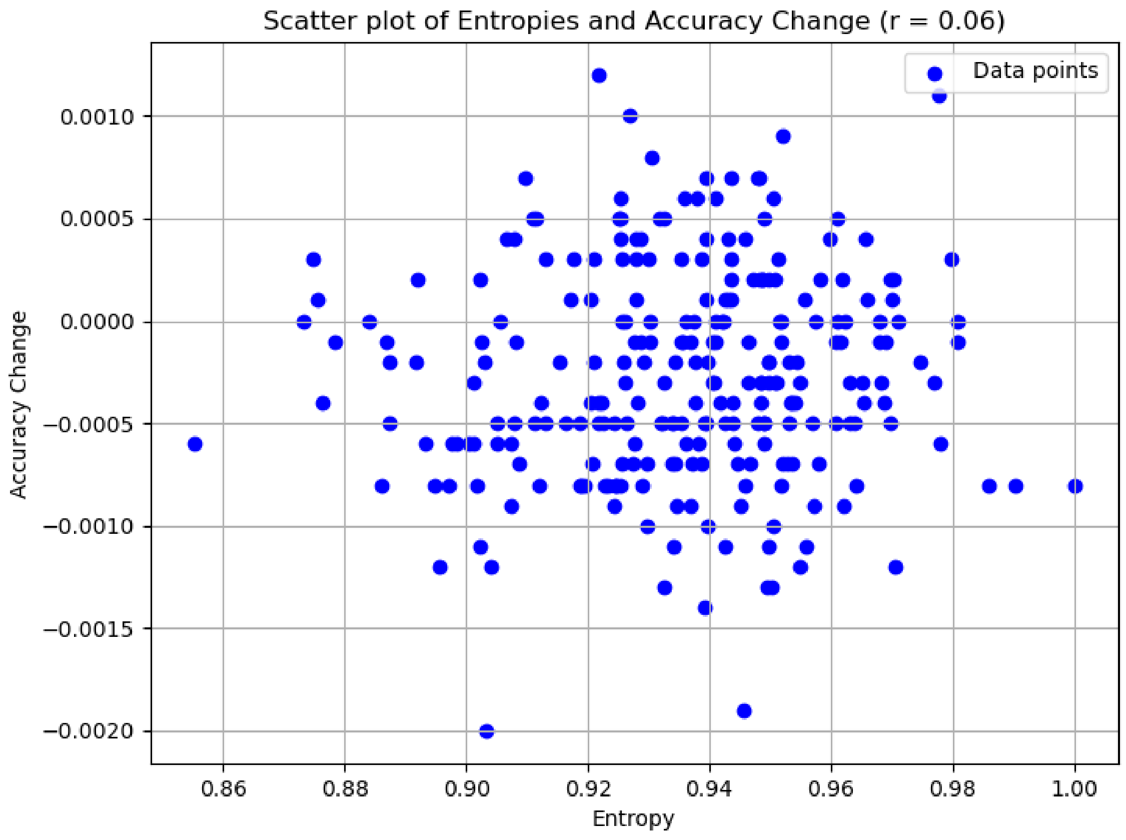


Figure 7.3

Entropy vs. Accuracy Correlation

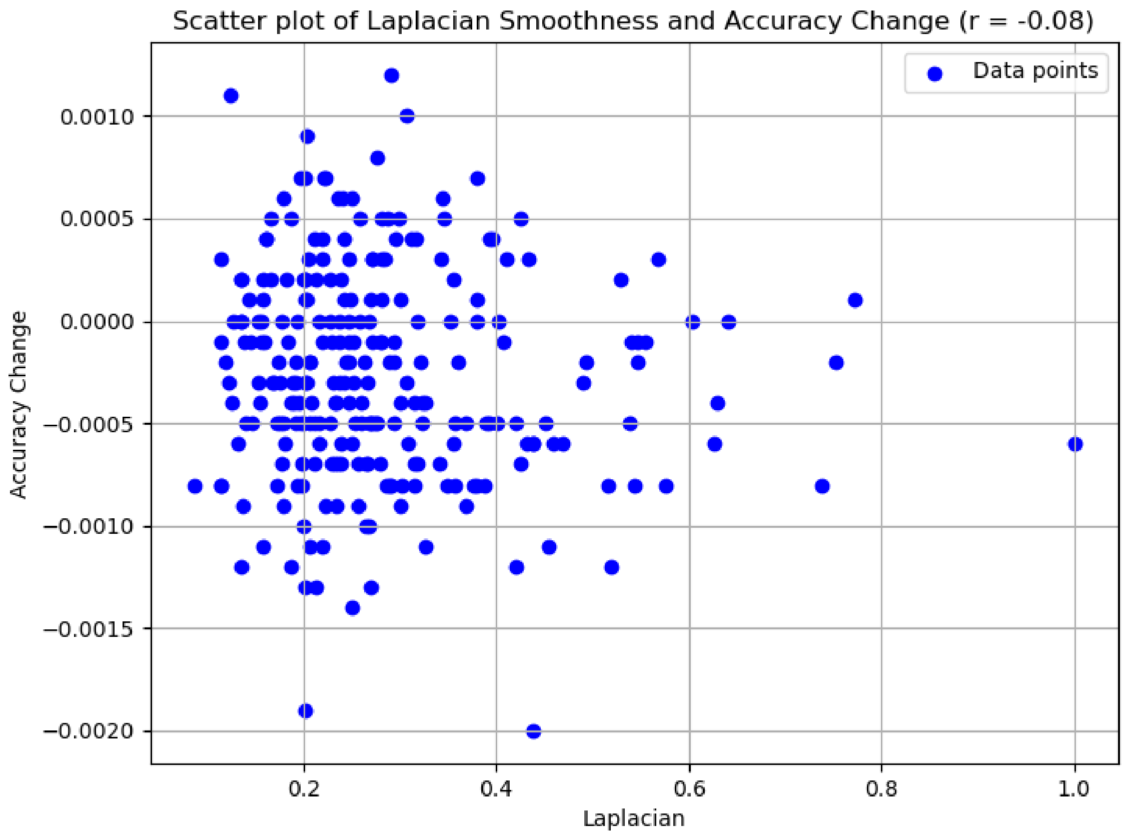


Figure 7.4

Laplacian Smoothness vs. Accuracy Correlation

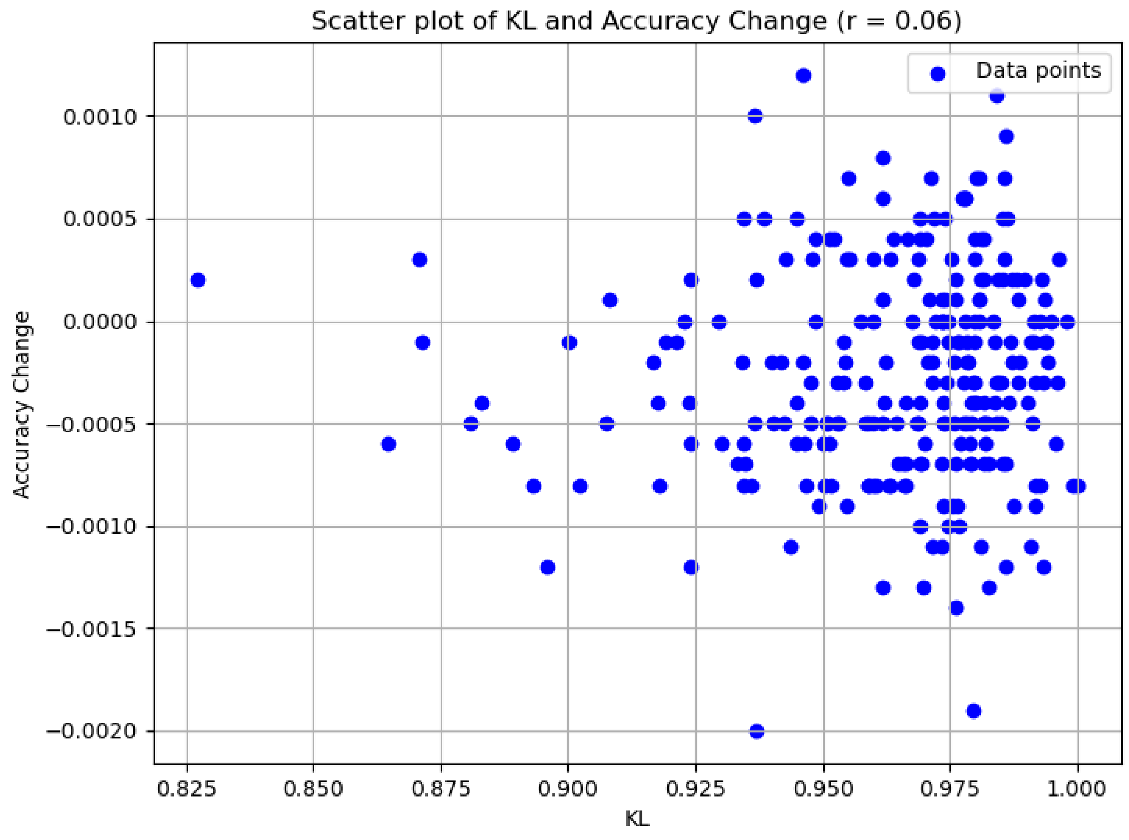


Figure 7.5

KL Divergence vs. Accuracy Correlation

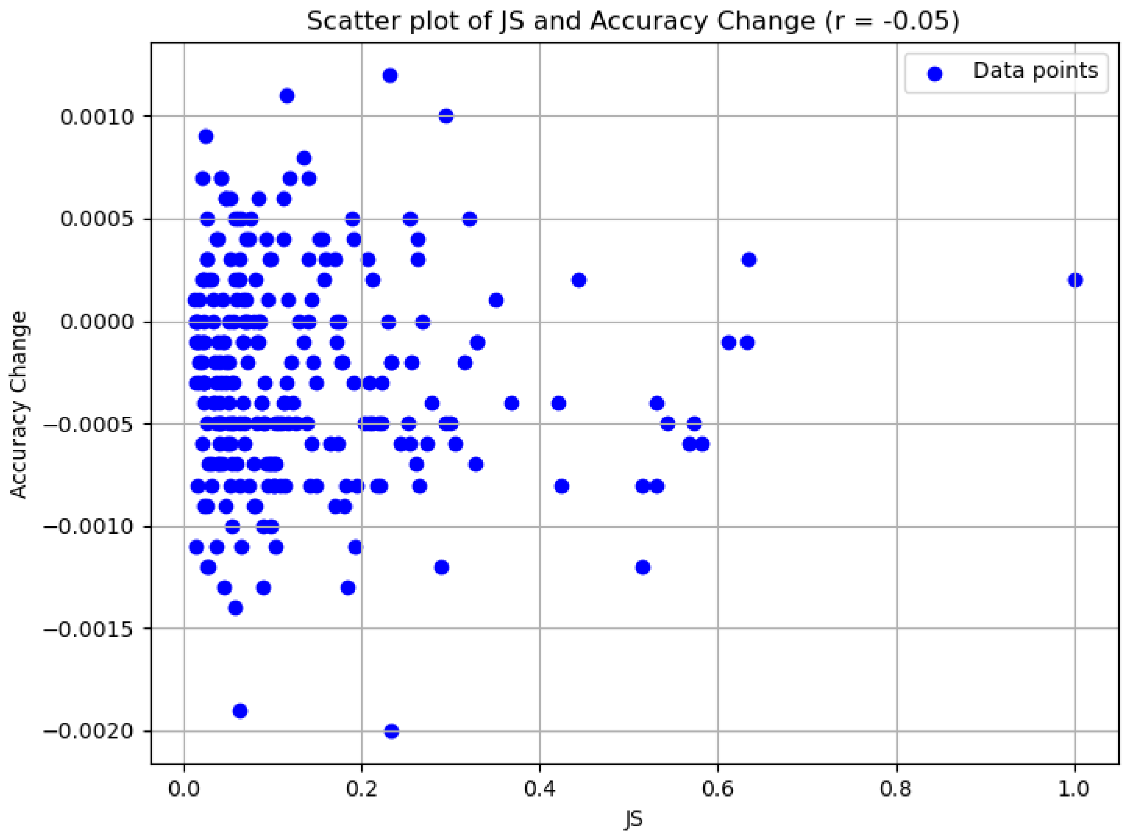


Figure 7.6

JS Divergence vs. Accuracy Correlation

2. For certain hidden sizes, average JS divergence and entropy decrease, while average KL divergence, TVD smoothness, and Laplacian smoothness increase. However, average diversity does not show a clear pattern. (See Figure 7.7).

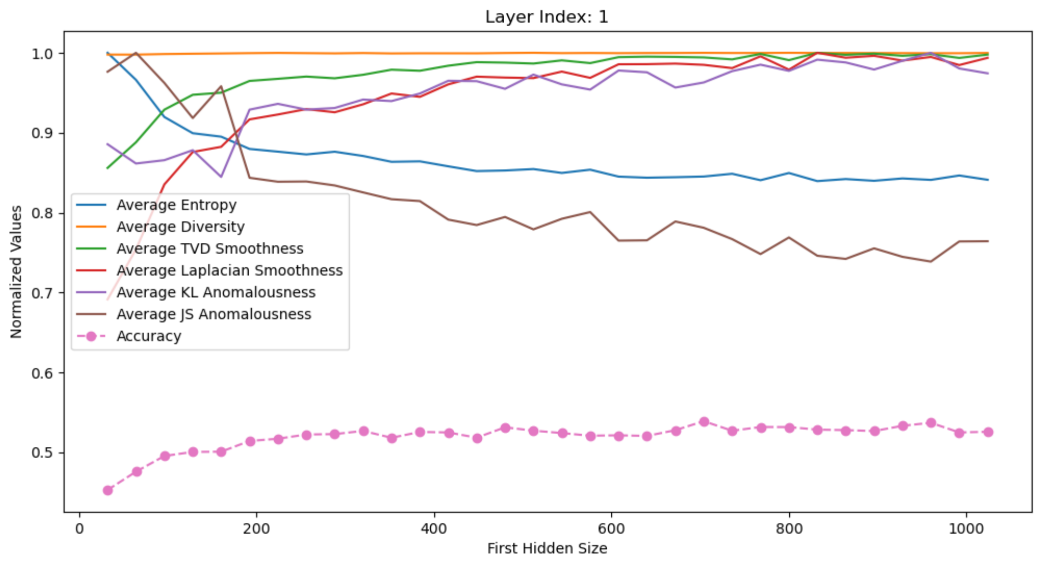


Figure 7.7

Layer Values for Different Metrics

3. Within specific layers, the impact of neuron removal varies, indicating that not all neurons contribute equally. (See Figures 7.8, 7.9, and 7.10).

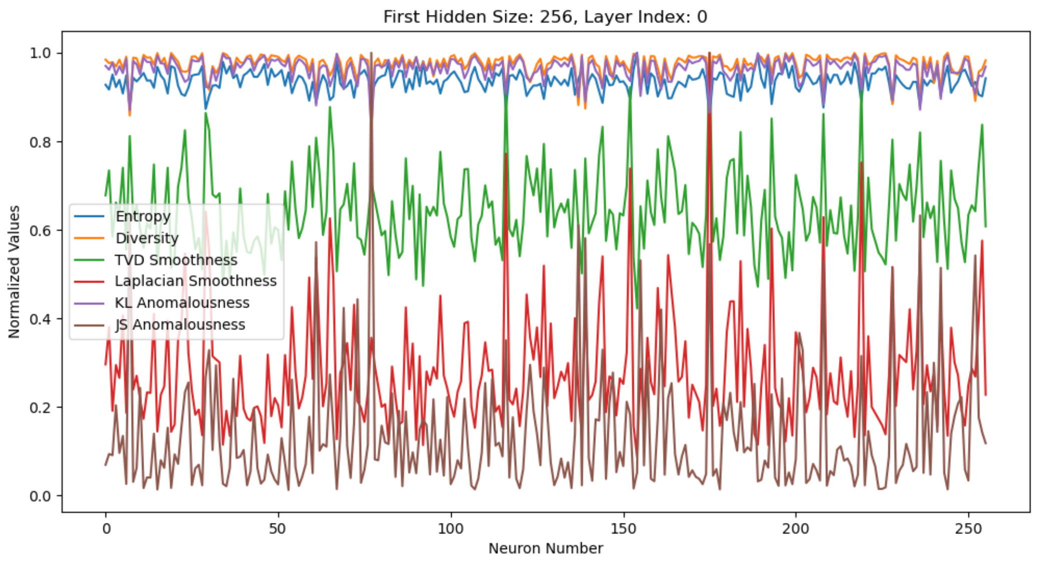


Figure 7.8

Normalized Metrics vs. Neuron No for Layer 0

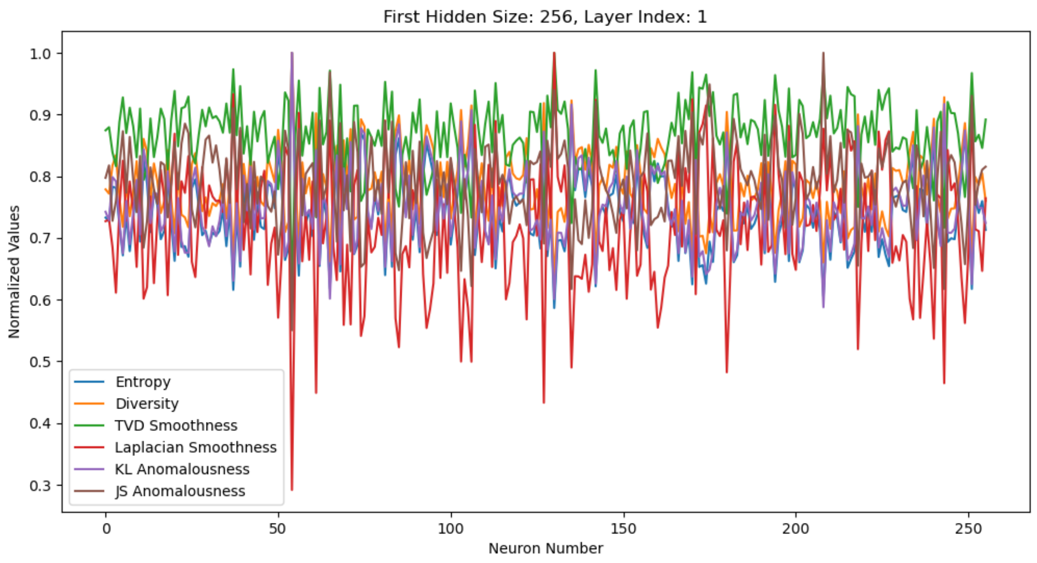


Figure 7.9

Normalized Metrics vs. Neuron No for Layer 1

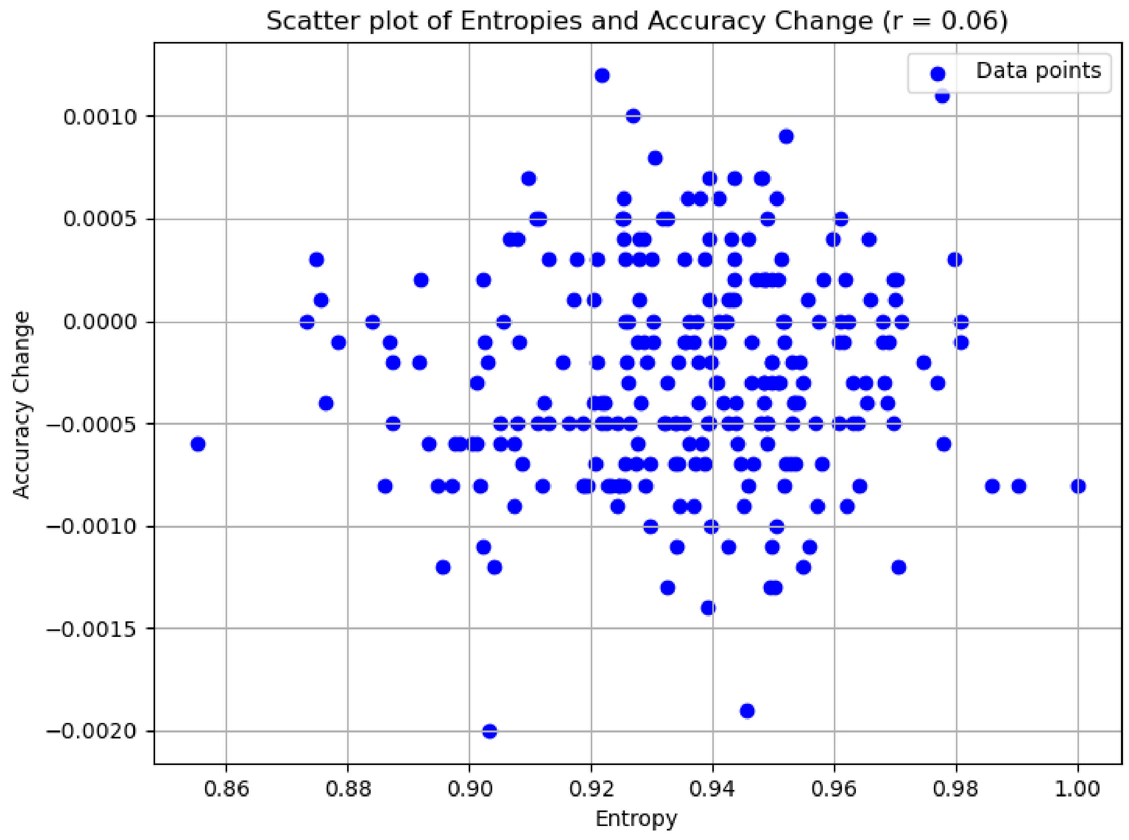


Figure 7.10

Entropy vs. Accuracy Correlation for Layer 0

4. Different layers exhibit different values for the calculated metrics. (See Figure 7.11).

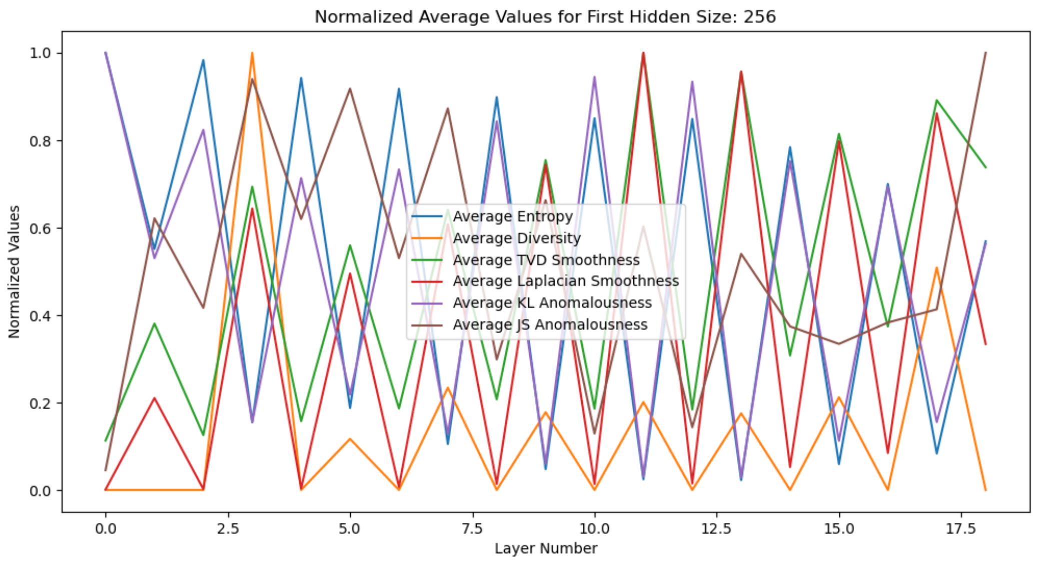


Figure 7.11

Average Values of Metrics for Different Layers

5. In some layers, the most significant accuracy changes occur(Removal always reduces the accuracy) when neurons with lower entropy are removed. (See Figures 7.12 and 7.13).

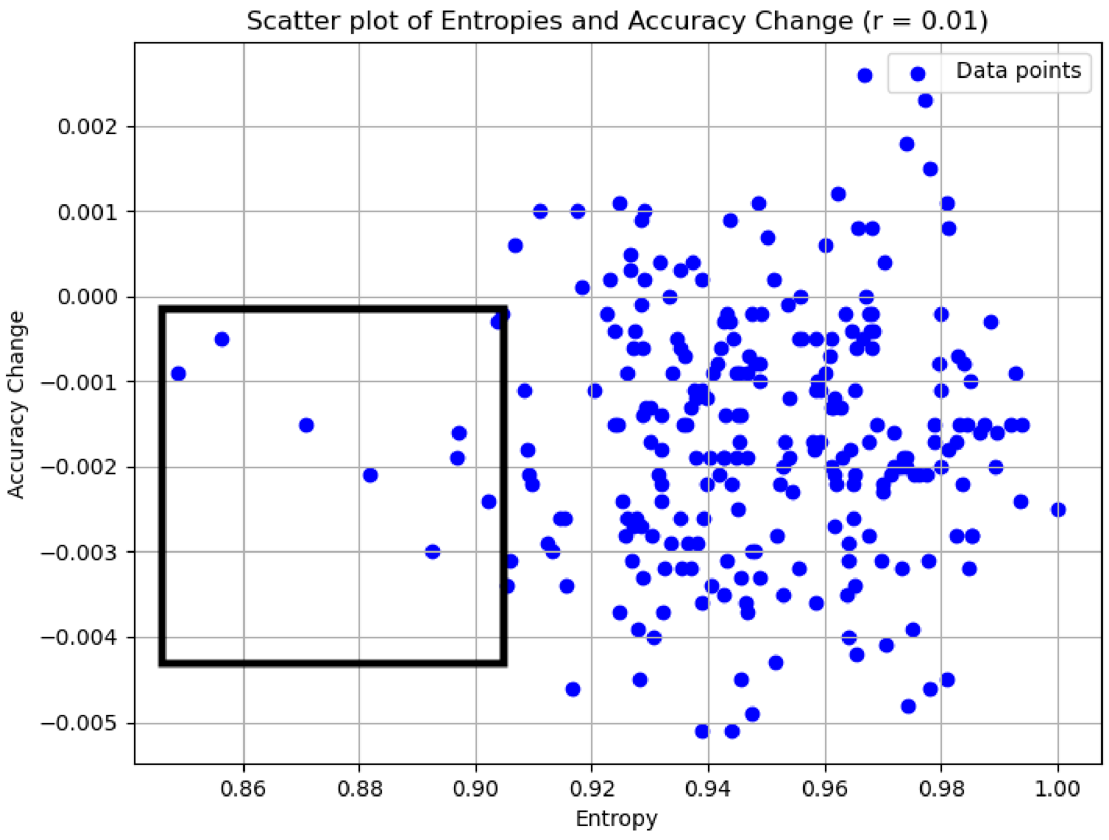


Figure 7.12

Entropy vs. Accuracy Correlation for Layer 2

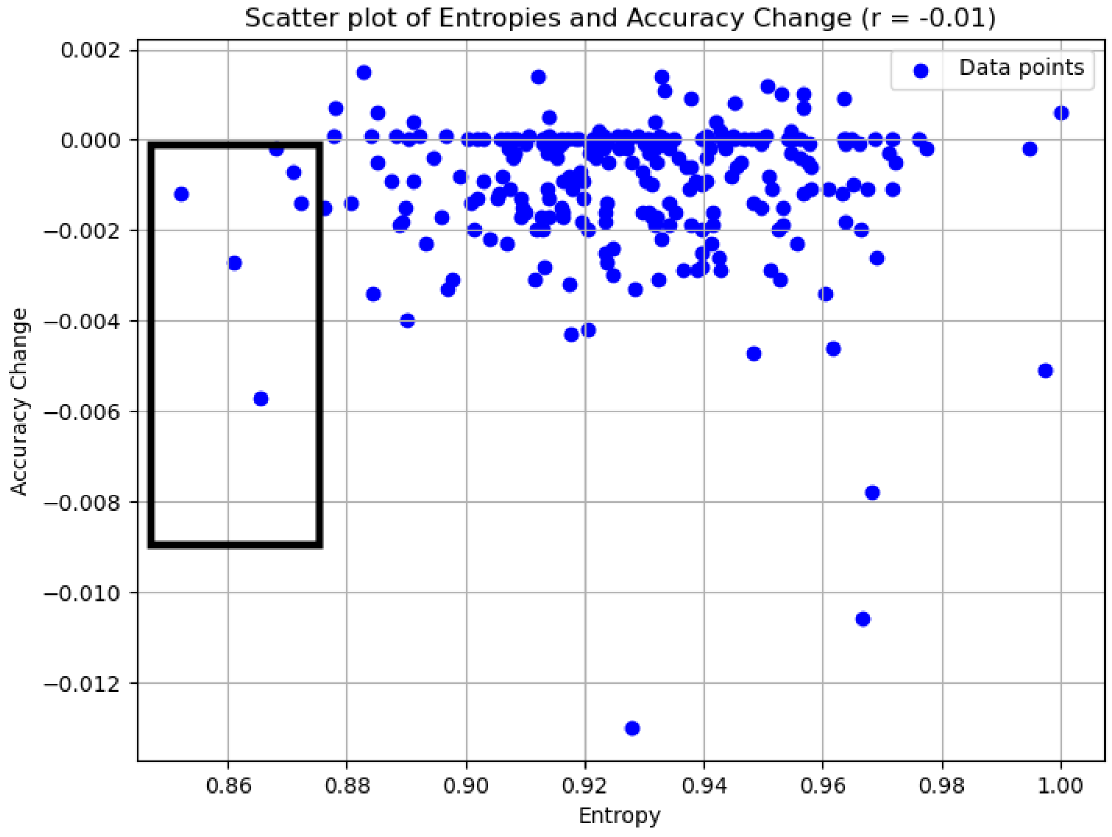


Figure 7.13

Entropy vs. Accuracy Correlation for Layer 4

6. In other layers, removing neurons with higher Laplacian smoothness results in significant accuracy changes(Removal always reduces the accuracy). (See Figures 7.14 and 7.15).

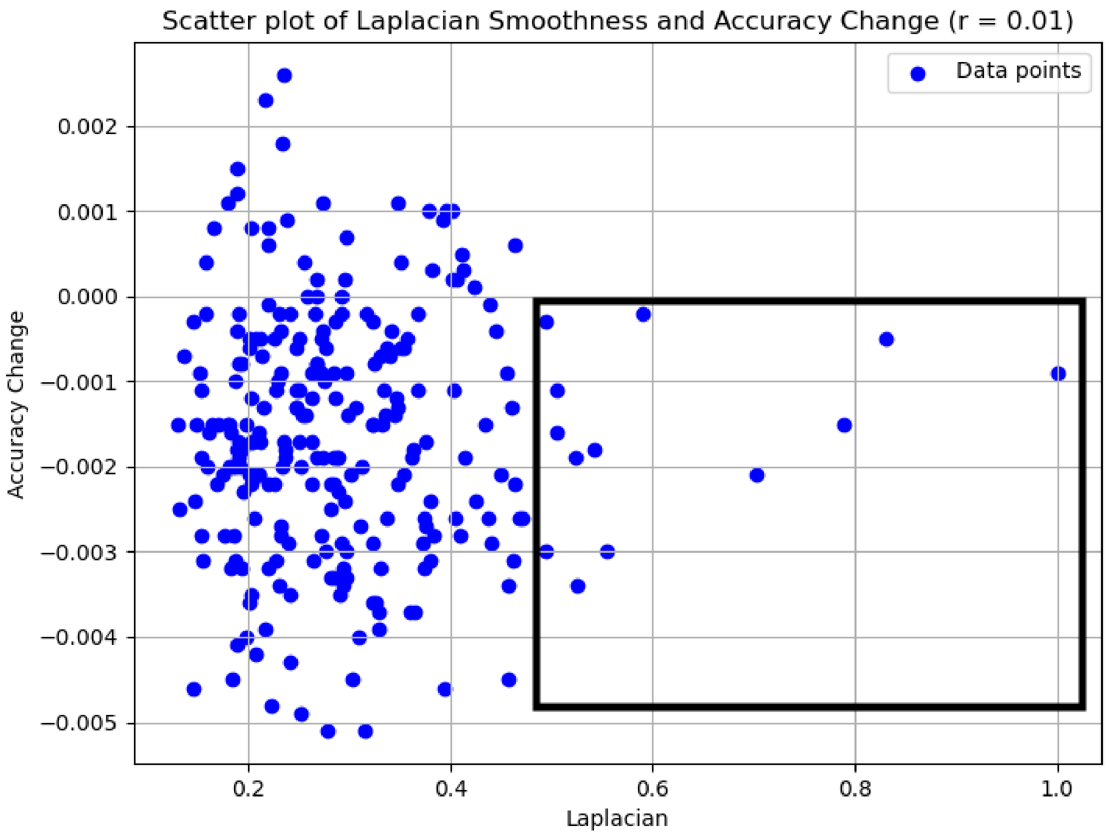


Figure 7.14

Laplacian Smoothness vs. Accuracy Correlation for Layer 2

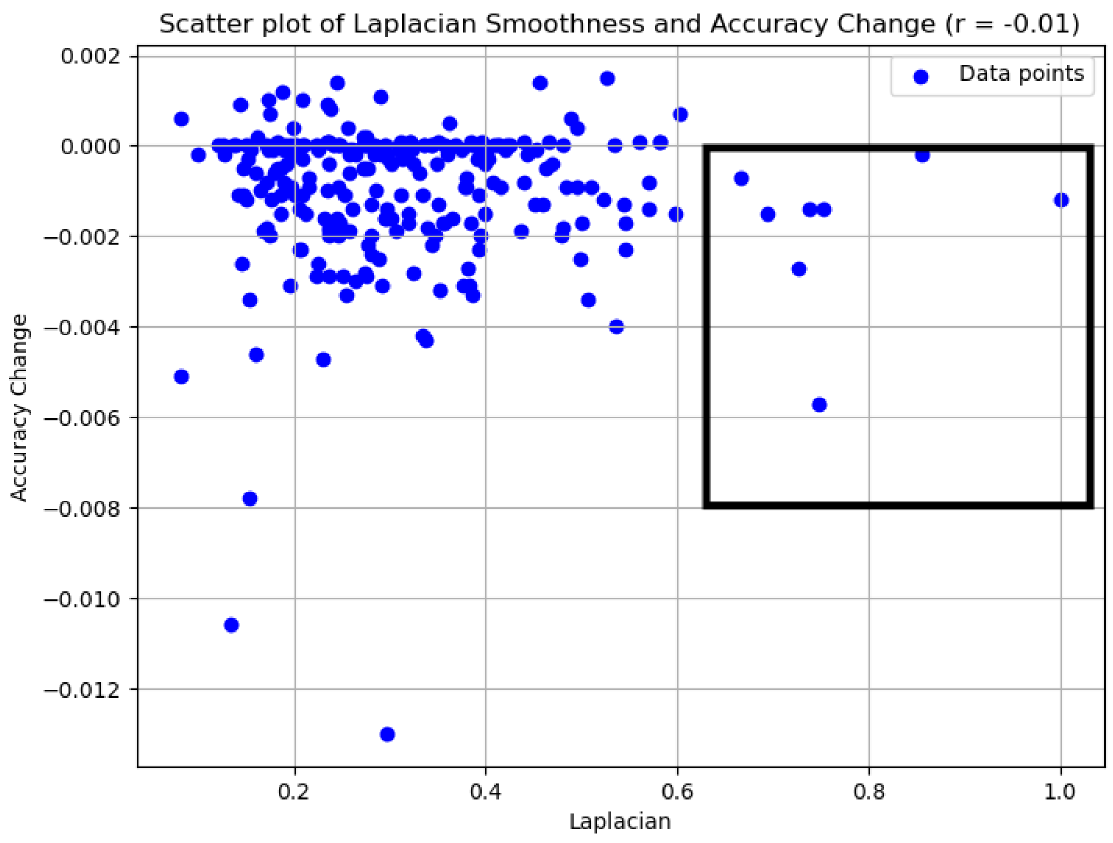


Figure 7.15

Laplacian Smoothness vs. Accuracy Correlation for Layer 4

7. Neurons with lower JS divergence tend to cause significant accuracy changes(Removal reduces the accuracy) when removed. (See Figures 7.16, 7.17, 7.18, 7.19, and 7.20).

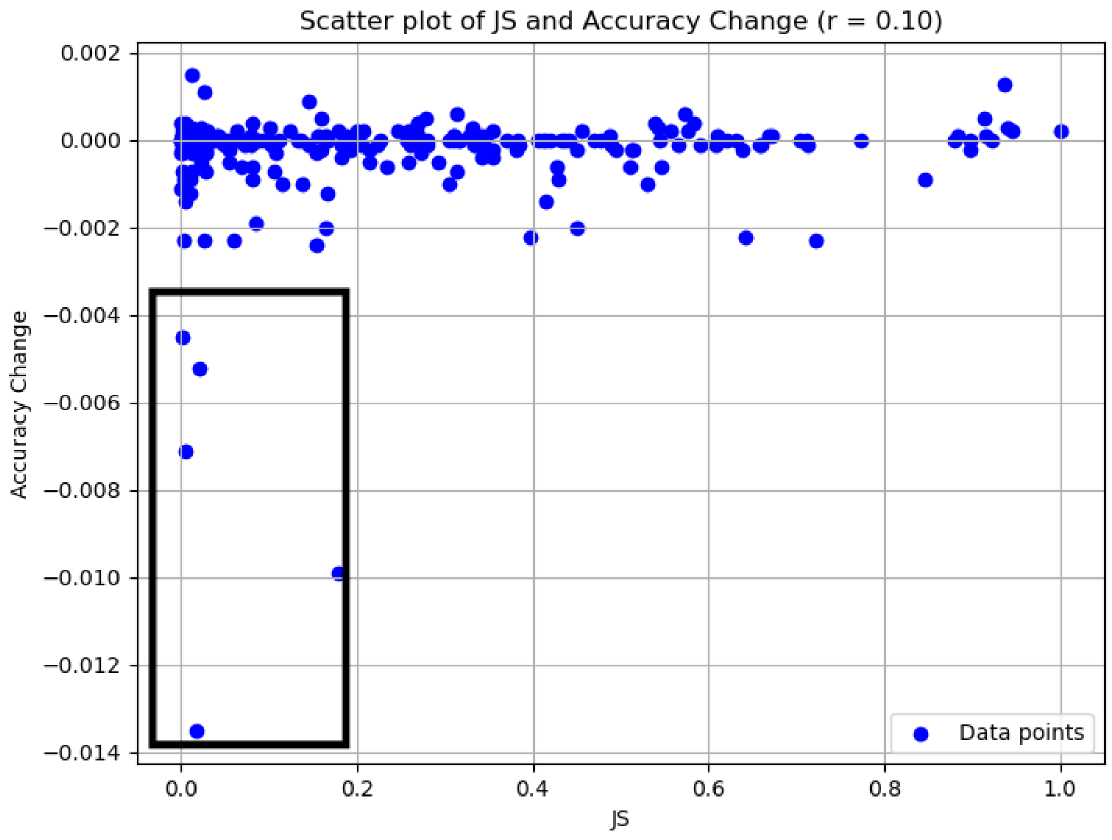


Figure 7.16

JS Divergence vs. Accuracy Correlation for Layer 8

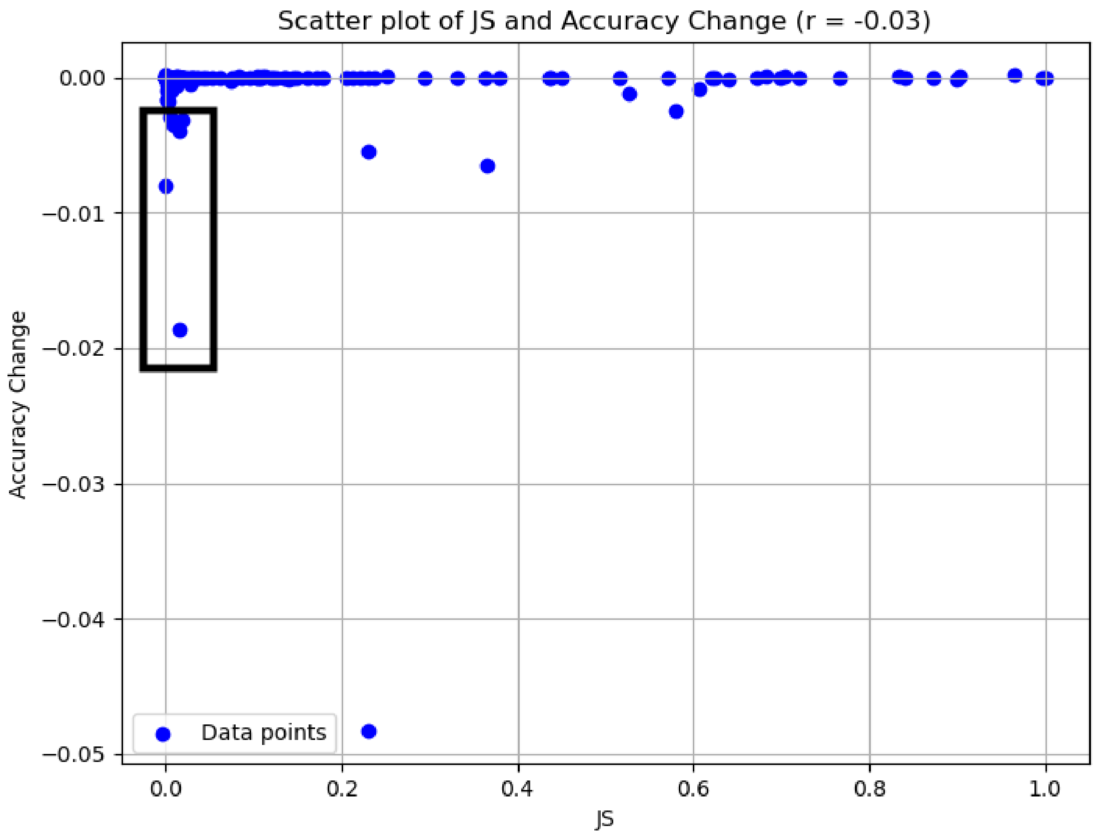


Figure 7.17

JS Divergence vs. Accuracy Correlation for Layer 10

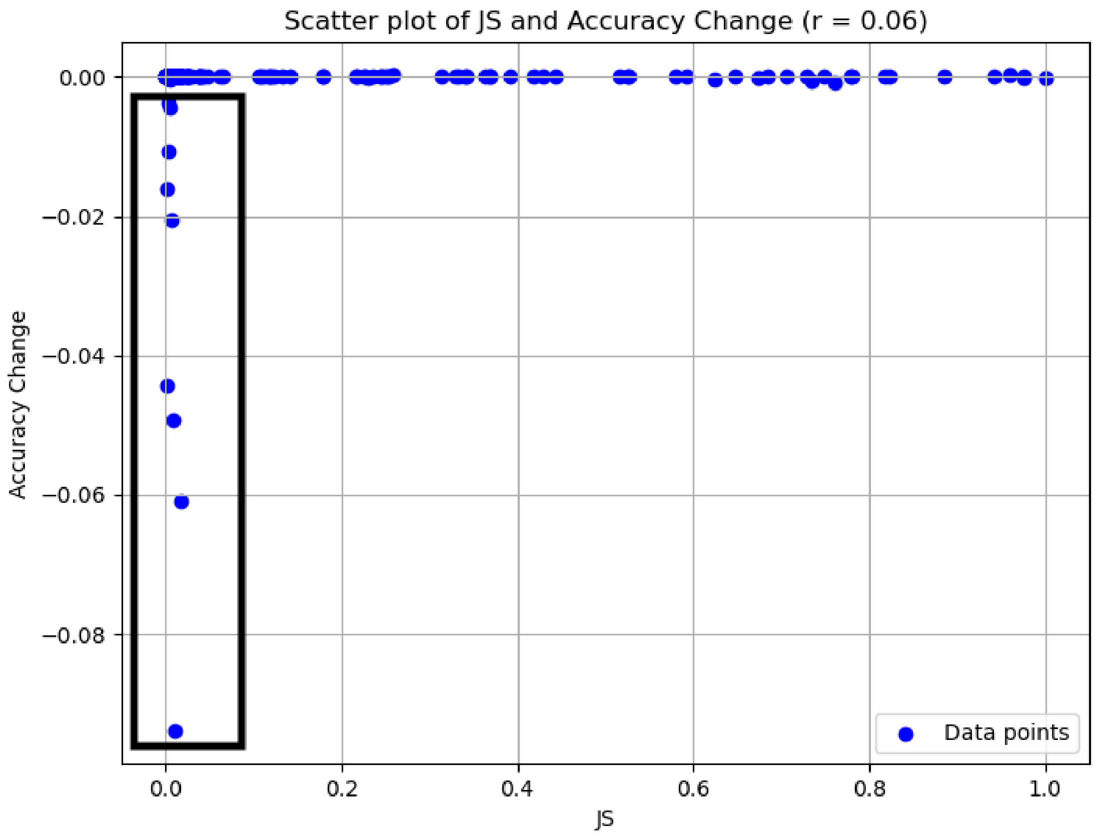


Figure 7.18

JS Divergence vs. Accuracy Correlation for Layer 12

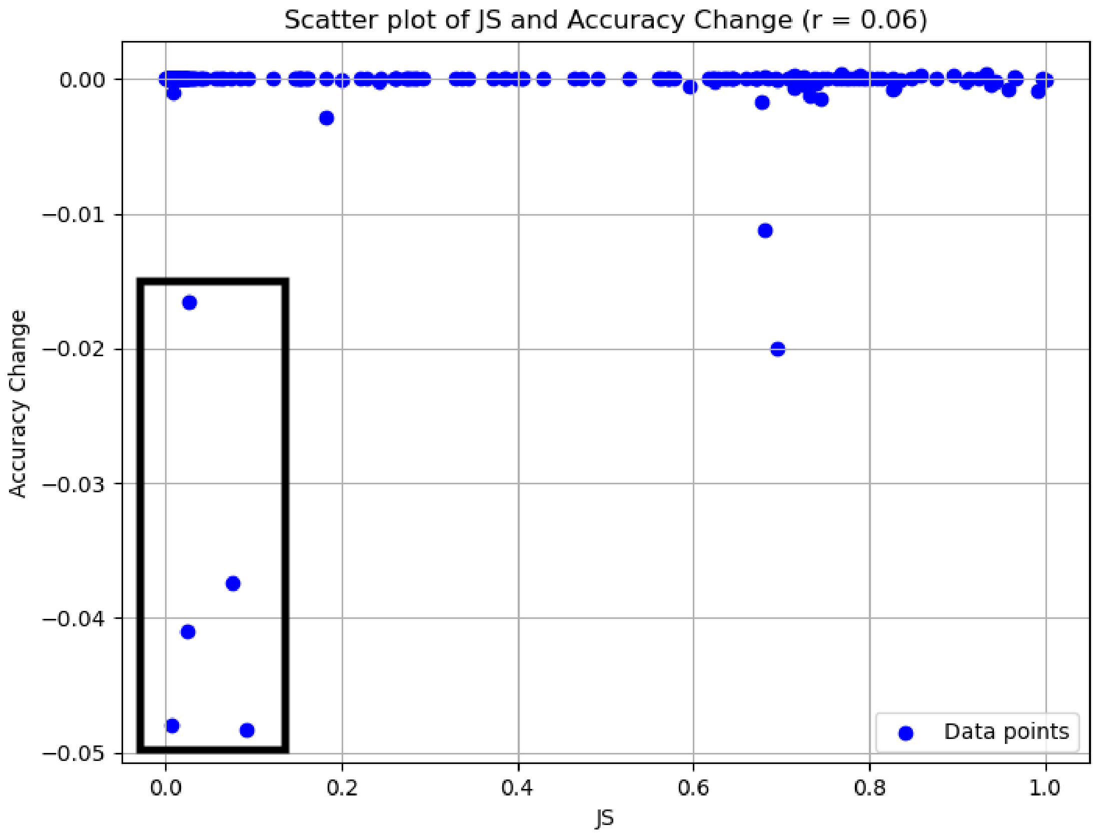


Figure 7.19

JS Divergence vs. Accuracy Correlation for Layer 14

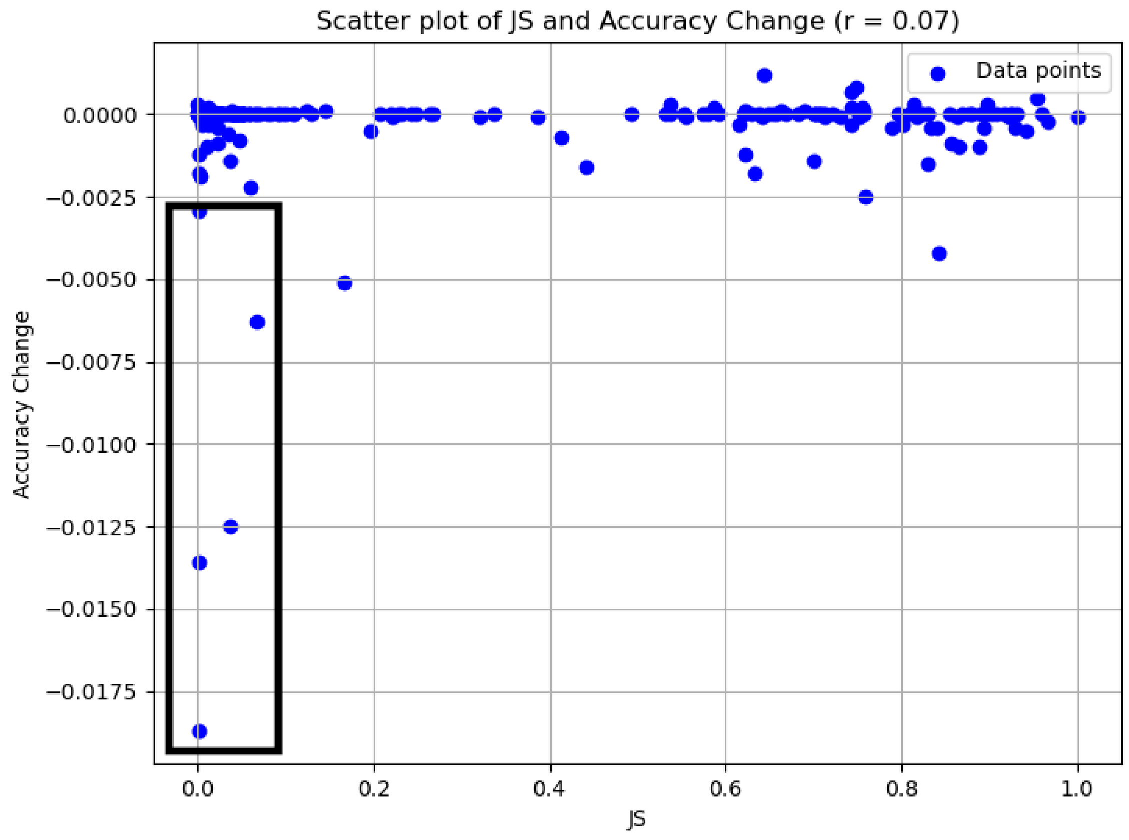


Figure 7.20

JS Divergence vs. Accuracy Correlation for Layer 16

7.7 Discussion

7.7.1 Theoretical and Practical Implications

The integration of 6-Set Discrete Probability Algebra into pruning techniques has significant theoretical and practical implications. The algebraic approach provides a robust framework for understanding the contribution of each neuron to the overall network performance. This understanding can lead to more informed decisions during the pruning process, enhancing the efficiency and effectiveness of neural network training and deployment.

7.7.2 Future Research Opportunities

Future research should focus on the following areas:

1. Extending the neuron selection process to include more complex network architectures and larger datasets.

2. Exploring the impact of different divergence metrics and their combinations on pruning effectiveness.

3. Investigating the application of 6-Set Algebra in other areas of neural network optimization beyond pruning, such as layer selection and network architecture search.

7.8 Conclusion

7.8.1 Chapter Summary

This chapter showcased the application of the 6-Set Discrete Probability Algebra in developing innovative pruning techniques for neural networks. By utilizing advanced algebraic metrics, we effectively pruned neurons based on their contribution to model performance, resulting in network architectures that are both more efficient and effective. These results underscore the significant potential of integrating sophisticated mathematical frameworks into neural network optimization. This approach not only streamlines the neural networks but also enhances their performance, setting a foundation for future innovations in the field.

CHAPTER 8

APPLICATION OF THE UNIFIED ALGEBRAIC FRAMEWORK IN INTEPRETABILITY AND EXPLAINABILITY OF DEEP LEARNING

8.1 Abstract

This chapter analyzes the neural network vertically(different layers) and horizontally(different neurons in the same layer) with the help of the Unified Algebraic Framework. Horizontal delving uses the changing amount of information(entropy) from the lower layer to the deeper layer to explain how the process of evaluation begins from the information-collecting process to the logical inference process and where is the bottleneck that shows the turning point. It also does some research on how the diversity and entropy of the neurons of a specific layer evolve during the training process of the neuron network epoch by epoch. We try to use the neuron-level analysis to improve the interpretability and transparency of understanding the neural network. This can partly help to break down the limitations of the black box that comes from our usual understanding of neural networks.

8.2 Introduction

8.2.1 Context and Importance

As artificial intelligence (AI), particularly deep neural networks becomes increasingly integral to critical technological and societal domains, the need for transparency and interpretability in these systems grows. The opaque 'black box' nature of many AI models, especially those based on deep learning, poses significant challenges. Users and regulators of AI technology often demand a clear understanding of how decisions are made by these models to ensure fairness, accountability, and safety.

Algebraic methods, especially through frameworks like the 6-Set Discrete Probability Algebra, offer substantial potential to address these challenges. These methods provide a mathematical foundation for dissecting and analyzing the information flow within neural networks, allowing for a quantifiable understanding of how data is processed across layers and among neurons, thus enhancing network interpretability and elucidating model decisions in understandable terms.

8.2.2 Objectives

This chapter aims to:

- **Apply the 6-Set Discrete Probability Algebra** to analyze neural network operations both vertically (across layers) and horizontally (across neurons within the same layer).

- **Utilize entropy measures** to track the evolution of information through the network, identifying critical information processing stages and bottlenecks.

- **Examine changes in the diversity and entropy of neurons** within specific layers during training to gain insights into the dynamics of learning and decision-making processes within the network.

- **Enhance neural network interpretability and transparency** by detailing internal mechanisms, addressing aspects of the 'black box' nature of these systems.

8.3 Literature Review

8.3.1 Current Research

Research on AI interpretability and explainability has largely depended on post hoc techniques that are meant to explain decisions made by models after they have been trained. These methods, including feature importance scores, saliency maps, and decision trees, strive to mimic the decision-making processes of neural networks. However, they frequently fail to uncover the deep complexities and inherent opacity of these models.

Feature importance scores evaluate the impact of each input feature on the model's predictions, identifying the most influential features. Techniques like SHAP (SHapley Additive exPlanations) consolidate various interpretability methods into a unified framework that offers consistent and precise feature attributions [91].

Saliency maps visually highlight areas within the input space—like pixels in an image—that significantly affect the model's output. Methods such as Grad-CAM (Gradient-weighted Class Activation Mapping) provide visual explanations for the decisions of convolutional neural networks by emphasizing input regions crucial to the model's predictions [124].

Decision trees simplify more complex models by generating straightforward, rule-based representations of their decision-making processes, thereby making the complex decision boundaries of deep neural networks more comprehensible.

Despite their usefulness, these post hoc methods often offer only limited insights because they do not change the fundamental opacity of the models. They provide a superficial understanding without probing the underlying mechanisms that drive model behavior [115] [128] [131].

Recent developments in models inherently designed for interpretability aim to provide transparency from the beginning. These models, designed with interpretability as a core feature, integrate mechanisms that allow for direct understanding, often at the cost of some performance. This marks a significant shift towards embedding transparency directly into AI systems from their inception [98] [88].

Incorporating algebraic methods into neural network analysis offers a balanced approach, yielding profound insights into traditional models without compromising their effectiveness. These algebraic methods, particularly those employing the 6-Set Discrete Probability Algebra, facilitate a detailed, quantitative analysis of information flow within networks. This enables a deeper understanding of model behaviors by quantifying how information is processed and transformed across various layers and among neurons within those layers [30] [116].

For example, using measures such as entropy can reveal how information evolves from the initial input layers (information gathering) to deeper layers (logical inference), identifying critical bottlenecks [128]. Methods like SmoothGrad enhance gradient-based interpretability techniques by adding noise to input samples, producing smoother, more informative visualizations of model sensitivity [131].

Additionally, these algebraic approaches can be expanded to analyze the diversity and entropy of neurons within specific layers during training, shedding light on the dynamics of learning and decision-making within the network. Understanding these dynamics can improve the interpretability and transparency of neural networks, thus addressing some of the limitations of their 'black box' nature [135] [124].

In summary, while traditional post hoc interpretability methods provide valuable insights into neural network behavior, they often fall short of delivering a comprehensive understanding. Recent advances in inherently interpretable models and the application of algebraic methods offer promising pathways to deeper transparency and explainability in AI systems. These strategies not only enhance our understanding of complex models but also foster the development of more reliable and accountable AI technologies.

8.4 Background Information

8.4.1 Theoretical Basis

The rationale for enhanced interpretability and the push for greater interpretability in AI is driven by several critical factors:

- **Trust and Adoption:** For AI systems to be fully embraced, especially in sensitive areas like healthcare and autonomous driving, they must be trusted by users and regulators. Comprehending these systems' decision-making processes is essential for fostering this trust.

- **Ethical and Legal Compliance:** AI systems are required to operate within established ethical and legal frameworks. Interpretability helps ensure that these systems do not inadvertently propagate biases or make unjustifiable decisions that could result in discriminatory outcomes.

- **Debugging and Improvement:** Interpretable models enable developers to identify and correct unexpected behaviors, which is crucial for refining and enhancing AI systems.

The 6-Set Discrete Probability Algebra provides a structured approach to dissect neural networks by quantifying entropy and diversity in neuron outputs. This method allows for the decomposition of the network's decision-making into measurable elements, mapping how inputs are transformed into complex decisions. It not only facilitates the identification and understanding of network dynamics but also highlights where information bottlenecks might impair processing efficiency or accuracy.

By leveraging these algebraic techniques, this chapter seeks to pioneer new pathways for making AI systems more transparent and comprehensible, thereby enhancing their reliability and suitability for diverse applications.

8.5 Methodology

8.5.1 Implementation of Algebra

In this research, we integrate the 6-Set Discrete Probability Algebra within the Unified Algebraic Framework to enhance the interpretability of AI systems. We use various methods, including `UAM_kl_divergence` and `UAM_kl_divergence_PD` for KL Divergence, and

`UAM_shannon_entropy` and `UAM_shannon_entropy_PD` for Shannon Entropy. These methods enable us to calculate the diversity (Average Diversity using KL Divergence) between the outputs of a specific neuron and the outputs of all other neurons in the same layer.

We construct a neural network architecture with 10 linear layers, similar to the one used in Chapter 7, and train it using the CIFAR-10 dataset. During training, we calculate the Average Diversity and Average Shannon Entropies of the neurons in each layer and trace how these metrics change over epochs. After training, we calculate the Average Diversity and Average Shannon Entropies layer by layer. For each layer, we compute the correlation between the Normalized Average Diversity by neuron and the Normalized Average Shannon Entropy by neuron. Additionally, we remove neurons in each layer to measure the average accuracy change per layer.

8.5.2 Evaluation Methods

A Experimental Setup

The experimental framework involves the following steps:

1. Dataset: We use the CIFAR-10 dataset for training and validation.

2. Tools: The Unified Algebraic Framework is implemented in Python, and the neural network is constructed using a deep learning library such as TensorFlow or PyTorch.

3. Model Architecture: We construct a neural network with 10 linear layers.

4. Training: The model is trained using the CIFAR-10 dataset.

5. Metric Calculation: After training, we calculate diversity, entropy, TVD smoothness, Laplacian smoothness, KL divergence, and JS divergence for all neurons in each layer.

6. Pruning Process: Neurons are pruned one by one, and the change in model accuracy is recorded.

7. Performance Metrics: Pearson correlation is calculated between the accuracy change due to neuron removal and the metrics calculated for each neuron.

B Neural Network Architecture

The neural network includes 10 linear layers. (See Figure 8.1, For a more legible version, Please see Figure IntepreteNN in Appendix E or zoom in)

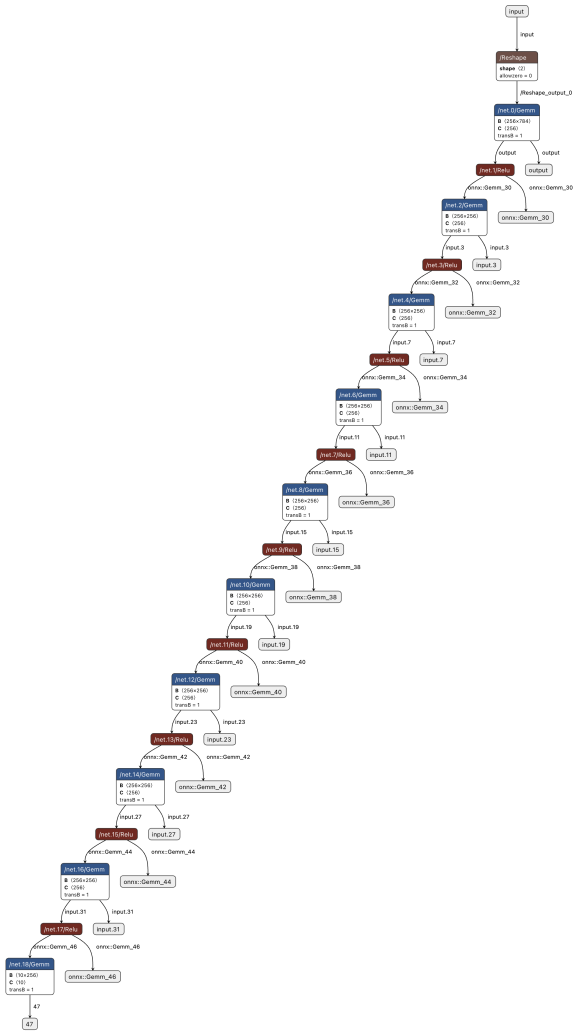


Figure 8.1

Neural Network Architecture

C Pseudocode for Calculating and Collecting Information

The pseudocode is as follows. (See Algorithm 8.14)

Algorithm 8.14 Calculating and Collecting Information

```
1: Train the neural network on the CIFAR-10 dataset
2: for each epoch do
3:     for each layer in the network do
4:         Calculate Average Diversity and Average Shannon Entropy for all neurons
5:     end for
6: end for
7: Calculate layer-wise Average Diversity and Shannon Entropy after training
8: for each layer in the network do
9:     Compute correlation between Normalized Average Diversity and Normalized Average Shannon Entropy
10:    for each neuron in the layer do
11:        Remove the neuron and record the change in model accuracy
12:        Restore the neuron
13:    end for
14: end for
```

8.6 Results

8.6.1 Results Interpretation

The results of applying 6-Set Algebra to AI systems reveal several insights:

A Three-Stage Explanation

In the neural network, even layers are linear transformation layers (GEMM - General Matrix Multiply) and odd layers are non-linear activation function layers (ReLU - Rectified Linear

Unit). (See Figures 8.2, 8.3, and 8.4) Stage One (Layers 0-10): Information increases with ReLU layers (Entropy reduces) but decreases with GEMM layers (Entropy increases). Stage Two (Layers 11-14): Information conversion from material type to inference type, with significant changes at Layers 11-12 and 13-14. Stage Three (Layers 15-19): Inference process, with information increasing along ReLU and GEMM layers (Entropy reduces).

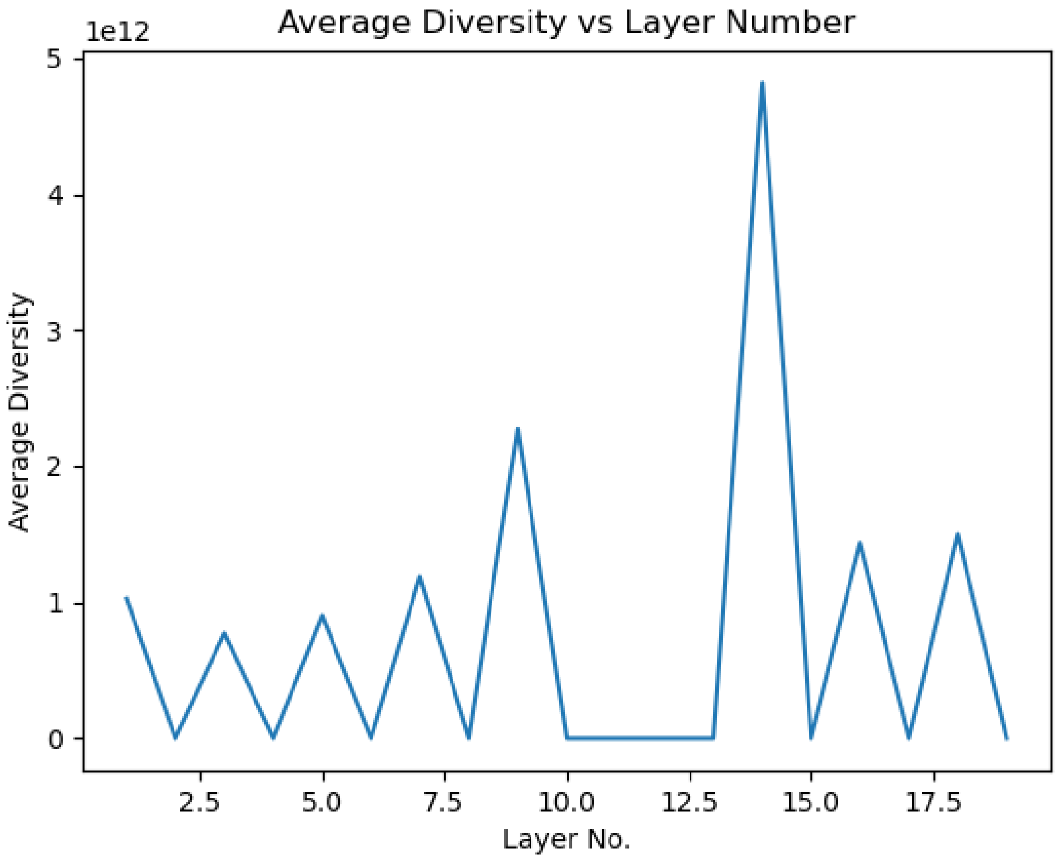


Figure 8.2

Average Diversity vs. Layer

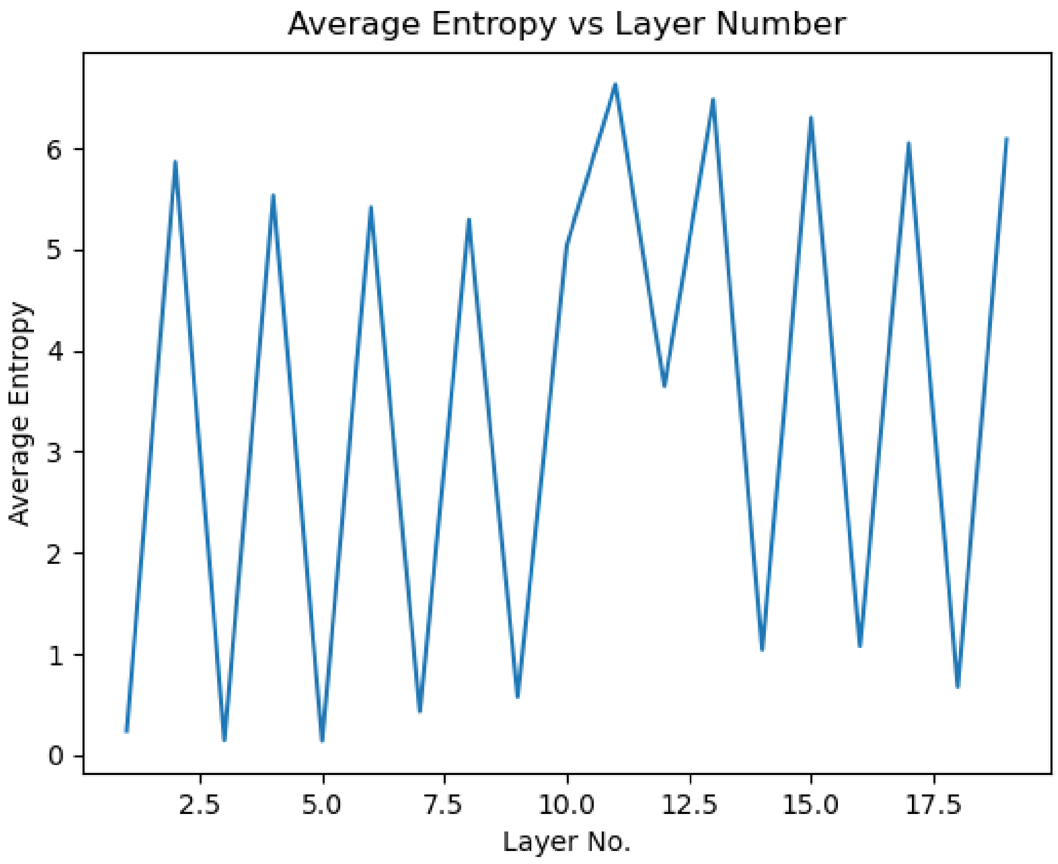


Figure 8.3

Average Entropy vs. Layer

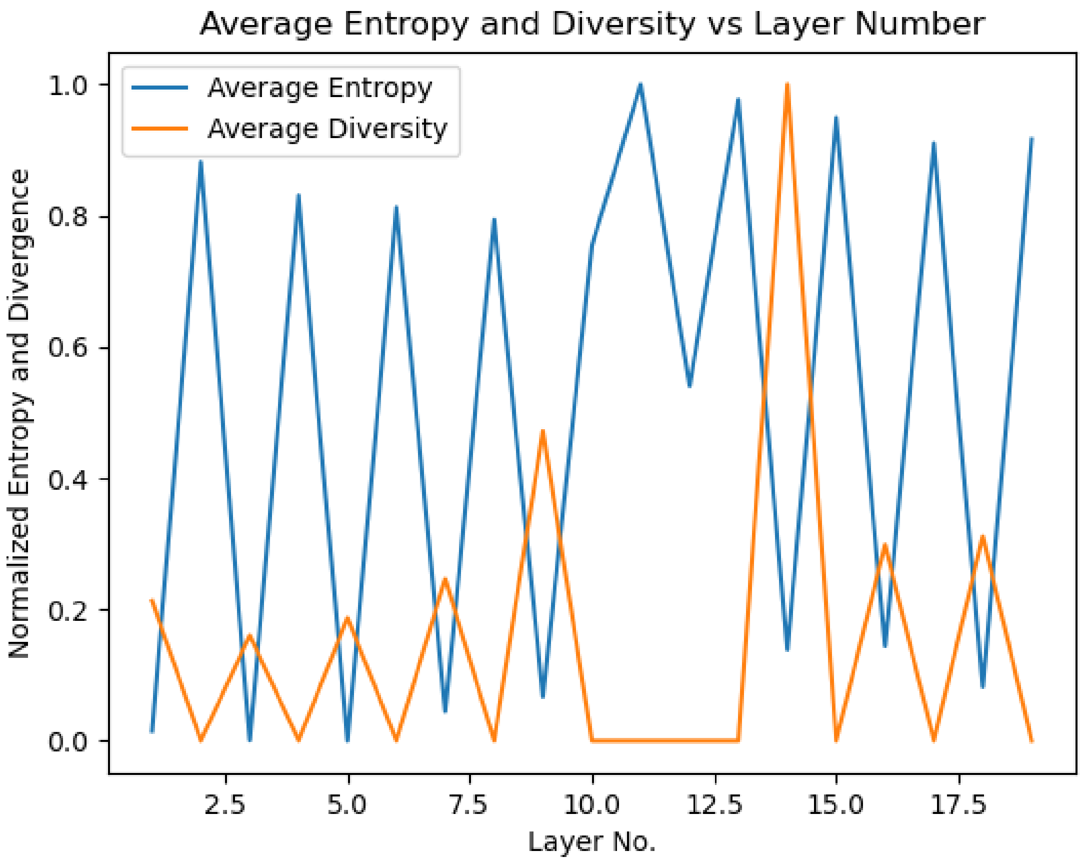


Figure 8.4

Average Entropy and Diversity vs. Layer

B Distillation Process

From GEMM (Summation) to ReLU layers, a process of distillation occurs, except for the turning point between Layers 10-11. (See Figures 8.5, 8.6, 8.7, and 8.8)

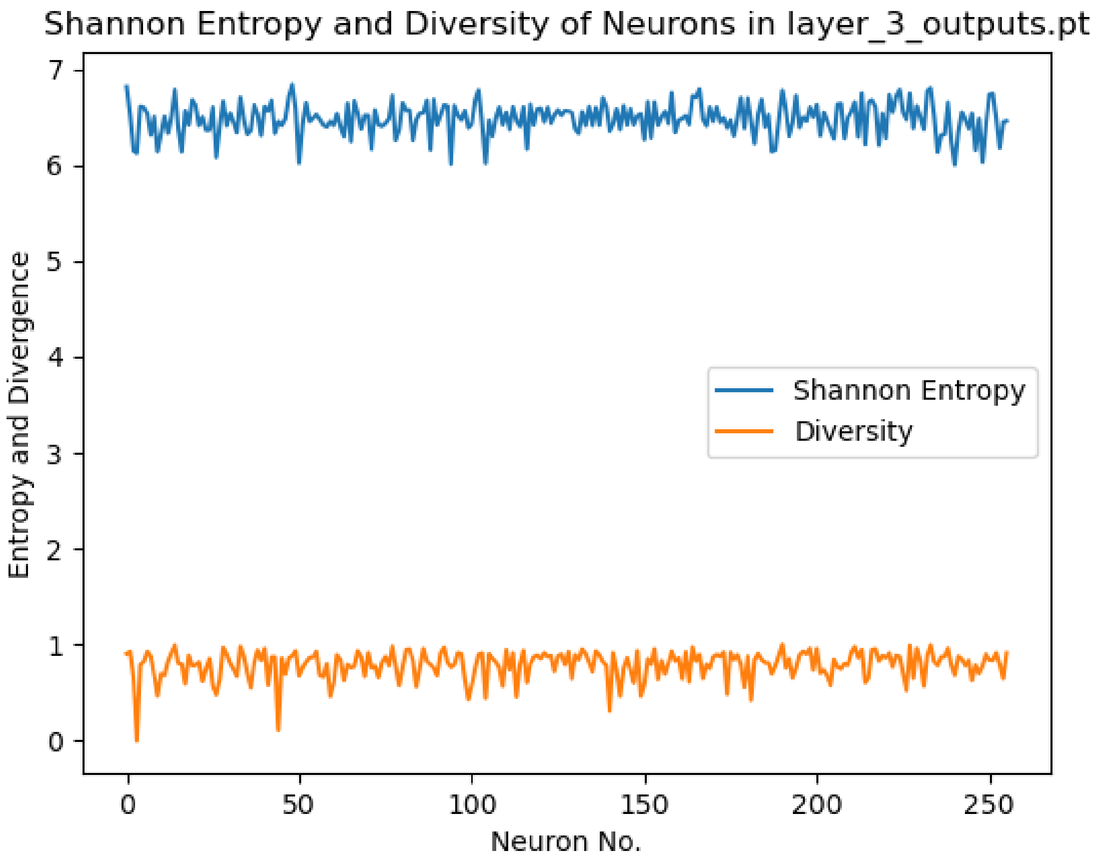


Figure 8.5

Entropy and Diversity for Layer 3

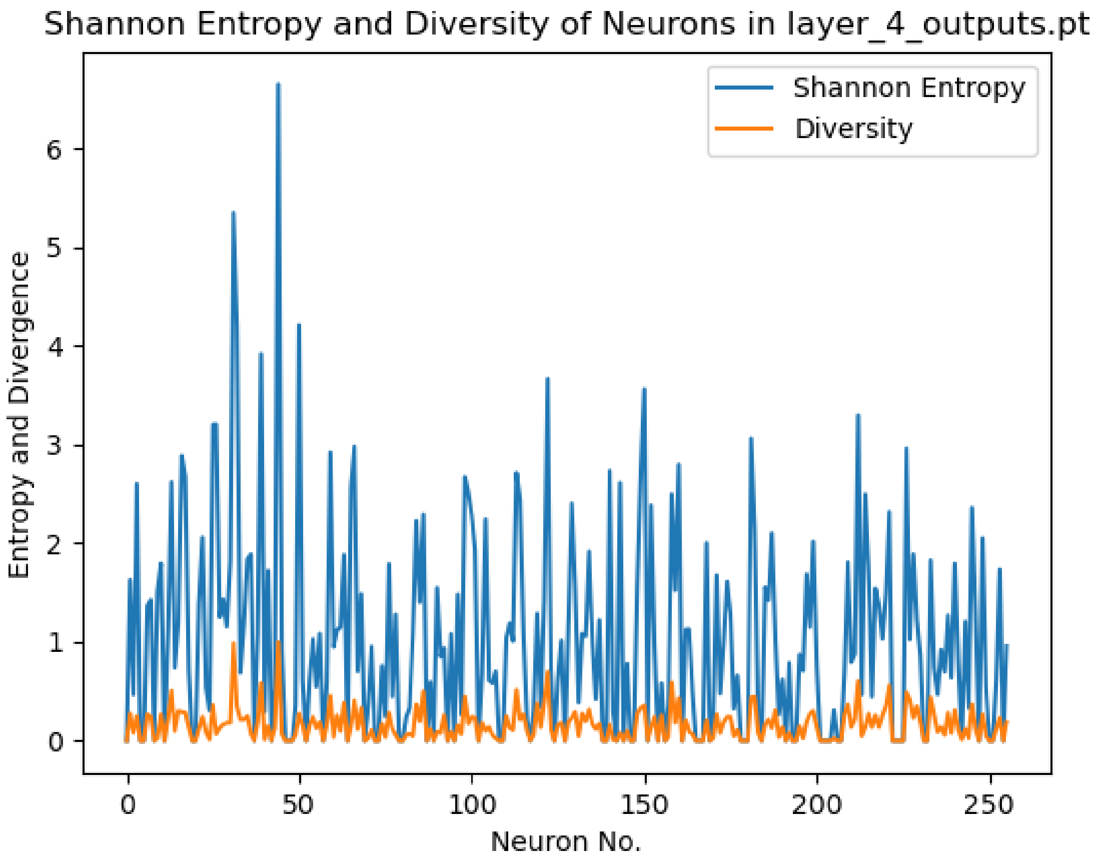


Figure 8.6

Entropy and Diversity for Layer 4

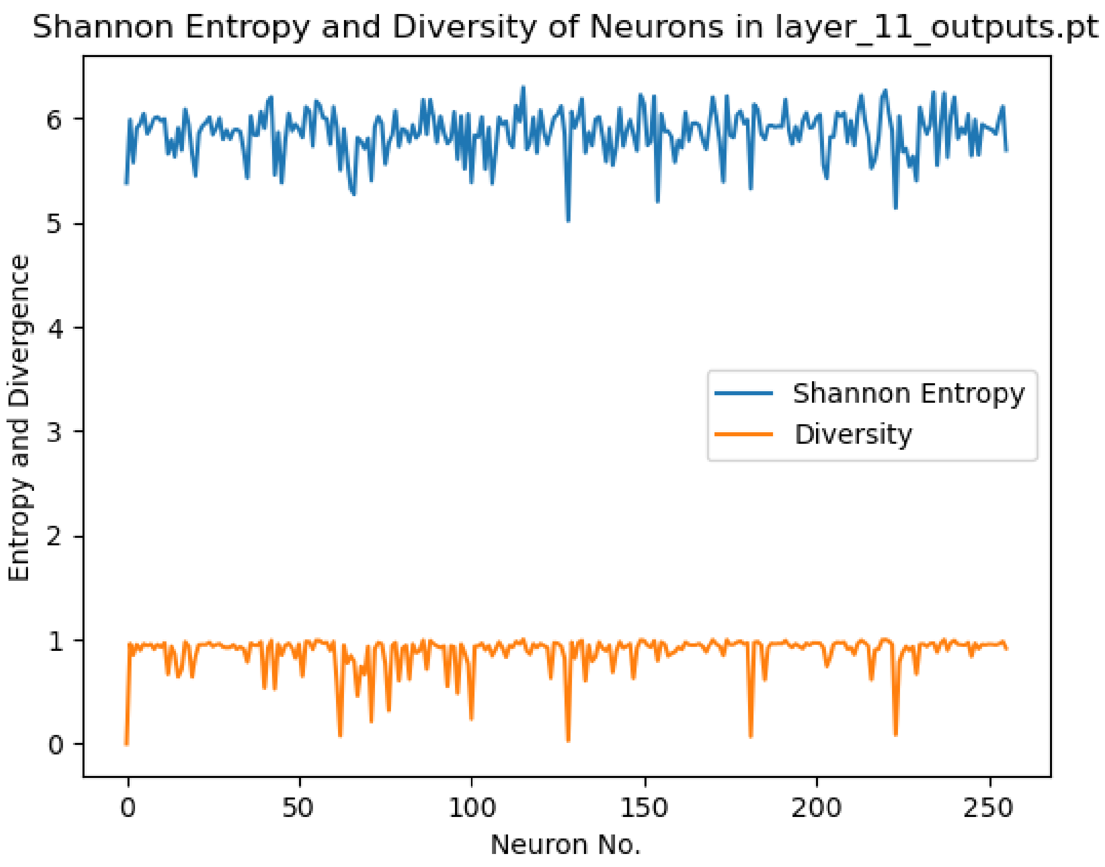


Figure 8.7

Entropy and Diversity for Layer 11

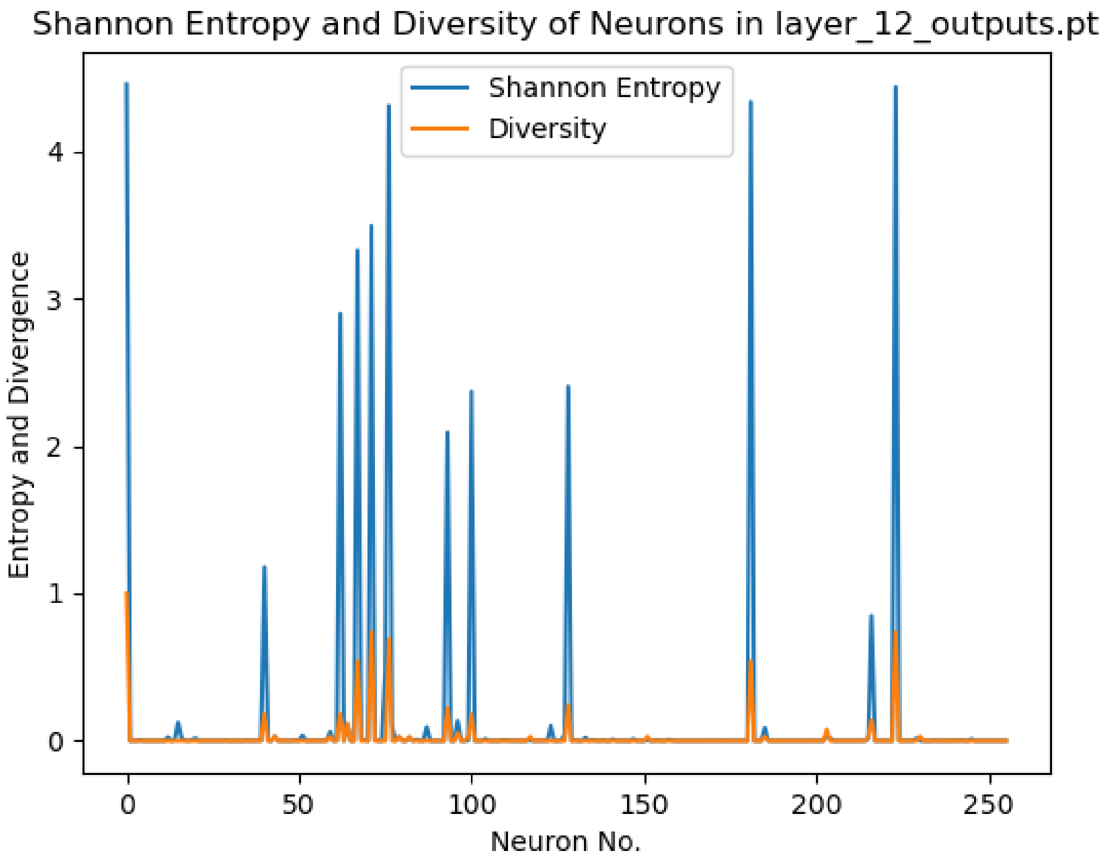


Figure 8.8

Entropy and Diversity for Layer 12

C Accuracy Change and Neuron Removal

At the beginning of Stages One and Three, neurons have the highest information density, and their removal causes significant accuracy loss. (See Figure 8.9)

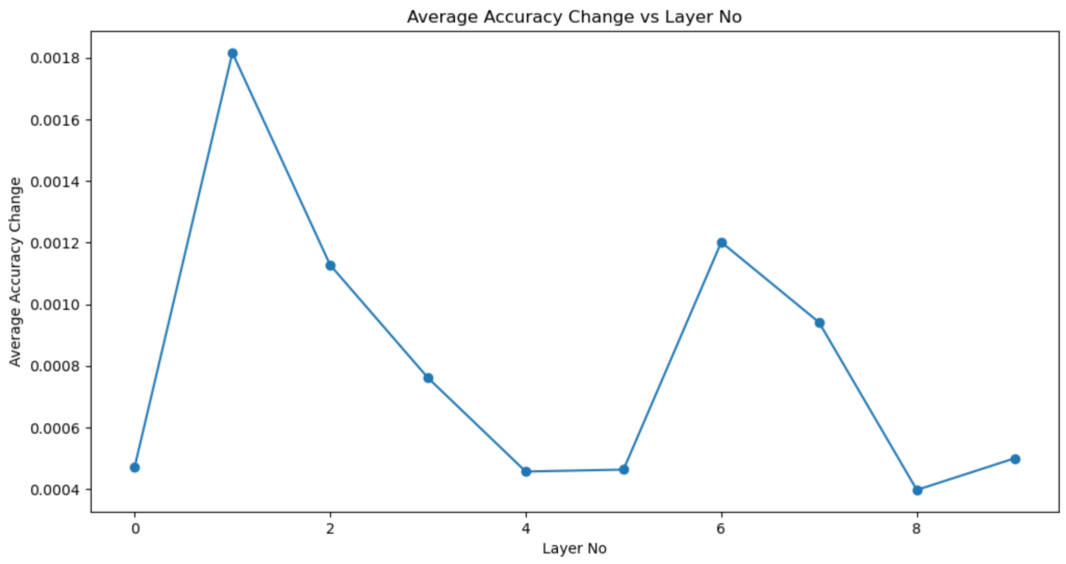


Figure 8.9

Accuracy Change by Layer Number

D Correlation in GEMM Layers

Average Entropy and Average Diversity have a weak correlation in GEMM layers. (See Figures 8.10, 8.11, 8.12, and 8.13)

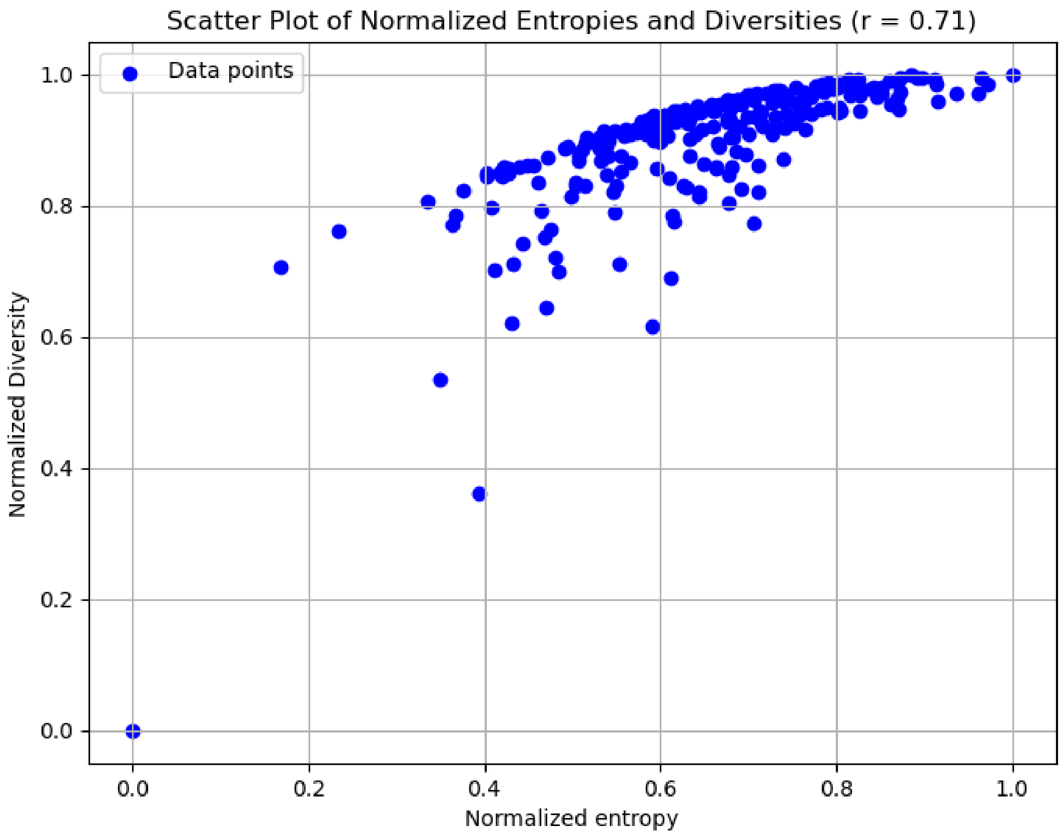


Figure 8.10

Correlation in Layer 1

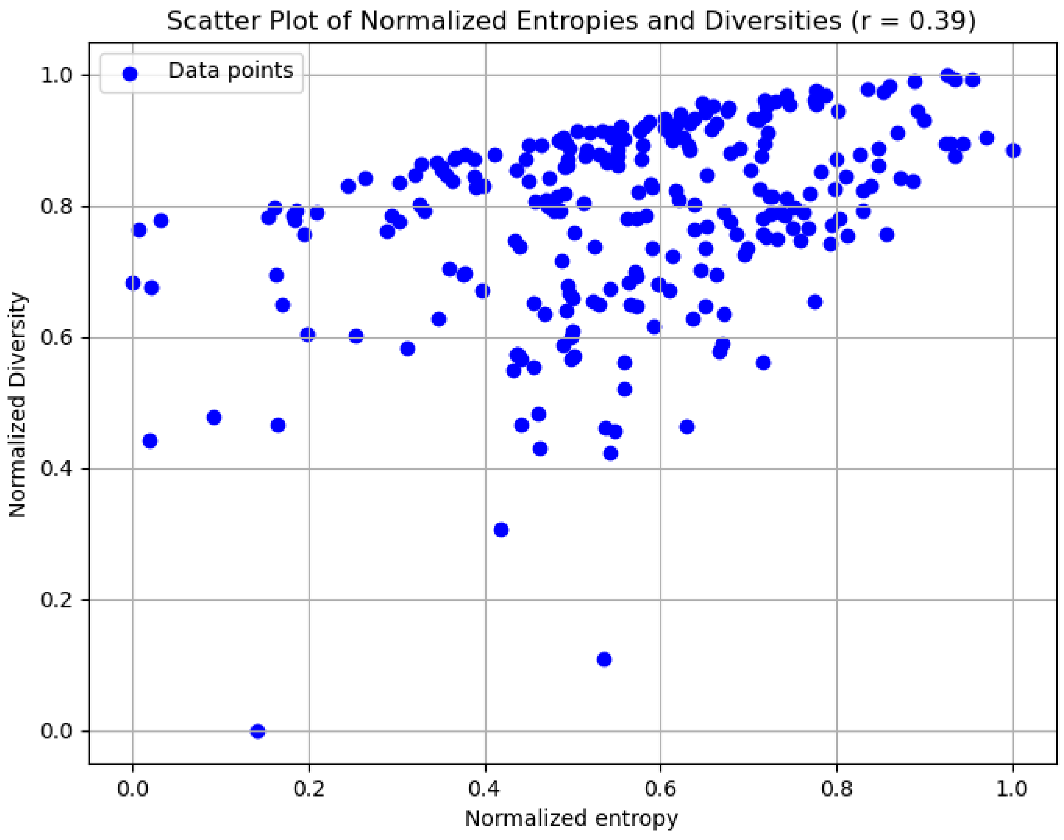


Figure 8.11

Correlation in Layer 3

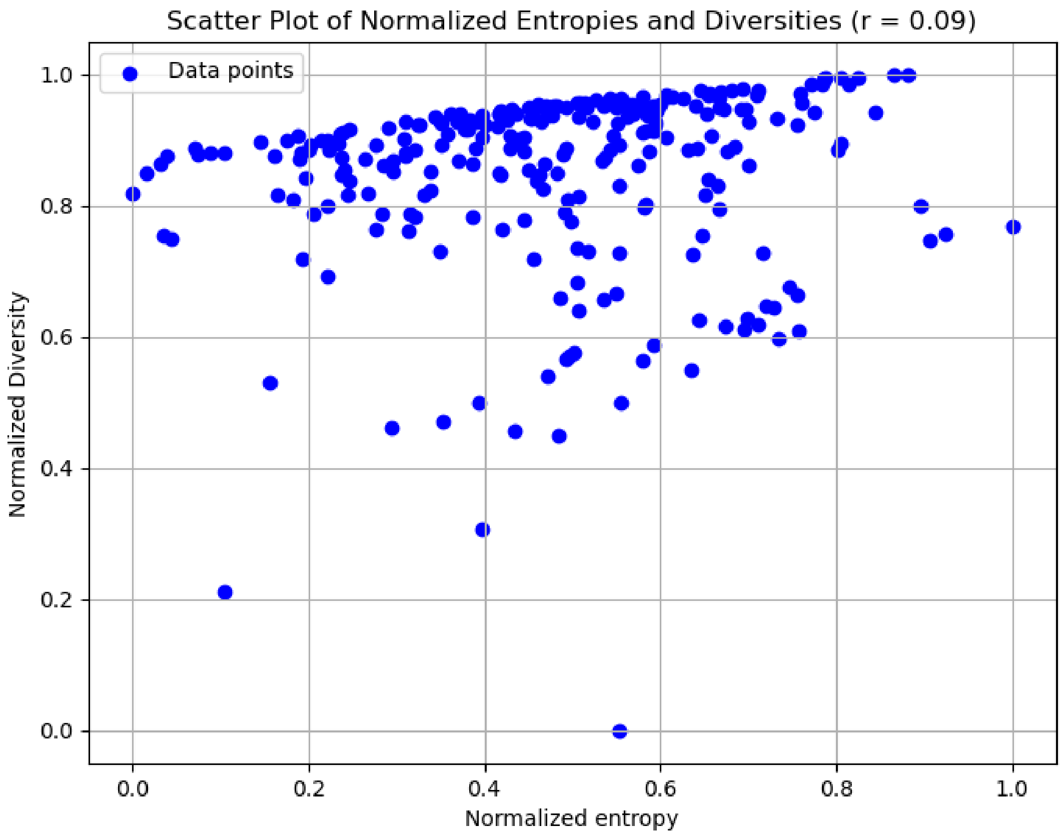


Figure 8.12

Correlation in Layer 5

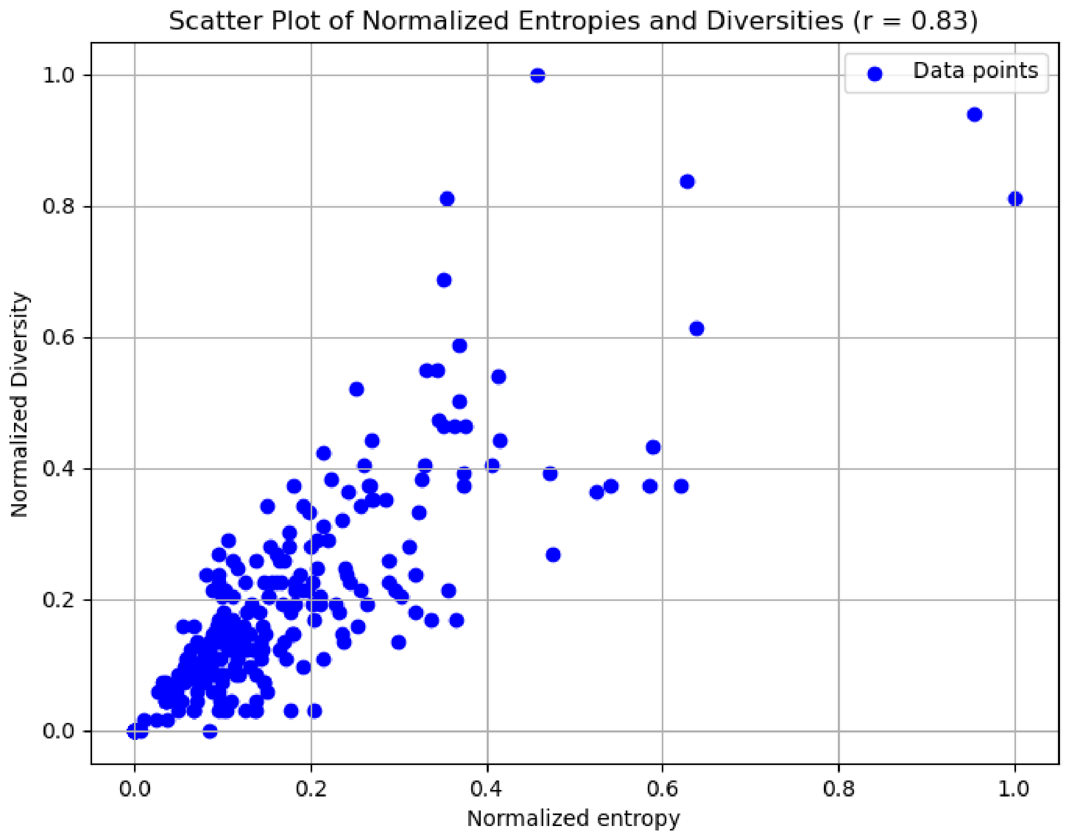


Figure 8.13

Correlation in Layer 6

E Correlation in ReLU Layers

Average Entropy and Average Diversity have a stronger correlation in ReLU layers. (See Figures 8.14, 8.15, 8.16, and 8.17)

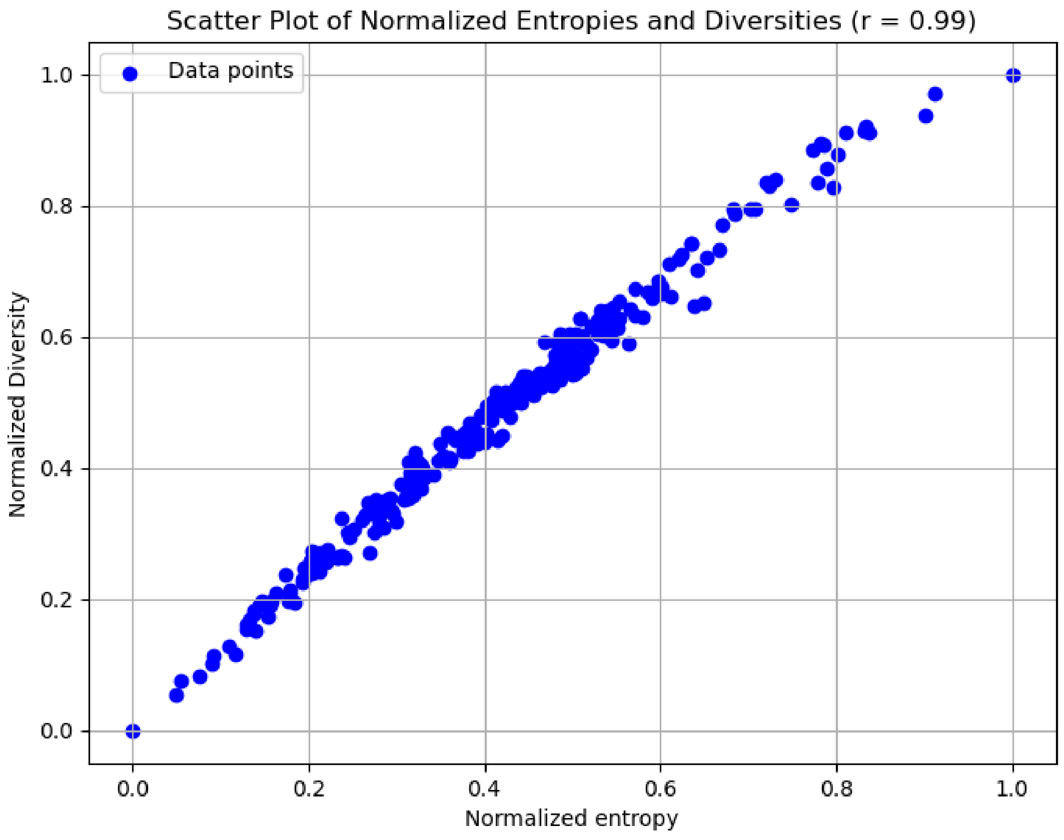


Figure 8.14

Correlation in Layer 2

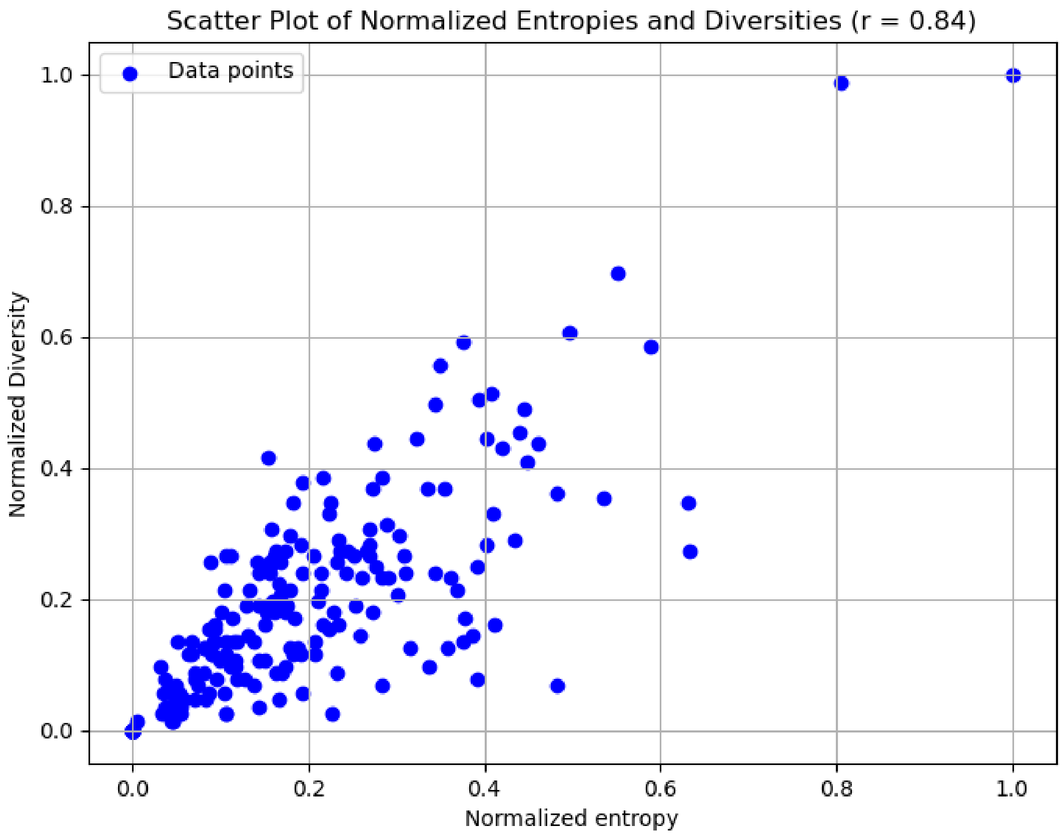


Figure 8.15

Correlation in Layer 4

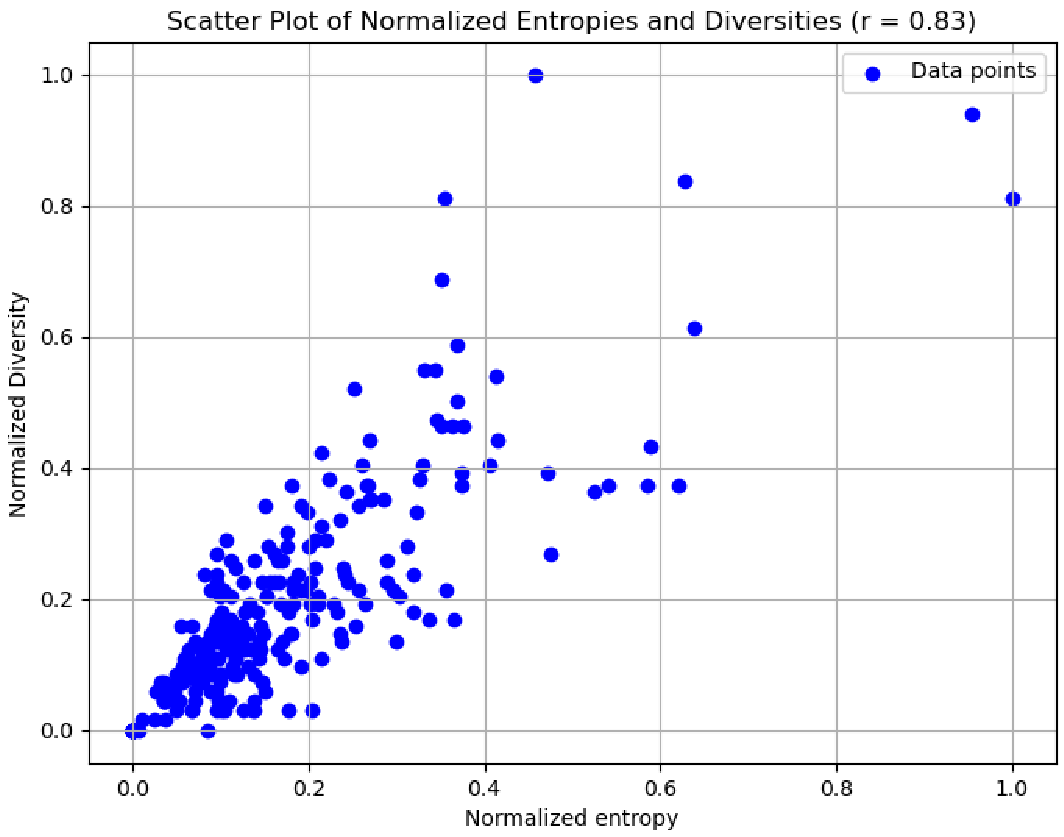


Figure 8.16

Correlation in Layer 6

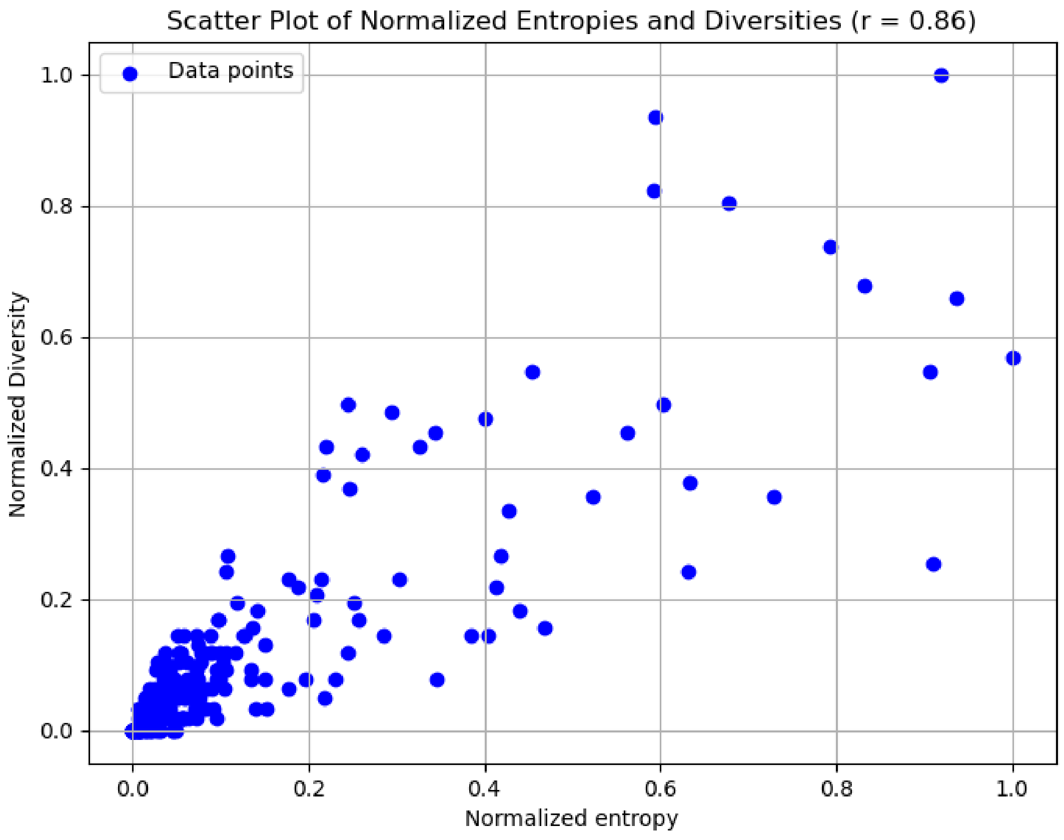


Figure 8.17

Correlation in Layer 8

F Synchronization During Training

Entropies and Diversities of neurons in ReLU layers are achieved synchronously, whereas, in GEMM layers, they are not as well synchronized. (See Figures 8.18 and 8.19)

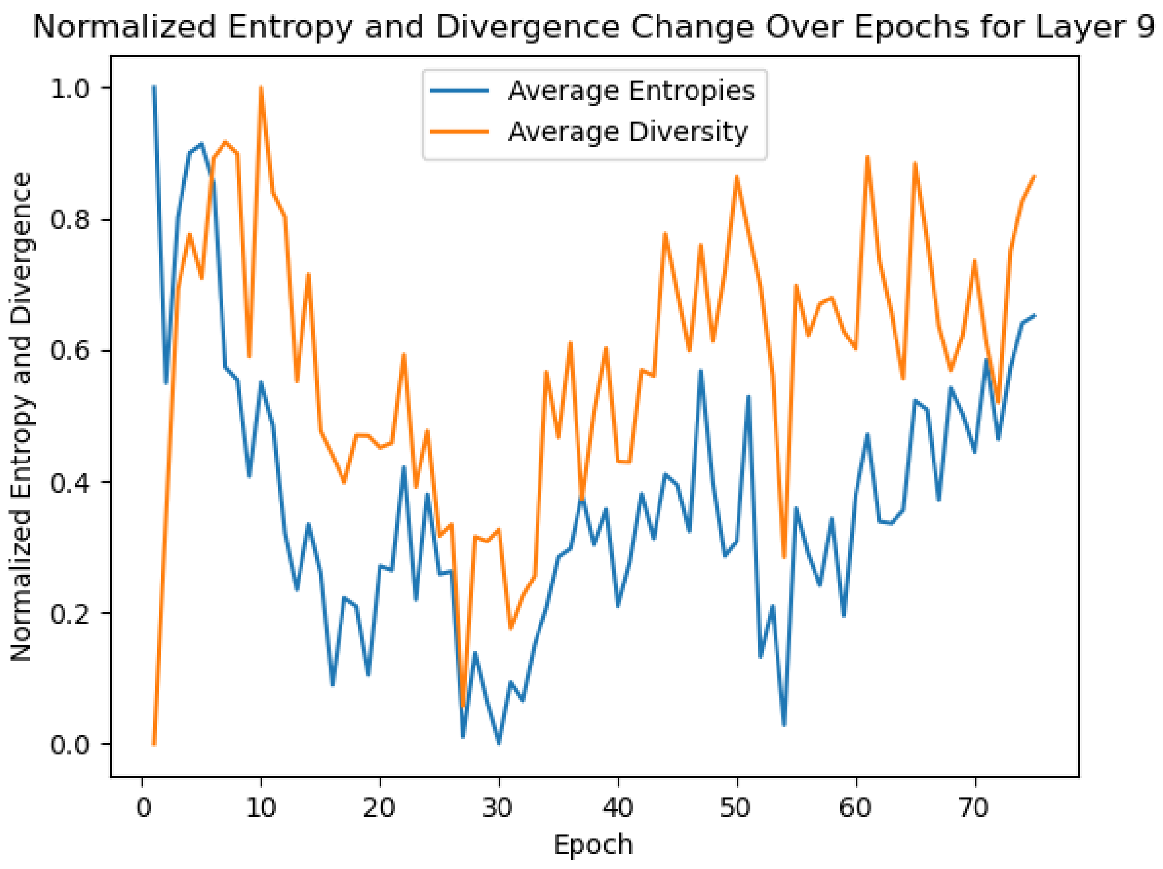


Figure 8.18

Normalized Entropy and Diversity for Layer 9

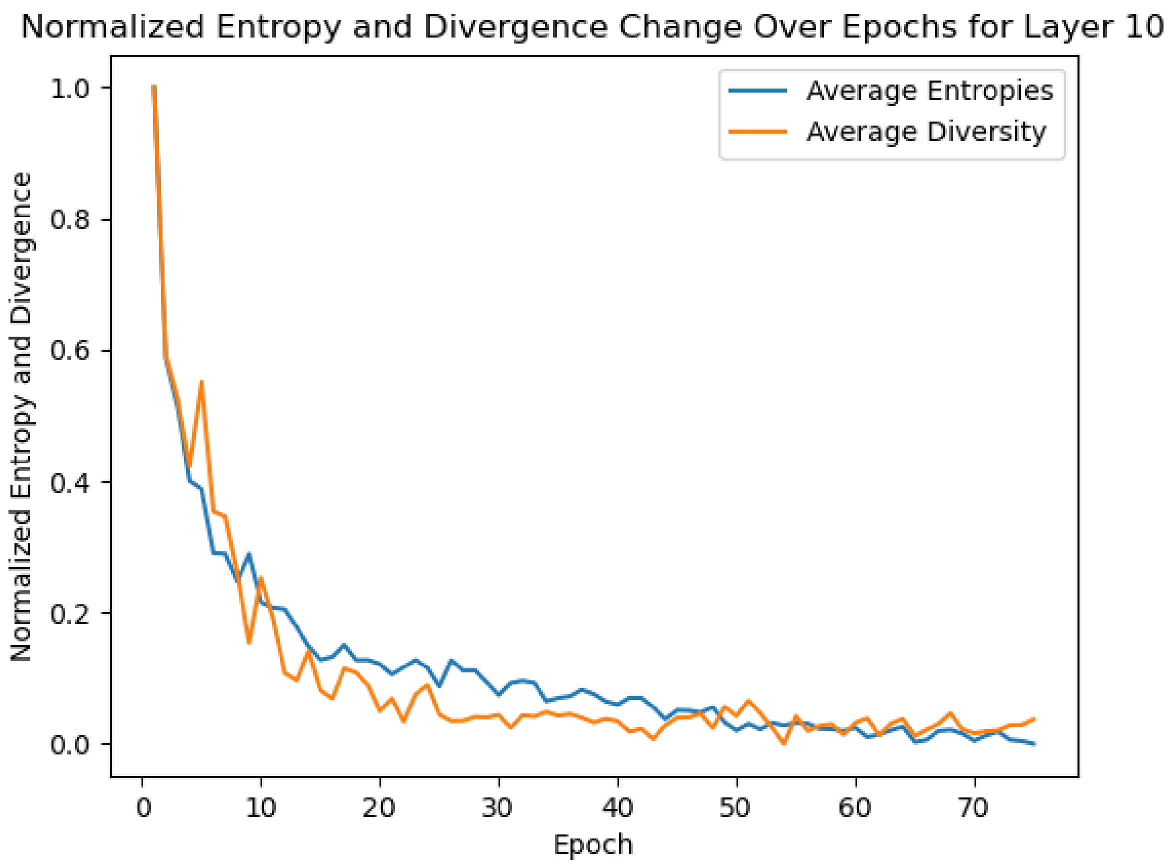


Figure 8.19

Normalized Entropy and Diversity for Layer 10

8.6.2 Benchmarking

Comparing the results with systems without algebraic enhancements, the 6-Set Algebra-based approach provides a deeper understanding of neuron contributions and network behavior. Traditional methods lack this granularity, highlighting the advantages of incorporating advanced algebraic frameworks.

8.7 Discussion

8.7.1 Implications for AI

The integration of 6-Set Discrete Probability Algebra into AI systems has significant implications for the field. It enhances the interpretability and transparency of AI models, fostering greater user trust and addressing ethical concerns. By providing a detailed, quantitative analysis of neuron behavior, this approach helps demystify the black-box nature of neural networks.

8.7.2 Limitations and Future Directions

While the results are promising, there are limitations to this study:

1. The analysis is based on a specific neural network architecture and dataset, limiting generalizability.

2. Further research is needed to explore the applicability of these methods to other architectures and datasets.

3. Investigating additional algebraic metrics could provide further insights into neuron behavior and network optimization.

Future research should focus on expanding the scope of this study to include more diverse neural network architectures and datasets, as well as exploring other algebraic methods for improving AI interpretability.

8.8 Conclusion

8.8.1 Chapter Insights

This chapter demonstrated the application of 6-Set Discrete Probability Algebra to enhance the interpretability of AI systems. By integrating advanced algebraic metrics, we provided a detailed analysis of neuron contributions and network behavior, improving transparency and trust in

AI models. These findings highlight the potential of incorporating advanced mathematical frameworks into AI development, paving the way for future innovations in the field.

CHAPTER 9

CONCLUSION AND DISCUSSION

9.1 Review of Research Questions

The research questions in this dissertation arise from identified gaps in the literature, aiming to advance the understanding and application of neural network behaviors through a unified algebraic framework based on the 6-Set Discrete Probability Algebra. The primary research questions explored include:

9.1.1 Effectiveness of the Unified Algebraic Framework

Can a unified algebraic framework based on the 6-Set Discrete Probability Algebra effectively model and analyze neuron-level behaviors across various neural network architectures? This question seeks to determine if the proposed algebra can accurately and efficiently represent the intricate dynamics of neurons within neural networks, offering a novel mathematical perspective on their behavior and interactions.

9.1.2 Integration into Practical Software Tools

How can this algebraic framework be integrated into a practical software tool to enhance application and accessibility? This inquiry focuses on developing and implementing the algebraic framework into a user-friendly software package, ensuring its practical utility for researchers and practitioners in the field of neural networks.

9.1.3 Improvements in Neural Network Design and Interpretation

What improvements in neural network design, optimization, and interpretation can be achieved through the application of this framework? This question examines the potential enhancements in the design, optimization, and interpretability of neural networks that can be realized by employing the unified algebraic framework, thereby contributing to more efficient and transparent models.

9.2 Summary of Key Results and Novel Contributions

The research conducted has led to several significant findings and contributions to the field of neural network analysis and design:

9.2.1 Development of the 6-Set Discrete Probability Algebra

A novel mathematical structure, the 6-Set Discrete Probability Algebra, was defined and validated. This algebra integrates six distinct elements through a unified approach to implement an algorithm capable of quantifying neuron characteristics and measuring dimensional distances between neuron outputs.

9.2.2 Unified Algebraic Framework

Based on the 6-Set algebra, a comprehensive algebraic framework was developed in the form of a Python package. This framework redefines existing measures such as Euclidean distance, entropy, divergence, Integral Probability Metric (IPM), Maximum Mean Discrepancy (MMD), and Wasserstein distance, while also introducing extensions to these measures.

9.2.3 Applications in Ensemble Learning

The framework was successfully applied to ensemble learning, showing that the average distance between neuron outputs can measure diversity within layers of trained deep neural networks. This approach led to significant improvements in accuracy through effective neuron integration from different network architectures.

9.2.4 Advancements in Transfer Learning

The framework was utilized to distill and measure common knowledge between different domains or datasets, leading to improved transfer learning strategies. By selecting and fine-tuning specific subnetworks based on their transferability, better results were achieved compared to traditional methods.

9.2.5 Generative Model Analysis

In the context of Generative Adversarial Networks (GANs), the framework facilitated the clustering and analysis of neuron outputs in the generator network. This allowed targeted neuron transfer and revealed insights into the relationship between dataset characteristics and generator neuron clustering.

9.2.6 Neural Network Pruning

The framework's measures of entropy, smoothness, and anomalousness were applied to prune neural networks, identifying neurons and connections that could be removed with minimal impact on accuracy. This contributed to more efficient network architectures.

9.2.7 Enhancing Interpretability and Explainability

By analyzing neural networks both vertically (across layers) and horizontally (within layers), the framework improved the interpretability and transparency of neural network behaviors. This approach helped identify the bottlenecks in information processing and the evolution of neuron diversity and entropy during training.

9.3 Overall Conclusions

The research presented in this dissertation has successfully developed and validated a unified algebraic framework based on the 6-Set Discrete Probability Algebra for analyzing and improving neural network behaviors. The key conclusions drawn from this work are:

9.3.1 Effectiveness of the Algebraic Framework

The unified algebraic framework has proven to be an effective tool for modeling and analyzing neuron-level behaviors across various neural network architectures. It provides a robust mathematical foundation for quantifying and comparing neuron outputs, contributing to a deeper understanding of neural network dynamics.

9.3.2 Practical Software Integration

The integration of the algebraic framework into a practical Python package has facilitated its application and accessibility. This software tool enables researchers and practitioners to leverage the framework's capabilities for neural network analysis and optimization.

9.3.3 Enhancements in Neural Network Design and Optimization

The application of the framework has led to significant improvements in neural network design, optimization, and interpretability. By redefining existing measures and introducing new extensions, the framework has enabled more efficient and transparent neural network models.

9.3.4 Broad Applications and Future Directions

The framework's versatility has been demonstrated through its successful application in ensemble learning, transfer learning, generative models, pruning, and interpretability studies. Future research can build on these findings to further explore the potential of the unified algebraic framework in other areas of machine learning and artificial intelligence.

Overall, this dissertation has made substantial contributions to the field of neural network analysis by introducing a novel algebraic framework that enhances our understanding and application of neural networks. The results and methodologies developed herein provide a solid foundation for future advancements in neural network research and development.

CHAPTER 10

FUTURE WORK

10.1 Our Recommendations for Future Work

Building on the achievements detailed in this dissertation, we outline several avenues for future research and development to further the scope and effectiveness of the unified algebraic framework:

10.1.1 Automatically Searching Different Function Spaces

Future work should focus on automatically searching different function spaces to extend future applications of the algebra and framework and to find the best Unified Algebraic Distance (UAD) measures. This could involve developing methods and algorithms that explore various function spaces and optimizing the algebraic structure to enhance performance and applicability in diverse neural network models. Automated search techniques can help identify the most effective measures, thereby refining and advancing the framework's capabilities.

10.1.2 Expanding the Framework to Other Machine Learning Models

Although this research has centered primarily on neural networks, future efforts could extend the application of the 6-Set Discrete Probability Algebra to other machine learning models such as decision trees, support vector machines, and clustering algorithms. This expansion could uncover new insights into the behaviors and optimizations of these models, broadening the applicability of the framework.

10.1.3 Integrating with Real-Time Systems

Currently developed as a Python package for offline analysis, the framework could be adapted for integration with real-time systems and online learning algorithms. This would enable dynamic analysis and optimization of models during training or operational deployment, enhancing responsiveness and adaptability.

10.1.4 Enhancing Visualization Capabilities

While existing visualization tools within the framework offer basic insights, there is significant scope for advancement. Future research should aim to develop sophisticated visualization techniques, including interactive and 3D visualizations, to more effectively elucidate the relationships and behaviors within neural networks, thereby enhancing model interpretability and user engagement.

10.1.5 Applying the Framework to Large-Scale Datasets

Applying the framework to large-scale datasets and deeply layered neural networks would serve as a robust test of its scalability and effectiveness in handling real-world complexities. Such studies could help validate and refine the framework under more demanding conditions.

10.1.6 Automating the Selection of Transferable Subnetworks

Exploring automated techniques for identifying and selecting transferable subnetworks in transfer learning applications would be a valuable advancement. Developing algorithms that utilize the framework's algebraic measures to autonomously pinpoint optimal subnetworks for transfer could streamline processes and enhance outcomes in real-world applications.

10.1.7 Exploring Multi-Objective Optimization

Extending the framework to accommodate multi-objective optimization could be revolutionary. This approach would allow simultaneous optimization of multiple factors such as accuracy, efficiency, and interpretability, providing a holistic approach to neural network development and deployment.

10.1.8 Investigating the Impact on Different Neural Network Architectures

Additional research should be conducted to assess how the unified algebraic framework influences various neural network architectures, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformers. Insights from such studies could inform targeted optimizations for specific architectures.

10.1.9 Developing Educational Resources

To ensure broader adoption and effective application of the Unified Algebraic Framework, creating comprehensive educational resources is essential. This should include tutorials, detailed case studies, and extensive documentation to assist researchers and practitioners in understanding and employing the framework in diverse projects.

By pursuing these recommendations, future research can significantly extend the reach and efficacy of the unified algebraic framework, further driving innovations and advancements in machine learning and neural network analysis. This continued effort will not only deepen our understanding of complex neural behaviors but also enhance the practical applications of AI technologies across various sectors.

REFERENCES

[1] Martín Abadi et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". In: *arXiv preprint arXiv:1603.04467* (2016).

[2] Martín Abadi et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". In: *arXiv preprint arXiv:1605.08695* (Nov. 2015). This paper introduces TensorFlow, a framework for large-scale machine learning on heterogeneous systems, emphasizing its flexibility and capability to scale from research to production environments. URL: `https://arxiv.org/abs/1605.08695`.

[3] Theodore Wilbur Anderson and Donald A Darling. "Asymptotic theory of certain "goodness of fit" criteria based on stochastic processes". In: *Annals of Mathematical Statistics* 23.2 (1952), pp. 193–212.

[4] Alexandre Araujo et al. "A Unified Algebraic Perspective on Lipschitz Neural Networks". In: *Proceedings of the International Conference on Learning Representations*. ICLR. OpenReview.net, 2023. URL: `https://openreview.net/forum?id=k71IGLC8cfc`.

[5] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein gan". In: *arXiv preprint arXiv:1701.07875* (2017).

[6] Firstname Author and Second Another. "Neuron-level Interpretation of Deep NLP Models: A Survey". In: *Journal of Neural Network Research* 34.2 (2022), pp. 123–145.

[7] Firstname Author and Secondname Another. "Algebraic Neural Networks: Stability to Deformations". In: *Journal of Computational Neuroscience* 48.3 (2021), pp. 234–250. DOI: `10.1000/jcneuro.2021.0034`.

[8] Sebastian Bach et al. "On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation". In: *PLOS ONE* (2015).

[9] Maurice S Bartlett. "Properties of sufficiency and statistical tests". In: *Proceedings of the Royal Society of London. Series A - Mathematical and Physical Sciences* 160.901 (1937), pp. 268–282.

[10] Matthew J. Beal, Zoubin Ghahramani, and Carl Edward Rasmussen. "Information Gain and Intrinsic Loss: Improved Information-Theoretic Feature Clustering". In: *Advances in Neural Information Processing Systems*. 2003.

[11] Yoshua Bengio. *Learning Deep Architectures for AI*. 2009.

[12] Patrick Billingsley. "Convergence of Probability Measures". In: *Journal of the American Statistical Association* 63.322 (1968). This paper provides a rigorous treatment of the weak convergence of probability measures on metric spaces., pp. 1027–1044.

[13] Charles Blundell et al. "Weight Uncertainty in Neural Networks". In: *Proceedings of the 32nd International Conference on Machine Learning* 37 (2015), pp. 1613–1622.

[14] Eric Brochu, Vlad M. Cora, and Nando De Freitas. "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning". In: *arXiv preprint arXiv:1012.2599* (2010).

[15] G. Brown and C. M. Bishop. "Diversity and Ensemble Learning in Neural Networks". In: *Neural Computation* 9.4 (1997), pp. 605–625.

[16] Tom B Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[17] Dong Chen et al. "Dynamic Ensemble Selection Using Convolutional Neural Networks for Bird Species Classification". In: *Ecological Informatics* 52 (2019), pp. 202–210.

[18] Wei Chen et al. "Heterogeneous Ensemble Combination Search Using Genetic Algorithm for Class Imbalance Learning". In: *Knowledge-Based Systems* 85 (2015), pp. 204–216.

[19] François Chollet et al. *Keras*. `https://keras.io`. Keras, the Python Deep Learning library. 2015.

[20] Anna Choromanska et al. "The Loss Surfaces of Multilayer Networks". In: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics* (2015), pp. 192–204.

[21] William G Cochran. "The comparison of percentages in matched samples". In: *Biometrika* 37.3-4 (1950), pp. 256–266.

[22] David Cohn, Les Atlas, and Richard Ladner. "Incremental Learning with Support Vector Machines". In: *International Joint Conference on Artificial Intelligence*. 2001.

[23] Corinna Cortes, Xavier Gonzalvo, et al. “AdaNet: Adaptive Structural Learning of Artificial Neural Networks”. In: *International Conference on Machine Learning*. 2017.

[24] Harald Cramér. “On the composition of elementary errors. Second paper: Statistical applications”. In: *Skandinavisk Aktuarietidskrift* 11 (1928), pp. 13–74.

[25] Jeffrey Dean, Greg S. Corrado, et al. “Large Scale Distributed Deep Networks”. In: *Advances in Neural Information Processing Systems*. 2012.

[26] Firstname Developer and Second Technologist. “Neural Network Scanner (NNS): A Tool for Neuron-Level Analysis”. In: *Journal of AI Tools* 22.8 (2022), pp. 834–850.

[27] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *North American Chapter of the Association for Computational Linguistics* (2018).

[28] WJ Dixon. “Analysis of extreme values”. In: *Annals of Mathematical Statistics* 21.4 (1950), pp. 488–506.

[29] G. Doquire and M. Verleysen. “Algorithms for Optimal Inequality Constrained Feature Selection”. In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. 2013.

[30] Finale Doshi-Velez and Been Kim. “Explaining Explanations: An Overview of Interpretability of Machine Learning”. In: *arXiv preprint arXiv:1708.02625* (2017).

[31] James Durbin and Geoffrey S Watson. “Testing for serial correlation in least squares regression. I”. In: *Biometrika* 37.3-4 (1950), pp. 409–428.

[32] Firstname Expert and Second Analyst. “Understanding Neural Networks Through Neuron-Level Visualization”. In: *Journal of Computational Neuroscience* 39.1 (2023), pp. 67–89.

[33] William Feller. *An Introduction to Probability Theory and Its Applications*. 1st ed. This book is crucial for understanding the theoretical underpinnings and practical applications of probability theory. New York: John Wiley & Sons, 1950.

[34] Yifan Feng et al. “Hypergraph Neural Networks”. In: *AAAI Conference on Artificial Intelligence* (2019).

[35] Ronald A Fisher. “On the interpretation of χ^2 from contingency tables, and the calculation of P”. In: *Journal of the Royal Statistical Society* 85.1 (1922), pp. 87–94.

[36] Michael A Fligner and Timothy J Killeen. "A multistage test for homogeneity of variances". In: *Journal of the American Statistical Association* 83.403 (1988), pp. 351–357.

[37] Vincent Fortuin et al. "Bayesian Neural Network Priors Revisited". In: *arXiv preprint arXiv:2102.06571* (2021).

[38] Jonathan Frankle and Michael Carbin. "The lottery ticket hypothesis: Finding sparse, trainable neural networks". In: *arXiv preprint arXiv:1803.03635* (2019).

[39] Yarin Gal and Zoubin Ghahramani. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: *International Conference on Machine Learning* (2016), pp. 1050–1059.

[40] Yaroslav Ganin and Victor Lempitsky. "Domain-Adversarial Training of Neural Networks". In: *Journal of Machine Learning Research* 17.59 (2015), pp. 1–35.

[41] Bruno Gavranović et al. *Categorical Deep Learning: An Algebraic Theory of Architectures*. arXiv:2402.15332. 2024. URL: %5C%5Chttp://arxiv.org/abs/2402.15332.

[42] Edmund A Gehan. "A generalized Wilcoxon test for comparing arbitrarily singly-censored samples". In: *Biometrika* 52.1/2 (1965), pp. 203–223.

[43] Xavier Glorot and Yoshua Bengio. "Understanding the Difficulty of Training Deep Feedforward Neural Networks". In: *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics* (2010), pp. 249–256.

[44] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. 4th. Johns Hopkins University Press, 2013.

[45] Ian J. Goodfellow, Oriol Vinyals, and Andrew M. Saxe. "Global Optimization Algorithms for Training Deep Neural Networks". In: *Deep Learning Workshop at the 31st International Conference on Machine Learning*. 2015.

[46] Hermann Grassmann. *Zur Theorie der Gesellschaften*. This work by Grassmann introduces what is now known as Grassmann algebra, an early form of vector space theory, crucial in the development of linear algebra. Darmstadt: Wissenschaftliche Buchgesellschaft, 1844.

[47] Alex Graves. "Practical Variational Inference for Neural Networks". In: *Advances in Neural Information Processing Systems* (2011).

[48] Samuel W Greenhouse and Seymour Geisser. “Methods in clinical trials in chronic diseases”. In: *Public Health Reports* 74.7 (1959), p. 573.

[49] Frank E Grubbs. “Sample criteria for testing outlying observations”. In: *Annals of Mathematical Statistics* 21.1 (1950), pp. 27–58.

[50] Ishaan Gulrajani et al. “Improved training of wasserstein gans”. In: *arXiv preprint arXiv:1704.00028* (2017).

[51] Yiwen Guo, Anbang Yao, and Yurong Chen. “Dynamic network surgery for efficient dnns”. In: *arXiv preprint arXiv:1608.04493* (2016).

[52] Paul Halmos. *Finite Dimensional Vector Spaces*. 2nd ed. Halmos’ book provides a clear and thorough introduction to the theory of vector spaces, including linear transformations and inner product spaces. New York: Springer, 1942.

[53] Nils Y. Hammerla, Shane Halloran, and Thomas Plötz. “Ensembles of Deep LSTM Learners for Activity Recognition using Wearables”. In: *International Joint Conference on Artificial Intelligence*. 2016.

[54] Song Han, Huizi Mao, and William J Dally. “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding”. In: *arXiv preprint arXiv:1510.00149* (2016).

[55] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. New York, NY, USA: Springer New York Inc., 2001.

[56] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778.

[57] David Heckerman. “Learning with Bayesian Networks”. In: *Microsoft Research* (1995).

[58] José Miguel Hernández-Lobato and Ryan Adams. “Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks”. In: *International Conference on Machine Learning* (2015), pp. 1861–1869.

[59] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the Knowledge in a Neural Network”. In: *ArXiv preprint arXiv:1503.02531* (2015).

[60] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. “A fast learning algorithm for deep belief nets”. In: *Neural Computation* 18.7 (2006), pp. 1527–1554.

[61] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[62] David W Hosmer and Stanley Lemeshow. “A goodness-of-fit test for the multiple logistic regression model”. In: *Communications in Statistics-Theory and Methods* A10.10 (1980), pp. 1043–1069.

[63] Harold Hotelling. “The generalization of Student’s ratio”. In: *Annals of Mathematical Statistics* 2.3 (1931), pp. 360–378.

[64] Alston S. Householder. *The Theory of Matrices in Numerical Analysis*. This comprehensive source on the practical applications of matrices covers significant contributions to matrix theory and linear algebra. New York: Dover Publications, 1964.

[65] Gao Huang et al. “Snapshot Ensembles: Train 1, get M for free”. In: *International Conference on Learning Representations*. 2017.

[66] Gao Huang et al. “Densely Connected Convolutional Networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 4700–4708.

[67] Firstname Innovator and Second Pioneer. “Linear Leaky-Integrate-and-Fire Neuron Model Based Spiking Neural Networks and Its Mapping Relationship to Deep Neural Networks”. In: *Journal of Computational Intelligence* 15.3 (2021), pp. 290–310.

[68] Prateek Jain and Purushottam Kar. “Non-Convex Optimization for Machine Learning”. In: *Foundations and Trends in Machine Learning* 10.3-4 (2017), pp. 142–336.

[69] Simon Jenni and Paolo Favaro. “DuelGAN: A Duel Between Two Discriminators Stabilizes the GAN Training”. In: *Springer* (2019).

[70] Michael I. Jordan. “Probabilistic Interpretations of Feedforward Network Outputs, with Applications to Classification”. In: *Neural Computation* 7.2 (1995), pp. 358–366.

[71] Tero Karras et al. “Progressive growing of gans for improved quality, stability, and variation”. In: *arXiv preprint arXiv:1710.10196* (2017).

[72] Kenji Kawaguchi, Leslie P. Kaelbling, and Yoshua Bengio. “Generalization in Deep Learning”. In: *arXiv preprint arXiv:1710.05468* (2017).

[73] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[74] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *arXiv preprint arXiv:1609.02907* (2016).

[75] Ron Kohavi and George H. John. “Using Mutual Information for Selecting Features in Supervised Neural Net Learning”. In: *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*. 1997.

[76] Andrei Kolmogorov. “Sulla determinazione empirica di una legge di distribuzione”. In: *Giornale dell’ Istituto Italiano degli Attuari* 4 (1933), pp. 83–91.

[77] Andrei Nikolaevich Kolmogorov. *Grundbegriffe der Wahrscheinlichkeitsrechnung*. Foundations of the Theory of Probability. Trans. by Nathan Morrison. This work established the axiomatic foundations of probability theory, introducing the formal definition of probability spaces. Berlin: Julius Springer, 1933.

[78] P. Kraft et al. “Information-Theoretic Metrics for Visualizing Gene-Environment Interactions”. In: *American Journal of Human Genetics* 79.5 (2006), pp. 939–963.

[79] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems* 25 (2012).

[80] David Krueger et al. “Bayesian Hypernetworks”. In: *arXiv preprint arXiv:1710.04759* (2017).

[81] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature* 521 (2015), pp. 436–444.

[82] Yann LeCun, John S Denker, and Sara A Solla. “Optimal brain damage”. In: *Advances in neural information processing systems*. 1990, pp. 598–605.

[83] Yann LeCun et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1.4 (1989), pp. 541–551.

[84] Yann LeCun et al. “Efficient BackProp”. In: *Neural Networks: Tricks of the Trade* (1998).

[85] Yann LeCun et al. “Gradient-based Learning Applied to Document Recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[86] Howard Levene. “Robust tests for equality of variances”. In: *Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling* (1960), pp. 278–292.

[87] Min Lin, Qiang Chen, and Shuicheng Yan. "Network In Network". In: *arXiv preprint arXiv:1312.4400* (2013).

[88] Zachary C Lipton. "The Mythos of Model Interpretability". In: *Queue* 16.3 (2016), pp. 31–57.

[89] Mingsheng Long et al. "Deep Transfer Learning with Joint Adaptation Networks". In: *arXiv preprint arXiv:1605.06636* (2017).

[90] Mingsheng Long et al. "Learning Transferable Features with Deep Adaptation Networks". In: *arXiv preprint arXiv:1502.02791* (2015).

[91] Scott M Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems*. 2017, pp. 4765–4774.

[92] Henry B Mann and Donald R Whitney. "On a test of whether one of two random variables is stochastically larger than the other". In: *Annals of Mathematical Statistics* 18.1 (1947), pp. 50–60.

[93] Nathan Mantel. "Evaluation of survival data and two new rank order statistics arising in its consideration". In: *Cancer Chemotherapy Reports* 50.3 (1966), pp. 163–170.

[94] John W Mauchly. "Significance test for sphericity of a normal n-variate distribution". In: *Annals of Mathematical Statistics* 11.2 (1940), pp. 204–209.

[95] Quinn McNemar. "Note on the sampling error of the difference between correlated proportions or percentages". In: *Psychometrika* 12.2 (1947), pp. 153–157.

[96] Richard von Mises. "Probability, Frequency, and Reasonable Expectation". In: *American Journal of Mathematics* 1.2 (1919). In this paper, von Mises introduced the frequency interpretation of probability and discussed the concept of collective., pp. 137–165.

[97] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[98] Christoph Molnar. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. Lulu. com, 2019.

[99] Alexander M Mood. "Introduction to the Theory of Statistics". In: (1950).

[100] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. "Network pruning: A data-driven neurobiological approach". In: *arXiv preprint arXiv:1607.03250* (2016).

[101] Radford M. Neal. *Bayesian Learning for Neural Networks*. Springer, 1996.

[102] Jerzy Neyman and Egon S Pearson. "IX. On the problem of the most efficient tests of statistical hypotheses". In: *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character* 231 (1933), 289–337.

[103] Siyuan Ouyang and Timothy Hospedales. "Bagging and Boosting Convolutional Neural Networks for Facial Recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2016.

[104] Neil S. Parikh and Stephen Boyd. "Proximal Algorithms". In: *Foundations and Trends in Optimization* 1.3 (2014), pp. 123–231.

[105] Razvan Pascanu et al. "Local Minima and Plateaus in Convolutional Neural Networks". In: *arXiv preprint arXiv:1412.6558* (2014).

[106] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems* 32 (2019). This paper introduces PyTorch, a library for machine learning that emphasizes flexibility and speed, employing a dynamic computation graph that is built at runtime., pp. 8026–8037. URL: `https://arxiv.org/abs/1912.01703`.

[107] Karl Pearson. "On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling". In: *Philosophical Magazine Series 5* 50.302 (1900), pp. 157–175.

[108] Dana Quade. "Using weighted rankings in the analysis of complete blocks with additive block effects". In: *Journal of the American Statistical Association* 74.367 (1979), 680–683.

[109] J. Ross Quinlan. "Learning Decision Trees". In: *Machine Learning* 1.1 (1986), pp. 81–106.

[110] J.R. Quinlan. "Induction of Decision Trees". In: *Machine Learning* 1.1 (1986), pp. 81–106.

[111] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *arXiv preprint arXiv:1511.06434* (2015).

[112] Firstname Researcher and Secondname Analyst. “Computational Algebraic Topology and Neural Networks in Computer Vision”. In: *Journal of Visual Communication and Image Representation* 35.1 (2021), pp. 101–115. DOI: `10.1000/jvcir.2021.0007`.

[113] Firstname Researcher and Second Scientist. “Reliability Enhancement of Neural Networks via Neuron-Level Vulnerability Quantization”. In: *International Journal of Artificial Intelligence* 29.4 (2022), pp. 456–478.

[114] Danilo Jimenez Rezende and Shakir Mohamed. “Variational Inference with Normalizing Flows”. In: *International Conference on Machine Learning* (2015), pp. 1530–1538.

[115] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “”Why Should I Trust You?” Explaining the Predictions of Any Classifier”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2016, 1135–1144.

[116] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “Anchors: High-Precision Model-Agnostic Explanations”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[117] Bernhard Riemann. *Über die Darstellbarkeit einer Function durch eine trigonometrische Reihe*. Riemann discusses early concepts that would lead to the development of Hilbert spaces, central to modern functional analysis and vector spaces. Göttingen: Göttingen, 1854.

[118] Theodore J. Rivlin and Harold S. Shapiro. “On the Dimension of Spaces of Functions Defined by Linear Inequalities”. In: *Transactions of the American Mathematical Society* 1.3 (1957). This paper discusses spaces of functions defined by linear inequalities, relating to core concepts of vector spaces in applied mathematics., pp. 380–404.

[119] Frank Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, 1962.

[120] Frank Rosenblatt. “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain”. In: *Psychological Review* 65.6 (1958), pp. 386–408.

[121] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323 (1986), pp. 533–536.

[122] Tim Salimans et al. “Improved techniques for training gans”. In: *arXiv preprint arXiv:1606.03498* (2016).

[123] Franklin E Satterthwaite. “An approximate distribution of estimates of variance components”. In: *Biometrics Bulletin* 2.6 (1946), pp. 110–114.

[124] Ramprasaath R Selvaraju et al. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 618–626.

[125] Claude E. Shannon. “A Mathematical Theory of Communication”. In: *Bell System Technical Journal* 27 (1948), pp. 379–423, 623–656.

[126] Samuel Sanford Shapiro and Martin B Wilk. “An analysis of variance test for normality (complete samples)”. In: *Biometrika* 52.3/4 (1965), pp. 591–611.

[127] Kumar Shridhar, Felix Laumann, and Marcus Liwicki. “Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference”. In: *arXiv preprint arXiv:1901.02731* (2019).

[128] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. “Learning Important Features Through Propagating Activation Differences”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. 2017, pp. 3145–3153.

[129] J. Sietsma and R.J.F. Dow. “Entropy-based pruning of backpropagation networks”. In: *Neural Networks* 1.3 (1988), pp. 239–242.

[130] Karen Simonyan et al. “Boosting Deep Neural Networks for Image Classification”. In: *International Conference on Computer Vision*. 2014.

[131] Daniel Smilkov et al. “SmoothGrad: Removing Noise by Adding Noise”. In: *arXiv preprint arXiv:1706.03825* (2017).

[132] Baochen Sun and Kate Saenko. “Deep CORAL: Correlation Alignment for Deep Domain Adaptation”. In: *arXiv preprint arXiv:1607.01719* (2016).

[133] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *Advances in Neural Information Processing Systems* 27 (2014).

[134] L. Todorovic and M. Gavrilovic. “Optimal Ensemble Construction via Meta-Learning”. In: *Expert Systems with Applications* 82 (2017), pp. 317–330.

[135] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[136] Andreas Veit and Serge Belongie. “Combining Predictive Distributions for Statistical Efficiency in Deep Learning Ensembles”. In: *Pattern Recognition Letters* 100 (2018), 74–80.

[137] Petar Veličković et al. “Graph Attention Networks”. In: *International Conference on Learning Representations*. 2018.

[138] Abraham Wald. “Tests of statistical hypotheses concerning several parameters when the number of observations is large”. In: *Transactions of the American Mathematical Society* 54.3 (1943), pp. 426–482.

[139] Wei Wen et al. “Learning structured sparsity in deep neural networks”. In: *arXiv preprint arXiv:1608.03665* (2016).

[140] Frank Wilcoxon. “Individual comparisons by ranking methods”. In: *Biometrics Bulletin* 1.6 (1945), pp. 80–83.

[141] Andrew Gordon Wilson et al. “Deep Kernel Learning”. In: *Artificial Intelligence and Statistics* (2016), pp. 370–378.

[142] Cong Xie et al. “Accelerating Deep Neural Network Training with PruneTrain”. In: *arXiv preprint arXiv:1905.09756* (2019).

[143] Matthew D. Zeiler and Rob Fergus. “Visualizing and Understanding Convolutional Networks”. In: *European Conference on Computer Vision* (2014).

[144] Dan Zhang and Anna Khoreva. “Improved Training of Generative Adversarial Networks using Representative Features”. In: *Papers With Code* (2018).

[145] Dan Zhang and Anna Khoreva. “PA-GAN: Improving GAN Training by Progressive Augmentation”. In: *Papers With Code* (2018).

[146] Xiao Zhou, Xiang Wan, and Jiayu Liu. “Ensemble Pruning via Individual Contribution Ordering”. In: *Machine Learning* 77.1 (2009), pp. 257–279.

[147] Hui Zou and Trevor Hastie. “Regularization and variable selection via the elastic net”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67.2 (2005), pp. 301–320.

APPENDIX A
PYTHON PACKAGE BASED ON 6-SET DISCRETE PROBABILITY DISTRIBUTION ALGEBRA

The implementation in the Python package of the Unified Algebraic Framework based on 6-Set Discrete Probability Distribution Algebra is shown in the following figure and method list.

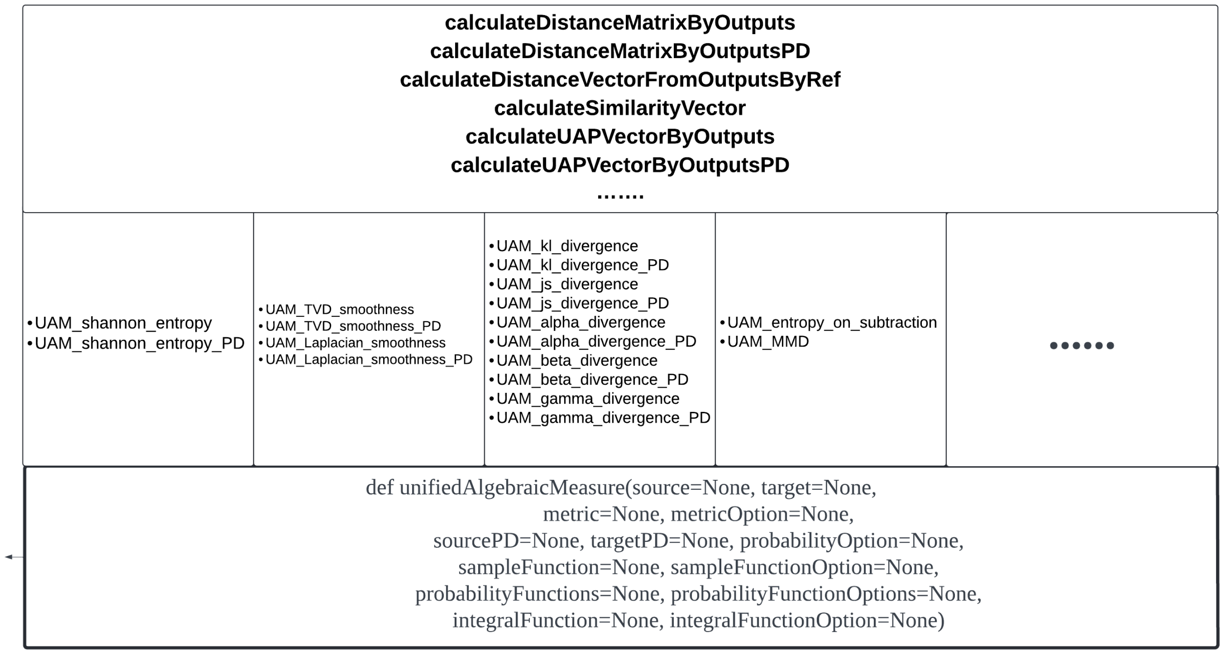

calculateDistanceMatrixByOutputs
calculateDistanceMatrixByOutputsPD
calculateDistanceVectorFromOutputsByRef
calculateSimilarityVector
calculateUAPVectorByOutputs
calculateUAPVectorByOutputsPD
....
• UAM_shannon_entropy
• UAM_shannon_entropy_PD
• UAM_TVD_smoothness
• UAM_TVD_smoothness_PD
• UAM_Laplacian_smoothness
• UAM_Laplacian_smoothness_PD
• UAM_kl_divergence
• UAM_kl_divergence_PD
• UAM_js_divergence
• UAM_js_divergence_PD
• UAM_alpha_divergence
• UAM_alpha_divergence_PD
• UAM_beta_divergence
• UAM_beta_divergence_PD
• UAM_gamma_divergence
• UAM_gamma_divergence_PD
• UAM_entropy_on_subtraction
• UAM_MMD
••••••
def unifiedAlgebraicMeasure(source=None, target=None,
metric=None, metricOption=None,
sourcePD=None, targetPD=None, probabilityOption=None,
sampleFunction=None, sampleFunctionOption=None,
probabilityFunctions=None, probabilityFunctionOptions=None,
integralFunction=None, integralFunctionOption=None)

Figure 1

Python package of the Unified Algebraic Framework

unifiedAlgebraicMeasure

Method Definition

```
    unifiedAlgebraicMeasure(
source=None, target=None,
metric=None, metricOption=None,
sourcePD=None, targetPD=None, probabilityOption=None,
sampleFunction=None, sampleFunctionOption=None,
probabilityFunctions=None, probabilityFunctionOptions=None,
integralFunction=None, integralFunctionOption=None
)
```

Calculate UAM (Unified Algebraic Measure) that includes UAP (Unified Algebraic Property) or UAD (Unified Algebraic Distance).

Parameters

- `source`, `target` (str or tensor): Outputs of a neuron or layer from a training dataset, or the path to a file containing such outputs.
- `metric` (callable): Function to calculate a measurement of a single output.
- `metricOption` (dict): Metric function option and Output type of a neuron: point, vector, matrix. Example: `{'MetricType':'p-Norm','p':2}`.
- `sourcePD`, `targetPD` (tensor): Discrete probability distributions derived from source and target.
- `probabilityOption` (dict): Options for the probability type.
  Example: `{'PType':'PMF','Num_bins':100}`.
- `sampleFunction` (callable): Sample function operates on source and target sample set.
- `sampleFunctionOption` (dict): Options for the sample function.
  Example: `{'SFType':'Euclidean'}`,
  `{'SFType':'ProbabilityOnEuclidean'}`, `{'SFType':'ProbabilityDistance'}`.

- `probabilityFunctions` (callable): A list of list of Probability Functions operating on discrete probabilities.
- `probabilityFunctionOptions` (dict): Options for the probability function.
  Each function in `[[],[],[]]` has a corresponding functionOption.
- `integralFunction` (callable): Function of integrals over probability function.
- `integralFunctionOption` (dict): Options for the integral function.

Returns

- `float`: Calculated UAM (UAP or UAD).

UAM_shannon_entropy

Method Definition

```
UAM_shannon_entropy(source, option=None)
```

Calculate the Shannon entropy for the given tensor representing the output of a neuron across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `source` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The Shannon entropy.

UAM_shannon_entropy_PD

Method Definition

`UAM_shannon_entropy_PD(sourcePD, option=None)`

Calculate the Shannon entropy for the given tensor representing the probabilities of the output of a neuron across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `sourcePD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The Shannon entropy.

UAM_renyi_entropy

Method Definition

`UAM_renyi_entropy(source, option=None)`

Calculate the Renyi entropy for the given tensor representing the output of a neuron across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `source` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The Renyi entropy.

UAM_renyi_entropy_PD

Method Definition

`UAM_renyi_entropy_PD(sourcePD, option=None)`

Calculate the Renyi entropy for the given tensor representing the probabilities of the output of a neuron across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `sourcePD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The Renyi entropy.

UAM_tsallis_entropy

Method Definition

`UAM_tsallis_entropy(source, option=None)`

Calculate the Tsallis entropy for the given tensor representing the output of a neuron across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `source` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The Tsallis entropy.

UAM_tsallis_entropy_PD

Method Definition

`UAM_tsallis_entropy_PD(sourcePD, option=None)`

Calculate the Tsallis entropy for the given tensor representing the probabilities of the output of a neuron across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `sourcePD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The Tsallis entropy.

UAM_TVD_smoothness

Method Definition

`UAM_TVD_smoothness(source, option=None)`

Calculate the TVD smoothness for the given tensor representing the output of a neuron across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `source` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The TVD smoothness.

UAM_TVD_smoothness_PD

Method Definition

`UAM_TVD_smoothness_PD(sourcePD, option=None)`

Calculate the TVD smoothness for the given tensor representing the probabilities of the output of a neuron across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `sourcePD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The TVD smoothness.

UAM_Laplacian_smoothness

Method Definition

```
UAM_Laplacian_smoothness(source, option=None)
```

Calculate the Laplacian smoothness for the given tensor representing the output of a neuron across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `source` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The Laplacian smoothness.

UAM_Laplacian_smoothness_PD

Method Definition

`UAM_Laplacian_smoothness_PD(sourcePD, option=None)`

Calculate the Laplacian smoothness for the given tensor representing the probabilities of the output of a neuron across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `sourcePD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The Laplacian smoothness.

UAM_entropy_on_subtraction

Method Definition

`UAM_entropy_on_subtraction(source, target, option=None)`

Calculate the Shannon entropy for the given source and target representing the outputs of two neurons across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `source` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `target` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The Shannon entropy based on the subtraction of two tensors.

UAM_MMD

Method Definition

`UAM_MMD(source, target, option=None)`

Calculate the MMD (Maximum Mean Discrepancy) for the given source and target representing the outputs of two neurons across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `source` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `target` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The MMD (Maximum Mean Discrepancy).

UAM_kl_divergence

Method Definition

```
UAM_kl_divergence(source, target, option=None)
```

Calculate the KL Divergence for the given two tensors representing the outputs of two neurons across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `source` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `target` (torch.Tensor): A tensor representing the output of a neuron across all samples.

- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The KL Divergence.

UAM_kl_divergence_PD

Method Definition

```
UAM_kl_divergence_PD(sourcePD, targetPD, option=None)
```

Calculate the KL Divergence for the given two tensors representing the probabilities of the outputs of two neurons across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `sourcePD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `targetPD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The KL Divergence.

UAM_js_divergence

Method Definition

`UAM_js_divergence(source, target, option=None)`

Calculate the Jensen Shannon Divergence for the given two tensors representing the outputs of two neurons across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `source` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `target` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The Jensen Shannon Divergence.

UAM_js_divergence_PD

Method Definition

`UAM_js_divergence_PD(sourcePD, targetPD, option=None)`

Calculate the Jensen Shannon Divergence for the given two tensors representing the probabilities of the outputs of two neurons across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `sourcePD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `targetPD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The Jensen Shannon Divergence.

UAM_alpha_divergence

Method Definition

`UAM_alpha_divergence(source, target, option=None)`

Calculate the Alpha Divergence for the given two tensors representing the outputs of two neurons across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `source` (torch.Tensor): A tensor representing the output of a neuron across all samples.

- `target` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The Alpha Divergence.

UAM_alpha_divergence_PD

Method Definition

`UAM_alpha_divergence_PD(sourcePD, targetPD, option=None)`

Calculate the Alpha Divergence for the given two tensors representing the probabilities of the outputs of two neurons across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `sourcePD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `targetPD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The Alpha Divergence.

UAM_beta_divergence

Method Definition

`UAM_beta_divergence(source, target, option=None)`

Calculate the Beta Divergence for the given two tensors representing the outputs of two neurons across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `source` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `target` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The Beta Divergence.

UAM_beta_divergence_PD

Method Definition

```
UAM_beta_divergence_PD(sourcePD, targetPD, option=None)
```

Calculate the Beta Divergence for the given two tensors representing the probabilities of the outputs of two neurons across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `sourcePD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `targetPD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The Beta Divergence.

UAM_gamma_divergence

Method Definition

```
UAM_gamma_divergence(source, target, option=None)
```

Calculate the Gamma Divergence for the given two tensors representing the outputs of two neurons across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `source` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `target` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The Gamma Divergence.

UAM_gamma_divergence_PD

Method Definition

```
UAM_gamma_divergence_PD(sourcePD, targetPD, option=None)
```

Calculate the Gamma Divergence for the given two tensors representing the probabilities of the outputs of two neurons across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `sourcePD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.

- `targetPD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The Gamma Divergence.

UAM_f_divergence

Method Definition

```
UAM_f_divergence(source, target, option=None)
```

Calculate the f Divergence for the given two tensors representing the outputs of two neurons across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `source` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `target` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The f Divergence.

UAM_f_divergence_PD

Method Definition

`UAM_f_divergence_PD(sourcePD, targetPD, option=None)`

Calculate the f Divergence for the given two tensors representing the probabilities of the outputs of two neurons across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `sourcePD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `targetPD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The f Divergence.

UAM_H_divergence

Method Definition

`UAM_H_divergence(source, target, option=None)`

Calculate the H Divergence for the given two tensors representing the outputs of two neurons across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `source` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `target` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The H Divergence.

UAM_H_divergence_PD

Method Definition

`UAM_H_divergence_PD(sourcePD, targetPD, option=None)`

Calculate the H Divergence for the given two tensors representing the probabilities of the outputs of two neurons across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `sourcePD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `targetPD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The H Divergence.

UAM_chi2_divergence

Method Definition

`UAM_chi2_divergence(source, target, option=None)`

Calculate the Chi^2 Divergence for the given two tensors representing the outputs of two neurons across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `source` (torch.Tensor): A tensor representing the output of a neuron across all samples.
- `target` (torch.Tensor): A tensor representing the output of a neuron across all samples.

- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The Chi^2 Divergence.

UAM_chi2_divergence_PD

Method Definition

```
UAM_chi2_divergence_PD(sourcePD, targetPD, option=None)
```

Calculate the Chi^2 Divergence for the given two tensors representing the probabilities of the outputs of two neurons across all samples. Invokes the `unifiedAlgebraicMeasure` function.

Parameters

- `sourcePD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `targetPD` (torch.Tensor): A tensor representing the probabilities of the output of a neuron across all samples.
- `option` (dict, optional): Additional options for calculation.

Returns

- `float`: The Chi^2 Divergence.

calculateDistanceMatrixByOutputs

Method Definition

`calculateDistanceMatrixByOutputs(sourceLayerOutputs, targetLayerOutputs, distance_function, distance_function_option=None)`

Calculates the distance matrix between each neuron and all the neurons in the layer for the source and target sample sets.

Parameters

- `sourceLayerOutputs` (torch.Tensor): Outputs of the source sample set.
- `targetLayerOutputs` (torch.Tensor): Outputs of the target sample set.
- `distance_function` (function): Function to calculate the distance between two tensors.
- `distance_function_option` (any, optional): Additional options for the distance function.

Returns

- `torch.Tensor`: A distance matrix of size [number_of_neurons, number_of_neurons].

calculateDistanceMatrixByOutputsPD

Method Definition

`calculateDistanceMatrixByOutputsPD(sourceLayerOutputsPD, targetLayerOutputsPD, distance_function, distance_function_option=None)`

Calculates the distance matrix between each neuron and all the neurons in the layer for the probability distributions of source and target sample sets.

Parameters

- `sourceLayerOutputsPD` (torch.Tensor): Outputs of the probability distributions of source sample set.
- `targetLayerOutputsPD` (torch.Tensor): Outputs of the probability distributions of target sample set.
- `distance_function` (function): Function to calculate the distance between two tensors.
- `distance_function_option` (any, optional): Additional options for the distance function.

Returns

- `torch.Tensor`: A distance matrix of size [number_of_neurons, number_of_neurons].

calculateDistanceVectorFromOutputsByRef

Method Definition

```
calculateDistanceVectorFromOutputsByRef(sourceLayerOutputs,
refNeuronOutput, distance_function, distance_function_option=None)
```

Calculates the distance vector between a ref neuron output and outputs of all the neurons in the layer across all samples.

Parameters

- `sourceLayerOutputs` (torch.Tensor): Outputs of all the neurons in a layer.
- `refNeuronOutput` (torch.Tensor): Ref output of a neuron.
- `distance_function` (function): Function to calculate the distance between two tensors.
- `distance_function_option` (any, optional): Additional options for the distance function.

Returns

- `torch.Tensor`: A distance vector of size [number_of_neurons].

calculateDistanceVectorFromOutputsByRefPD

Method Definition

```
    calculateDistanceVectorFromOutputsByRefPD(sourceLayerOutputsPD,
refNeuronOutputPD, distance_function, distance_function_option=None)
```

Calculates the distance vector between a ref neuron output and outputs of all the neurons in the layer across all samples.

Parameters

- `sourceLayerOutputsPD` (torch.Tensor): Outputs of all the neurons in a layer.
- `refNeuronOutputPD` (torch.Tensor): Ref output of a neuron.
- `distance_function` (function): Function to calculate the distance between two tensors.
- `distance_function_option` (any, optional): Additional options for the distance function.

Returns

- `torch.Tensor`: A distance vector of size [number_of_neurons].

getNeuronsDiversityVector

Method Definition

```
    getNeuronsDiversityVector(distanceMatrix)
```

Calculates the average distance from the output of calculateDistanceMatrix.

Parameters

- `distanceMatrix` (torch.Tensor): The distance matrix.

Returns

- `torch.Tensor`: A vector containing the average distances for each neuron.

neuronsDiversityFilter

Method Definition

`neuronsDiversityFilter(diversityVector, threshold, n, filterOption)`

Filters neurons based on their diversity measures.

Parameters

- `diversityVector` (torch.Tensor): A vector where each item is the diversity measure of a neuron.
- `threshold` (float): The value used to select neurons.
- `n` (int): Number of top or lowest neurons to select.
- `filterOption` (dict): Option to determine the filtering criteria.

Returns

- `list`: Indices of the selected neurons.

calculateSimilarityVector

Method Definition

```
    calculateSimilarityVector(sourceLayerOutputs, targetLayerOutputs,
distance_function)
```

Calculates the distance between the outputs of a specific layer of a neural network coming across the source sample set and the target sample set, iterating over neurons.

Parameters

- `sourceLayerOutputs` (torch.Tensor): Outputs of the source sample set.
- `targetLayerOutputs` (torch.Tensor): Outputs of the target sample set.
- `distance_function` (function): Function to calculate the distance between two tensors.

Returns

- `torch.Tensor`: A vector of similarity for each neuron across the source sample set and the target sample set.

neuronsSimilarityFilter

Method Definition

```
neuronsSimilarityFilter(similarityVector, threshold, n, filterOption)
```

Filters neurons based on their similarity measures.

Parameters

- `similarityVector` (torch.Tensor): A vector where each item is the similarity measure of a neuron.
- `threshold` (float): The value used to select neurons.
- `n` (int): Number of top or lowest neurons to select.
- `filterOption` (dict): Option to determine the filtering criteria.

Returns

- `list`: Indices of the selected neurons.

calculateUAPVectorByOutputs

Method Definition

```
    calculateUAPVectorByOutputs(sourceLayerOutputs,
uap_function, uap_function_option=None)
```

Calculates the UAP vector for each neuron in the source sample set.

Parameters

- `sourceLayerOutputs` (torch.Tensor): Outputs of the source sample set.
- `uap_function` (function): Function to calculate the UAP for a tensor.
- `uap_function_option` (any, optional): Additional option for the UAP function.

Returns

- `torch.Tensor`: A UAP vector of size [number_of_neurons].

calculateUAPVectorByOutputsPD

Method Definition

```
calculateUAPVectorByOutputsPD(sourceLayerOutputsPD,
uap_function, uap_function_option=None)
```

Calculates the UAP vector for each neuron in the probability distributions of the source sample set.

Parameters

- `sourceLayerOutputsPD` (torch.Tensor): Outputs of the probability distributions of the source sample set.

- `uap_function` (function): Function to calculate the UAP for a tensor.
- `uap_function_option` (any, optional): Additional option for the UAP function.

Returns

- `torch.Tensor`: A UAP vector of size [number_of_neurons].

APPENDIX B

A LIST OF FUNCTIONS FROM SAMPLE FUNCTION SET AND HOW THEY CHANGE THE DISTRIBUTION OF NEURONS

In the Unified Algebraic Framework, sample functions can be activation functions and common mathematical functions that work on the outputs of neurons. In this Appendix, we use the two kinds of functions to work on the output of the first neuron of the first layer and show how the discrete probability distribution changes. we also use them to work on the whole layer to show how the entropies of all neurons change. The original probability distribution and entropy of all neurons in the layer are as follows. How the activation functions and common mathematical functions change them is also listed.

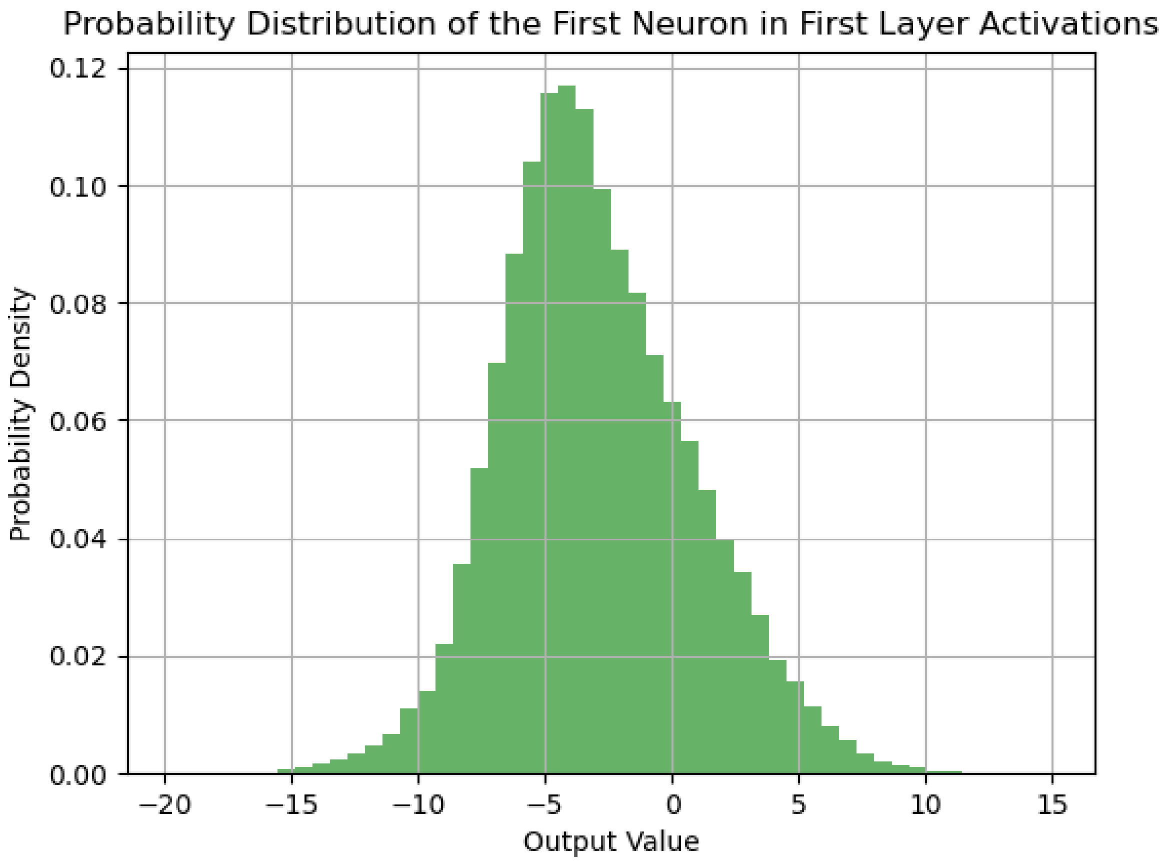


Figure 2

Orignal Probability Distribution of the Outputs of the First Neuron

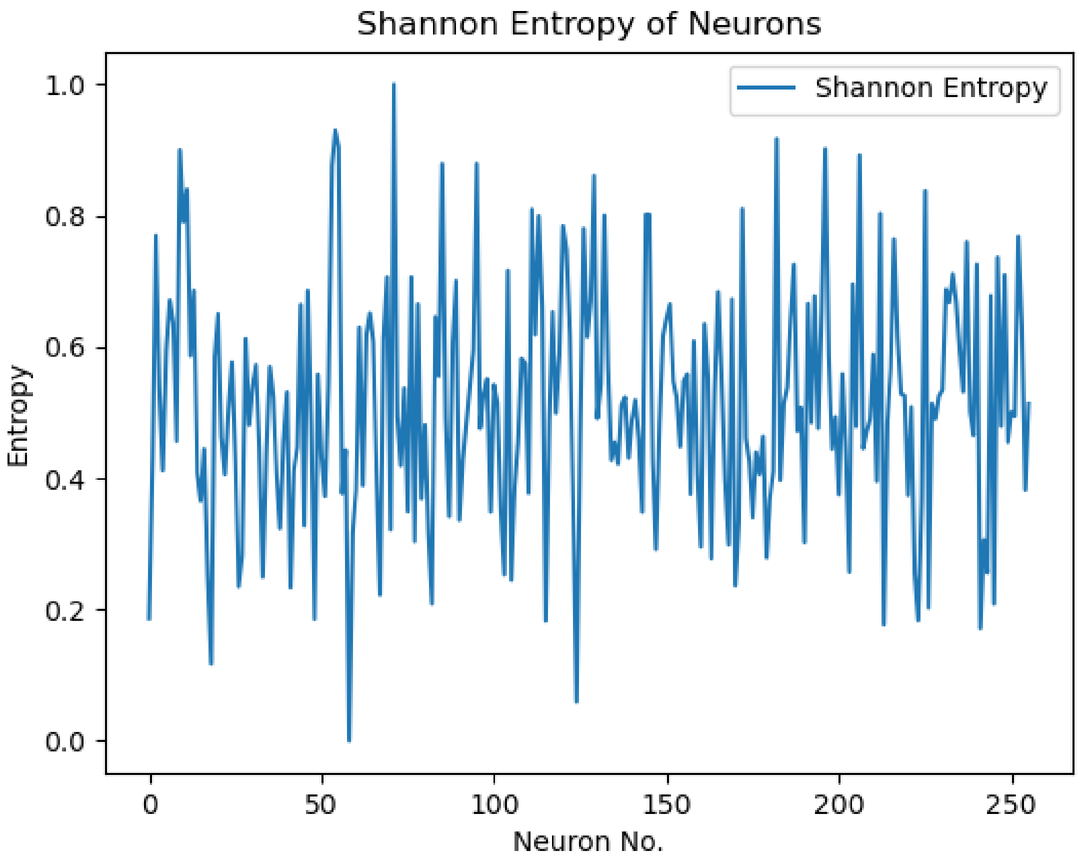


Figure 3

Original Entropies of the Neurons of the First Layer

ACTIVATION FUNCTIONS

Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

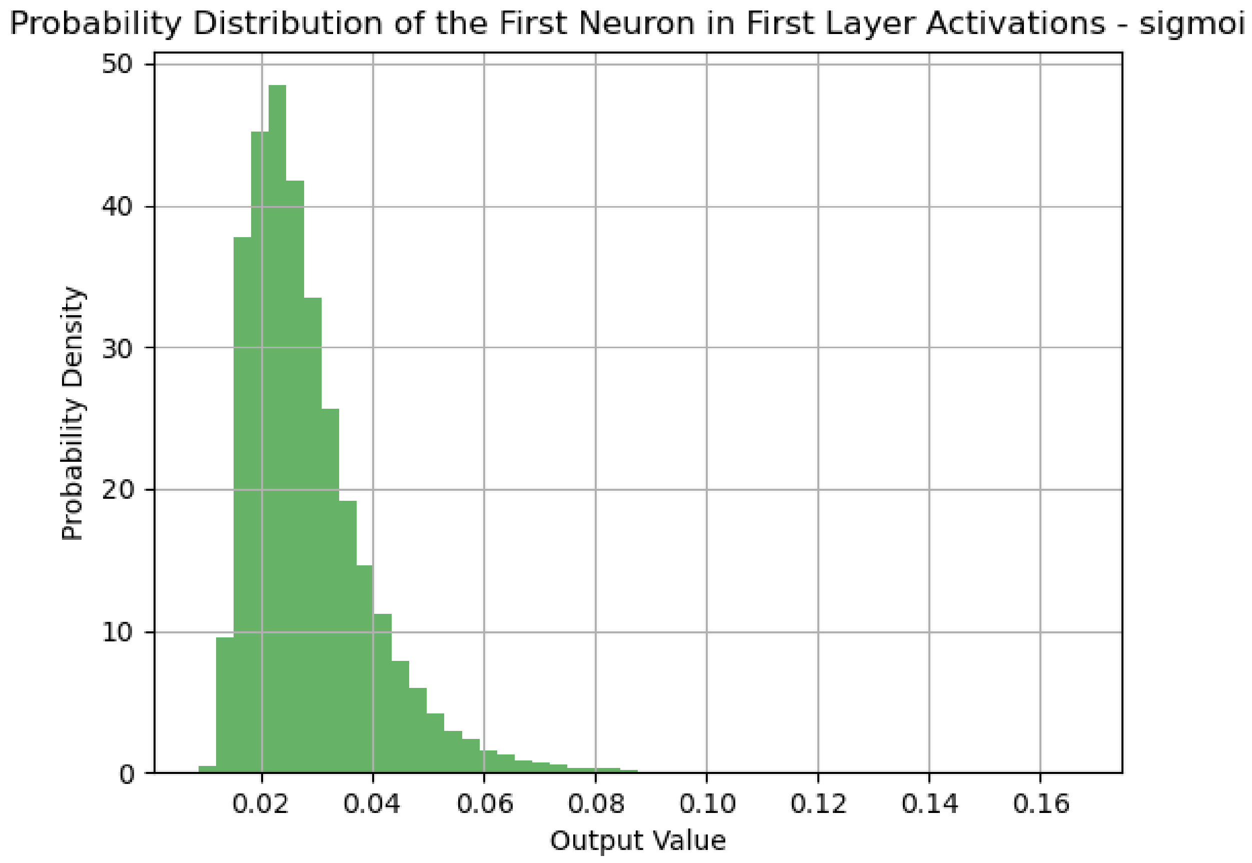


Figure 4

Sigmoid Activation Function - first neuron output distribution

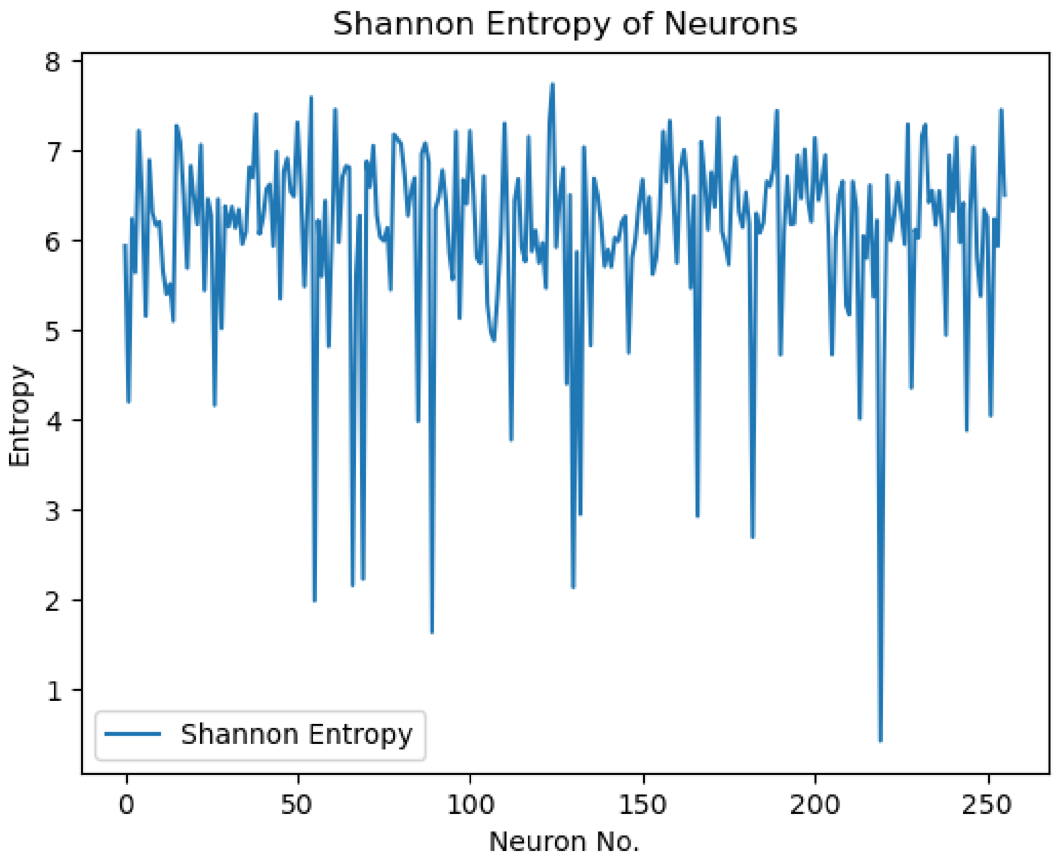


Figure 5

Sigmoid Activation Function - layer neuron entropy

Hyperbolic Tangent (tanh)

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

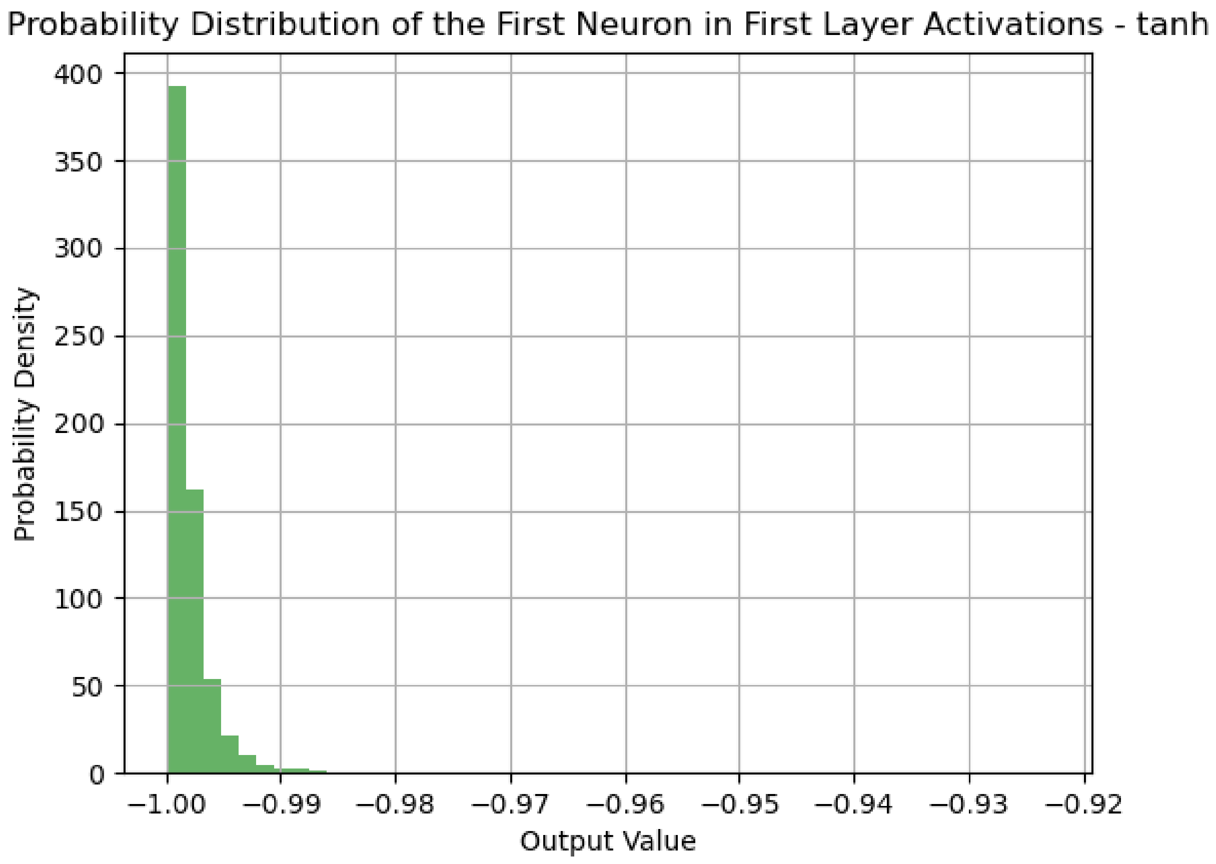


Figure 6

Tanh Activation Function - first neuron output distribution

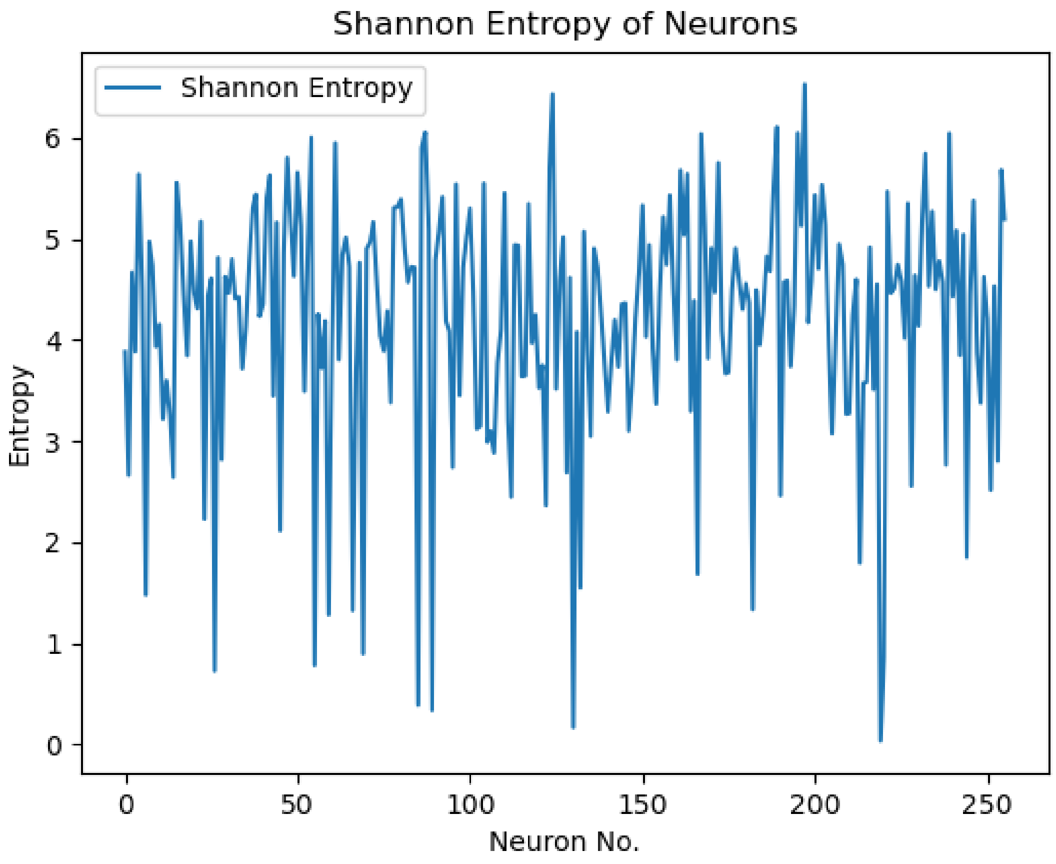


Figure 7

Tanh Activation Function - layer neuron entropy

Rectified Linear Unit (ReLU)

$$\mathrm{ReLU}(x) = \max(0, x)$$

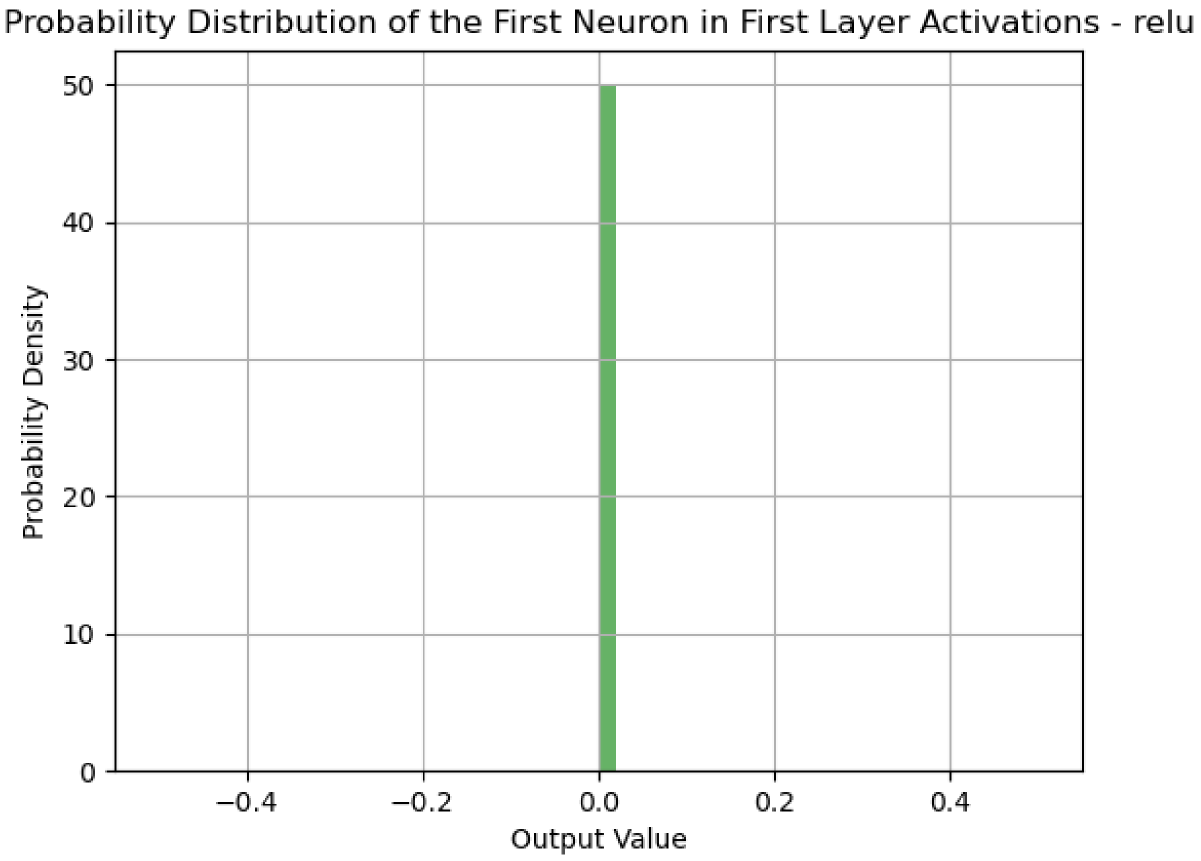


Figure 8

ReLU Activation Function - first neuron output distribution

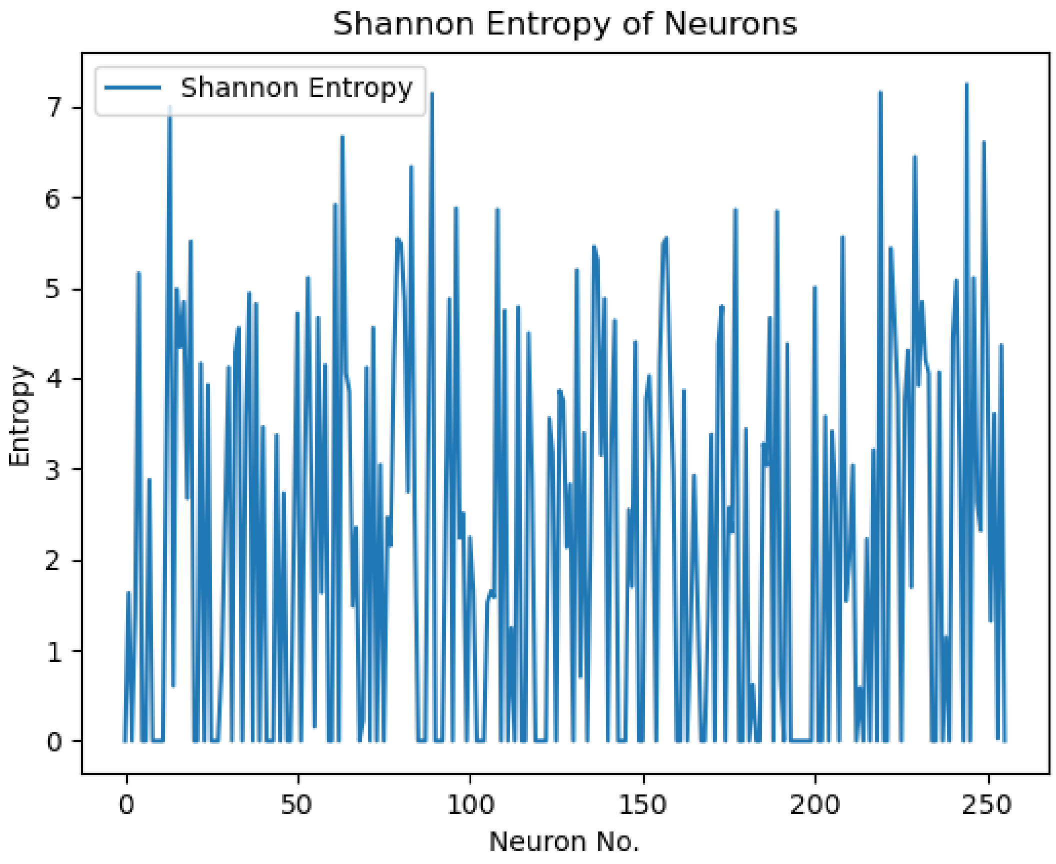


Figure 9

ReLU Activation Function - layer neuron entropy

Leaky ReLU

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases}$$

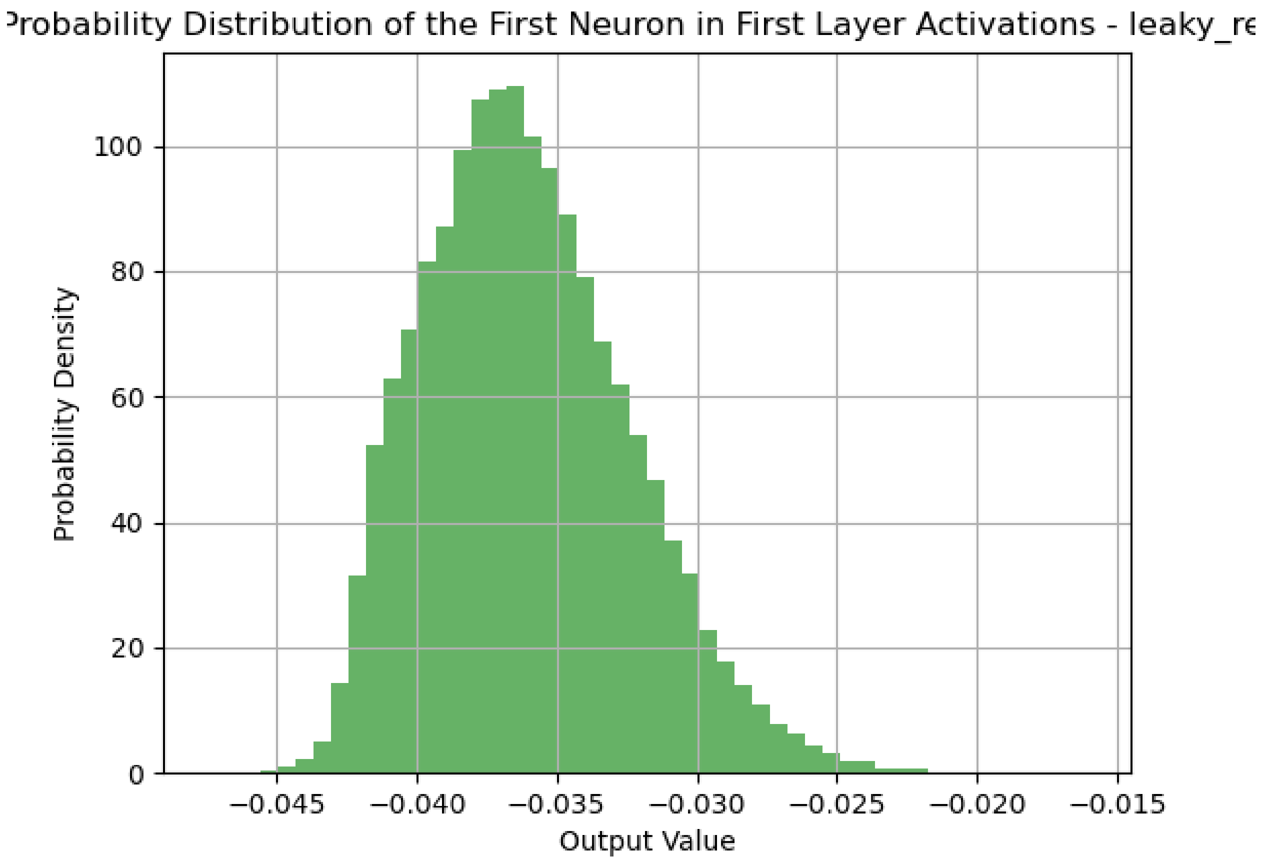


Figure 10

Leaky ReLU Activation Function - first neuron output distribution

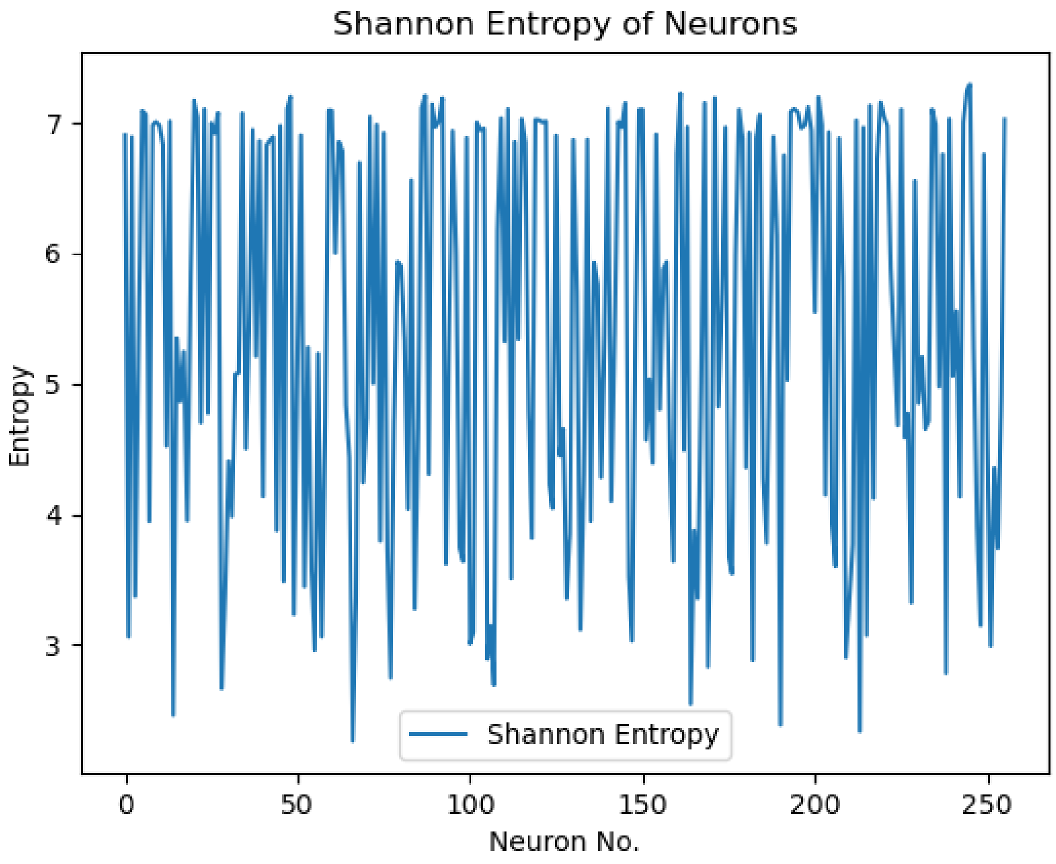


Figure 11

Leaky ReLU Activation Function - layer neuron entropy

Parametric ReLU (PReLU)

$$\text{PReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases}$$

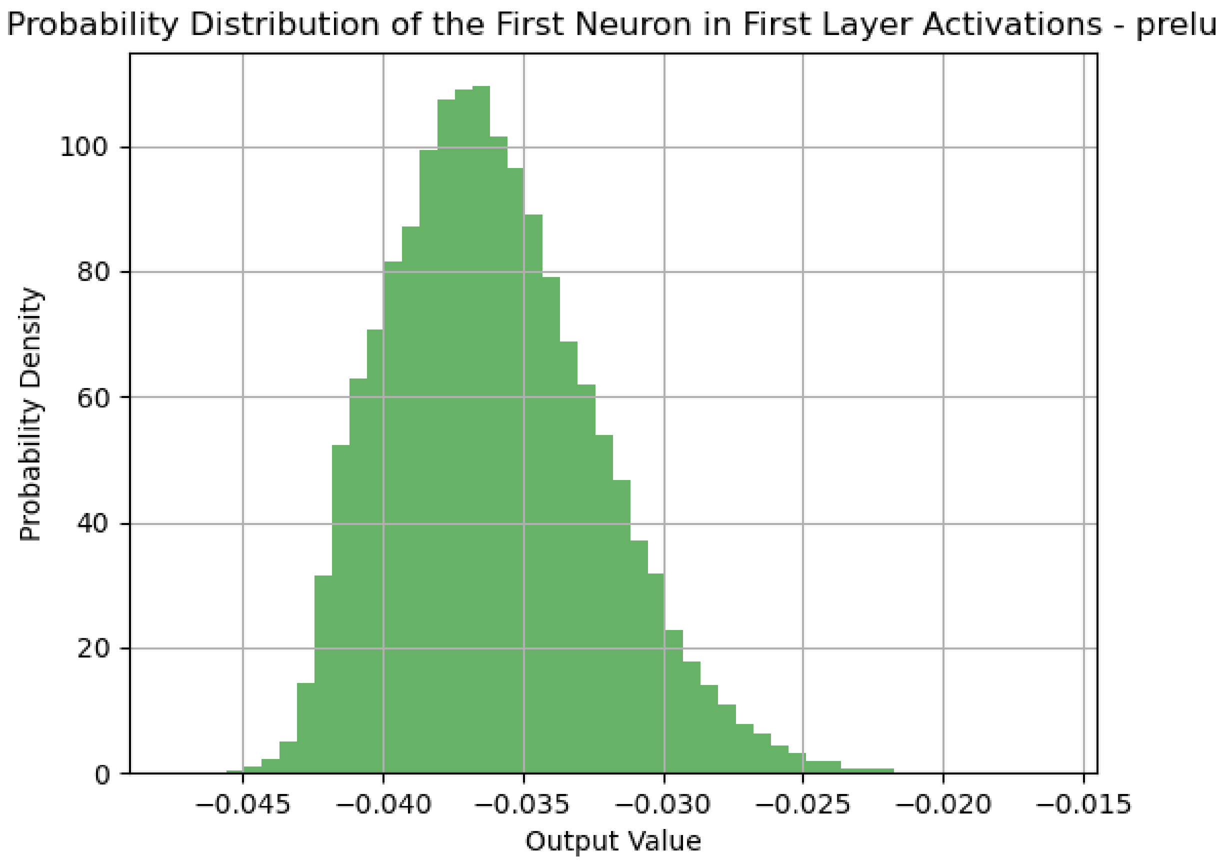


Figure 12

PReLU Activation Function - first neuron output distribution

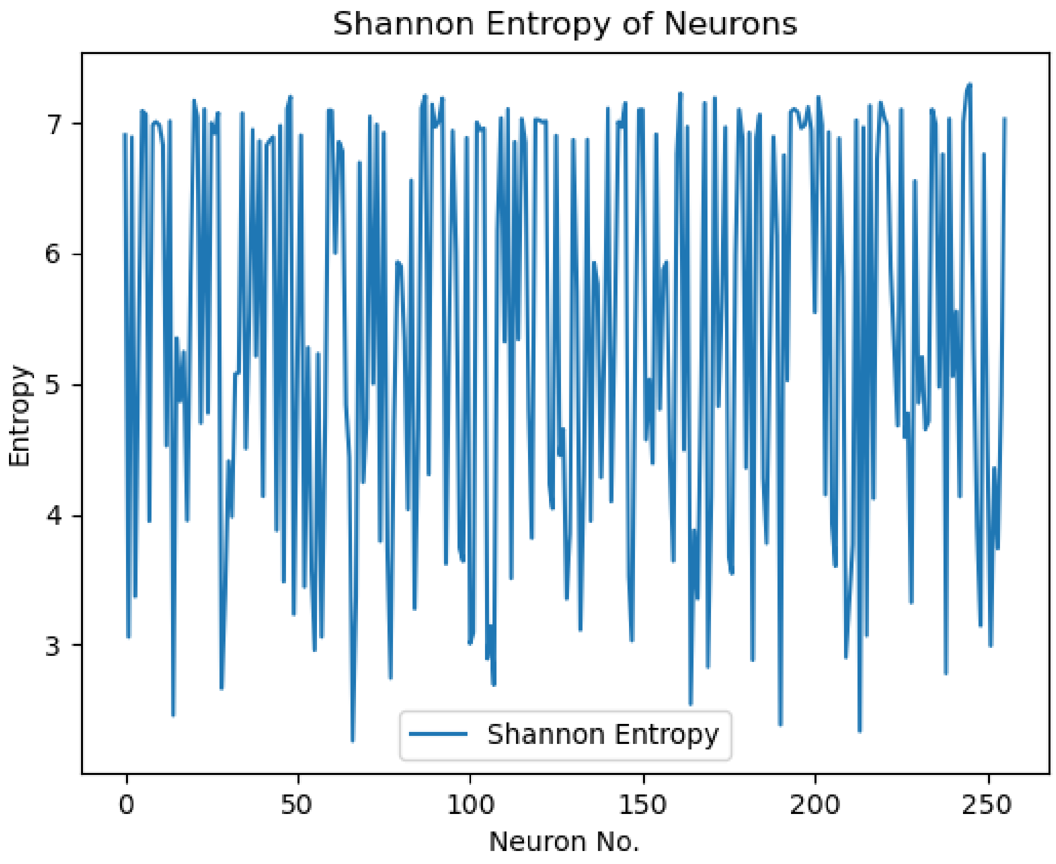


Figure 13

PReLU Activation Function - layer neuron entropy

Exponential Linear Unit (ELU)

$$\text{ELU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases}$$

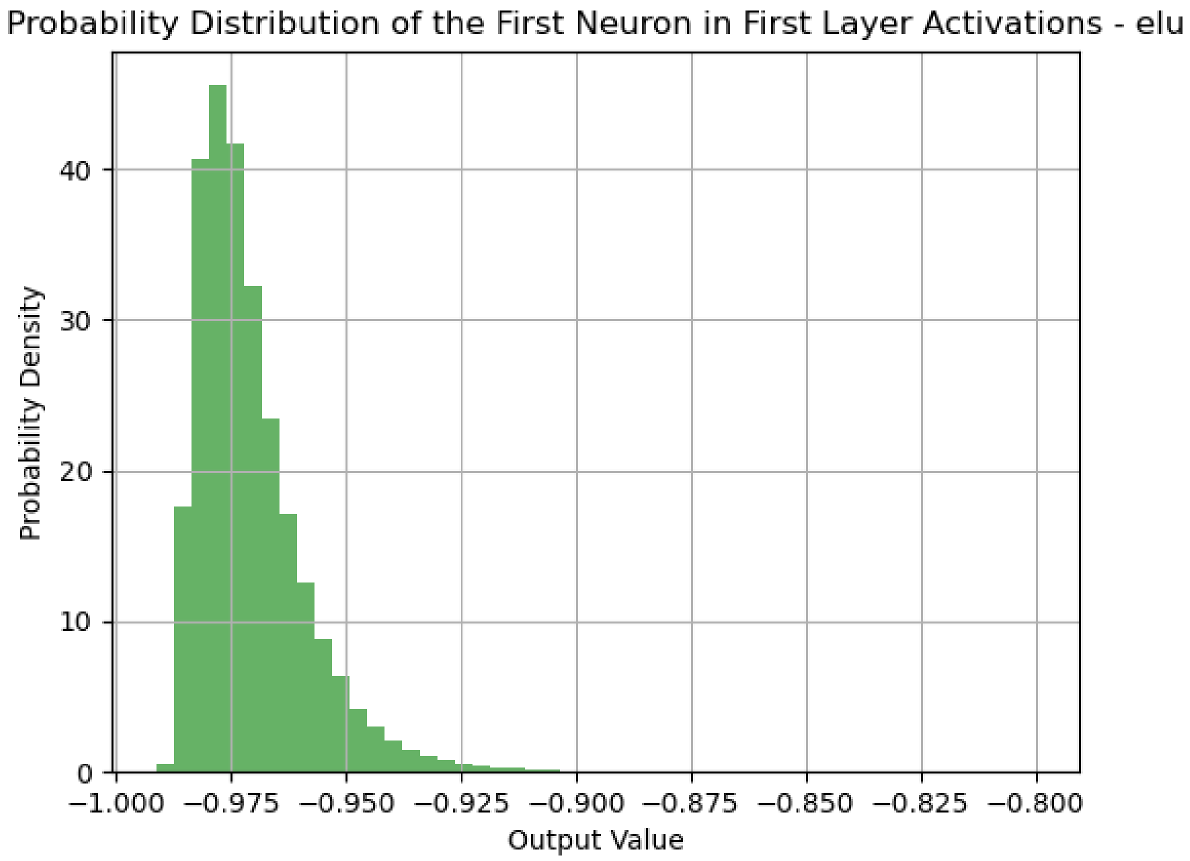


Figure 14

ELU Activation Function - first neuron output distribution

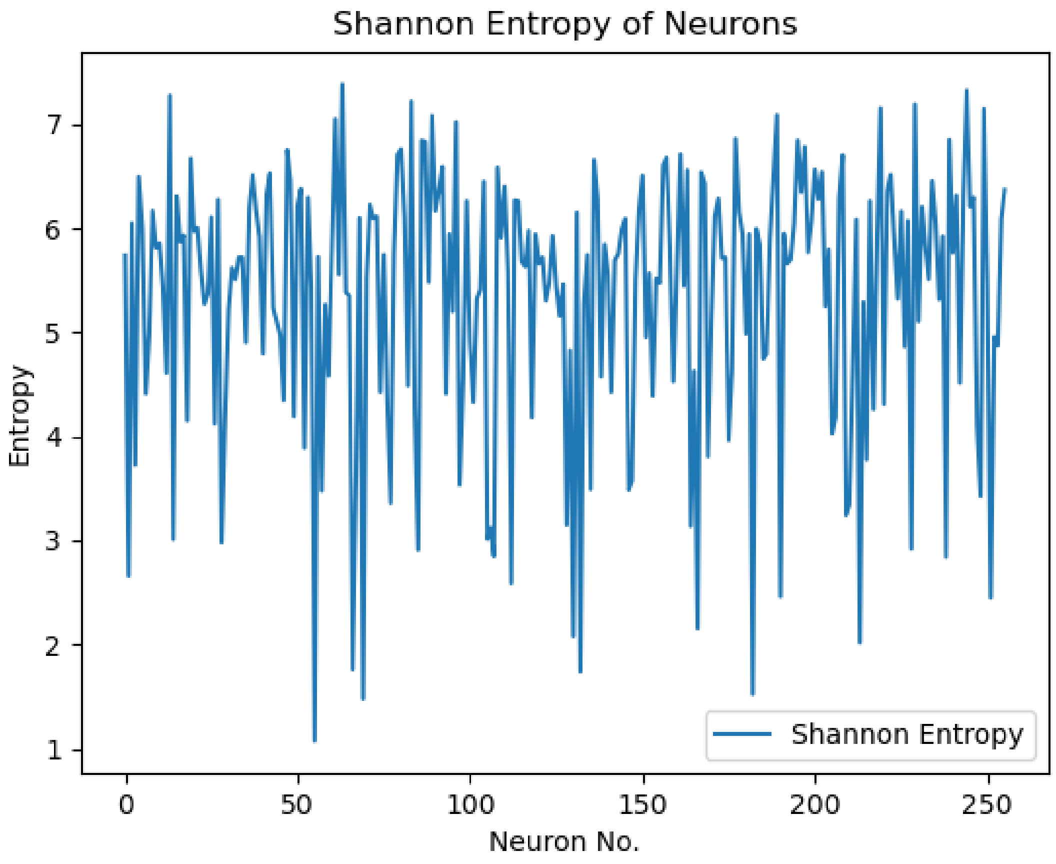


Figure 15

ELU Activation Function - layer neuron entropy

Scaled Exponential Linear Unit (SELU)

$$\text{SELU}(x) = \lambda \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases}$$

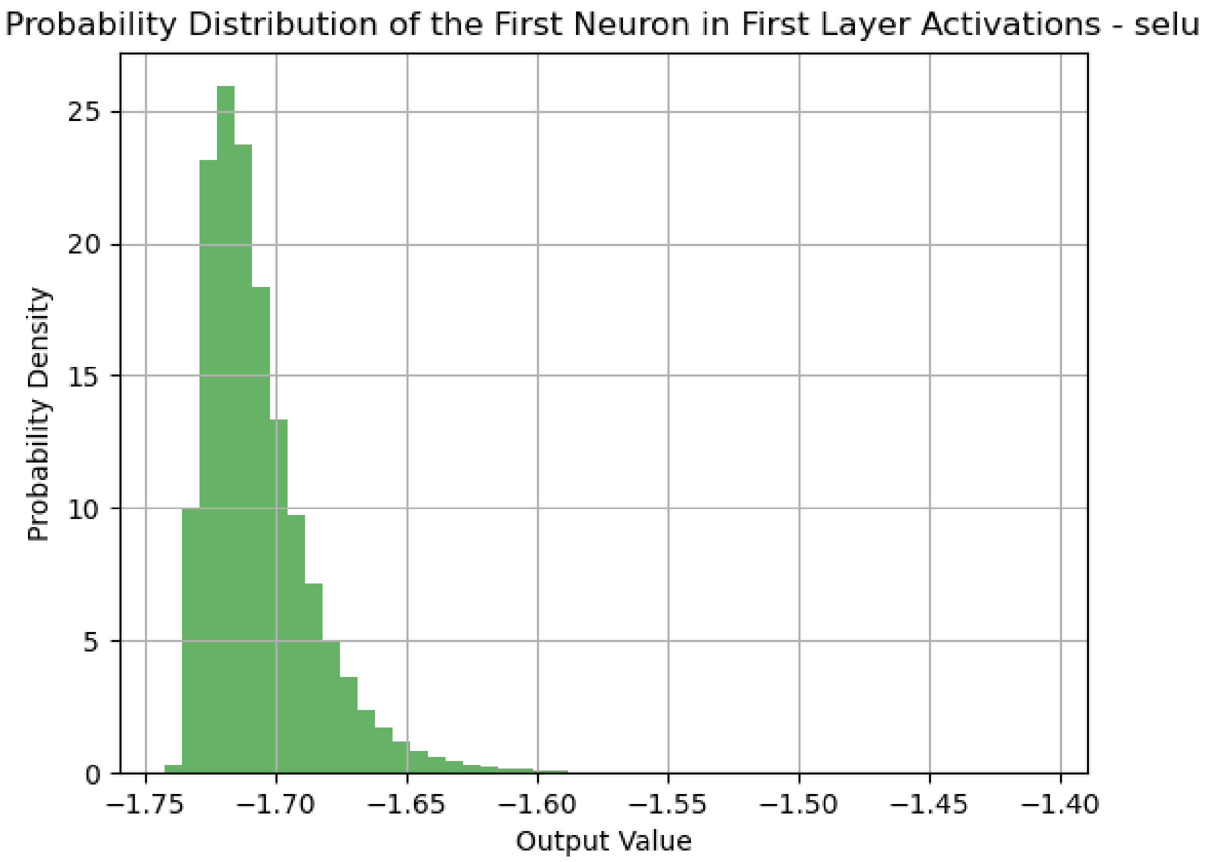


Figure 16

SELU Activation Function - first neuron output distribution

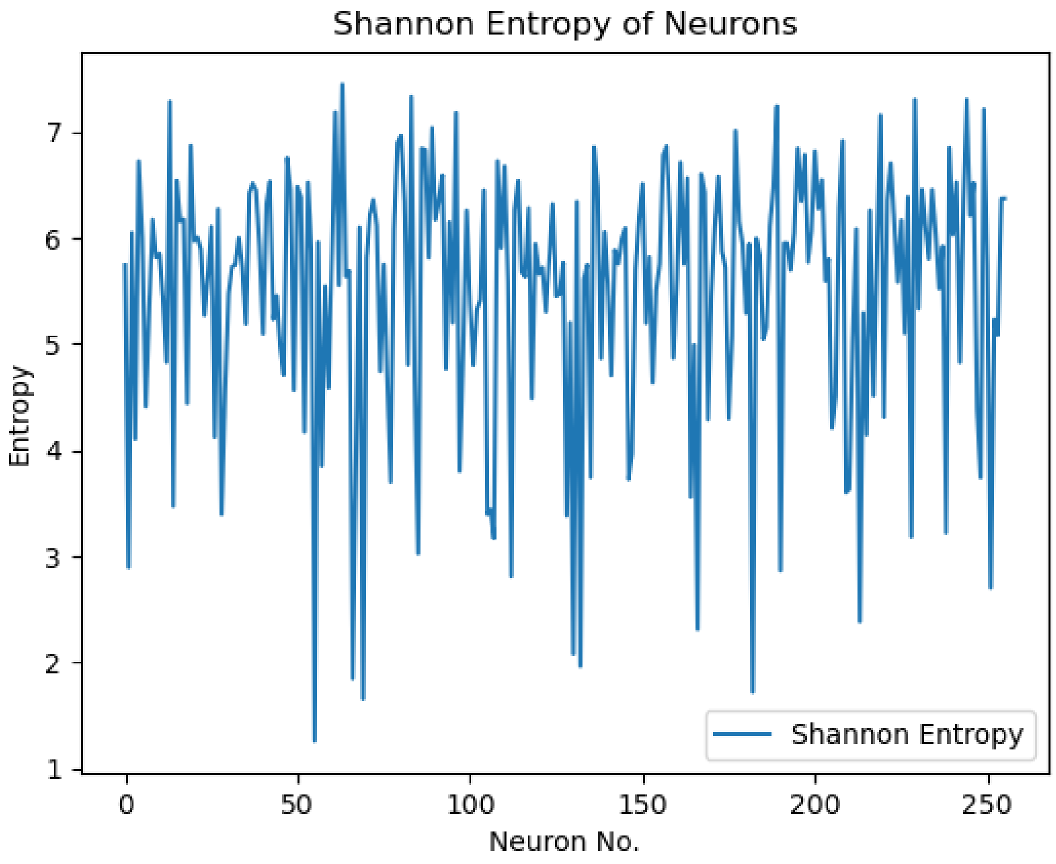


Figure 17

SELU Activation Function - layer neuron entropy

Softplus

$$\text{Softplus}(x) = \ln(1 + e^x)$$

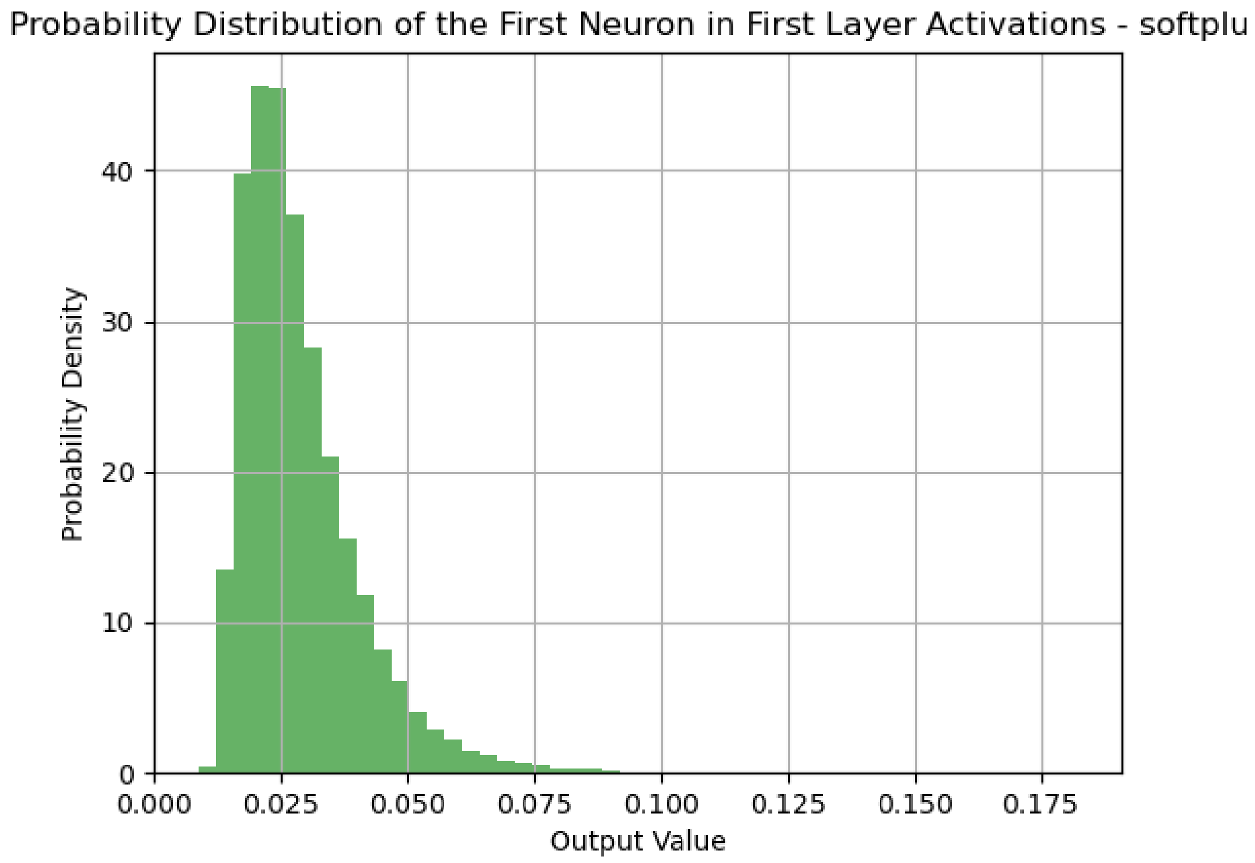


Figure 18

Softplus Activation Function - first neuron output distribution

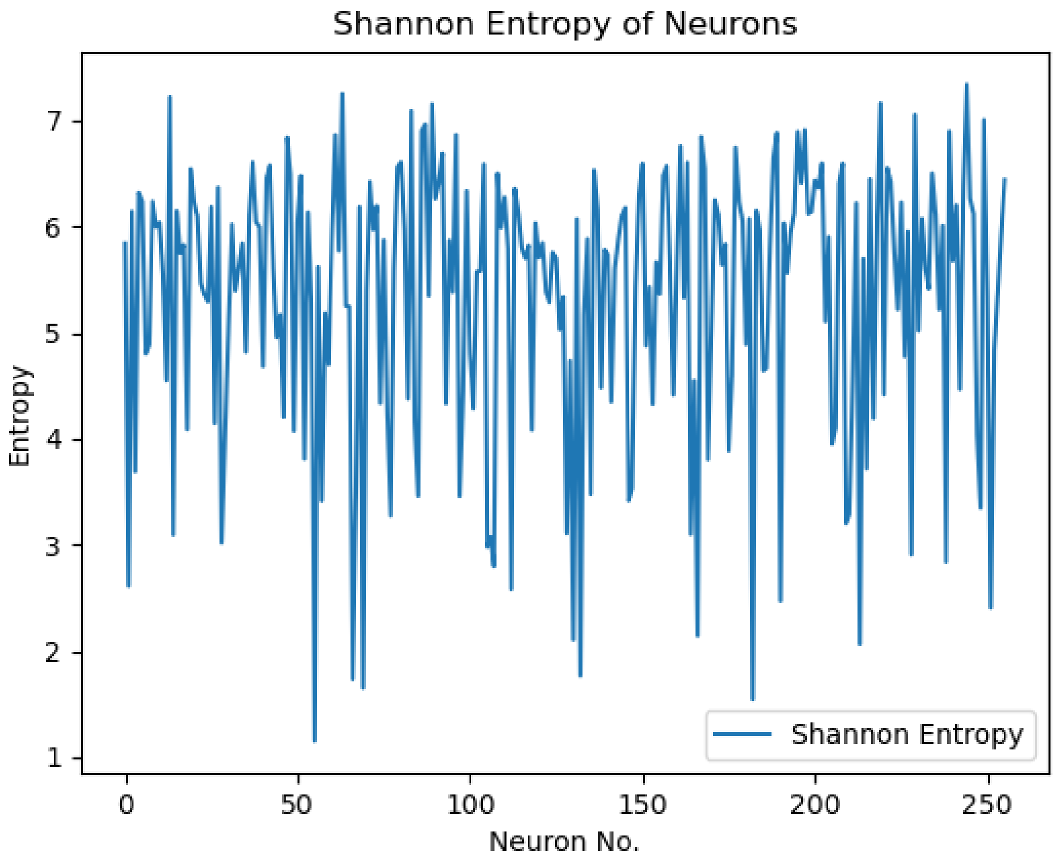


Figure 19

Softplus Activation Function - layer neuron entropy

Swish

$$\text{Swish}(x) = x \cdot \sigma(x) = x \cdot \frac{1}{1+e^{-x}}$$

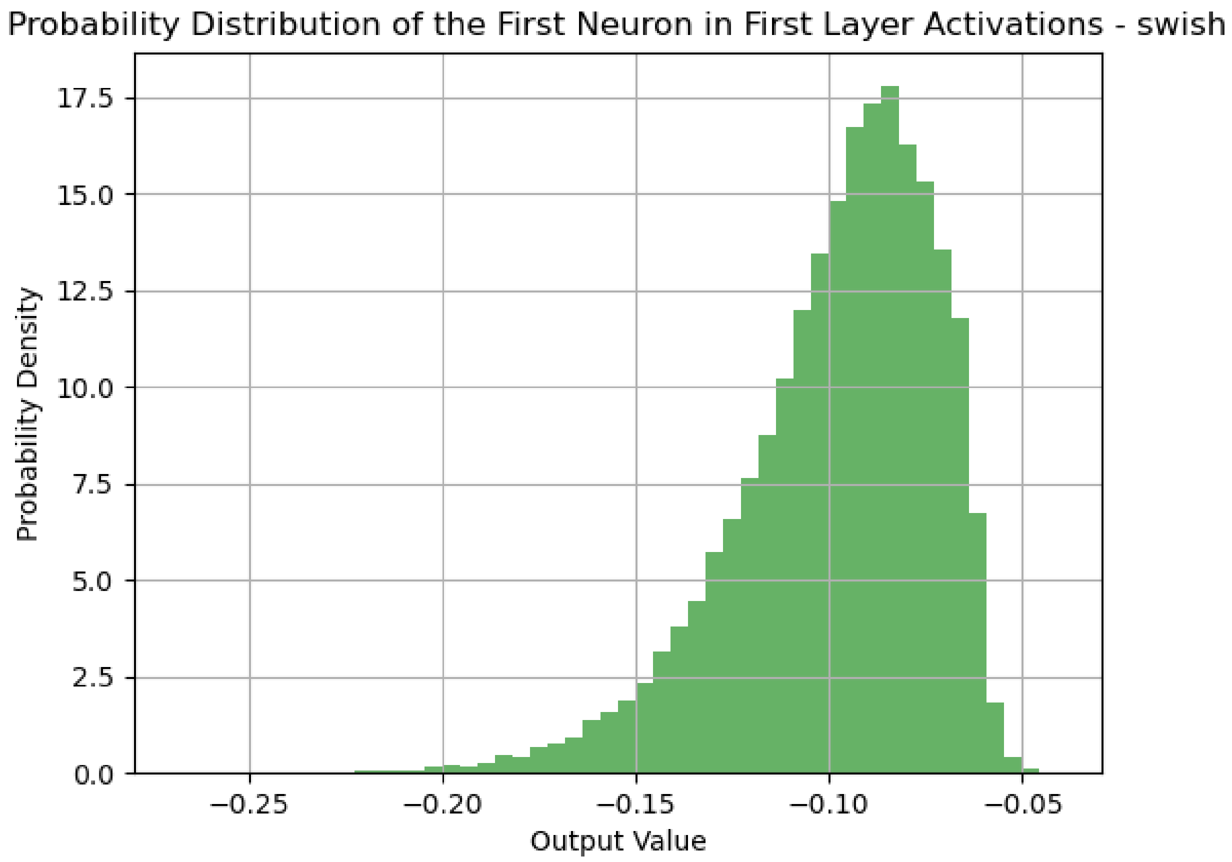


Figure 20

Swish Activation Function - first neuron output distribution

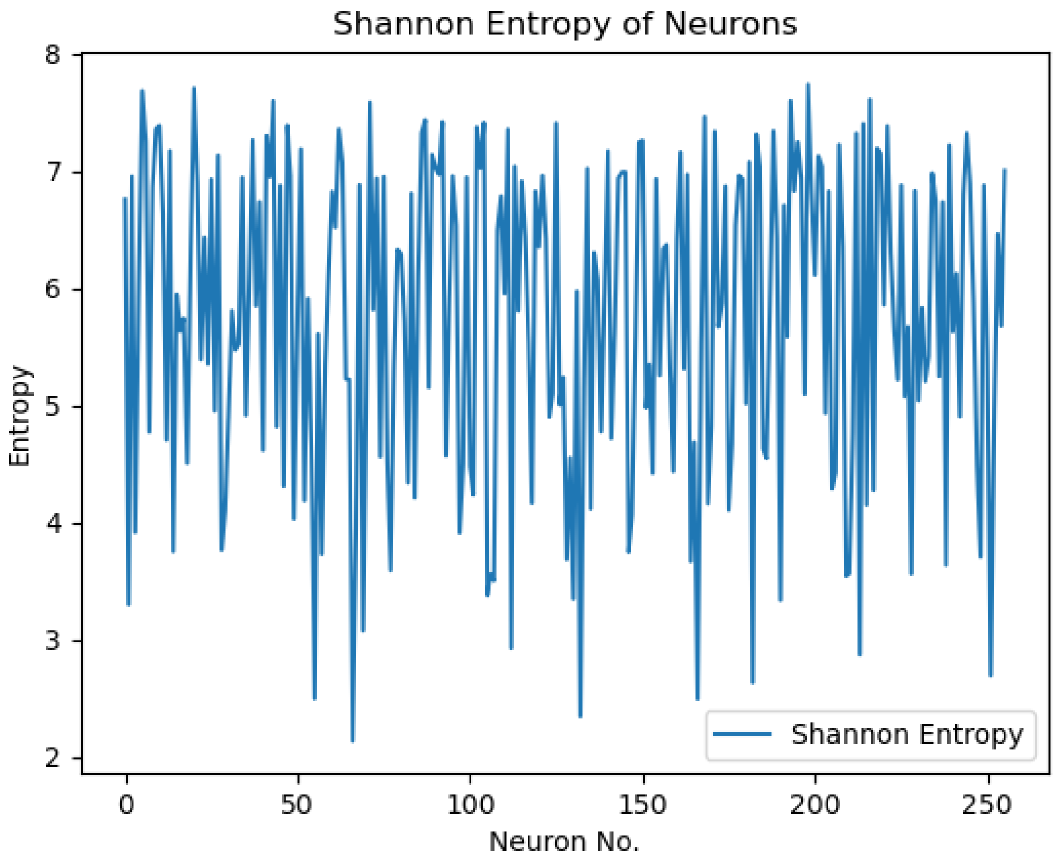


Figure 21

Swish Activation Function - layer neuron entropy

Gaussian Error Linear Unit (GELU)

$$\text{GELU}(x) = x \cdot \Phi(x)$$

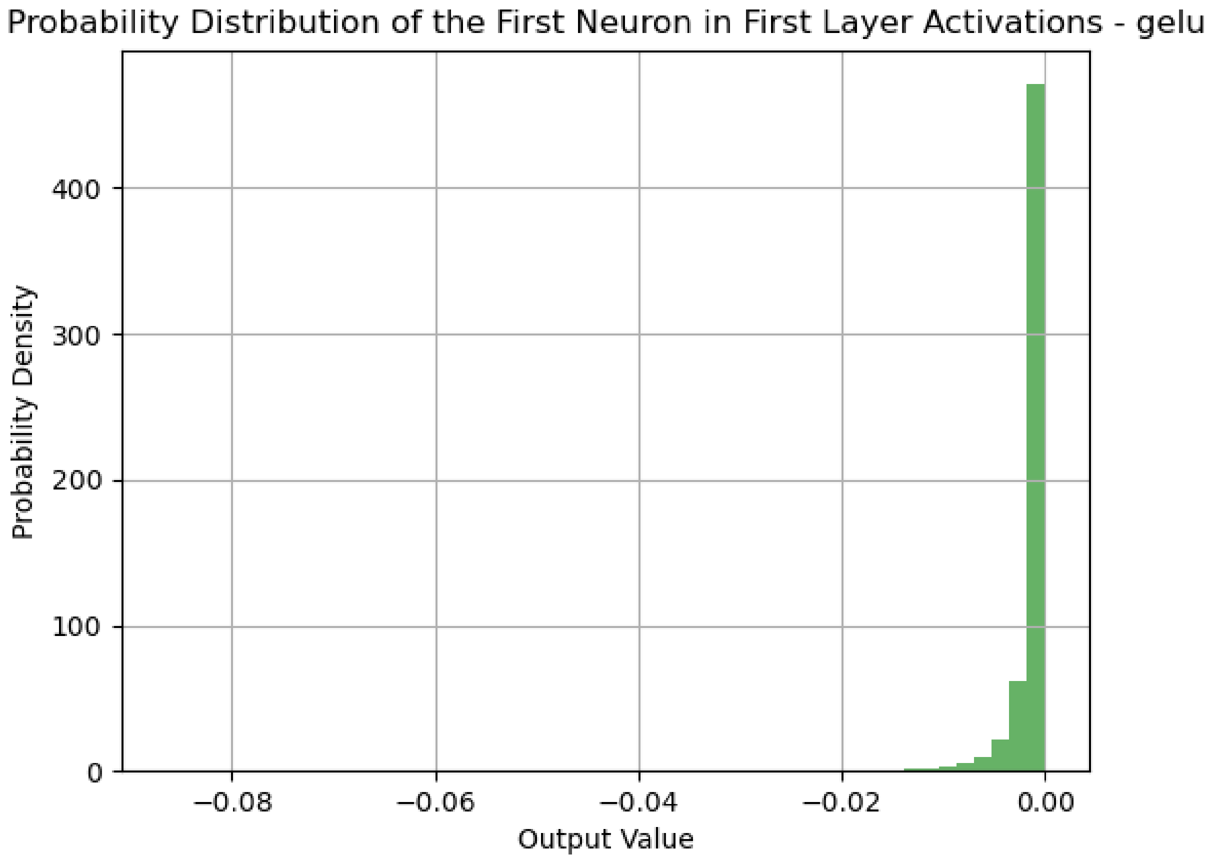


Figure 22

GELU Activation Function - first neuron output distribution

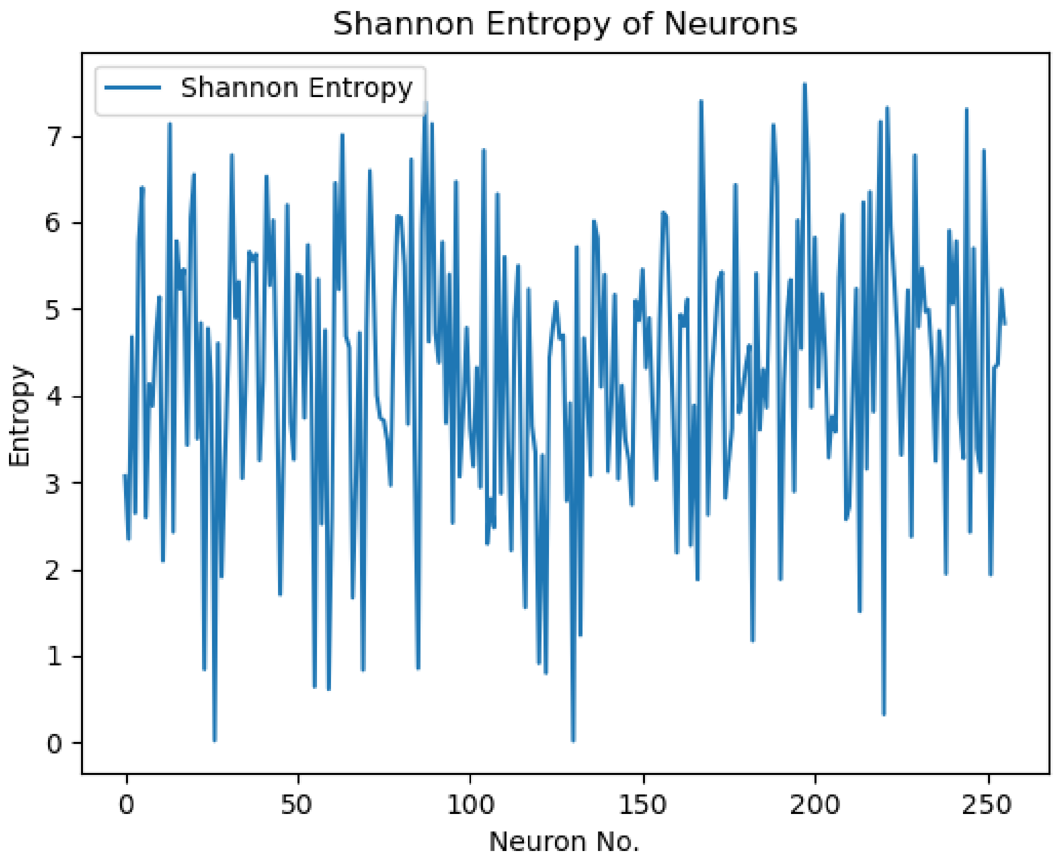


Figure 23

GELU Activation Function - layer neuron entropy

Mish

$$\text{Mish}(x) = x \cdot \tanh(\ln(1 + e^x))$$

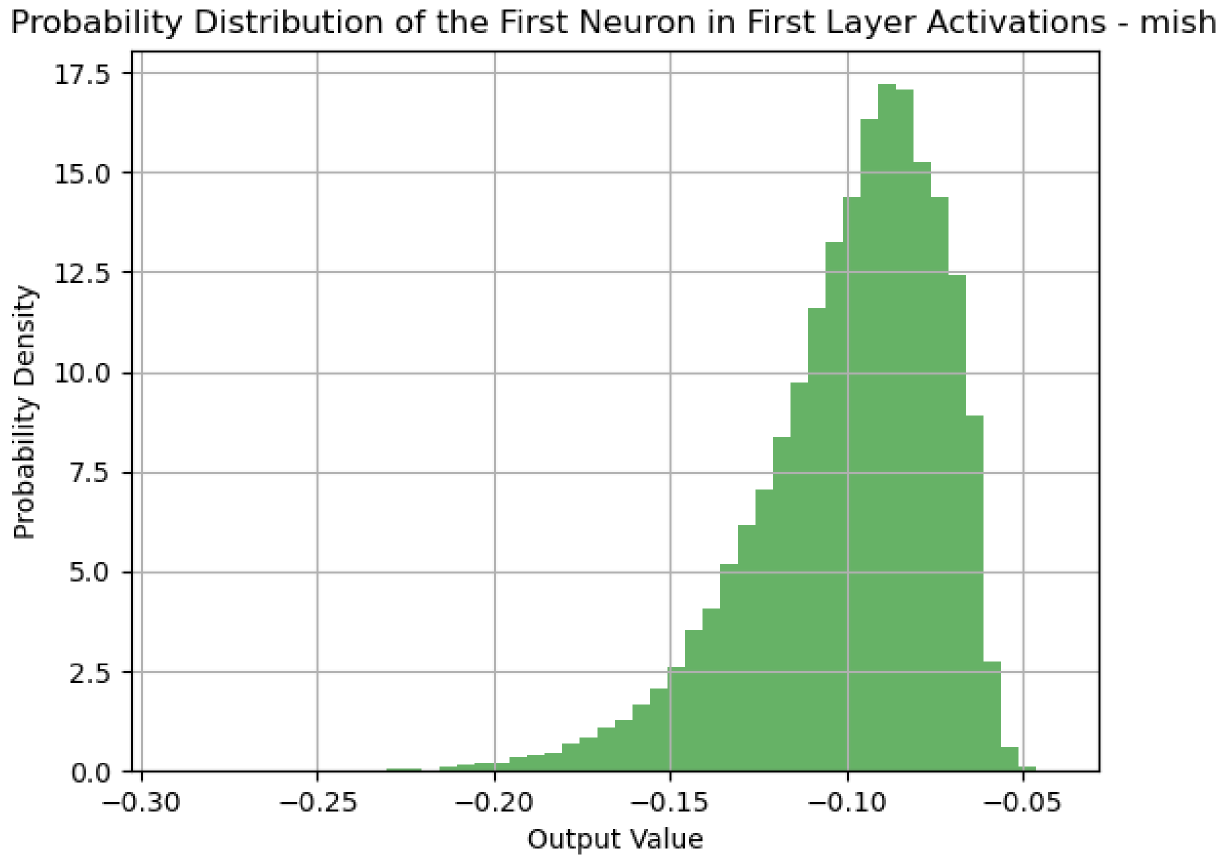


Figure 24

Mish Activation Function - first neuron output distribution

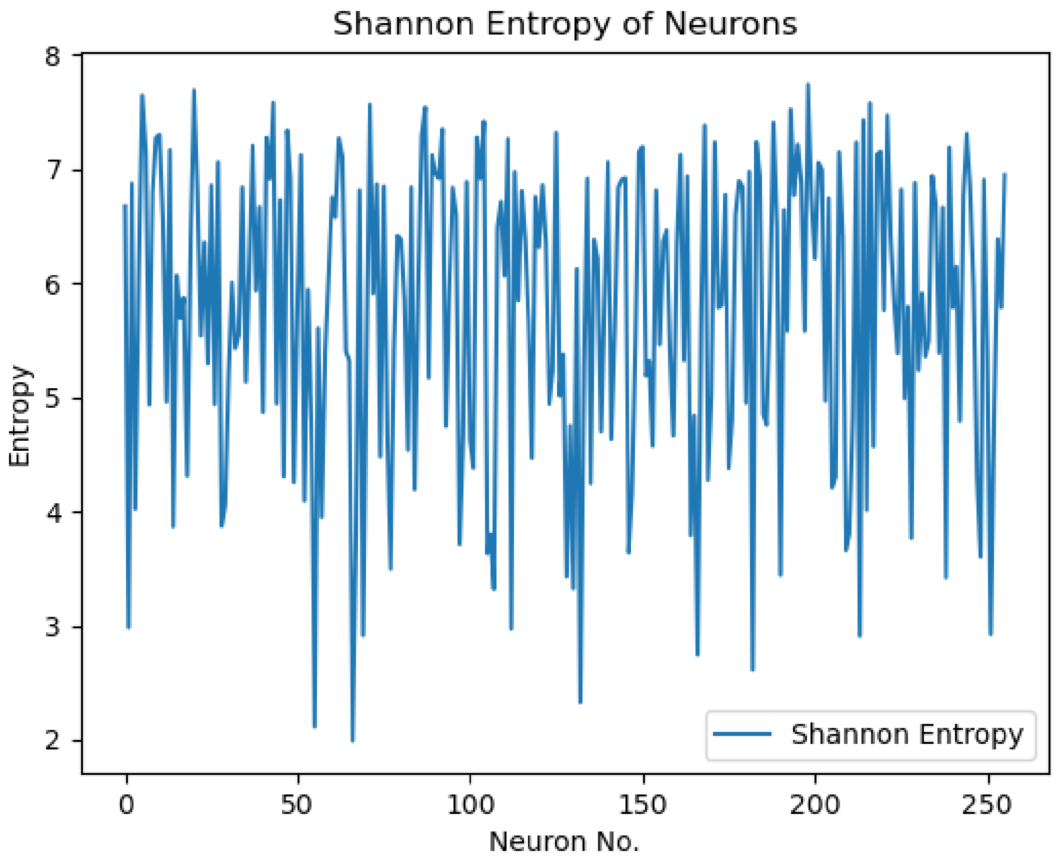


Figure 25

Mish Activation Function - layer neuron entropy

Hard Sigmoid

$$\text{Hard Sigmoid}(x) = \begin{cases} 0 & \text{if } x \leq -2.5 \\ 1 & \text{if } x \geq 2.5 \\ 0.2x + 0.5 & \text{otherwise} \end{cases}$$

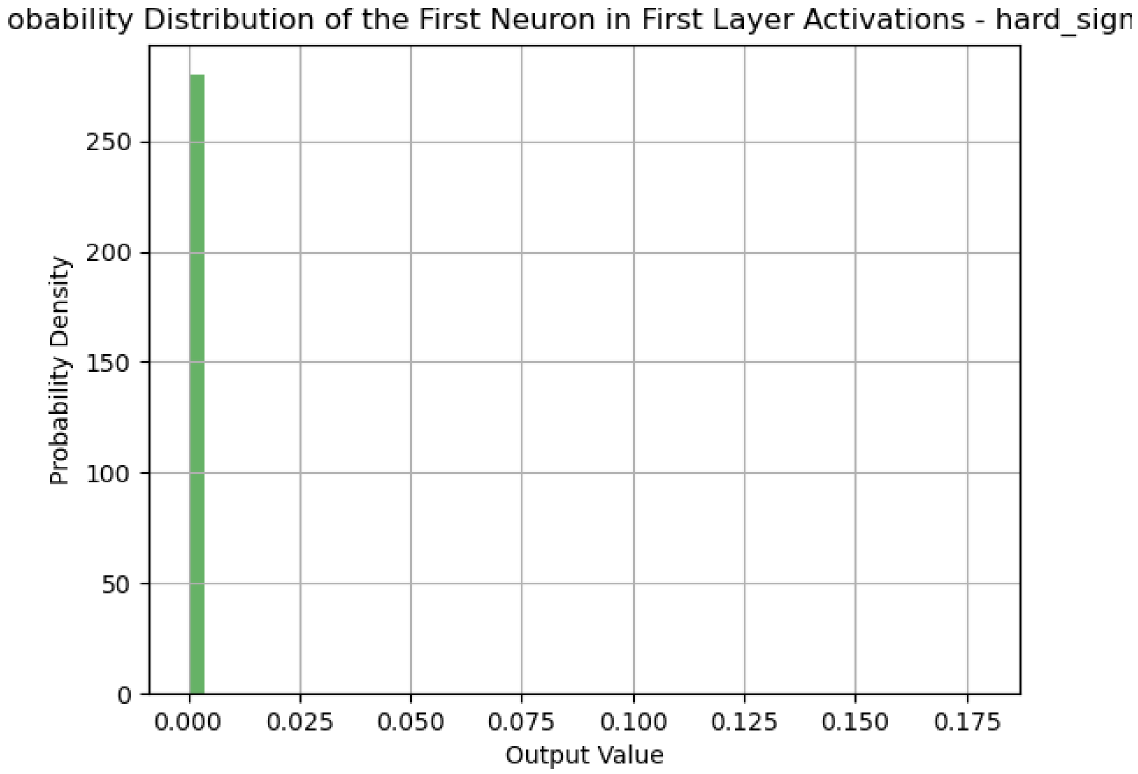


Figure 26

Hard Sigmoid Activation Function - first neuron output distribution

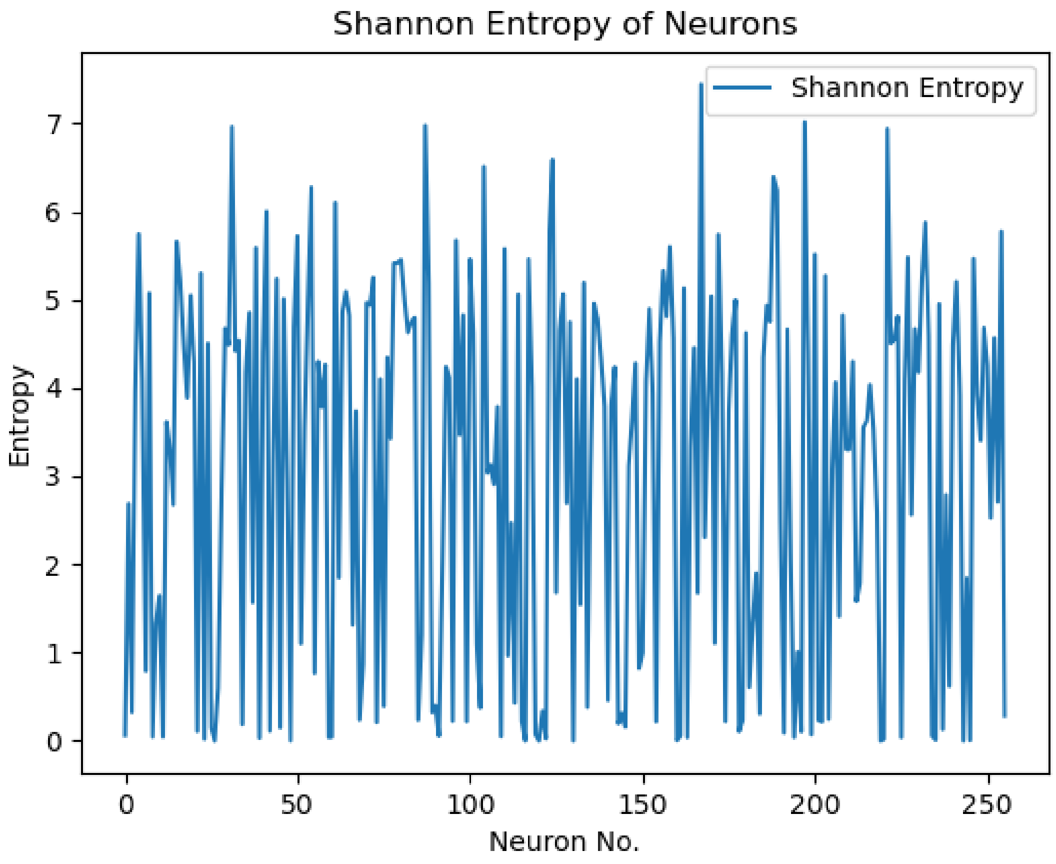


Figure 27

Hard Sigmoid Activation Function - layer neuron entropy

Hard Swish

$$\text{Hard Swish}(x) = x \cdot \text{Hard Sigmoid}(x)$$

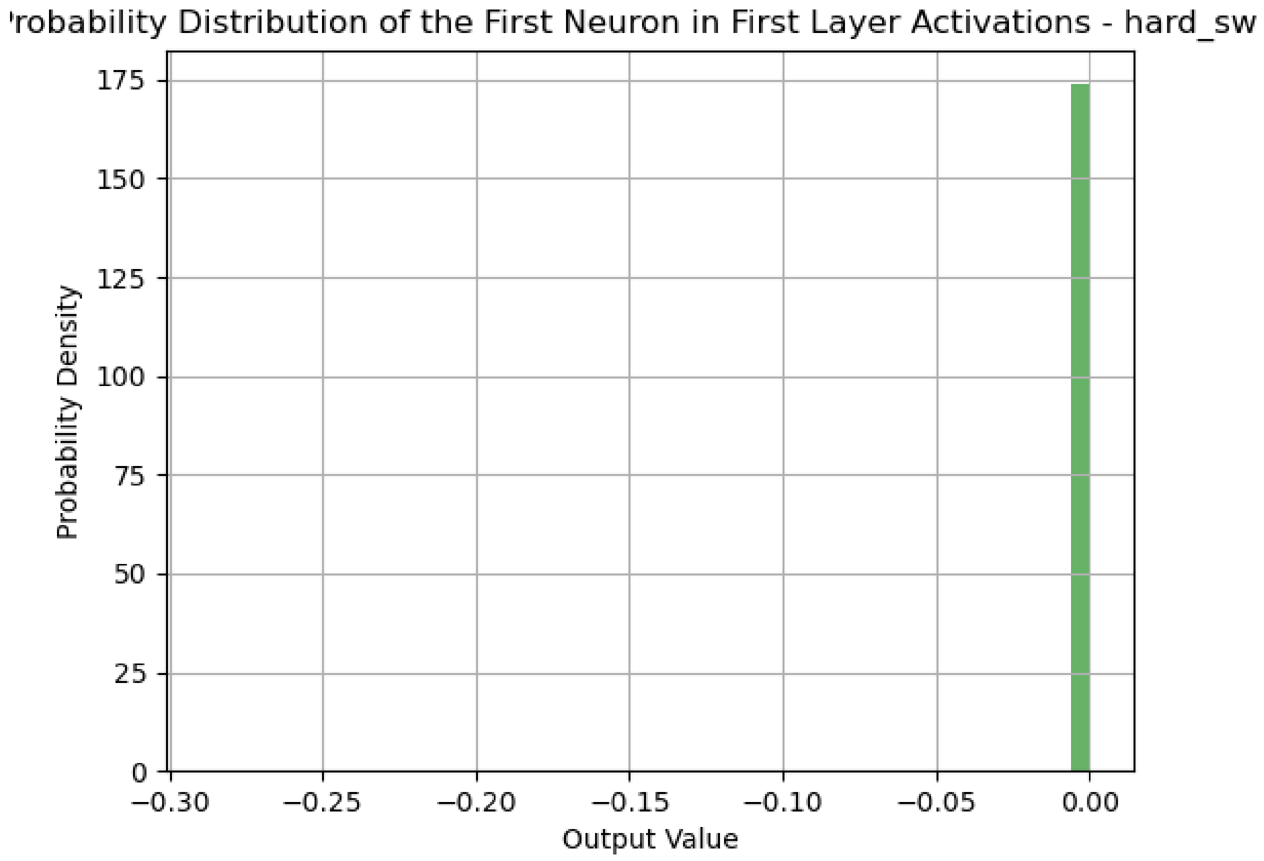


Figure 28

Hard Swish Activation Function - first neuron output distribution

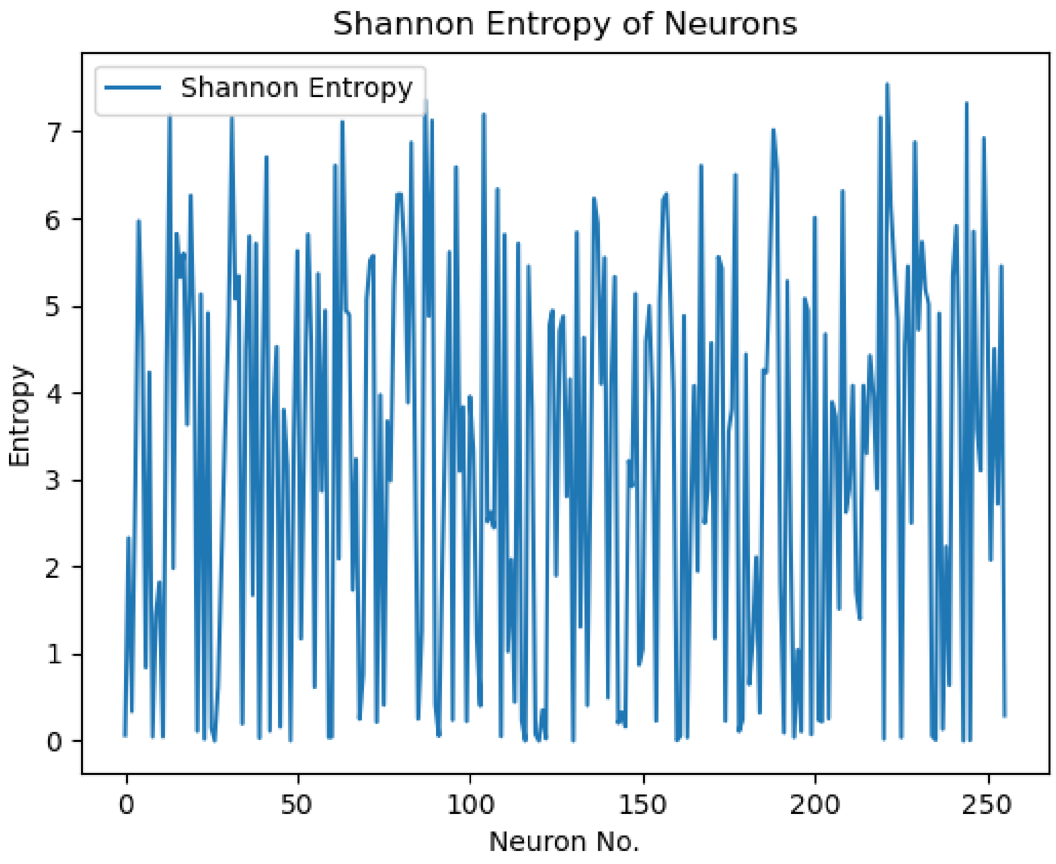


Figure 29

Hard Swish Activation Function - layer neuron entropy

Binary Step

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

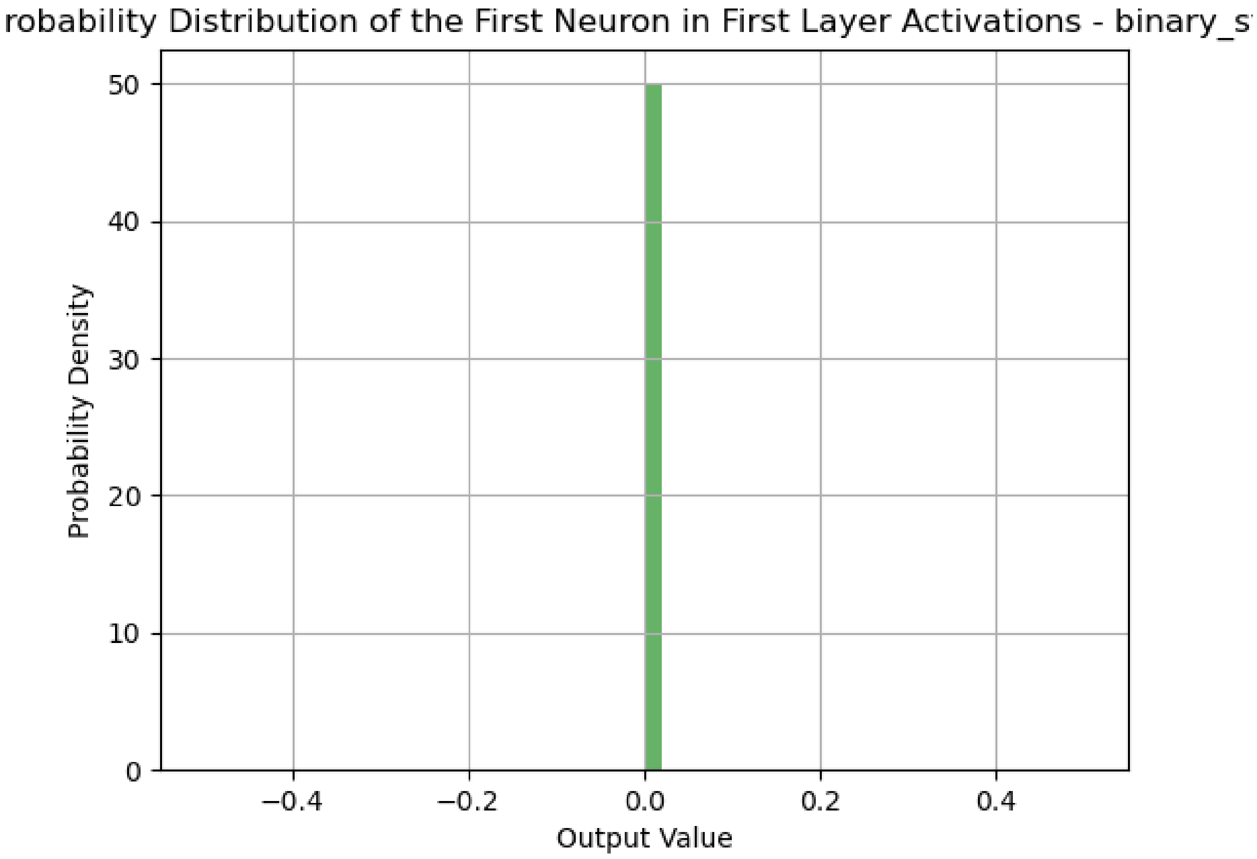


Figure 30

Binary Step Activation Function - first neuron output distribution

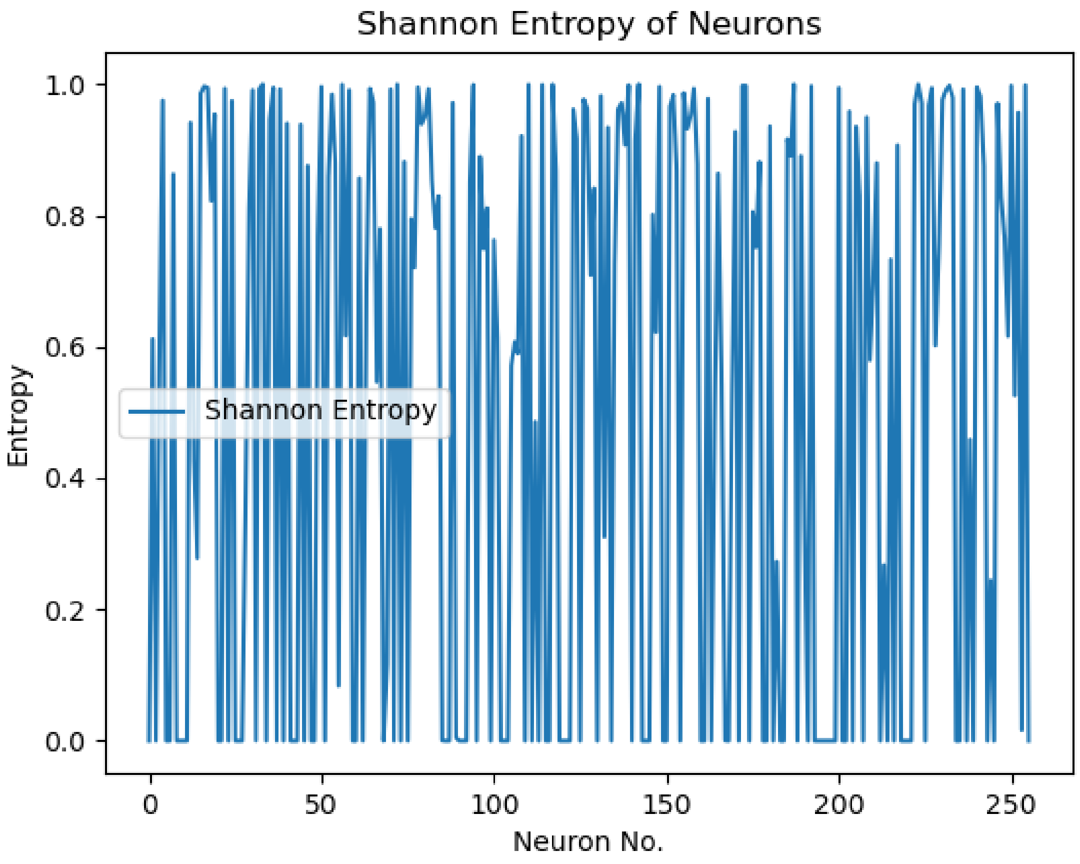


Figure 31

Binary Step Activation Function - layer neuron entropy

Thresholded ReLU

$$\text{Thresholded ReLU}(x) = \begin{cases} x & \text{if } x > \theta \\ 0 & \text{if } x \leq \theta \end{cases}$$

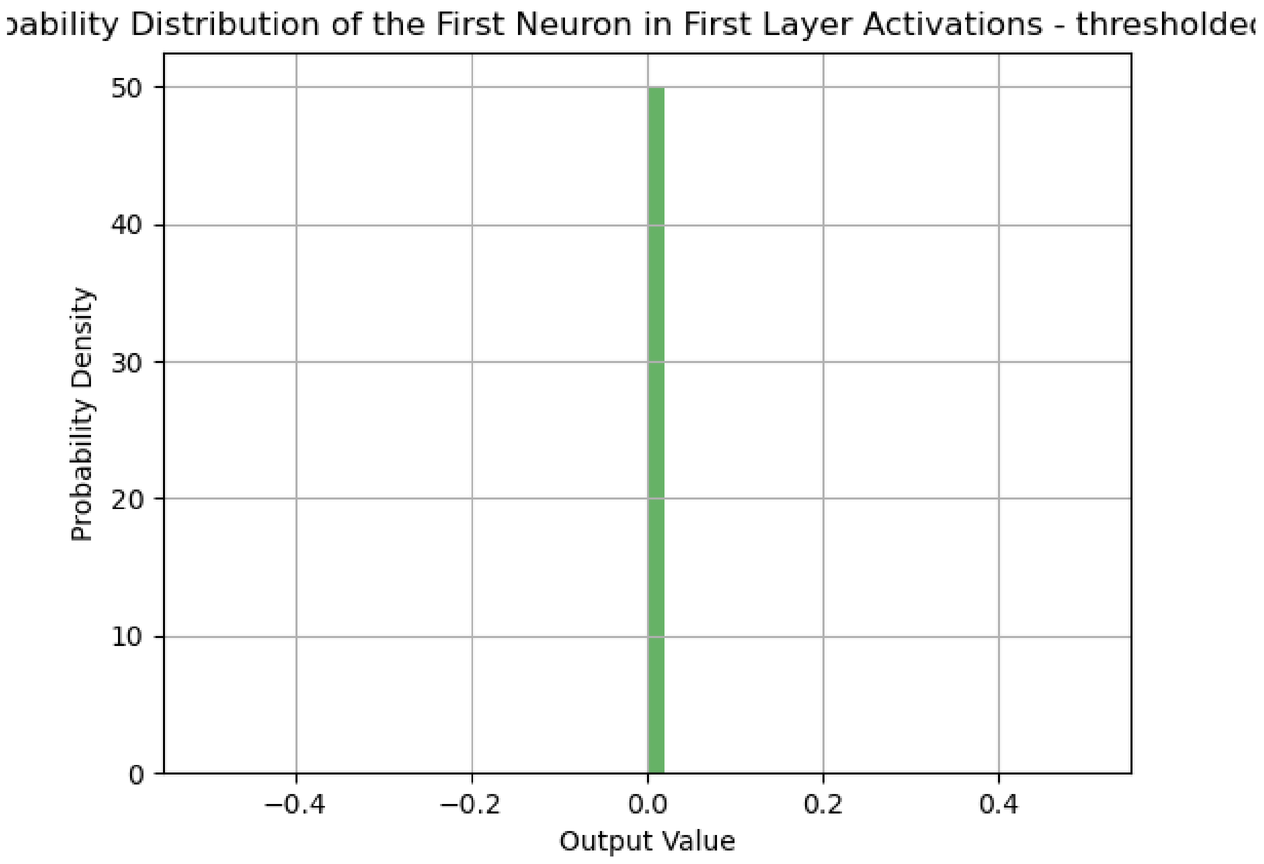


Figure 32

Thresholded ReLU Activation Function - first neuron output distribution

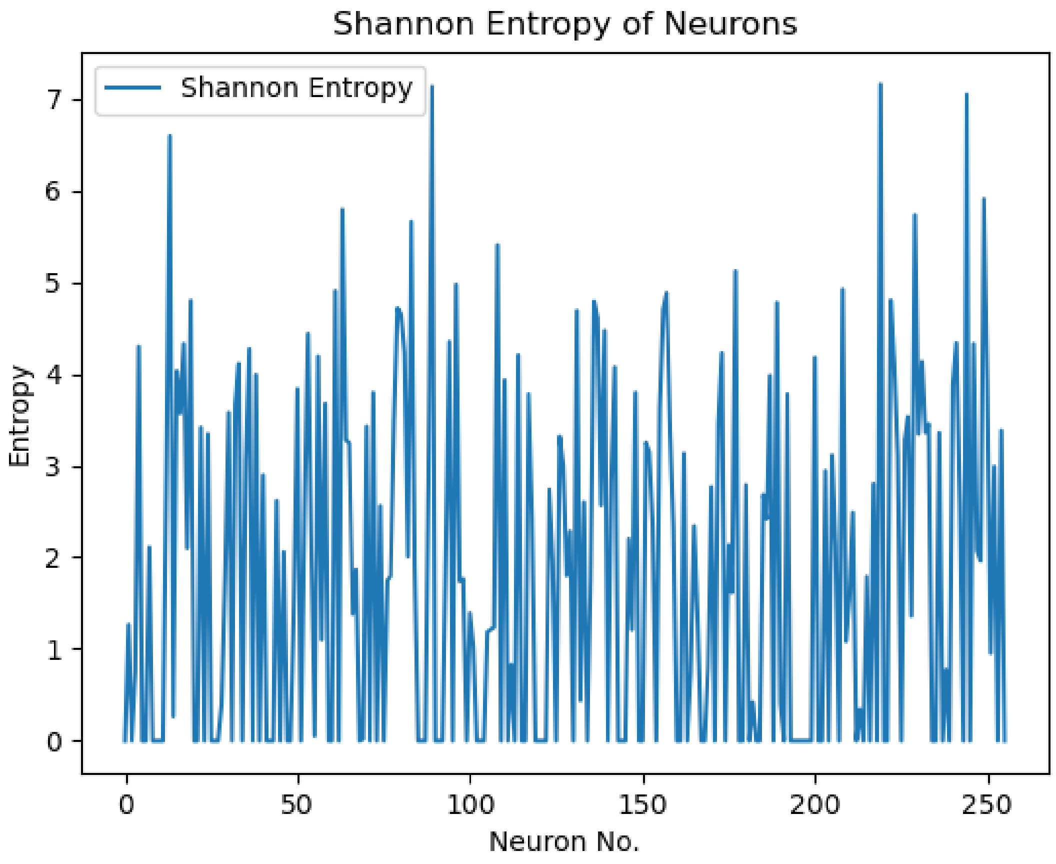


Figure 33

Thresholded ReLU Activation Function - layer neuron entropy

Sine Activation

$$\text{Sine}(x) = \sin(x)$$

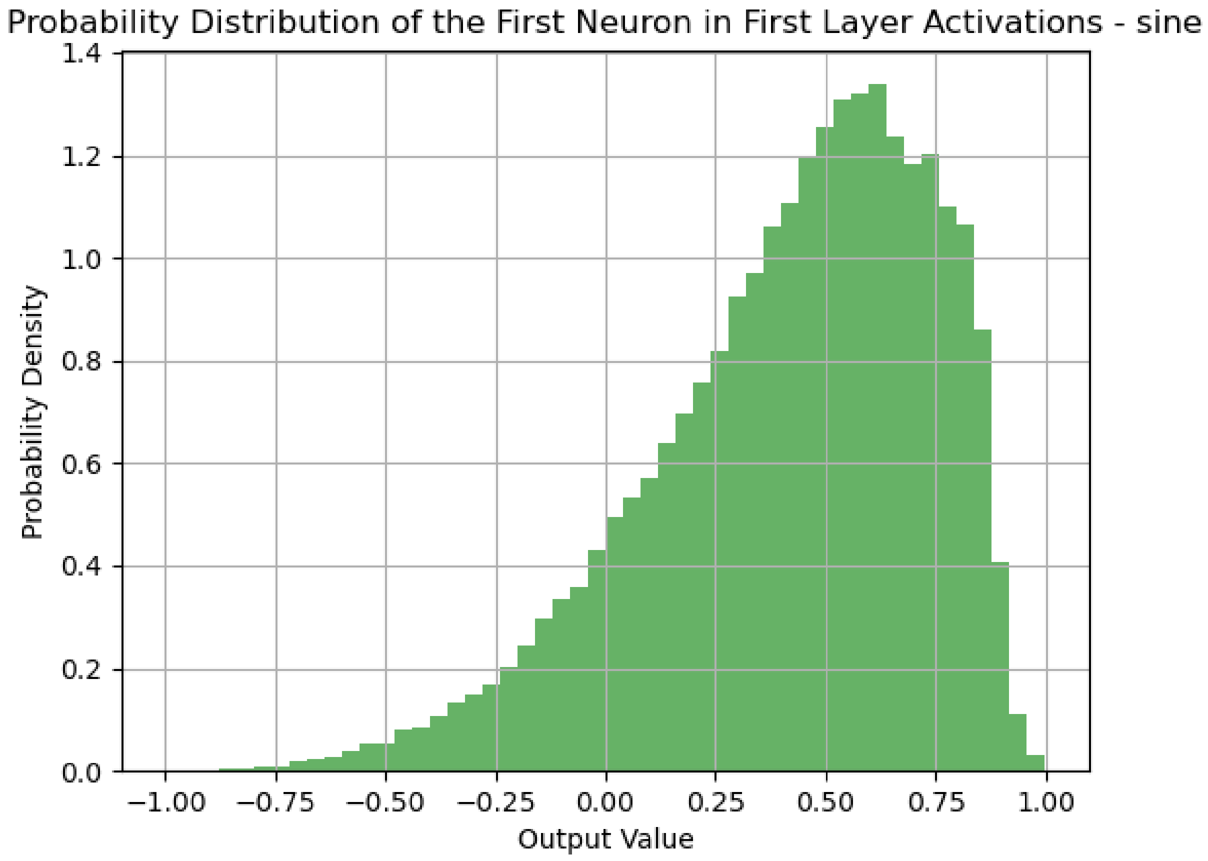


Figure 34

Sine Activation Function - first neuron output distribution

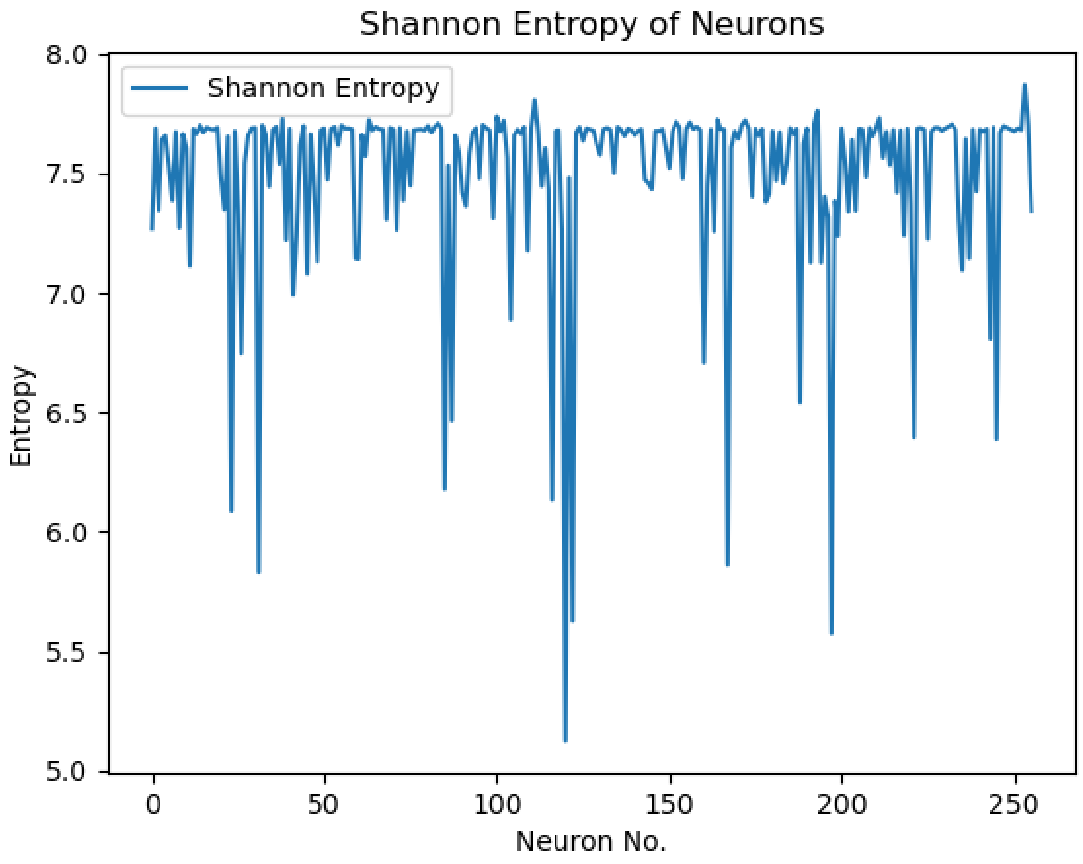


Figure 35

Sine Activation Function - layer neuron entropy

Cosine Activation

$$\text{Cosine}(x) = \cos(x)$$

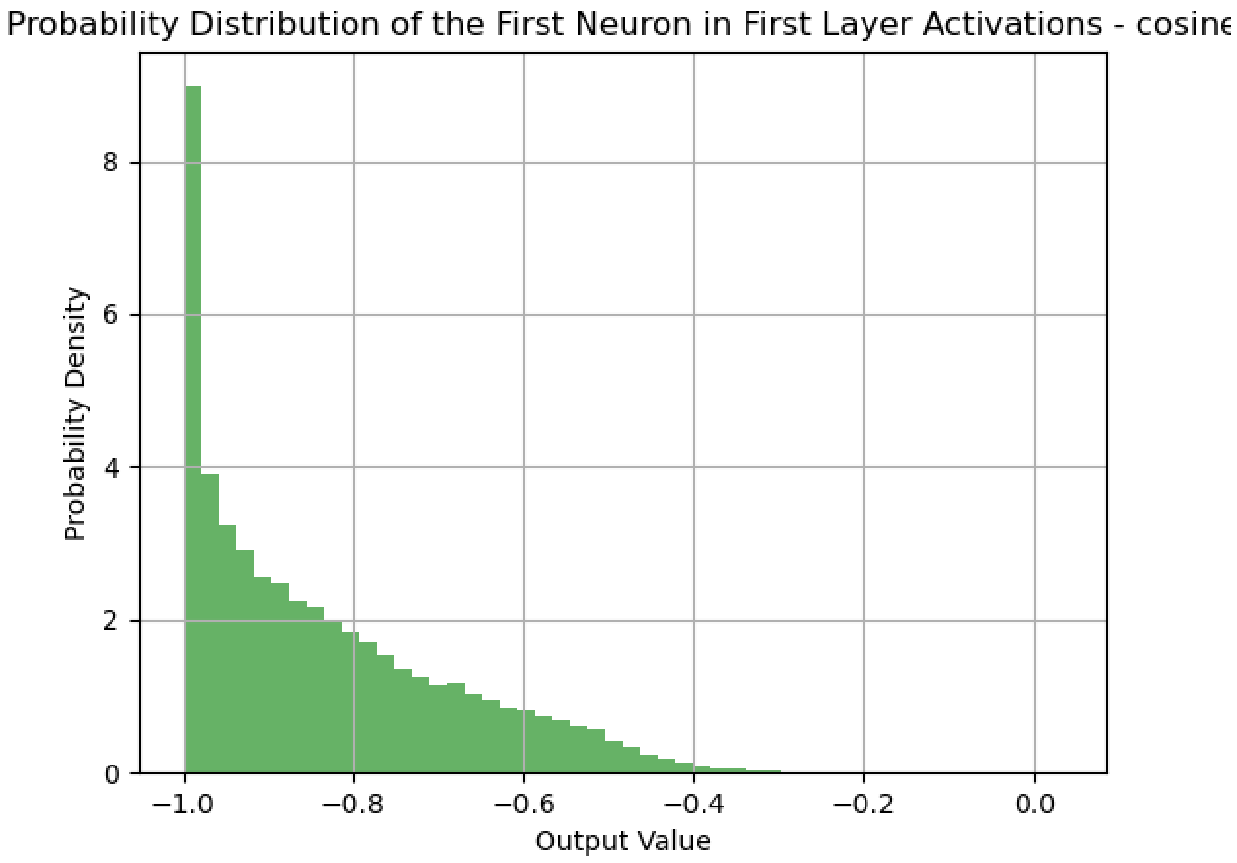


Figure 36

Cosine Activation Function - first neuron output distribution

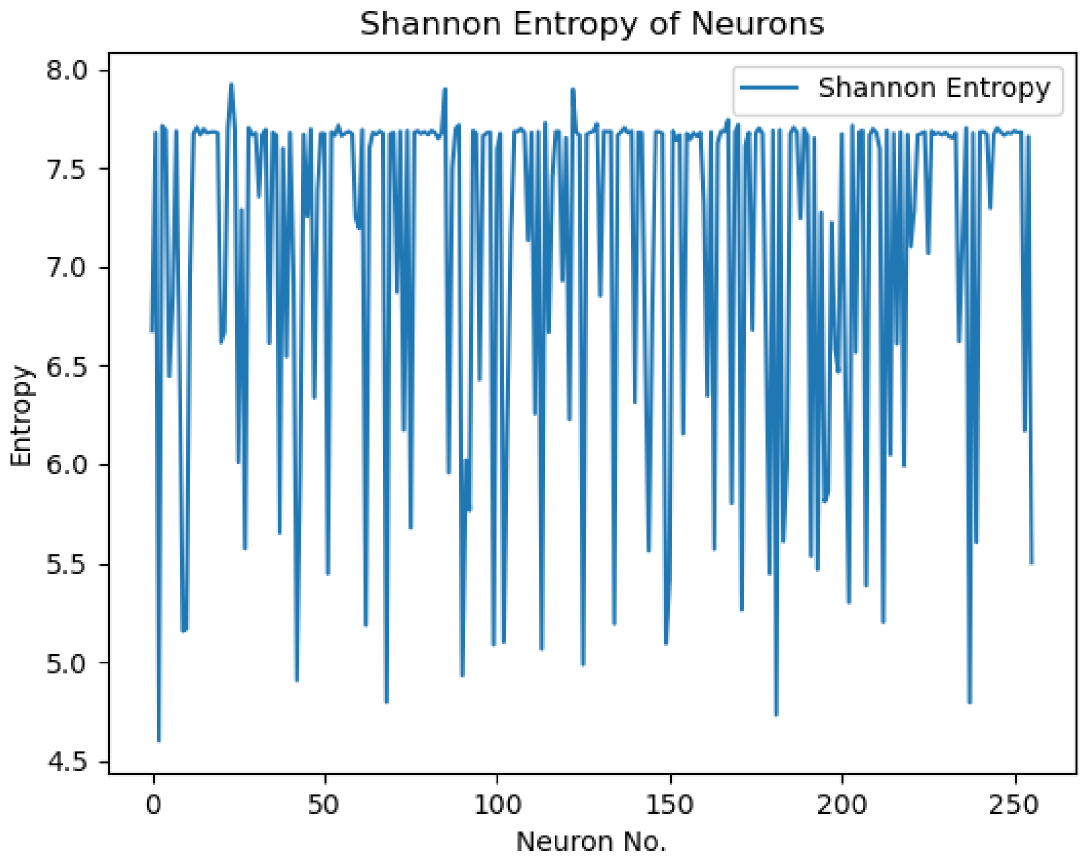


Figure 37

Cosine Activation Function - layer neuron entropy

ArcTan Activation

$$\text{ArcTan}(x) = \arctan(x)$$

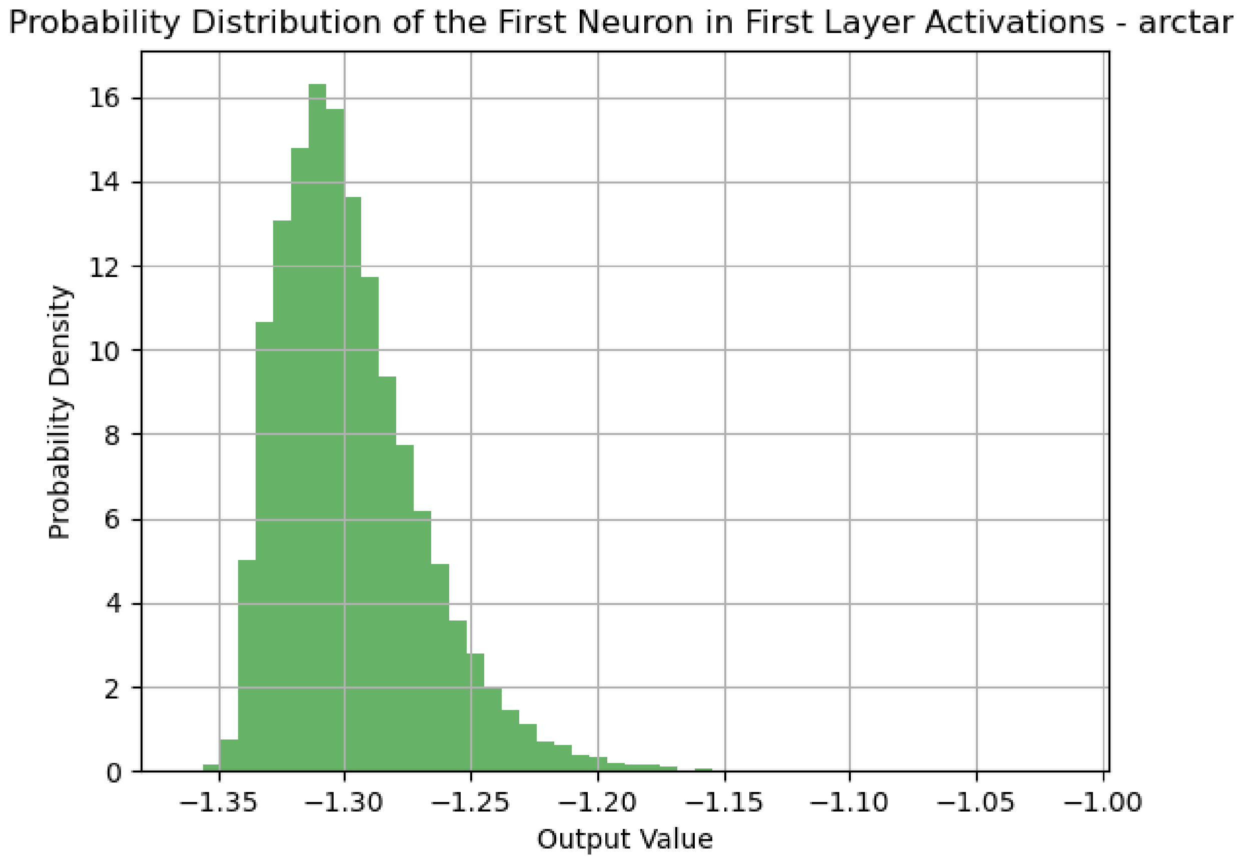


Figure 38

ArcTan Activation Function - first neuron output distribution

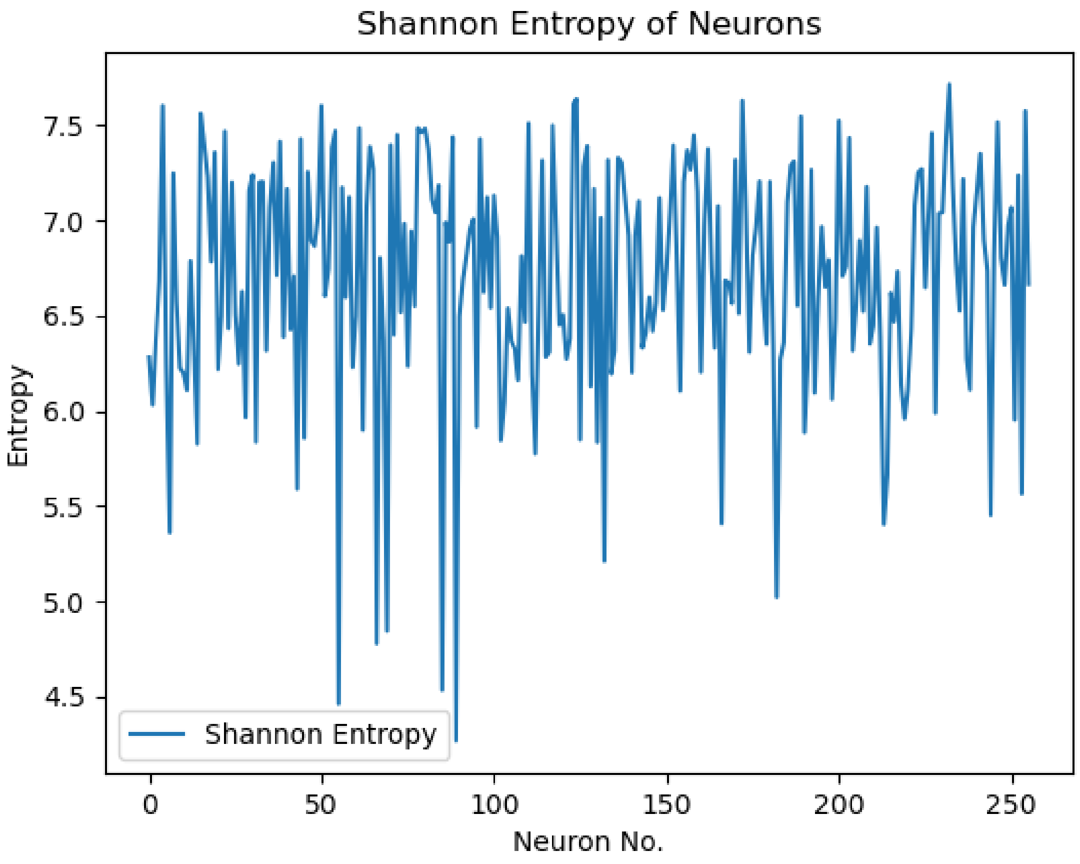


Figure 39

ArcTan Activation Function - layer neuron entropy

Linear Function

$$f(x) = x$$

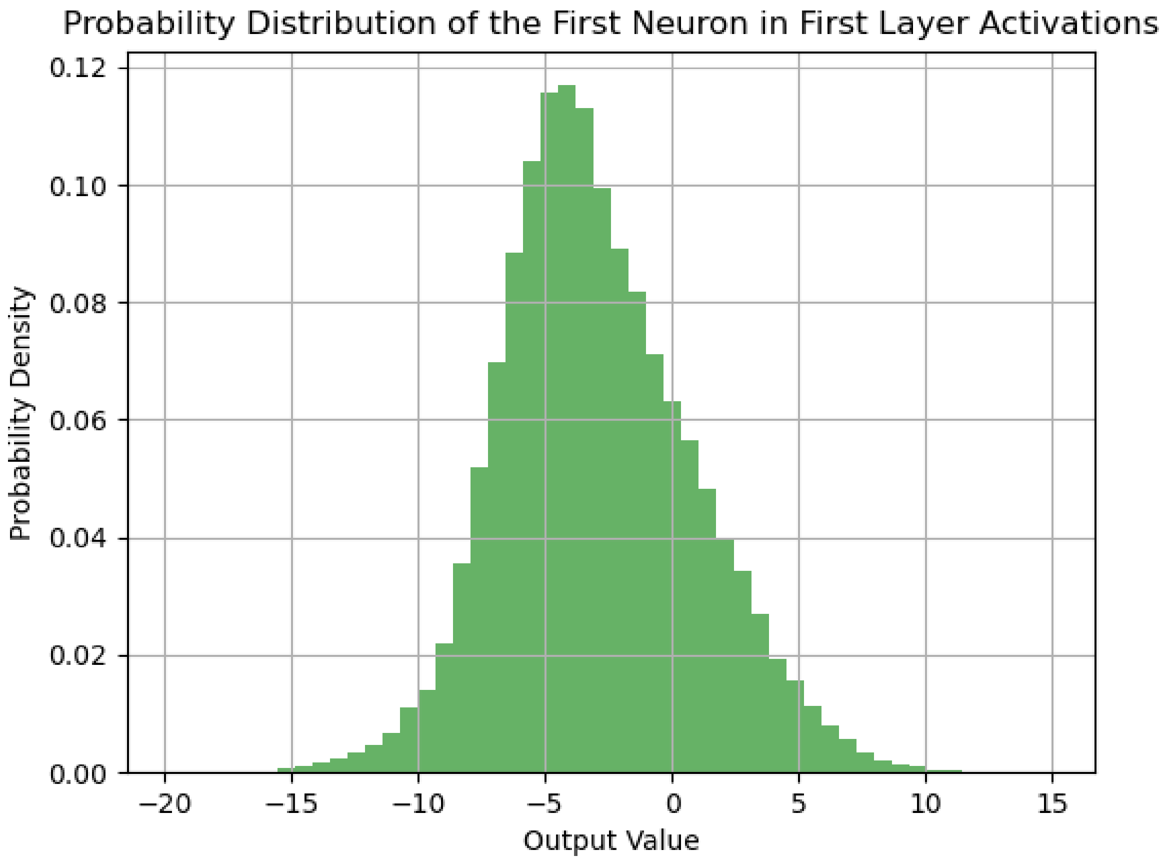


Figure 40

Linear function - first neuron output distribution

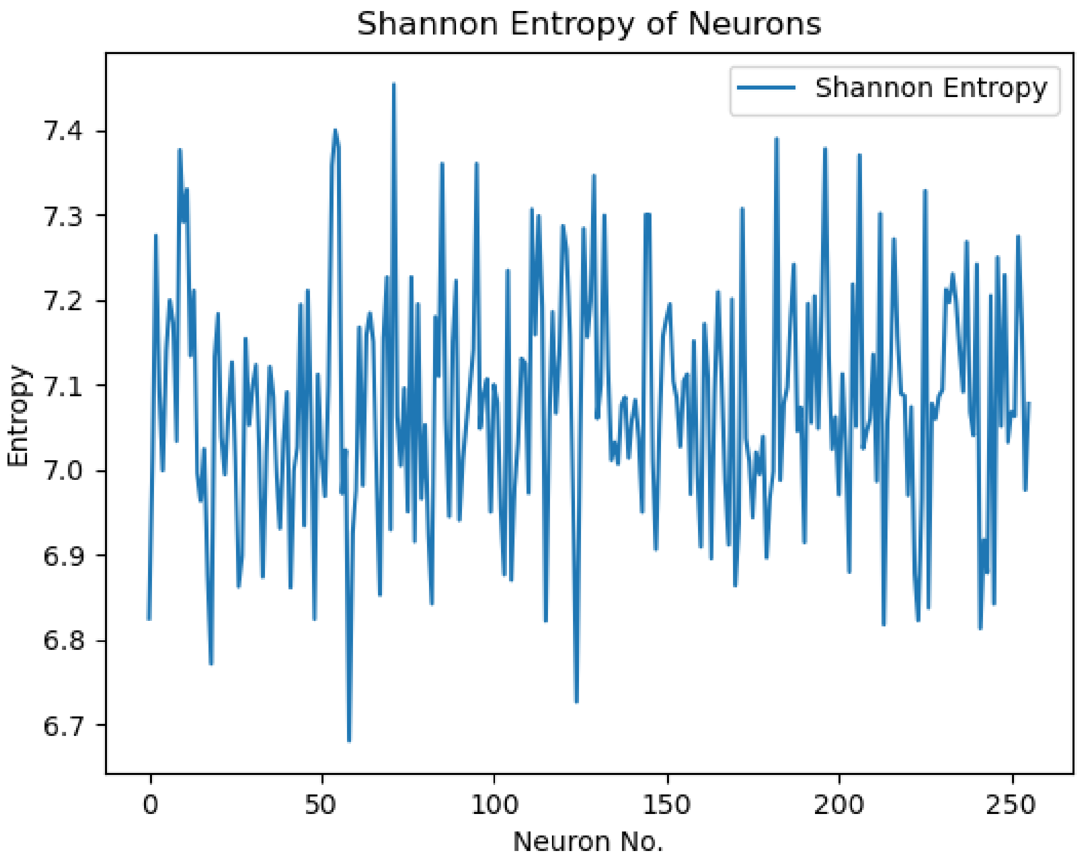


Figure 41

Linear function - layer neuron entropy

Quadratic Function

$$f(x) = x^2$$

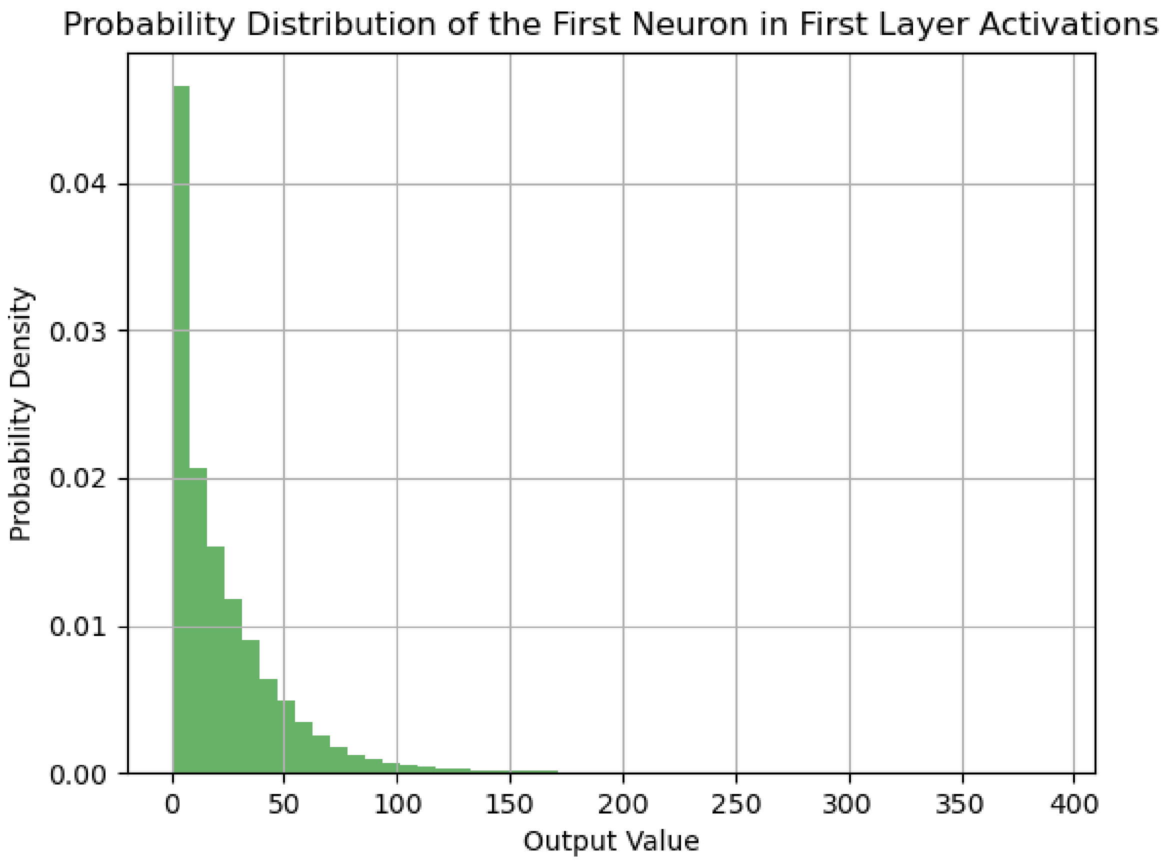


Figure 42

Quadratic function - first neuron output distribution

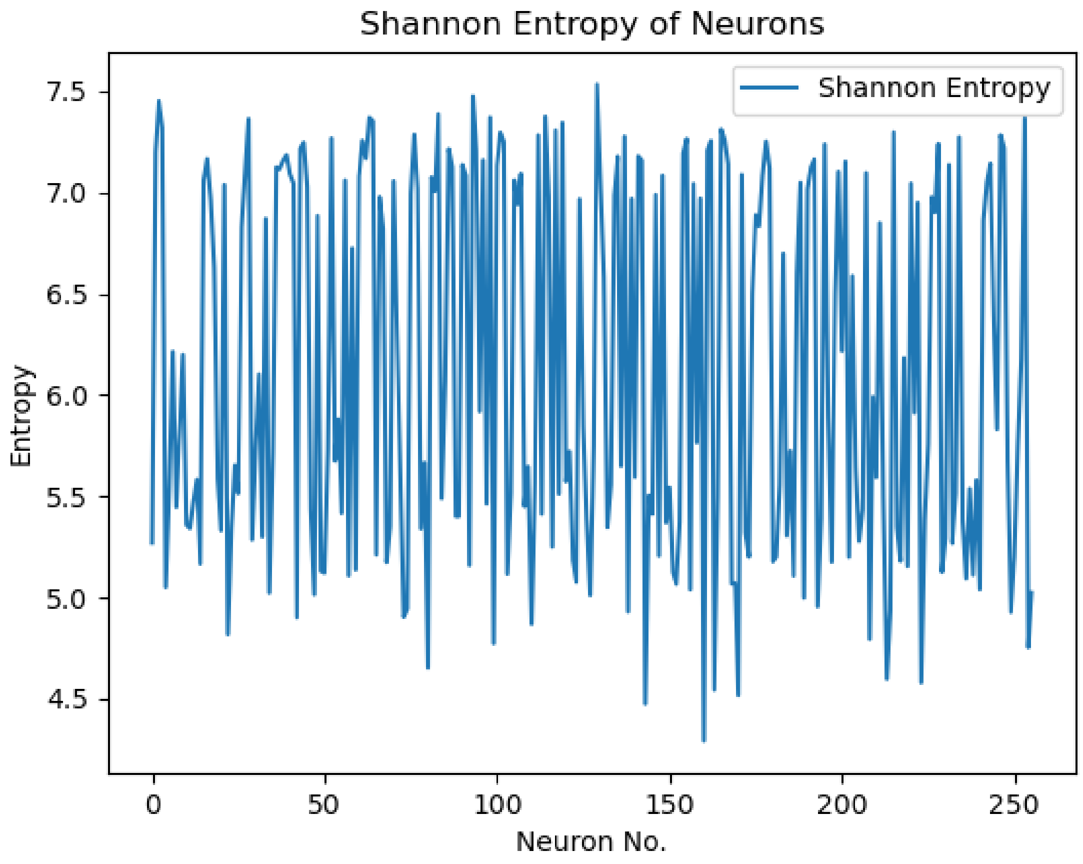


Figure 43

Quadratic function - layer neuron entropy

Cubic Function

$$f(x) = x^3$$

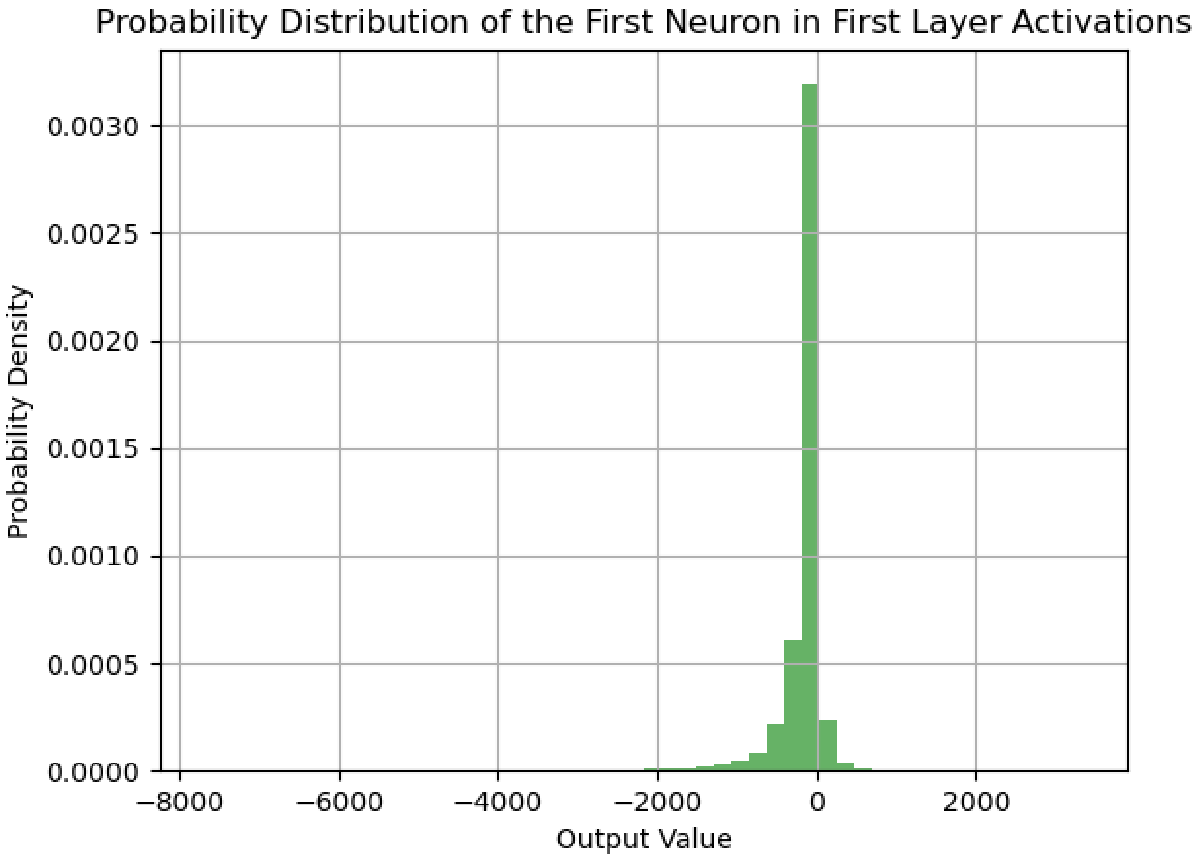


Figure 44

Cubic function - first neuron output distribution

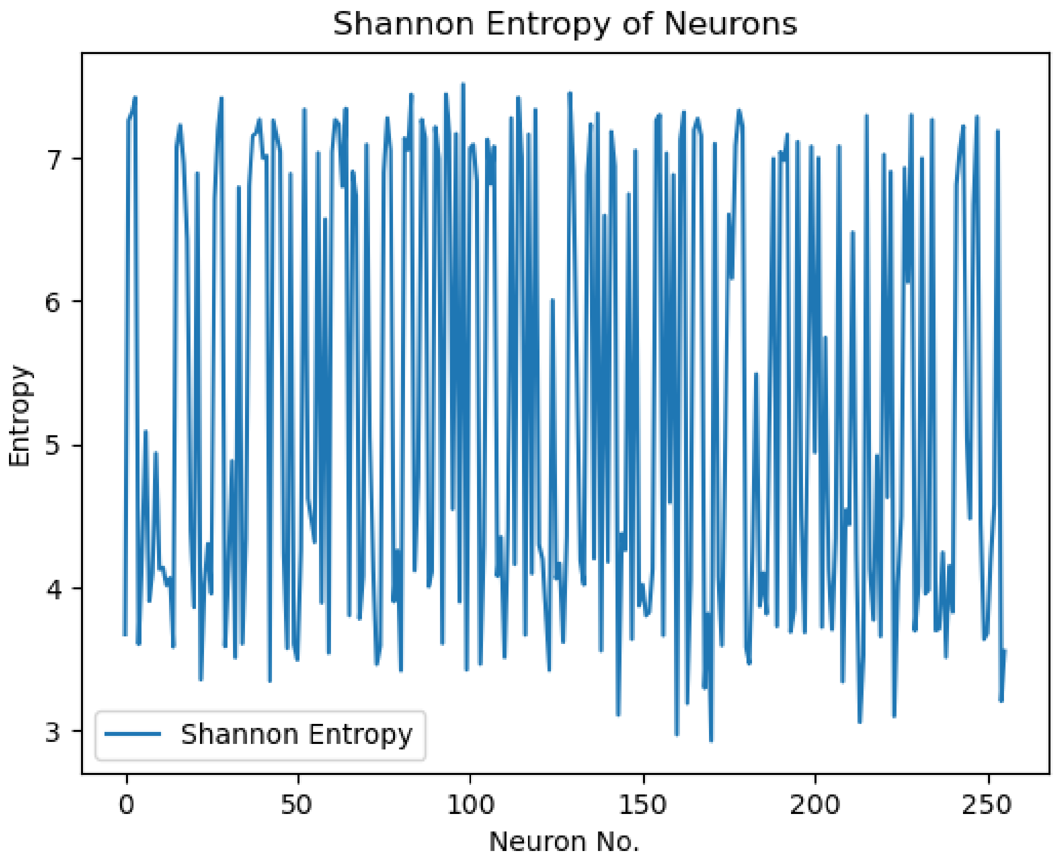


Figure 45

Cubic function - layer neuron entropy

Square Root

$$f(x) = \sqrt{x}$$

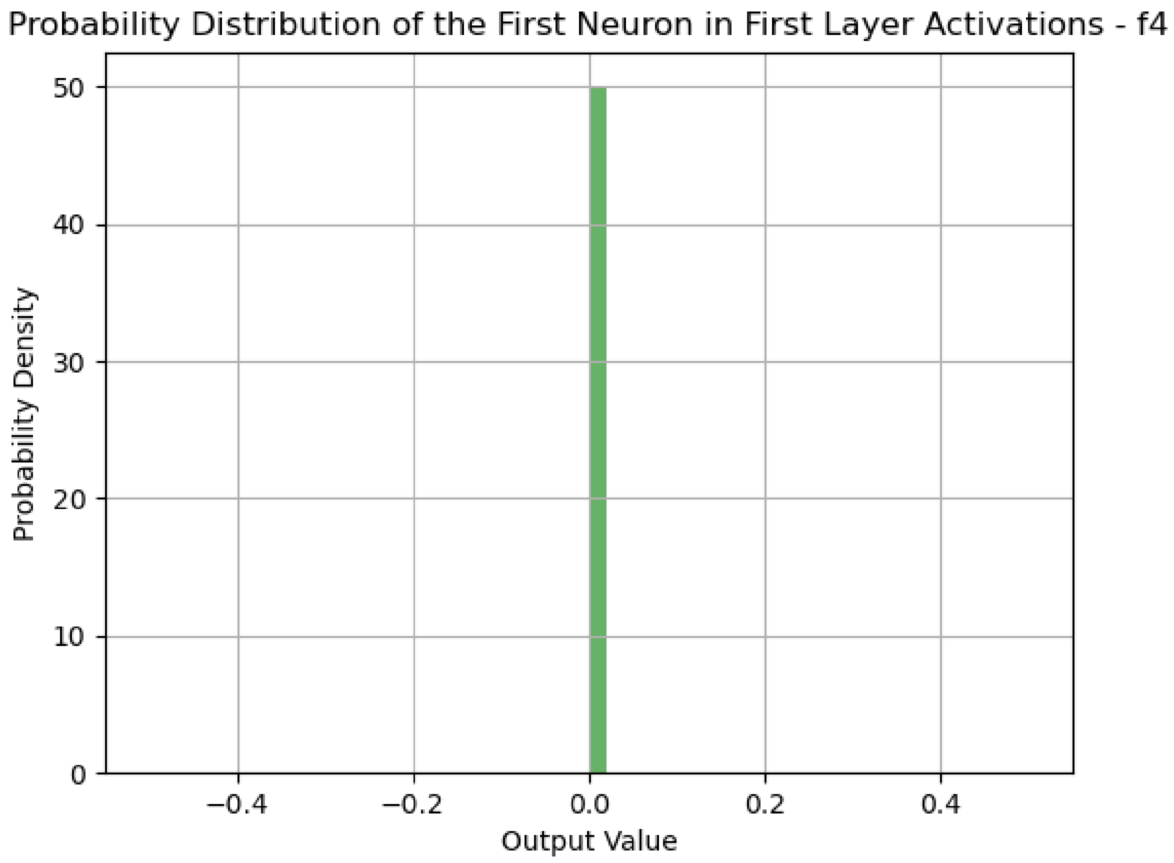


Figure 46

Square Root function - first neuron output distribution

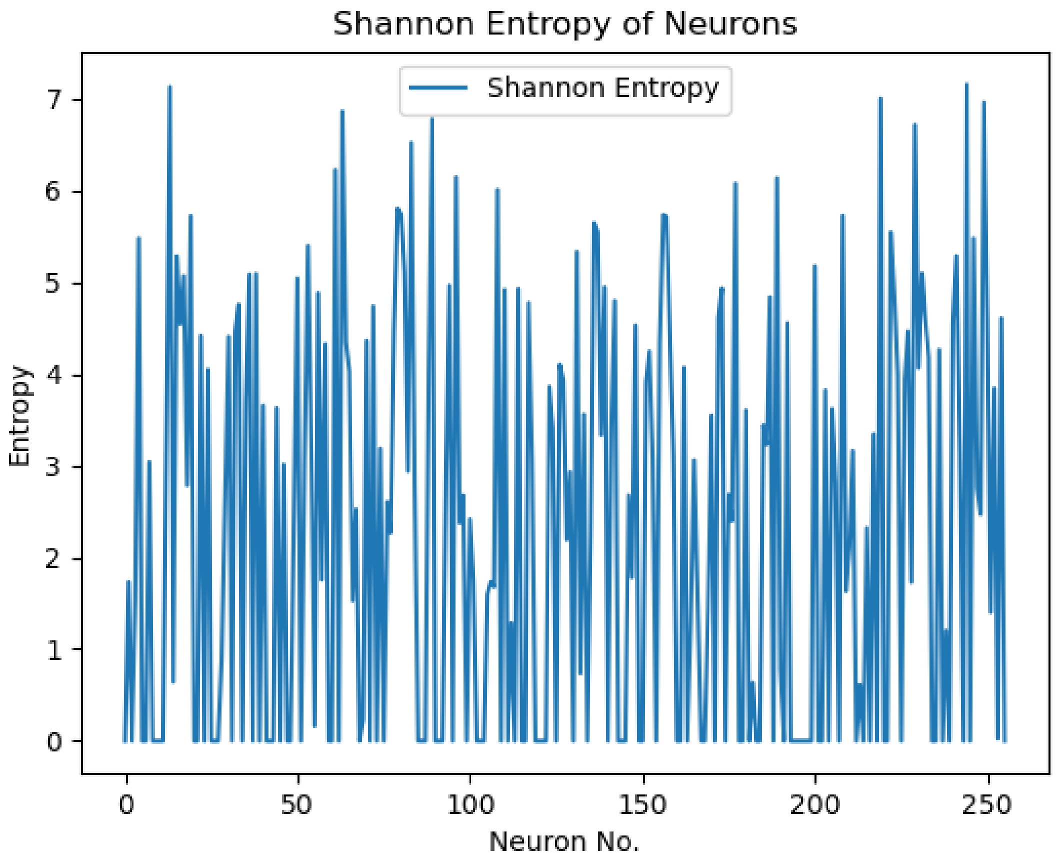


Figure 47

Square Root function - layer neuron entropy

Reciprocal

$$f(x) = \frac{1}{x}$$

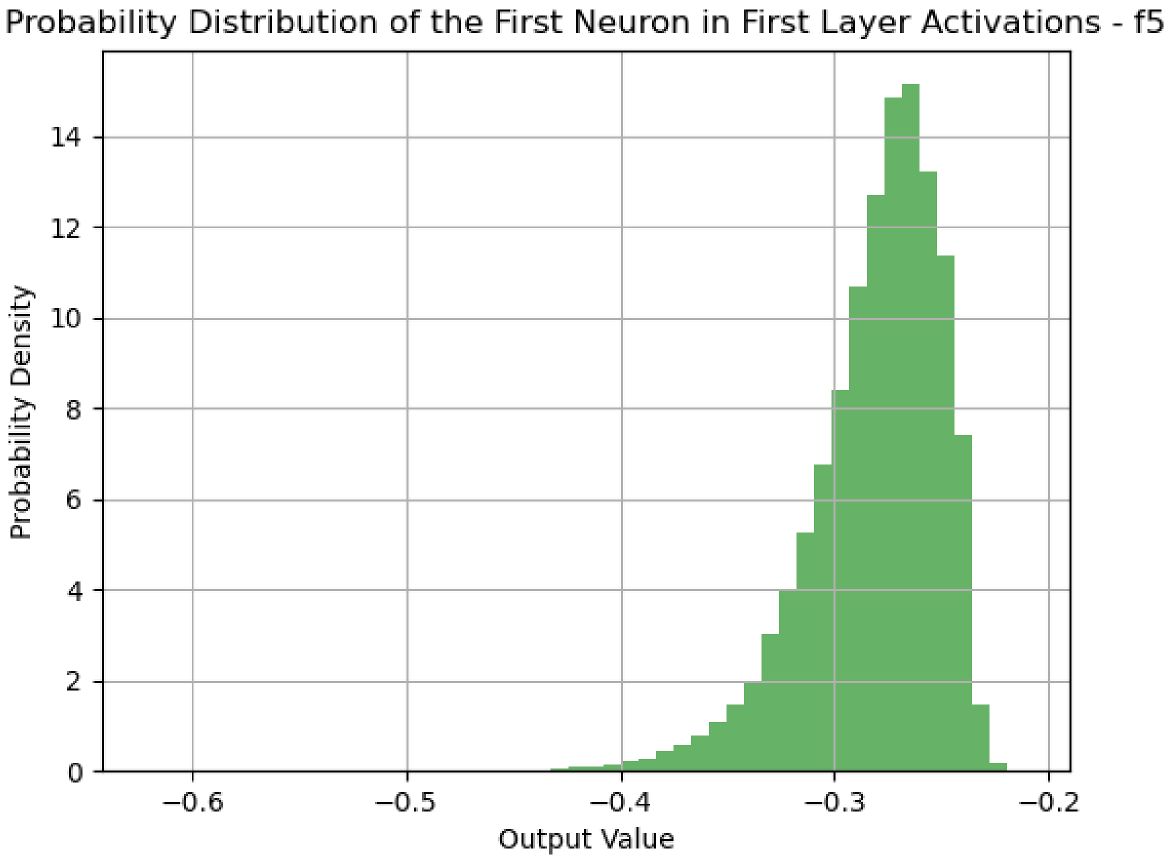


Figure 48

Reciprocal function - first neuron output distribution

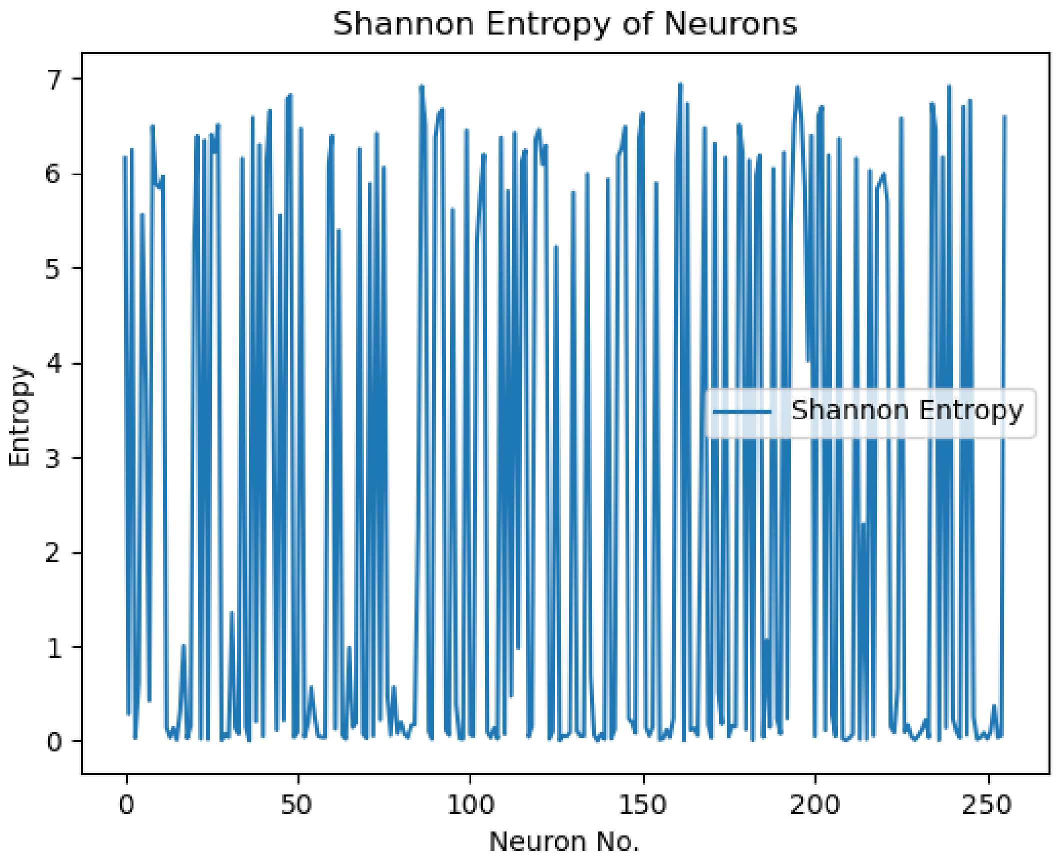


Figure 49

Reciprocal function - layer neuron entropy

Tangent Function

$$f(x) = \tan(x)$$

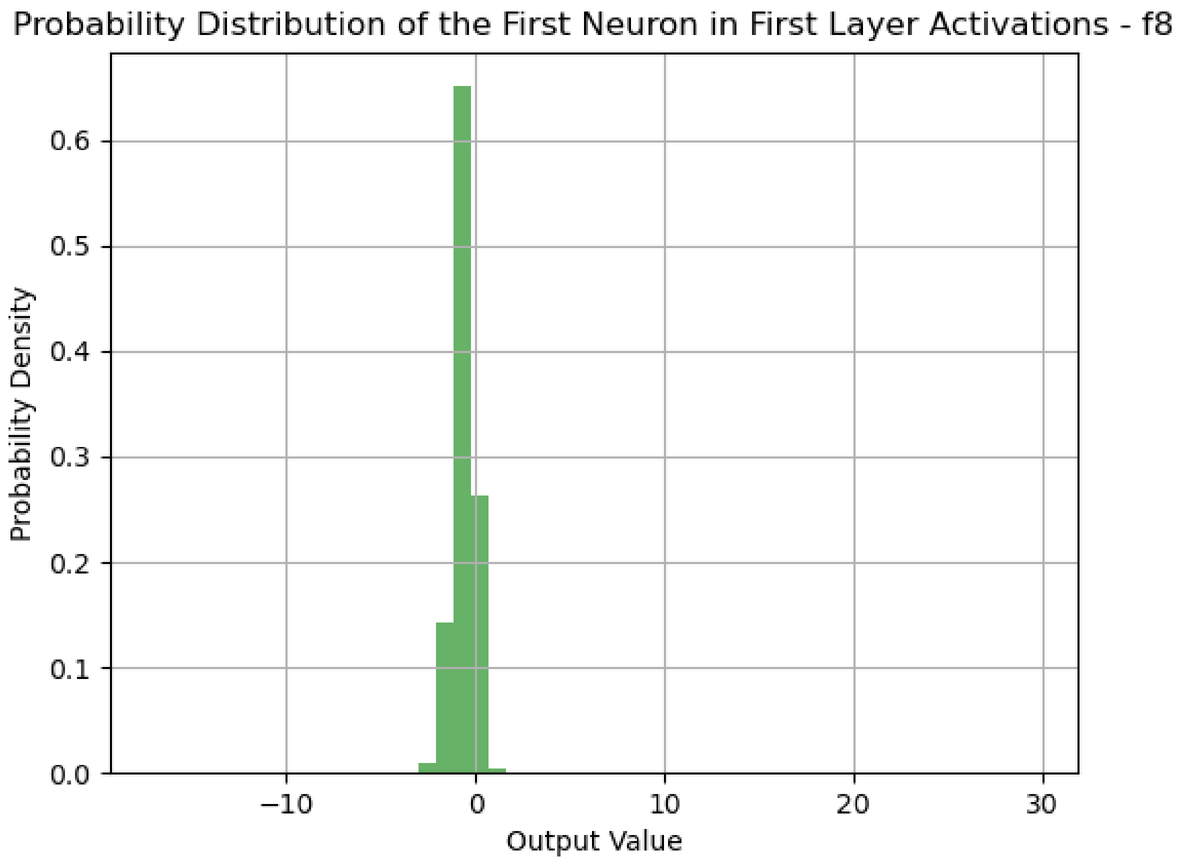


Figure 50

Tangent function - first neuron output distribution

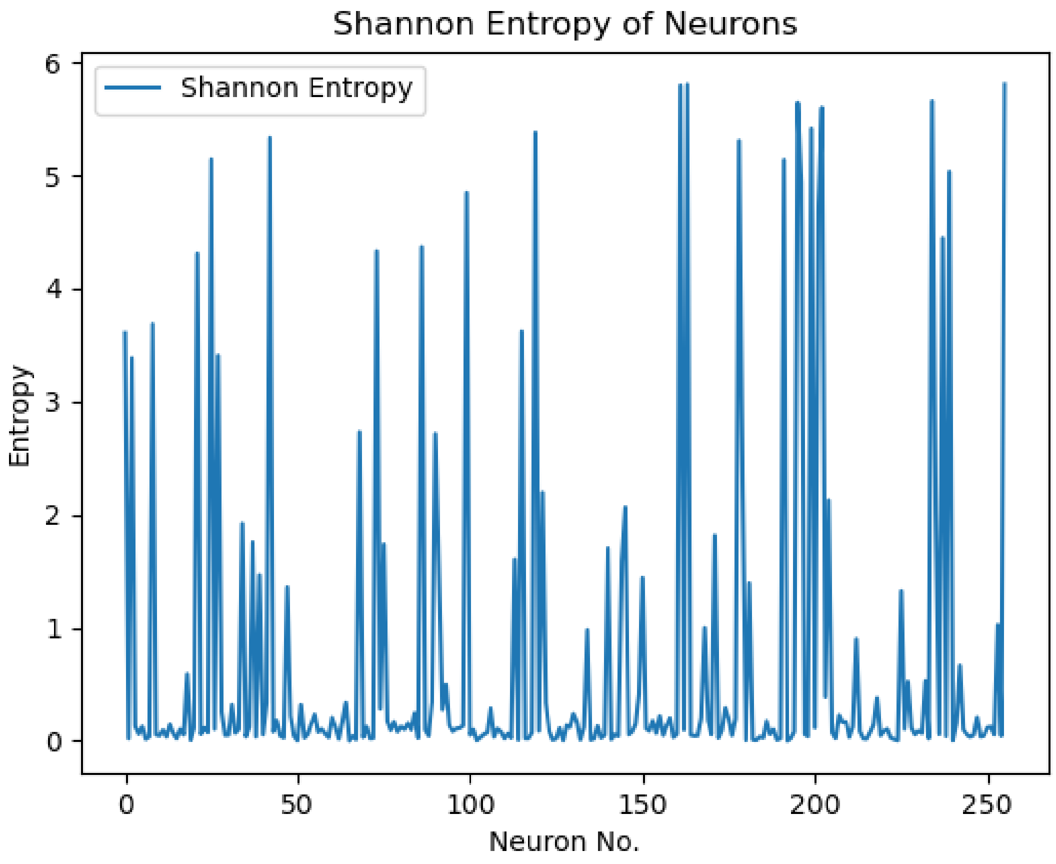


Figure 51

Tangent function - layer neuron entropy

Cosecant Function

$$f(x) = \csc(x) = \frac{1}{\sin(x)}$$

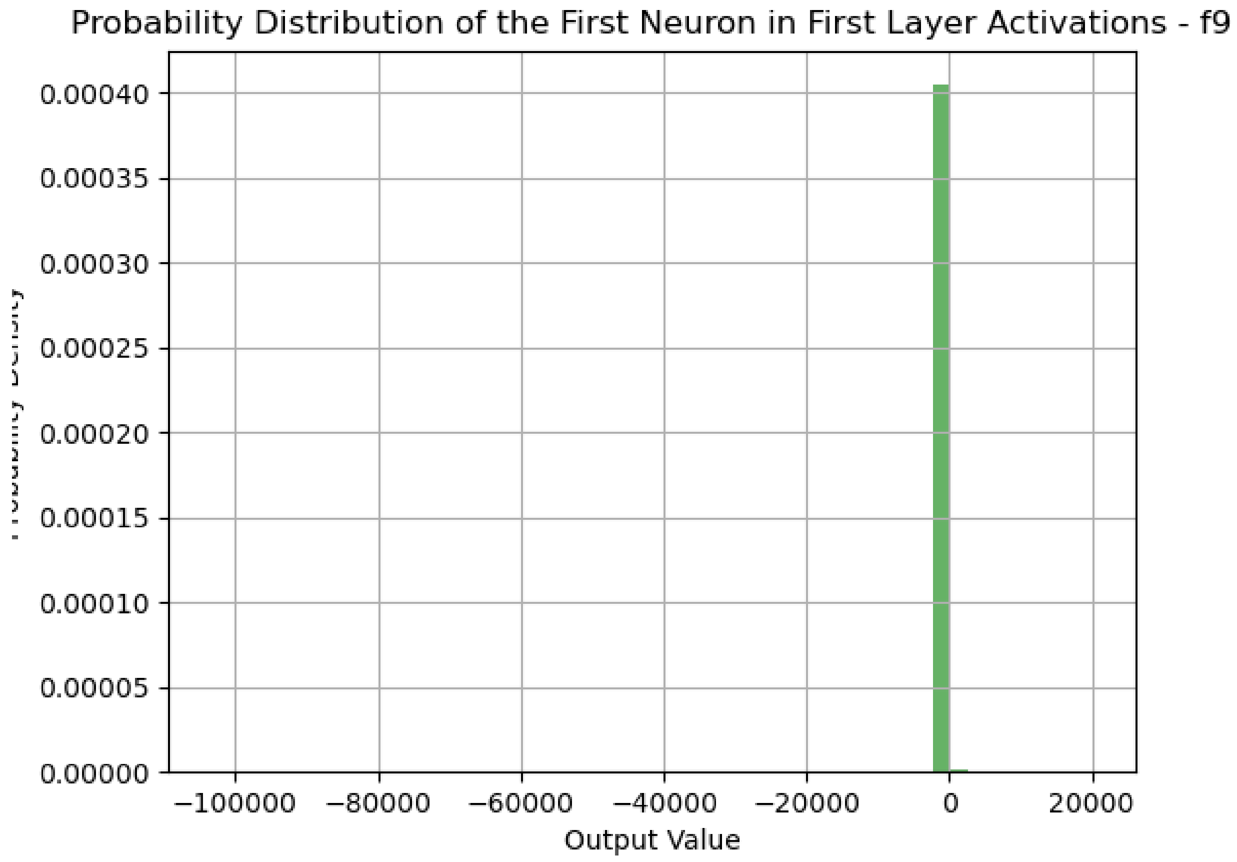


Figure 52

Cosecant function - first neuron output distribution

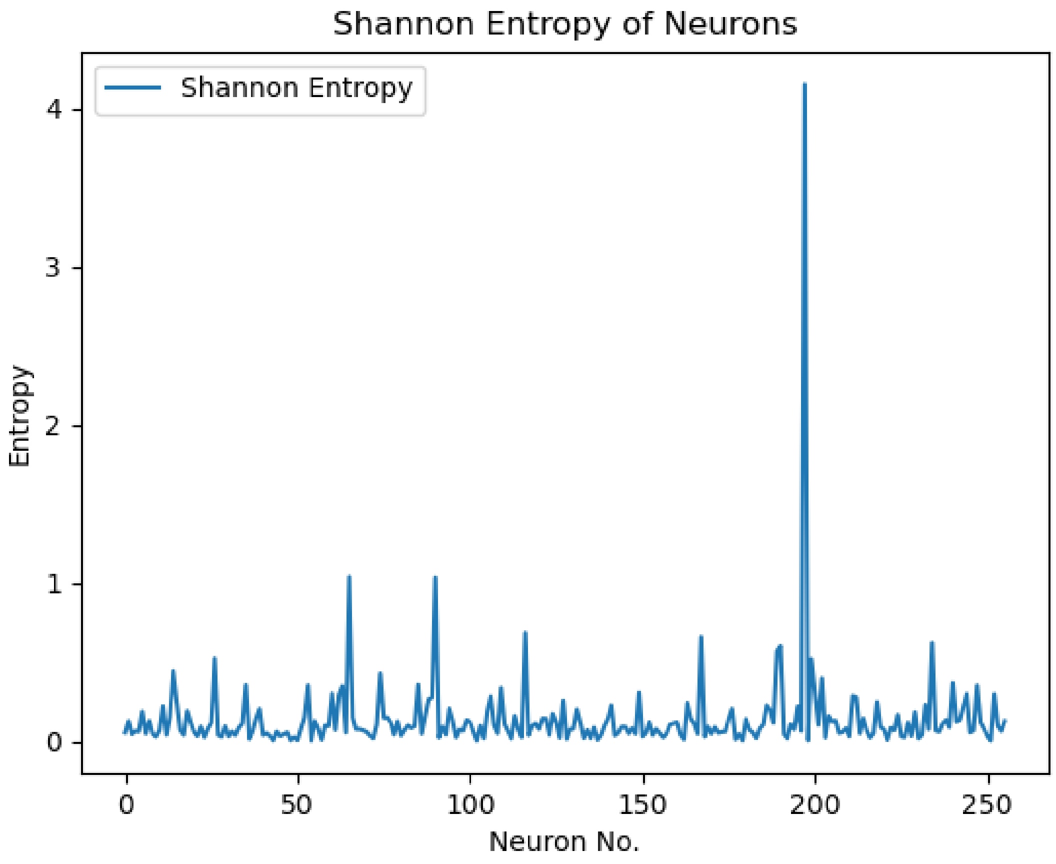


Figure 53

Cosecant function - layer neuron entropy

Secant Function

$$f(x) = \sec(x) = \frac{1}{\cos(x)}$$

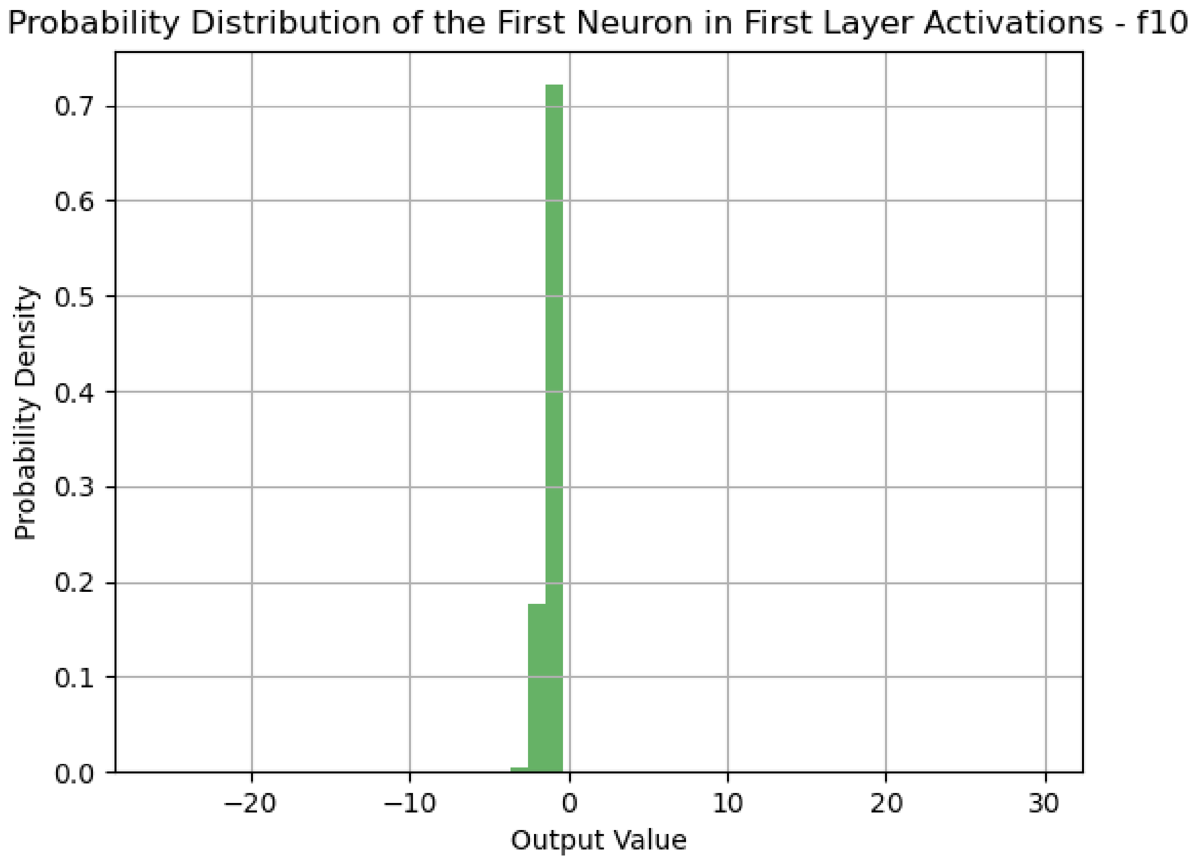


Figure 54

Secant function - first neuron output distribution

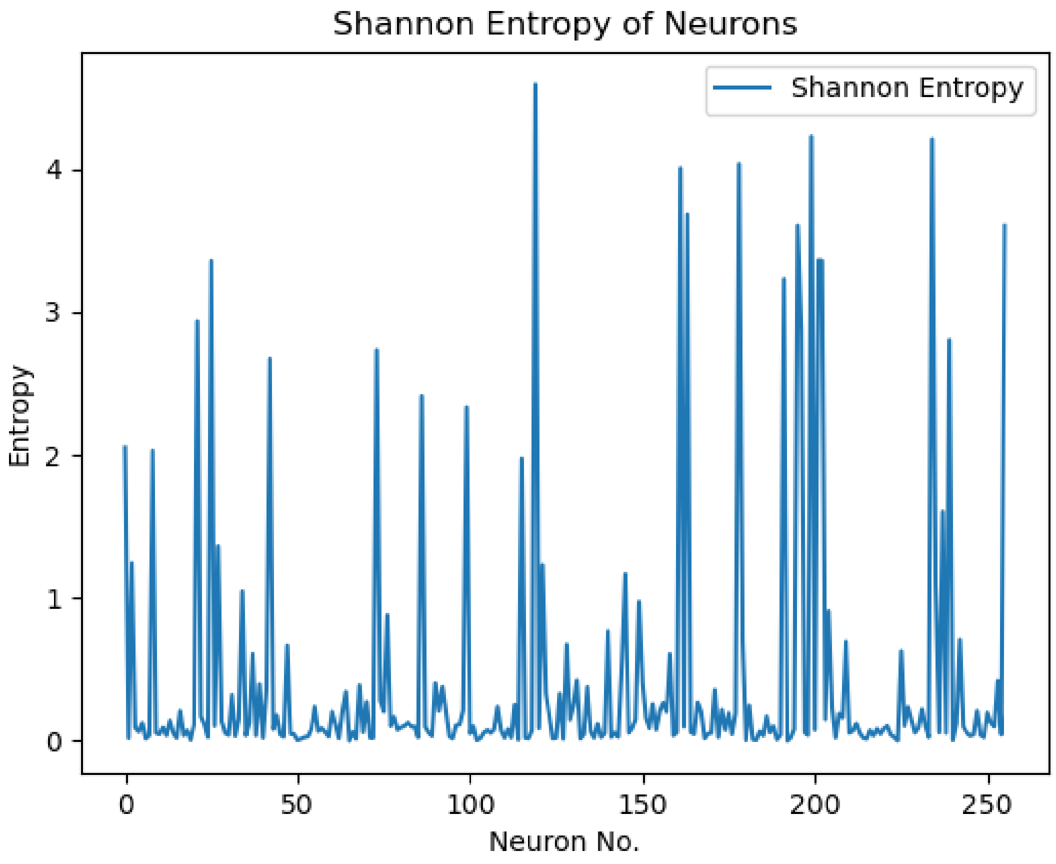


Figure 55

Secant function - layer neuron entropy

Cotangent Function

$$f(x) = \cot(x) = \frac{1}{\tan(x)}$$

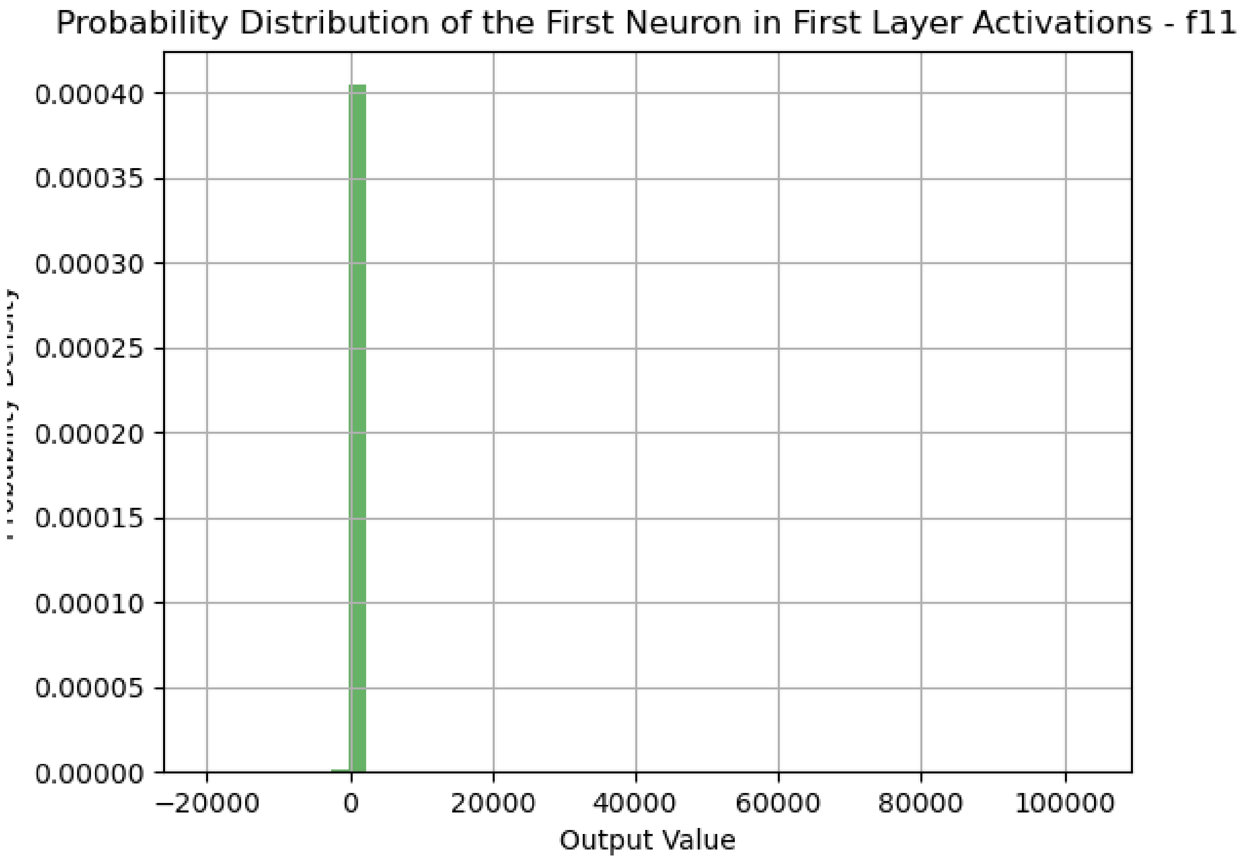


Figure 56

Cotangent function - first neuron output distribution

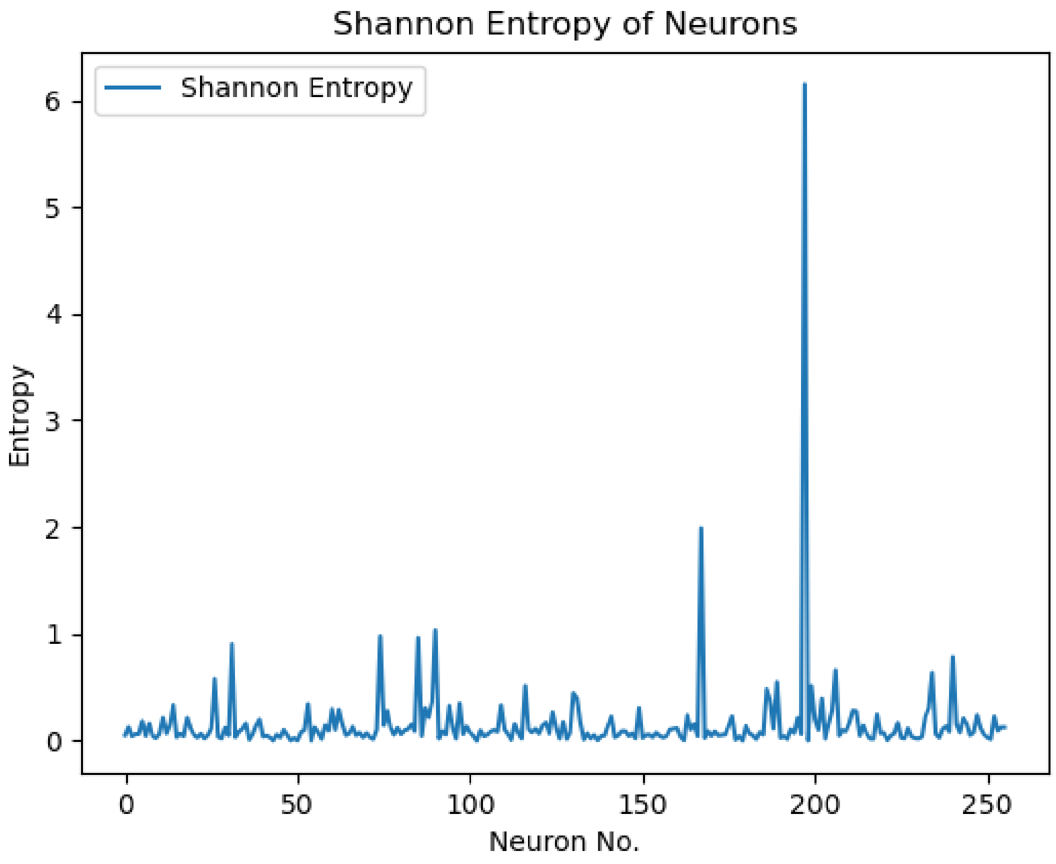


Figure 57

Cotangent function - layer neuron entropy

Inverse Sine

$$f(x) = \arcsin(x)$$

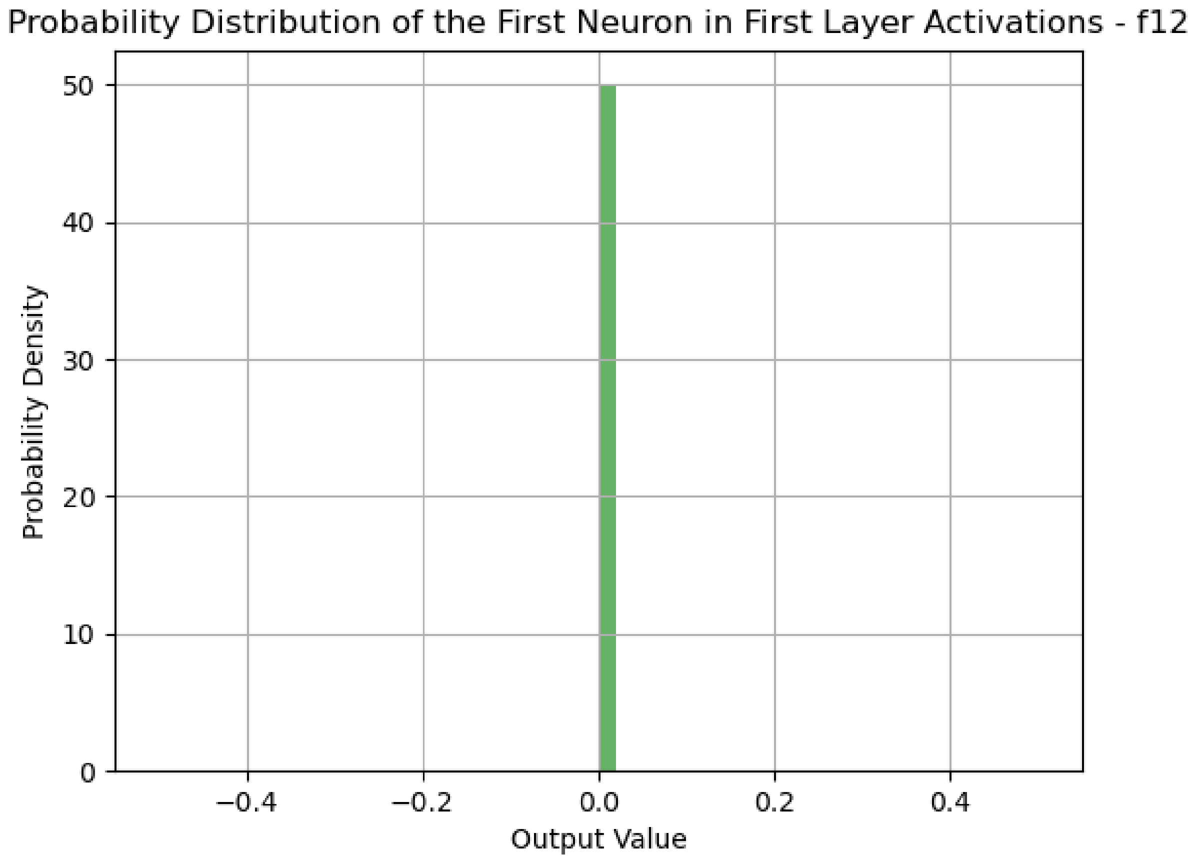


Figure 58

Inverse Sine function - first neuron output distribution

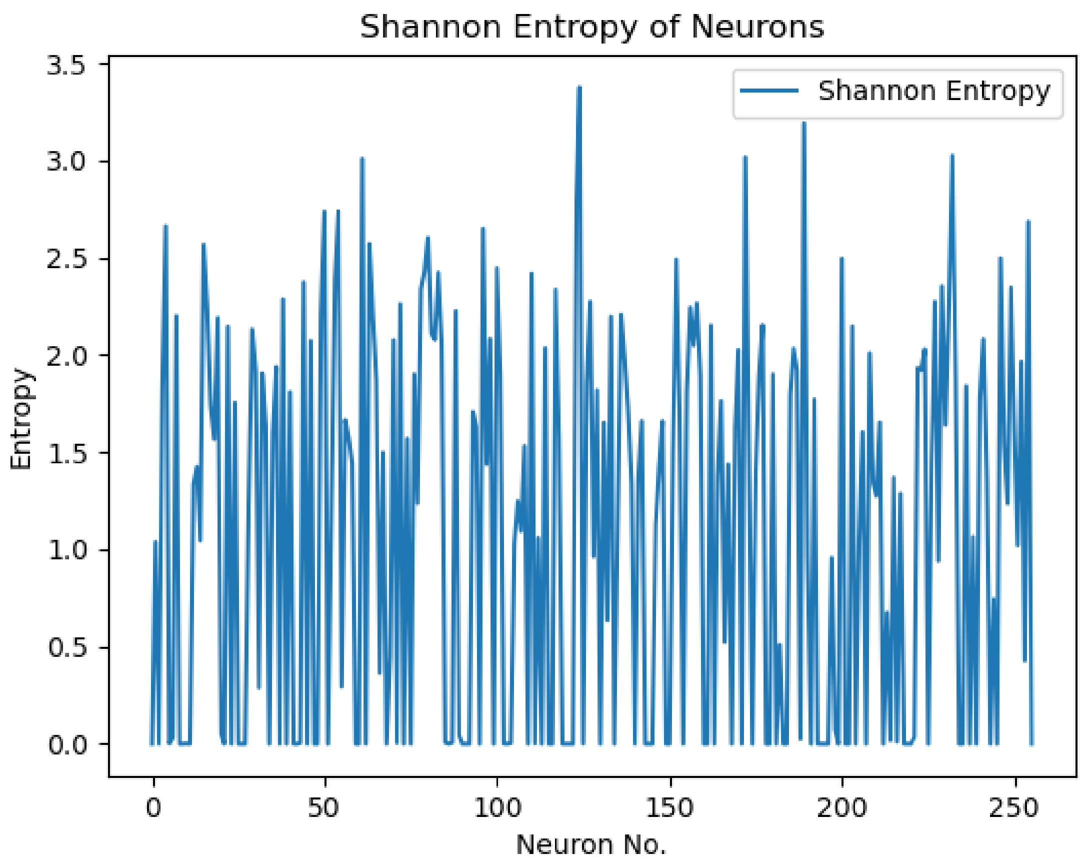


Figure 59

Inverse Sine function - layer neuron entropy

Inverse Cosine

$$f(x) = \arccos(x)$$

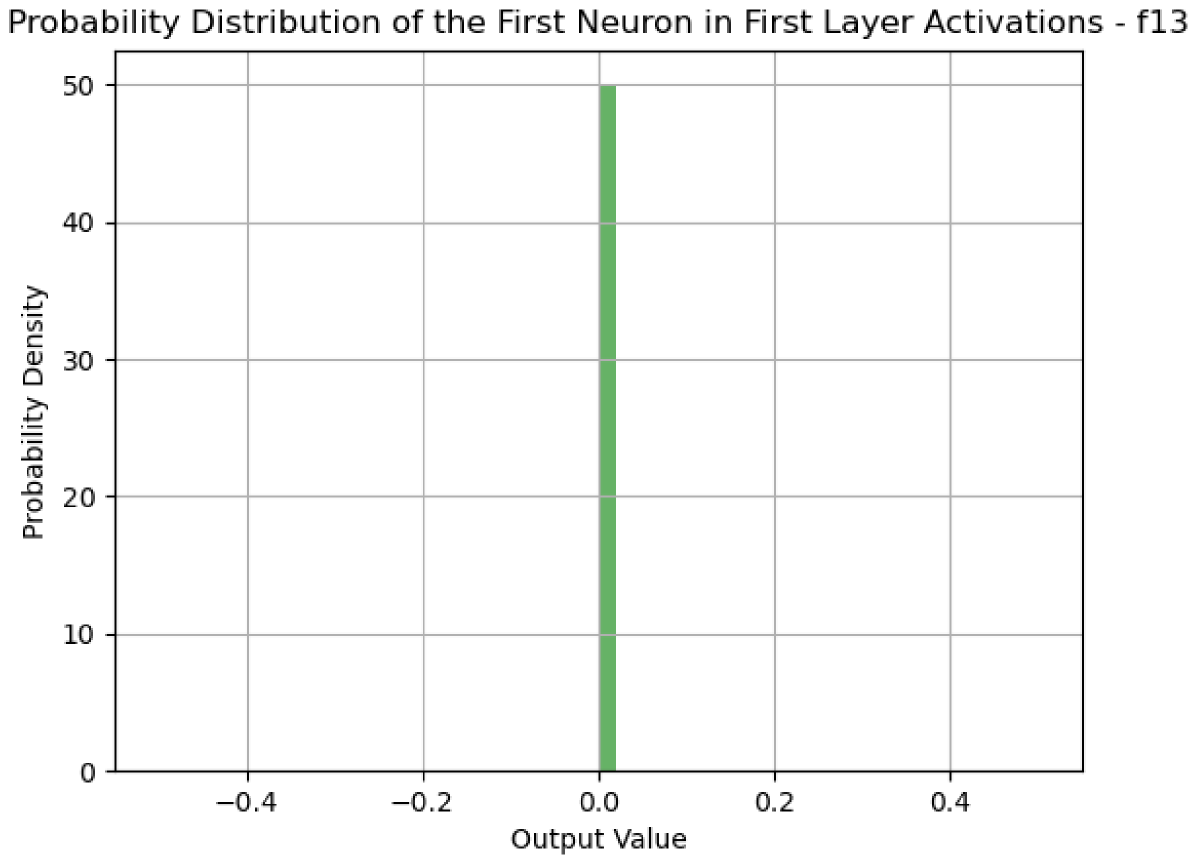


Figure 60

Inverse Cosine function - first neuron output distribution

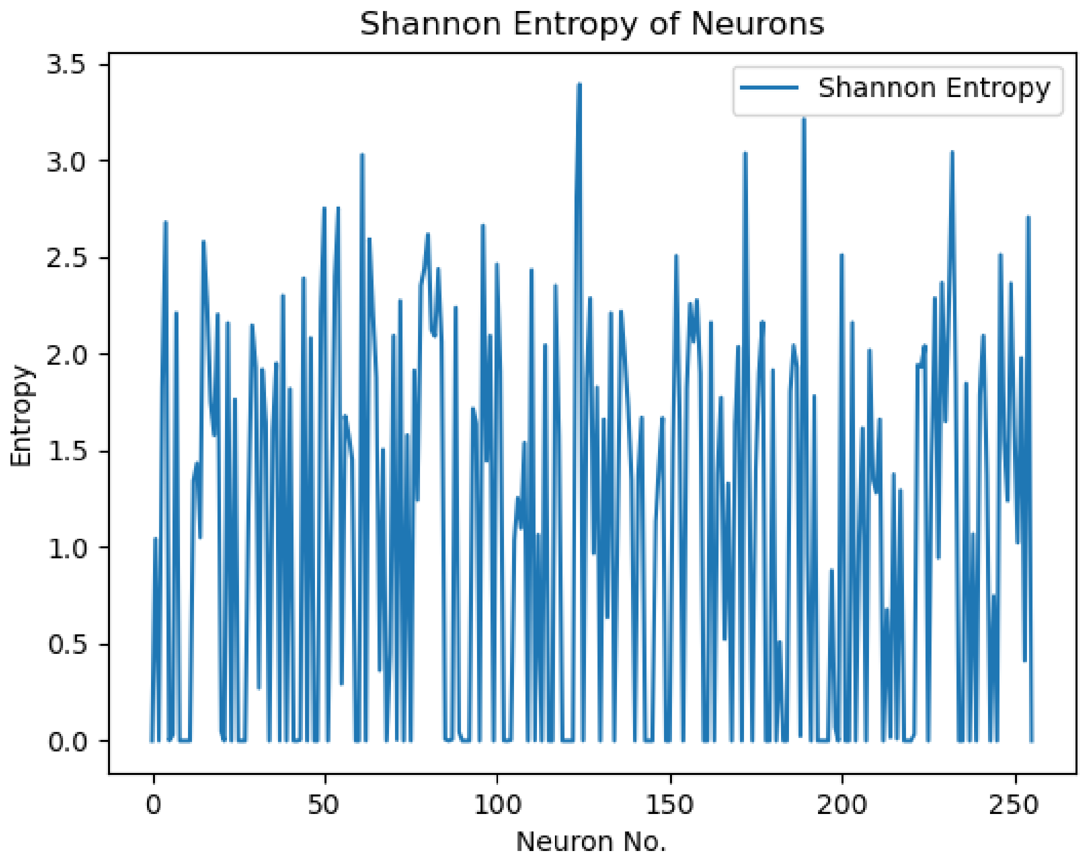


Figure 61

Inverse Cosine function - layer neuron entropy

Inverse Tangent

$$f(x) = \arctan(x)$$

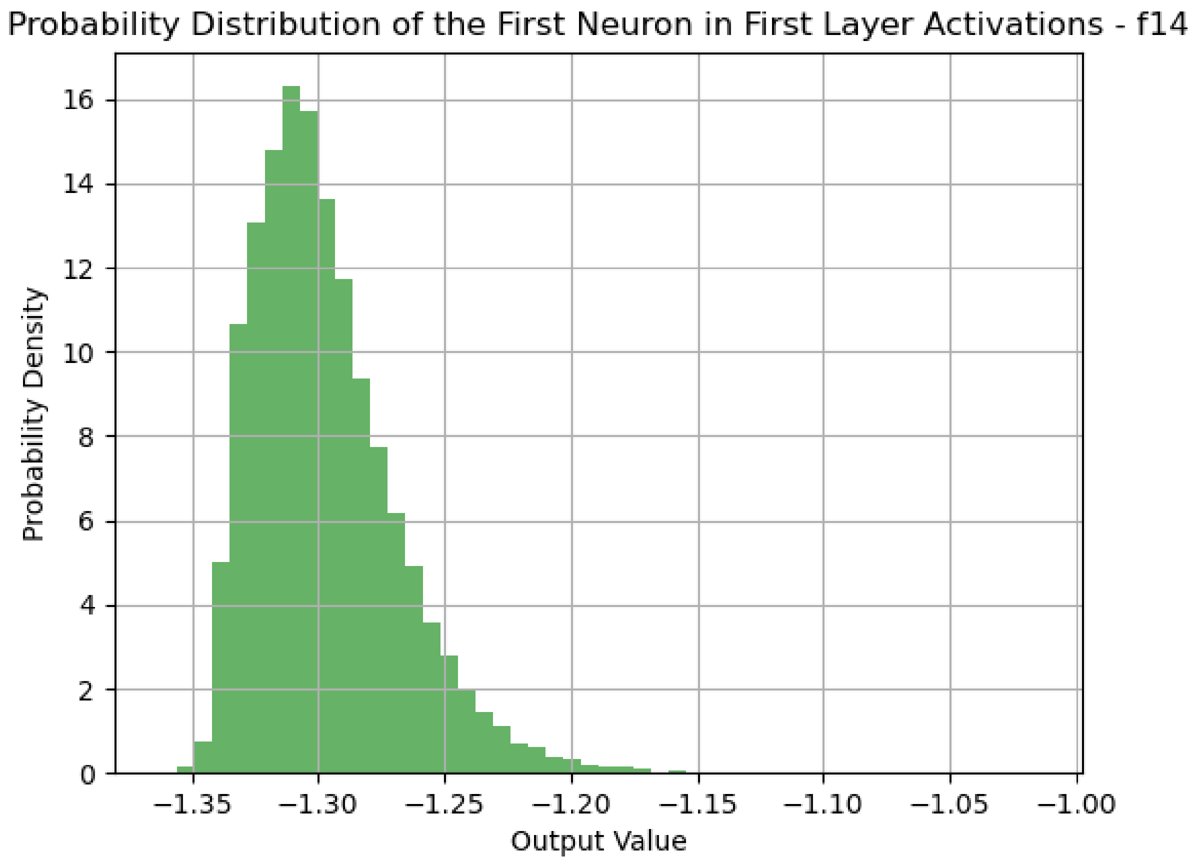


Figure 62

Inverse Tangent function - first neuron output distribution

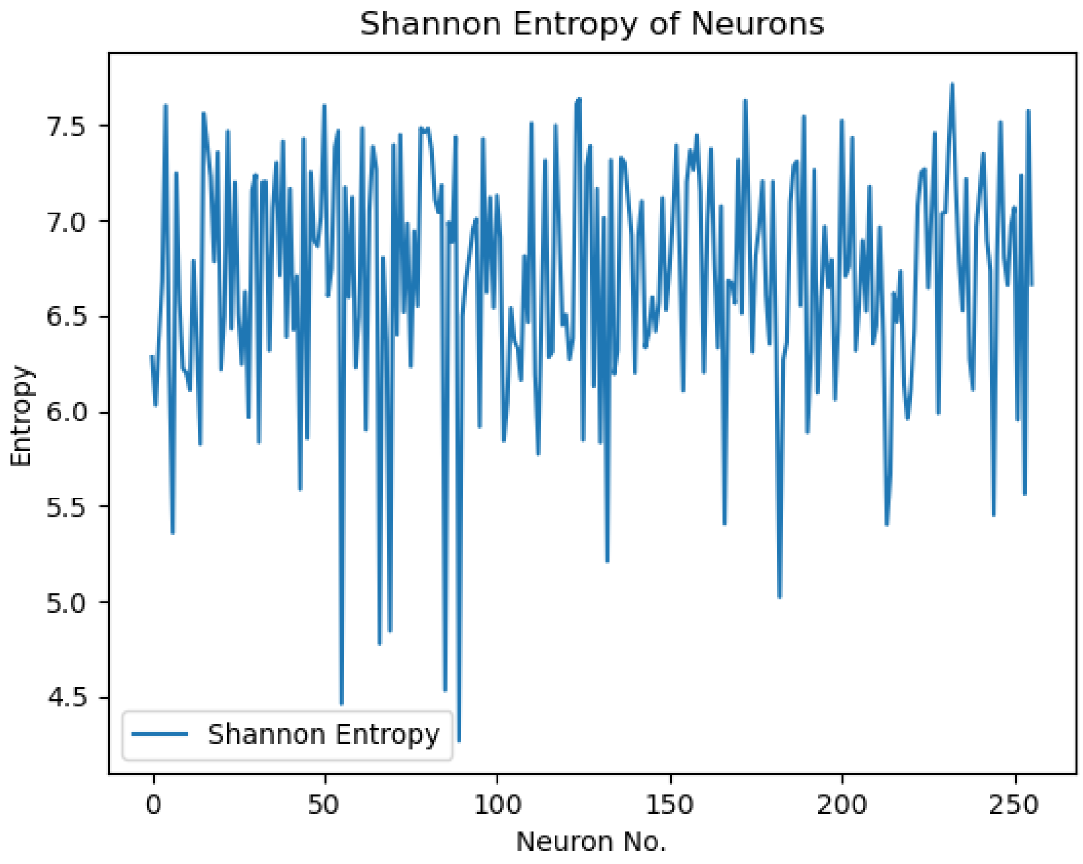


Figure 63

Inverse Tangent function - layer neuron entropy

Hyperbolic Sine

$$f(x) = \sinh(x) = \frac{e^x - e^{-x}}{2}$$

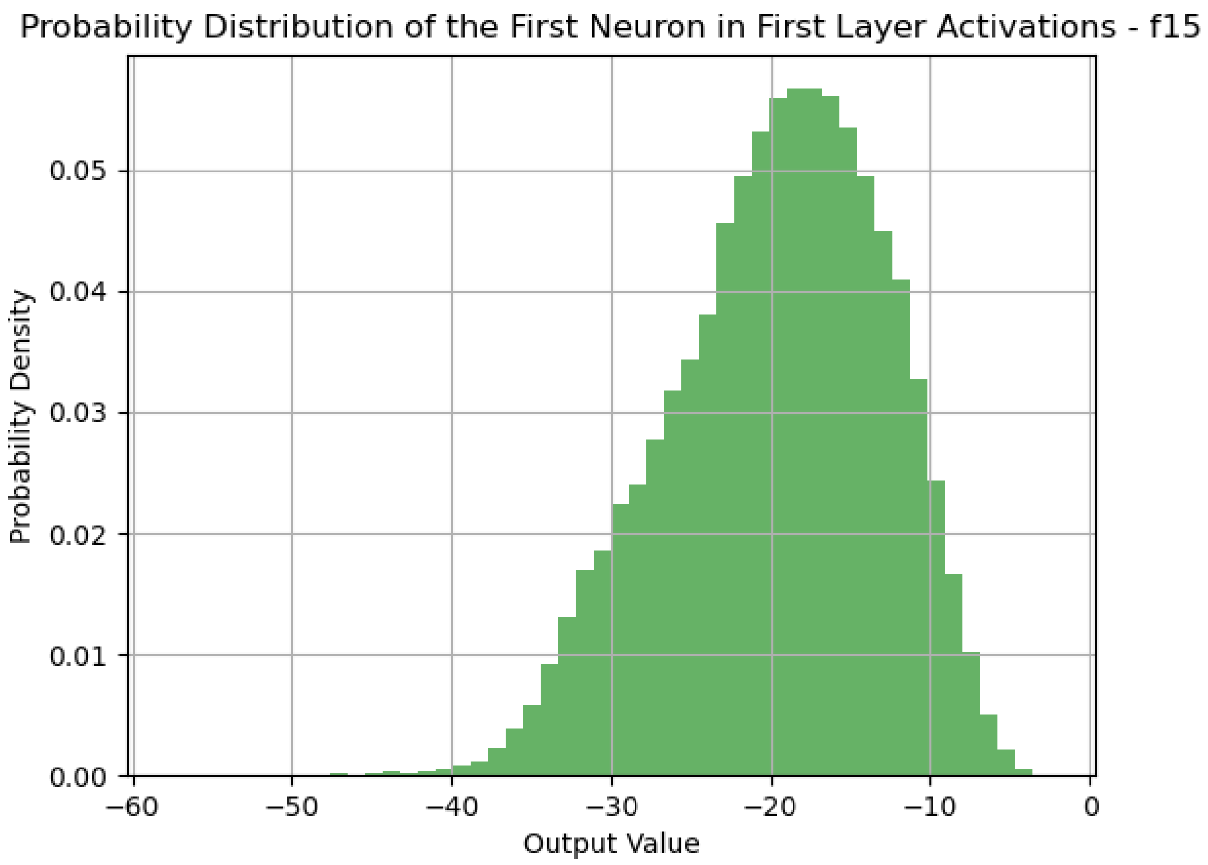


Figure 64

Hyperbolic Sine function - first neuron output distribution

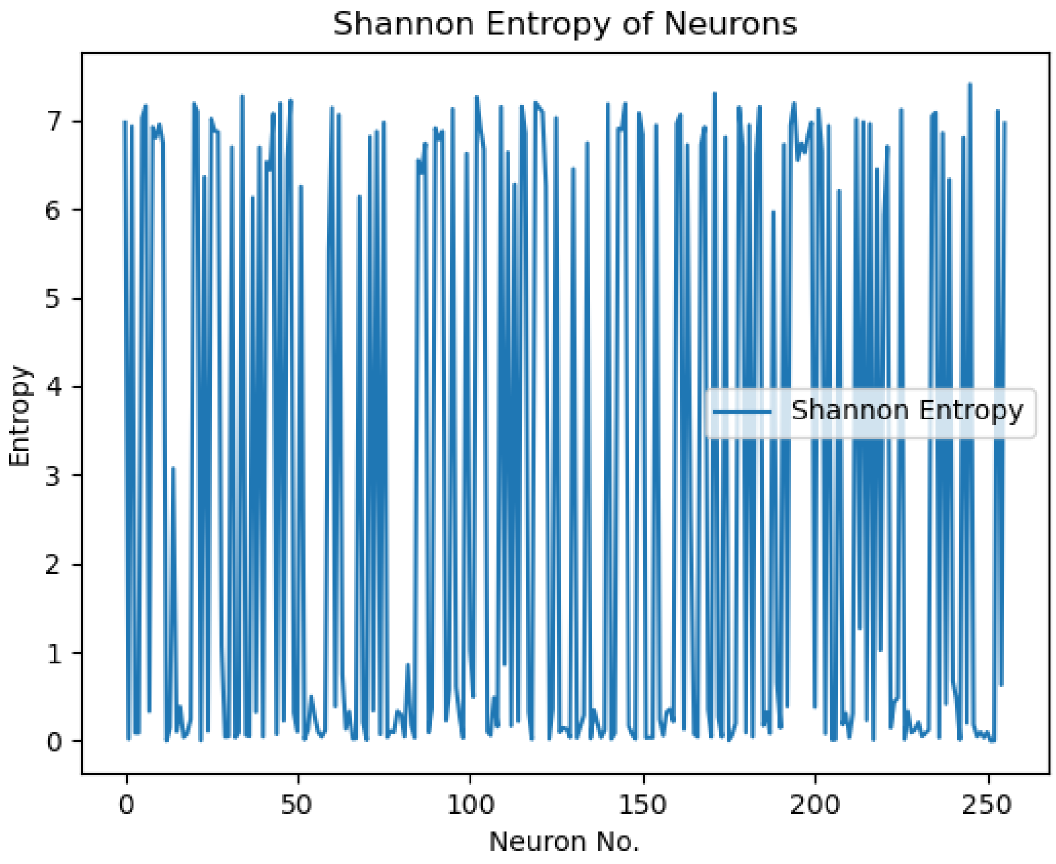


Figure 65

Hyperbolic Sine function - layer neuron entropy

Hyperbolic Cosine

$$f(x) = \cosh(x) = \frac{e^x + e^{-x}}{2}$$

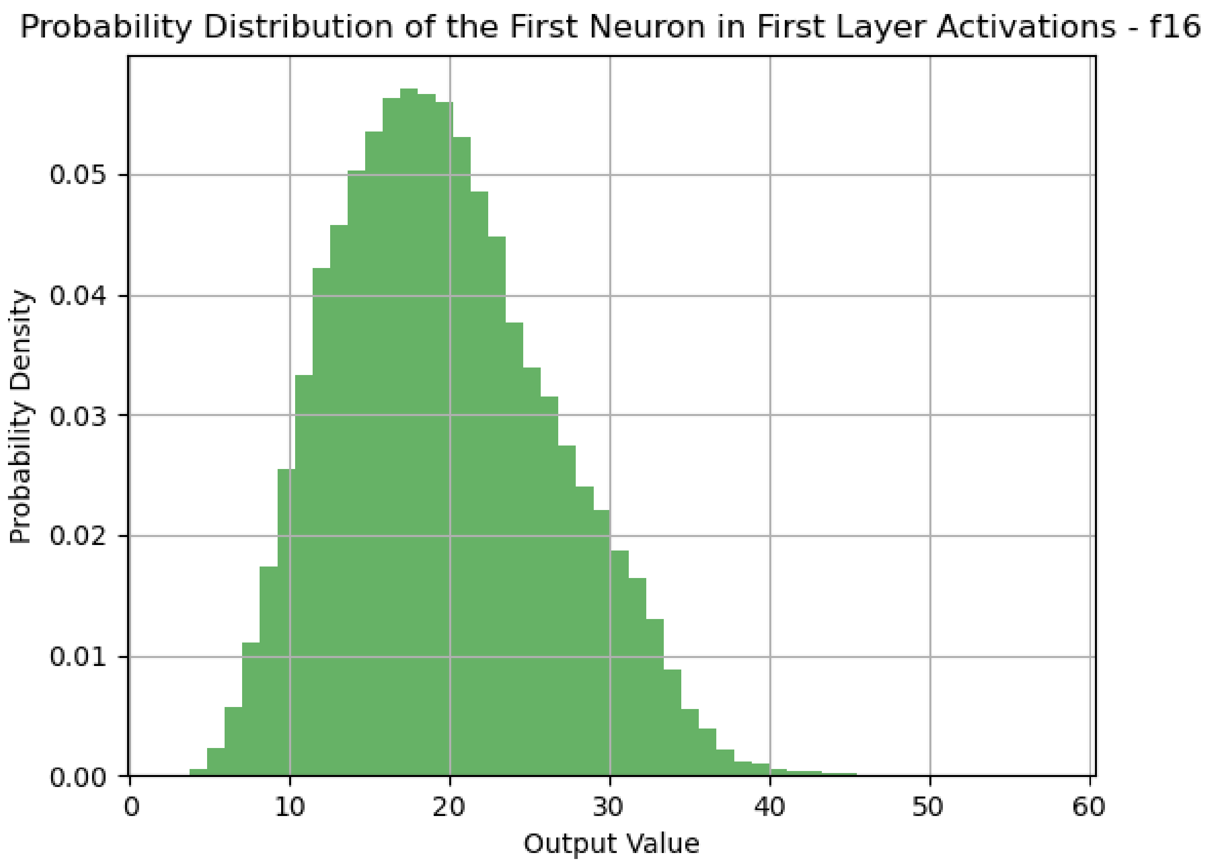


Figure 66

Hyperbolic Cosine function - first neuron output distribution

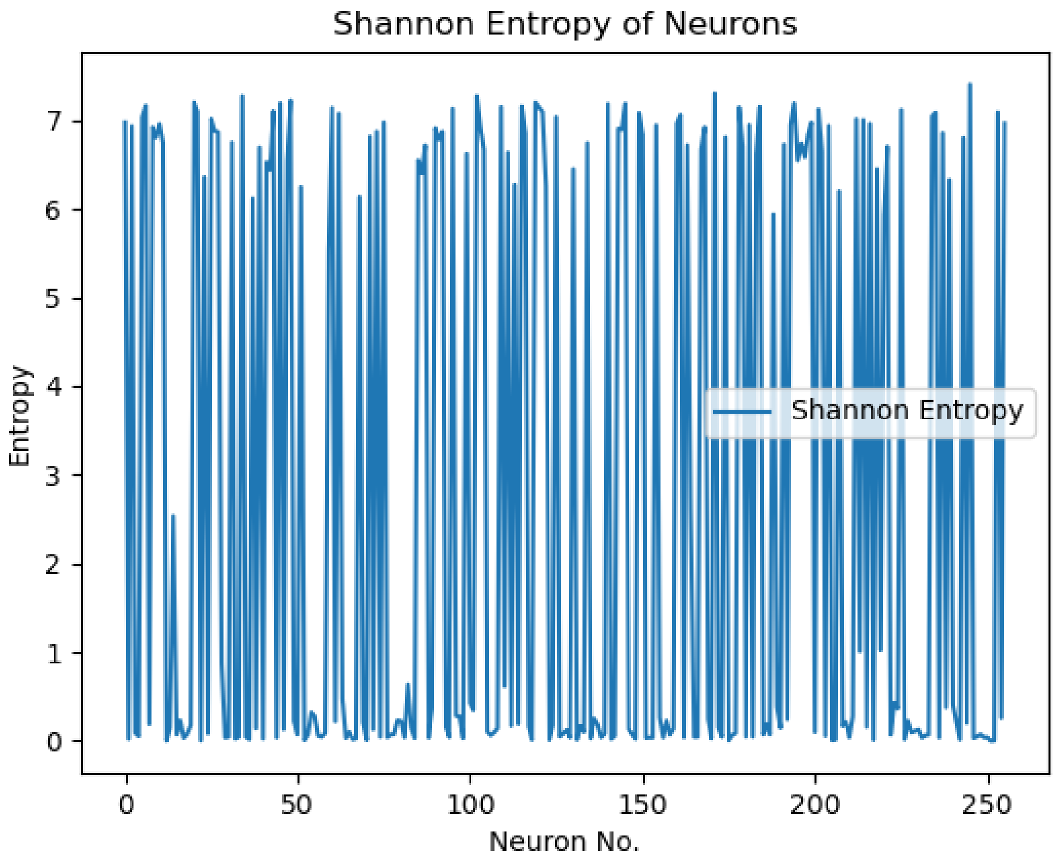


Figure 67

Hyperbolic Cosine function - layer neuron entropy

Hyperbolic Tangent

$$f(x) = \tanh(x) = \frac{\sinh(x)}{\cosh(x)}$$

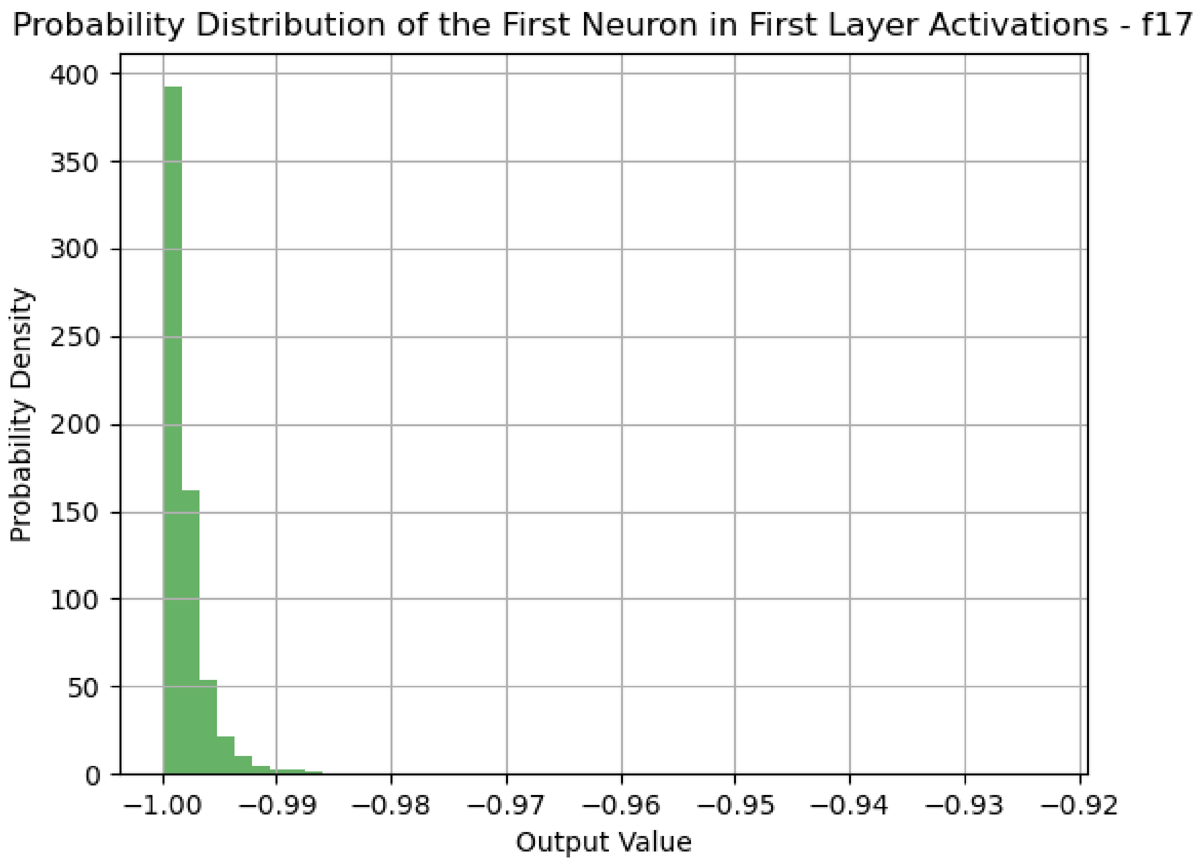


Figure 68

Hyperbolic Tangent function - first neuron output distribution

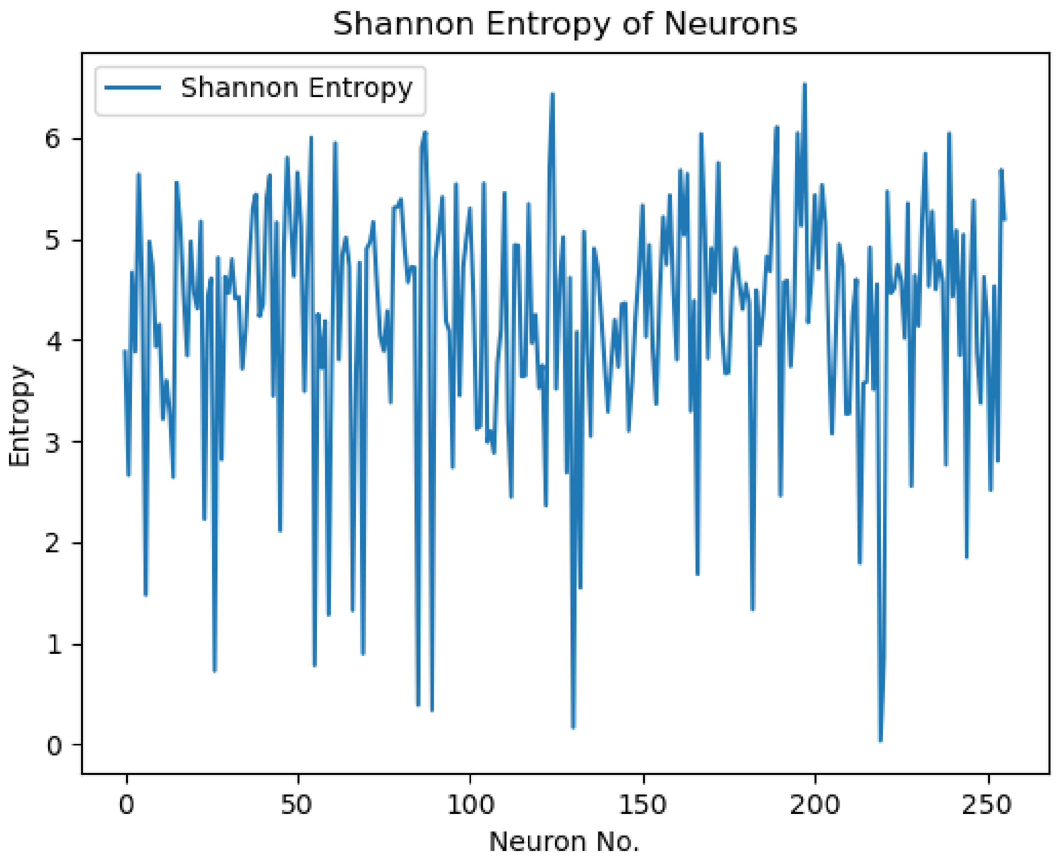


Figure 69

Hyperbolic Tangent function - layer neuron entropy

Natural Logarithm

$$f(x) = \ln(x)$$

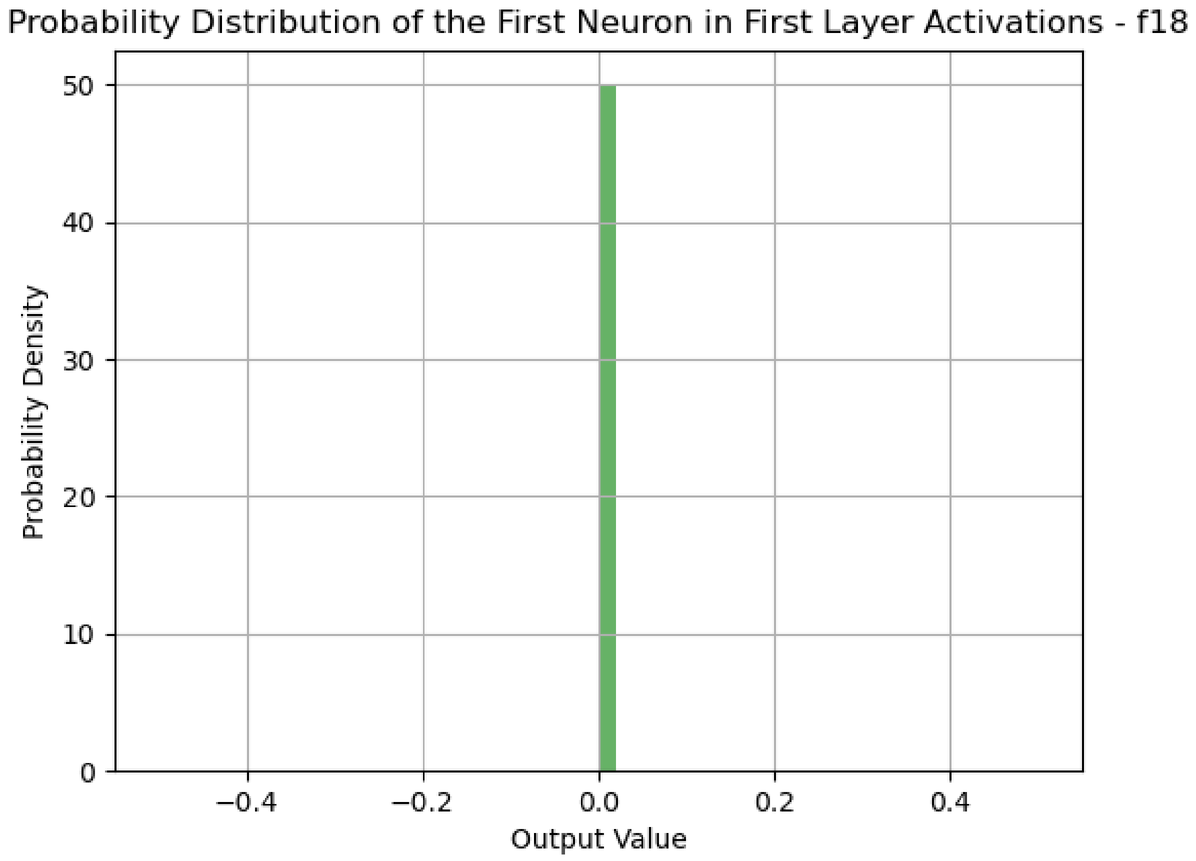


Figure 70

Natural Logarithm function - first neuron output distribution

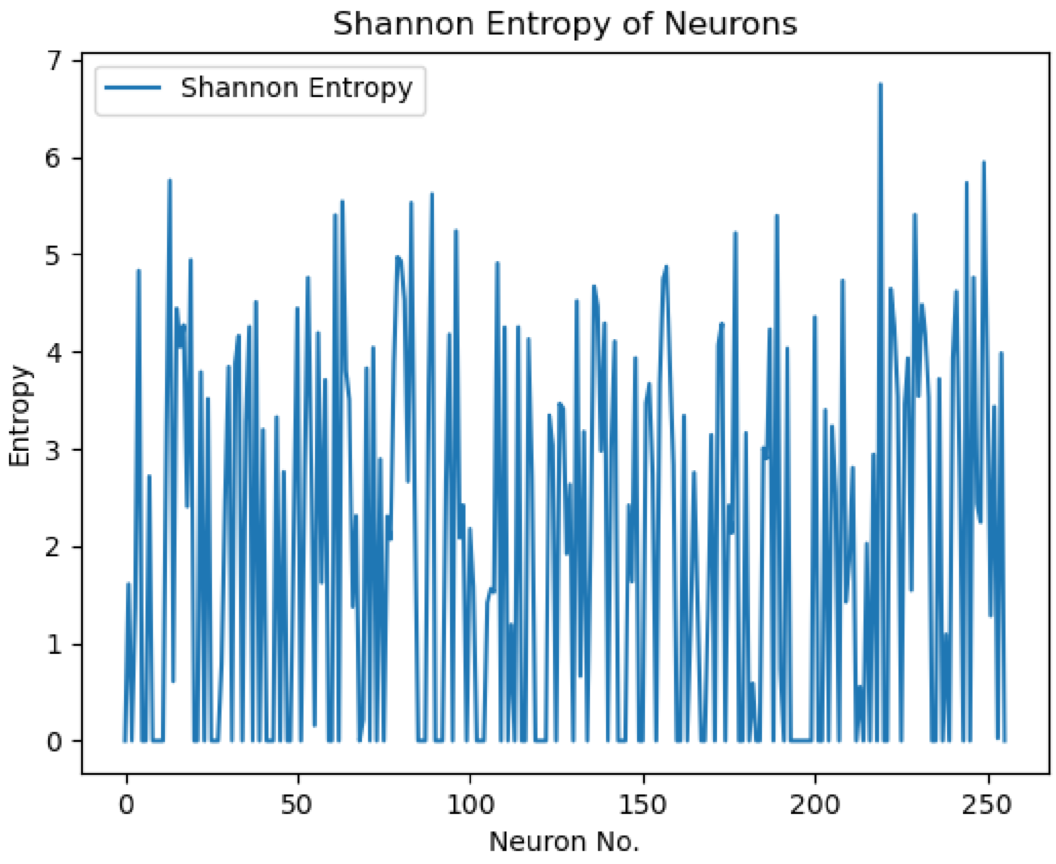


Figure 71

Natural Logarithm function - layer neuron entropy

Common Logarithm

$$f(x) = \log_{10}(x)$$

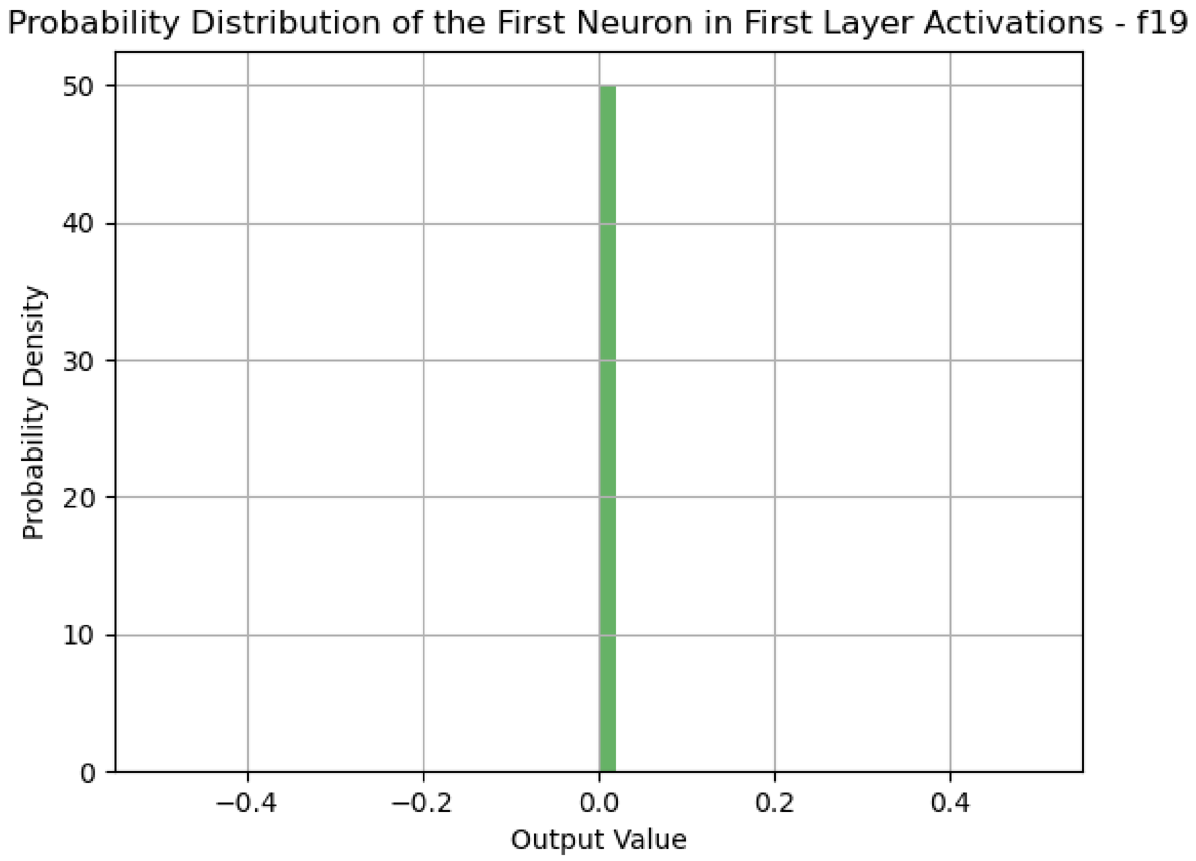


Figure 72

Common Logarithm function - first neuron output distribution

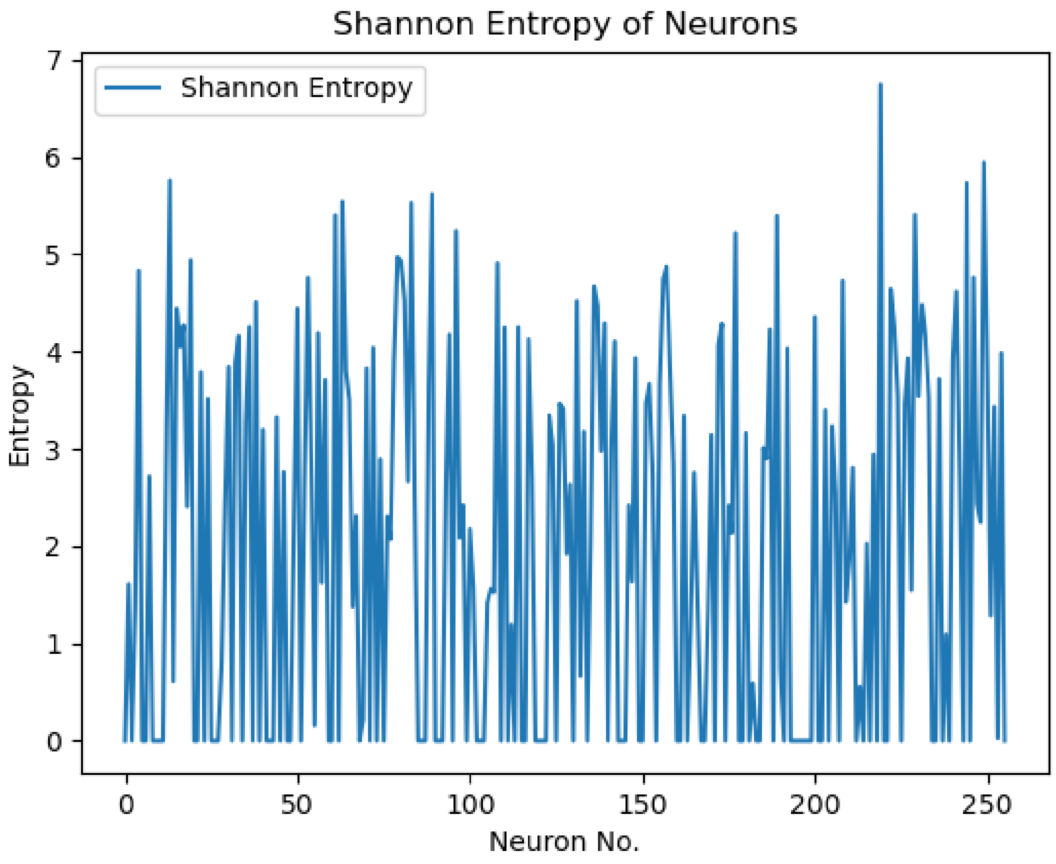


Figure 73

Common Logarithm function - layer neuron entropy

Exponential Function

$$f(x) = e^x$$

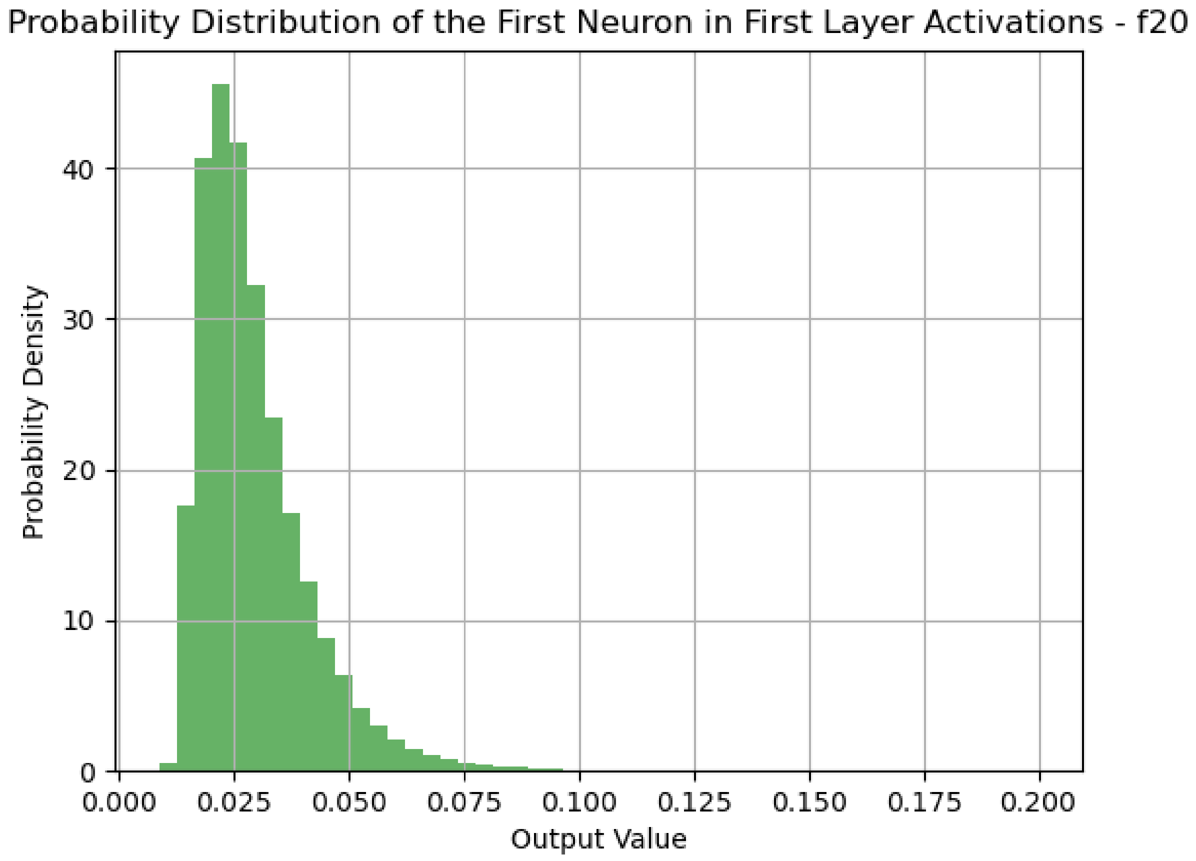


Figure 74

Exponential function - first neuron output distribution

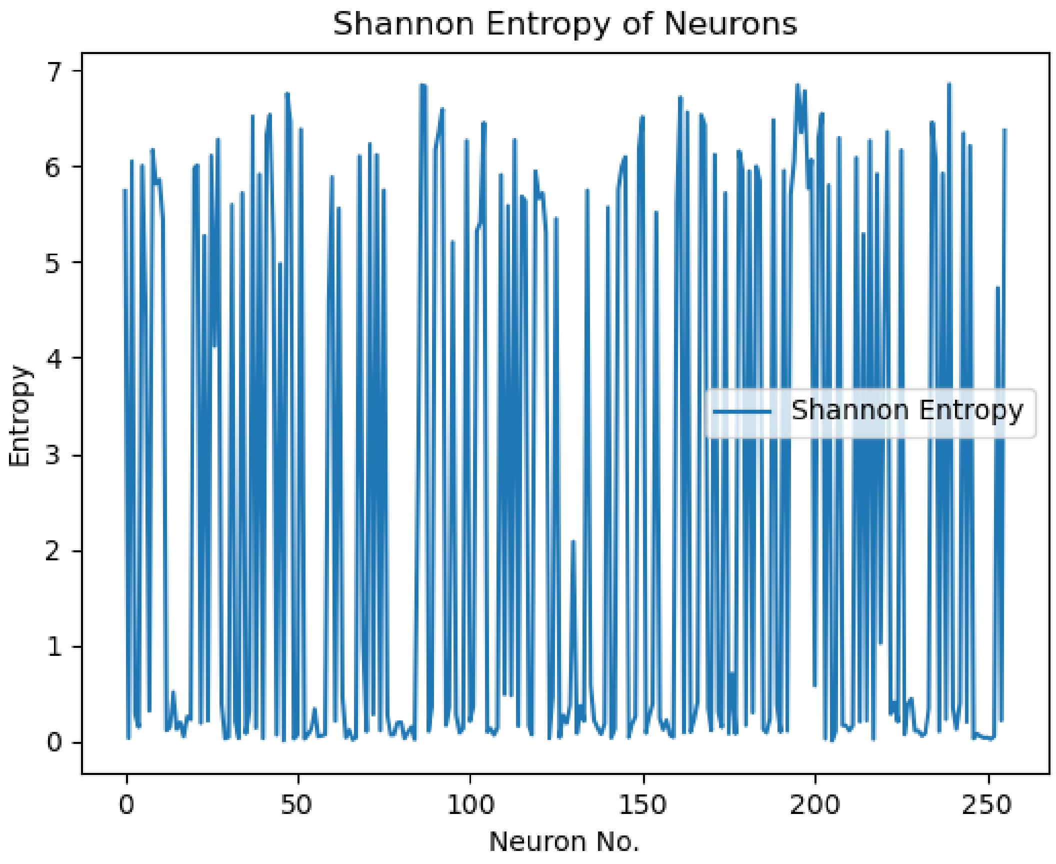


Figure 75

Exponential function - layer neuron entropy

APPENDIX C

UAM(UNIFIED ALGEBRAIC MEASURE) BETWEEN THE DISTRIBUTIONS FROM NEURONS AND CLASSICAL DISTRIBUTIONS

In this Appendix, We use KL Divergence as an example to show how we can calculate the KL divergence between the outputs of a specific neuron and the classical distributions or the mean distribution of the specific layer. We can select the maximum divergence to normalize the corresponding UAM(Unified Algebraic Measure). We can also use these reference distributions to compare different distributions.

MEAN DISTRIBUTION AND CLASSICAL DISTRIBUTIONS

Layer Neurons Mean Distribution

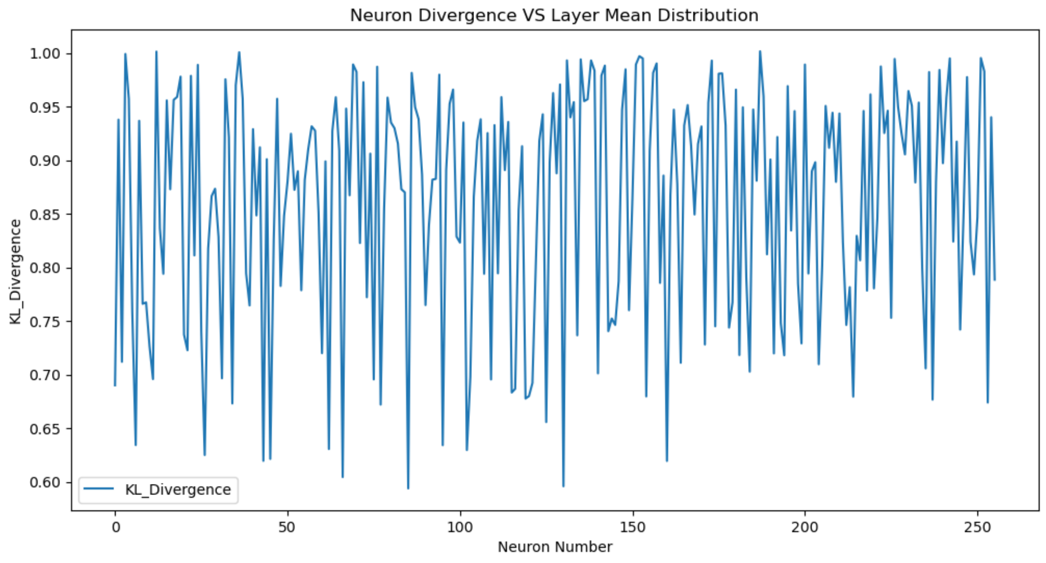


Figure 76

KL Divergence between Layer Outputs and Layer Neurons Mean Distribution

Normal Distribution

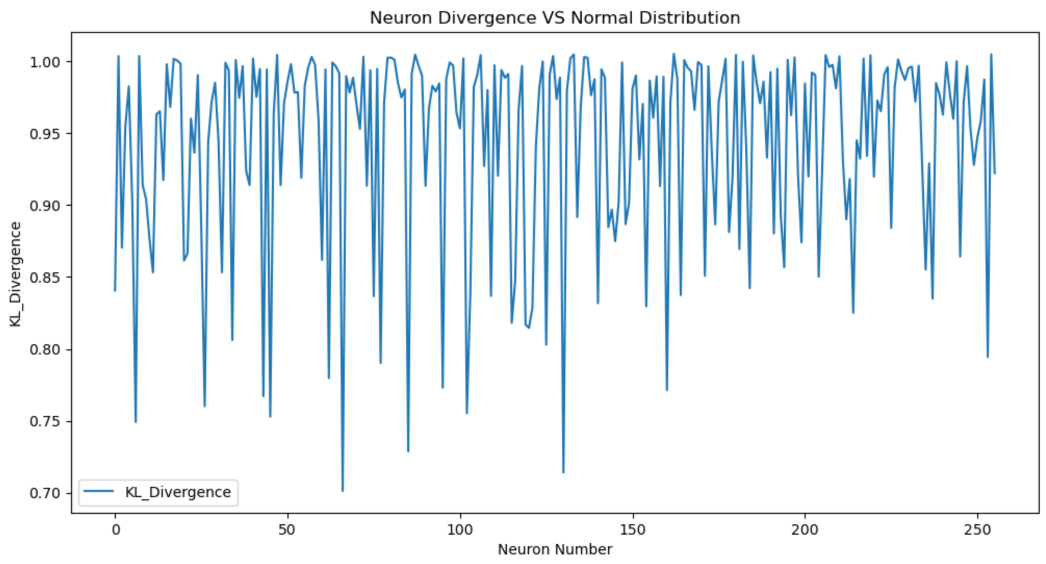


Figure 77

KL Divergence between Layer Outputs and Normal Distribution

Uniform Distribution

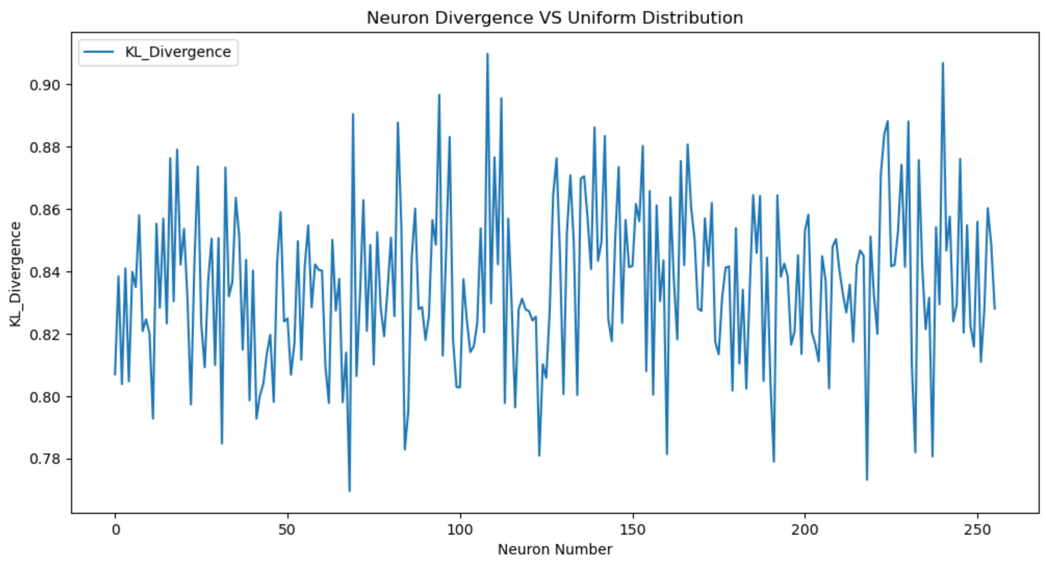


Figure 78

KL Divergence between Layer Outputs and Uniform Distribution

Exponential Distribution

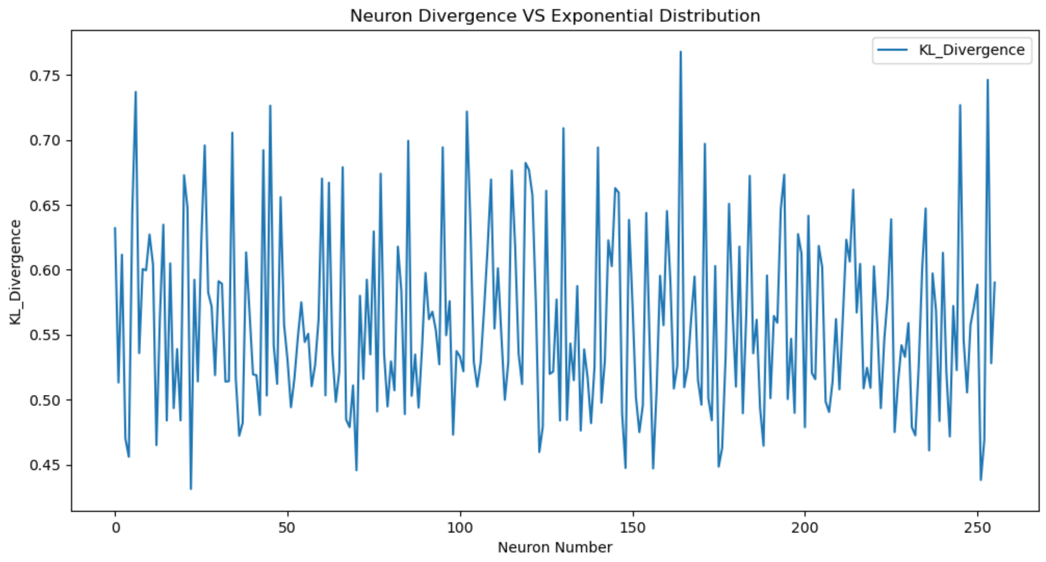


Figure 79

KL Divergence between Layer Outputs and Exponential Distribution

Poisson Distribution

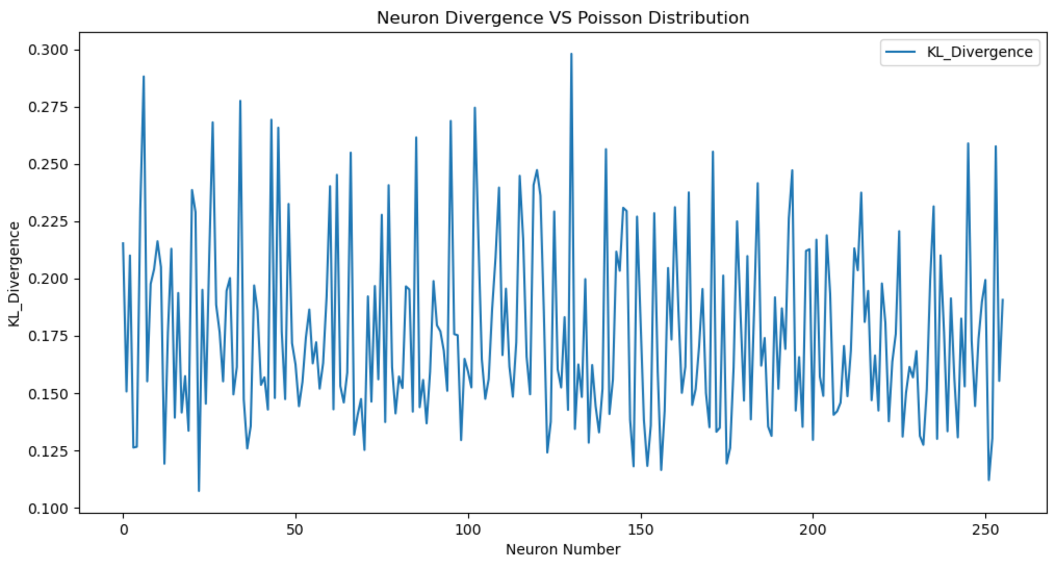


Figure 80

KL Divergence between Layer Outputs and Poisson Distribution

Binomial Distribution

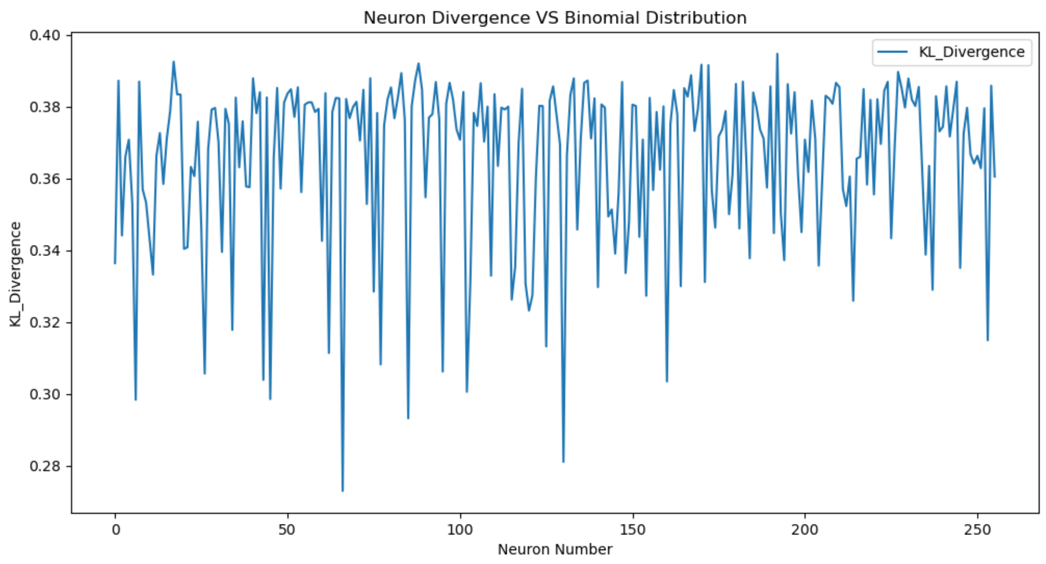


Figure 81

KL Divergence between Layer Outputs and Binomial Distribution

Beta Distribution

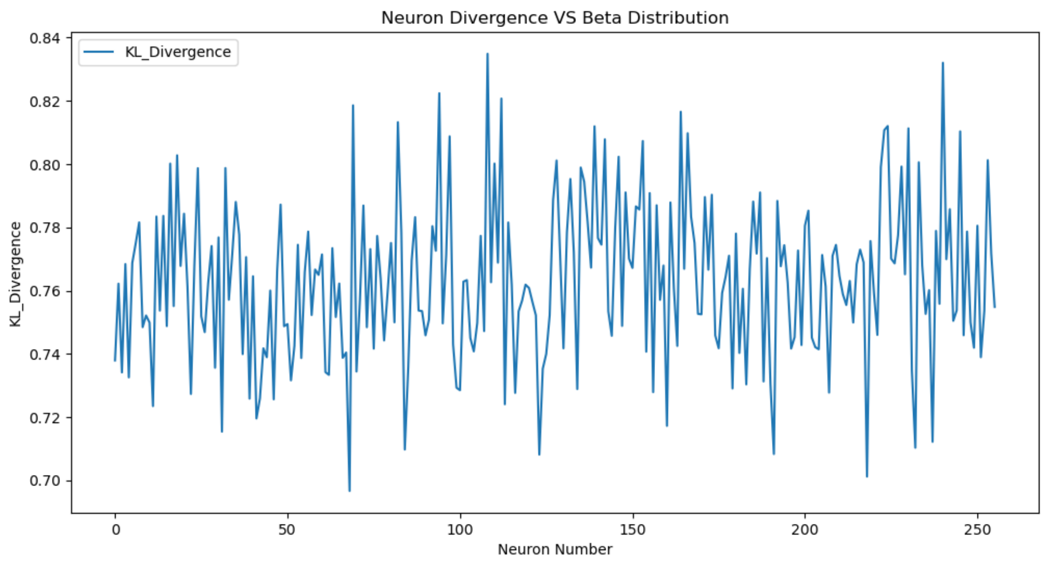


Figure 82

KL Divergence between Layer Outputs and Beta Distribution

Gamma Distribution

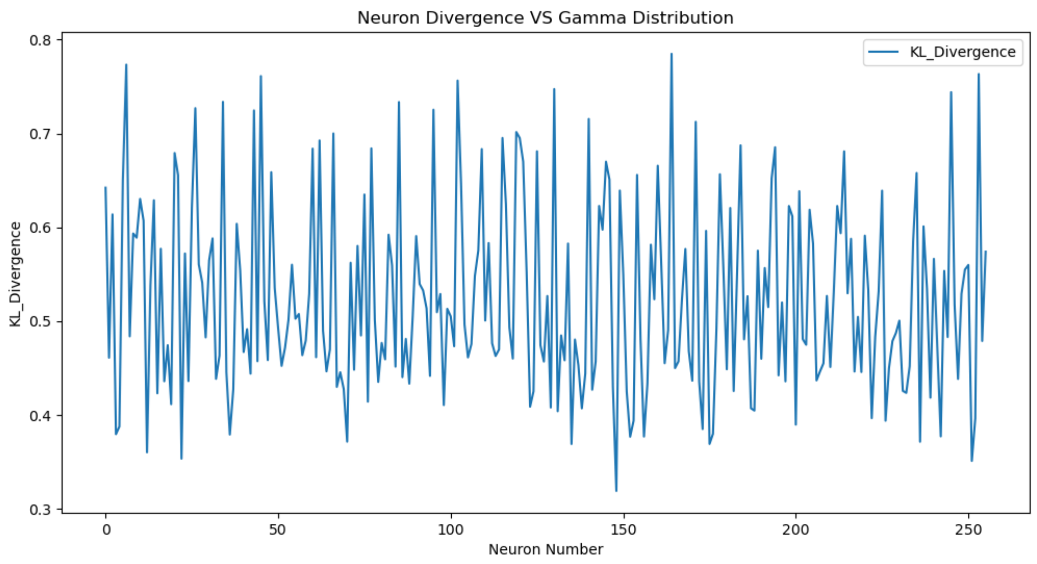


Figure 83

KL Divergence between Layer Outputs and Gamma Distribution

Chi-Squared Distribution

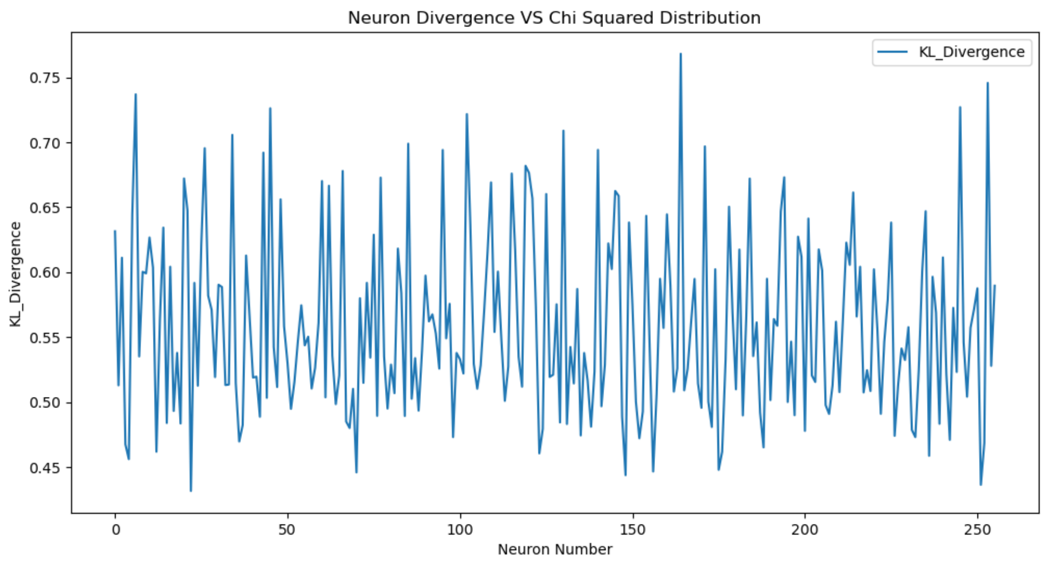


Figure 84

KL Divergence between Layer Outputs and Chi-Squared Distribution

Log-Normal Distribution

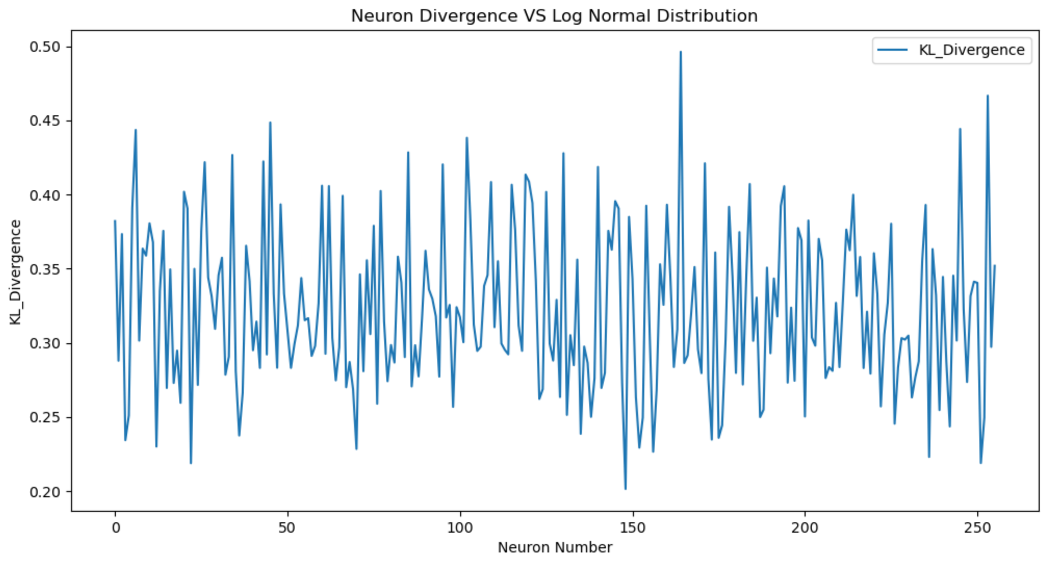


Figure 85

KL Divergence between Layer Outputs and Log-Normal Distribution

Cauchy Distribution

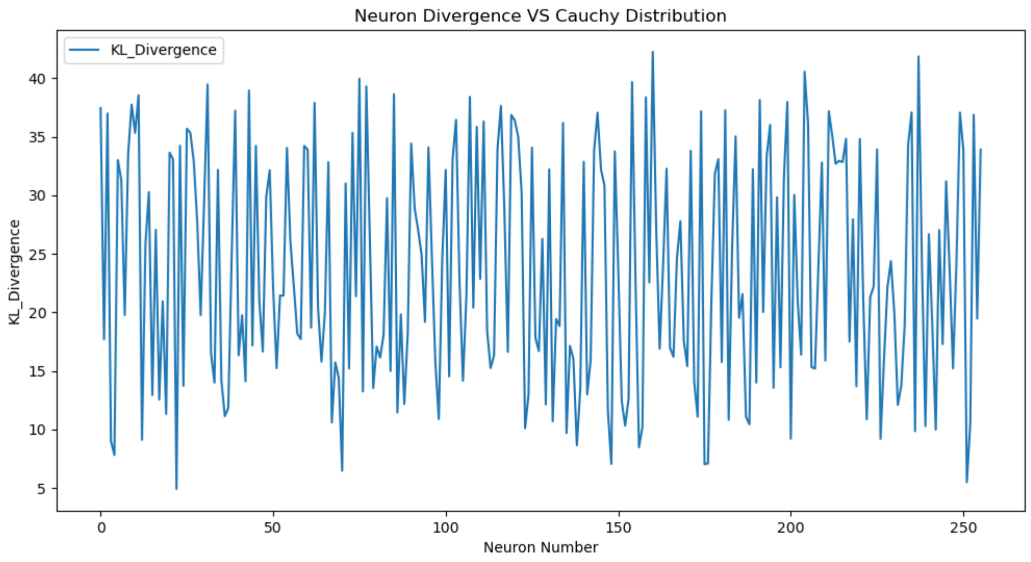


Figure 86

KL Divergence between Layer Outputs and Cauchy Distribution

APPENDIX D

STATISTICAL TESTS LIST

Table Overview of Statistical Tests

No	Test Name	Mathematical Formula	Principle	Citation
1	Kolmogorov-Smirnov Test	$D = \sup_x \lvert F_1(x) - F_2(x) \rvert$	Measures the maximum distance between the cumulative distribution functions of two samples.	[76]
2	Anderson-Darling Test	$A^2 = n \int_{-\infty}^{\infty} \frac{(F_n(x)-S(x))^2}{S(x)(1-S(x))} dS(x)$	places more weight on the tails of the distributions compared to the KS Test.	[3]
3	Chi-Square Test	$\chi^2 = \sum \frac{(O_i-E_i)^2}{E_i}$	Tests whether observed frequencies in categorical data match expected frequencies.	[107]
4	Cramér-von Mises Test	$\omega^2 = n \int_{-\infty}^{\infty} [F_n(x) - F(x)]^2 dF(x)$	Compares the squared differences between the empirical and theoretical CDFs.	[24]
5	Mann-Whitney U Test	U statistic based on ranks	Non-parametric test for assessing whether two independent samples come from the same distribution.	[92]

>>>Next Page

Continued

No	Test Name	Mathematical Formula	Principle	Citation
6	Wilcoxon Signed-Rank Test	Sum of signed ranks	Tests whether two paired samples come from the same distribution.	[140]
7	Shapiro-Wilk Test	W statistic	Tests the normality of a distribution.	[126]
8	Fisher's Exact Test	Exact probability calculation	Tests for independence in a 2x2 contingency table, suitable for small sample sizes.	[35]
9	F-Test of Equality of Variances	$F = \frac{s_1^2}{s_2^2}$	Compares the variances of two populations.	[123]
10	Levene's Test	Based on the mean absolute deviations from the group medians or means	Tests equality of variances across groups, more robust to non-normal distributions.	[86]
11	Log-Rank Test	Comparison of observed vs. expected survival times	Used in survival analysis to compare the survival distributions of two or more groups.	[93]

>>>Next Page

Continued

No	Test Name	Mathematical Formula	Principle	Citation
12	Mood's Median Test	χ^2 statistic based on counts above/below the grand median	Tests whether multiple samples come from populations with the same median.	[99]
13	Dixon's Q Test	$Q = \frac{\text{Gap}}{\text{Range}}$	Identifies outliers in a dataset.	[28]
14	Grubbs' Test	$G = \frac{\max \lvert X_i - \bar{X} \rvert}{s}$	Identifies outliers, particularly in normally distributed datasets.	[49]
15	McNemar's Test	$\chi^2 = \frac{(b-c)^2}{b+c}$	Tests for changes in proportions in paired nominal data.	[95]
16	Z-Test	$Z = \frac{(\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_2)}{\sqrt{\sigma_1^2/n_1 + \sigma_2^2/n_2}}$	Tests the difference between two means from different populations when variance is known.	[102]
17	Bartlett's Test	$K^2 = 2.3026 \times \frac{(N-k)(\log S_p^2 - \sum n_i \log s_i^2/N)}{1+1/(3(k-1))(\sum(1/n_i - 1/N) - 1/(N-k))}$	Tests for equality of variances across k groups, sensitive to non-normality.	[9]

>>>Next Page

Continued

No	Test Name	Mathematical Formula	Principle	Citation
18	Durbin-Watson Test	$d = \frac{\sum_{i=2}^{n}(e_i - e_{i-1})^2}{\sum_{i=1}^{n} e_i^2}$	Tests for autocorrelation in the residuals from a regression analysis.	[31]
19	Cochran's Q Test	$Q = \frac{(k-1)(\sum_{i=1}^{n}\sum_{j=1}^{k} x_{ij}^2 - T^2/n)}{k\sum_{j=1}^{k} t_j - T^2/n}$	Tests whether k treatments have the same effects.	[21]
20	Hotelling's T-squared Test	$T^2 = n(\mathbf{x}_1 - \mathbf{x}_2)'S^{-1}(\mathbf{x}_1 - \mathbf{x}_2)$	Used to compare the means of two groups in a multivariate setting.	[63]
21	Likelihood Ratio Test	$\lambda = 2(\log L(\hat{\theta}_1) - \log L(\hat{\theta}_0))$	Compares the fit of two nested models, usually within maximum likelihood estimation.	[102]
22	Gehan's Test	Based on the Wilcoxon rank-sum test, but pairs each observation from one group with all observations in the other group, applying weights to the ranks.	Used in biomedical research to compare survival distributions.	[42]
23	Greenhouse-Geisser Test	$\varepsilon = \frac{(n-1)(1-\text{Trace}(W)/\text{Trace}(W^2))}{(n-1)(1-\text{Trace}(W))+\text{Trace}(W^2)}$	Used when the sphericity assumption in repeated measures ANOVA is violated.	[48]

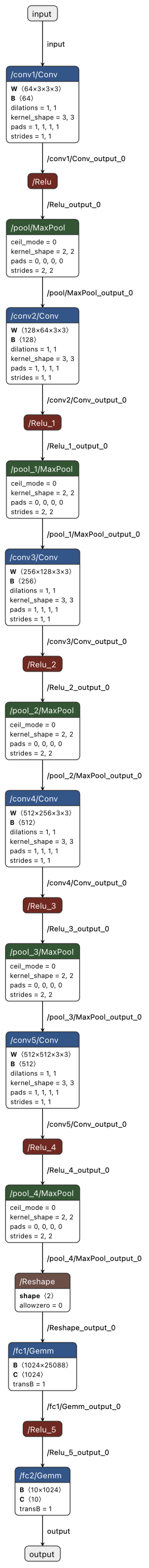
>>>Next Page

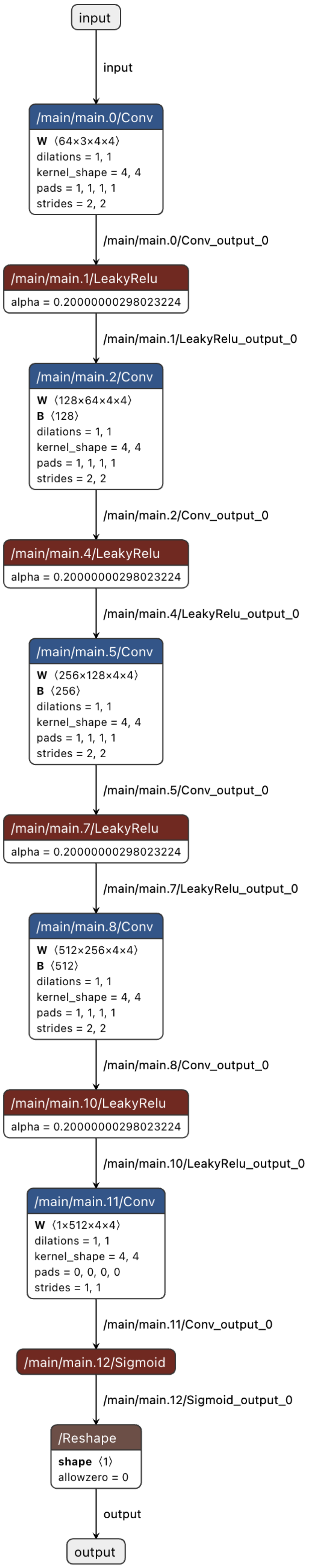
Continued

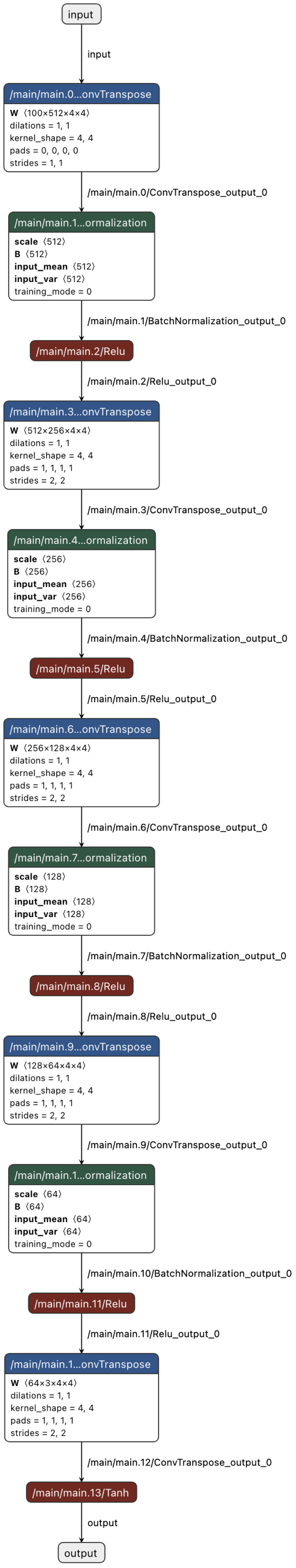
No	Test Name	Mathematical Formula	Principle	Citation
24	Hosmer-Lemeshow Test	$H = \sum \frac{(O-E)^2}{E}$	Assesses the goodness of fit for logistic regression models.	[62]
25	Mauchly's Test	The test statistic evaluates the hypothesis that the observed covariance matrix of the differences among repeated measures is proportional to an identity matrix.	Tests whether the assumption of sphericity is violated in designs involving repeated measures.	[94]
26	Fligner-Killeen Test	Based on medians of groups, not means, and computes a chi-square statistic from ranked variances.	Tests for equal variances among groups, especially when the data are not normally distributed.	[36]
27	Wald Test	$W = (\hat{\beta} - \beta_0)'[\text{Var}(\hat{\beta})]^{-1}(\hat{\beta} - \beta_0)$	Used to test the significance of individual regression coefficients.	[138]
28	Quade Test	The test involves ranking the responses within each block or subject and uses a variance analysis on the ranks.	An alternative to the Friedman test for non-parametric ANOVA with a repeated measures design.	[108]

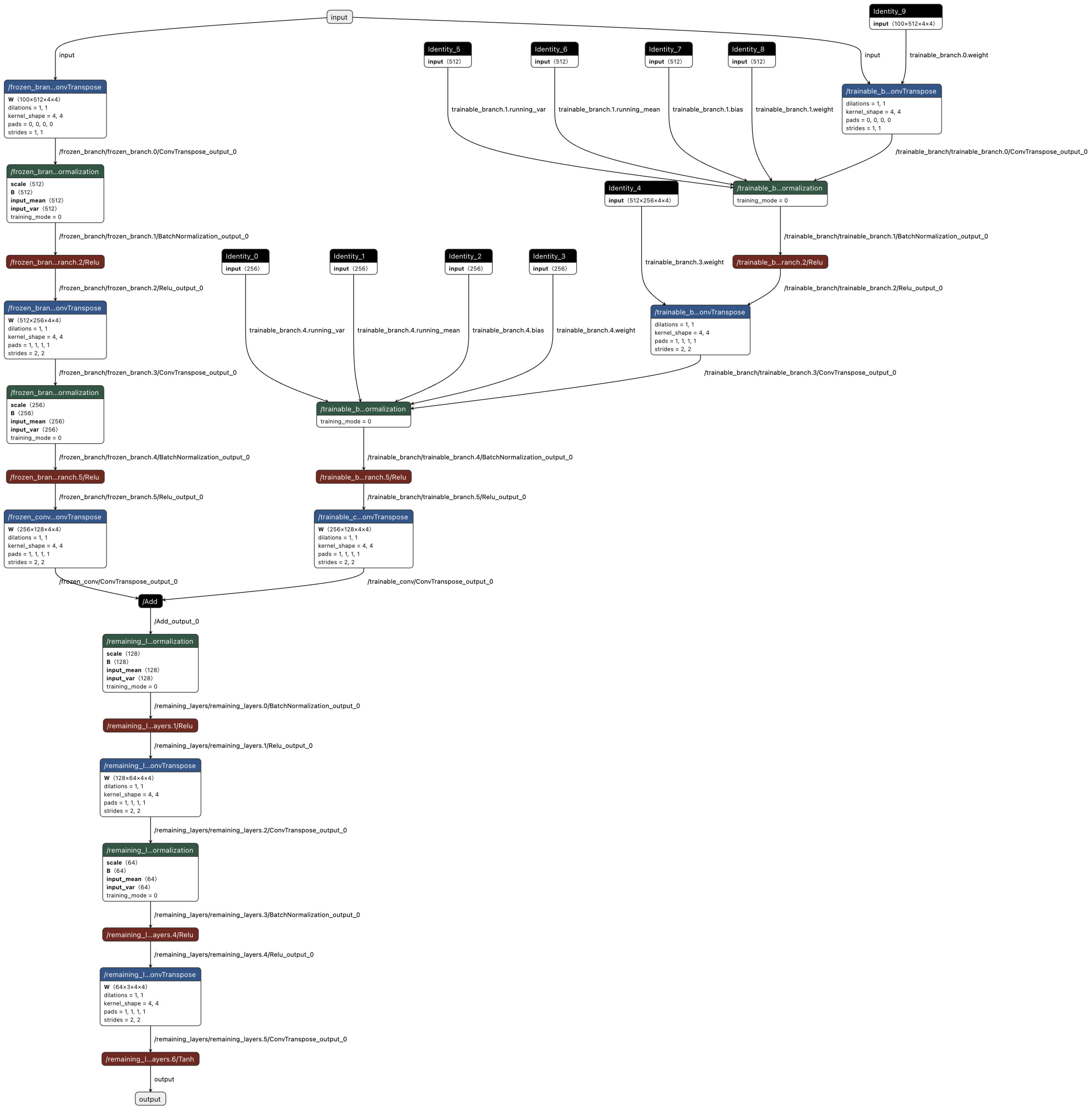
APPENDIX E

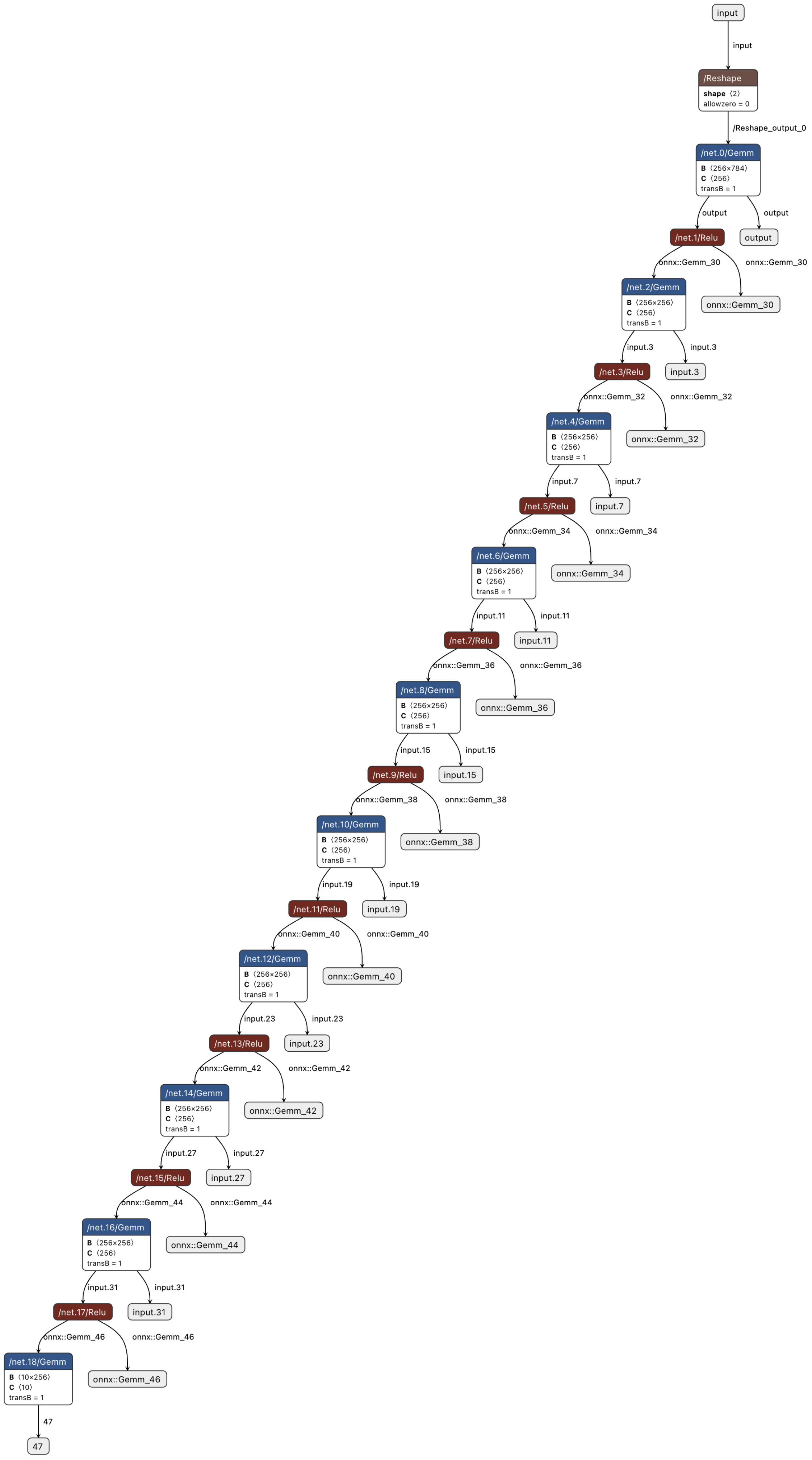
LARGER AND MORE LEGIBLE IMAGES OF THE NEURAL NETWORK ARCHITECTURES


input
input
/conv1/Conv
W ⟨64×3×3×3⟩
B ⟨64⟩
dilations = 1, 1
kernel_shape = 3, 3
pads = 1, 1, 1, 1
strides = 1, 1
/conv1/Conv_output_0
/Relu
/Relu_output_0
/pool/MaxPool
ceil_mode = 0
kernel_shape = 2, 2
pads = 0, 0, 0, 0
strides = 2, 2
/pool/MaxPool_output_0
/conv2/Conv
W ⟨128×64×3×3⟩
B ⟨128⟩
dilations = 1, 1
kernel_shape = 3, 3
pads = 1, 1, 1, 1
strides = 1, 1
/conv2/Conv_output_0
/Relu_1
/Relu_1_output_0
/pool_1/MaxPool
ceil_mode = 0
kernel_shape = 2, 2
pads = 0, 0, 0, 0
strides = 2, 2
/pool_1/MaxPool_output_0
/conv3/Conv
W ⟨256×128×3×3⟩
B ⟨256⟩
dilations = 1, 1
kernel_shape = 3, 3
pads = 1, 1, 1, 1
strides = 1, 1
/conv3/Conv_output_0
/Relu_2
/Relu_2_output_0
/pool_2/MaxPool
ceil_mode = 0
kernel_shape = 2, 2
pads = 0, 0, 0, 0
strides = 2, 2
/pool_2/MaxPool_output_0
/conv4/Conv
W ⟨512×256×3×3⟩
B ⟨512⟩
dilations = 1, 1
kernel_shape = 3, 3
pads = 1, 1, 1, 1
strides = 1, 1
/conv4/Conv_output_0
/Relu_3
/Relu_3_output_0
/pool_3/MaxPool
ceil_mode = 0
kernel_shape = 2, 2
pads = 0, 0, 0, 0
strides = 2, 2
/pool_3/MaxPool_output_0
/conv5/Conv
W ⟨512×512×3×3⟩
B ⟨512⟩
dilations = 1, 1
kernel_shape = 3, 3
pads = 1, 1, 1, 1
strides = 1, 1
/conv5/Conv_output_0
/Relu_4
/Relu_4_output_0
/pool_4/MaxPool
ceil_mode = 0
kernel_shape = 2, 2
pads = 0, 0, 0, 0
strides = 2, 2
/pool_4/MaxPool_output_0
/Reshape
shape ⟨2⟩
allowzero = 0
/Reshape_output_0
/fc1/Gemm
B ⟨1024×25088⟩
C ⟨1024⟩
transB = 1
/fc1/Gemm_output_0
/Relu_5
/Relu_5_output_0
/fc2/Gemm
B ⟨10×1024⟩
C ⟨10⟩
transB = 1
output
output


input
input
/main/main.0/Conv
W ⟨64×3×4×4⟩
dilations = 1, 1
kernel_shape = 4, 4
pads = 1, 1, 1, 1
strides = 2, 2
/main/main.0/Conv_output_0
/main/main.1/LeakyRelu
alpha = 0.20000000298023224
/main/main.1/LeakyRelu_output_0
/main/main.2/Conv
W ⟨128×64×4×4⟩
B ⟨128⟩
dilations = 1, 1
kernel_shape = 4, 4
pads = 1, 1, 1, 1
strides = 2, 2
/main/main.2/Conv_output_0
/main/main.4/LeakyRelu
alpha = 0.20000000298023224
/main/main.4/LeakyRelu_output_0
/main/main.5/Conv
W ⟨256×128×4×4⟩
B ⟨256⟩
dilations = 1, 1
kernel_shape = 4, 4
pads = 1, 1, 1, 1
strides = 2, 2
/main/main.5/Conv_output_0
/main/main.7/LeakyRelu
alpha = 0.20000000298023224
/main/main.7/LeakyRelu_output_0
/main/main.8/Conv
W ⟨512×256×4×4⟩
B ⟨512⟩
dilations = 1, 1
kernel_shape = 4, 4
pads = 1, 1, 1, 1
strides = 2, 2
/main/main.8/Conv_output_0
/main/main.10/LeakyRelu
alpha = 0.20000000298023224
/main/main.10/LeakyRelu_output_0
/main/main.11/Conv
W ⟨1×512×4×4⟩
dilations = 1, 1
kernel_shape = 4, 4
pads = 0, 0, 0, 0
strides = 1, 1
/main/main.11/Conv_output_0
/main/main.12/Sigmoid
/main/main.12/Sigmoid_output_0
/Reshape
shape ⟨1⟩
allowzero = 0
output
output


input
input
/main/main.0...onvTranspose
W ⟨100×512×4×4⟩
dilations = 1, 1
kernel_shape = 4, 4
pads = 0, 0, 0, 0
strides = 1, 1
/main/main.0/ConvTranspose_output_0
/main/main.1...ormalization
scale ⟨512⟩
B ⟨512⟩
input_mean ⟨512⟩
input_var ⟨512⟩
training_mode = 0
/main/main.1/BatchNormalization_output_0
/main/main.2/Relu
/main/main.2/Relu_output_0
/main/main.3...onvTranspose
W ⟨512×256×4×4⟩
dilations = 1, 1
kernel_shape = 4, 4
pads = 1, 1, 1, 1
strides = 2, 2
/main/main.3/ConvTranspose_output_0
/main/main.4...ormalization
scale ⟨256⟩
B ⟨256⟩
input_mean ⟨256⟩
input_var ⟨256⟩
training_mode = 0
/main/main.4/BatchNormalization_output_0
/main/main.5/Relu
/main/main.5/Relu_output_0
/main/main.6...onvTranspose
W ⟨256×128×4×4⟩
dilations = 1, 1
kernel_shape = 4, 4
pads = 1, 1, 1, 1
strides = 2, 2
/main/main.6/ConvTranspose_output_0
/main/main.7...ormalization
scale ⟨128⟩
B ⟨128⟩
input_mean ⟨128⟩
input_var ⟨128⟩
training_mode = 0
/main/main.7/BatchNormalization_output_0
/main/main.8/Relu
/main/main.8/Relu_output_0
/main/main.9...onvTranspose
W ⟨128×64×4×4⟩
dilations = 1, 1
kernel_shape = 4, 4
pads = 1, 1, 1, 1
strides = 2, 2
/main/main.9/ConvTranspose_output_0
/main/main.1...ormalization
scale ⟨64⟩
B ⟨64⟩
input_mean ⟨64⟩
input_var ⟨64⟩
training_mode = 0
/main/main.10/BatchNormalization_output_0
/main/main.11/Relu
/main/main.11/Relu_output_0
/main/main.1...onvTranspose
W ⟨64×3×4×4⟩
dilations = 1, 1
kernel_shape = 4, 4
pads = 1, 1, 1, 1
strides = 2, 2
/main/main.12/ConvTranspose_output_0
/main/main.13/Tanh
output
output


input
Identity_9
input (100×512×4×4)
Identity_5
input (512)
Identity_6
input (512)
Identity_7
input (512)
Identity_8
input (512)
input
trainable_branch.0.weight
input
/frozen_bran...onvTranspose
W (100×512×4×4)
dilations = 1, 1
kernel_shape = 4, 4
pads = 0, 0, 0, 0
strides = 1, 1
trainable_branch.1.running_var
trainable_branch.1.running_mean
trainable_branch.1.bias
trainable_branch.1.weight
/trainable_b...onvTranspose
dilations = 1, 1
kernel_shape = 4, 4
pads = 0, 0, 0, 0
strides = 1, 1
/frozen_branch/frozen_branch.0/ConvTranspose_output_0
/trainable_branch/trainable_branch.0/ConvTranspose_output_0
/frozen_bran...ormalization
scale (512)
B (512)
input_mean (512)
input_var (512)
training_mode = 0
Identity_4
input (512×256×4×4)
/trainable_b...ormalization
training_mode = 0
/frozen_branch/frozen_branch.1/BatchNormalization_output_0
/trainable_branch/trainable_branch.1/BatchNormalization_output_0
Identity_0
input (256)
Identity_1
input (256)
Identity_2
input (256)
Identity_3
input (256)
/frozen_bran...ranch.2/Relu
trainable_branch.3.weight
/trainable_b...ranch.2/Relu
/frozen_branch/frozen_branch.2/Relu_output_0
/trainable_branch/trainable_branch.2/Relu_output_0
/frozen_bran...onvTranspose
W (512×256×4×4)
dilations = 1, 1
kernel_shape = 4, 4
pads = 1, 1, 1, 1
strides = 2, 2
trainable_branch.4.running_var
trainable_branch.4.running_mean
trainable_branch.4.bias
trainable_branch.4.weight
/trainable_b...onvTranspose
dilations = 1, 1
kernel_shape = 4, 4
pads = 1, 1, 1, 1
strides = 2, 2
/frozen_branch/frozen_branch.3/ConvTranspose_output_0
/trainable_branch/trainable_branch.3/ConvTranspose_output_0
/frozen_bran...ormalization
scale (256)
B (256)
input_mean (256)
input_var (256)
training_mode = 0
/trainable_b...ormalization
training_mode = 0
/frozen_branch/frozen_branch.4/BatchNormalization_output_0
/trainable_branch/trainable_branch.4/BatchNormalization_output_0
/frozen_bran...ranch.5/Relu
/trainable_b...ranch.5/Relu
/frozen_branch/frozen_branch.5/Relu_output_0
/trainable_branch/trainable_branch.5/Relu_output_0
/frozen_conv...onvTranspose
W (256×128×4×4)
dilations = 1, 1
kernel_shape = 4, 4
pads = 1, 1, 1, 1
strides = 2, 2
/trainable_c...onvTranspose
W (256×128×4×4)
dilations = 1, 1
kernel_shape = 4, 4
pads = 1, 1, 1, 1
strides = 2, 2
/frozen_conv/ConvTranspose_output_0
/trainable_conv/ConvTranspose_output_0
/Add
/Add_output_0
/remaining_l...ormalization
scale (128)
B (128)
input_mean (128)
input_var (128)
training_mode = 0
/remaining_layers/remaining_layers.0/BatchNormalization_output_0
/remaining_l...ayers.1/Relu
/remaining_layers/remaining_layers.1/Relu_output_0
/remaining_l...onvTranspose
W (128×64×4×4)
dilations = 1, 1
kernel_shape = 4, 4
pads = 1, 1, 1, 1
strides = 2, 2
/remaining_layers/remaining_layers.2/ConvTranspose_output_0
/remaining_l...ormalization
scale (64)
B (64)
input_mean (64)
input_var (64)
training_mode = 0
/remaining_layers/remaining_layers.3/BatchNormalization_output_0
/remaining_l...ayers.4/Relu
/remaining_layers/remaining_layers.4/Relu_output_0
/remaining_l...onvTranspose
W (64×3×4×4)
dilations = 1, 1
kernel_shape = 4, 4
pads = 1, 1, 1, 1
strides = 2, 2
/remaining_layers/remaining_layers.5/ConvTranspose_output_0
/remaining_l...ayers.6/Tanh
output
output


input
input
/Reshape
shape ⟨2⟩
allowzero = 0
/Reshape_output_0
/net.0/Gemm
B ⟨256×784⟩
C ⟨256⟩
transB = 1
output
output
/net.1/Relu
output
onnx::Gemm_30
onnx::Gemm_30
/net.2/Gemm
B ⟨256×256⟩
C ⟨256⟩
transB = 1
onnx::Gemm_30
input.3
input.3
/net.3/Relu
input.3
onnx::Gemm_32
onnx::Gemm_32
/net.4/Gemm
B ⟨256×256⟩
C ⟨256⟩
transB = 1
onnx::Gemm_32
input.7
input.7
/net.5/Relu
input.7
onnx::Gemm_34
onnx::Gemm_34
/net.6/Gemm
B ⟨256×256⟩
C ⟨256⟩
transB = 1
onnx::Gemm_34
input.11
input.11
/net.7/Relu
input.11
onnx::Gemm_36
onnx::Gemm_36
/net.8/Gemm
B ⟨256×256⟩
C ⟨256⟩
transB = 1
onnx::Gemm_36
input.15
input.15
/net.9/Relu
input.15
onnx::Gemm_38
onnx::Gemm_38
/net.10/Gemm
B ⟨256×256⟩
C ⟨256⟩
transB = 1
onnx::Gemm_38
input.19
input.19
/net.11/Relu
input.19
onnx::Gemm_40
onnx::Gemm_40
/net.12/Gemm
B ⟨256×256⟩
C ⟨256⟩
transB = 1
onnx::Gemm_40
input.23
input.23
/net.13/Relu
input.23
onnx::Gemm_42
onnx::Gemm_42
/net.14/Gemm
B ⟨256×256⟩
C ⟨256⟩
transB = 1
onnx::Gemm_42
input.27
input.27
/net.15/Relu
input.27
onnx::Gemm_44
onnx::Gemm_44
/net.16/Gemm
B ⟨256×256⟩
C ⟨256⟩
transB = 1
onnx::Gemm_44
input.31
input.31
/net.17/Relu
input.31
onnx::Gemm_46
onnx::Gemm_46
/net.18/Gemm
B ⟨10×256⟩
C ⟨10⟩
transB = 1
onnx::Gemm_46
47
47

VITA

Li Dai received his B.S. in Computer Science from North China University of Technology and an M.S. in Computer Science from the Beijing University of Technology, where his research focused on Internet statistics algorithms. He joined the University of Tennessee at Chattanooga (UTC) as an Adjunct Faculty and PhD candidate in Computer Science in August 2014, specializing in Data Science with a focus on Deep Learning.

Professionally, Mr. Dai served as R&D CTO at several companies in Beijing, leading the development of various enterprise management and educational systems. He has also held positions such as Product Development Manager and CTO at the Beijing R&D Center of SuntechGroup and CNNIC.

His teaching at UTC includes courses ranging from Introduction to Computing to Advanced Programming Languages. Li Dai's research interests encompass deep learning, transfer learning, reinforcement learning, and the application of AI methodologies in educational software, enterprise systems, and stock trading systems.