

Investigating the Evolution of Modularity in Neural Networks

by
Andreas Werle van der Merwe

*Thesis presented in partial fulfilment of the requirements for the degree
of Master of Engineering (Mechatronic) in the Faculty of Engineering at
Stellenbosch University*



Supervisor: Professor Dawie van der Heever
Co-supervisor: Doctor Stefan du Plessis

March 2020

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

March 2020

Copyright © 2020 Stellenbosch University

All rights reserved

Abstract

Neural networks are not inherently interpretable as a direct consequence of their operating principle and the high dimensional opacity of their internal computations. The neural network interpretability problem is detrimental to reliability, meaningful human-AI interaction and the ethics of deployment. The problem can be approached from the perspective of neural modularity which frames a *modular* network as one that contains any number of disjoint subnetworks and identifies an *interpretable* modular network as one that groups its internal representations within such subnetworks in an explanative, task specific way. This study aims to investigate how neural modularity evolves and how it can benefit interpretability. HyperNEAT under connectivity constraints is the chosen neuroevolutionary method, and the following key points of research are investigated with respect to the evolution of neural modularity: general substrates, a variety of connection cost and novel input competition constraints, HyperNEAT modifications based on CPPN disjoints, and the interaction between lifetime and evolutionary learning with neuron nomination given the inclusion of a training phase. The results indicate that the connectivity constraints successfully promote the evolution of neural modularity across a variety of tasks on a general substrate and show that the novel input competition constraints are competitive with the established connection costs as a means of driving the evolution of neural modularity. The HyperNEAT modifications based on CPPN disjoints did not benefit the evolution of neural modularity. Investigating the interaction between lifetime and evolutionary learning with neuron nomination links greater concurrency between the processes that determine a network's form and function with higher levels of evolved neural modularity for the connection cost constraints. The interpretability assessment shows that while the evolved networks' interpretable qualities are task dependent, two connectivity constraints deliver statistically different functional module overlap distributions. This study highlights new possibilities for future research and contributes to the knowledge basis on evolving neural modularity by showing that input competition constraints are competitive with connection cost constraints, by examining how the interaction between lifetime and evolutionary learning influences the evolution of neural modularity as well as looking at the interpretable qualities of the evolved networks.

Opsomming

Neurale netwerke is nie inherent interpreteerbaar nie as 'n gevolg van hulle werkbeginsel en die hoë dimensionele ondeursigtigheid van hulle interne berekeninge. Die neurale netwerk interpreteerbaarheidsprobleem is nadelig tot betroubaarheid, betekenisvolle mens-AI-interaksie en die etiek van ontplooiing. Die probleem kan benader word vanuit die perspektief van neurale modulariteit. 'n Modulêre netwerk bevat 'n aantal afwykende subnetwerke. 'n Interpreteerbare modulêre netwerk is 'n neurale netwerk met interne voorstellings wat binne sulke afwykende subnetwerke op 'n verklarende en taakspesifieke manier groepeer is. Hierdie studie ondersoek hoe neurale modulariteit ontwikkel en hoe dit interpreteerbaarheid kan bevoordeel. HyperNEAT met konnektiwiteitsbeperkings is die gekose neuro-evolutionêre metode en die volgende navorsing sleutelpunte word ondersoek met betrekking tot die evolusie van neurale modulariteit: (1) algemene raamwerke, (2) 'n verskeidenheid van verbindingskoste en nuwe insetkompetisiebeperkings, (3) HyperNEAT-modifikasies gebaseer op CPPN-afwykings, en (4) die interaksie tussen lewenslange en evolutionêre leer met neuronominasie gegewe die insluiting van 'n opleidingsfase. Die resultate dui daarop dat die konnektiwiteitsbeperkings die evolusie van neurale modulariteit oor 'n verskeidenheid take op 'n algemene raamwerk suksesvol bevorder en toon dat die nuwe insetkompetisiebeperkings mededingend is met die vasgestelde verbindingskoste as 'n manier om die evolusie van neurale modulariteit te dryf. Die HyperNEAT-modifikasies gebaseer op CPPN-afwykings het nie die ontwikkeling van neurale modulariteit bevoordeel nie. Die ondersoek van interaksie tussen leeftyd en evolutionêre leer met neuronominasie dui aan dat hoër neurale modulariteits vlakke met verbindingskoste veroorsaak word 'n groter mate van samewerking tussen die prosesse wat die vorm en funksie van 'n netwerk bepaal. Die interpretasie evaluering toon aan dat alhoewel die ontwikkelde netwerke se interpreteerbare eienskappe taakafhanklik is, die funksie-oorvleuel van twee konnektiwiteitsbeperkings oorvleuel is statisties verskillend. Hierdie studie belig nuwe moontlikhede vir toekomstige navorsing en dra by tot die kennisbasis oor die ontwikkeling van neurale modulariteit deur aan te toon dat insetkompetisiebeperkings mededingend is met verbindingskoste beperkings, deur te ondersoek hoe die interaksie tussen leeftyd en evolutionêre leer die ontwikkeling van neurale modulariteit beïnvloed, sowel as die ondersoek van die interpreteerbare eienskappe van die ontwikkelde netwerke.

Acknowledgements

I would like to thank my project supervisor, Professor Dawie van den Heever, and my co-supervisor, Dr Stefan Du Plessis, for their guidance and enthusiasm as well as for giving me the freedom to pursue the type of research that I am passionate about. I would like to thank Cognitive Systems (Pty) Limited (Cape Town, South Africa) for the valuable opportunity to do an internship with them and for giving me a bursary for my second year of masters. I would like to thank Martha Asiimwe, Dale Sparrow, Erika Braune and Matthew Wawn for their support and help in proofreading the thesis document. Finally, I would like to thank my family for their love and for making it possible for me to study engineering at a postgraduate level.

Table of Contents

	Page
Declaration	i
Abstract.....	ii
Opsomming.....	iii
Acknowledgements	iv
Table of Contents	v
List of Figures.....	vii
List of Tables	xiii
Nomenclature	xiv
1 Introduction.....	1
1.1 Background and Motivation	1
1.2 Aim and Objectives	2
1.3 Chapter Overviews	2
2 Literature Review	3
2.1 Neural Networks	3
2.1.1 Deconstructing AI's Wunderkind.....	3
2.1.2 Fundamental Principles	4
2.1.3 Adversarial Examples.....	10
2.1.4 XAI: Towards Meaningful Interpretability	12
2.2 Interpretability through Modularity	15
2.2.1 Modularity as a Concept.....	15
2.2.2 Interpretability and Neural Modularity	16
2.3 Evolving Modular Neural Networks	20
2.3.1 Evolutionary Computation	20
2.3.2 Factors of Evolutionary Success	25
2.3.3 Neuroevolution	29
2.3.4 Analysis of Neuroevolutionary Encoding Strategies	32
2.3.5 Evolutionary Drivers of Neural Modularity	38
2.4 Conclusions	39
3 Experimental Method.....	41
3.1 Research Design	41
3.1.1 Preconditioned versus General Substrates.....	41

3.1.2	Connectivity Constraints	44
3.1.3	Improving HyperNEAT’s Modular Variational Properties with CPPN Disjoints.....	47
3.1.4	Studying the Interaction between Lifetime and Evolutionary Learning with Neuron Nomination	49
3.1.5	Interpretability Assessment	50
3.2	Methodology	51
3.2.1	Neural Network and Training Details	51
3.2.2	HyperNEAT Implementation	52
3.2.3	Fitness Function.....	53
3.2.4	Quantifying Structural Modularity	55
3.2.5	Learning Task Design.....	56
3.2.6	Program Overview.....	57
3.2.7	Computational Speed Considerations.....	58
3.2.8	Statistics and Analysis.....	59
3.3	Experiments.....	60
3.3.1	Experiment 1: Connectivity Constraints	60
3.3.2	Experiment 2: CPPN Disjoints.....	60
3.3.3	Experiment 3: Lifetime and Evolutionary Learning	60
3.3.4	Experiment 4: Interpretability Assessment	60
4	Results.....	61
4.1	Experiment 1	61
4.1.1	Geometrically Separable Task.....	61
4.1.2	Geometrically Inseparable Task	64
4.1.3	Without Supralayer Connections.....	67
4.1.4	Supplementary Neuroimaging Dataset.....	70
4.1.5	Analysis of Experiment 1: Constraint Characterisation	71
4.2	Experiment 2	73
4.3	Experiment 3	73
4.4	Experiment 4	77
5	Conclusion	78
6	References.....	81
Appendix A Neuroevolutionary Encodings.....		97
Appendix B Experimental Data		100

List of Figures

	Page
Figure 1. Diagram of an artificial neuron (alternatively, ‘perceptron’).	5
Figure 2. One-layer perceptron decision boundary; adapted from [21].	7
Figure 3. Diagram of a two-layer perceptron.	8
Figure 4. Intersection and integration of decision boundaries for a two-layer perceptron; adapted from [21].	9
Figure 5. Solving XOR and meshed region problems; adapted from [24].	9
Figure 6. Images depicting the shape hypothesis, where low level patterns are successively integrated into more complex forms; adapted from [74].	14
Figure 7. A fully connected network’s occurrence matrix and node graph.	18
Figure 8. A modular network’s occurrence matrix and node graph.	18
Figure 9. The genotype space, phenotype space and fitness landscape of an evolutionary algorithm; adapted from [137].	22
Figure 10. Example mapping between the genotype space, phenotype space and fitness landscape; adapted from [139].	23
Figure 11. Competing conventions under crossover; reprinted from [170].	30
Figure 12. Structure and operation of CPPNs; reprinted from [160].	37
Figure 13. Abstraction of development by CPPNs; reprinted from [160].	37
Figure 14. Modular control in evolved CPPNs. Reprinted from [160].	37
Figure 15. HyperNEAT substrates; reprinted from [227].	37
Figure 16. Substrate spatial geometry showing layer scaling and hierarchy.	42
Figure 17. Substrate data structure.	43
Figure 18. Substrate configuration with (a) both superlayer and supralayer connections vs (b) only superlayer connections.	43
Figure 19. Differences between images produced (a) only with sum aggregation and (b) with both sum and maximum absolute aggregation.	48
Figure 20. Images produced by evolved CPPNs (with sum aggregation) which use only step activation functions.	48
Figure 21. Images produced by evolved CPPNs (with sum aggregation) which use step and sine activation functions.	48
Figure 22. <i>True</i> class FMs for the fully connected network (a) [94.8%] and modular network (d) [97.4%]; <i>False</i> class FMs for the fully connected	

network (b) [97.0%] and the modular network (e) [97.6%]. CFNs for the fully connected network (c) [97.8%] and the modular network (f) [95.0%].....	51
Figure 23. Q-scores and allocated modules in the (a) fully connected network $Q_H=0$, $Q_{HO}=0$, $Q_{ALL}=0.078$ and (b) modular neural network $Q_H=1$, $Q_{HO}=1$, $Q_{ALL}=0.087$. Input neurons shown in green.....	55
Figure 24. The (a) geometrically separable and (b) inseparable learning tasks. ...	57
Figure 25. Patterns in the geometrically separable and inseparable tasks.	57
Figure 26. HyperNEAT's (a) main and (b) NEAT genotype evaluation loops.	58
Figure 27. Comparison of Q-scores and task performance per connectivity constraint for Experiment 1 (with the geometrically separable task).	61
Figure 28. Convergence curves per connectivity constraint showing the evolution of (a) modularity and (b) task performance in Experiment 1 (with the geometrically separable task).....	62
Figure 29. LCC generates the 'sparse' motif (a) in 6.7% of runs, the 'dense' motif – the most notable shown in (b) – in 36.7% of runs, the 'tower' motif (c) in 30% of runs and the 'messy' motif (d) in 16.7% of runs. GIC generates the 'block' motif (e) in 83.3% of runs which showcases split connectivity (f).....	64
Figure 30. Comparison of Q-scores and task performance per connectivity constraint for Experiment 1 (with the geometrically inseparable task).	65
Figure 31. Convergence curves per connectivity constraint showing the evolution of (a) modularity and (b) task performance in Experiment 1 (with the geometrically inseparable task).....	66
Figure 32. GIC (a) generates 'collapsed' motifs like LCC (b) (c) but without input parcellation. FIC and EIC generate notable 'dense' motifs (d) (e) (f).	67
Figure 33. Comparison of Q-scores and task performance per connectivity constraint for Experiment 1 (with the geometrically inseparable task, without supralayer connections).	68
Figure 34. Convergence curves per connectivity constraint showing the evolution of (a) modularity and (b) task performance in Experiment 1 (with the geometrically separable task, without supralayer connections).....	69
Figure 35. LCC and GCC generate the 'tower' motif (a) in 26.7% and 53.3% of runs respectively and the 'null' motif (b) in 10% and 30% of runs respectively. FIC generates the 'pleated' motif (c) in 16.7% of runs.	70
Figure 36. Convergence curves per connectivity constraint showing the evolution of (a) modularity and (b) task performance in Experiment 1 (with the neuroimaging dataset).	71

Figure 37. The interaction between lifetime and evolutionary learning.	74
Figure 38. Repeated measures ANOVA for experiment 3.	76
Figure 39. Interpretability assessment for Experiment 4.	77
Figure 40. Neuroevolutionary L-systems; adapted from [188].	98
Figure 41. Cellular encodings by Gruau; adapted from [188].	99
Figure 42. Experiment 1, geometrically separable task, LCC's final networks (1-15); The 'dense' motif (marked <i>D</i>) appears in 36.7% of runs. The 'tower' motif (marked <i>T</i>) appears in 30% of runs. The 'messy' motif (marked <i>M</i>) appears in 16.7% of runs. The 'collapsed' motif (marked <i>C</i>) appears in 10% of runs. The 'sparse' motif (marked <i>S</i>) appears in 6.7% of runs.	101
Figure 43. Experiment 1, geometrically separable task, LCC's final networks (16-30); The 'dense' motif (marked <i>D</i>) appears in 36.7% of runs. The 'tower' motif (marked <i>T</i>) appears in 30% of runs. The 'messy' motif (marked <i>M</i>) appears in 16.7% of runs. The 'collapsed' motif (marked <i>C</i>) appears in 10% of runs. The 'sparse' motif (marked <i>S</i>) appears in 6.7% of runs.	102
Figure 44. Experiment 1, geometrically separable task, GCC's final networks (1-15); The 'messy' motif (marked <i>M</i>) appears in 33.3% of runs. The 'tower' motif (marked <i>T</i>) appears in 30%. The 'collapsed' motif (marked <i>C</i>) appears in 23.3%. The 'dense' motif (marked <i>D</i>) in 13.3% of runs.	103
Figure 45. Experiment 1, geometrically separable task, GCC's final networks (16-30); The 'messy' motif (marked <i>M</i>) appears in 33.3% of runs. The 'tower' motif (marked <i>T</i>) appears in 30%. The 'collapsed' motif (marked <i>C</i>) appears in 23.3%. The 'dense' motif (marked <i>D</i>) in 13.3% of runs.	104
Figure 46. Experiment 1, geometrically separable task, FIC's final networks (1-15). The 'dense' motif (marked <i>D</i>) appears in 53.3% of runs. The 'messy' motif (marked <i>M</i>) appears in 40%. The 'half' motif (marked <i>H</i>) appears in 6.7% of runs.	105
Figure 47. Experiment 1, geometrically separable task, FIC's final networks (16-30). The 'dense' motif (marked <i>D</i>) appears in 53.3% of runs. The 'messy' motif (marked <i>M</i>) appears in 40%. The 'half' motif (marked <i>H</i>) appears in 6.7% of runs.	106
Figure 48. Experiment 1, geometrically separable task, EIC's final networks (1-15). The 'dense' motif (marked <i>D</i>) appears in 100% of runs.	107
Figure 49. Experiment 1, geometrically separable task, EIC's final networks (16-30). The 'dense' motif (marked <i>D</i>) appears in 100% of runs.	108

- Figure 50. Experiment 1, geometrically separable task, GIC's final networks (1-15). The 'block' motif (marked *B*) appears in 83.3% of runs. The 'collapsed' motif (marked *C*) appears in 16.7% of runs. 109
- Figure 51. Experiment 1, geometrically separable task, GIC's final networks (16-30). The 'block' motif (marked *B*) appears in 83.3% of runs. The 'collapsed' motif (marked *C*) appears in 16.7% of runs. 110
- Figure 52. Experiment 1, geometrically inseparable task, LCC's final networks (1-15). The 'collapsed' motif (marked *C*) appears in 50% of runs. The 'messy' motif (marked *M*) appears in 36.7% of runs. The 'sparse' motif (marked *S*) appears in 10% of runs. The 'dense' motif (marked *D*) appears in 3.3% of runs. 112
- Figure 53. Experiment 1, geometrically inseparable task, LCC's final networks (1-15). The 'collapsed' motif (marked *C*) appears in 50% of runs. The 'messy' motif (marked *M*) appears in 36.7% of runs. The 'sparse' motif (marked *S*) appears in 10% of runs. The 'dense' motif (marked *D*) appears in 3.3% of runs. 113
- Figure 54. Experiment 1, geometrically inseparable task, GCC's final networks (1-15). The 'collapsed' motif (marked *C*) appears in 76.7% of runs. The 'messy' motif (marked *M*) appears in 20% of runs. The 'tower' motif appears in 3.3% of runs. 114
- Figure 55. Experiment 1, geometrically inseparable task, GCC's final networks (16-30). The 'collapsed' motif (marked *C*) appears in 76.7% of runs. The 'messy' motif (marked *M*) appears in 20% of runs. The 'tower' motif appears in 3.3% of runs. 115
- Figure 56. Experiment 1, geometrically inseparable task, FIC's final networks (1-15). The 'messy' motif (marked *M*) appears in 73.3% of runs. The 'dense' motif (marked *D*) appears in 23.3% of runs. 116
- Figure 57. Experiment 1, geometrically inseparable task, FIC's final networks (16-30). The 'messy' motif (marked *M*) appears in 73.3% of runs. The 'dense' motif (marked *D*) appears in 23.3% of runs. 117
- Figure 58. Experiment 1, geometrically inseparable task, EIC's final networks (1-15). The 'dense' motif (marked *D*) appears in 73.3% of runs. The 'messy' motif (marked *M*) appears in 10% of runs. The 'collapsed' motif (marked *C*) appears in 6.7% of runs. The 'sparse' motif (marked *S*) appears in 3.3% of runs. The 'block' motif (marked *B*) appears in 3.3% of runs. 118
- Figure 59. Experiment 1, geometrically inseparable task, EIC's final networks (16-30). The 'dense' motif (marked *D*) appears in 73.3% of runs. The 'messy' motif (marked *M*) appears in 10% of runs. The 'collapsed' motif (marked *C*) appears in 6.7% of runs. The 'sparse' motif (marked *S*) appears in 3.3% of runs. The 'block' motif (marked *B*) appears in 3.3% of runs. 119

- Figure 60. Experiment 1, geometrically inseparable task, GIC's final networks (1-15). The 'collapsed' motif (marked *C*) appears in 93.3% of runs. The 'block' motif (marked *B*) appears in 6.7% of runs. 120
- Figure 61. Experiment 1, geometrically inseparable task, GIC's final networks (16-30). The 'collapsed' motif (marked *C*) appears in 93.3% of runs. The 'block' motif (marked *B*) appears in 6.7% of runs. 121
- Figure 62. Experiment 1, geometrically inseparable task, without supralayer connections, LCC's final networks (1-15). The 'dense' motif (marked *D*) appears in 46.7% of runs. The 'tower' motif (marked *T*) appears in 26.7% of runs. The 'messy' motif (marked *M*) appears in 16.7% of runs. The 'null' motif (marked *N*) appears in 10% of runs. 123
- Figure 63. Experiment 1, geometrically inseparable task, without supralayer connections, LCC's final networks (16-30). The 'dense' motif (marked *D*) appears in 46.7% of runs. The 'tower' motif (marked *T*) appears in 26.7% of runs. The 'messy' motif (marked *M*) appears in 16.7% of runs. The 'null' motif (marked *N*) appears in 10% of runs. 124
- Figure 64. Experiment 1, geometrically inseparable task, without supralayer connections, GCC's final networks (1-15). The 'tower' motif (marked *T*) appears in 53.3% of runs. The 'null' motif (marked *N*) appears in 30% of runs. The 'dense' motif (marked *D*) appears in 10% of runs. The 'messy' motif (marked *M*) appears in 6.7% of runs. 125
- Figure 65. Experiment 1, geometrically inseparable task, without supralayer connections, GCC's final networks (16-30). The 'tower' motif (marked *T*) appears in 53.3% of runs. The 'null' motif (marked *N*) appears in 30% of runs. The 'dense' motif (marked *D*) appears in 10% of runs. The 'messy' motif (marked *M*) appears in 6.7% of runs. 126
- Figure 66. Experiment 1, geometrically inseparable task, without supralayer connections, FIC's final networks (1-15). The 'dense' motif (marked *D*) appears in 46.7%. The 'tower' motif (marked *T*) appears in 26.7%. The 'pleated' motif (marked *P*) appears in 16.7%. The 'null' motif (marked *N*) appears in 6.7%. 127
- Figure 67. Experiment 1, geometrically inseparable task, without supralayer connections, FIC's final networks (16-30). The 'dense' motif (marked *D*) appears in 46.7%. The 'tower' motif (marked *T*) appears in 26.7%. The 'pleated' motif (marked *P*) appears in 16.7%. The 'null' motif (marked *N*) appears in 6.7%. 128
- Figure 68. Experiment 1, geometrically inseparable task, without supralayer connections, EIC's final networks (1-15). The 'dense' motif (marked *D*) appears in 93.3% of runs. The 'null' motif (marked *N*) appears in 6.7% of runs. 129

Figure 69. Experiment 1, geometrically inseparable task, without supralayer connections, EIC’s final networks (16-30). The ‘dense’ motif (marked *D*) appears in 93.3% of runs. The ‘null’ motif (marked *N*) appears in 6.7% of runs. 130

Figure 70. Experiment 1, geometrically inseparable task, without supralayer connections, GIC’s final networks (1-15). The ‘tower’ motif (marked *T*) appears in 66.7% of runs. The ‘messy’ motif (marked *M*) appears in 23.3% of runs. The ‘dense’ motif (marked *D*) appears in 10% of runs. 131

Figure 71. Experiment 1, geometrically inseparable task, without supralayer connections, GIC’s final networks (16-30). The ‘tower’ motif (marked *T*) appears in 66.7% of runs. The ‘messy’ motif (marked *M*) appears in 23.3% of runs. The ‘dense’ motif (marked *D*) appears in 10% of runs 132

Figure 72. Comparison of Q-scores and task performance per connectivity constraint for Experiment 2 (with the geometrically separable task). 133

Figure 73. Convergence curves per connectivity constraint showing the evolution of (a) modularity and (b) task performance in Experiment 2 (with the geometrically inseparable task). 134

Figure 74. Convergence curves per connectivity constraint showing the evolution of (a) modularity and (b) task performance for experiment with no interaction between lifetime and evolutionary learning. 137

Figure 75. Convergence curves per connectivity constraint showing the evolution of (a) modularity and (b) task performance for experiment with weak interaction between lifetime and evolutionary learning. 138

Figure 76. Convergence curves per connectivity constraint showing the evolution of (a) modularity and (b) task performance for experiment with strong interaction between lifetime and evolutionary learning. 139

Figure 77. Neuron nomination values for the weak explicit interaction. 140

Figure 78. Neuron nomination values for the strong explicit interaction. 140

List of Tables

	Page
Table 1. Data for an example evolutionary problem	24
Table 2. Connectivity constraints	44
Table 3. Comparison of connectivity constraint values for a fully connected neural network and a modular neural network.	47
Table 4. Interaction categories between lifetime and evolutionary learning.....	50
Table 5. Percentile data and Mann-Whitney U-Tests & <i>P</i> -values for experiment 1 (geometrically separable task).	100
Table 6. Percentile data and Mann-Whitney U-Tests & <i>P</i> -values for experiment 1 (geometrically inseparable task).	111
Table 7. Percentile data and Mann-Whitney U-Tests & <i>P</i> -values for experiment 1 (geometrically inseparable task, sans supralayer connections).....	122
Table 8. Percentile data and Mann-Whitney U-Tests & <i>P</i> -values.....	133
Table 9. Experiment 3: no interaction.	135
Table 10. Experiment 3: weak explicit interaction.	135
Table 11. Experiment 3: strong explicit interaction.	136
Table 12. Experiment 4 on the geometrically separable task.	141
Table 13. Experiment 4 on the geometrically inseparable task.	141

Nomenclature

Acronyms

AI	<i>artificial intelligence</i>
CFN	<i>core functional network</i>
CPPN	<i>compositional pattern producing network</i>
FM	<i>functional module</i>
GP	<i>genotype-phenotype (map)</i>
HyperNEAT	<i>Hypercube-based Neuroevolution of Augmenting Topologies</i>
NEAT	<i>Neuroevolution of Augmenting Topologies</i>
REEVE	<i>representation efficiency, expression, variability, embryogeny</i>
SRC	<i>subsets regression on network connectivity</i>
XAI	<i>explainable artificial intelligence</i>
XOR	<i>exclusive-or</i>

Symbols

CpN_{var}	<i>connections per neuron variance</i>
EIC	<i>'equal' input competition</i>
FIC	<i>'free' input competition</i>
GCC	<i>global connection cost</i>
GIC	<i>'greedy' input competition</i>
LCC	<i>local connection cost</i>

1 Introduction

1.1 Background and Motivation

Contemporary advances in speed, trainability, and versatility have made artificial neural networks practically synonymous with the term ‘artificial intelligence’. Neural networks have both benefitted from and driven the development of GPU accelerated computing and unified symbolic mathematics libraries such as Theano and Tensorflow. As such, it has become increasingly feasible to build, train and deploy neural networks at scale within a reasonable timeframe.

Despite such practical and theoretical advances, a significant weakness remains: neural networks are not interpretable – which is to say that it is difficult to understand or examine their internal operations in a meaningful way. The neural network interpretability problem is a consequence of their operating principle and the high dimensional opacity of the intermediate computations which inform their outputs. A growing understanding of previously obscure learning fragilities and adversarial threats has introduced significant uncertainty in the reliability of network performance and the adequacy of current training methods. While not fundamentally crippling, the neural network interpretability problem in context of such uncertainty complicates robust performance characterisation, meaningful human-AI interaction and the ethics of deployment especially when applied to self-driving cars, finance, criminal justice and automated medical diagnosis

Addressing the interpretability problem is an active area of research but one not easily solved. While there are many approaches, the problem can be viewed from the perspective of neural modularity. In system and product design as well as management philosophy, modularity is useful because it structures a set of operations or components to the benefit of versatility, flexibility, stable complexification and evolvability. A modular neural network would be one that is composed of any number of structurally and/or functionally disjoint subnetworks. While neural modularity is advantageous for its own sake, an *interpretable* modular neural network would be one that not only disentangles its internal representations but also groups them within such independent subnetworks in an explanative way. Prior research has had a measure of success in generating neural modularity with neuroevolutionary approaches, but the evolution of neural modularity in relation to its potential contribution to solving the interpretability problem has remained largely unexplored. As such, the aim of this research is not only to investigate the means of evolving neural modularity but also to study how such modularity can benefit the neural network interpretability problem.

1.2 Aim and Objectives

The aim is to identify and/or develop a neuroevolutionary method capable of generating modular neural networks and to investigate how the evolved neural modularity benefits interpretability.

The research hypothesis is that (1) it is possible to develop a neuroevolutionary method that can generate modular neural networks and (2) that modularity can make a positive contribution to the neural network interpretability problem.

Objectives:

1. Select or develop a neuroevolutionary method capable of producing modular neural networks.
2. Select or develop procedures that can identify neural network modules and quantitatively determine the level of neural modularity present relative to non-modular neural networks of equal dimensions.
3. Investigate the evolution of neural modularity (as driven by the selected or developed neuroevolutionary method) across a variety of tasks that are known to be modular and non-modular.
4. Investigate how the evolved neural modularity benefits interpretability

1.3 Chapter Overviews

This work is divided into five chapters. With chapter 1 presenting the introduction, aim and objectives, chapter 2 presents the literature study which reviews the following (1): neural network theory with respect to fundamental principles, adversarial examples and the interpretability problem, (2) modularity as a concept and in terms of its potential contribution to solving the neural network interpretability problem and (3) relevant neuroevolutionary theory and the evolutionary drivers of neural modularity. Chapter 3 documents the research design and experimental method used to achieve the aims and objectives. Chapter 4 presents and examines the experimental results. Chapter 5 presents the study's conclusions and recommendations for future research. Following the references, Appendix A contains additional theory and details relevant to the literature review of Chapter 2. Appendix B contains additional experimental data referred to by the analysis of results given in Chapter 4.

2 Literature Review

The purpose of this chapter is to review and analyse relevant literature in support of identifying/developing the necessary neuroevolutionary method and means of driving the evolution of neural modularity. Section 2.1 reviews and examines neural networks' background and fundamental principles in relation to adversarial threats and interpretability. Section 2.2 examines modularity as a concept and substantiates its contribution to solving the neural network interpretability problem. Section 2.3 reviews and motivates neuroevolution as a method capable of generating modular neural networks in addition to examining extant encoding techniques and the evolutionary drivers of neural modularity. The chapter concludes in section 2.4 with a summary of selected methods and key points of research.

2.1 Neural Networks

2.1.1 Deconstructing AI's Wunderkind

Neural networks are a connectionist, highly parallel category of the field of artificial intelligence (AI) which can be defined as “*the study of the computations that make it possible to perceive, reason, and act*” [1], “*the capability of a system to adapt its behaviour to meet its goals in a range of environments*” [2] or “*the ability to generate maximally successful behaviour given the available information and computational resources*” [3]. The commonality between the many definitions of AI suggests the motive of capturing the properties of intelligence (as recognised in human beings) within an actionable digital format. A definition that succinctly captures this idea is the following statement of intent by Fogel [2]:

“Artificial intelligence should not seek to merely solve problems, but should rather seek to solve the problem of how to solve problems.”

As far as definitions go, it is quite useful; both specific enough to guide a research perspective as well as recursively vague enough to avoid serious scrutiny. Crucially, the definition is also free of the pervasive tendency to anthropomorphise the technology that underpins modern AI. This tendency seems to be the result of a combination of two factors: a fundamental human inclination to look for faces in clouds and the proverbial ‘starting conditions’ of the field itself. Despite McCarthy coining the name [4], the formal origin of the field of artificial intelligence lies with Turing and the simple premise that intelligence is an achievable, deterministic technological problem [5]. While simultaneously considering its mathematical requirements, Turing directly pondered the question of human-equivalent AI in his eponymous test. In that moment and as a result of his subsequent AI forecasts for the year 2000 [6], Turing set a precedent for speculative predictions and hype.

Since then, a certain sci-fi enthusiasm has been the constant companion of formal AI research. Hype culture seems to be an unavoidable temptation, but this tendency to entertain future possibilities has the rather uncomfortable side effect of skewing

the conversation to the fantastical [7], [8]. This is not a peripheral factor; hype culture directly shapes commercial expectations, the availability of research funding [9]–[11] as well as the avenues of research that are pursued and taken seriously [2]. As a result, AI as a field has proven prone to winters [12]–[14]. Coupled with popular science writings and somewhat unfounded warnings of industry disruption [15], [16], hype culture reveals itself to be a blanket for a volume of misconceptions about the present reality and progress of AI.

No stranger to the hype phenomenon, modern neural networks (denoting all deep learning variants) are seen as synonymous with current references to AI [17]. Historically, they were a largely dusty subset of AI until Hinton et al. [18] debuted the AlexNet’s surprising success in image classification. Since then, neural networks have become AI’s wunderkind, quickly adapted and scaled to tackle new problems. In part due to their titular similarity to their biological inspiration, neural networks have been the subject of intense interest as potentially the first general solution to the problem of machine learning. Unfortunately, since the boom of 2006, cracks have appeared in the once sure image of that future to such an extent that Hinton himself has advocated to “*scrap backpropagation and start over*” [19].

2.1.2 Fundamental Principles

In technical terms, an artificial neural network can be thought of as a hierarchical combination of nonlinear class separators which are conceptually similar to biological neurons. Individually, the class separators are a type of binary classifier composed of a linear discriminant function that outputs to a nonlinear activation function. The computational encapsulation of this set of operations as a discrete entity can be referred to as any one of the following: *perceptron*, *node*, *unit*, *cell*, *neuode* or *artificial neuron*. The term *perceptron* is arguably the most historically correct but has been superseded by *artificial neuron* (interchangeably contracted to *neuron*) as the dominant modern term of use (although comparisons to biological neurons are weak at best). The term *neural network* (or equivalently, *multilayer perceptron*) refers to any group of interconnected neurons that are arranged in more than one layer. (The naming conventions used here are those presented in Beale and Fiesler’s *Handbook of Neural Computation* [20].)

As directed computational graphs, neural networks are most commonly classified according to the specific attributes of their *topology* – a term which describes the way a particular network’s neurons are arranged and connected. *Topology* is often used interchangeably with *structure* and *architecture*. Beale and Fiesler use topology to denote both the *neural framework* (the positions of neurons relative to one another) and the *neural connectivity* (the connections between neurons) of a given neural network. ‘Neural framework’ and ‘neural connectivity’ are distinct but overlapping definitions since it is often (but not always) possible to infer the one from the other. Beale and Fiesler’s definition is extended by taking *neural architecture* as a blanket term that simultaneously refers to a network’s topology as well as its *synaptic weights* (weight and bias values). The term *layer* reflects the neuron hierarchy within a neural network. Layers are defined according to their

function and order within the hierarchy as well as the number of neurons within them. Typically indexed starting at one, the *input layer* receives external data and feeds into *hidden layer(s)* that are only exposed to one another and the *output layer* (which presents the outcome of the network’s internal computations).

Usually represented with a binary matrix, neural connectivity can include *interlayer connections* (between adjacent layers), *intralayer connections* (between neurons of the same layer), *self/recurrent connections* (of a neuron to itself) and *supralayer connections* (connections that jump one or more layers). Each connection has an associated synaptic strength (its *weight*) which can be positive (*excitatory*), negative (*inhibitory*) or zero indicating a dead connection. The connectivity of an individual neuron can be described by its *in-degree* (total incoming connections), *out-degree* (total outgoing connections) and *connectivity density* (the ratio of active connections versus total possible connections). The synaptic weights of a neuron are collectively referred to as a filter or kernel (especially in convolutional nets). Neural connectivity can be *symmetric* or *asymmetric* indicating the direction of information flow. For example, neurons in feedforward networks only receive input from neurons in layers below their own.

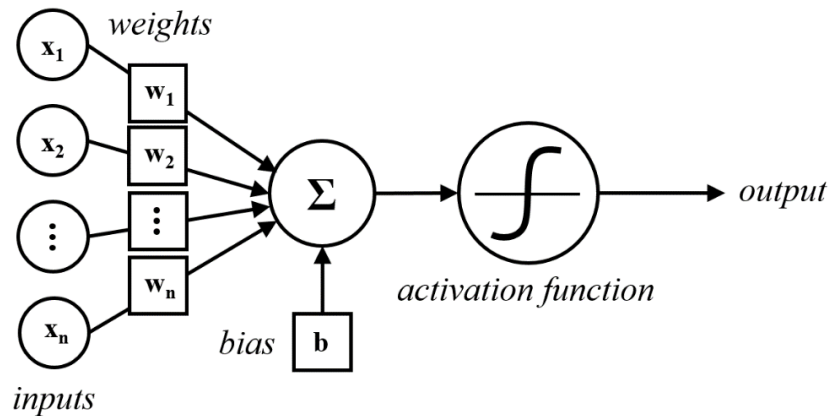


Figure 1. Diagram of an artificial neuron (alternatively, ‘perceptron’).

The ancestor of the neural network is also its fundamental unit: the perceptron. The following explanation is taken from Theodoridis and Koutroumbas’s *Pattern Recognition* [21]. A perceptron maps from an input feature space to an output feature space on the basis of an internal class prediction. As depicted in Figure 1, a perceptron consists of two parts: a linear discriminant function and a nonlinear activation function. The linear discriminant function $g(x)$ in equation 2.1 takes the weighted sum of n input variables (with $w = [w_1, w_2, w_3, \dots, w_n]$ and $x = [x_1, x_2, x_3, \dots, x_n]$) plus a bias (b).

$$g(x) = w^T x + b \quad (2.1)$$

According to the value of its weights and bias, the linear discriminant function describes a hypersurface of dimension $n - 1$ where $g(x) = 0$. This hypersurface is a decision boundary which divides the input feature space into two regions where the value of $g(x)$ is either positive or negative. A point in the input feature space can then be mapped onto a specific part of the output feature space by the activation function depending on whether it occurs in the positive or negative region. In equation 2.2, a step function is used, and therefore the output feature space corresponds to one of the two classes (1 or 0) defined by the step function.

$$f(g(x)) = \begin{cases} 1 & g(x) > 0 \\ 0 & g(x) < 0 \end{cases}. \quad (2.2)$$

In combination, a perceptron is the composed function:

$$p(x) = f(g(x)). \quad (2.3)$$

The mapping of $g(x) \rightarrow f(x)$ is the perceptron's crucial innovation. Mathematically speaking, it allows the perceptron to transform its input by mapping from one feature space to another. While facilitated by the activation function, the output value is determined by the weight and bias parameters. Therefore, a perceptron can be tuned until its input-output mapping corresponds to the desired classes of an arbitrary classification task. In other words, a perceptron 'learns' by iteratively adjusting its weights and bias values in response to a given input and output. Typically implemented as a form of gradient descent, this tuning process continues until the perceptron becomes a functional representation of a dataset [22].

More conceptually, the mapping of $g(x) \rightarrow f(x)$ represents the critical junction where the value of an internal computation can be mapped onto and acted on (by higher layers) as an abstract concept. In other words, a perceptron that is sensitive to a particular input pattern in x can map it onto a binary feature space which represents the x pattern as either class 1 or 0. In the new feature space, x is no longer a numeric pattern but the abstract concept of 1 or 0 – whatever either may represent. Once there, x can be integrated with other abstract concepts and mapped into even higher feature spaces to form ever more complex internal representations. What these internal abstractions represent is purely a consequence of the neuron's parameters and therefore not necessarily interpretable. The properties of the new feature spaces are dependent on the choice of activation function and can be modified linear spaces (by choosing any variant of rectified linear functions) or probability spaces (by selecting sigmoid or softmax functions). To accommodate training algorithms based on gradient descent, the chosen activation function should be differentiable (or sub-differentiable) and have a non-constant gradient. (Step functions are therefore typically not used in practice but remain explanatory.)

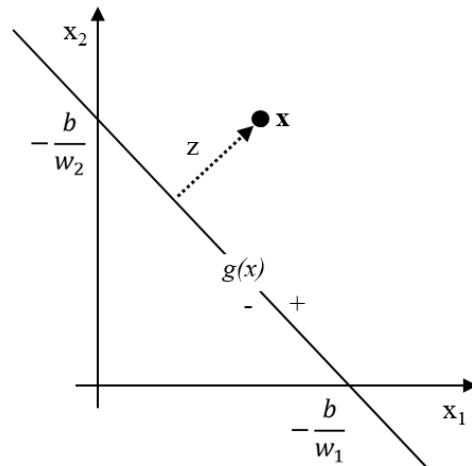


Figure 2. One-layer perceptron decision boundary; adapted from [21].

By selecting $x = [x_1, x_2]$ as the input, it is possible to visualise a perceptron's decision boundary as a line in two-dimensional space. As shown in Figure 2, this line divides the input feature space into two half planes depending on the sign of $g(x)$. The activation function $f(x)$ then maps all points in these half plane regions onto the binary feature space of the step function:

$$x = [x_1, x_2] \rightarrow \bar{y} = [0, 1]. \quad (2.4)$$

Equation 2.5 is the Euclidean distance between input x and the decision boundary:

$$z = \frac{|g(x)|}{\sqrt{w_1^2 + w_2^2}}. \quad (2.5)$$

Crucially, z is not relevant to the classification process. The linear discriminant function makes a classification by mapping a particular region of its input space to a particular class. Thus, only the sign of $g(x)$ dictates which class x belongs to. This benefits generalisation because as long as x remains on one side of the decision boundary, its precise position does not matter. In formal terms, an n -dimensional single layer perceptron divides its input feature space into two half spaces which correspond the classes of the classification problem it has been trained to solve.

A single perceptron is sufficient for linear problems – i.e. where the different classes of a problem can be distinguished by a single hyperplane. Also depicted in Figure 5 (pg. 9), nonlinear problems such as exclusive-or (XOR) or meshed regions are not linearly separable by definition, and as a result a single layer perceptron is not capable of determining the decision boundary that distinguishes class A from B. However, a two-layer perceptron network (shown in Figure 3) solves the nonlinear XOR problem by transforming the nonseparable input feature space into a separable intermediate feature space which the second perceptron layer can act on.

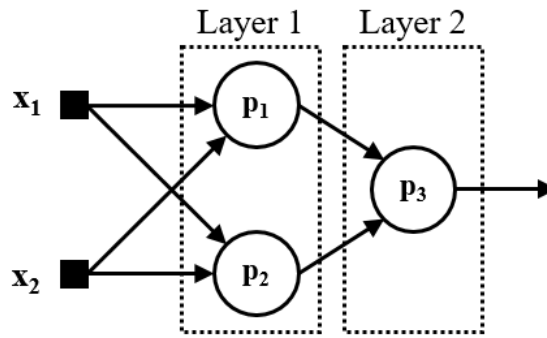


Figure 3. Diagram of a two-layer perceptron.

The equations that define the first layer are:

$$g_1(x) = w_1^T x + b_1, \quad (2.6)$$

$$p_1(x) = f(g_1(x)) = \begin{cases} 1 & g_1(x) > 0 \\ 0 & g_1(x) < 0 \end{cases}, \quad (2.7)$$

$$g_2(x) = w_2^T x + b_2, \quad (2.8)$$

$$p_2(x) = f(g_2(x)) = \begin{cases} 1 & g_2(x) > 0 \\ 0 & g_2(x) < 0 \end{cases}. \quad (2.9)$$

The output of the first layer is therefore:

$$p = [p_1(x), p_2(x)]. \quad (2.10)$$

The second layer equations (equation 2.12) are defined in terms of the intermediate representation mapped by equation 2.11:

$$g_3(p) = w_3^T p + b_3 \quad (2.11)$$

$$p_3(p) = f(g_3(p)) = \begin{cases} 1 & g_3(p) > 0 \\ 0 & g_3(p) < 0 \end{cases} \quad (2.12)$$

Represented as a two-dimensional problem, the decision boundaries defined by equations 2.6, 2.8 and 2.11 are visualised in Figure 4 and Figure 5. While the overall problem is nonlinear, the first layer is able to decompose it into a linearly separable subproblem by combining two decision boundaries to form polyhedral regions that separate the classes. Since the second layer receives the intermediate feature space, it solves a linear form of the decomposed nonlinear problem.

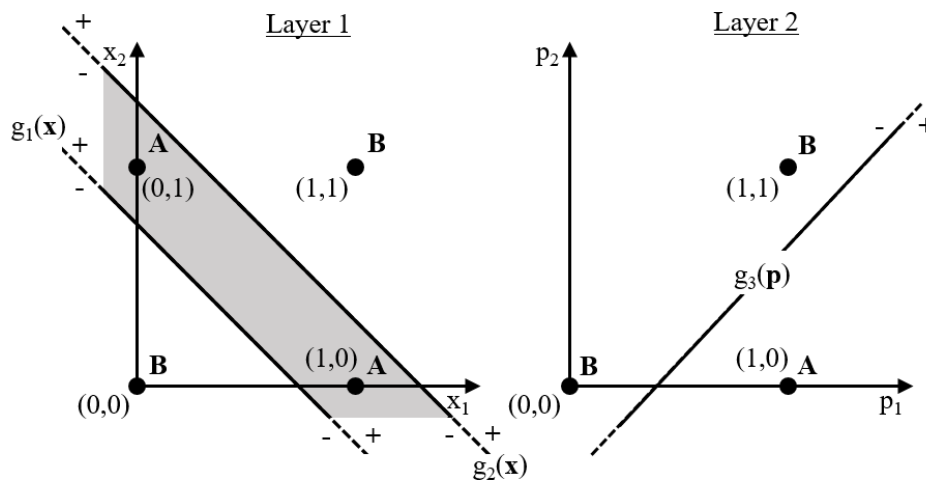


Figure 4. Intersection and integration of decision boundaries for a two-layer perceptron; adapted from [21].

However, two-layer perceptrons are not guaranteed to be able to solve more complicated nonlinear problems such as the meshed regions problem since “a two-layer perceptron can separate classes each consisting of unions of polyhedral regions but not any union of such regions” [21]. A three-layer network allows classification based on any union of polyhedral regions whose complexity is limited only by the number of perceptrons in each layer [23]. Summarised in Figure 5, this cumulative stacking of hyperplanes into polyhedral regions and into unions of regions is the unique operating principle of neural networks that allows them to solve complex, nonlinear problems. The algorithmic elegance of the fact that such solutions are discoverable by iterative methods such as backpropagation is what makes neural networks such powerful and versatile machine learning models.

	Network Structure	Regions	XOR	Meshed Regions
Single layer		Half plane		
Two layer		Convex open or convex closed plane		
Three layer		Arbitrary (limited by # of neurons)		

Figure 5. Solving XOR and meshed region problems; adapted from [24].

Convolutional models operate on the same principles but aggregate their inputs through any number of convolution and pooling steps. Convolution increases computational speed by reducing the number of necessary trainable parameters while also introducing translational invariance. Apart from autoencoder models, the outputs of typical convolutional networks are still computed by one or more perceptron layers. Regardless of such differences in input aggregation, the operations of convolutional networks are equivalent to and can be approximated by a sufficiently expressive multi-layer perceptron network [14].

Despite their computational elegance, there are several important caveats which bear mentioning. Firstly, neural networks are strictly continuous; if the learning task is discontinuous, the closest continuous approximation is found. Secondly, while a sufficiently expressive neural network is capable of approximating any continuous function, the ability to learn the required weight and bias parameters depends on the learning algorithm and is not a given. Thirdly, neural networks are strictly correlation machines (i.e. they can interpolate but not extrapolate) because the learnt approximation exists only within the distribution of the training data. Learning the sine function is a good example. If a feedforward neural network is trained on the range $-5, +5$, it will only be accurate within that range. To get a truly general solution, it would need to be trained over the range $-\infty, +\infty$. Yet even within the training distribution, stable local generalisation requires an assumption of smoothness (i.e. that small input perturbations do not disrupt a network's classification ability) that does not necessarily hold [25]. Finally, precisely what neural networks rely on to make their classifications remains unclear; recent work indicates that the idea of neural networks classifying images (or any other data) by integrating successively more complex shapes into the final whole (the shape hypothesis [14, p. 6], [26, p. 9]) is unfounded; instead neural networks appear to be overly reliant on texture [27] and statistical surface regularities [28] with decision strategies that are not qualitatively different from earlier bag-of-feature classifiers [29]. Such results stand in stark contrast to the hype narrative noted in section 2.1.1 surrounding these machine learning models.

2.1.3 Adversarial Examples

Currently, one of the dominant problems facing general neural network research and interpretability research is the phenomenon of adversarial examples which can be defined as “*input artefacts that are created from natural data by adding adversarial distortions*” [30] which exist anywhere on a continuum between robust (i.e. human interpretable distortions) and non-robust (distortions only apparent to the network models) [31]. Creating adversarial examples is typically an optimisation problem in which the attacker has partial/full knowledge of the target network model (white-box attack) or no knowledge (black-box attack) [32]. White-box attacks can directly co-opt the gradients and responses of a network to perform any variety of direction sensitivity estimation (such as L-BFGS [25], Fast Gradient Sign Method [33], One Step Target Class Method, Basic Iterative Method, Iterative Least-Likely Class Method [34], Jacobian Based Saliency Map [35], One Pixel

Attack [36], DeepFool [37] and Houdini [38]). Black-box attacks (such as Utilising Transferability [39], Model Inversion [40], Model Extraction [41] or evolutionary methods [42]) do not have access to their target network's internal gradients but can also achieve high attack success rates.

The adversarial distortions introduced by Szegedy et al. are invisible to the human eye yet are able to force the Krizhevsky et. al convolutional network (AlexNet) to misclassify images (of speakers, insects, dogs, etc.) as ostriches using their L-BFGS method [25]. Such results are particularly unsettling in the context of similar attacks on medical data [43]. Goodfellow et al. [33] designed the Fast Gradient Sign Method (FGSM) to investigate why adversarial examples generalise across architectures and explained their origin as a consequence of the linearity of the perceptron's discriminant function. Nguyen et al. investigated both gradient and evolutionary techniques, demonstrating that *"it is easy to produce images that are completely unrecognizable to humans, but that state-of-the art DNNs believe to be recognizable objects with 99.99% confidence"* [42]. Extremely minute changes (down to single pixel attacks [36]) can disrupt a network's classification ability.

Adversarial attacks are not brittle or confined to digital settings; physical attacks have revealed that adversarial examples taken in through real-time camera systems are robust under different light conditions and perspectives [44]. Brown et al. were able to produce printable adversarial examples capable of forcing a range of network models (Inceptionv3, Resnet50, Xception, VGG16, and VGG19) to misclassify their input images as toasters [45]. Adversarial examples also extend to three-dimensional physical objects: 3-d printed turtles are misclassified as guns [46], and adversarial glass frames allow attackers to impersonate other people [47]. Malicious attacks aside, it is even possible that the existence of relatively simple, physically realisable attacks (such as lines [48] or stickers[49] on the road or stop signs [50] in the case of self-driving cars) can potentially be mirrored by naturally occurring phenomenon that cause the same predictive weaknesses.

Adversarial attacks are dangerous because they are *"not random, they are not due to overfitting or incomplete model training, they occupy only a comparatively small subspace of the feature landscape, they are robust to random noise, and they have been shown to transfer in many cases from one model to another"* [51]–[53]. Adversarial examples do not prove that neural networks are fundamentally broken or flawed. In fact, Ilyas et al. argue that adversarial are inherent properties of neural networks rather than flukes [31] – a claim shared by Goodfellow et al. [33] and Shafahi et al. [54]. That being said, it remains necessary to quantify the robustness and safety of neural network solutions. While some countermeasures have been developed (such as adversarial training [25], [33], [55], [56], blocking adversarial transferability [57] or defensive distillation [58]), no current method has been proven to be a truly general defence against every adversarial attack [32]. In lieu of this, the current reality is that neural networks cannot be conscientiously applied to problems which involve high risk.

2.1.4 XAI: Towards Meaningful Interpretability

The fact that neural networks are not interpretable is a direct consequence of their operating principles: predictions are generated by many successive nonlinear mappings between feature spaces determined by multi-dimensional learnt decision boundaries. These decision boundaries are not human interpretable. Those depicted in Figure 5 of subsection 2.1.2 can be visualised and understood because they are two-dimensional. If the three-layer network defined by equation 2.12 had even four perceptrons in its first layer, the resultant four-dimensional decision boundary of the second layer is rendered unplotable. Because practical neural networks can have millions of parameters [59], attempting to map and understand the interacting decision boundaries is simply not a feasible strategy. Training algorithms are equally opaque since parameter update rules rely on equivalently high dimensional gradient calculations. As a result, neural networks are considered black box models.

The interpretability problem would not matter at all if it were proven beyond doubt that neural networks consistently form sensible decision boundaries that map patterns which exist in the real world. Unfortunately, this is not the case. The need for explainable AI (XAI) is summarised by Ross and Velez [60]: “*[deep] neural networks have proven remarkably effective at solving many classification problems but have been criticized recently for two major weaknesses: the reasons behind their predictions are uninterpretable, and the predictions themselves can often be fooled by small adversarial perturbations.*” These two weaknesses combine with the learning fragilities documented in subsection 2.1.2 to make a third: there is a lack of (and a need for) meaningful and collaborative human-AI interaction that will allow users “*to understand, appropriately trust, and effectively manage the emerging generation of artificially intelligent partners*” [61].

Safe deployment to tasks with high risk requires a level of collaboration and trust between human and AI models that neural networks cannot deliver yet. Self-driving cars, finance, criminal justice, automated medical diagnosis – such tasks are dominated by issues of safety, ethics, the discovery of scientific understanding, mismatched and/or multi-objective trade-offs. They are defined by a degree of incompleteness in knowledge and problem formulation greater than simple uncertainty that “*can be rigorously quantified and formally reasoned about*” [62]. In such situations, transparent methods are important [62] since there is an inherent “*mismatch between the formal objectives of supervised learning [...] and the real world costs in a deployment setting*” [63]. AI that cannot explain its methods and conclusions cannot participate in discussion and critique since any scepticism is vulnerable to an endless circle of unknowns that is unrelated to the problem at hand.

While it is relatively simple to identify the need and justification for XAI, creating a formal description of the idea seems so elusive and perspective-dependent that attempts often stall at the level of the quasi-scientific [63]. Admitting the lack of consensus about its use and meaning, Lipton sketches a general description of XAI as models that emphasise trust, causality, transferability, informativeness as well as fair and ethical decision making [63]. Such outcomes are the result of some intrinsic

trade-offs between model transparency (by simulatability, decomposability and learning algorithm transparency) and post-hoc explanations (through text, visualisations, local interactions or by example) [63].

By contemplating such concepts with specific reference to neural networks, Gilpin et al [64] provide a framework within which to reason about and assess strategies for tackling the XAI problem. Notably, they argue that it is important to take “interpretability” and “explainability” as independent concepts. While declining to weigh in on the philosophical definition of an explanation, Gilpin et al. note that explanations are assessed in terms of the trade-offs between their interpretability and completeness. If completeness is the ‘complete’ description of each state and action of a system, interpretability is the level of abstraction that is necessary to convert that mass of details into a form that a human being can interpret. Therefore, explainability is the optimal tradeoff between interpretability and completeness relative to the task at hand. The needs of the observer will dictate the nature of the cognitive chunks (basic units) of the explanation, the level of compositionality and the types interactions that are understandable [62]. Under this definition, interpretability is a necessary but not sufficient condition for total explainability. Gilpin et al. [64] defines three broad categories of explainability techniques that are relevant to neural networks specifically: internal processing explanations, data representation based explanations and explanation-producing systems.

Explainability with reference to internal processing aims to increase interpretability by circumventing the inherent complexity of a neural network. One approach is to recreate the functionality of a trained neural network in a different mathematical model that is inherently more understandable (such techniques include linear [65] and decision tree [66], [67] proxy models and automatic rule extraction [68]). Another approach is salience mapping (particularly applicable to visual classifiers) which aims to identify and highlight the most important causal relationships between the inputs and corresponding outputs of neural networks [69]. A clear example of explainability benefitting design, Zeiler and Fergus’s work on salience mapping directly informed the structure of their neural architecture, allowing them to beat the 2013 state-of-the-art on the Caltech-101 and Caltech-256 datasets [69].

Explainability via an analysis of data representations is interested in understanding the internal data structures that neural networks create and manipulate in order to make a prediction. The idea is to understand the behaviour of neural networks by studying the structure and functionality of their constituent parts: individual perceptron units, layers and representation vectors. This is commonly done by discovering (via backpropagation [70], sampling [71] or generative networks [72]) and visualising a neuron or layer’s preferred inputs – in other words, inputs that cause a high activation. However, interpretation via preferred inputs is not a general solution, since it almost exclusively favours image data.

In addition, recent work has challenged the validity of using preferred inputs as cognitive chunks in the interpretation process. Interpreting internal representations through preferred inputs relies on the assumption that the shape hypothesis holds.

Depicted in Figure 6 below, the shape hypothesis assumes that a neural network builds increasing complex concepts with every layer and primarily relies on such high level abstractions to make its predictions [14, p. 6], [26, p. 9]. Nie et al.’s work indicates that this assumption is unfounded; in contrast to saliency mapping, the shapes that successive layers are purportedly sensitive to are instead partial image reconstructions which do not reflect the network’s decision making [73]. The shape hypothesis is further undermined by work indicating that neural networks are overly reliant on texture [27] and statistical surface regularities [28] rather than high level internal representations. Furthermore, neural network decision making is not qualitatively different from earlier (and inherently more interpretable) bag-of-feature classifiers [29] regardless of the complexity of the internal representations.



Figure 6. Images depicting the shape hypothesis, where low level patterns are successively integrated into more complex forms; adapted from [74].

While the two previous categories are interested in translating internal processes or representations into a human-interpretable format, explanation-producing systems shift the burden of interpretability to the neural network itself. Here the objective is to design architectures and training algorithms that produce explanations of their own behaviour directly. Attention networks [75]–[77] learn functions that map *which* and *which parts* of previous inputs (or internal representations) are incorporated into current computations; this intrinsic attention mechanism is explanative because it shows how previous inputs (and components of previous inputs) inform current outputs [78]. Such attention maps can either form indirectly, or can be trained to be explicitly explanative [79].

Learning disentangled representations is another explanation producing system that works in tandem with an analysis of internal data structures; here, the goal is to discover and organise representations which “*describe meaningful and independent factors of variation*” [64] by means of variational autoencoders [80], generative-adversarial networks that maximise representation-observation mutual information [81], similarly adapted convolutional networks [82] and capsule networks [83]. In contrast to attention networks and disentangled representations, explanation generation networks are more explicit; these methods justify their decision making post-hoc with either visual cues and/or text explanations [84]–[86].

2.2 Interpretability through Modularity

2.2.1 Modularity as a Concept

Definitions of modularity appears to be rather context dependent [87] and have many overlapping formulations that have been explored in “*fields as diverse as industrial engineering, construction, robotics, computer science, mathematics, biology, medicine, cognitive science, psychology and art*” [88]. Within these fields, modularity is studied according to two basic paradigms: as an outcome of a system/product’s design process or as a property of a network described by graph theory. The first scenario is often satisfied by context dependant qualitative measures while the second typically only admits exact measures which describe both the presence and relative level of modularity in a given network. Miraglia documents several definitions of modularity according to their use within the context of management and system architecture research [88]. According to these definitions, modularity can be any of the following:

1. Modules as functionally specialised subunits; in other words, “*a one-to-one mapping between [system] components and [system] functions*” [88]–[93].
2. Modules as functionally specific inputs; in other words, “*[a] module is a system’s component on which a specific function can be performed independently of other modules*” [88], [94]. With multiple input hierarchies, this becomes the modularity of *process* [95].
3. Modules as structurally separable subunits; in other words, “*[a] module is made of components that are tightly connected among themselves and loosely connected with the components of other modules*” [88], [96], [97]
4. Modules as a combination of any or all of the previous definitions [98].

There is a dualistic interest in integrality – often defined as the converse of modularity. Integrality describes systems whose subunits are both functionally and structurally integrated in a way that promotes “*coordination and unity of effort among various parts of a system*” (organisational integration) through “*components being specific to one another*” (synergistic specificity) [97], [99]–[103].

Many systems of interest can be described by graph theory which contains a formal measure of modularity. In graph theory, system components are represented by nodes, and the interactions between them are represented by edges which describe one-way (directed), two-way (undirected), binary (on or off) or weighted (fractional values) connections. In this framework, modularity is a specific pattern of connectivity where individual modules are “*communities of nodes [with] greater numbers of mutual connections within each community and fewer connections between them*” [104]. Therefore, by representing a system according to its structural, functional and causal connectivity, it is possible to investigate the four concepts of modularity consistently and systematically.

Modularity is useful because it reveals the underlying structure of a set of operations and has been shown to benefit versatility, flexibility, stable complexification and evolvability [105]–[108]. In addition, it is a dedicated and sought-after outcome of system design, product design, software development as well as any management system [88]. The same benefits accrue to living organisms. The phenotypes of biological systems are overwhelmingly modular and have strong indications of modularity in their genetic control systems [109] and neural pathways [110]. Abstracting cellular machinery, the transition between unicellular and multicellular lifeforms hinged on the evolvability of modular organisations and allowed life to occupy more complicated niches and to respond to more complex stimuli [111].

2.2.2 Interpretability and Neural Modularity

The predecessors of modern neural networks stem from attempts to model the biological brain [112], [113] and replicate a crucial property: associative memory. A good way to explain associative memory is to contrast it with its opposite – random access memory – which relies on knowing the location of a particular piece of information. The actual data stored at the target location in memory is secondary to the retrieval process. Conversely, associative memory allows memories to be retrieved from the full or partial content of stimulus [114]. The basis of associative memory is the distributed representation of information in the brain in concert with the ability to link incoming sensory patterns to previously experienced patterns.

Mathematically, random access memory can be expressed as the set of messages (s) taken from an alphabet (Σ) [115]:

$$M_l(\Sigma) = (s_1, \dots, s_n): n \in \mathbb{N}, s_1, \dots, s_n \in \Sigma, \quad (2.13)$$

whereas associative memory takes the form

$$M_m(P, A) = \{m : Q \rightarrow A, Q \subseteq P, |Q| < |\mathbb{N}| \}. \quad (2.14)$$

In equation 2.14, an associative memory (M_m) is defined as the mapping (m) between a question (Q , a finite subset of all possible questions P) and an answer (A). In other words, some input/question (Q) is mapped onto an output/answer (A) and retrieved by the recall process (i.e. the mapping $Q \rightarrow A$) rather than a location in the set (s_1, \dots, s_n) defined in equation 2.13. Reviewing equations 2.7, 2.9 and 2.12, retrieval by mapping between sets is precisely how neural networks operate. An individual neuron creates a nonlinear mapping between its input feature space and an output feature space. Because the z independence (equation 2.5) of their mappings allows them to generalise, the response of any single neuron is not locked to an exact input. Therefore, as different inputs flow into a neural network, individual mappings are expressed as unique, distributed patterns of activations that collectively represent the memory of those specific inputs. In other words, neural network prediction is not defined by location but rather the mapping process.

As such, neural networks implement associative learning because their “*structures of interactions can be made to correspond to structures of knowledge*” [23]. A memory (i.e. a classification) is the process of recalling the correct structure of interactions (a pattern of activations) in which “*each active unit might stand for a microfeature and the connection strengths might represent microinferences between these microfeatures*” [23]. That being said, classification through associative recall can be disrupted or limited because of the inherent disadvantages of storing information in a distributed format. In this context, the most relevant issue is the problem of limited concept communication. If a concept is recalled by a pattern of activity described by the entire network, then only one concept can be present at any one time. For example, if a neural network that classifies dogs and cats represents them as patterns of activity distributed across all of its neurons, it would not be able to interact with both representations at the same time. As a consequence, Khanna notes that “*for simultaneous communication between a variety of concepts, they must all be represented in disjoint regions*” [23], a statement supported by further work on the topic [106], [116]–[119].

A modular neural network is precisely one which is composed of a number of disjoint regions. More specifically, these disjoint regions would be sparsely interconnected subnetwork modules demarcated by more intra-module than inter-module connections. As computational graphs, subnetworks that are *structurally* disjoint are necessarily *functionally* disjoint as well. In other words, a modular neural network would unavoidably isolate different activation patterns to different subnetworks as a direct consequence of limited inter-module communication. As a result, a modular neural network would be able to compute independent representations in parallel and flexibly integrate them upstream.

Neural modules as structurally isolated, functionally disjoint subnetworks also mirror the principles of problem decomposition. In a paper describing formal decomposition techniques, Himmelblau states that the “*most effective form of decomposition is to form disjoint subsystems, that is to form subsets of relations that do not contain any common variables so that each subset can be treated independently*” [120]. The concept of modularity as a problem decomposition effect is echoed by Lipson [108] and Jacobs et al. [121]. It is relatively straightforward to demonstrate that neural modularity reflects the decomposition of a large system of relations. Information flow in such a system is mapped by an occurrence matrix whose rows denote system equations and whose columns denote system variables. Its entries s_{ij} can be 1 or 0 as defined by equation 2.15. A hidden neuron in a feedforward multilayer network is both a variable and a function (of other neurons but not itself).

$$s_{ij} = \begin{cases} 1, & \text{if variable } j \text{ appears in equation } i \\ 0, & \text{otherwise} \end{cases} \quad (2.15)$$

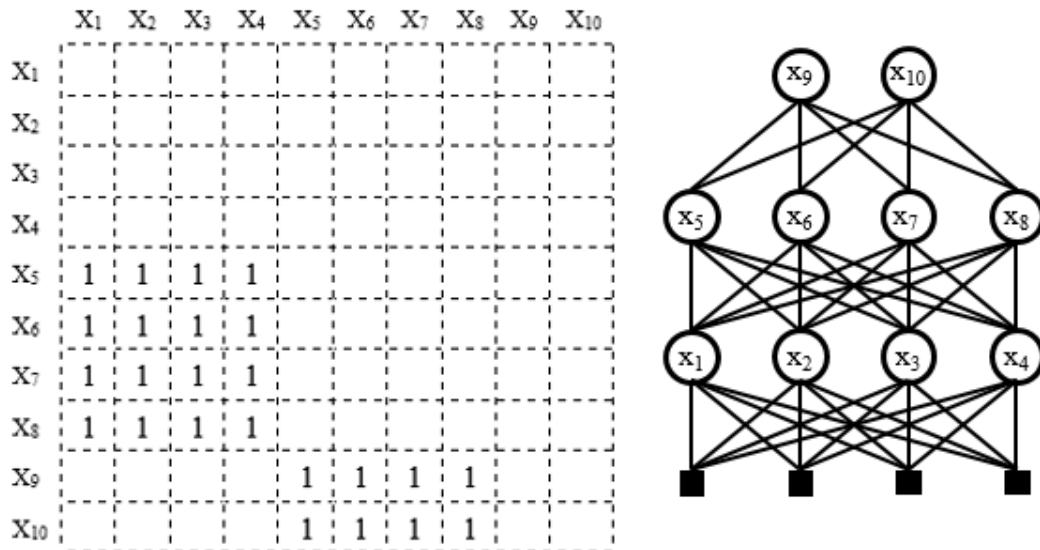


Figure 7. A fully connected network’s occurrence matrix and node graph.

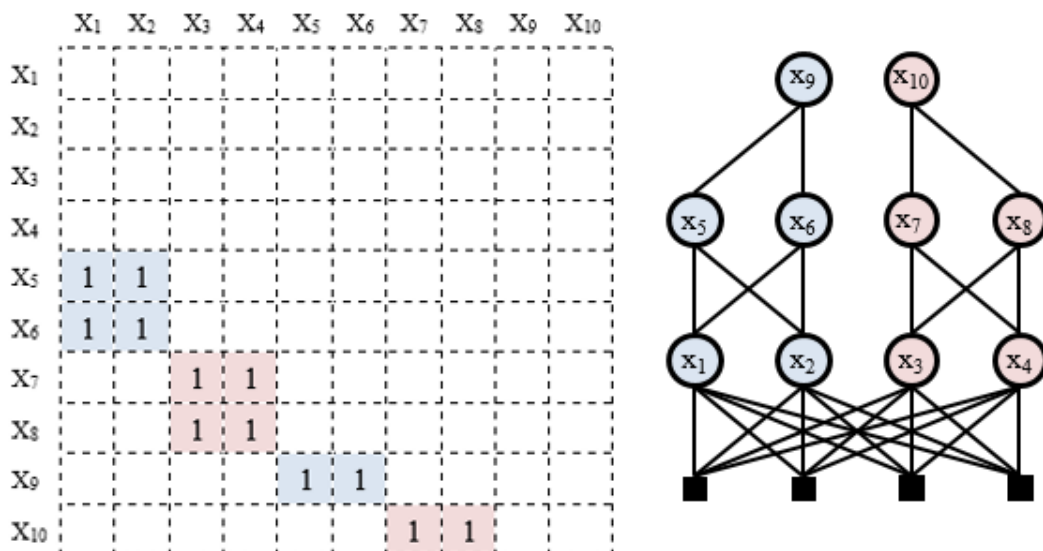


Figure 8. A modular network’s occurrence matrix and node graph.

Depicted in Figure 7, the occurrence matrix of a fully connected network has off-diagonal blocks which are only disjoint with respect to layer. The corresponding set of the system equations (equation 2.16) clearly indicate that such a network is a single monolithic function. In other words, the representations within a fully connected multilayer network are not grouped or exclusive to any particular output.

$$\text{output set} = \begin{bmatrix} x_9(x_5(x_1, x_2, x_3, x_4), x_6(x_1, x_2, x_3, x_4)), \\ x_{10}(x_7(x_1, x_2, x_3, x_4), x_8(x_1, x_2, x_3, x_4)) \end{bmatrix} \quad (2.16)$$

In contrast, the modular configuration depicted in Figure 8 contains two independent subnetworks. Again, the off-diagonal blocks are disjoint subsystems, yet here they correspond to a specific subnetwork. As shown by equation 2.17, these subnetworks compute independent functions since there is little to no overlap between their constituent representations. There is also an aspect of dimensionality reduction. To understand the function of the left subnetwork, it is only necessary to examine the interactions of $[x_3, x_4, x_7, x_8]$ rather than $[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8]$.

$$\text{output set} = \begin{bmatrix} x_9(x_5(x_1, x_2), x_6(x_1, x_2)) \\ x_{10}(x_7(x_3, x_4), x_8(x_3, x_4)) \end{bmatrix} \quad (2.17)$$

If the modular network of Figure 8 were applied to classifying cats and dogs, the modules may share low-level representations such as fur texture and body shape but would integrate them separately within the different sub-tasks of detecting a cat or a dog. In this way, a modular neural network that isolates independent activation patterns to independent subnetworks would not only untangle its representations but also localise sub-task functionality. This can potentially benefit interpretability, because where representations compute single functions, a sub-task would be an explanative grouping of such representations. In other words, “*modular networks have the advantage of a clear task division, where it is always clear which module is responsible for each action*” [122]. The complexity (i.e. the interactions and number of representations it contains) and purpose of a sub-task specific module would be relative to the problem being solved.

That being said, modularity as measured by graph theory is a continuous property. While structural disjoints create functional disjoints, functionally *disjoint* modules are not automatically equivalent to functionally *specialised* modules. This is because inter-module communication can be ‘messy’. The sheer expressivity of neural networks essentially guarantees a complete continuum of inter-module configurations that are trainable and computationally feasible but co-dependent and functionally overlapping. Unfortunately, only fully independent modules with ‘clean’ divisions will reveal the problem decomposition structure in an interpretable way. Therefore, modularity’s benefit as an explanation of internal processing or internal data structures will depend heavily on both internal module structure and inter-module communication. Furthermore, while modularity does act to disentangle representations, its effect is more indirect compared to the explanation producing systems [80]–[83] discussed in subsection 2.1.4 which disentangle representations explicitly. Overall, this suggests that modularity’s benefit to interpretability may be ambiguous and potentially quite discontinuous.

In conclusion, *extant* modularity does not necessarily equal *interpretable* modularity. To be useful with respect to interpretability, neural modularity must

take on a particular form: functionally specialised subnetworks that solve distinct subtasks. Formed properly, such a modular network would be able to compute and integrate independent concepts simultaneously in an inherently more interpretable way than an equivalent fully connected network. In this scenario, a subnetwork module becomes a cognitive chunk in the explanation process that can be used to explain and/or attribute specific behaviours, failures or adversarial weaknesses to specific parts of the greater neural network. If they form consistently and reliably, the explanative potential of such cognitive chunks can potentially benefit data analysis as a form of hypothesis generation. In addition, there are also indications that neural modularity helps to combat catastrophic forgetting [119], [123].

2.3 Evolving Modular Neural Networks

2.3.1 Evolutionary Computation

Neural networks are machine learning tools; by definition, they apply to situations where the designer lacks sufficient prior knowledge to manually solve the problem. Given the aim of creating modular neural networks, the designer would not know (or be expected to know) how to construct the modules by hand. If it were possible to manually construct and integrate a set of functionally specialised subnetworks, there would be no reason to use a neural network in the first place. The alternative where different subnetworks are trained separately and then trained to integrate is also insufficient [124] as a general case solution since the designer must still choose what modules to use. The issue is a lack of prior knowledge about the problem.

Apart from experience and rules of thumb, there is also lack of prior knowledge about the design process itself. Designing a neural network is the act of “*searching the surface defined by the optimality level of [a network’s] architecture in the architecture space, which is composed of all possible architectures*” [125, p. 9]. As identified by Miller [126] and listed by Yao [125], this surface is:

- “*infinitely large since the number of possible nodes and connections is unbounded*”
- “*nondifferentiable since changes in the number of nodes or connections is discrete and can have a discontinuous effect on [network] performance*”
- “*complex and noisy since mapping from [neural] architecture to [network] performance after training is indirect, strongly epistatic and dependent on initial conditions*”
- “*deceptive since [networks] with similar architectures may have dramatically different information processing abilities and performances*”
- “*multimodal since [networks] with quite different architectures can have very similar capabilities*”

Yet this lack of both problem and design prior knowledge is not fatal. Instead, it implies that the solution should rely on methods that are capable of building

modular neural networks without relying on human insight [127] – in other words, submitting to the central tenet of Fogel’s definition of AI: self-design by exposure to the problem. Yao motivates evolutionary computation as a method that is well suited to the problem of searching the space of all possible neural architectures to find those with the desired attributes [125].

Contemporary evolutionary theory – formally known as the Extended Modern Evolutionary Synthesis (the shorthand is Neo-Darwinism) – describes “*the descent, with modification, of different lineages from common ancestors*” [128] and builds on the classical Darwinian model by incorporating genetics, ontogeny and developmental biology. While biological evolutionary theory studies DNA as the mechanism of inheritance with mutation, recombination and gene flow as sources of variation, Darwin’s core postulates do not explicitly depend on DNA as a vector since adaptation is only predicated on the presence of (1) a sexually/asexually reproducing population whose individuals (2) exhibit heritable variation that is (3) acted on by differential fitness selection pressure (4) caused by competition for finite resources. The exact mechanics of the genetic system that allows a preceding generation to pass its characteristics on to the next is irrelevant, and in that sense evolution is model blind [129], [130]. As long as the variation is heritable and is directly/indirectly related to survival fitness, adaptations will occur that make fitter organisms more likely to survive and pass on their traits.

Because evolutionary principles are model blind, the process of adaptation can be harnessed to create digital structures that evolve to maximise their fitness in relation to some task. Typically, this is a process of creating direct analogues or abstracting the principles behind the way that evolution acts on the hierarchy of genes, individuals, species and ecosystems within an environment in response to reproductive fitness. The theory and techniques of harnessing evolution digitally constitute the field of evolutionary computation; it describes a family of iterative, population-based metaheuristic algorithms typically subdivided according to the type of problem being solved. Among others, these include genetic programming [127], evolutionary programming [131], evolutionary strategies [132], genetic algorithms [133] and neuroevolution [134]. Such divisions are largely cosmetic or defined by omission since all evolutionary algorithms rely on the same principles.

Evolutionary algorithms must specify the fitness function, variation operators, reproduction criteria and structure of the artificial genetic system to work [135]. In biology, fitness is environmental feedback that describes how well an organism can solve the various problems of survival; in digital settings, the fitness measures that guide evolution can be arbitrary functions that assess the quality of a given solution. The variation operators (typically *mutation* and *crossover*) and reproduction criteria collectively describe when and how solution candidates reproduce as well as the properties and magnitude of the variation introduced to the offspring. Mutation invokes random changes. Crossover mirrors sexual recombination and allows an exchange of genetic information when creating offspring. Research indicates that crossover is the dominant source of adaptive variation while mutation primarily acts to preserve genetic diversity [136]. The artificial genetic system is a two-way map

between a candidate solution (*phenotype*) of the chosen problem and some descriptive data structure (*genotype*) that the variation operators can act on; this is known as the *genome*, *chromosome(s)*, *genetic characters*, *genotype-phenotype map*, *representation (scheme)* or *encoding*. For clarity, artificial genetic systems are referred to here as ‘genotype-phenotype maps’ (abbreviated to GP maps) when speaking in general and as ‘encodings’ when referring to specific implementations. The term ‘genetic characters’ is used to refer to an encoding’s subunits.

The typical evolutionary algorithm operates across three spaces shown in Figure 9: the genotype space, the phenotype space and the fitness landscape. The function of any evolutionary algorithm is to discover a mapping between the genotype and phenotype space whose properties correspond to a sufficiently optimal part of the fitness landscape. It does this by introducing and selecting for adaptive variation in a population of reproducing candidate solutions. More specifically, it does this by generating a population of candidate solutions, assessing those solutions in relation to the specified fitness function and breeding the next generation (with variation) based on their predecessor’s fitness profile. Since fitter candidates are better able to reproduce, generational adaptation shifts the population towards higher fitness values. As a result, an evolutionary algorithm acts to ultimately generate sufficiently optimal solution(s) by discovering an evolutionary path up the fitness peaks defined by the fitness function.

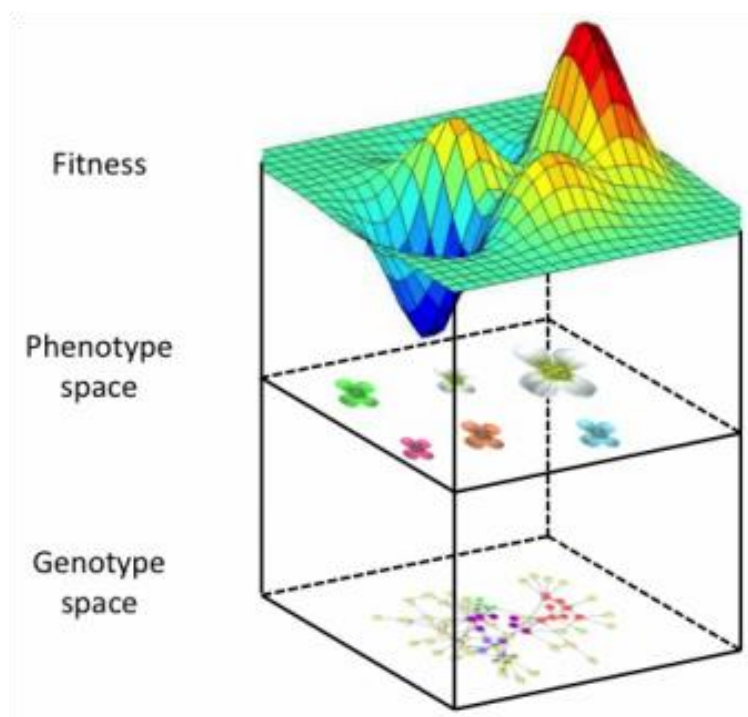


Figure 9. The genotype space, phenotype space and fitness landscape of an evolutionary algorithm; adapted from [137].

The following example is adapted from Goldberg's problem of maximising the single variable function shown in equation 2.18 [138]. In this case, the phenotypes are candidate x inputs to the function $f(x)$.

$$f(x) = \frac{-x^2}{10} + 3x \quad (2.18)$$

Goldberg restricts the phenotype population to the set of integers $[0, \dots, 31]$ and chooses to represent these with a five-bit encoding scheme. In other words, the GP map is a mapping between the binary genotype set $[00000, \dots, 11111]$ and the integer-valued phenotype set $[0, \dots, 31]$. The mapping is finite since all possible variations of the 5-bit genotype space can only result in the integer set $[0, \dots, 31]$. Additionally, it is a linear, one-to-one transformation since binary encodings are operation-preserving under addition, subtraction and multiplication.

Bitwise mutation is chosen with a 0.001 probability and crossover as exchange at a random point. The reproduction criteria is the roulette wheel where a phenotype's chance to reproduce is the fraction of its fitness and the population's total fitness. This acts to favour fitter candidates while also affording less fit candidates a chance to reproduce. Goldberg defines the fitness function as equation 2.18 itself. As shown in Figure 10, a binary genotype space is mapped onto an integer-based phenotype space which in turn maps onto the fitness function defined by $f(x)$; the goal is to encourage changes in the genotype space that move the distribution in the phenotype space closer to the peak at $x = 15$ of the fitness landscape.

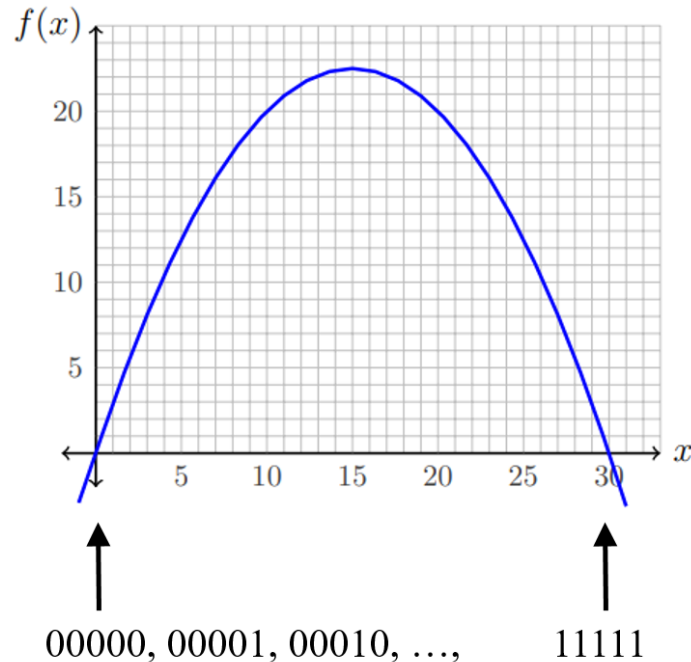


Figure 10. Example mapping between the genotype space, phenotype space and fitness landscape; adapted from [139].

In Table 1 below, the evolutionary process starts with a population of four individuals. The genotypes of the random starting candidates 1 – 4 are mapped onto the phenotype space to give x values of 12, 3, 19 and 7. The fitness of each respective phenotype is the output of $f(x)$, and its fraction versus the total fitness represents its reproductive probability. The first generation has a maximum fitness of 21.6 centred on a mean of 16.7; the phenotype distribution is centred on 10.3.

Also shown in Table 1, the next generation is made by performing crossover on the candidates selected by a spin of the roulette wheel. High probability candidates 1 and 4 are selected for crossover at index 4; this splits their genotypes (01100 and 00111) and produces two offspring: 01111 and 00100. Finally, mutation acts on the new population's genotypes but given its low probability (0.001), no random bit changes are made in this generation. Calculating the fitness of the second generation reveals that both its maximum (22.5) and mean fitness (18.4) values are higher than those of the first generation. This result shows that the population's mean position has shifted from 10.3 to 12.5. In other words, recombination innovation applied to the genotype-phenotype map has shifted the population to a higher position in the fitness landscape defined by $f(x)$.

Table 1. Data for an example evolutionary problem

Generation 1				
ID Number	Genotype	Phenotype	Fitness	Probability
1	01100	12	21.6	0.33
2	00011	3	8.1	0.12
3	10011	19	20.9	0.31
4	00111	7	16.1	0.24
Mean		<u>10.3</u>	<u>16.7</u>	
Max			<u>21.6</u>	
Generation 2				
ID Number	Genotype	Phenotype	Fitness	Probability
1	01111	15	22.5	0.31
2	00100	4	10.4	0.14
3	10100	20	20.0	0.27
4	01011	11	20.9	0.28
Mean		12.5	<u>18.4</u>	
Max			<u>22.5</u>	

2.3.2 Factors of Evolutionary Success

This subsection reviews the formulation of the fitness function and the GP map's capacity for adaptive variation as the dominant drivers of evolutionary success. The evolutionary success of biological organisms is measured by high abundance and wide geographic distribution in both stable and unstable environments [140]; similarly, evolutionary algorithms are considered successful if they can solve a wide variety of problems. If binary encodings under crossover and mutation were universally successful, all current and future design challenges could be considered solved. Unsurprisingly, this is not the case. In fact, such genetic algorithms – while successful in some problems – fail utterly in others [111], [141]–[143].

An evolutionary algorithm is an artificial population whose adaptive success or failure is exclusively dictated by the fitness function. In nature, variation is acted on by the environment which provides rich, implicit feedback continuously across an organism's lifetime. In evolutionary algorithms, variation is acted on by the chosen fitness function, and it alone creates the landscape within which the artificial organisms struggle or flourish. Consequently, useful evolution grinds to a halt if the fitness function is poorly formulated. In less technical terms, a fitness function is a 'blind idiot god', tirelessly optimising a metric which may be irrelevant [144].

Poorly formulated fitness functions will allow evolving populations to ruthlessly exploit boundary conditions [145], [146]. A good example comes from Karl Sims's pioneering work in the evolution of morphology and behaviour of virtual creatures. When fitness was scored as the average ground velocity of their centre of mass, Karl Sims observed that digital evolution produced structures that efficiently moved that centre of mass by simply being tall and falling over [147], [148]. Another example comes from Ofria's work on mutational robustness under a reverse fitness scoring method where candidates that replicated faster than their predecessors are eliminated [149]. As it turned out, the replicators had evolved to mask their success by learning which tasks were used to evaluate performance and halting their replication in order to "*play dead' in front of what amounted to a predator*" [146].

Yet even if the fitness function does govern the correct metric, it may still be blind to other factors such as developmental potential or partial solutions. In the situation where an evolutionarily stagnant solution occupies a local fitness peak, all variation leads downwards to lower fitness values [150]. A fitness function that exclusively prioritises performance will aggressively conserve this solution's structure and allow it to outcompete all candidates that are in the process of crossing the valleys to potentially higher fitness peaks. For the evolving organism, this represents a sort of a Malthusian trap where evolutionarily stagnant candidates at local fitness peaks outcompete all other candidates which must make enormous evolutionary jumps to even be recognised by the fitness function. In this way, genotypic expressivity can quickly become decoupled from fitness maximisation. Such candidates must typically wait for extinction events where their higher evolvability becomes more valuable than their immediate fitness values [151].

Therefore, the fitness function must not only measure performance but must also act to preserve genetic diversity to avoid converging the entire population to the first local minimum it encounters. In other words, it must ensure that it “[raises] the upper level of organization [...], while still permitting the lower types of organization to survive” [152, p. 234]. In evolutionary biology, lineages diverge when subgroups cannot reproduce with one another; this is known as *speciation*. While different species may compete for the same resources, their gene pools do not overlap, and therefore their genetic identity remains distinct across generations. In evolutionary algorithms, speciation mechanisms preserve genetic diversity by segmenting the larger population of candidate solutions into subpopulations defined by computational ‘geography’ or organisational similarity. Because candidate solutions only compete within their own species, a variety of problem-solving strategies can be explored and refined in parallel.

While reproductive selection pressure acts on the adaptive fitness of variation, it does not produce it. Wagner et al. explain that “*adaptation can proceed only to the extent that favourable mutations [variations] occur*” and note that it is a consequence of how genotypic variation maps onto phenotypic variation [111]. Therefore, *adaptive variation* is purely a property of the GP map. The term is sometimes used interchangeably with *adaptive radiation* (which refers to evolution as a concept) [153] but is used here to refer to an instance of phenotypic variation that improves fitness. Adaptive variation is crucial because it supports both *evolvability* and *stable complexification*. Evolvability is “*an organism’s capacity to generate phenotypic variation*” [154]. Stable complexification describes a GP map’s ability to preserve and elaborate existing complexity. The GP map properties and mechanisms thought to promote adaptive variation are the following:

- *Representation efficiency* describes the information density of the genotype. As noted by Stanley et al. [155] and echoed by [125], [156], [157], “*if every gene were to map directly to a single unit of phenotypic structure, evolution would be searching through an intractable [high]-dimensional genotypic space*”; they note that this is at odds with biological observations: the 100 trillion connections of the human brain are coded for by just 30 000 genes.
- *Expression* describes the phenotypic character of genotypic variation. More specifically, it describes whether a phenotype changes in a *modular, regular* or *hierarchical* way in response to genotypic variation. In this sense, the term describes the coordination of its variational properties. The ideal GP map supports all three variational properties with flexible genetic control. For example, Halder et al.’s *Drosophila* “eyeless” experiments demonstrate the expression of Hox gene variation is modular (since it only modifies the spatial relationship between the eye and body), regular (since multiple eye placements can be expressed in a coordinated way) and hierarchical (since the sub-functionality of the eyes remain intact) [158].

- *Variability* describes “*the potential or the propensity to vary*” rather than the variation itself [111]. Whereas a GP map’s expression describes its variational properties, variability describes a GP map’s capacity to change. Variability can be *finite* or *open-ended*.
- *Embryogeny* is defined as an “*embryological process of development*” [155] where genetic information is embedded symmetrically in the mapping between the genotype space, any number of intermediate developmental spaces and the phenotype space. Embryogeny relies on gene reuse to repeat patterns and trigger developmental pathways [155]. It is a means of achieving representation efficiency, coordinated expression and open-ended variability [159] as well as otherwise inaccessible developmental patterns like repetition (with/without variation), symmetry (perfect and imperfect) as well as elaborated regularity and preservation of regularity [160]. Embryogeny subsumes descriptions of the functional and temporal relationships between bits of genetic information according to the following developmental dimensions as compiled by Stanley and Miikkulainen [155]:
 - *Cell fate*: mechanisms that determine the ultimate form and role of developing structures;
 - *Targeting*: mechanisms that determine the manner and pathways by which developing structures connect;
 - *Heterochrony*: mechanisms that determine the “*timing and ordering of [developmental] events*”;
 - *Canalisation*: mechanisms which “*allow developing components to adjust to changes caused by mutations in connected components*”;
 - *Complexification*: mechanisms that accommodate genotypes with variable lengths

These points are abbreviated as REEVE (**R**epresentational **E**fficiency, **E**xpression, **V**ariability, **E**mbryogeny) properties. To illustrate the appeal of REEVE properties, consider the GP map used in the example of section 2.3.1 when expanded to solve a problem of n variables. To reiterate, an individual phenotype is a set of candidate integer values corresponding to the set of problem variables in $f(x)$. The genotypic space maps the phenotypic set onto a set of binary numbers which recombination and mutation act on to produce variation. It is a linear, one-to-one transformation.

The most apparent problem is that such an encoding would scale poorly as the dimensionality of the problem increases [161]; to discover a solution to the n -dimensional problem, evolution must search an equivalent n -dimensional space. In other words, one-to-one encodings have poor *representation efficiency*. Furthermore, this n -dimensional genotypic space is a predetermined hypercube of finite size and resolution. Evolution cannot complexify beyond this hypercube

because such solution configurations are not defined. In the example, the phenotype space is explicitly defined as the positive integer set $[0, \dots, 31]$. Consequently, evolution can only discover solutions within that finite set; in addition, all recombination and mutation effects can only produce phenotypic variation with a minimum resolution of one integer. In other words, a one-to-one encoding's *variability* is both finite and discrete. Simply expanding the range or resolution of encoded values does not create open-ended variability; it simply makes the predetermined hypercube larger. While restricted variability is appropriate for the problem of discovering a set number of variable values, more complex problems often require solution configurations that grow and complexify.

Even if its variability was open-ended, a one-to-one encoding places inherent limitations on the *expression* of its variational properties. Because the phenotypic effect of an individual genetic character is independent of any and all other genetic characters, a one-to-one encoding's expression is strictly *modular*. The converse would be an encoding where the expression of all genetic characters influences all phenotypic features equally; such an encoding would be strictly *regular*. Omitting *hierarchical* expression for the moment, it is necessary to emphasise how necessary a trade-off between modularity and regularity in a GP map actually is.

In his defence [162], Clune used the example of evolving tables to illustrate the trade-off between modular and regular GP map expressions. In this scenario, a reasonable fitness function would be one that measures the surface stability of the table. Following the given example, a modular GP map would be a one-to-one encoding that represents a set of four binary numbers each of which correspond to the length of an individual table leg. Naturally, the expression of this encoding is strictly modular but very irregular. Crossover and mutation would modify each variable independently and overwhelmingly produce tables with unequal individual leg lengths. Phenotypic variation would be largely indistinguishable from noise, and evolution would become trapped in a buzz of unstable tables whose leg lengths have no functional relationship to one another. Furthermore, such a one-to-one encoding would suffer from poor *evolvability* since the evolutionary path between a table with short legs and a table with long legs is fraught with unstable configurations. However, if the phenotypic characteristics are coded indirectly, it becomes possible to discover strictly regular configurations. Consider redefining all four of the phenotype's legs as the set $[cx_1, cx_2, cx_3, cx_4]$ where $x_1 = x_2 = x_3 = x_4 = 1$ and c is an evolving length multiplier. In this scenario, all table phenotypes have legs of equal length. As a result, evolution searches a genotype space that exclusively defines (flat surface) stable configurations, and the evolutionary path between a short table and a tall table is smooth and continuous.

However, creating this strictly regular encoding admits a significant amount of prior knowledge about tables and the relationships between their functional parts – i.e. that tables have four, equally long legs. Injecting this into the GP map directly excludes designs that do not have exactly four legs of equal length and explicitly excludes modular 'spot' adjustments. If the standing surface suddenly became

bumpy, a strictly regular encoding would fail where a strictly modular encoding would succeed. The greater problem of injecting prior knowledge into the GP map is that it is a cheap win. It does nothing to advance the mandate of “*solving the problem of solving problems*”. Therefore, the GP map should be designed to support modular and regular expressions since complex problems require both.

The challenge of designing GP maps that are representationally efficient, have open-ended variability and support modular, regular and hierarchical variational properties is significant. The current consensus appears to be that *embryogeny* is a promising potential solution [143], [155], [159], [163]. Despite describing the relatively simple concept of development through gene reuse, the definition of embryogeny is model blind and not dependent on any specific implementation or heuristic. Furthermore, while embryogeny is often a way to achieve representation efficiency, variability and coordinated expression, it is not guaranteed to surpass the performance of GP maps that omit development and gene reuse. Therefore, this study views embryogeny as a principle more than a method and groups it alongside formal attributes (being representation efficiency, variability and expression) with the abbreviation REEVE. The conclusion here is that – alongside well-defined fitness landscapes which protect solution diversity – GP maps that fully or partially incorporate REEVE properties are likely candidates for evolutionary success.

2.3.3 Neuroevolution

Concisely, neuroevolution describes a class of systems that “*applies evolutionary algorithms to construct artificial neural networks, taking inspiration from the evolution of biological nervous systems in nature*” [134]. For those with a more general interest in the emergence of complexity, it is a surprisingly good starting point since neural networks embody geometric relationships, non-linear functionality and complex internal organisations that (crucially) avoid any need to simulate/quantify the physics of physical interactions. Stanley, Clune, Lehman and Miikkulainen motivate that neuroevolution can directly inspire and contribute to general deep learning research because it “*enables important capabilities that are typically unavailable to gradient-based approaches, including learning neural network building blocks [...], hyperparameters, architectures and even the algorithms for learning themselves*”, predicting (Turing-esquely) that it “*could prove to be a critical tool in the long-term pursuit of artificial general intelligence*” [164] with a biomimetic gesture at the evolutionary origins of the unmatched performance of living brains.

Neuroevolution evolves the topology, connectivity, synaptic weights, activation functions and learning rules of neural networks. Yao distinguishes three categories “*the evolution of connection weights, of architectures and of learning rules*” [125] with the evolution of connection weights (i.e. training) being the most common. It is possible to argue that using neuroevolution as a training algorithm is somewhat groundless since gradient methods often outpace and outperform it. While there are some results to the contrary [165], neuroevolution’s major contribution is that it can scalably generate an essentially endless variety of networks while selecting for very

specific properties. Designing neural architectures by hand remains both possible and current but becomes an increasingly unfeasible strategy as networks grow in size and complexity. That said, any neuroevolutionary work is ultimately subject to all the problems (independent and intersectional) of neural network design and evolutionary computation. This section will focus on three such concerns: (1) the issue of competing conventions, (2) the interaction between lifetime and evolutionary learning and (3) the knowledge basis problem.

The first focus point is the issue of *competing conventions* [166], [167] (also called the *permutation problem* [168], *isomorphism problem* [169], *multimodality* [126] or the *problem of symmetries* [22]). Competing conventions is a consequence of the fact that neural networks can compute the same function using different weights and/or topologies. In general, n hidden units have $n!$ mathematically equivalent configurations [170]. While such expressivity may at first seem useful, crossover between two functionally identical but structurally different solutions can often result in impaired offspring [170] (as depicted in Figure 11). As a result, crossover can fail to preserve the functionality of recombinant units and could threaten stable complexification. While the severity of the problem remains under debate [22], [169], [171]–[176], the importance of crossover as a source of adaptive variation [177] strongly motivates that the chosen neuroevolutionary technique should address or at least make reference to the problem of competing conventions.

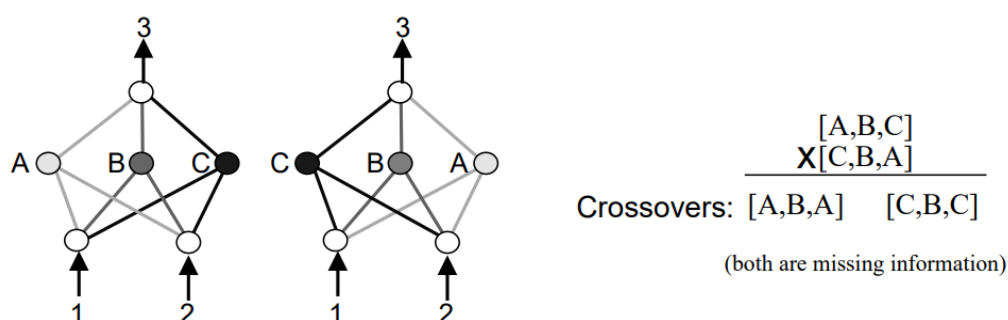


Figure 11. Competing conventions under crossover; reprinted from [170].

The second point of concern – the interaction between learning and evolution – is less of a problem and more of a design or research consideration. Functionally, learning and evolution are both a form of adaptation to the environment. The only difference is the respective temporal resolution. Whereas evolution describes generational, population-based adaptations, learning is an individual form of adaptation that occurs within each organism’s lifetime. In other words, evolution “[captures] relatively slow environmental changes that might encompass several generations” while learning “allows an individual to adapt to environmental changes that are unpredictable at the generational level”. In a sense, evolution is an optimiser taking large steps informed by the global fitness landscape while learning involves smaller steps informed by more local scenery. Furthermore, while

evolution affects the genotype-phenotype mapping as a whole, learning involves more minor phenotypic adaptations that do not modify the genotype [178].

While the benefit is that individuals have a chance to improve their relative fitness within their own lifetimes, learning costs both time and energy with the added risk that learnt behaviour may be maladaptive if the necessary stimuli is absent. Yet this trade-off is actually beneficial. While maladapted individuals gain a better chance at survival (to the benefit of genetic diversity), the incurred learning costs promote evolutionary canalisation because individuals who possess adaptive traits at birth are favoured [178]–[181]. This genetic assimilation of adaptive behaviours which otherwise would have to be learnt is known as the Baldwin effect [182].

For neural networks, lifetime learning is a training phase. If neuroevolution is used to optimise neural architecture, the designer has the option of including or omitting a training phase. Including lifetime learning means that instead of generating a population of neural architectures and directly assessing their optimality, each individual network is first trained and then assessed with respect to their performance on the chosen task. This has the added benefit of incorporating the power and fine tuning ability of gradient-based training methods [125]. In this way, evolution provides a set of starting synaptic weight values, and the training algorithm delivers the final task performance.

However, the interaction between learning and evolution in neural networks is not necessarily straightforward especially in multi-objective fitness landscapes. The Baldwin effect will only take hold if there are costs associated with learning and/or if learning is restricted in some way (i.e. learning cannot completely reconfigure the functionality of the organism). As noted previously, neural networks are highly expressive with many functionally identical competing conventions. This means that as long as they are sufficiently expressive, a set of neural networks with very different topologies can all achieve identically high performance scores. Crudely put, it is not possible to discover an evolutionary path when all candidate solutions score 100% on the test. Furthermore, modern training algorithms vastly outperform earlier methods and make even the starting synaptic values provided by evolution largely irrelevant. To summarise, this is to say that – with training – the form (topology) and function (synaptic weights) of neural networks are only partially coupled. If performance (function) is the only desired outcome, this is not an issue. However, modularity is a topological property, and therefore it is necessary to examine what effect the interaction between lifetime learning and evolution has.

The third focus point is the fact that neuroevolution (as a subset of evolutionary computation) lacks a coherent knowledge basis. Like AI, the family tree of evolutionary computation traces descent but rarely speaks to or attempts to consolidate a unified mathematical description. That is to say, for all the fascinating and empirically feasible ideas out there, the field as a whole lacks a coherent theoretical framework [183]–[185]. As a result, many neuroevolutionary ideas are islands in a sea of mismatched terminology that often settle for disorienting amalgamations of buzzworthy metaphors. The validity of such a claim is self-

evident: if a unified theoretical framework existed, there would be no legitimacy in invoking ‘DNA triplet codons’ [186] or ‘quantum qubits’ [187]. But by the vigilance of the Navier-Stokes equations, fluid mechanics has never suffered thusly.

The consequence of having no coherent knowledge basis is that most efforts are governed by the heuristic “*try to get probably good solutions to your problem, for provably good solutions are overwhelmingly hard to obtain*” [185, p. 2] with the result that most solutions skirt the edges of relevance by claiming that they are ‘competitive’ and not much else. With the inherited representation problem as neuroevolution’s dominant design problem, it becomes difficult for new research to comprehensively characterise and achieve REEVE properties while also providing comparisons to related methods when previous works do not or cannot do so themselves. The inevitable consequence is that – while these properties do have empirical and observational foundations – their application to neuroevolution comes down to conceptual insight, and any choice cannot escape familiarity bias.

Nonetheless, it remains necessary to examine as many extant neuroevolutionary encodings as possible. To do this as objectively as possible, only classical neuroevolutionary encodings are reviewed. ‘Classical’ is used here in the restrictive sense to refer to sufficiently studied work of the 1980s to early 2000s that can be compared and contrasted based on lineage and REEVE properties. The (untested but strongly suspected) assumption is that any of their unexamined descendants can be safely overlooked as merely ‘competitive’ variations or combinations of these priors. A compare-and-contrast strategy is necessary because analysing any encoding in isolation will ultimately stall at the conceptual given the lack of governing mathematics. The overview of classical encodings presented in Appendix A and the analysis presented in subsection 2.3.4 are possible courtesy of the work of Fekiač et al [188], Floreano et al [189], Koehn [190] and Yao [125].

2.3.4 Analysis of Neuroevolutionary Encoding Strategies

This section conducts a comparative analysis of the encodings documented in Appendix A with reference to their REEVE properties to support the selection of an encoding method. The appendix documents a set of direct encodings (also known as *blueprint* or *explicit* encodings) and indirect encodings (also known as *developmental*, *recipe*, *generative* or *implicit* encodings). Direct encodings (enumerated as DE-1, DE-2, DE-3, etc...) can be classified as connection-based, unit-based or pathway-based (including context-free grammar encodings and split encodings). Population-based direct encodings [191] are not reviewed since the phenotype is modelled as a population of agents which does not translate smoothly to the Tensorflow implementation of neural networks as computational graphs. Additionally, the work of Lucas [192] is excluded because apart from treating weights as independent summation units, the technique is generic. Indirect encodings (enumerated as IE-1, IE-2, IE-3, etc...) can be divided into three categories: L-systems, cellular encodings and hypercube methods. DSE [193], MENNAG [93] and G/GRADE [194] are not analysed here since they are not yet widely studied. For both direct and indirect encodings, the overall format presented

here is comparative and qualitative given the absence of a fixed standard for quantitative comparisons. Direct encodings do not incorporate embryogeny meaning only representation efficiency, expression and variability are applicable.

Connection-based direct encodings include connection bit encodings (DE-1) [126], [195], [196] as well as connection + weight bit encodings (DE-2) [166], [197]–[199]. These are neuroevolution’s earliest ancestors; their essential structure can be described as a linear, one-to-one map between connections (with/without synaptic weights) in a predetermined neural framework and individual genetic characters. While this is not necessarily a drawback, the need for a predetermined neural framework constrains and biases the type of evolved topologies that can emerge. As discussed in subsection 2.3.2, one-to-one encodings have finite variability; their expression is strictly modular, lacking both regularity and hierarchy. As such, they are good at representing individual connections but have no inherent mechanism to represent larger patterns of connections except as by-products of fitness selection. While their open-endedness and evolvability have been improved under extensions such as Messy Genetic Models (which introduce variable length genotypes) [200]–[202] and Structured Genetic Models (which introduce diversity preserving regulatory genes) [203], connection-based, bit encodings have low representation efficiencies and scale poorly [184]. That being said, bit encodings are widely used in neuroevolutionary work [123], [204], [205] since the fine control afforded by their strictly modular expressions suit problem domains with low regularity [206].

Unit-based encodings include neurons as parameter strings (DE-3) [207]–[212], neurons as parameter trees (DE-4) [127] and layers as parameter strings (DE-5) [213]–[215]. By harnessing abstractions that imply functional relationships, unit-based encodings gain a certain level of flexibility. Instead of representing the discrete components (i.e. each weight, bias, etc.) of a neuron (or layer) with multiple genes, a single gene contains all the necessary information to describe a whole neuron (or layer) as well as its functional relationship to all other neurons (or layers). Grouping the genetic information that describes a unit in this way favours open-ended variability and complexification because the functional relationship between units is preserved under insertion/deletion. However, unit-based encodings do not inherently have higher representation efficiencies since all weight and bias values must still be explicitly coded. While also strictly modular and irregular, their expression is more hierarchical compared to connection-based encodings; this is especially prominent in Koza’s parameter tree encoding, where single bit mutations can affect whole genotype subtrees. This sweeping hierarchical variation is even more pronounced in Mandischer’s encoding where genetic characters determine the relationships between whole layers; the trade-off is that such low-resolution genetic variation cannot target single connections between neurons.

Pathway-based encodings (DE-6) [216] take a different perspective; instead of explicitly representing discrete components or whole units, a genetic character codes the link between two or more neurons. Individual genetic characters are context free (i.e. any given path is not defined in terms of another), and therefore the phenotype is built by overlapping the paths. This introduces a measure of

regularity since all or part of any given pathway can be affected by multiple genetic characters. Because variation can target individual neurons or a set of links between neurons, pathway-based encodings have stronger expression hierarchies compared to both connection and unit-based encodings (barring Koza's parameter trees). Like unit-based encodings, variability is open-ended since variation operators can insert new pathways or neurons without disrupting the functional relationships of existing pathways. While the method can be extended to encode synaptic weights, pathway-based encodings are not any more representationally efficient than bit encodings.

By splitting its genotype into a node chromosome and a connection chromosome, NEAT (DE-7, *Neuroevolution of Augmenting Topologies*) [217] could be considered a hybrid between a unit-based and a connection-based encoding. While its expression modularity and regularity are only on par with connection-based encodings, NEAT's major contribution is that its genotypes can complexify in a stable and open-ended manner. At the cost of significantly reduced expression hierarchy, NEAT limits its genotype to expressing single links between neurons but gains the ability to accurately track the mutational history of individual phenotypic features using an innovation number. These innovation numbers enumerate all mutations as they happen and form a record of a given genotype's evolutionary lineage. Stanley et al. [217] motivate that networks who share a common evolutionary origin are likely to encode similar functionality and argue that restricting crossover to genotypes with sufficiently similar evolutionary lineages can help to alleviate the competing conventions problem. To protect genetic diversity, NEAT incorporates dedicated speciation mechanisms, and the phenotypic expression of its individual genetic characters can be turned on and off by mutation (mirroring Structured Genetic Models). These features make NEAT the most advanced direct encoding compared to the methods discussed previously.

Turning to indirect encodings, Lindenmayer's eponymous L-systems [218] shine in their own right as methods that can grow highly regular, biologically plausible structures with multiple symmetries through the evolution of starting characters and a set of production rules – a process that mimics parallel cell division. Such forms are fractal in nature – in other words, self-similar through recursion. L-systems have open-ended variability and high representation efficiency, since the scale of the structures they generate is only dictated by the number of rewrites; their expression is highly regular and hierarchical even if the potential for modularity is somewhat ambiguous. Despite their systems of embryogeny and expressive power, L-systems as neuroevolutionary GP maps are unwieldy and do not naturally extend to the neural language that describes a neuroevolutionary genotype or phenotype [219].

For example, prior studies [161], [220]–[223] involving L-systems had to develop somewhat ad hoc implementations derived from bit encodings. In addition, their phenotypes need significant repair work which degrades their overall efficiency [220], [223]. Nolfi et al. did not use L-systems but preserved their phenotypic fractal effect by parametrically coding for branching patterns directly [223]. Similarly, Merrill et al. [224] and Mjolsness [156] rely on recursion rather than formal L-systems to harness fractal patterns but even so, extensions to neural language

remain largely interpretative. Merrill and Port looked at fractal segmentation of network connectivity [224] but Yao argues that this is unlikely to scale smoothly [125]. The greater problem is that while L-systems can express very regular phenotypes, it is unclear how well a set of production rules can support modular, localised changes to the phenotype. Genotypic variation in L-systems typically manifests across the entire phenotype since production rules are not temporally or spatially ‘gated’ [224] and are therefore active everywhere simultaneously. In the words of Merrill and Port, “*it is not easy to see how the expression of a set of serial rules could be synchronized in a temporally and spatially continuous device like a developing organism*” [224].

Gruau contrasts his cellular encoding with L-systems – specifically those employed by Kitano [161] – by arguing for its superior modularity and scalability while also noting that (as a language) cellular encodings describe neural networks more naturally [219]. Like L-systems, cellular encodings mimic cell division but do so by acting on a graph of cells rather than by rewriting a string. The production rules of L-systems act whenever and however many times they encounter a compatible string segment; by contrast, a cellular encoding only acts when and where instructed by the genotype program tree and only for a specific number of recursive steps. In other words, cellular encodings are temporally and spatially gated while L-systems are not. In turn, this allows mutation and crossover to produce regular, modular and hierarchical effects in any part of the program tree’s hierarchy even if “*recurrence is not always portable between trees*” [194]. Essentially, Gruau’s achievement was to transform Koza’s parameter tree encoding into a developmental encoding by recasting his unit symbols as program symbols. Nonetheless, Stanley et al. show that standard cellular encodings do not necessarily outperform direct encodings [170]. In addition, cellular encodings have weak control over edges and a tendency to generate highly connected networks prompting further research [194], [225], [226] to propose operators that target edges, nodes and/or hyperedges.

The core principle of embryogeny is gene reuse which not only benefits representation efficiency but also acts to coordinate the expression of genetic information. Gated or not, both L-systems and cellular encodings are essentially biological metaphors of cell division where growth and development are the consequences of events unfolding over time. In other words, both implement embryogeny as a temporal process of iteration. However, gene reuse can also be modelled as an information network that describes the nonlinear relationships between a set of genetic bits and the phenotypic expression of those bits.

While L-systems and cellular encodings rely on temporal unfolding, HyperNEAT (*Hypercube-based NeuroEvolution of Augmenting Topologies*) [227] describe such information networks using CPPNs (*Compositional Pattern Producing Networks*) [160] which inscribe the developmental information as patterns within hypercubes in Cartesian space. Concisely, CPPNs “*describe the structural relationships that result from a process of development without simulating the process itself*” [160]. HyperNEAT’s expression is not dictated by iteration steps or intermediate forms but rather by the structure of the CPPN’s composed function. As a result, the entire

phenotype is built by executing the CPPN once. HyperNEAT extends NEAT in a way that is analogous to the mapping of DNA \rightarrow RNA \rightarrow cell structure: NEAT encodes and evolves the CPPN that is activated to produce the phenotype. This means that simply reading the directly encoded NEAT genetic characters will not give any insight since phenotypic expression is defined by the CPPN's input-output mapping. Consequently, the information encoded by HyperNEAT's GP map is distributed symmetrically between the NEAT encoding and the CPPN it describes.

Structurally, CPPNs are (typically small) networks that implement a large variety of activation functions (e.g. sine, cosine, tangent, exponential, relu, sigmoid, etc.) and input aggregation methods (e.g. weighted sum, weighted absolute maximum, weight absolute minimum, etc.) compared to formal neural networks. As a result, their structure is very naturally supported and evolved by NEAT. As shown in Figure 12 below, the nonlinear interactions between a CPPN's different activation functions interact with one another to describe a pattern within a hypercube whose dimensionality is determined by the CPPN's inputs. CPPNs work by taking inputs from a predetermined substrate (such as those shown in Figure 15) and predicting what value it should be. The two-dimensional CPPN shown in Figure 12 can be used to evolve images such as those shown in Figure 13; Stanley [160] evolved such images to demonstrate that CPPNs can express regular effects such as repetition (with/without variation), symmetry (perfect and imperfect) as well as preserved and elaborated regularity. Furthermore, CPPNs allow modular control over complex features; for example, shifting the eye structure shown in Figure 14 was done by mutating a single synaptic weight. Such control over complex structural hierarchies is reminiscent of the effects of the Hox gene manipulations in Halder's *Drosophila* experiments [158]. To evolve neural networks, HyperNEAT only requires the appropriate neural substrate, e.g. the grid, three-dimensional, sandwich or circular substrates shown in Figure 15.

Such factors make HyperNEAT a refined abstraction of embryogeny. Furthermore, HyperNEAT's phenotype complexity is decoupled from the complexity of its genotype making is more representationally efficient than either L-systems or cellular encodings since small CPPNs can encode dense patterns. Additionally, HyperNEAT supports open-ended variability and is scale free as demonstrated by Stanley et al. [227] who evolved a neural network with 14, 641 connections which preserved its functionality when applied (without additional evolution) to nine million connections. While its performance is dependent on the structure of its substrate [228], HyperNEAT supports regular, modular and hierarchical expressions. Because HyperNEAT evolution searches the space of all CPPNs networks, it benefits from NEAT's preservation of genetic diversity and protections against competing conventions. Although HyperNEAT is capable of placing the same emphasis on neurons and edges, it has been shown to be better at solving regular problems than modular problems [229], [230]. Whereas standard HyperNEAT implementations require a predetermined substrate, this restriction has been lifted by extensions that allow the substrate itself to evolve [231]–[233].

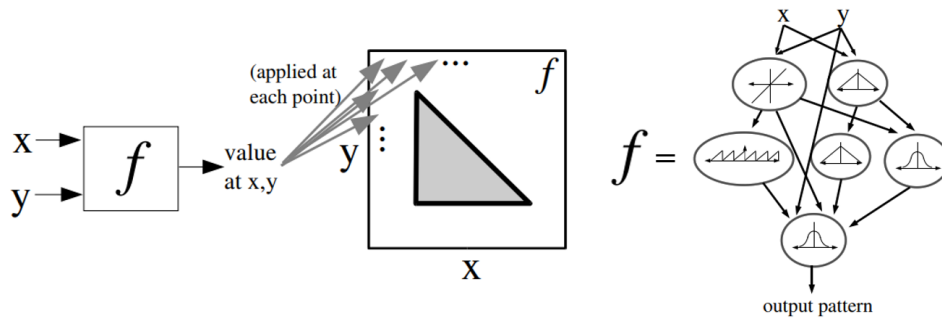


Figure 12. Structure and operation of CPPNs; reprinted from [160].

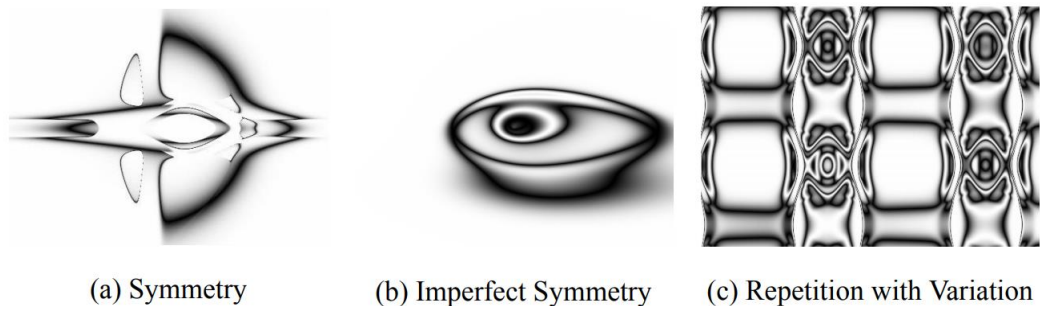


Figure 13. Abstraction of development by CPPNs; reprinted from [160].

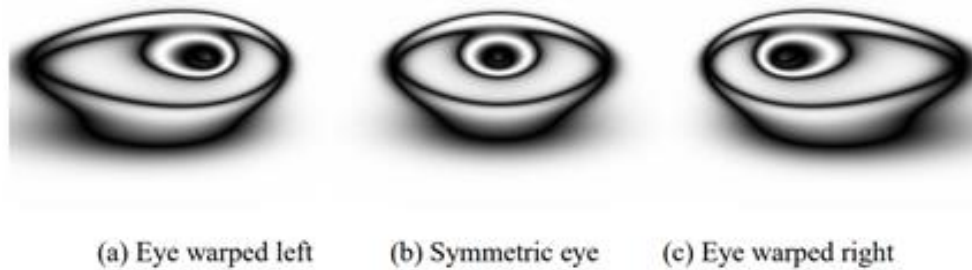


Figure 14. Modular control in evolved CPPNs. Reprinted from [160].

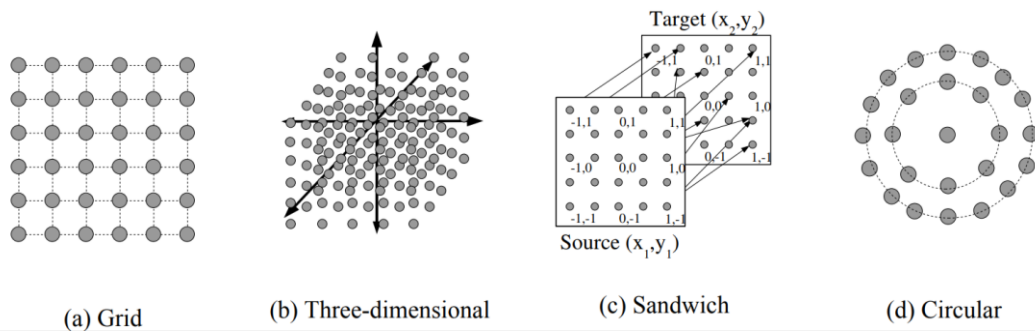


Figure 15. HyperNEAT substrates; reprinted from [227].

2.3.5 Evolutionary Drivers of Neural Modularity

Previous studies have shown no universal consensus about what the fundamental drivers of neural modularity are but do constitute a fascinating and diverse body of literature. Methods that rely on predesigned modules [234] or handcrafted structural incentives [122] will not be reviewed here since the aim is generality through evolutionary techniques. As discussed previously, specific evolutionary outcomes (such as modularity) can be achieved as a result of the structure of the GP map and/or through fitness incentivisation. The REEVE properties of novel or significantly modified GP maps must first be studied before they can be applied to any work focusing on neural modularity. Since this is beyond the scope of this work, this section will review methods which involve learning or fitness incentivisation.

The broadest hypothesis is that evolution directly selects for neural modularity because it is a universal optimum with regards to performance (in situ static optimality by avoiding disruptive interference/crosstalk between independent functions [119], [121], [235]) and evolvability (generational adaptive optimality [111], [116], [236] across different environments [106]–[108] and extinction events [151]). Although valid and encouraging results have been presented, this idea has its difficulties. Firstly, the interaction between genotypic and phenotypic modularity is often vague [205] yet one does not guarantee the other. Modular genetic control may benefit evolvability but can produce integrated phenotypes just as easily as modular ones. Nondeterministic GP maps may inherently bias evolution toward modularity, but most artificial genetic systems lack dedicated protections for existing modules [237]. Secondly, Bullinaria's work points out that the modular benefits of avoiding crosstalk in neural networks is largely problem dependent since *“for many tasks there is no learning advantage for modularity because the reduction in cross-task interference that modularity provides is outweighed by the extra computational power allowed by full connectivity”* [235, p. 124]. This is echoed by Kashtan who notes that modular neural networks are often less optimal than nonmodular ones [106]. Thirdly, modularity as a response to varying environments may be biological reality, but as an algorithmic technique it is sensitive to experimental conditions and therefore impractical [237], [238]. Overall, these three points agree well with statements by Kim et al. – in addition to statements by [95], [239] – which frame the emergence of modularity as an unusual and nontrivial evolutionary outcome [240].

Since it does not appear to be an automatic outcome, *“it seems that the origin of modularity requires both a mutational process that favours the origin of modularity and selection pressures that can take advantages of and reinforce the mutational bias”* [116]. Since neural modularity is a connectivity property, selection pressures that incentivise evolution towards locality are particularly well explored [111], [121], [123], [204], [205], [241]–[245]. Though not implemented as an evolutionary algorithm, Jacobs and Jordan motivated that short and local connectivity in neural networks would produce faster real-time processing, reduced spatial requirements and crosstalk, task decomposition, decoupling of recurrent dynamics as well as improved interpretability [121]. They used a modified weight pruning algorithm by

Rumelhart [241] that favours strong local and weak distant weights and concluded (by visual inspection of the weight matrices) that the evolved neural networks have a tendency towards task decomposition and local-sensitive internal representations.

Although Jacobs and Jordan's result is sensitive to the learning vagaries mentioned previously [242], the principle of using a connectivity constraint to encourage the evolution of neural modularity is corroborated by Clune et al. [204] who make the argument that the energy costs of manufacturing, maintaining and signalling through long and/or many connections are a relevant evolutionary constraint. Using a direct encoding and a retina problem as the learning task, Clune et al. tested two constraints: P&CC which measures the sum of a network's squared connection distances and P&CC-NC which measures the total number of connections in a network. Their results showed that both produced statistically significant levels of modularity (as measured by Newman's Q-score). This indicates that evolving neural networks while constraining connectivity has a parcellation effect that promotes the formation of distinct subnetworks. This result is further supported by related work (also using direct encodings) that investigated the evolution of internal models [205] and how neural modularity can alleviate catastrophic forgetting [123].

By itself, HyperNEAT does not automatically favour modular networks [243]. However, Huizinga et al. [244] showed that networks evolved by HyperNEAT under connection costs (CCT) are consistently more modular than those produced by HyperNEAT alone or HyperNEAT plus a Gaussian seed (a modularity technique described by [245] that incorporates a bias toward short connections); interestingly, HyperNEAT under connection costs did not produce higher levels of neural modularity than Clune et al.'s direct encoding but did produce modular networks which were more regular than those produced by the direct encoding. Overall, such results are good indications that connectivity constraints can guide evolution toward modular architectures. That being said, it is important to note that [204], [244] used preconditioned substrates whose node placement favours modular decompositions.

2.4 Conclusions

There is increasing recognition of the fact that the neural network interpretability problem frustrates meaningful human-AI interaction and compounds their learning fragilities as well as the threat of adversarial attacks. Unfortunately, this problem is difficult and remains unsolved. The legitimacy of interpretability through neural modularity as a potential solution is established with reference to the fundamental principles of neural networks, the theory of associative memory and problem decomposition principles. The argument is that when a neural network is modular, its internal representations are similarly disjoint. If neural modules contain explanative groupings of such representations, a modular network becomes more interpretable than an equivalent fully connected network. Because a process of functional identification is still required, neural modularity does not fit neatly within the interpretability frameworks discussed in subsection 2.1.4 but can feasibly benefit all three depending on what form it takes.

The question of generating neural modularity admits the obvious bifurcation of manual design versus emergence. Emergence through neuroevolution is motivated from the observation that manual design lacks sufficient prior knowledge and scalability as well as being at odds with Fogel's philosophy of AI being the solution of the problem of solving problems. Neuroevolution is examined with focus on the competing conventions problem as well as the interaction between lifetime and evolutionary learning. Because evolutionary computation's lack of a unified theoretical framework makes objective comparisons difficult, only 'classical' encodings are evaluated on the basis of their REEVE properties. Of these, HyperNEAT is identified as the most suitable, because it has been widely studied, addresses the competing conventions problem, includes speciation mechanisms that protect genetic diversity and supports REEVE properties. The evolutionary drivers of neural modularity are examined, identifying the work of Clune et al. [204] as an effective method of evolving neural modularity with fitness incentivisation. Their approach abstracts the biological metaphor of energy conservation as a connectivity constraint that measures the summed squares of a network's connection distances. With this technique, Clune et al. evolved neural modularity by formulating a fitness function that selects for networks which maximise task performance and minimise the connectivity constraint. Huizinga et al. [244] subsequently demonstrated that connectivity constraints are also effective when applied to HyperNEAT.

While demonstrating that connectivity constraints are a feasible method of generating neural modularity, both Clune et al. [204] and Huizinga et al. [244] relied on preconditioned substrates whose node placement inherently favours modular decompositions. Therefore, it remains unclear what effect such connectivity constraints will have when applied to more general substrates. It is also possible that connectivity constraints other than those based on connection costs could also promote the evolution of neural modularity. In both cases [204], [244], topology and synaptic weights were evolved simultaneously; in other words, no lifetime learning (i.e. training) took place. As such, there is room to speculate whether such methods can benefit from the power of gradient-based training. However, the inclusion of lifetime learning prompts questions about the coupling between form and function (given the Baldwin effect) that would make it necessary to investigate how the interaction between lifetime and evolutionary learning affects the evolution of neural modularity. Furthermore, it is interesting to note that HyperNEAT under connection costs does not produce higher modularity levels than a direct encoding [244]; Huizinga et al. interpret this as the consequence of HyperNEAT's modularity-regularity trade-off. By mediating the phenotypic expression of its genotype with a CPPN, most variation in HyperNEAT's genotype space causes collective and coordinated movements in the phenotype space. In contrast, genetic variation in direct encodings causes strictly one-to-one movements in the phenotype space. Therefore, a direct encoding can more easily prune specific connections while leaving others intact purely because it lacks regularity of any kind – indicating that neural modularity benefits from a highly modular genotypic expression. As such, it would be interesting to study minor modifications to HyperNEAT which improve its modular variational properties.

3 Experimental Method

The purpose of this chapter is to document the research design, methodology and experiments conducted in support of the aims. Section 3.1 reviews the findings of Chapter 2 and synthesises the research perspective. Section 3.2 details the techniques and execution of the previous. Section 3.3 details the experiment design.

3.1 Research Design

With the aim of investigating the evolution of neural modularity as well as its benefit to solving the neural network interpretability problem, there is a logical demand to make some statement about which neuroevolutionary methods work best with respect to the evolution of neural modularity. While HyperNEAT's properties make it the most suitable of the reviewed encodings, a key insight of the literature review is that no neuroevolutionary method can be described as "*provably good*" [185, p. 2] because all are vulnerable to the unknowns, points of contention and hyperparameter sensitivities arising from evolutionary computation's lack of a coherent knowledge basis. Given this provability limitation, investigating the evolution of neural modularity (objective 3) and studying its benefit to solving the interpretability problem (objective 4) are framed as explorative rather than declarative endeavours; in other words, objective 3 and 4's supporting experiments are structured to examine previously unexplored research points of interest and to contribute to the knowledge basis on evolving neural modularity using HyperNEAT under connectivity constraints. For this reason and in contrast to prior work [244], [245], HyperNEAT is implemented with standard speciation and crossover as described by Stanley et al. [227]. Testing different hyperparameters or genetic algorithm paradigms is left up to future research. As such, the research points of interest are the following: general substrates, alternative connectivity constraints, modifications that improve HyperNEAT's modular variational properties, and how the interaction between lifetime and evolutionary learning influences the evolution of neural modularity. Once evolved, the resultant neural modularity is subsequently assessed with respect to its interpretable qualities.

3.1.1 Preconditioned versus General Substrates

The purpose of the substrate is to represent the spatial geometry of all evolvable neural features for use as inputs to HyperNEAT's CPPNs. The perspective of this study is that preconditioned substrates will inevitably outperform general substrates in facilitating the evolution of neural modularity because they inherently bias the process towards a predetermined decomposition pattern. As such, the outcome of a comparison between preconditioned and general substrates is considered too self-evident to pursue experimentally. Furthermore, preconditioned substrates are restrictive. Those used by Clune et al. [204] and Huizinga et al. [244] only permitted interlayer connections meaning that evolution could not alter the layer hierarchy or scale of the evolving networks. While HyperNEAT does require a predetermined

substrate, this study will apply the heuristic of imposing as few constraints as possible. Therefore, the substrate design used here will be general and explicitly allow supralayer connections. By allowing each neuron to connect to any neuron below it with supralayer connections, entire layers can be omitted, and it becomes possible for evolving networks to collapse into smaller networks. With such a substrate, evolution can express small networks (with a single hidden neuron) or large networks (with all possible neurons in all possible layers described by the substrate) – as well as all topologies in between. As a result – using a substrate that allows supralayer connections – HyperNEAT can explore a larger region in the space of all possible neural networks.

Depicted in Figure 16, neurons are embedded in three-dimensional stacks of one or two-dimensional layer grids. Using one-dimensional layer grids for the hidden neurons makes the composed neural network two-dimensional (i.e. a flat ‘sheet’ of neurons connected to a two-dimensional grid of input neurons). Using two-dimensional layer grids make the composed neural network three-dimensional. While the number of neuron grid points can vary, all $[x, y]$ neuron coordinates are normalised between $[-1, -1]$ and $[+1, +1]$ and scaled to 2 (input layer), 1 (hidden layers) and 0.5 (output layer) respectively. Successive layer stacks are indicated by increments of z ; the input layer is at $z = 0$; the first hidden layer is at $z = 1$; the last hidden layer is at $z = n - 1$, and the output layer is at $z = n$. Each neuron position is stored as an $[x, y, z]$ coordinate within this hierarchy. In addition to the Euclidean distance between them, the $[x, y, z]$ positions of two neurons are fed into HyperNEAT’s CPPN to determine that connection’s synaptic weight. To determine a neuron’s individual bias or nomination value, the CPPN is fed the $[x, y, z]$ position of the target neuron and $[0, 0, -1]$ as the position of the dummy neuron.

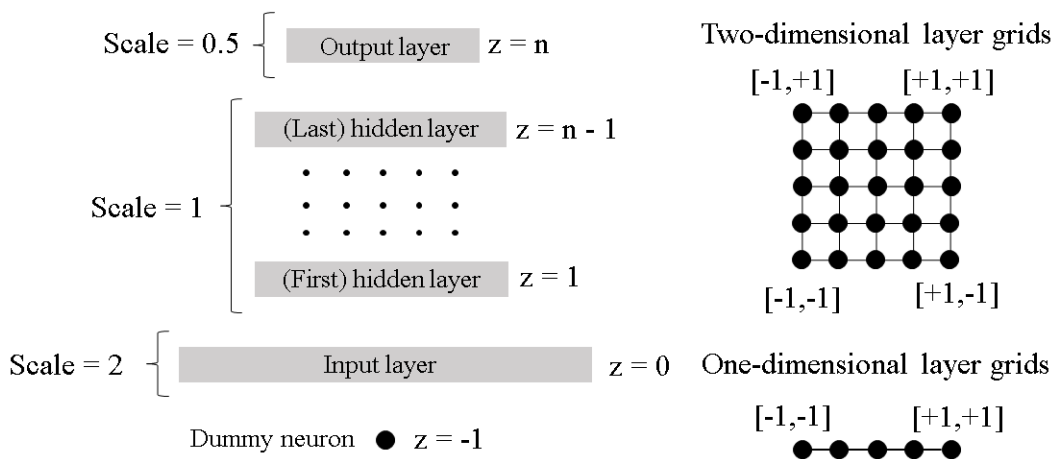


Figure 16. Substrate spatial geometry showing layer scaling and hierarchy.

All spatial geometry is stored in the same format as the network’s synaptic weight matrix in order to access the information with minimal indexing (Figure 17). In other words, a matrix of equivalent dimensions and structure is reserved for the

coordinate from a given neuron ($[x_{from}, y_{from}, z_{from}]$) and the coordinate to any other neuron ($[x_{to}, y_{to}, z_{to}]$) as well as the distance (d) between. As an example, the positions and distance necessary for a CPPN call on the first hidden neuron's (dark blue at $[i_4, j_4, -]$) connection to the first input (green at $[i_1, j_1, -]$) are obtained by simply targeting the connection's index at $[i_4, j_1, -]$ and pulling the values from each matrix array at $k_1, k_2, k_3, k_4, k_5, k_6, k_7$. HyperNEAT's CPPN takes this set as its input, and its outputs are subsequently applied to the same index in the connection weight matrix (k_8) but not the connection nomination matrix (k_9). While they can be controlled individually, the nomination values of individual connections are conditioned on the nomination status of the corresponding neuron's bias. To apply to bias parameters, CPPN takes the target neuron's position ($[i_4, j_4, -]$) and the position of the dummy neuron ($[0, 0, -1]$) as input. The bias value and nomination value are set in matrix k_{10} and k_{11} respectively. The bias nomination value is then retroactively applied to all input connections in matrix k_9 corresponding to its parent neuron. For convenience, all spatial geometry and output matrices are appended to the same data structure.

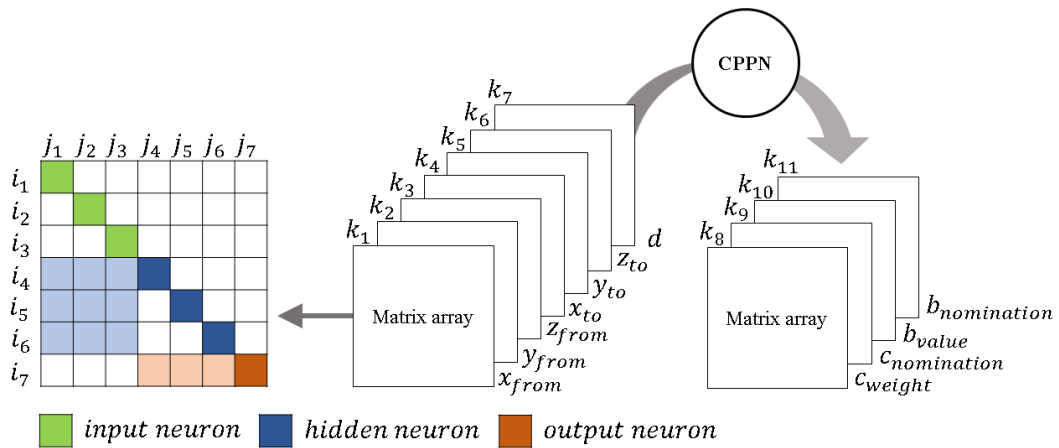


Figure 17. Substrate data structure.

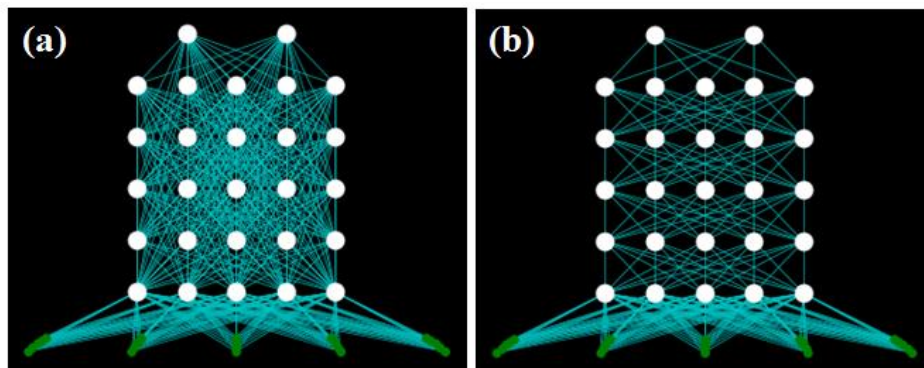


Figure 18. Substrate configuration with (a) both superlayer and supralayer connections vs (b) only superlayer connections.

As discussed, the substrate permits supralayer feedforward connections to allow scale variation. In other words, a given layer can connect to any hidden layers below it. Since inputs only feed into the first hidden layer, all evolving networks must have a minimum of one active neuron in the first hidden layer to connect to the output(s). Matrix arrays k_8, k_9, k_{10}, k_{11} store HyperNEAT's CPPN outputs and define the network structure. In the default substrate (Figure 18, a), all superlayer and supralayer connections are available to evolution; to restrict supralayer connections, the relevant connection weights and nominations can be retroactively zeroed after a CPPN call (Figure 18, b).

3.1.2 Connectivity Constraints

With a connectivity constraint based on connection costs, Clune et al. posit that the energy costs of manufacturing, maintaining and signalling through long and/or many connections constitutes a biologically plausible evolutionary constraint that works in favour of neural modularity [204]; the success of this method indicates that variants and novel constraints are worth exploring. Therefore, the connectivity constraints listed in Table 2 are examined in terms of their ability to promote the evolution of neural modularity. The constraints fall into two categories: connection costs (derived from Clune et al. [204]) and input competition (novel). The essential distinction between the two is that where connection costs abstract the energy conservation principle as a selection pressure against long synaptic connections, the input competition constraints promote input orthogonality. The connection cost constraints are LCC ('local connection cost') – implemented as described by Clune et al. [204] – and GCC ('global connection cost') as a variant. The input competition constraints are FIC ('free input competition'), GIC ('greedy input competition') and EIC ('equal input competition'). All constraints are formulated for feedforward neural networks and are not necessarily applicable to different architectures.

Table 2. Connectivity constraints

Abbreviation	Name	Equation
LCC	Local connection cost	3.4
GCC	Global connection cost	3.5
FIC	'Free' input competition	3.9
GIC	'Greedy' input competition	3.10
EIC	'Equal' input competition	3.11

The constraints of Table 2 are defined in terms of a neural network's weight, connectivity and distance matrices, each with m rows and n columns; since the matrices describe neurons, the matrix is square ($m = n$). Each matrix includes the network's input nodes; hidden neurons and output neurons starting at index h and index o respectively. Equation 3.1 defines W as the weight value matrix. Equation 3.2 defines C as the binary connectivity matrix, where $c_{ij} = 1$ if a connection exists

and $c_{ij} = 0$ if it does not. Equation 3.3 defines D which describes the Euclidean distance of each connection described by W .

$$W = [w_{ij}] = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix} \quad (3.1)$$

$$C = [c_{ij}] = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{bmatrix} \quad (3.2)$$

$$D = [d_{ij}] = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1n} \\ d_{21} & d_{22} & \cdots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m1} & d_{m2} & \cdots & d_{mn} \end{bmatrix} \quad (3.3)$$

LCC (equation 3.4) measures the summed squares of a neural network's expressed connection distances. It is a modified variance equation that defines the mean connection distance as zero. Since D describes all possible connection distances (expressed or not), LCC is formulated here with a nonzero condition on the connection's associated synaptic weight. Because LCC measures variance, it acts locally: selecting for a low LCC value will minimise the expressed connection distances of each individual neuron. As a result, minimising LCC during evolution selects for neurons with few, short connections.

$$LCC = \sum_{i=h}^m \sum_{j=1}^n d_{ij}^2 [w_{ij} \neq 0] \quad (3.4)$$

By contrast, GCC (equation 3.5) is a global connection cost that measures the fraction of expressed connection distances (conditioned on w_{ij} being nonzero) versus total possible connection distances. Minimising GCC does not distinguish between many, short connections and few, long connections, because it is not a measure of variance. It only requires that the sum of all expressed connection distances be low. In other words, GCC acts on all neurons as a collective and permits a greater variety of long and short connections than LCC.

$$GCC = \frac{\sum_{i=h}^m \sum_{j=1}^n d_{ij} [w_{ij} \neq 0]}{\sum_{i=h}^m \sum_{j=1}^n d_{ij}} \quad (3.5)$$

FIC ('free' input competition), GIC ('greedy' input competition) and EIC ('equal' input competition) are a class of constraints that simulate a competition for inputs among neurons. In this scenario, neurons in all layers do not want to 'share' inputs

with any other neurons. In more technical terms, FIC, GIC and EIC promote orthogonality between any given neuron's inputs and those of all other neurons. While complete input orthogonality breaks functionality, a neural network whose neurons are partially orthogonal to most other neurons is one that contains a number of disjoint regions; in other words, it may be modular. FIC, GIC and EIC share a base measure of input competition defined in terms of the row vectors of C :

$$C = \begin{pmatrix} r_1 \\ \vdots \\ r_m \end{pmatrix}. \quad (3.6)$$

To measure input competition (IC in equation 3.7), the outer summation starts with the first hidden neuron's input vector r_k at index h and continues until $k = n - 1$. The inner summation starts with vector r_i at index $k + 1$ and continues until $i = m$. At each step, the elementwise addition of r_k and r_i (any overlap adds to 2, no overlap adds to 1 or 0) is shifted by -1 (which shifts overlaps to 1 and no overlaps to 0 or -1), clipped to zero values (which indicates overlaps as 1 and no overlaps as 0), and the resultant vector entries are summed. This total value is then normalised with respect to the number of active hidden and output neurons, making it a measure of input overlap per active neuron.

$$IC = \sum_{k=h}^{n-1} \sum_{i>k}^m \text{sum}\{((r_k + r_i) - [1,1,1, \dots, 1])^+\} / (H_{active} + O_{active}) \quad (3.7)$$

In FIC (equation 3.9), IC is minimised exactly as it is defined in equation 3.7. In other words, input overlap per neuron is minimised with no selection pressure on the variance of the number of inputs per neuron. 'Free' variance means that as long as there is minimal input overlap, both densely and sparsely connected neurons are permissible. In contrast, GIC and EIC constrain this variance. Selecting for a high variance (GIC, equation 3.10) favours 'greedy' neurons that have as many non-overlapping inputs as possible. Selecting for a low variance (EIC, equation 3.11) encourages 'equal' neurons that have roughly the same number of non-overlapping inputs. Connections per neuron variance (CpN_{var}) is defined by equation 3.8, conditioned on whether the i^{th} neuron is active.

$$CpN_{var} = \sum_{i=1}^m (\text{sum}\{r_i\} - \mu)^2 [\text{neuron}_i \text{ is active}] / (H_{active} + O_{active}) \quad (3.8)$$

Therefore, FIC, GIC and EIC can be formulated as the following:

$$FIC \rightarrow \text{minimize } IC, \quad (3.9)$$

$$GIC \rightarrow \text{minimize } IC + \text{maximize } CpN_{var}, \quad (3.10)$$

$$EIC \rightarrow \text{minimize } IC + \text{minimize } CpN_{var}. \quad (3.11)$$

To demonstrate that the constraints favour modularity, the value of each is calculated for the fully connected network of Figure 7 and the modular network of Figure 8. Listed in Table 3, all constraint values are smaller for the modular network and higher for the fully connected network with the exception of CpN_{var} . This indicates that while maximising CpN_{var} (such as in the case of GIC) favours the example modular topology, it does not rule out EIC (which minimises CpN_{var}) as a modularity inducing constraints since many different modular configurations exist.

Table 3. Comparison of connectivity constraint values for a fully connected neural network and a modular neural network.

Constraint	Fully connected network	Modular network
LCC	1085.260	1051.370
GCC	0.910	0.871
IC	32.800	30.400
CpN_{var}	507.840	552.960

3.1.3 Improving HyperNEAT's Modular Variational Properties with CPPN Disjoints

Image evolution experiments reveal interesting differences between the images built by CPPNs that use only sum aggregation and those which use sum and maximum absolute aggregation. By inspection, the images built with sum aggregation are smoother and decidedly more regular with few (if any) strong disjoints (Figure 19, a); these correspond well to those by Taylor et al. [246]. By contrast, CPPNs that use sum and maximum absolute aggregation produce images with stronger disjoints and express square structures more easily (Figure 19, b). While such images are not necessarily more modular, it is reasonable to expect that strong disjoints could potentially benefit the evolution of modular connectivity matrices that match those of Figure 8. The disjoint effect appears to be dependent on the nonlinear interactions between the two aggregation methods since CPPNs using maximum absolute aggregation alone produce only monotone images.

However, including maximum absolute as an aggregation option does not always produce disjoints – possibility due to nonlinear interactions between it, sum aggregation and the various activation functions. However, the effect of maximum absolute aggregation can be approximated with step functions (Figure 20). Restricting the range of available activation functions to step and sine functions exclusively allows the corresponding CPPNs to produce graded patterns with repetition that also incorporate strong disjoints (Figure 21). The work of Huizinga et al. [244] and Verbancsics et al. [245] do not mention aggregation settings or specific activation functions, and therefore the effect of strong disjoints on the emergence of neural modularity as a result of either appears previously unexplored.

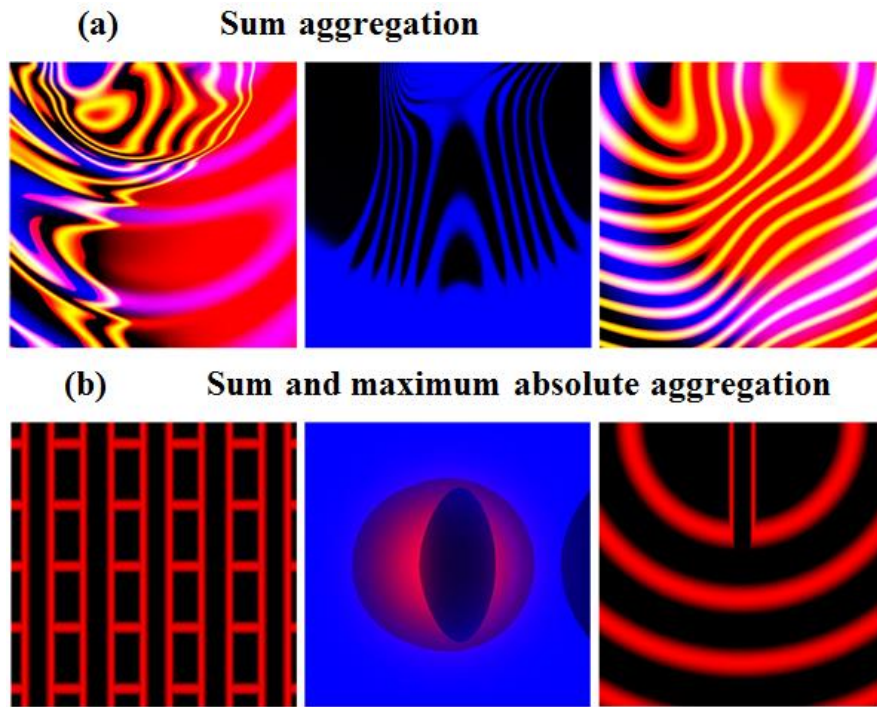


Figure 19. Differences between images produced (a) only with sum aggregation and (b) with both sum and maximum absolute aggregation.

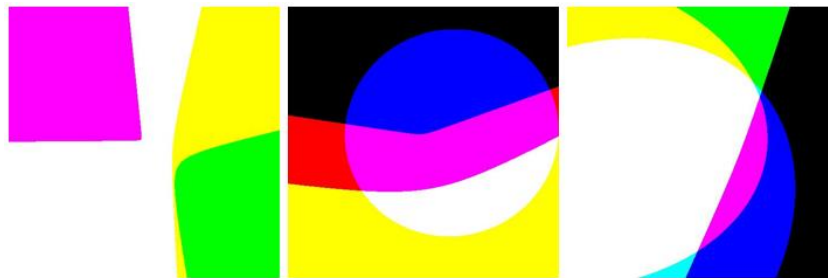


Figure 20. Images produced by evolved CPPNs (with sum aggregation) which use only step activation functions.

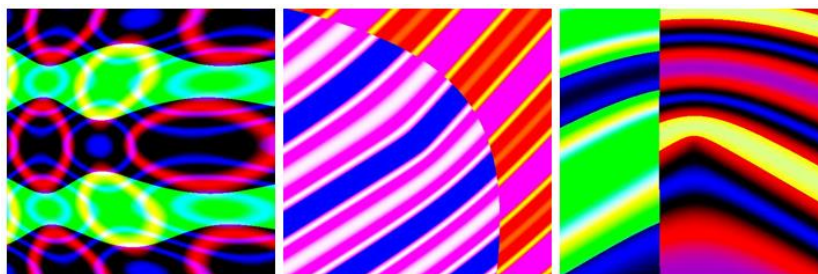


Figure 21. Images produced by evolved CPPNs (with sum aggregation) which use step and sine activation functions.

3.1.4 Studying the Interaction between Lifetime and Evolutionary Learning with Neuron Nomination

Allowing evolving networks to train (to improve their fitness by learning within their lifetimes) combines the power and fine tuning of gradient-based training algorithms with neuroevolutionary topology optimisation. However, the including training phase may decouple the processes that determine a network's form (topology) and function (synaptic weights) to such an extent that the former does not influence the latter. The issue of competing conventions paired with powerful gradient-based training ensures that as long as a given network is expressive enough, its topological configuration may contribute little to its final task performance. As such, it is necessary to investigate how the interaction between lifetime and evolutionary learning influences the evolution of neural modularity.

The intersection between lifetime and evolutionary learning is the Baldwin effect: the assimilation of adaptive behaviours which must otherwise be learnt [182]; in other words, a Baldwin effect describes heritable learnt characteristics contributing to ultimate performance. To study the interaction between lifetime and evolutionary learning, it is necessary to approximate and force various levels of the Baldwin effect (i.e. to force heritable learnt characteristics to have an impact on ultimate performance). To approximate the Baldwin effect, each neuron is paired with an evolved binary parameter which dictates whether it is nominated for training. A nomination value of 1 means that the neuron can tune its synaptic weights during the network's training 'lifetime'. Conversely, a nomination value of 0 means that the neuron's synaptic weights are locked to the values set by evolution and cannot be tuned by training. As such, an unnominated neuron's functionality is determined by evolution alone. Minimising the number of neurons that are nominated for training forces evolution to hardcode some or all of the network's functionality – in other words, heritable characteristics (i.e. evolved synaptic weights) are forced to contribute to the ultimate performance of the neural network. In this way, neuron nomination can prevent the training phase from completely decoupling a network's evolved topological properties (like modularity) from its learnt properties.

Table 4 lists four categories of interaction between lifetime and evolutionary learning described by the strength of the Baldwin effect, whether neuron nomination is static or evolving, whether neuron nomination is minimised by selection pressure and whether a training phase is included (i.e. whether the Baldwin effect is present at all). The first evolves the networks without a training phase meaning that evolution exclusively determines both the form and function of the networks (i.e. no interaction between lifetime and evolutionary learning). Since no changes as a result of interaction with its environment can occur, there is no Baldwin effect. The second is an implicit interaction with a weak Baldwin effect where evolving networks can train, but their nomination values do not evolve (i.e. default to 1) and are not minimised by the fitness function; therefore, the only interaction between evolution and lifetime learning is implicitly through starting synaptic weight values. The third is an explicit interaction with a weak Baldwin

effect where evolving networks can train, and nomination values do evolve but are not minimised by the fitness function. This creates a scenario where evolution can hardcode functionality but only if it provides an adaptive advantage. The fourth is an explicit interaction with a strong Baldwin effect where networks train and their nomination values evolve but the total number of neuron nominations is also minimised by the fitness function. This pushes evolution to hardcode functionality while also harnessing lifetime learning.

Table 4. Interaction categories between lifetime and evolutionary learning.

Interaction	Baldwin Effect	Nominations evolve?	Nominations minimised?	Trains?
None	None	False	False	False
Implicit	Weak	False	False	True
Explicit	Weak	True	False	True
Explicit	Strong	True	True	True

3.1.5 Interpretability Assessment

The idea put forward in section 2.2.2 is that an interpretable modular neural network is one that disentangles its internal representations and groups them within functionally specialised subnetworks. In other words, a modular neural network's contribution to interpretability can be framed as how functionally independent its internal subnetworks are. While analysing representation disentanglement requires more dedicated study, here it suffices to assess the evolved neural networks' interpretable qualities in terms of how functionally independent the pathways responsible for the different outputs are. Networks that share fewer neurons between their internal functional modules are considered more interpretable than networks with more overlap. Assessing interpretability in terms of functional module overlap is not meant to make comprehensive explanative statements or serve as a practical tool. Rather, the method is meant to quantify levels of functional independence for the purpose of characterising the different ways that neural modularity could evolve in lieu of more dedicated future research.

The method used to identify a network's functional modules is an implementation of Velez and Clune's subsets regression on network connectivity (SRC) which highlights the functional modules (FMs) and core functional network (CFN) within a trained neural network [247]. SRC's essential feature is multivariable regression applied to model the relationship between the activations that flow into a target neuron and that neuron's response to those inputs. As defined in equation 3.12, the algorithm models a target neuron's output y in terms of predictor variables $x_1, x_2, x_3, \dots, x_p$ (and their second order interactions, I defined in equation 3.13) which represent the neurons the target neuron takes input from. The idea is to discover the best (as measured by the coefficient of determination, R^2) and smallest

subset of predictor variables that adequately predicts the target neuron's response. Having found those, the algorithm is recursively applied to the neurons in the predictor set until the first hidden layer is reached.

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p + I \quad (3.12)$$

$$I = \beta_{12} x_1 x_2 + \beta_{13} x_1 x_3 + \beta_{23} x_2 x_3 + \dots + \beta_{p-1} x_{p-1} x_p \quad (3.13)$$

Applied to the fully connected (Figure 7, pg. 18) and modular network (Figure 8, pg. 18) examples, SRC is able to discover the functional modules (preserving relevant task performance) corresponding to the *True* class (Figure 22, a, d), the *False* class (Figure 22, b, e) and the internal CFN (Figure 22, c, f).

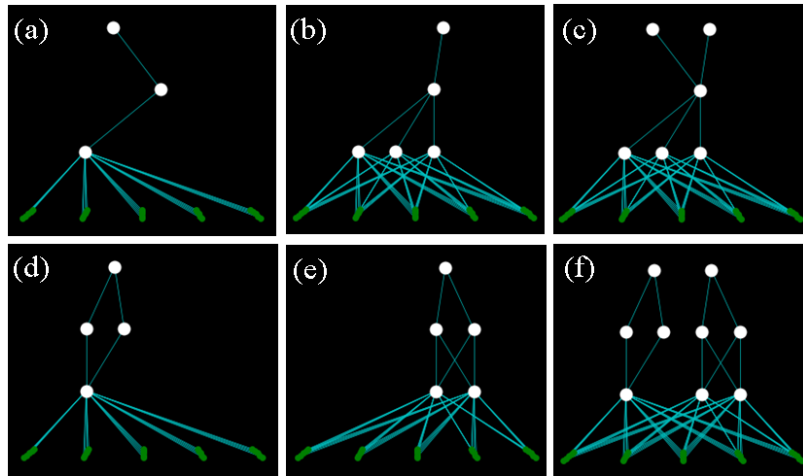


Figure 22. *True* class FMs for the fully connected network (a) [94.8%] and modular network (d) [97.4%]; *False* class FMs for the fully connected network (b) [97.0%] and the modular network (e) [97.6%]. CFNs for the fully connected network (c) [97.8%] and the modular network (f) [95.0%].

3.2 Methodology

3.2.1 Neural Network and Training Details

The evolved neural networks are ported to Tensorflow for training and ported back to the substrate data structure for subsequent analysis; these Tensorflow models have been validated against manually coded network computations to confirm that supralayer connections work and train as expected. Hidden and output neurons use relu (chosen to avoid the problem of vanishing gradients) and softmax activation functions respectively. Network outputs are one-hot encodings. Given the use of a one-hot encoding, CFNs are constructed by combining the functional modules identified by SRC that correspond to each output. For training, Adam optimiser [248] is applied to a softmax cross entropy loss function with early stopping based on generalisation error and training loss stall [249].

To evolve both architecture and neuron nominations requires that the evolved neural networks' Tensorflow models be capable of zeroing and/or freezing individual synaptic connections while allowing others to train; this does not refer to control over whole tensors but rather to entries within them. To achieve this level of control without disrupting gradient flow, matrix masks in the form of static binary tensors whose trainable flags are set to *False* (to prevent the optimiser from changing their values during training) are used. Network weights are extracted as matrix slices from the evolved weight matrix (*k8* in Figure 17) and converted into trainable tensors (referred to here as *weight tensors*). These are used to construct a binary mask in which weight tensor values are cast to integers with 1 representing an extant connection and 0 representing a dead/absent connection (referred to here as the *active connection mask*). The Tensorflow model is then defined in terms of an elementwise multiplication between the active connection masks and their corresponding weight tensors. In this way, gradient flow is defined in terms of the static active connection mask meaning that synaptic connections zeroed by evolution remain zero throughout training. The same is applied to bias values.

For neuron nomination, the aim is to freeze evolved starting weights during training. To do this, neuron nomination matrices (*k9* in Figure 17) are converted into two static masks: (1) the *nomination mask* (which indicates a trainable synaptic weight with 1 and 0 otherwise) and (2) the *inverse nomination mask* (which shows the opposite). The nomination masks are multiplied elementwise with their corresponding weight tensors (already masked for active connections) which zeroes all weight entries that are not nominated for training. The inverse nomination masks are multiplied elementwise with corresponding slices of the original evolved weight matrix (*k8* in Figure 17) which zeroes all starting weight entries that are nominated for training and preserves unnominated starting weights; the resulting tensors are referred to as *inverse nominated original weight tensors*. The weight tensors and the inverse nominated original weight tensors are then added together which fills zeroed unnominated weight entries in the weight tensors with the original weight entries determined by the evolutionary algorithm. As a result, the resultant weight tensors are composites of trainable weights and unchanging evolved weight values. The same is applied to bias values. Like before, the nomination masks are active with each training iteration and thus do not interfere with gradient flow. Neuron nomination is validated by comparing the weight tensors before and after training.

3.2.2 HyperNEAT Implementation

HyperNEAT is implemented using McIntyre et al.'s [250] Python implementation of NEAT. This implementation is designed to be extended to new projects and only requires changes to the configuration file and the genotype evaluation function. The configuration file provides control over fitness function, population, reproduction, activation function and starting topology settings for the evolving networks. To apply NEAT's evolved networks as CPPNs, the range of available activation functions is set to include absolute, clamped, cube, exponential, gaussian, hat, identity, inverse, log, relu, sigmoid, sine, softplus, square, tanh and step functions.

Sum aggregation is used throughout. The purpose of the genotype evaluation function is to assign fitness values to the genotype set. All subsequent selection, reproduction and speciation operations are then handled by the NEAT algorithm.

With the default structure implemented by McIntyre et al. [250], the CPPNs take seven inputs $([x_{from}, y_{from}, z_{from}, x_{to}, y_{to}, z_{to}, d])$ and output a synaptic weight, nomination and LEO (link expression output) value (from the work of [245]). To generate a network phenotype, the default CPPN must be activated per input for all connection and bias coordinates. Because even small networks can contain thousands of evolvable parameters, the default process (even with multiprocessing) is slow. Therefore, the CPPNs used here are reconstructed as Tensorflow graphs to activate them on all inputs simultaneously (i.e. on a tensor of dimension $[n, 7]$ where n is the total number of evolvable parameters). The outputs of the CPPNs ported to Tensorflow have been validated against McIntyre et al.’s default CPPNs.

Because CPPN apply patterns to neural connectivity but are not defined in terms of neural structure, the phenotypes built by HyperNEAT can include artefacts such as disconnected neurons and networks with no input or output connections. While removing disconnected neurons is innocuous, it is prudent to flag ‘dead’ phenotypes that are missing inputs/outputs to avoid pointless training and evaluation. Again, there are potentially thousands of parameters to check; if any neuron is deleted, all remaining neurons must be rechecked in case their connections were dependent on the deleted neuron. For this reason, a recursive neuron deletion function is implemented that targets a mutable list of all neuron indices and has two early stopping conditions: there are (1) no input or (2) no output connections. When the function targets a neuron in the list, it checks its in-degree and out-degree; if either are zero, all of the neuron’s parameters are deleted. The deleted neuron’s index is then removed from the list, and the neuron deletion function is called again on the shortened list (checking both early stopping conditions). If a network phenotype has no input or output connections prior to or after recursive neuron deletion, it is flagged, and its fitness is set to zero without further training evaluation. If a network phenotype passes neuron deletion and the connectivity checks, the connectivity constraints of Table 2 and additional descriptor metrics are calculated.

3.2.3 Fitness Function

Evolving neural networks with respect to performance, connectivity constraints and/or neuron nomination is a multi-objective optimisation problem (equation 3.14) which needs to minimise a set of objective functions $(f_k: R^n \rightarrow R \text{ where } k \geq 2)$ simultaneously; the objective functions map between the *design variable* region (S within decision variable space R^n) and the *objective region* R^k [251, p. 5].

$$\begin{array}{ll} \text{minimize} & \{f_1(x), f_2(x), f_3(x), \dots, f_k(x)\} \\ \text{subject to} & x \in S \end{array} \quad (3.14)$$

The fitness function is a significant design challenge since the objectives are incommensurable and potentially contradictory in addition to the fact that the neural

network search space is infinitely large, nondifferentiable, complex, noisy, deceptive and multimodal [125], [126]. There are four broad categories of multi-objective optimisation methods defined in terms of the role of the decision maker: *no preference* methods (which have no hierarchy of objective importance), *a posteriori* methods (which generate a set of Pareto-optimal solutions and the decision maker chooses one), *a priori* methods (which embed the preferences of the decision maker directly) and *interactive* methods (which interactively cooperate with the decision maker). While a number of multi-objective solvers within these categories exist (such as NSGA-II [252], SPEA2 [253] and MOEA/D [254]), the “No Free Lunch” theorem means the choice is ultimately problem dependant [255]. Because the neural networks are not being evolved for any other purpose than to observe trends and outcomes, a scalarisation approach is sufficient with respect to the aims. Validating the techniques across different solvers is left to future research.

Standard linear weighting techniques (equation 3.15, with $w_i \geq 0$ for all $i = 1, \dots, k$ and $\sum_{i=1}^k w_i = 1$) scalarise multiple objectives into a single objective by applying real valued weight coefficients to each and minimising the weighted sum [251].

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^k w_i f_i(x) && (3.15) \\ & \text{subject to} && x \in S \end{aligned}$$

The fitness function (equation 3.16) deployed within HyperNEAT is a modified linear weighting technique which makes it an *a priori* method.

$$\begin{aligned} & \text{minimize} && PO(x)_{norm} + \sum_{i=1}^k \chi_{SO}(x) && (3.16) \\ & \text{subject to} && x \in S \end{aligned}$$

$$\begin{aligned} & \text{where} \\ \chi_{SO}(x) = & \begin{cases} \frac{1}{k} \sum_{i=1}^k (SO_i(x))_{norm} & \text{if } \frac{1}{k} \sum_{i=1}^k (SO_i(x))_{norm} < (PO_i(x))_{norm} \\ (PO_i(x))_{norm} & \text{if } \frac{1}{k} \sum_{i=1}^k (SO_i(x))_{norm} > (PO_i(x))_{norm} \end{cases} \end{aligned}$$

Since a network’s ultimate functionality takes precedence over its topological characteristics, the primary objective ($PO(x)$, equation 3.16) is learning task performance, and the secondary objectives ($SO(x)$, equation 3.16) are local connection costs, global connection costs, input competition, neuron per connection variance and neuron nominations. If standard real-valued weighting were used, the weight coefficient of the primary objective would have to be higher than the sum of all other weight coefficients to achieve strong selection preference. For more than two secondary objectives (such as IC and CPN_{var} with neuron nomination minimisation), the secondary weight coefficients become correspondingly small, and expressed variation is artificially attenuated. Instead of applying real valued weight coefficients, the objectives are implicitly weighted by normalising each

objective function value with respect to all candidates in the population and summing all objective values but capping the secondary objectives' contribution to the value of the primary objective. Objective function values are normalised to make them comparable between objectives while preserving inter-objective magnitude differences. (Standard min-max normalisation maps the minimum to 0 and the maximum to 1; for GIC (equation 3.10), CPN_{var} (equation 3.8) is normalised with respect to the minimum.) For multiple secondary objectives, the mean of their normalised values is capped to the normalised primary objective value; this means that while there is a strong selection preference for the primary objective, there is no preference hierarchy among the secondary objectives. The effect is that candidates compete in terms of the topological features within brackets defined by their performance on the learning task. The best candidate is a neural network that topologically outcompetes other candidates in the highest performing bracket. This fitness function is algorithmically implemented as a sorting function.

3.2.4 Quantifying Structural Modularity

The method used to quantify structural modularity is an implementation of Newman and Leicht's Q-score for directed networks [256], [257]. The Q-score method compares the number of edges arranged in community structures versus the number of edges that would occur at random. The method allocates nodes to modules by successively bisecting a neural network's connectivity matrix according to the signs of the eigenvector that corresponds to the largest positive eigenvalue of a characteristic matrix called the modularity matrix (with symmetry adjustments to accommodate directed edges). Large, positive Q-scores indicate higher levels of modularity. Applied to the example networks in Figure 7 (pg. 18) and Figure 8 (pg. 18), the Q-score is higher for the modular network, and neurons are correctly allocated to modules that reflect the structural division (Figure 23).

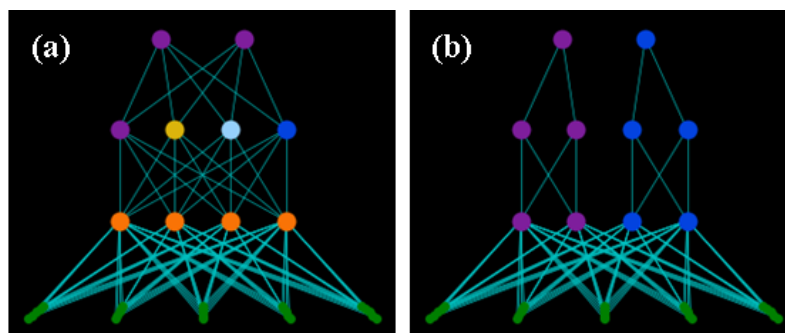


Figure 23. Q-scores and allocated modules in the (a) fully connected network $Q_H=0$, $Q_{HO}=0$, $Q_{ALL}=0.078$ and (b) modular neural network $Q_H=1$, $Q_{HO}=1$, $Q_{ALL}=0.087$. Input neurons shown in green.

Because neural networks (with or without supralayer connections) are hierarchical, their structure is not fully random; consequently, fully connected networks have nonzero Q-scores. Therefore, all reported Q-scores are normalised with respect to

fully connected networks of equivalent dimensions. With normalisation, a network must be more modular than a fully connected network to achieve a nonzero Q-score. In addition, final Q-scores and node allocations are dependent on the initial bisection of the connectivity matrix. Scale disparities between the number of input neurons versus hidden or output neurons can affect the initial bisection and skew Q values to predominantly reflect the structure of the input to first hidden layer neurons. Since the substrate restricts connections to the input neurons, the structure of the hidden and output neurons is arguably more interesting. To achieve a more holistic description, all reported Q-scores are the sum of three (normalised) Q-scores that result from the method being applied to (1) the hidden neurons alone, (2) the hidden and output neurons alone and finally (3) all neurons.

3.2.5 Learning Task Design

With the learning tasks, the aim is not to solve specific problems but rather to support the investigation of neural modularity without introducing extraneous factors. Therefore, the learning tasks are not complex, and ultimate task performance is not benchmarked for generalisation apart from a validation set. The learning tasks are designed to be computationally quick, with a modular solution decomposition and to incorporate two forms of problem modularity: inputs that are geometrically separable and inseparable respectively. Like prior work [244], [245], the geometrically separable learning task is a modified retina problem that involves classifying two arbitrary patterns with an XOR hierarchical component. On the retina, Pattern A appears exclusively on the left while Pattern B appears exclusively on the right. The problem is structured so that if either Pattern A or Pattern B appear, the output is true. If both or neither appear, the output is false. Therefore, the task can be solved modularly with functionally and spatially independent pattern detectors for left and right with a higher XOR integrator that constructs the output (Figure 24, a). In the inseparable learning task, the patterns are not presented in adjacent retina regions and therefore occupy the same space. When Pattern A or B appears, the output is true; when neither appear the output is false. The nonseparable task can be solved by two functionally modular pattern detectors which have overlapping inputs (Figure 24, b). As shown in Figure 25, the chosen patterns are a triangle and T shape in any 90° orientation with random noise as the null pattern. All experiments use 1000 training samples and 500 validation samples.

Since these two tasks are artificial with known solution decompositions, a supplementary neuroimaging dataset is used to characterise the evolution of neural modularity on real-world data with an unknown solution decomposition. The clinical neuroimaging dataset [258]–[260] consists the baseline assessments of 148 first-episode schizophrenia patients and 171 matching controls including clinical assessments of positive and negative symptoms, a childhood trauma assessment, substance abuse and cognitive assessment. The neuroimaging dataset consists of subcortical volume and cortical thickness values derived using the Freesurfer toolbox [261]–[268]. Like the geometrically separable and inseparable tasks, this task is framed as a *True* or *False* classification problem (in this case, of patients diagnosed with schizophrenia versus controls).

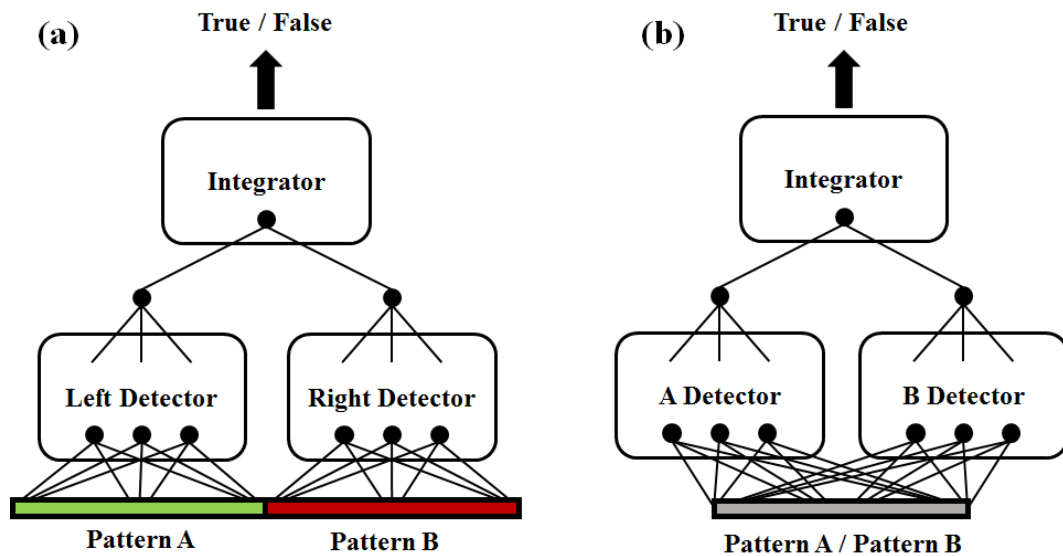


Figure 24. The (a) geometrically separable and (b) inseparable learning tasks.

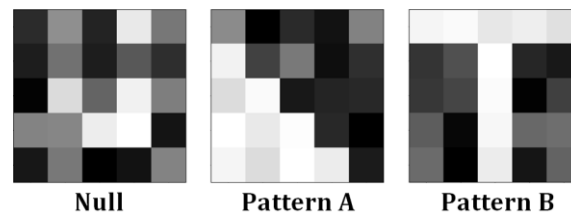


Figure 25. Patterns in the geometrically separable and inseparable tasks.

3.2.6 Program Overview

Shown in Figure 26 (a), the main program constituting HyperNEAT integrates the substrate, Tensorflow models, training, CPPN, fitness selection and modularity quantification subfunctions with McIntyre et al.'s NEAT implementation with two nested loops. The outer loop reruns HyperNEAT for an arbitrary number of sessions on a specific learning task; for each new session, NEAT's population instance is reset with new random candidates. The inner loop runs the NEAT genotype evaluation loop per generation. Within the NEAT genotype evaluation loop, the CPPNs corresponding to all genotypes are built and ported to Tensorflow to construct the network phenotypes. The descriptor metrics of all network phenotypes are calculated (with/without training on the selected learning task in Tensorflow but always pre-emptively checked for broken connectivity) and appended (together with accuracy and the genotypes' ID) to an evaluations list. Using the evaluations list, candidate fitness scores are calculated using the fitness function and assigned to each corresponding genotype. The best candidate's Q-score is calculated, and its architecture is saved. Finally, the best candidate's descriptor metrics (together with population averages) are saved to the session's convergence data file. Depending on the current generation, the NEAT genotype evaluation loop is then rerun.

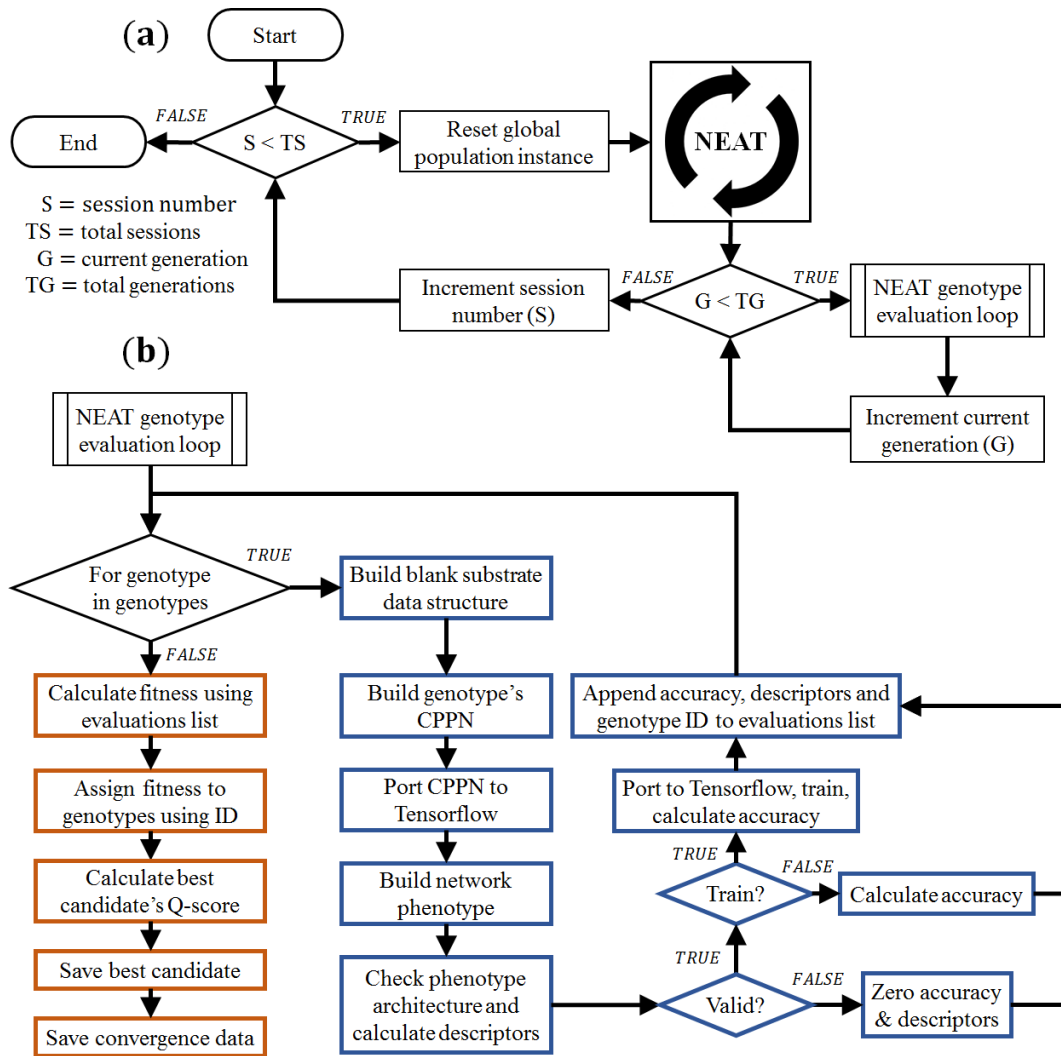


Figure 26. HyperNEAT's (a) main and (b) NEAT genotype evaluation loops.

3.2.7 Computational Speed Considerations

Even with Tensorflow-based phenotype generation and training, parallelisation is necessary to achieve results in a realistic timeframe. Therefore, the main program is adapted to run on the Centre for High Performance Computing Rosebank, Cape Town, Sun Intel Lengau cluster (<http://www.chpc.ac.za/>) where NEAT, the fitness function and data saving operations (orange in Figure 26, b) run on a primary processor and all genotype evaluations (blue in Figure 26, b) are distributed across multiple secondary processors. Parallelising these reduces the computational time per generation to the time needed to evaluate one genotype. To evaluate one genotype, the following operations occur in series: (1) a blank substrate data structure is built, (2) the genotype's CPPN is constructed, (3) the CPPN is ported to Tensorflow, (4) the Tensorflow CPPN builds the network phenotype using the substrate, (5) connectivity is checked and descriptor metrics are calculated, (6) the

network phenotype is ported to Tensorflow and (7) trained if required. Beyond standard algorithmic optimisation, the computational time of operations 1-6 is dependent on the size of the evolved networks while that of operation 7 is dependent of the number of training epochs. Because the learning tasks are simple and the interest is in topological properties, the evolved neural networks do not need to be large. Therefore, the substrate is set to describe 25 hidden neurons in 5 layers with 200 training epochs (having confirmed that randomly initialised networks of this scale can achieve 90-100% accuracy on the learning tasks). Evolution is likewise limited to 1000 generations. The assumption is that while highly refined solutions are unlikely to arise within 1000 generations, enough architectural evolution will occur to characterise the connectivity constraints, the HyperNEAT modification and the learning scenarios.

3.2.8 Statistics and Analysis

The goal is to characterise how the connectivity constraints, the HyperNEAT modification and learning scenarios influence how neural modularity evolves on a general substrate. Each experiment consists of 30 runs, and the evolution of neural modularity is studied in terms of the median values, convergence curves (averaged across all runs) and statistical differences of the evolved networks' Q-scores and task performances at generation 1000. The null hypothesis that there are no statistically significant differences between any two distributions is tested with a two-tailed Mann-Whitney U-Test with an alpha level of 0.05 where $n_1 = n_2 = 30$ (from the number of runs). Given the number of comparisons being made, *P*-values are not shown as asterisks in the presented graphs; rather, statistical differences are discussed in text and provided in the experiments' corresponding tables in the appendices. For experiment 3 where the constraints undergo multiple interaction conditions, a repeated measures ANOVA is used with an alpha level of 0.05. Trends are analysed using the Mann-Kendall Test with an alpha level of 0.05. Outliers are not removed because rare/anomalous topologies are relevant. The Q-scores presented here are calculated differently and therefore not comparable to prior work. The resultant neural modularity's interpretable qualities are assessed in terms of functional module overlaps and recovered accuracies in a similar manner.

Since Q-scores only measure divisibility, it is also necessary to study the evolved neural modularity's topological structure to gain insight into what form the modularity takes. Therefore, final network topology motifs – identified visually – that reappear across runs are studied to understand what topologies the various modularity incentives favour. While visual identification is subjective, more objective comparative measures (such as Euclidean distance, mean squared error, graph metrics, etc.) typically obscure one characteristic with another and are not rotationally or translationally invariant. In addition, CPPN comparisons are inadequate given the competing conventions problem. For this reason, a subjective visual approach is considered an adequate but strictly supplementary analysis tool.

3.3 Experiments

3.3.1 Experiment 1: Connectivity Constraints

Experiment 1 is about characterising how the connectivity constraints influence the evolution of neural modularity across four scenarios: (1) the geometrically separable learning task, (2) the geometrically inseparable learning task, (3) without supralayer connections (using the geometrically separable learning task) and (4) the supplementary neuroimaging dataset. Each category is run 30 times for 1000 generations each per connectivity constraint except the neuroimaging dataset which is run only 3 times of 1000 generations (given its computational time). Since 3 runs do not support significance testing or motif analysis, the neuroimaging dataset is a supplementary check to confirm that the evolution of neural modularity extends to real-world data and an unknown solution decomposition.

3.3.2 Experiment 2: CPPN Disjoints

Experiment 2 investigates whether CPPN disjoints benefit the evolution of neural modularity. To this purpose, each connectivity constraint is run with the CPPN settings described in section 3.1.3 for 30 runs of 1000 generations each on the geometrically separable learning task.

3.3.3 Experiment 3: Lifetime and Evolutionary Learning

Experiment 3 is about examining how the interaction between lifetime and evolutionary learning influences the evolution of neural modularity by imposing various levels of the Baldwin effect using neuron nomination. Each connectivity constraint is run for 30 runs of 1000 generations each (on the geometrically separable task) within the four interaction categories of Table 4: (1) none, (2) weak implicit, (3) weak explicit and (4) strong explicit. Since the experimental conditions of the second category (weak implicit) match that of experiment 1, that data is reused. Here, the use of only 1000 generations is relevant in the no interaction category, because it is unlikely that the task performance of networks that evolve without training will match those with training. Since this is likely to influence the evolution of neural modularity, the question is whether to run until comparable task performance is achieved or to restrict the analysis to the 1000 generations window. Given the computational expense and uncertainty of the former, the latter is chosen.

3.3.4 Experiment 4: Interpretability Assessment

Experiment 4 is about characterising the interpretable qualities of the neural modularity generated by the different constraints on the geometrically separable and inseparable task. This is evaluated in terms of functional module overlap and recovered accuracies (the mean of the *True* class functional module, the *False* class functional module and the network's CFN). The purpose is only to characterise which constraints deliver less overlap since perfect divisions between the *True* and *False* functional modules are unlikely to arise within 1000 generations

4 Results

4.1 Experiment 1

4.1.1 Geometrically Separable Task

Both the connection cost (LCC and GCC) and input competition (GIC, FIC and EIC) constraints successfully promote the evolution of neural modularity with the geometrically separable task. Final Q-score and task performance comparisons for each constraint are shown in Figure 27 and tabulated (with Mann-Whitney U-Test P -values) in Table 5 of Appendix B.1; GIC delivers the highest levels of modularity ($Q = 1.016 [0.734, 1.073]$). Ranked from lowest to highest in terms of their median Q-scores, the order is GCC, LCC, EIC, FIC and GIC. With the exception of FIC and EIC ($U = 333, P = 0.554$), there are statistically significant differences between the Q-score distributions of all constraints indicating that each have a unique influence on the evolution of neural modularity. The lack of statistically significant differences between FIC and EIC implies that minimising connections per neuron variance (in EIC) does not provide a unique benefit to the evolution of neural modularity within 1000 generations with the geometrically separable task. On the other hand, maximising the same variance measure in GIC does produce a statistically different outcome. Except GCC (87.4%), all constraints achieve above 90% median task performance with GIC at 93.4% suggesting that modularity benefits task performance.

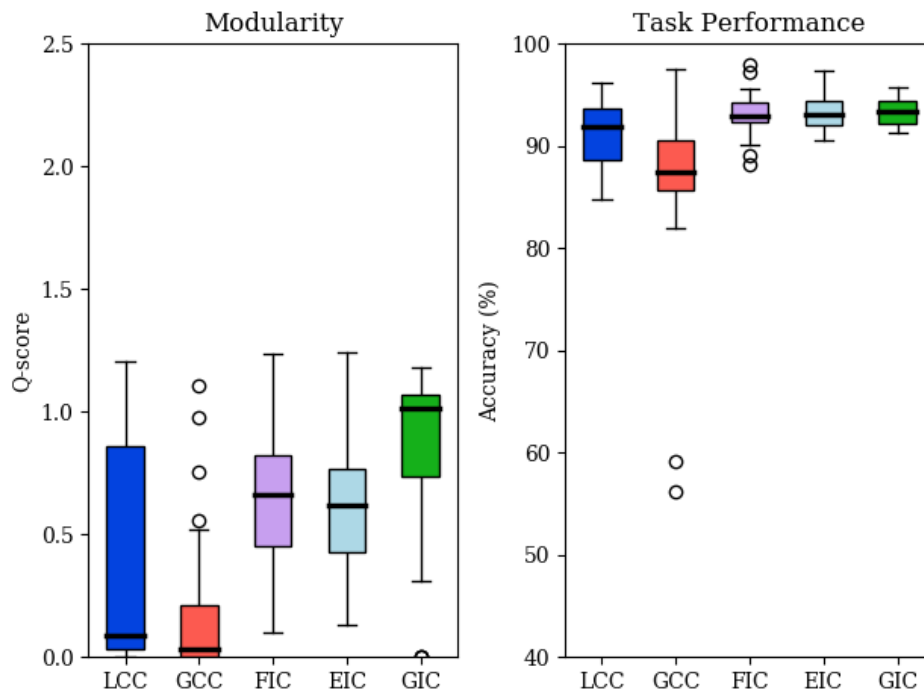


Figure 27. Comparison of Q-scores and task performance per connectivity constraint for Experiment 1 (with the geometrically separable task).

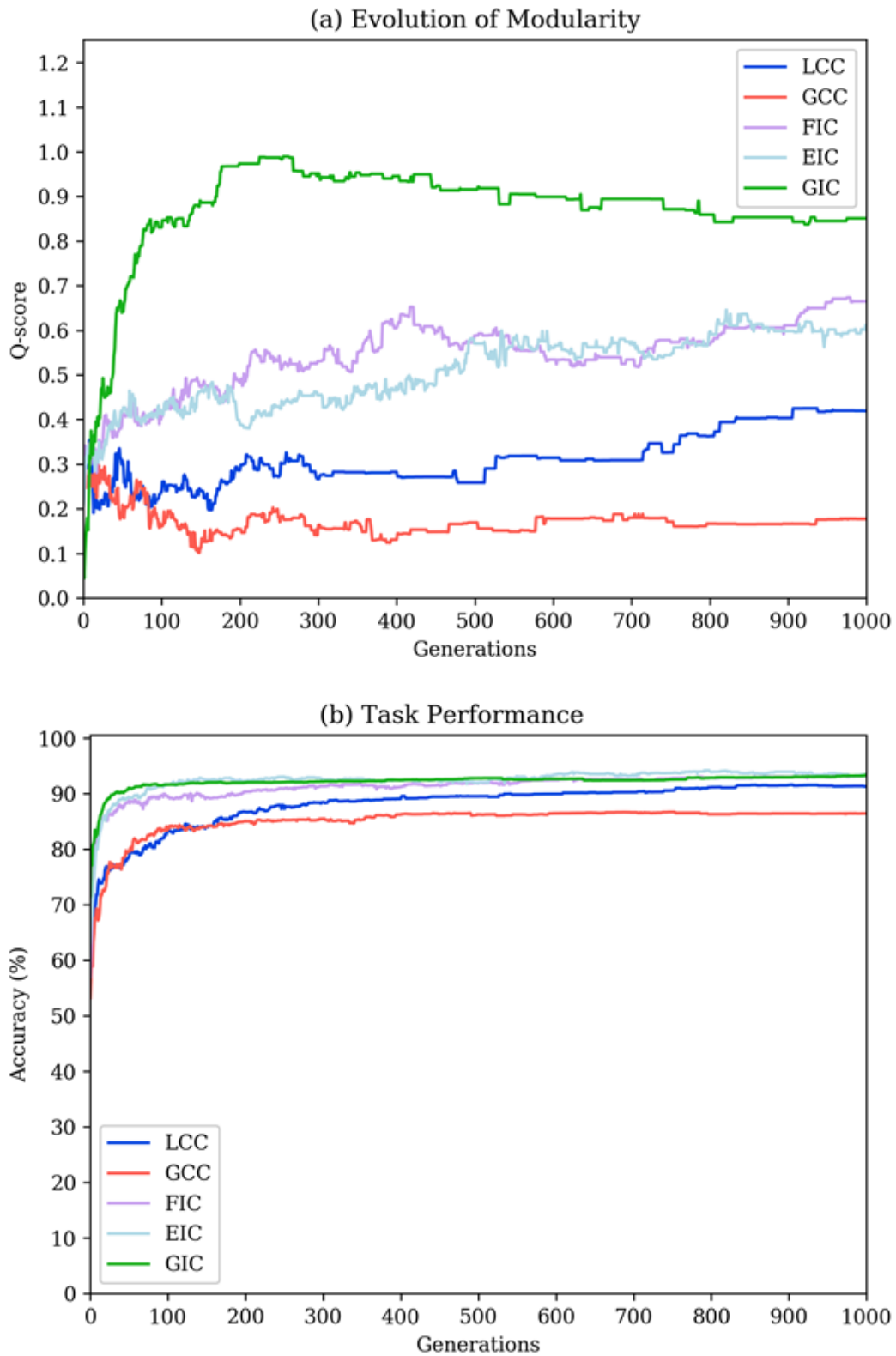


Figure 28. Convergence curves per connectivity constraint showing the evolution of (a) modularity and (b) task performance in Experiment 1 (with the geometrically separable task).

Examining the Q-score convergence curves shown in Figure 28 (a), Q-scores for GIC peak at 300 generations and slope downward. Q-scores for GCC decline throughout but increase steadily for LCC, FIC and EIC. FIC and EIC's convergence curves intersect often and follow essentially the same trajectory. The task performance curves for all connectivity constraints (Figure 28, b) stabilise between 85%-95% at 200 generations and stagnate (except for LCC) until 1000 generations.

Final network topologies (with motifs labelled) are shown in Figure 42 to Figure 51 of Appendix B.1. Common and noteworthy network motifs are shown here in Figure 29. The prevalence of the 'tower' (Figure 29, c), 'messy' (Figure 29, d) and 'collapsed' (not depicted in Figure 29) motifs (which collectively account for 56.7% and 86.6% of LCC and GCC's respective runs) reveals that LCC and GCC appear to have an atrophic rather than modularising effect: neurons are few (the 'collapsed' motif) or placed close together (the 'messy' and 'tower' motifs) which suggests an emphasis on geometric minimisation rather than topological modularity. While LCC's ability to generate modular topologies is evidenced by its interquartile range, two 'sparse' motifs (Figure 29, a) and certain 'dense' motifs (Figure 29, b), it does not generate a dominant motif. GCC delivers lower modularity than LCC with several outliers, favouring small and/or irregular networks (with the 'collapsed' and 'messy' motifs accounting for 56.6% of runs). By contrast, GIC generates the modular 'block' motif (Figure 29, c) in 83.3% of runs that (apart from its supralayer connections) appears to contain two independent pathways that visually resemble the geometrically separable solution decomposition shown in Figure 24 (a). FIC and EIC generate modular 'dense' motifs in 53.3% and 100% of runs more successfully than either LCC or GCC.

The assumption that supports connection costs as a modularity incentive is that modular networks are also those with short connection distances. The use of preconditioned substrates like those of Clune et al. [204] and Huizinga et al. [244] makes the association between energy conserving networks and modular networks artificially true. In comparison, free neuron expression and placement means that short connection distances can be achieved by moving the expressed neurons closer together and/or minimising the total number of neurons all together. This successfully satisfies the connection distance costs while omitting a modular outcome. By contrast, the input competition connectivity constraints appear to succeed in generating modular outcomes because they specify a mode of construction (i.e. the heuristic: minimise shared connections) rather than atrophying and shrinking the expressed networks.

Comparing FIC and EIC in terms of their recurring motifs, the lack of statistically significant differences between them is evident in that both favour the 'dense' motif. While EIC exclusively generates the 'dense' motif, FIC demonstrates more variability with the 'dense' motif appearing in 53.3% of runs, the 'messy' motif appearing in 40% of runs and the 'half' motif appearing in 6.7% of runs. Since EIC both scores a lower median Q-score than FIC and fixates on a single motif, it appears that minimising connections per neuron variance (which favours the expression of all connections) while simultaneously minimising the overlap

between connections is suboptimal for the evolution of neural modularity. In other words, having to navigate a constraint which simultaneously favours the expression of all connections while allocating them equally without any overlaps may be too restrictive and in conflict with modular principles. In contrast, GIC's emphasis on maximising connections per neuron variance (i.e. maximising the disparities between the number of connections per neuron) while simultaneously minimising overlap between connections appears to benefit neural modularity on this task.

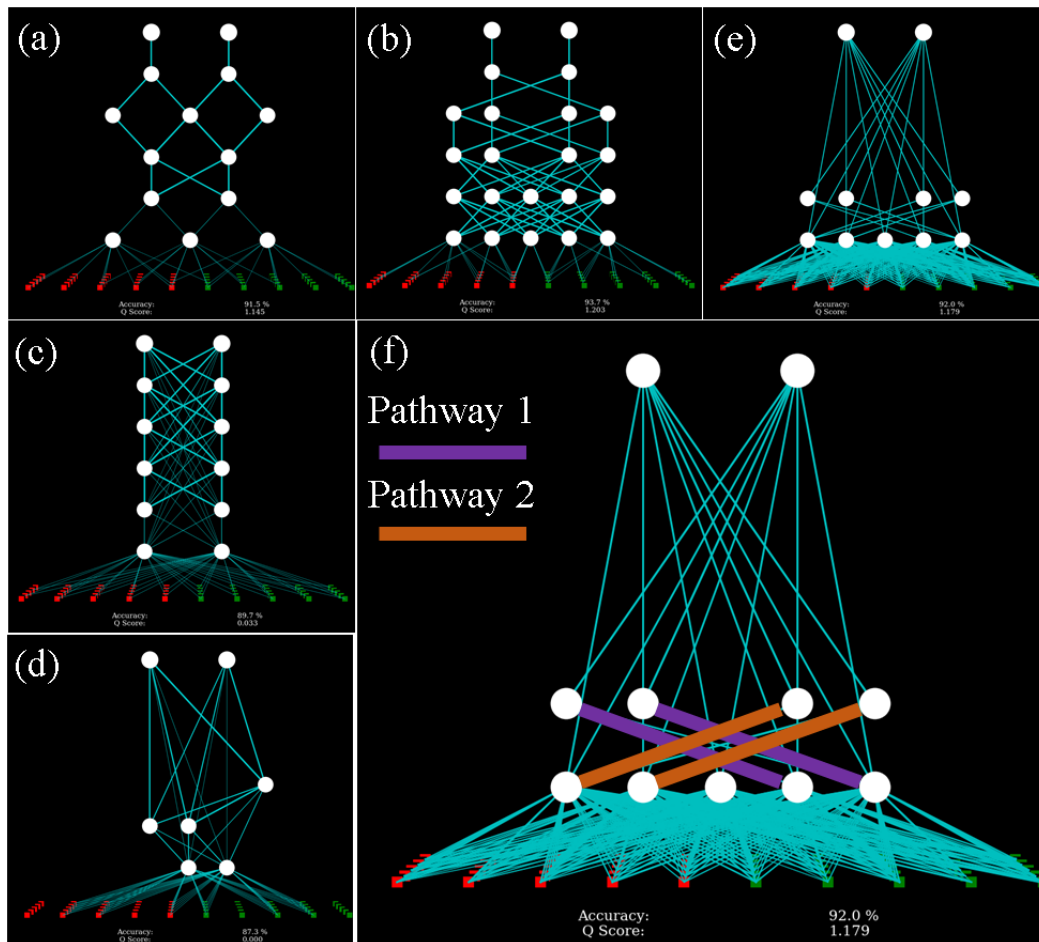


Figure 29. LCC generates the ‘sparse’ motif (a) in 6.7% of runs, the ‘dense’ motif – the most notable shown in (b) – in 36.7% of runs, the ‘tower’ motif (c) in 30% of runs and the ‘messy’ motif (d) in 16.7% of runs. GIC generates the ‘block’ motif (e) in 83.3% of runs which showcases split connectivity (f).

4.1.2 Geometrically Inseparable Task

Like the geometrically separable task, both the connection cost (LCC and GCC) and input competition (FIC, EIC and GIC) constraints successfully promote the evolution of neural modularity with the geometrically inseparable task indicating that the constraints are effective regardless of whether the problem’s modularity is

geometrically apparent or not. Q-score and task performance comparisons for each constraint are shown in Figure 30 and tabulated (together with Mann-Whitney U-Test P -values) in Table 6 of Appendix B.2; LCC ($Q = 1.131 [0.613, 1.330]$) and GCC ($Q = 1.131 [0.953, 1.324]$) tie for the highest while FIC ($Q = 0.675 [0.501, 1.092]$) delivers the lowest levels of neural modularity. Ranked from lowest to highest median Q-scores, the order is FIC, GIC, EIC, LCC and GCC. Collectively, the input competition constraints deliver higher task performances (FIC at 98.4%, EIC at 98.3% and GIC at 96.9%) than the connection cost constraints (LCC at 96.9% and GCC at 97.1%). Unlike the geometrically separable task, there are fewer statistically significant differences between the constraint types. GIC is the only input competition constraint that is statistically different from the connection cost constraints (with GIC and LCC at $U = 290, P = 0.014$, GIC and GCC at $U = 165, P < 0.001$) and EIC ($U = 279, P = 0.009$), although it is not statistically different from FIC ($U = 353, P = 0.107$). Unlike the previous experiment, the Q-score distributions generated by FIC and EIC on the geometrically inseparable task are statistically different ($U = 320, P = 0.027$). These results indicate that while the constraints can successfully drive the evolution of neural modularity on both the geometrically separable and inseparable tasks, the form of that modularity is dependent on the problem's solution decomposition.

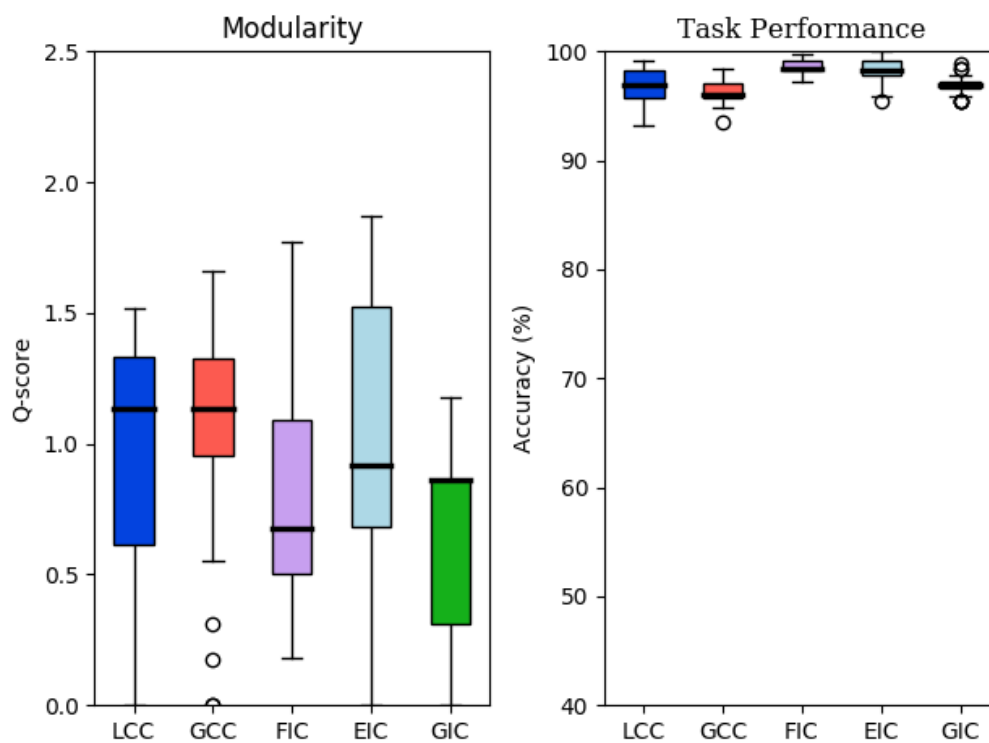


Figure 30. Comparison of Q-scores and task performance per connectivity constraint for Experiment 1 (with the geometrically inseparable task).

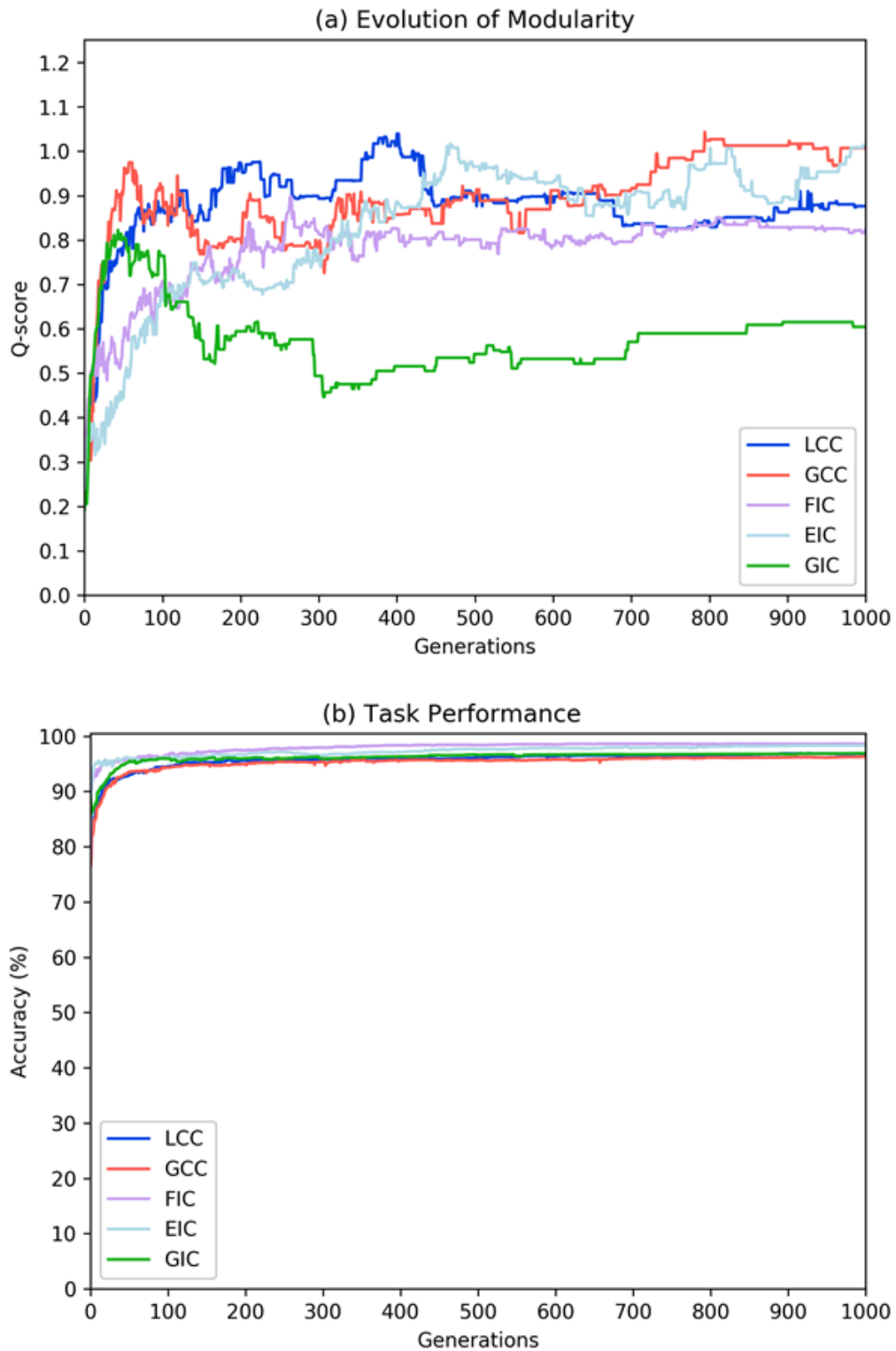


Figure 31. Convergence curves per connectivity constraint showing the evolution of (a) modularity and (b) task performance in Experiment 1 (with the geometrically inseparable task).

The Q-score convergence curves shown in Figure 31 (a) for LCC, GCC, FIC and EIC increase steadily while GIC declines before rising again. The task performance curves (Figure 31, b) for all constraints reaches 90%-100% at 100 generations and subsequently plateaus.

Final network topologies (with motifs labelled) are shown in Figure 52 to Figure 61 of Appendix B.2. Common and noteworthy network motifs are shown here in Figure 32. LCC, GCC and GIC generate a dominant ‘collapsed’ motif in 50%, 76.7% and 93.3% of runs respectively; the difference being that LCC and GCC’s higher modularity levels appear to be influenced by parcellation on the input space that would not overtly contribute to the problem solution (since the learnt patterns for the geometrically inseparable task are presented across the entire input space). In contrast, GIC discovers the same topologies without splitting the input space. That being said, LCC and GCC’s parcellation effects also exhibit of their emphasis on locality – a useful property not directly described by the formulation of the input competition constraints. While LCC, GCC and GIC favour ‘collapsed’ motifs when applied to the geometrically inseparable task, FIC and EIC favour the ‘dense’ motifs as shown in Figure 32 (d), (e) and (f) in 23.3% and 73.3% of runs respectively.

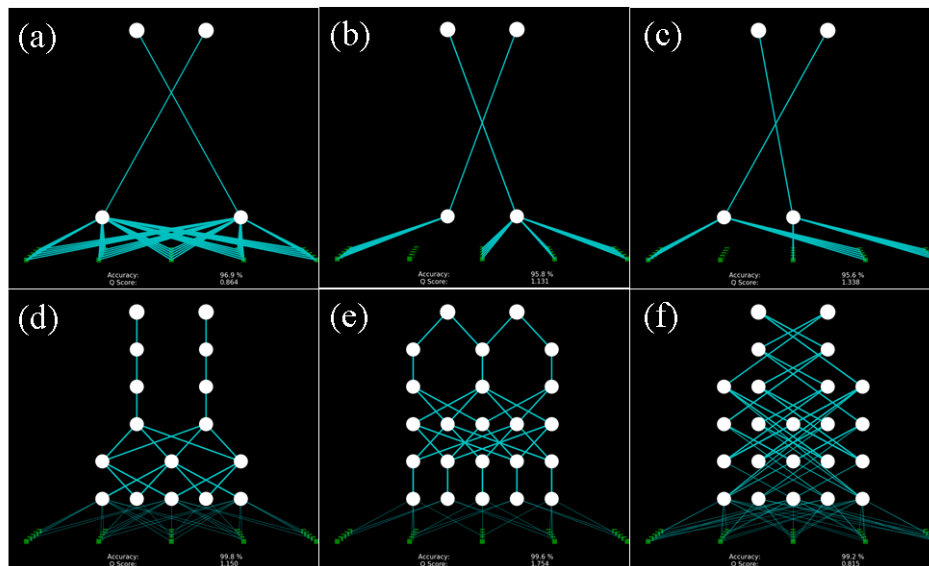


Figure 32. GIC (a) generates ‘collapsed’ motifs like LCC (b) (c) but without input parcellation. FIC and EIC generate notable ‘dense’ motifs (d) (e) (f).

4.1.3 Without Supralayer Connections

Without supralayer connections, all connectivity constraints deliver extant but lower levels of neural modularity compared to equivalent runs with supralayer connections on the geometrically separable task. All interquartile Q-score ranges are smaller (with several outliers) indicating that the topological flexibility afforded by supralayer connections provides greater evolutionary freedom to discover modular outcomes. Q-score and task performance comparisons for each constraint

are shown in Figure 33 and tabulated (with Mann-Whitney U-Test P -values) in Table 7 of Appendix B.3. The input competition constraints (except GIC) outperform the connection cost constraints with FIC ($Q = 0.296$ [0.157,0.536]) and EIC ($Q = 0.388$ [0.197,0.519]) generating the highest modularity levels while LCC ($Q = 0.095$ [0.091,0.248]) and GCC ($Q = 0.095$ [0.095,0.242]) tie with GIC ($Q = 0.095$ [0.023,0.195]) as the lowest. Ranked from lowest to highest in terms of their median Q -scores, the order is LCC-GCC-GIC (tied), FIC and EIC. FIC, EIC and GIC reach 90% median task performance while LCC and GCC peak at 88.5% and 87.1% respectively. The connection cost constraints are statistically different from the input competition constraints with respect to FIC and EIC but not GIC. There are no statistically significant differences between the Q -score distributions of LCC and GCC ($U = 402, P = 0.308$), LCC and GIC ($U = 399, P = 0.223$) or GCC and GIC ($U = 357, P = 0.118$). Like the geometrically separable task with supralayer connections, there are no statistically significant differences between FIC and EIC's Q -score distributions ($U = 400, P = 0.462$).

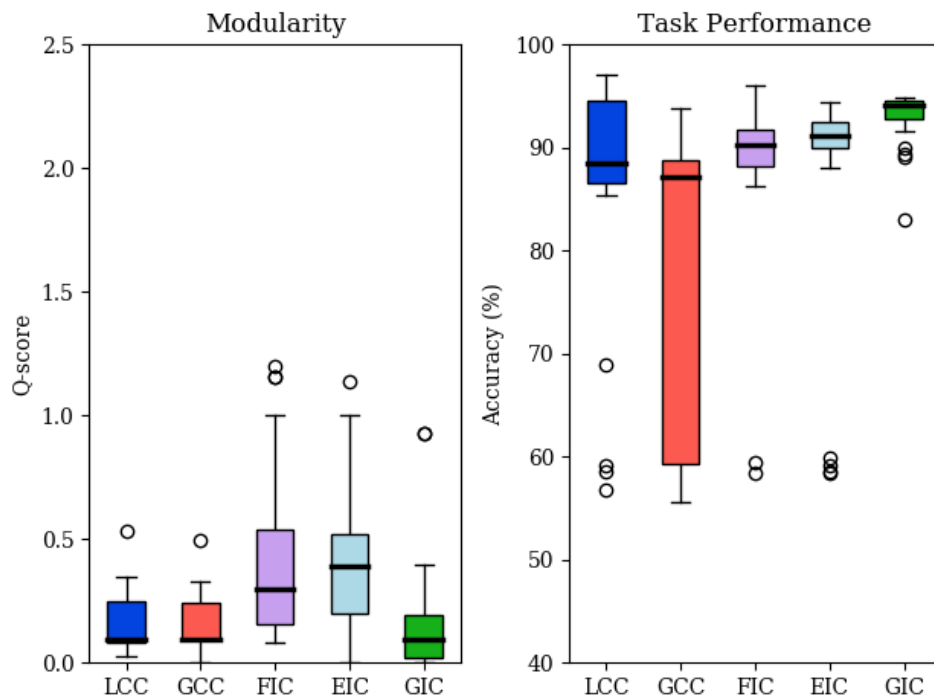


Figure 33. Comparison of Q -scores and task performance per connectivity constraint for Experiment 1 (with the geometrically inseparable task, without supralayer connections).

Examining the averaged Q -score convergence curves (Figure 34, a), LCC and GCC plateau while FIC, EIC (both after initial fluctuations) and GIC increase steadily until 1000 generations. The averaged task performance curves (Figure 34, b) show that GIC alone reaches 90% performance within 150 generations. LCC, FIC and EIC reach 90% performance at 1000 generations with steady improvements prior. At 75% at 1000 generations, GCC has the poorest task performance.

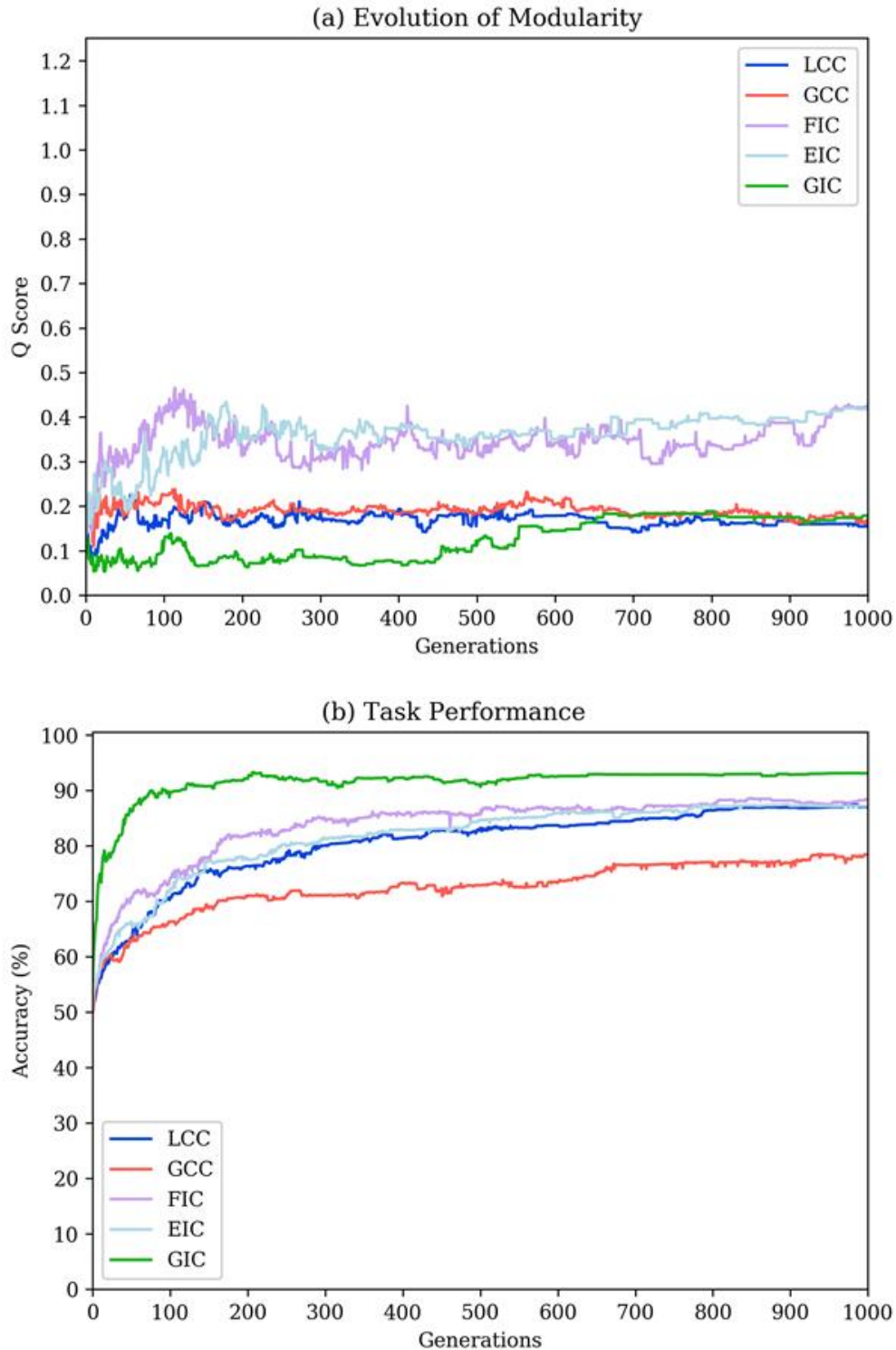


Figure 34. Convergence curves per connectivity constraint showing the evolution of (a) modularity and (b) task performance in Experiment 1 (with the geometrically separable task, without supralayer connections).

Final network topologies (with motifs labelled) are shown in Figure 62 to Figure 71 of Appendix B.3. Common and noteworthy network motifs are shown here in Figure 35. Reviewing their recurring motifs, it is clear that LCC and GCC respond poorly to the absence of supralayer connections with the ‘tower’ (Figure 35, a), ‘messy’ (not depicted) and dysfunctional ‘null’ (Figure 35, b) motifs accounting for 53.4% and 90% of runs respectively. While it is possible that this is due to the hyperparameter selection, the substrate configuration and/or insufficient evolutionary generations, atrophic node placement is so prominent that it is reasonable to assume that this is what contributes to the poor performance. GIC generates the same distribution of motifs as LCC and GCC with the ‘tower’ and ‘messy’ motifs accounting for 90% of runs indicating an overlap between high connections per neuron variance and the connection cost’s atrophic effect. By contrast, FIC and EIC’s bias toward ‘dense’ motifs allows them to generate comparatively higher neural modularity. Notably, FIC also generates a ‘pleated’ motif (Figure 35, c) occurring in 16.7% of runs which effectively splits the functional pathways responsible for the *True* and *False* output neurons.

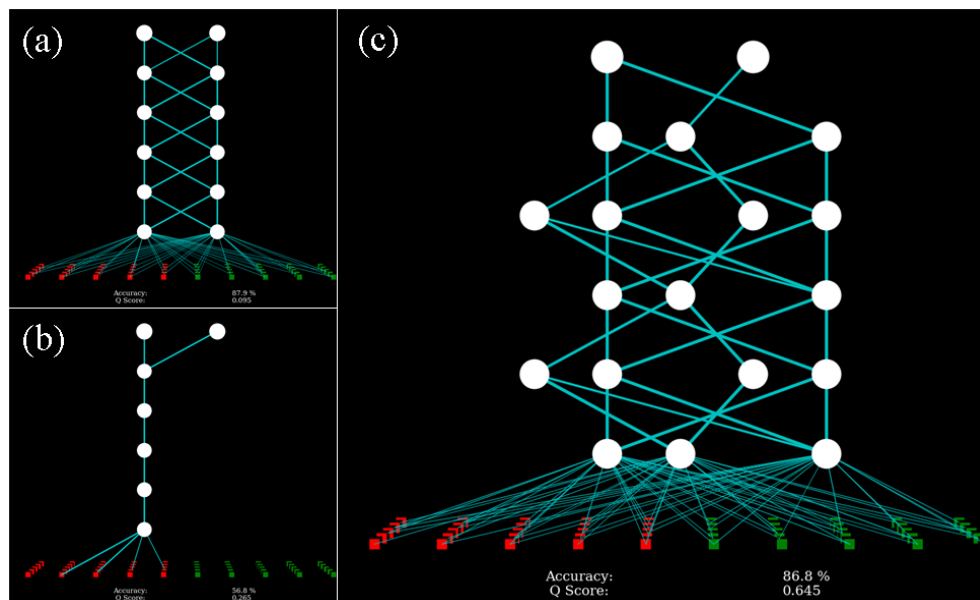


Figure 35. LCC and GCC generate the ‘tower’ motif (a) in 26.7% and 53.3% of runs respectively and the ‘null’ motif (b) in 10% and 30% of runs respectively. FIC generates the ‘pleated’ motif (c) in 16.7% of runs.

4.1.4 Supplementary Neuroimaging Dataset

Applied to the neuroimaging dataset for 3 runs each, all connectivity constraints (with the exception of GIC) successfully promote the evolution of neural modularity on the neuroimaging dataset with an unknown solution decomposition. Ranked in terms of final Q-scores at generation 1000, the order is GIC, GCC, LCC, EIC and FIC; all constraints achieve 90%-100% task performance. Examining the

averaged convergence curves shown in Figure 36, Q-scores for LCC, GCC, EIC and FIC increase steadily across 1000 generations; FIC delivers the highest levels of neural modularity. GIC fails to generate neural modularity in any of the 3 runs.

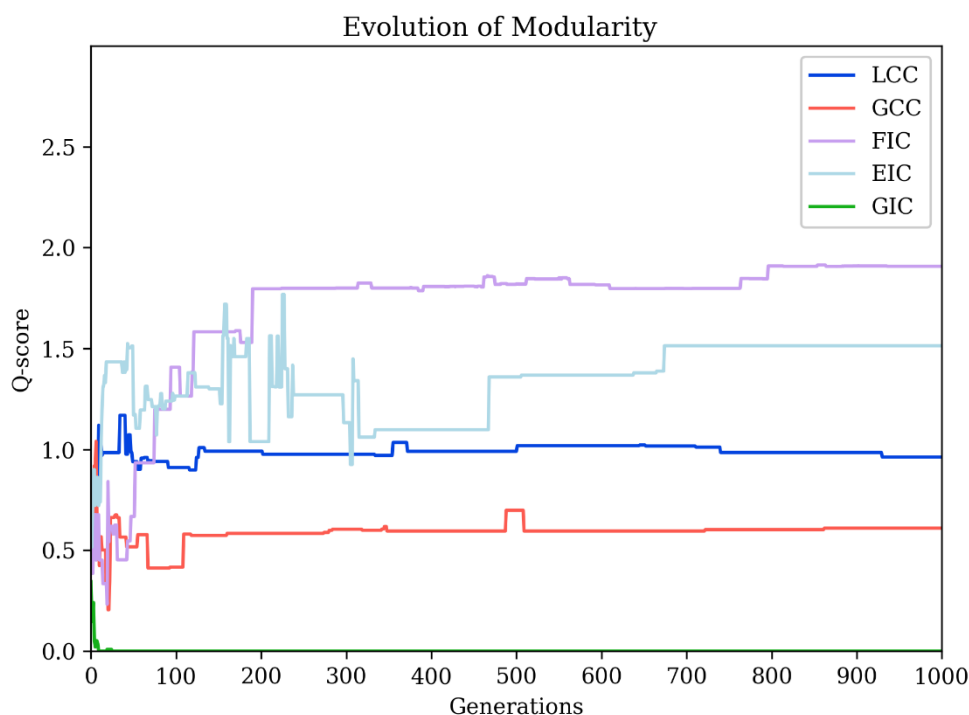


Figure 36. Convergence curves per connectivity constraint showing the evolution of (a) modularity and (b) task performance in Experiment 1 (with the neuroimaging dataset).

4.1.5 Analysis of Experiment 1: Constraint Characterisation

Evolved with training across 1000 generations on a general substrate that includes supralayer connections, both the connection cost (LCC and GCC) and input competition (FIC, EIC and GIC) constraints successfully promote the evolution of neural modularity irrespective of whether the learning task is geometrically separable, geometrically inseparable or constituted of real-world data with an unknown solution decomposition. In addition, neural modularity also evolves when supralayer connections are omitted from the substrate. While Clune et al. [204] and Huizinga et al. [244] have previously demonstrated how connectivity constraints based on connection costs can benefit the evolution of neural modularity, this work demonstrates competitive levels of evolved neural modularity for connectivity constraints based on a novel measure of input competition.

On the geometrically separable task, the connection cost and input competition constraints influence the evolutionary process in statistically different ways as evidenced by Q-score distribution comparisons and recurring motifs. However, on the geometrically inseparable task, the Q-score distributions generated by

connection cost and input competition constraints are not statistically distinguishable. This suggests that while both constraint categories are able to promote the evolution of neural modularity irrespective of tested task types, the structure of that evolved neural modularity is also related to the learning task not only the type of constraint used.

In comparison, the input competition constraints generate higher modularity levels than the connection cost constraints when applied to the geometrically separable task with and without supralayer connections but not the geometrically inseparable task. Applied to this task, LCC, GCC and GIC all generate dominant ‘collapsed’ motifs that are identical save for significant input space parcellation in the case of LCC and GCC. This parcellation effect boosts both LCC and GCC’s modularity levels but is functionally unnecessary. Without such parcellation, it is reasonable to assume that LCC and GCC’s performance would be more comparable to the input competition constraints. However, the connection cost constraints’ parcellation effects highlights a beneficial emphasis on locality that is not directly referenced by the formulation of the input competition constraints.

Of the connection cost constraints, GCC only matches LCC’s neural modularity levels indicating that applying connection costs globally is less effective than applying them locally. Both are an atrophic effect which leads to a prevalence of ‘tower’, ‘messy’ and ‘collapsed’ motifs. In contrast, LCC’s greater emphasis on locality generates more regular topologies than GCC (plus a minority of highly modular ‘sparse’ motifs). Being competitive with the input competition constraints, LCC demonstrates the modular benefit of the energy conservation principle.

Of the input competition constraints, GIC exhibits the most fluctuation; it delivers the highest, middle and lowest modularity levels in the geometrically separable, geometrically inseparable and sans supralayer connections scenarios. Apart from a dominant (and modular) ‘block’ motif with the geometrically separable task, GIC’s recurring network motifs are not very different from LCC and GCC. EIC – while competitive – does not exhibit any notable variations of its dominant ‘dense’ motif (often simply the rediscovery of a fully connected topology without supralayer connections), and its Q-score distribution is not statistically different from FIC’s on the geometrically separable task. In contrast, FIC is a strong middle contender that consistently delivers competitive neural modularity levels and generates structurally promising recurring motifs (especially the ‘pleated’ motif in the sans supralayer connections scenario). This result indicates that selection pressure on connections per neuron variance in addition to input orthogonality does not benefit the evolution of neural modularity in a way that is stable (in the case of GIC) or distinct from FIC (in the case of EIC).

Overall, the characterisation tests identify LCC and FIC as the most promising connectivity constraints. However, FIC’s ‘pleated’ motif and others raise the question of whether such split connectivity is primarily a consequence of the constraints or of the one-hot encoding which provides two output neurons to append pathways to. With training, it is clear that in all cases evolution does not proceed in

relation to learnt properties but rather selects topologies predominantly on the basis of geometry. In other words, the training algorithm tunes the output as needed while evolution searches for sufficiently expressive topologies that also satisfy the connectivity constraints. The unity neurons (with one input and one output) in LCC's 'sparse' (Figure 29, a) and 'dense' (Figure 29, b) motifs are a case in point. Such unity neurons are unlikely to emerge if there was a tight coupling between the processes that determine the form and function of the evolving networks (since unity neurons do not contribute to function but are 'excusable' by training).

4.2 Experiment 2

Modifying HyperNEAT's CPPNs to rely exclusively on step and sine functions to improve its modular variational properties with disjoints has an adverse effect on both the evolution of neural modularity and ultimate task performance. Q-score and task performance comparisons for each constraint are shown in Appendix B.4 in Figure 72 and tabulated (together with Mann-Whitney U-Test P-values) in Table 8. Applied to the geometrically separable task, EIC delivers the highest ($Q = 0.608$ [0.37, 0.96]) while GIC delivers the lowest ($Q = 0.051$ [0.0, 0.229]) levels of modularity. Overall performance is notably poor with LCC, GCC and GIC's interquartile ranges starting at zero. LCC and FIC ($U = 388, P = 0.178$), LCC and EIC ($U = 363, P = 0.098$) and GCC and GIC ($U = 429, P = 0.375$) do not generate statistically different Q-score distributions. While it is possible that additional evolutionary generations could help, the absence of upward trajectories in any of the constraints' Q-score convergence curves (Figure 73 (a) in Appendix B.4) is apparent. Task performance (Figure 73, b) is poor with only EIC reaching a 76.4% within 1000 generations. This result is significant because it suggests that HyperNEAT requires a minimum level of activation function diversity to work.

4.3 Experiment 3

Investigating the interaction between lifetime and evolutionary learning with respect to Table 4's interaction categories suggests that the concurrency between the determination of form (topology) and function (synaptic weights) has a clearer influence on the evolution of neural modularity than the Baldwin effect. Q-score comparisons for each interaction category are shown in Figure 37 and tabulated (with Mann-Whitney U-Test P-values) in Table 9 (none), Table 5 (weak implicit from experiment 1 of Appendix B.1), Table 10 (weak explicit) and Table 11 (strong explicit) of Appendix B.5. Averaged convergence curves are shown in Figure 74 (none), Figure 28 (weak implicit from experiment 1), Figure 75 (weak explicit) and Figure 76 (strong explicit) in Appendix B.5. In the no interaction category, FIC delivers the highest modularity levels ($Q = 0.905$ [0.575, 1.432]); ranked from lowest to highest in terms of median Q-scores, the order is GIC, EIC, LCC, GCC and FIC. Apart from GIC, all constraints deliver higher levels of neural modularity compared to all other interaction categories. Corresponding task performances are low but slower convergence in the absence of training is expected.

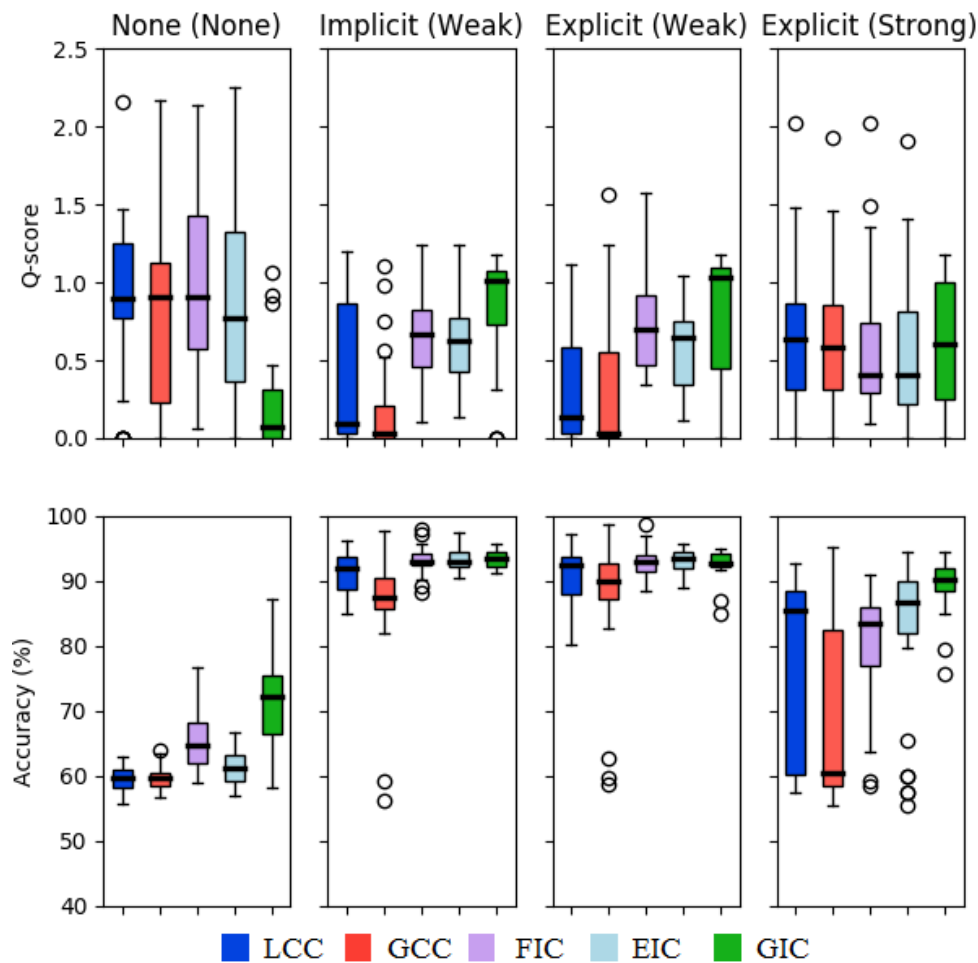


Figure 37. The interaction between lifetime and evolutionary learning.

The weak implicit and weak explicit interaction categories deliver very similar modularity levels (as shown in Figure 37) and convergence curves as shown in Figure 28 (from experiment 1) and Figure 75 (Appendix B.5). In the weak explicit case, neuron nomination levels remain high throughout as shown in Figure 77 (Appendix B.5) indicating that embedding heritable learnt characteristics does not provide an adaptive advantage within 1000 generations. In both the weak implicit ($Q = 1.016 [0.734, 1.073]$) and weak explicit case ($Q = 0.998 [0.399, 1.106]$), Ranked from lowest to highest in terms of median Q-scores, the order in both is GCC, LCC, EIC, FIC and GIC. In comparison, the constraints in both the weak implicit and weak explicit interaction categories deliver lower modularity levels than in the no interaction category with the exception of GIC.

In the strong explicit interaction category, there is minimising pressure on neuron nomination levels as shown in Figure 78 (Appendix B.5). All constraints generate similar Q-score distributions with no statistically significant differences between them. GIC delivers the highest levels of modularity ($Q = 0.669 [0.262, 0.998]$); ranked from lowest to highest in terms of median Q-scores, the order is FIC, EIC,

GCC, LCC and GIC. The connection cost constraints generate higher modularity levels than in the weak implicit and weak explicit interaction categories but lower modularity levels than in the no interaction category. The input competition constraints (except GIC) show the opposite order. Task performance is lower than the categories that include training, but slower convergence is not anomalous given that evolution determines a greater proportion of functionality.

Analysed with a repeated measures ANOVA for an increasing Baldwin effect across the five constraint groups (LCC, GCC, FIC, EIC and GIC), the Baldwin effect has a significant main effect ($F(2.665) = 6.306, P < 0.001$) as well as a significant within-groups interaction effect ($F(10.660) = 8.793, P < 0.001$) across the different constraints. Despite the significant main and interaction effect, the trend lines shown in Figure 38 (a) do not exhibit a consistent relationship between estimated marginal mean neural modularity and an increasing Baldwin effect. Mann-Kendall trend analysis identifies no trend for LCC ($z = -0.961, P = 0.337$) as well as GCC ($z = -0.0134, P = 0.989$). A decreasing trend is identified for FIC ($z = -3.121, P = 0.002$) and (marginally) for EIC ($z = -1.964, P = 0.049$). An increasing trend is identified for GIC ($z = 2.365, P = 0.018$), but this is somewhat misleading given the bell shape of its line in Figure 38 (a).

A different pattern emerges when the four interaction categories are rearranged in order of how concurrent the determination of form and function is. In the weak implicit and weak explicit categories, form (topology) and function (synaptic weights) are determined separately: form by evolution and function by training. In the strong explicit category, part of a network's function is determined concurrently with its form. With no interaction, both form and function are determined by evolution alone. Reordered according to this paradigm as shown in Figure 38 (b), higher modularity is linked to stronger concurrency between the determination of form and function for connection costs. Post-hoc testing revealed these tests survive Bonferroni correction for multiple comparisons. LCC ($z = 4.404, P < 0.001$) and GCC's ($z = 5.118, p < 0.001$) Q-score distributions show a positive trend with increasing concurrency. FIC ($z = 0.522, P = 0.602$) and EIC ($z = 0.526, P = 0.599$) show visually discernible positive trends that nonetheless are not apparent to the Mann-Kendall test. This is not true for GIC ($z = -5.665, P < 0.001$) which shows a very distinct negative trend with increasing concurrency.

These results indicate that the evolution of neural modularity (within the 1000 generation window) under connection constraints benefits more strongly from concurrent determination of form and function rather than an increasing Baldwin effect as approximated by neuron nomination under selection pressure. The same does not appear to be true for the input competition constraints. However, these results include the caveat that the no interaction category's modularity levels may decline as task performance increases with more evolutionary generations, and this warrants further study. Overall, concurrency is potentially a better explanation for these results than the Baldwin effect.

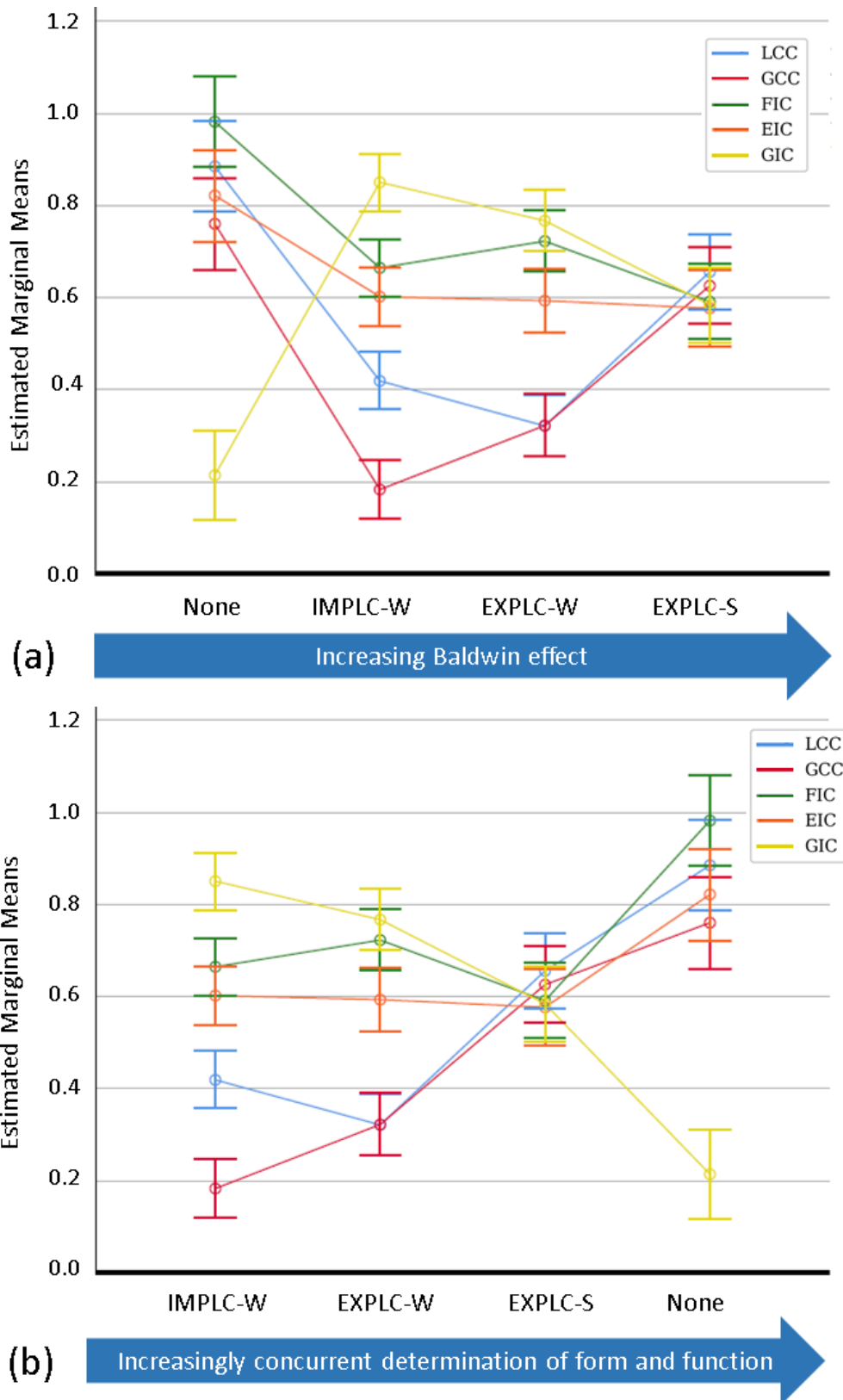


Figure 38. Repeated measures ANOVA for experiment 3.

4.4 Experiment 4

Functional module overlap and recovered accuracies are shown in Figure 39 and tabulated (with Mann-Whitney U-Test P -values) in Table 12 and Table 13 of Appendix B.6. With the geometrically separable task, the connection cost and input competition constraint categories are statistically different from each other (with LCC and FIC at $U = 291, P = 0.009$, LCC and EIC at $U = 245, P = 0.001$, LCC and GIC at $U = 291, P = 0.009$), although there are no statistically significant differences within the two categories. With the geometrically inseparable task, there are fewer statistically significant differences within and between the constraint categories in addition to more outliers. However, LCC (11.8% median overlap) and FIC (14.3% median overlap) are statistically different ($U = 336, P = 0.046$). Together, these results indicate that an evolved network's interpretable qualities (framed as the functional independence of its outputs) will vary between learning tasks. Restricting the analysis to LCC and FIC (the most promising constraints identified in experiment 1), it is clear that both have statistically different effects on both the evolving neural networks' modularity levels and their internal functional overlaps. Therefore, LCC and FIC are worth exploring in future research.

This result does include a caveat regarding SRC's performance. With the geometrically separable task, recovered accuracies are low compared to their parent networks (with LCC, GCC, FIC, EIC and GIC at 55%, 62.5%, 63.5%, 54.5% and 74.4%) but higher (with LCC, GCC, FIC, EIC and GIC at 84.6%, 83.7%, 97.8%, 67.9% and 95.9%) with the geometrically inseparable task. The observation that some recovered functional module accuracies are low is significant because any discussion about functional module overlap depends on knowing that the extracted functional modules are the right ones. Since the recovered accuracies remain above 60% for the majority of instances on both tasks, the functional modules identified here are considered representative, but it is clear that SRC struggles with certain evolved topologies. This could be a consequence of the networks' size, the one-hot encoding or the use of relu activation functions where decision boundaries can occur anywhere on the positive x -axis.

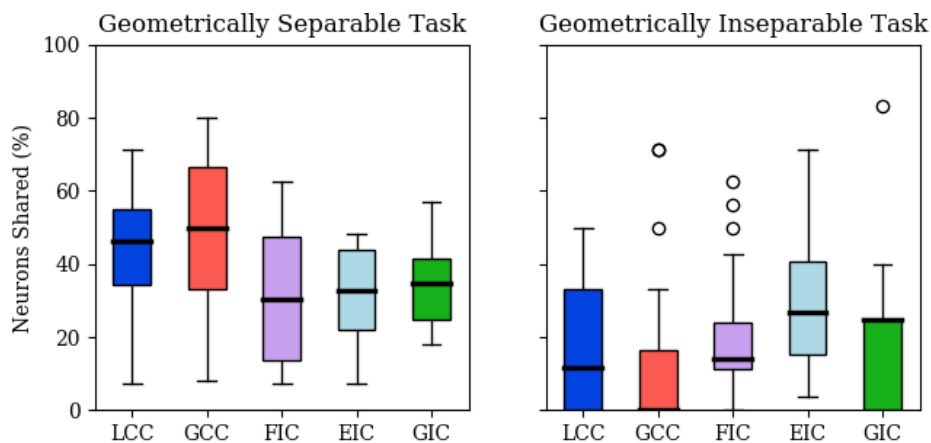


Figure 39. Interpretability assessment for Experiment 4.

5 Conclusion

Neural modularity is a useful topological property both for its own sake and its potential contribution to solving the neural network interpretability problem. This research investigates both the means of evolving neural modularity and the explanative properties of the modular networks that result. Informed by prior work [123], [204], [205], [244], the chosen neuroevolutionary method is HyperNEAT with a fitness function that selects for connectivity constraints based on connection costs (from the work of Clune et al. [204]) and input competition (a novel method which favours orthogonality between the incoming connections of all neurons). Structural modularity is quantified using the Q-score method for directed networks [256], [257], and functional modularity is quantified using subsets regression on network connectivity (SRC) [247]. Unlike Clune et al. [204] and Huizinga et al. [244], the networks are evolved with training and supralayer connections (for greater scale variation) on a general substrate rather than a preconditioned substrate.

The literature study reveals that while HyperNEAT has been widely studied, addresses the competing conventions problem, includes speciation mechanisms (to protect genetic diversity) and supports REEVE properties, there are significant provability limitations to the question of which neuroevolutionary encodings work best which derives from the fact that neuroevolution lacks a unified theoretical framework. Without a basis for comparison, any choice between the wealth of empirically feasible encodings strategies is prone to stall at the conceptual. The lack of such a unified theoretical framework informs the decision to characterise and explore the chosen neuroevolutionary method rather than to make a comparative assessment of its performance relative to other methods. As such, the experiments are structured to investigate (1) the various connectivity constraints, (2) a minor modification to HyperNEAT aimed at improving its modular variational properties with CPPN disjoints, (3) how the interaction between lifetime and evolutionary learning influences the evolution of neural modularity and (4) the explanative properties of the networks evolved under the different connectivity constraints.

The connectivity constraints include two based on connection costs – applied locally (LCC) and globally (GCC) – and three based on input competition with unconstrained (FIC), minimising (EIC) and maximising (GIC) selection pressure on connections per neuron variance. Both the connection cost and input competition constraints successfully promote the evolution of neural modularity across a geometrically separable task, a geometrically inseparable task, a real-world neuroimaging dataset and with/without supralayer connections. With the geometrically separable task, the connection cost and input competition constraints influence the evolution of neural modularity in statistically different ways. However, their effects are not statistically different when applied with the geometrically separable task which suggests that while the constraints do promote the evolution of neural modularity irrespective of task, the form of the resultant neural modularity is also task dependent. These results show that the novel input competition constraints are competitive with connection costs as a means of driving

the evolution of neural modularity. Based on their evolved modularity levels and recurring motifs (discussed hereafter), LCC and FIC are considered promising and relevant to future work investigating the evolution of neural modularity.

Of the two connection cost constraints, applying connection costs globally (GCC) is shown to be inferior to applying them locally (LCC). The analysis of recurring motifs indicates that connection costs have an atrophic effect when applied to a general substrate given the possibility of free neuron placement and expression. In other words, minimising a constraint based on connection distance can be co-opted by simply moving expressed neurons closer together and/or minimising the total number of expressed neurons – omitting a modular outcome but satisfying the constraint. Preconditioned substrates (such as those used by Clune et al. [204] and Huizinga et al. [244]) avoid this problem because they artificially associate modular topologies with those containing short connection distances. Nonetheless, the competitive modularity levels generated by LCC in addition to its strong locality principle indicates that constraining connection costs is beneficial to the evolution of neural modularity on a general substrate.

Of the input competition constraints, FIC and EIC's Q-score distributions show no statistically significant differences with the geometrically separable task. Motif analysis reveals that while FIC is capable of generating a variety of topologies, EIC is consistently unable to generate anything other than the 'dense' motif. Furthermore, while GIC generates higher modularity than FIC on the geometrically separable task, its performance fluctuates across the other tasks. These results indicate that EIC and GIC's additional selection pressure on connections per neuron variance introduces maladaptive rigidity to the range of acceptable topologies. Therefore, FIC – which does not constrain connections per neuron variance – is considered the most promising of the three input competition constraints.

To improve its modular variational properties, HyperNEAT's CPPNs were modified to use only step and sine activation functions to approximate the graded visual disjoints introduced by maximum absolute aggregation in image evolution experiments. Applied with the geometrically separable task, the resultant neural modularity levels are low and statistically indistinguishable with no upward trends in any of the Q-score convergence curves. This indicates that the disjoints expressed by CPPNs through step functions do not contribute to the evolution of neural modularity through connectivity constraints and that HyperNEAT requires a minimum level of activation function complexity to function optimally.

Including a training phase implies the need to study the interaction between lifetime and evolutionary learning given the possibility of decoupling an evolving network's form and function via the competing conventions issue. Here the Baldwin effect is approximated by pairing each neuron with an evolvable nomination parameter that dictates whether it can train or not. In this way, heritable learnt characteristics (unnominated neurons) contribute to a network's ultimate functionality, and interaction between lifetime and evolutionary learning can be investigated by applying selection pressure on the total number of nominated neurons. For the

connection cost constraints, the results link greater concurrency between the determination of form (topology) and function (synaptic weights) with higher levels of neural modularity. In other words, higher modularity levels evolve when the processes that determine a network's form (topology) and function (synaptic weights) strongly influence one another – with the caveat that evolution without lifetime learning may converge to lower neural modularity levels given more than 1000 generations. While the same does not hold for the input competition constraints, this is an intuitive result which indicates that future work should consider not pairing neuroevolution with a training phase and that highly concurrent methods of determining the form and function of a neural network such as the weight pruning approach by Jacobs and Jordan [121] should be revisited and investigated with respect to connection costs and input competition. In addition, the same experiment can be revisited but across a greater variety of learning tasks to more fully assess how sensitive the results are to task structure.

Neural modularity's contribution to interpretability is framed in terms of the overlap between a given network's internal functional modules. In other words, a network whose functional modules share fewer active neurons is considered more interpretable than one whose functional modules are more co-dependent. Assessing the networks evolved with the geometrically separable and inseparable tasks according to this criterion reveals that their interpretable qualities vary across learning tasks. However, LCC and FIC (identified as the most promising constraints in experiment 1) do generate statistically different functional overlap distributions across both tasks. However, in some cases the recovered functional module accuracies were low indicating that SRC may struggle to extract the functional modules correctly. This is potentially a consequence of the size of the networks, the use of relus (where a decision boundary can occur anywhere on the positive x -axis) or of the one-hot encoding (which required adapting SRC to work on multiple outputs instead of one). Therefore, there is room to explore not only improvements to SRC but also further work that makes minimising functional overlap an evolutionary objective. Given improved functional module extraction, the identified subnetworks can benefit data analysis as a form of hypothesis generation based on how consistently they reappear in multiple evolutionary runs.

To conclude, the results show that connection cost constraints (LCC and GCC, informed by the work of Clune et al. [204]) and the novel input competition constraints (FIC, EIC and GIC) are successful in promoting the evolution of neural modularity through HyperNEAT across a variety of tasks with/without supralayer connections on a general substrate. Neither constraint category conclusively outperforms the other across all tasks, but LCC and FIC do generate statistically different neural modularity more consistently and with better interpretable qualities than GCC, EIC or GIC. Given the aim of harnessing neural modularity's inherent advantages and its potential to contribute to solving the neural network interpretability problem, these results indicate that the energy conservation principle of connection costs and the orthogonality principle of input competition are both relevant to future work investigating the evolution of neural modularity.

6 References

- [1] P. H. Winston, *Artificial Intelligence*, Third Edit. Addison-Wesley Publishing Company, 1992.
- [2] David B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, Third Edit. Wiley-IEEE Press, 2005.
- [3] P. Wang, “What Do You Mean by ‘AI’?,” in *Proceedings of the 2008 Conference on Artificial General Intelligence 2008: Proceedings of the First AGI Conference*, 2008, pp. 362–373.
- [4] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, “A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence.” 1955.
- [5] A. M. Turing, “Computing Machinery and Intelligence,” *Mind*, vol. LIX, no. 236, pp. 433–460, 1950.
- [6] V. Akman and P. Blackburn, “Editorial: Alan Turing and Artificial Intelligence,” *J. Logic, Lang. Inf.*, vol. 9, no. 4, pp. 391–395, 2000.
- [7] R. A. Brooks, “The Seven Deadly Sins of Predicting the Future of AI.” 2017.
- [8] S. Armstrong, K. Sotala, and S. S. . Héigeartaigh, “The errors, insights and lessons of famous AI predictions-and what they mean for the future,” *J. Exp. Theor. Artif. Intell.*, vol. 26, no. 3, pp. 317–342, 2014.
- [9] J. Kahn, “What Happens if AI Doesn’t Live Up to the Hype?,” *Bloomberg*, 2018. [Online]. Available: <https://www.bloomberg.com/news/articles/2018-06-18/what-happens-if-ai-doesn-t-live-up-to-the-hype>. [Accessed: 02-May-2019].
- [10] J. Bresnick, “Navigating the Hype of Healthcare Artificial Intelligence Companies,” *Health IT Analytics*, 2017. [Online]. Available: <https://healthitanalytics.com/features/navigating-the-hype-of-healthcare-artificial-intelligence-companies>. [Accessed: 02-May-2019].
- [11] IBM, “Beyond the hype: A guide to understanding and successfully implementing artificial intelligence within your business,” New York, 2018.
- [12] G. Marcus, “Deep Learning: {A} Critical Appraisal,” *CoRR*, vol. abs/1801.0, 2018.
- [13] F. Piekiewicz, “AI Winter Is Well On Its Way,” *Piekiewicz’s blog*, 2018. [Online]. Available: <https://blog.piekiewicz.info/2018/05/28/ai-winter-is-well-on-its-way/>.
- [14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
- [15] Nathaniel Scharping, “Artificial Intelligence Experts Respond to Elon Musk’s Dire Warning for U.S. Governors,” *Discover*, 2017. [Online]. Available: <http://blogs.discovermagazine.com/d-brief/2017/07/18/artificial-intelligence-elon-musk/#.XMq-Tegzbcd>. [Accessed: 02-May-2019].
- [16] R. Dillet, “Google’s AI chief thinks reports of the AI apocalypse are greatly exaggerated,” *TechCrunch*, 2017. [Online]. Available: <https://techcrunch.com/2017/09/19/googles-ai-chief-thinks-reports-of-the->

- ai-apocalypse-are-greatly-exaggerated/. [Accessed: 02-May-2019].
- [17] M. Mayo, “Is ‘Artificial Intelligence’ Dead? Long Live Deep Learning?!?” 15-Jul-2016.
- [18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.
- [19] Steve LeVine, “Artificial intelligence pioneer says we need to start over,” *Axios*, 2017. [Online]. Available: <https://www.axios.com/artificial-intelligence-pioneer-says-we-need-to-start-over-1513305524-f619efbd-9db0-4947-a9b2-7a4c310a28fe.html>. [Accessed: 02-May-2019].
- [20] E. Fiesler and R. Beale, *Handbook of Neural Computation*. New York, NY, USA, NY, USA: Oxford University Press, Inc., 1997.
- [21] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, vol. 1. 1999.
- [22] O. Urfalioglu and O. Arikan, “Symmetry Breaking in Neuroevolution: A Technical Report,” Jul. 2011.
- [23] T. Khanna, *Foundations of neural networks*. 1990.
- [24] M. Johnson, “Lecture 7 The Multilayer Perceptron,” *Massey University, Computer Science*, 2008. [Online]. Available: <http://www.massey.ac.nz/~mjjohnso/notes/59302/110.html>. [Accessed: 22-Aug-2019].
- [25] C. Szegedy *et al.*, “Intriguing properties of neural networks,” *CoRR*, vol. abs/1312.6, 2013.
- [26] N. Kriegeskorte, “Deep neural networks: a new framework for modelling biological vision and brain information processing,” *bioRxiv*, 2015.
- [27] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel, “ImageNet-trained {CNN}s are biased towards texture; increasing shape bias improves accuracy and robustness.,” in *International Conference on Learning Representations*, 2019.
- [28] J. Jo and Y. Bengio, “Measuring the tendency of CNNs to Learn Surface Statistical Regularities,” *CoRR*, vol. abs/1711.1, 2017.
- [29] W. Brendel and M. Bethge, “Approximating CNNs with Bag-of-local-Features models works surprisingly well on ImageNet,” *CoRR*, vol. abs/1904.0, 2019.
- [30] W. Wei, L. Liu, S. Truex, L. Yu, and M. E. Gursoy, “Adversarial Examples in Deep Learning: Characterization and Divergence,” *CoRR*, vol. abs/1807.0, 2018.
- [31] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry, “Adversarial Examples Are Not Bugs, They Are Features,” *arXiv e-prints*, p. arXiv:1905.02175, May 2019.
- [32] S. Qiu *et al.*, “Review of Artificial Intelligence Adversarial Attack and Defense Technologies,” *Appl. Sci.*, vol. 9, no. 5, p. 909, Mar. 2019.
- [33] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *arXiv Prepr. arXiv1412.6572*, 2014.
- [34] A. Kurakin, I. J. Goodfellow, and S. Bengio, “Adversarial Machine Learning at Scale,” *CoRR*, vol. abs/1611.0, 2016.
- [35] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A.

- Swami, “The Limitations of Deep Learning in Adversarial Settings,” *CoRR*, vol. abs/1511.0, 2015.
- [36] J. Su, D. V. Vargas, and K. Sakurai, “One pixel attack for fooling deep neural networks,” *CoRR*, vol. abs/1710.0, 2017.
- [37] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “DeepFool: a simple and accurate method to fool deep neural networks,” *CoRR*, vol. abs/1511.0, 2015.
- [38] M. Cisse, Y. Adi, N. Neverova, and J. Keshet, “Houdini: Fooling Deep Structured Prediction Models,” *arXiv e-prints*, p. arXiv:1707.05373, Jul. 2017.
- [39] N. Papernot, P. D. McDaniel, I. J. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, “Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples,” *CoRR*, vol. abs/1602.0, 2016.
- [40] M. Fredrikson, S. Jha, and T. Ristenpart, “Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures,” in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1322–1333.
- [41] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, “Stealing Machine Learning Models via Prediction APIs,” *CoRR*, vol. abs/1609.0, 2016.
- [42] A. M. Nguyen, J. Yosinski, and J. Clune, “Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images,” *CoRR*, vol. abs/1412.1, 2014.
- [43] I. S. K. Samuel G. Finlayson, John D. Bowers, Joichi Ito, Jonathan L. Zittrain, Andrew L. Beam, “Adversarial attacks on medical machine learning,” *Science (80-.)*, vol. 363, no. 6433, pp. 1287–1289, 2019.
- [44] A. Kurakin, I. J. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” *CoRR*, vol. abs/1607.0, 2016.
- [45] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, “Adversarial Patch,” *CoRR*, vol. abs/1712.0, 2017.
- [46] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok, “Synthesizing Robust Adversarial Examples,” *CoRR*, vol. abs/1707.0, 2017.
- [47] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, “Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1528–1540.
- [48] A. Bloor, X. He, C. D. Gill, Y. Vorobeychik, and X. Zhang, “Simple Physical Adversarial Examples against End-to-End Autonomous Driving Models,” *CoRR*, vol. abs/1903.0, 2019.
- [49] “Experimental Security Research of Tesla Autopilot,” 2019.
- [50] I. Evtimov *et al.*, “Robust Physical-World Attacks on Machine Learning Models,” *CoRR*, vol. abs/1707.0, 2017.
- [51] S. G. Finlayson, I. S. Kohane, and A. L. Beam, “Adversarial Attacks Against Medical Deep Learning Systems,” *CoRR*, vol. abs/1804.0, 2018.
- [52] F. Tramèr, N. Papernot, I. J. Goodfellow, D. Boneh, and P. D. McDaniel, “The Space of Transferable Adversarial Examples,” *CoRR*, vol. abs/1704.0,

- 2017.
- [53] N. Papernot, P. D. McDaniel, and I. J. Goodfellow, “Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples,” *CoRR*, vol. abs/1605.0, 2016.
 - [54] A. Shafahi, W. R. Huang, C. Studer, S. Feizi, and T. Goldstein, “Are adversarial examples inevitable?,” in *International Conference on Learning Representations*, 2019.
 - [55] R. Huang, B. Xu, D. Schuurmans, and C. Szepesvári, “Learning with a Strong Adversary,” *CoRR*, vol. abs/1511.0, 2015.
 - [56] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, “Ensemble Adversarial Training: Attacks and Defenses,” *arXiv e-prints*, p. arXiv:1705.07204, May 2017.
 - [57] H. Hosseini, Y. Chen, S. Kannan, B. Zhang, and R. Poovendran, “Blocking Transferability of Adversarial Examples in Black-Box Learning Systems,” *CoRR*, vol. abs/1703.0, 2017.
 - [58] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, “Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks,” *arXiv e-prints*, p. arXiv:1511.04508, Nov. 2015.
 - [59] S. Das, “CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more,” *Medium*, 2017. [Online]. Available: <https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5>. [Accessed: 19-Jun-2019].
 - [60] A. S. Ross and F. Doshi-Velez, “Improving the Adversarial Robustness and Interpretability of Deep Neural Networks by Regularizing their Input Gradients,” *CoRR*, vol. abs/1711.0, 2017.
 - [61] D. Gunning, “Explainable Artificial Intelligence (XAI),” *Defence Advanced Research Projects Agency*, 2018. [Online]. Available: <https://www.darpa.mil/program/explainable-artificial-intelligence>. [Accessed: 06-Jun-2019].
 - [62] F. Doshi-Velez and B. Kim, “Towards a rigorous science of interpretable machine learning,” *arXiv Prepr. arXiv1702.08608*, 2017.
 - [63] Z. C. Lipton, “The mythos of model interpretability,” *arXiv Prepr. arXiv1606.03490*, 2016.
 - [64] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, “Explaining Explanations: An Approach to Evaluating Interpretability of Machine Learning,” *CoRR*, vol. abs/1806.0, 2018.
 - [65] M. T. Ribeiro, S. Singh, and C. 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*, 2016, pp. 1135–1144.
 - [66] J. R. Zilke, E. Loza Mencía, and F. Janssen, “DeepRED – Rule Extraction from Deep Neural Networks BT - Discovery Science,” 2016, pp. 457–473.
 - [67] G. P. J. Schmitz, C. Aldrich, and F. S. Gouws, “ANN-DT: an algorithm for extraction of decision trees from artificial neural networks,” *IEEE Trans. Neural Networks*, vol. 10, no. 6, pp. 1392–1401, 1999.
 - [68] R. Andrews, J. Diederich, and A. B. Tickle, “Survey and critique of

- techniques for extracting rules from trained artificial neural networks,” *Knowledge-Based Syst.*, vol. 8, no. 6, pp. 373–389, 1995.
- [69] M. D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks,” *CoRR*, vol. abs/1311.2, 2013.
- [70] K. Simonyan, A. Vedaldi, and A. Zisserman, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps,” *CoRR*, vol. abs/1312.6, 2013.
- [71] B. Zhou, A. Khosla, À. Lapedriza, A. Oliva, and A. Torralba, “Object detectors emerge in Deep Scene CNNs,” Dec. 2014.
- [72] A. M. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune, “Synthesizing the preferred inputs for neurons in neural networks via deep generator networks,” *CoRR*, vol. abs/1605.0, 2016.
- [73] W. Nie, Y. Zhang, and A. Patel, “A Theoretical Explanation for Perplexing Behaviors of Backpropagation-based Visualizations,” *CoRR*, vol. abs/1805.0, 2018.
- [74] U. Güçlü and M. A. ~J. van Gerven, “Deep Neural Networks Reveal a Gradient in the Complexity of Neural Representations across the Brain’s Ventral Visual Pathway,” *arXiv e-prints*, p. arXiv:1411.6422, Nov. 2014.
- [75] A. Vaswani *et al.*, “Attention is All you Need,” in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5998–6008.
- [76] T. Xiao, Y. Xu, K. Yang, J. Zhang, Y. Peng, and Z. Zhang, “The Application of Two-level Attention Models in Deep Convolutional Neural Network for Fine-grained Image Classification,” *CoRR*, vol. abs/1411.6, 2014.
- [77] J. Lu, J. Yang, D. Batra, and D. Parikh, “Hierarchical Question-Image Co-Attention for Visual Question Answering,” *CoRR*, vol. abs/1606.0, 2016.
- [78] D. Britz, “Attention and Memory in Deep Learning and NLP,” *WILDML*, 2016. [Online]. Available: <http://www.wildml.com/2016/01/attention-and-memory-in-deep-learning-and-nlp/>. [Accessed: 22-Aug-2019].
- [79] A. S. Ross, M. C. Hughes, and F. Doshi-Velez, “Right for the Right Reasons: Training Differentiable Models by Constraining their Explanations,” *CoRR*, vol. abs/1703.0, 2017.
- [80] D. P. Kingma and M. Welling, “Auto-Encoding Variational Bayes,” *arXiv e-prints*, p. arXiv:1312.6114, Dec. 2013.
- [81] X. Chen, Y. Duan, R. Houthoofd, J. Schulman, I. Sutskever, and P. Abbeel, “InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets,” *CoRR*, vol. abs/1606.0, 2016.
- [82] Q. Zhang, Y. N. Wu, and S.-C. Zhu, “Interpretable Convolutional Neural Networks,” *CoRR*, vol. abs/1710.0, 2017.
- [83] S. Sabour, N. Frosst, and G. E. Hinton, “Dynamic Routing Between Capsules,” *CoRR*, vol. abs/1710.0, 2017.
- [84] S. Antol *et al.*, “{VQA:} Visual Question Answering,” *CoRR*, vol. abs/1505.0, 2015.
- [85] L. A. Hendricks, Z. Akata, M. Rohrbach, J. Donahue, B. Schiele, and T.

- Darrell, "Generating Visual Explanations," *CoRR*, vol. abs/1603.0, 2016.
- [86] D. H. Park *et al.*, "Multimodal Explanations: Justifying Decisions and Pointing to the Evidence," *CoRR*, vol. abs/1802.0, 2018.
- [87] S. K. Fixson, *The multiple faces of modularity—a literature analysis of a product concept for assembled hardware products*. 2003.
- [88] S. Miraglia, "Systems Architectures and Innovation: the Modularity-Integrality Framework," Cambridge, 2014.
- [89] K. J. Ishii, C. Juengel, and C. F. Eubanks, "Design for Product Variety: Key to Product Line Structuring," 1995.
- [90] N. P. Suh, "Development of the science base for the manufacturing field through the axiomatic approach," *Robot. Comput. Integr. Manuf.*, vol. 1, no. 3, pp. 397–415, 1984.
- [91] K. Ulrich, "The role of product architecture in the manufacturing firm," *Res. Policy*, vol. 24, no. 3, pp. 419–440, 1995.
- [92] K. T. Ulrich and S. D. Eppinger, *Product Design and Development*. McGraw-Hill/Irwin, 2012.
- [93] J.-B. Mouret and S. Doncieux, "MENNAG: a modular, regular and hierarchical encoding for neural-networks based on attribute grammars," *Evol. Intell.*, vol. 1, no. 3, pp. 187–207, 2008.
- [94] C. Y. Baldwin and K. B. Clark, "Managing in an age of modularity.," *Harv. Bus. Rev.*, vol. 75 5, pp. 84–93, 1997.
- [95] L. Marengo, C. Pasquali, and M. Valente, "Modularity: Understanding the Development and Evolution of Complex Natural Systems," 2005.
- [96] C. Y. Baldwin and K. B. Clark, *Design Rules: The Power of Modularity Volume 1*. Cambridge, MA, USA, MA, USA: MIT Press, 1999.
- [97] M. Schilling, "Toward a General Modular Systems Theory and Its Application to Interfirm Product Modularity," *Acad. Manag. Rev.*, vol. 25, 2000.
- [98] M. Sako, "Modularity and Outsourcing," 2005, pp. 229–253.
- [99] C. I. Barnard, *The functions of the executive*. Cambridge, MA, US, MA, US: Harvard University Press, 1938.
- [100] A. D. Chandler, *Strategy and Structure: Chapters in the History of the Industrial Enterprise*. Martino Publishing, 2013.
- [101] M. P. Follett, *Freedom & co-ordination: lectures in business organization*. Garland Pub., 1987.
- [102] P. R. Lawrence, J. Lorsch, P. Lawrence, and J. Lorsch, "Differentiation and Integration in Complex Organizations," *Adm. Sci. Q.*, vol. 12, no. 12, pp. 1–47, Aug. 1967.
- [103] J. D. Thompson, W. R. Scott, and M. N. Zald, *Organizations in Action: Social Science Bases of Administrative Theory*. Transaction Publishers, 2003.
- [104] O. Sporns, *Networks of the Brain*, 1st ed. The MIT Press, 2010.
- [105] D. Foray and C. Freeman, *Technology and the wealth of nations: the dynamics of constructed advantage*. Pinter, 1993.
- [106] N. Kashtan and U. Alon, "Spontaneous evolution of modularity and network motifs," *Proc. Natl. Acad. Sci.*, vol. 102, no. 39, pp. 13773–13778,

- 2005.
- [107] N. Kashtan, E. Noor, and U. Alon, “Varying environments can speed up evolution,” *Proc. Natl. Acad. Sci.*, vol. 104, no. 34, pp. 13711–13716, 2007.
- [108] H. Lipson, J. B. Pollack, and N. P. Suh, “On the Origin of Modular Variation,” *Evolution (N. Y.)*, vol. 56, no. 8, pp. 1549–1556, 2002.
- [109] “Homeotic Genes and Body Patterns,” *Genetic Science Learning Center*, 2016. [Online]. Available: <https://learn.genetics.utah.edu/content/basics/hoxgenes/>. [Accessed: 01-Jul-2019].
- [110] T. Dean, G. Corrado, and J. Shlens, “Three Controversial Hypotheses Concerning Computation in the Primate Cortex,” *Twenty-Sixth AAAI Conf. Artif. Intell.*, no. January, pp. 1543–1549, 2012.
- [111] G. P. Wagner and L. Altenberg, “Perspective: Complex Adaptations and the Evolution of Evolvability,” *Evolution (N. Y.)*, vol. 50, no. 3, pp. 967–976, 1996.
- [112] W. S. McCulloch and W. Pitts, “Neurocomputing: Foundations of Research,” J. A. Anderson and E. Rosenfeld, Eds. Cambridge, MA, USA, MA, USA: MIT Press, 1988, pp. 15–27.
- [113] F. Rosenblatt, “The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain,” *Psychol. Rev.*, pp. 65–386, 1958.
- [114] ScienceDaily, “Associative Memory -- Learning At All Levels,” *ScienceDaily*, 2007. [Online]. Available: <https://www.sciencedaily.com/releases/2007/03/070314134812.htm>. [Accessed: 25-Jun-2019].
- [115] G. Palm, “On representation and approximation of nonlinear systems,” *Biol. Cybern.*, vol. 31, no. 2, pp. 119–124, 1978.
- [116] G. P. Wagner, M. Pavlicev, and J. M. Cheverud, “The road to modularity,” *Nat. Rev. Genet.*, vol. 8, pp. 921–931, 2007.
- [117] D. Paranyushkin and N. Labs, “Metastability of Cognition in the Body-Mind-Environment Network,” 2014.
- [118] D. C. Plaut and G. E. Hinton, “Learning sets of filters using back-propagation,” *Comput. Speech Lang.*, vol. 2, no. 1, pp. 35–61, 1987.
- [119] J. A. Bullinaria, “To Modularize or Not To Modularize?,” in *Proceedings of the 2002 U.K Workshop on Computational Intelligence: UKCI-02*, 2002, pp. 3–10.
- [120] on Decomposition and D. M. Himmelblau, *Decomposition of large-scale problems*. Editor: David M. Himmelblau. North-Holland Pub. Co.; American Elsevier Pub. Co Amsterdam, New York, 1973.
- [121] R. A. Jacobs and M. I. Jordan, “Computational Consequences of a Bias Toward Short Connections,” *J. Cogn. Neurosci.*, vol. 4, no. 4, pp. 323–336, Oct. 1992.
- [122] K. O. Ellefsen, J. Huizinga, and J. Tørresen, “Guiding Neuroevolution with Structural Objectives,” *CoRR*, vol. abs/1902.0, 2019.
- [123] K. O. Ellefsen, J.-B. Mouret, and J. Clune, “Neural modularity helps organisms evolve to learn new skills without forgetting old skills,” *PLoS*

- Comput. Biol.*, vol. 11, no. 4, pp. e1004128–e1004128, Apr. 2015.
- [124] J. Huizinga, J.-B. Mouret, and J. Clune, “Does Aligning Phenotypic and Genotypic Modularity Improve the Evolution of Neural Networks?,” in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, 2016, pp. 125–132.
- [125] X. Yao, “A Review of Evolutionary Artificial Neural Networks,” *Int. J. Intell. Syst.*, vol. 4, pp. 539–567, 1993.
- [126] G. F. Miller, P. M. Todd, and S. U. Hegde, “Designing Neural Networks Using Genetic Algorithms,” in *Proceedings of the Third International Conference on Genetic Algorithms*, 1989, pp. 379–384.
- [127] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA, MA, USA: MIT Press, 1992.
- [128] T. R. Meagher, “Evolution and Today’s Society,” *Bioscience*, vol. 49, no. 11, pp. 923–925, Nov. 1999.
- [129] R. Dawkins, “Universal Darwinism,” in *The Nature of Life: Classical and Contemporary Perspectives from Philosophy and Science*, C. E. Cleland and M. A. Bedau, Eds. Cambridge: Cambridge University Press, 2010, pp. 360–373.
- [130] D. C. Dennett, “New Replicators, The.” Oxford University Press, 2005.
- [131] L. J. Fogel, *Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1999.
- [132] I. Rechenberg, *Evolutionsstrategie; Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Mit einem Nachwort von Manfred Eigen*. [Stuttgart-Bad Cannstatt]: Frommann-Holzboog, 1973.
- [133] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA, MA, USA: MIT Press, 1992.
- [134] J. Lehman and R. Miikkulainen, “Neuroevolution,” *Scholarpedia*, vol. 8, no. 6, p. 30977, 2013.
- [135] A. E. Eiben and J. Smith, “From evolutionary computation to the evolution of things,” *Nature*, vol. 521, p. 476, May 2015.
- [136] T. Bäck, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. New York, NY, USA: Oxford University Press, Inc., 1996.
- [137] “Why self-incompatibility in the Brassicaceae is totally cool,” *VINCENT CASTRIC LAB PAGE*. [Online]. Available: <http://vincentcastric.weebly.com/how-cool-is-si.html>. [Accessed: 14-Aug-2019].
- [138] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Boston, MA, USA, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.
- [139] J. A. Carr, “An Introduction to Genetic Algorithms,” 2014.
- [140] W. A. Melo, C. G. Freitas, C. D. Bacon, and R. G. Collevatti, “The road to evolutionary success: insights from the demographic history of an

- Amazonian palm,” *Heredity (Edinb.)*, vol. 121, no. 2, pp. 183–195, 2018.
- [141] W. Larson, “Genetic Programming: A Novel Failure,” 2009. [Online]. Available: <https://lethain.com/genetic-programming-a-novel-failure/>. [Accessed: 26-Aug-2019].
- [142] D. Whitley, M. Lunacek, and J. Knight, “Ruffled by Ridges: How Evolutionary Algorithms Can Fail BT - Genetic and Evolutionary Computation – GECCO 2004,” 2004, pp. 294–306.
- [143] N. Jakobi, “Harnessing Morphogenesis,” in *International Conference on Information Processing in Cells and Tissues*, 1995, pp. 29–41.
- [144] S. Alexander, “Meditations On Moloch,” *LESSWRONG*, 2014. [Online]. Available: <https://www.lesswrong.com/posts/TxcRbCYHaeL59aY7E/meditations-on-moloch>. [Accessed: 14-Aug-2019].
- [145] K. E. Kinnear Jr., Ed., *Advances in Genetic Programming*. Cambridge, MA, USA: MIT Press, 1994.
- [146] J. Lehman *et al.*, “The Surprising Creativity of Digital Evolution: {A} Collection of Anecdotes from the Evolutionary Computation and Artificial Life Research Communities,” *CoRR*, vol. abs/1803.0, 2018.
- [147] K. Sims, “Evolving 3D Morphology and Behavior by Competition,” *Artif. Life*, vol. 1, no. 4, pp. 353–372, Jan. 1994.
- [148] K. Sims, “Evolving Virtual Creatures,” in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, 1994, pp. 15–22.
- [149] C. O. Wilke, J. L. Wang, C. Ofria, R. E. Lenski, and C. Adami, “Evolution of digital organisms at high mutation rates leads to survival of the flattest,” *Nature*, vol. 412, no. 6844, pp. 331–333, 2001.
- [150] S. M. Stanley, *Macroevolution, Pattern and Process*. Johns Hopkins University Press, 1979.
- [151] N. Kashtan, M. Parter, E. Dekel, A. E. Mayo, and U. Alon, “EXTINCTIONS IN HETEROGENEOUS ENVIRONMENTS AND THE EVOLUTION OF MODULARITY,” *Evolution (N. Y.)*, vol. 63, no. 8, pp. 1964–1975, Aug. 2009.
- [152] J. B. S. (John B. S. Haldane 1892-1964, *Animal biology / by J.B.S. Haldane and Julian Huxley*. Oxf, 1934.
- [153] P. Neige, “3 - The Phenomenon of Evolutionary Radiation,” P. B. T.-E. of I. B. Neige, Ed. Elsevier, 2015, pp. 47–64.
- [154] M. Kirschner and J. Gerhart, “Evolvability,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 95, no. 15, pp. 8420–8427, Jul. 1998.
- [155] K. O. Stanley and R. Miikkulainen, “A Taxonomy for Artificial Embryogeny,” *Artif. Life*, vol. 9, no. 2, pp. 93–130, Apr. 2003.
- [156] E. Mjolsness, D. H. Sharp, and B. K. Alpert, “Scaling, Machine Learning, and Genetic Neural Nets,” *Adv. Appl. Math.*, vol. 10, no. 2, pp. 137–163, Jun. 1989.
- [157] M. Luerssen and D. Powers, *Graph design by graph grammar evolution*. 2007.
- [158] G. Halder, P. Callaerts, and W. J. Gehring, “Induction of Ectopic Eyes by

- Targeted Expression of the Eyeless Gene in Drosophila,” *Science* (80-.), vol. 267, no. 5205, pp. 1788–1792, 1995.
- [159] P. Bentley and S. Kumar, “Three Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem,” in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 1*, 1999, pp. 35–43.
- [160] K. O. Stanley, “Compositional Pattern Producing Networks: A Novel Abstraction of Development,” *Genet. Program. Evolvable Mach.*, vol. 8, no. 2, pp. 131–162, Jun. 2007.
- [161] H. Kitano, “Designing neural networks using genetic algorithms with graph generation system,” *Complex Syst.*, vol. 4, pp. 461–476, 1990.
- [162] J. Clune, “Evolving artificial neural networks with generative encodings inspired by developmental biology,” 2011. [Online]. Available: <https://www.youtube.com/watch?v=OwqCG3lcfKs>. [Accessed: 14-Aug-2019].
- [163] F. Dellaert and A. D. Beer, “Toward an evolvable model of development for autonomous agent synthesis,” 1994, pp. 246–257.
- [164] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, “Designing neural networks through neuroevolution,” *Nat. Mach. Intell.*, vol. 1, no. 1, pp. 24–35, 2019.
- [165] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, “Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning,” *CoRR*, vol. abs/1712.0, 2017.
- [166] D. J. Montana and L. Davis, “Training Feedforward Neural Networks Using Genetic Algorithms,” in *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1*, 1989, pp. 762–767.
- [167] J. D. Schaffer, D. Whitley, and L. J. Eshelman, “Combinations of genetic algorithms and neural networks: a survey of the state of the art,” in *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, 1992, pp. 1–37.
- [168] N. J. Radcliffe, “Genetic set recombination and its application to neural network topology optimisation,” *Neural Comput. Appl.*, vol. 1, no. 1, pp. 67–90, 1993.
- [169] P. J. B. Hancock, “Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification,” in *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, 1992, pp. 108–122.
- [170] K. O. Stanley and R. Miikkulainen, “Evolving Neural Networks Through Augmenting Topologies,” *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, Jun. 2002.
- [171] P. Stagg and C. Igel, “Neural network structures and isomorphisms: random walk characteristics of the search space,” in *2000 IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks. Proceedings of the First IEEE Symposium on Combinations of*

- Evolutionary Computation and Neural Networks (Cat. No.00, 2000, pp. 82–90.*
- [172] T. Froese, E. Spier, T. Froese, and E. Spier, “Convergence and crossover: The permutation problem revisited.” 2008.
- [173] R. C. Eberhart, *Computational Intelligence: Concepts to Implementations*. San Francisco, CA, USA, CA, USA: Morgan Kaufmann Publishers Inc., 2007.
- [174] M. Dragoni, A. Azzini, and A. Tettamanzi, “SimBa: A novel similarity-based crossover for neuro-evolution,” *Neurocomputing*, vol. 130, pp. 108–122, Apr. 2014.
- [175] S. Hafliðason and R. Neville, “Quantifying the Severity of the Permutation Problem in Neuroevolution BT - Natural Computing,” 2010, pp. 149–156.
- [176] J. Sánchez-velazco, “A Gendered Selection Strategies in Genetic Algorithms for Optimization,” Jan. 2003.
- [177] G. M. Edelman, *Neural Darwinism: The theory of neuronal group selection*. New York, NY, US: Basic Books, 1987.
- [178] S. Nolfi and D. Floreano, “Learning and Evolution,” *Auton. Robot.*, vol. 7, no. 1, pp. 89–113, Jul. 1999.
- [179] C. H. WADDINGTON, “CANALIZATION OF DEVELOPMENT AND THE INHERITANCE OF ACQUIRED CHARACTERS,” *Nature*, vol. 150, no. 3811, pp. 563–565, 1942.
- [180] G. E. Hinton and S. J. Nowlan, “Adaptive Individuals in Evolving Populations,” R. K. Belew and M. Mitchell, Eds. Boston, MA, USA, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1996, pp. 447–454.
- [181] S. Nolfi, D. Parisi, and J. L. Elman, “Learning and Evolution in Neural Networks,” *Adapt. Behav.*, vol. 3, no. 1, pp. 5–28, Jun. 1994.
- [182] J. M. Baldwin, “A New Factor in Evolution,” *Am. Nat.*, vol. 30, no. 354, pp. 441–451, 1896.
- [183] R. V Yampolskiy, “Why We Do Not Evolve Software? Analysis of Evolutionary Algorithms,” *Evol. Bioinform. Online*, vol. 14, pp. 1176934318815906–1176934318815906, Dec. 2018.
- [184] P. Koehn, “Genetic Encoding Strategies for Neural Networks,” Jul. 2019.
- [185] C. Cotta and P. Moscato, “Evolutionary Computation: Challenges and Duties,” 2006, pp. 53–72.
- [186] X. Yang, S. Deng, M. Ji, J. Zhao, and W. Zheng, “Neural Network Evolving Algorithm Based on the Triplet Codon Encoding Method,” *Genes (Basel)*, vol. 9, no. 12, 2018.
- [187] R. Lahoz-Beltra, “Quantum Genetic Algorithms for Computer Scientists,” *Computers*, vol. 5, no. 4, 2016.
- [188] J. Fekiac, I. Zelinka, and J. C. Burguillo, “A Review Of Methods For Encoding Neural Network Topologies In Evolutionary Computation,” in *ECMS*, 2011.
- [189] D. Floreano, P. Dürr, and C. Mattiussi, “Neuroevolution: from architectures to learning,” *Evol. Intell.*, vol. 1, pp. 47–62, 2008.
- [190] P. Koehn, “Combining Genetic Algorithms and Neural Networks : The Encoding Problem,” University of Tennessee, 1994.

- [191] A. Cangelosi, D. Parisi, and S. Nolfi, *Cell division and migration in a "genotype" for neural networks*, vol. 5. 1994.
- [192] S. Lucas, *Evolving Neural Network Learning Behaviours with Set-Based Chromosomes*. 1970.
- [193] M. Suchorzewski and J. Clune, "A Novel Generative Encoding for Evolving Modular, Regular and Scalable Networks," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, 2011, pp. 1523–1530.
- [194] M. Luerssen, *Experimental investigations into graph grammar evolution: a novel approach to evolutionary design*. 2009.
- [195] P. M. Todd, *Evolutionary Methods for Connectionist Architectures*. Psychology Department, Stanford, 1988.
- [196] A. Guha, S. A. Harp, and T. Samad, "Genetic algorithm synthesis of neural networks," 5,140,530, 1989.
- [197] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: optimizing connections and connectivity," *Parallel Comput.*, vol. 14, no. 3, pp. 347–361, 1990.
- [198] V. Maniezzo, "Searching among Search Spaces: Hastening the genetic evolution of feedforward neural networks BT - Artificial Neural Nets and Genetic Algorithms," 1993, pp. 635–642.
- [199] V. Maniezzo, "Genetic Evolution of the Topology and Weight Distribution of Neural Networks," *Trans. Neur. Netw.*, vol. 5, no. 1, pp. 39–53, Jan. 1994.
- [200] K. Deb, "Binary and Floating-point Function Optimization Using Messy Genetic Algorithms," University of Alabama, Tuscaloosa, AL, USA, AL, USA, 1991.
- [201] D. E. Goldberg, K. Deb, and B. Korb, "Messy Genetic Algorithms Revisited: Studies in Mixed Size and Scale," *Complex Syst.*, vol. 4, 1990.
- [202] D. E. Goldberg, B. Korb, and K. Deb, "Messy Genetic Algorithms: Motivation, Analysis, and First Results," *Complex Syst.*, vol. 3, 1989.
- [203] D. Dasgupta and D. R. Mcgregor, "sGA: A Structured Genetic Algorithm," 1992.
- [204] J. Clune, J.-B. Mouret, and H. Lipson, "The evolutionary origins of modularity," *arXiv e-prints*, p. arXiv:1207.2743, Jul. 2012.
- [205] K. O. Ellefsen and J. Torresen, *Evolving neural networks with multiple internal models*. 2017.
- [206] J. Clune, K. O. Stanley, R. T. Pennock, and C. Ofria, "On the Performance of Indirect Encoding Across the Continuum of Regularity," *IEEE Trans. Evol. Comput.*, vol. 15, no. 3, pp. 346–367, 2011.
- [207] W. Schiffmann, *Encoding Feedforward Networks for Topology Optimization By Simulated Evolution*, vol. 1. 2000.
- [208] W. Schiffmann, M. Joost, and R. Werner, "Performance Evaluation of Evolutionarily Created Neural Network Topologies," in *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, 1991, pp. 274–283.
- [209] W. Schiffmann, M. Joost, and R. Werner, "Synthesis and Performance

- Analysis of Multilayer Neural Network Architectures.” 1992.
- [210] W. Schiffmann, M. Joost, and R. Werner, “Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons,” 1994.
- [211] W. Schiffmann, M. Joost, and R. Werner, “Application of Genetic Algorithms to the Construction of Topologies for Multilayer Perceptrons BT - Artificial Neural Nets and Genetic Algorithms,” 1993, pp. 675–682.
- [212] D. White and P. Ligomenides, “GANNet: A genetic algorithm for optimizing topology and weights in neural network design BT - New Trends in Neural Computation,” 1993, pp. 322–327.
- [213] S. A. Harp, T. Samad, and A. Guha, “Towards the Genetic Synthesis of Neural Network,” in *Proceedings of the Third International Conference on Genetic Algorithms*, 1989, pp. 360–369.
- [214] S. A. Harp and T. Samad, “Genetic synthesis of neural network architecture,” 1991.
- [215] M. Mandischer, “Representation and Evolution of Neural Networks BT - Artificial Neural Nets and Genetic Algorithms,” 1993, pp. 643–649.
- [216] C. Jacob and J. Rehder, “Evolution of neural net architectures by a hierarchical grammar-based genetic system BT - Artificial Neural Nets and Genetic Algorithms,” 1993, pp. 72–79.
- [217] K. O. Stanley and R. Miikkulainen, “Efficient evolution of neural network topologies,” *Proc. 2002 Congr. Evol. Comput. CEC’02 (Cat. No.02TH8600)*, vol. 2, pp. 1757–1762 vol.2, 2002.
- [218] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants*. Berlin, Heidelberg: Springer-Verlag, 1990.
- [219] F. Gruau *et al.*, “Neural Network Synthesis Using Cellular Encoding And The Genetic Algorithm.” 1994.
- [220] E. J. W. Boers and H. Kuiper, “Biological Metaphors and the Design of Modular Artificial Neural Networks.” 1992.
- [221] L. Campos, L. L. M. L. de Campos, M. Roisenberg, R. Célio, R. Oliveira, and R. C. L. de Oliveira, *Automatic design of Neural Networks with L-Systems and genetic algorithms - A biologically inspired methodology*. 2011.
- [222] G. Hornby, *Generative Representations for Evolving Families of Designs*, vol. 2724. 2003.
- [223] S. Nolfi and D. Parisi, “Growing Neural Networks,” in *THE HANDBOOK OF BRAIN THEORY AND NEURAL NETWORKS*, 1991, pp. 431–434.
- [224] J. W. L. Merrill and R. F. Port, “Fractally configured neural networks,” *Neural Networks*, vol. 4, no. 1, pp. 53–60, 1991.
- [225] S. Luke and L. Spector, “Evolving Graphs and Networks with Edge Encoding: Preliminary Report,” in *Late Breaking Papers at the Genetic Programming 1996 Conference*, 1996, pp. 117–124.
- [226] E. D. De Jong and J. B. Pollack, “Utilizing bias to evolve recurrent neural networks,” in *IJCNN’01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, 2001, vol. 4, pp. 2667–2672 vol.4.
- [227] K. O. Stanley, D. B. D’Ambrosio, and J. Gauci, “A Hypercube-based

- Encoding for Evolving Large-scale Neural Networks,” *Artif. Life*, vol. 15, no. 2, pp. 185–212, Apr. 2009.
- [228] J. Clune, C. Ofria, and R. T. Pennock, “The Sensitivity of HyperNEAT to Different Geometric Representations of a Problem,” in *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, 2009, pp. 675–682.
- [229] J. Clune, C. Ofria, and R. T. Pennock, “How a Generative Encoding Fares As Problem-Regularity Decreases,” in *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature --- PPSN X - Volume 5199*, 2008, pp. 358–367.
- [230] J. Clune, B. E. Beckmann, R. T. Pennock, and C. Ofria, “HybrID: A Hybridization of Indirect and Direct Encodings for Evolutionary Computation,” in *Proceedings of the 10th European Conference on Advances in Artificial Life: Darwin Meets Von Neumann - Volume Part II*, 2011, pp. 134–141.
- [231] S. Risi, J. Lehman, and K. O. Stanley, “Evolving the Placement and Density of Neurons in the Hyperneat Substrate,” in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, 2010, pp. 563–570.
- [232] S. Risi and K. O. Stanley, “Enhancing Es-hyperneat to Evolve More Complex Regular Neural Networks,” in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, 2011, pp. 1539–1546.
- [233] S. Risi and K. O. Stanley, “A unified approach to evolving plasticity and neural geometry,” in *The 2012 International Joint Conference on Neural Networks (IJCNN)*, 2012, pp. 1–8.
- [234] B. L. M. Happel and J. M. J. Murre, “Design and evolution of modular neural network architectures,” *Neural Networks*, vol. 7, no. 6, pp. 985–1004, 1994.
- [235] J. Bullinaria, “Understanding the Emergence of Modularity in Neural Systems,” *Cogn. Sci.*, vol. 31, pp. 673–695, Jul. 2007.
- [236] J. Pepper, “The evolution of modularity in genome architecture,” *Artif. Life - ALIFE*, Jan. 2000.
- [237] B. A. Høverstad, “Noise and the Evolution of Neural Network Modularity,” *Artif. Life*, vol. 17, no. 1, pp. 33–50, Jan. 2011.
- [238] S. Li and J. Yuan, “The modularity in freeform evolving neural networks,” in *2011 IEEE Congress of Evolutionary Computation (CEC)*, 2011, pp. 2605–2610.
- [239] A. Di Ferdinando, R. Calabretta, and D. Parisi, “Evolving Modular Architectures for Neural Networks BT - Connectionist Models of Learning, Development and Evolution,” 2001, pp. 253–262.
- [240] J. Kim and M. Kim, “The Mathematical Structure of Characters and Modularity,” in *The Character Concept in Evolutionary Biology*, G. P. Wagner, Ed. Academic Press, 2001.
- [241] D. E. Rumelhart, “Lecture at the 1988 Connectionist Models Summer School,” Pittsburgh, PA, 1988.

- [242] J. Bullinaria, “Simulating the Evolution of Modular Neural Systems,” May 2001.
- [243] J. Clune, B. E. Beckmann, P. K. McKinley, and C. Ofria, “Investigating Whether hyperNEAT Produces Modular Neural Networks,” in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, 2010, pp. 635–642.
- [244] J. Huizinga, J. Clune, and J.-B. Mouret, “Evolving Neural Networks That Are Both Modular and Regular: HyperNEAT Plus the Connection Cost Technique,” in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, 2014, pp. 697–704.
- [245] P. Verbancsics and K. O. Stanley, “Constraining Connectivity to Encourage Modularity in HyperNEAT,” in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, 2011, pp. 1483–1490.
- [246] T. Taylor *et al.*, “WebAL Comes of Age: A Review of the First 21 Years of Artificial Life on the Web,” *Artif. Life*, vol. 22, pp. 364–407, Jul. 2016.
- [247] R. Velez and J. Clune, “Identifying Core Functional Networks and Functional Modules Within Artificial Neural Networks via Subsets Regression,” in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, 2016, pp. 181–188.
- [248] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization.” 2014.
- [249] L. Prechelt, “Early Stopping — But When? BT - Neural Networks: Tricks of the Trade: Second Edition,” G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 53–67.
- [250] A. McIntyre, M. Kallada, C. G. Miguel, and C. F. da Silva, “neat-python.”
- [251] K. Miettinen, *Nonlinear Multiobjective Optimization*. Springer US, 1999.
- [252] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II,” *Trans. Evol. Comp*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
- [253] E. Zitzler, M. Laumanns, and L. Thiele, “SPEA2: Improving the Strength Pareto Evolutionary Algorithm,” 2001.
- [254] Q. Zhang and H. Li, “MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition,” *IEEE Trans. Evol. Comput.*, vol. 11, no. 6, pp. 712–731, Dec. 2007.
- [255] D. H. Wolpert and W. G. Macready, “No Free Lunch Theorems for Optimization,” *Trans. Evol. Comp*, vol. 1, no. 1, pp. 67–82, Apr. 1997.
- [256] M. E. J. Newman, “Modularity and community structure in networks,” *Proc. Natl. Acad. Sci.*, vol. 103, no. 23, pp. 8577–8582, 2006.
- [257] E. A. Leicht and M. E. J. Newman, “Community Structure in Directed Networks,” *Phys. Rev. Lett.*, vol. 100, no. 11, Mar. 2008.
- [258] L. Asmal, S. Du Plessis, M. Vink, B. Chiliza, S. Kilian, and R. Emsley, “Symptom attribution and frontal cortical thickness in first-episode schizophrenia,” *Early Interv. Psychiatry*, vol. 12, Aug. 2016.
- [259] L. Asmal *et al.*, “Childhood Trauma Associated White Matter Abnormalities in First-Episode Schizophrenia,” *Schizophr. Bull.*, vol. 45,

- no. 2, pp. 369–376, Jun. 2018.
- [260] S. Du Plessis *et al.*, “Childhood trauma and hippocampal subfield volumes in first-episode schizophrenia and healthy controls,” *Schizophr. Res.*, Oct. 2019.
- [261] A. M. Dale, B. Fischl, and M. I. Sereno, “Cortical Surface-Based Analysis: I. Segmentation and Surface Reconstruction,” *Neuroimage*, vol. 9, no. 2, pp. 179–194, 1999.
- [262] R. S. Desikan *et al.*, “An automated labeling system for subdividing the human cerebral cortex on MRI scans into gyral based regions of interest,” *Neuroimage*, vol. 31, no. 3, pp. 968–980, 2006.
- [263] B. Fischl *et al.*, “Automatically Parcellating the Human Cerebral Cortex,” *Cereb. Cortex*, vol. 14, no. 1, pp. 11–22, Jan. 2004.
- [264] B. Fischl and A. M. Dale, “Measuring the thickness of the human cerebral cortex from magnetic resonance images,” *Proc. Natl. Acad. Sci.*, vol. 97, no. 20, pp. 11050 LP – 11055, Sep. 2000.
- [265] B. Fischl *et al.*, “Whole Brain Segmentation: Automated Labeling of Neuroanatomical Structures in the Human Brain,” *Neuron*, vol. 33, no. 3, pp. 341–355, 2002.
- [266] B. Fischl, M. I. Sereno, and A. M. Dale, “Cortical Surface-Based Analysis: II: Inflation, Flattening, and a Surface-Based Coordinate System,” *Neuroimage*, vol. 9, no. 2, pp. 195–207, 1999.
- [267] B. Fischl, M. I. Sereno, R. B. H. Tootell, and A. M. Dale, “High-resolution intersubject averaging and a coordinate system for the cortical surface,” *Hum. Brain Mapp.*, vol. 8, no. 4, pp. 272–284, 1999.
- [268] X. Han *et al.*, “Reliability of MRI-derived measurements of human cerebral cortical thickness: The effects of field strength, scanner upgrade and manufacturer,” *Neuroimage*, vol. 32, no. 1, pp. 180–194, 2006.
- [269] H. Koch, “Une methode geometrique elementaire pour l’etude de certaines questions de la theorie des courbes planes,” *Acta Math.*, vol. 30, pp. 145–174, 1906.
- [270] B. B. Mandelbrot, W. H. Freeman, and Company, *The Fractal Geometry of Nature*. Henry Holt and Company, 1983.
- [271] S. Wolfram, “Computer Software in Science and Mathematics,” *Sci. Am.*, vol. 251, no. 3, pp. 188–203, 1984.
- [272] N. Chomsky, “Three models for the description of language,” *IRE Trans. Inf. Theory*, vol. 2, no. 3, pp. 113–124, 1956.
- [273] G. Ochoa, “On Genetic Algorithms and Lindenmayer Systems,” in *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, 1998, pp. 335–344.

Appendix A Neuroevolutionary Encodings

A.1 Direct Encoding Strategies

Connection-based

(DE-1) – Bit encoding (only connections); Described by Todd [195], this method (named *Innervator*) was presented by Miller [126] noting influences from Guha, Harp and Samad [196] and is perhaps the earliest known neuroevolutionary attempt. The entries in a phenotype network’s connectivity matrix are directly represented by equivalent binary strings that are concatenated to form the genotype. Miller implements crossover as simple row swapping and mutation as single-bit perturbations to the connectivity matrix.

(DE-2) – Bit encoding (connections and weights): Whitley’s *Genitor* extends Miller’s *Innervator* to evolve synaptic weights by appending weight encoding bits to an index bit indicating the existence of a connection [197]. Variation is produced using crossover as recombination and standard single bit mutation. The genotype is not limited to binary encodings as evidenced by real-valued variants by Montana et al. [166] and granularity evolution improvements by Maniezzo [198], [199].

Unit-based

(DE-3) – Neurons as parameter strings; designed by Schiffmann, Joost and Werner [207]–[211], nodes are abstractly represented as the discrete elements of the genotype. Each genetic character contains a numeric node identifier, a crossover/mutation mutable list of input nodes and a terminating character which can be randomly selected as a crossover point. White’s GANNet [212] extends this method to include synaptic weights, activation functions and additional constraints.

(DE-4) – Neurons as parameter trees: Koza’s work focused on genetic programming, and he applied his methods to neural networks as an example [127]. Instead of representing them as strings, this method represents neurons as nodes in LISP expressions. Since crossover is defined as an exchange between two subtrees, recombinant variation can be an exchange of weight values by cutting at W elements) or neurons (by cutting at P elements) at arbitrary scale. Mutation replaces a compatible target subtree with a randomly generated one.

(DE-5) – Layers as parameter strings; a variant of Harp’s Genesys [213], [214], Mandischer [215] used on genotype representations that evolve layers as their discrete unit. Individual connections are not represented; instead layers are specified in terms of their number of neurons (size) as well as the radius and density of incoming/outgoing connections. Each genotype specifies the learning rate and momentum which allows learning characteristics to evolve simultaneously.

Pathway-based

(DE-6) – Context-free grammar encoding; put forward by Jacob and Rehder [216], a network phenotype is specified by pathways linking input (specified as $\{i_1, i_2, i_3, \dots, i_n\}$), hidden (enumerated as $\{1, 2, 3, \dots, n\}$) and output neurons (specified as $\{o_1, o_2, o_3, \dots, o_n\}$). Mutation can insert/remove whole paths or individual neurons; by screening duplicate paths, connectivity can be modified by inserting a mutated copy of an existing path. Crossover swaps existing paths between genotypes.

(DE-7) – NEAT; in full, *Neuroevolution of Augmenting Topologies* [217] specifies genotypes using two internal chromosomes that encode node and connection information respectively. The node chromosome is simply a list of all neurons in the network, each distinguished by a numeric identifier and type (input – sensor – /hidden/output). A genetic character in the connection chromosome specifies a connection's terminal nodes, weight, activation (which regulates gene expression) and an innovation number that tracks the gene's mutational origin. For example, a mutation that adds a neuron is enumerated and stored as the new gene's innovation number. The assumption behind the innovation number is that networks which followed similar evolutionary paths encode similar functionality; this helps to avoid the competing conventions problem since crossover only acts on genotypes with sufficiently similar evolutionary lineages. Genotype similarity is defined in terms of excess, disjointness and average weight distance.

A.2 Indirect Encoding Strategies

(IE-1) – Formal L-system; the Lindenmayer's eponymous L-systems [218] grew out of research in describing fractal patterns through rewriting systems that can express polygons [269], [270], arrays [271] or strings [272]; Lindenmayer's systems introduced parallel string rewriting that more closely resembles the parallel nature of biological embryogenesis. Bracketed, context-sensitive and/or probabilistic L-systems can create structures that mirror plant growth; those same L-systems can be used to construct neural networks. In the neuroevolutionary scenario (Figure 40), an individual genotype would be a starting string/symbol and a set of production rules; the phenotype is the neural network that results. In this way, evolution proceeds by discovering better production rules. Variation operators in Lindenmayer systems are implementation specific. Ochoa implements crossover as an exchange of bracketed substrings (like crossover in Koza's parameter trees) and mutation as the replacement of a single symbol or bracketed substring [273].

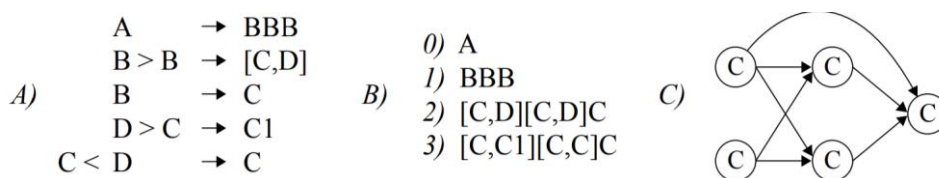


Figure 40. Neuroevolutionary L-systems; adapted from [188].

(IE-2) – Matrix-generalized L-system; as an “*augmentation of Lindenmayer’s L-system*”, Kitano’s rewriting system acts on very similar principles [161]. It expands a starting matrix of symbols by replacing individual symbols according to the set of production rules which culminates in symbols being rewritten as 1s and 0s (although this is not necessarily a terminating condition). Again, genotypes are individual sets of production rules acting on starting symbols that can be expanded into network phenotypes. Kitano does not elaborate on the variation operators used in his experiments, but methods comparable to those used by formal L-systems can be inferred.

(IE-3) – Program-tree grammar encoding; in formal L-systems and its matrix generalizations, rewritten symbols correspond to phenotype structures such as neurons and connections. In contrast, Gruau’s cellular encodings [219] encode symbols that represent program operations that act on ancestor cells and internal registers that indicate connections, bias and threshold values as well as development cues. Such program operations not only instruct cells divide to sequentially (SEQ) or in parallel (PAR) but can also specify a fixed number of recursive readings (REC) of the program tree. Noting their identical structure, Gruau implements crossover and mutation like Koza’s parametric trees.

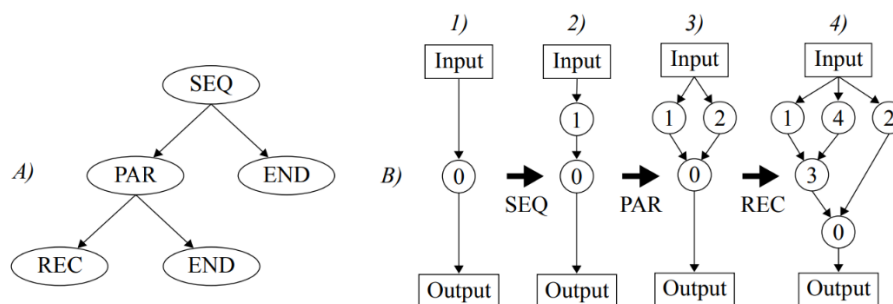


Figure 41. Cellular encodings by Gruau; adapted from [188].

(IE-4) – Hypercube-based indirect encoding; as implied by the name, HyperNeat (*Hypercube-based Neuroevolution of Augmenting Topologies*) extends NEAT to create a novel encoding strategy that uses CPPNs (compositional pattern producing networks) to abstract embryogeny. As explained by Stanley et al. [227], the process “*can produce connectivity patterns with symmetries and repeating motifs by interpreting spatial patterns generated within a hypercube as connectivity patterns in a lower-dimensional space*”. This method involves two mapping steps. First, NEAT’s direct split encoding is read to construct a CPPN, and then the queried CPPN’s outputs (taking inputs from a predetermined substrate) are used to build the phenotype. Crossover and mutation only act on the split encoding; therefore, the variation operators are identical to their implementation in NEAT.

Appendix B Experimental Data

B.1 Experiment 1 – Geometrically Separable Task

Table 5. Percentile data and Mann-Whitney U-Tests & P-values for experiment 1 (geometrically separable task).

	LCC		GCC		FIC		EIC		GIC	
	U	P	U	P	U	P	U	P	U	P
LCC	-	-	279	0.006	304	0.015	333	0.041	220	<0.001
GCC	279	0.006	-	-	99	<0.001	110	<0.001	115	<0.001
FIC	304	0.015	99	<0.001	-	-	410	0.277	285	0.007
EIC	333	0.041	110	<0.001	410	0.277	-	-	233	0.001
GIC	220	<0.001	115	<0.001	285	0.007	233	0.001	-	-
Q-score percentiles										
25%	0.033		0.000		0.454		0.430		0.734	
50%	0.089		0.032		0.665		0.620		1.016	
75%	0.860		0.212		0.820		0.770		1.073	
Task performance percentiles (%)										
25%	88.695		85.625		92.405		92.110		92.125	
50%	91.940		87.400		92.930		93.020		93.420	
75%	93.695		90.575		94.305		94.485		94.460	

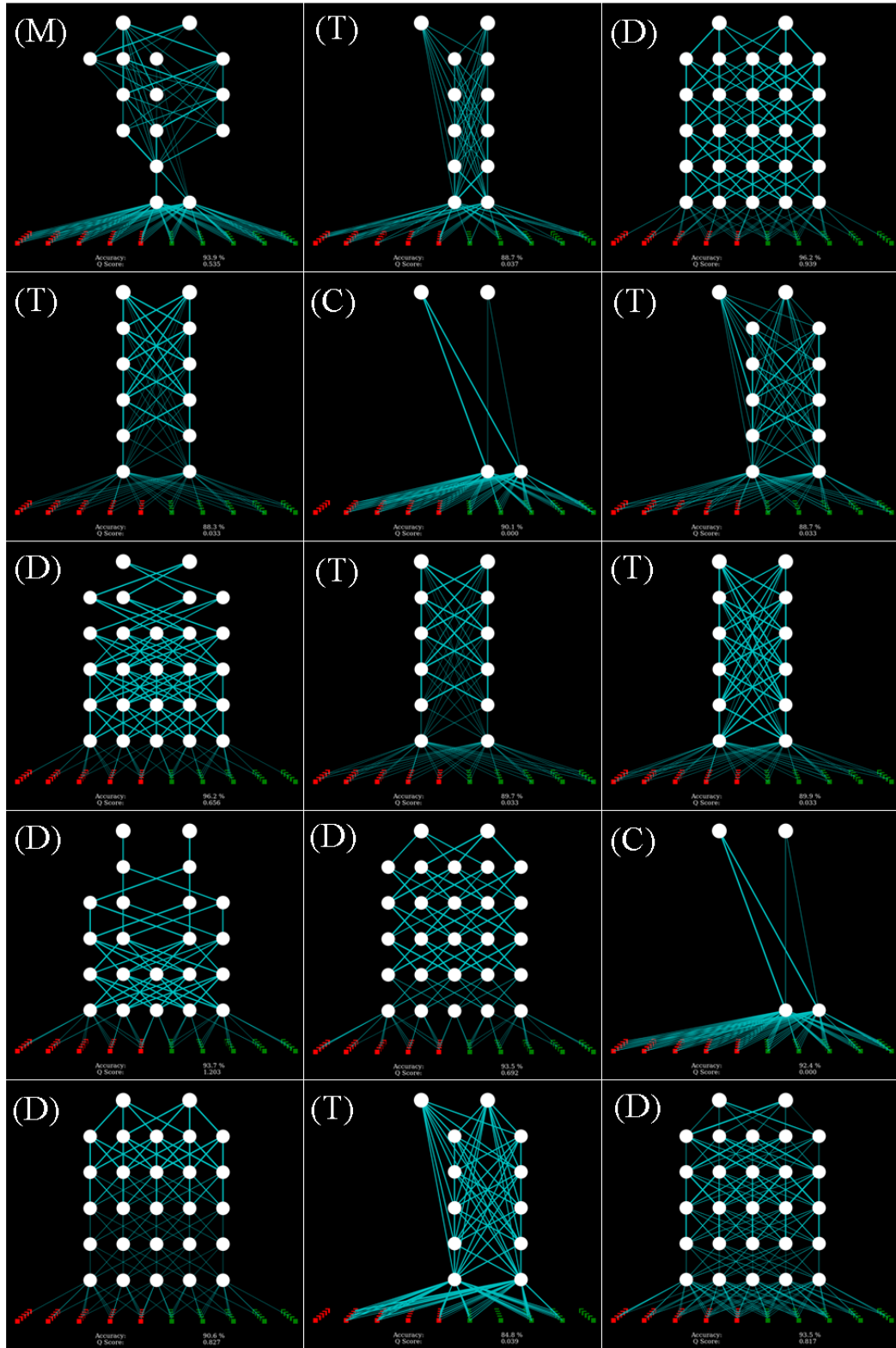


Figure 42. Experiment 1, geometrically separable task, LCC’s final networks (1-15); The ‘dense’ motif (marked D) appears in 36.7% of runs. The ‘tower’ motif (marked T) appears in 30% of runs. The ‘messy’ motif (marked M) appears in 16.7% of runs. The ‘collapsed’ motif (marked C) appears in 10% of runs. The ‘sparse’ motif (marked S) appears in 6.7% of runs.

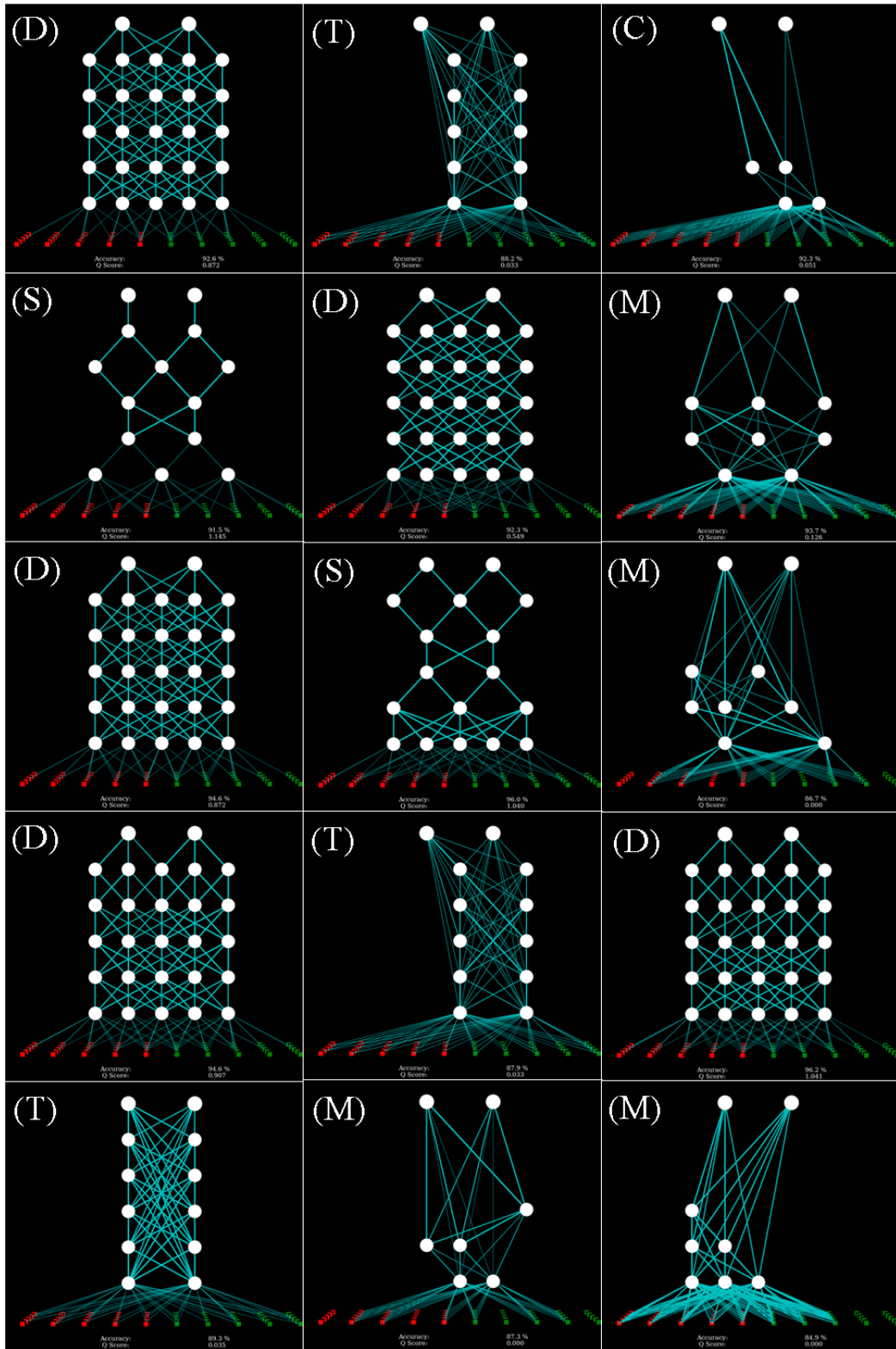


Figure 43. Experiment 1, geometrically separable task, LCC's final networks (16-30); The 'dense' motif (marked *D*) appears in 36.7% of runs. The 'tower' motif (marked *T*) appears in 30% of runs. The 'messy' motif (marked *M*) appears in 16.7% of runs. The 'collapsed' motif (marked *C*) appears in 10% of runs. The 'sparse' motif (marked *S*) appears in 6.7% of runs.

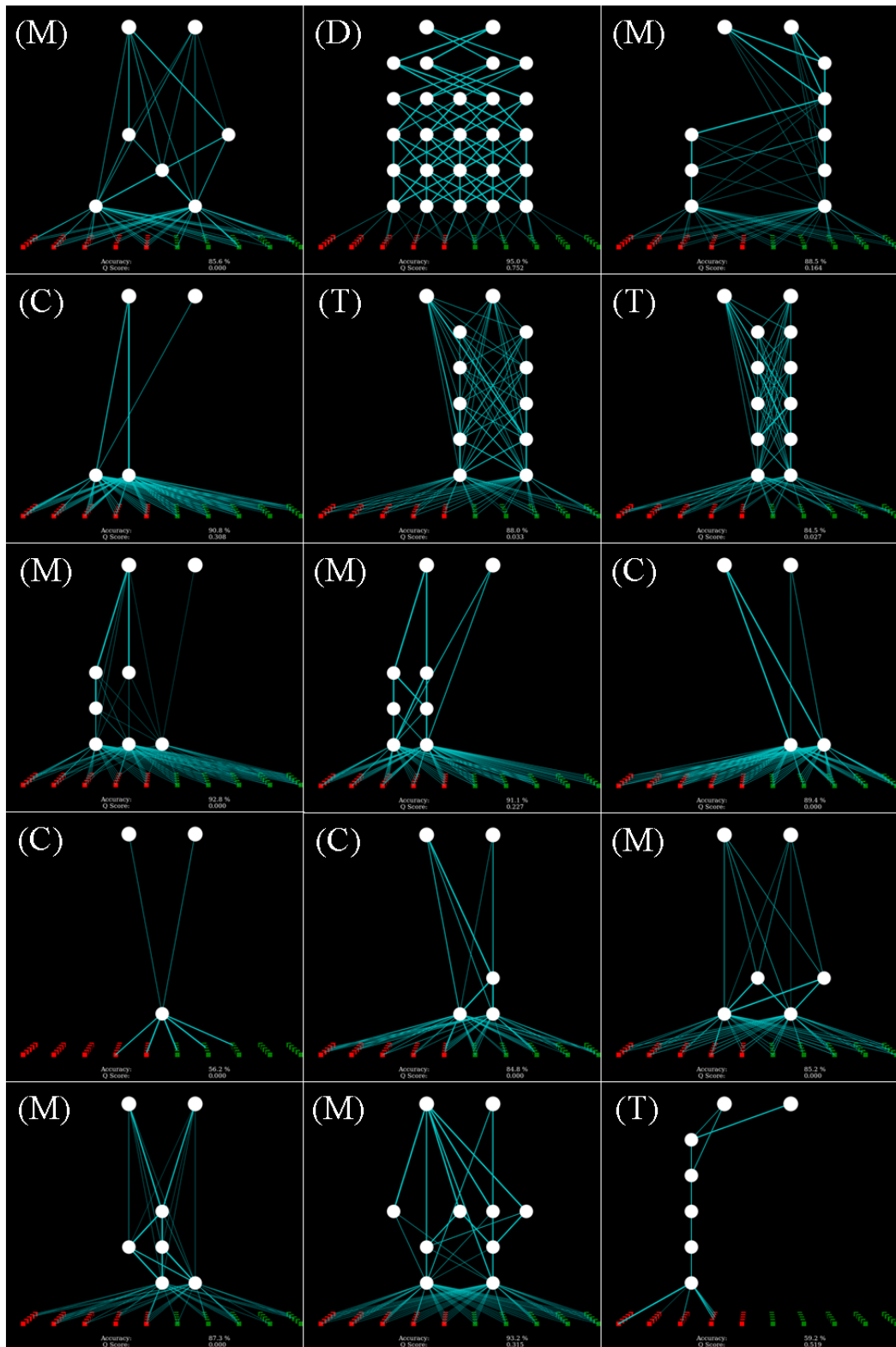


Figure 44. Experiment 1, geometrically separable task, GCC's final networks (1-15); The 'messy' motif (marked *M*) appears in 33.3% of runs. The 'tower' motif (marked *T*) appears in 30%. The 'collapsed' motif (marked *C*) appears in 23.3%. The 'dense' motif (marked *D*) in 13.3% of runs.

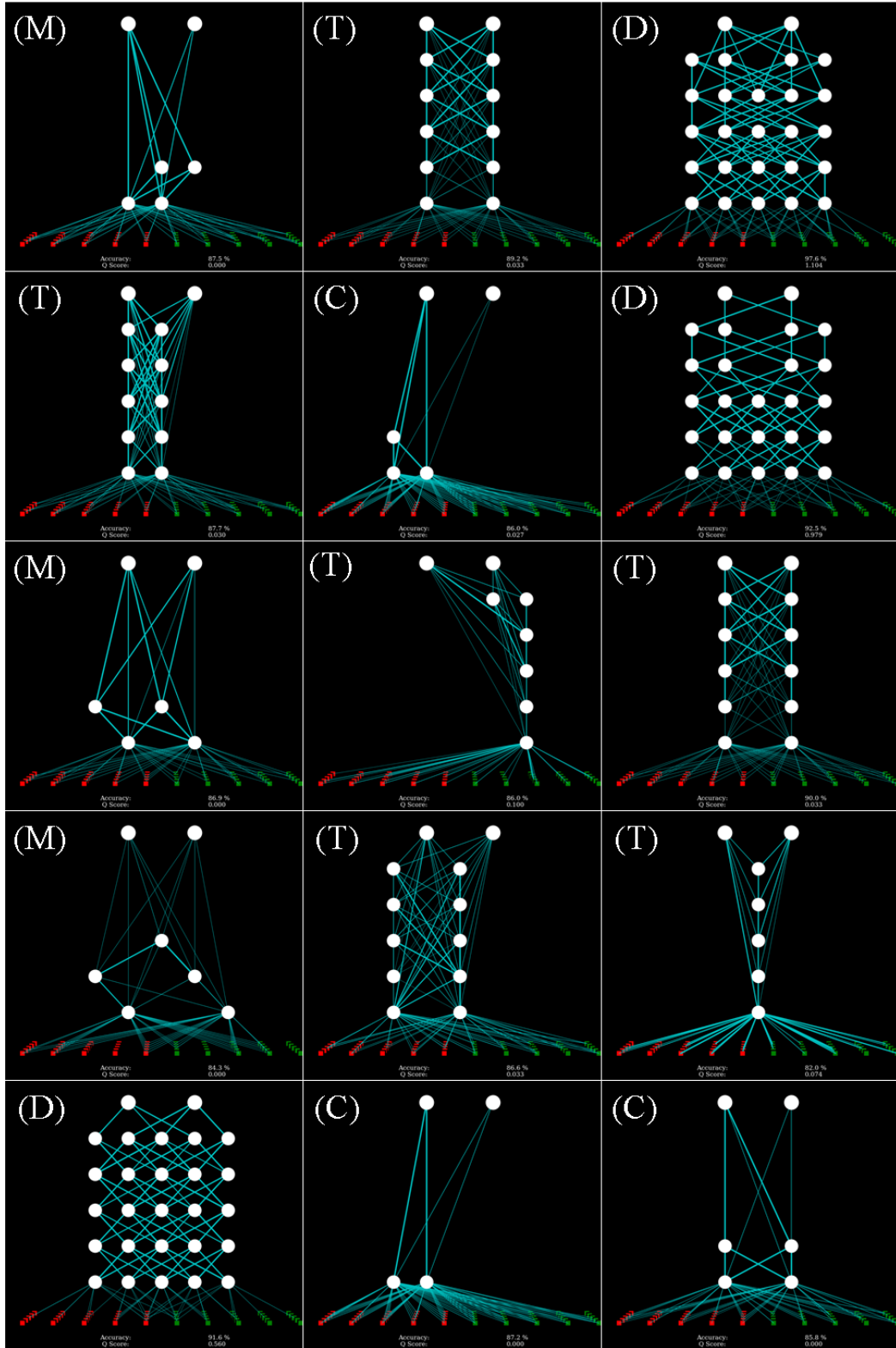


Figure 45. Experiment 1, geometrically separable task, GCC's final networks (16-30); The 'messy' motif (marked *M*) appears in 33.3% of runs. The 'tower' motif (marked *T*) appears in 30%. The 'collapsed' motif (marked *C*) appears in 23.3%. The 'dense' motif (marked *D*) in 13.3% of runs.

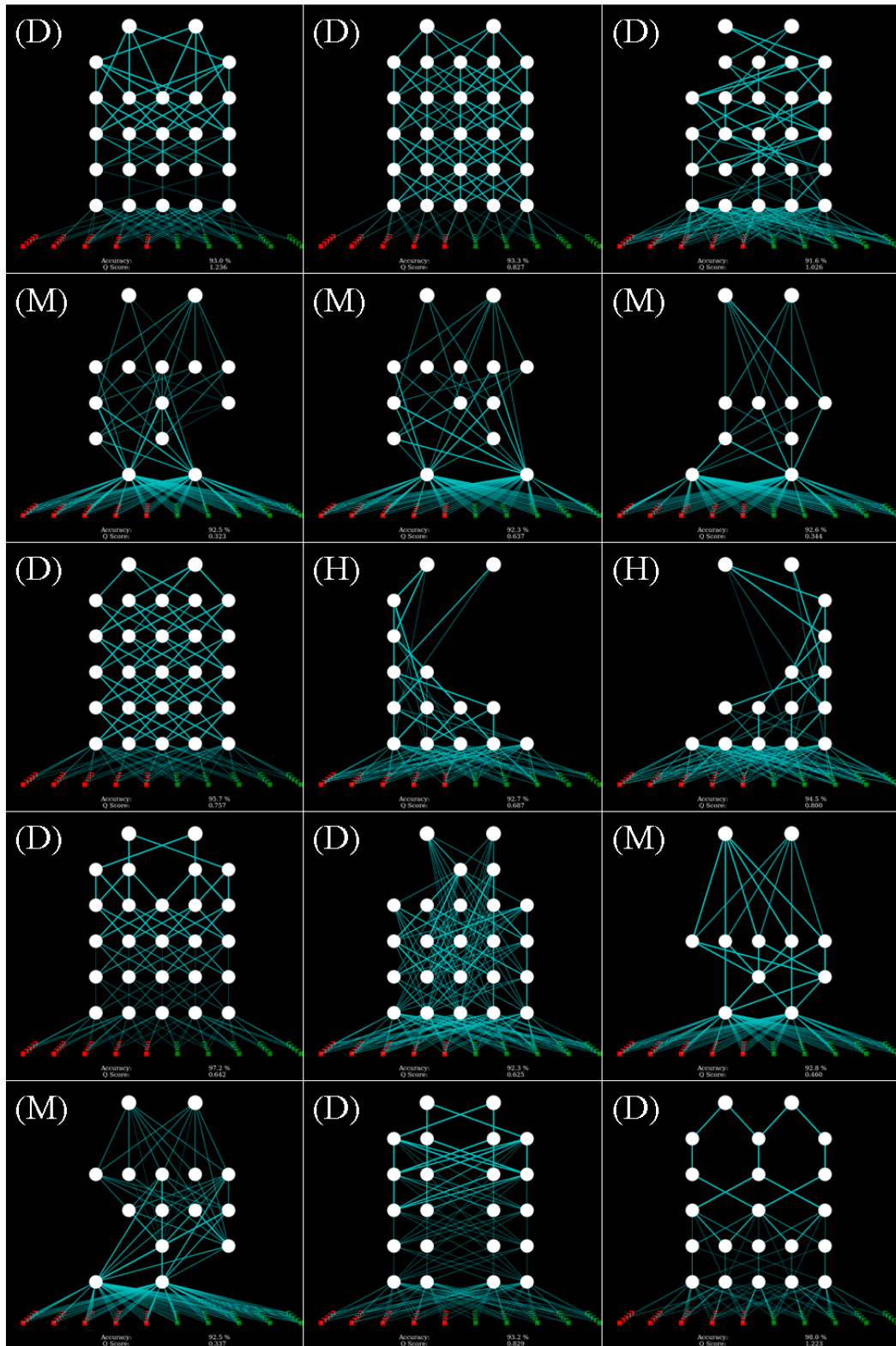


Figure 46. Experiment 1, geometrically separable task, FIC's final networks (1-15). The 'dense' motif (marked *D*) appears in 53.3% of runs. The 'messy' motif (marked *M*) appears in 40% of runs. The 'half' motif (marked *H*) appears in 6.7% of runs.

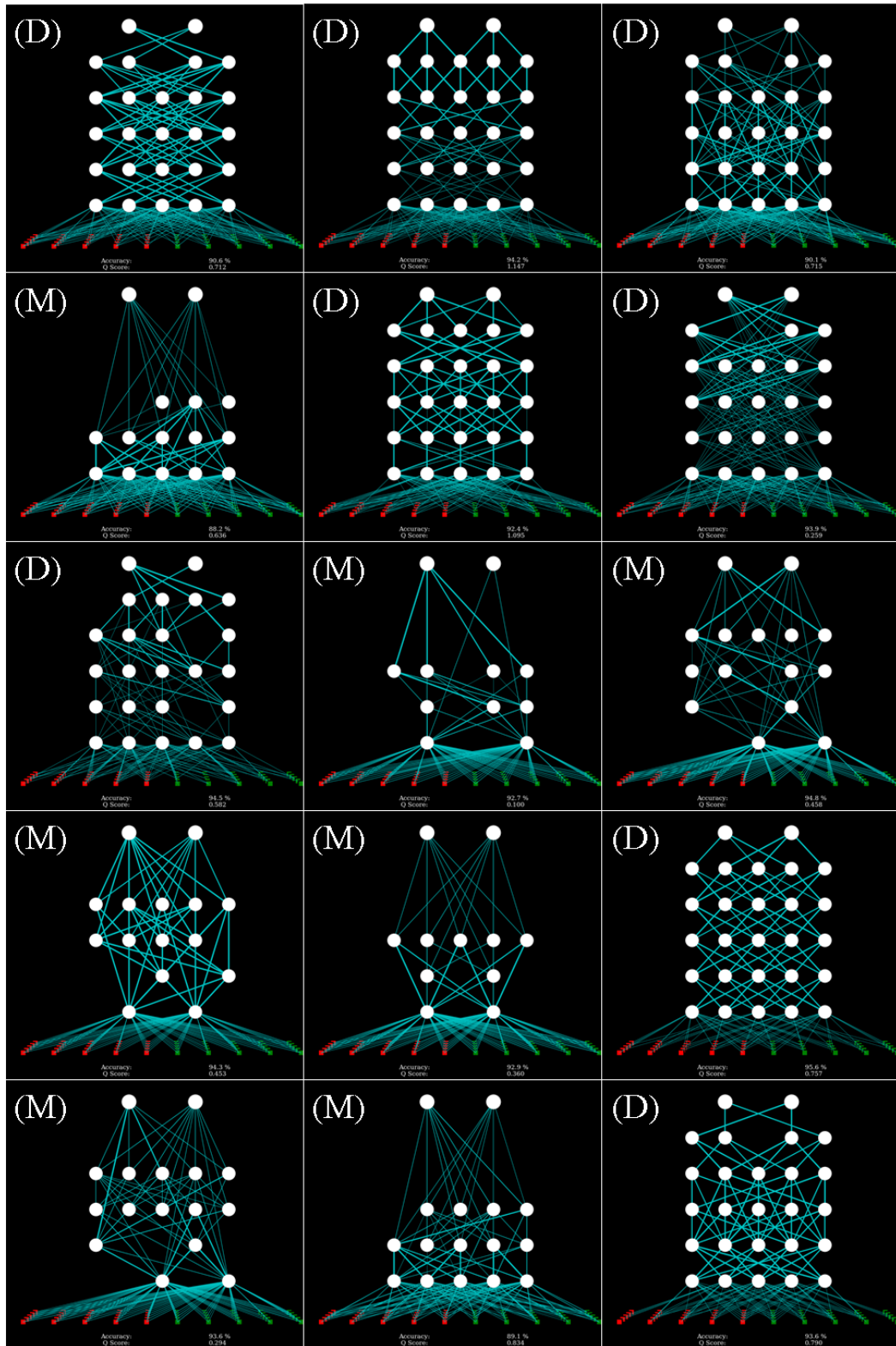


Figure 47. Experiment 1, geometrically separable task, FIC's final networks (16-30). The 'dense' motif (marked *D*) appears in 53.3% of runs. The 'messy' motif (marked *M*) appears in 40% of runs. The 'half' motif (marked *H*) appears in 6.7% of runs.

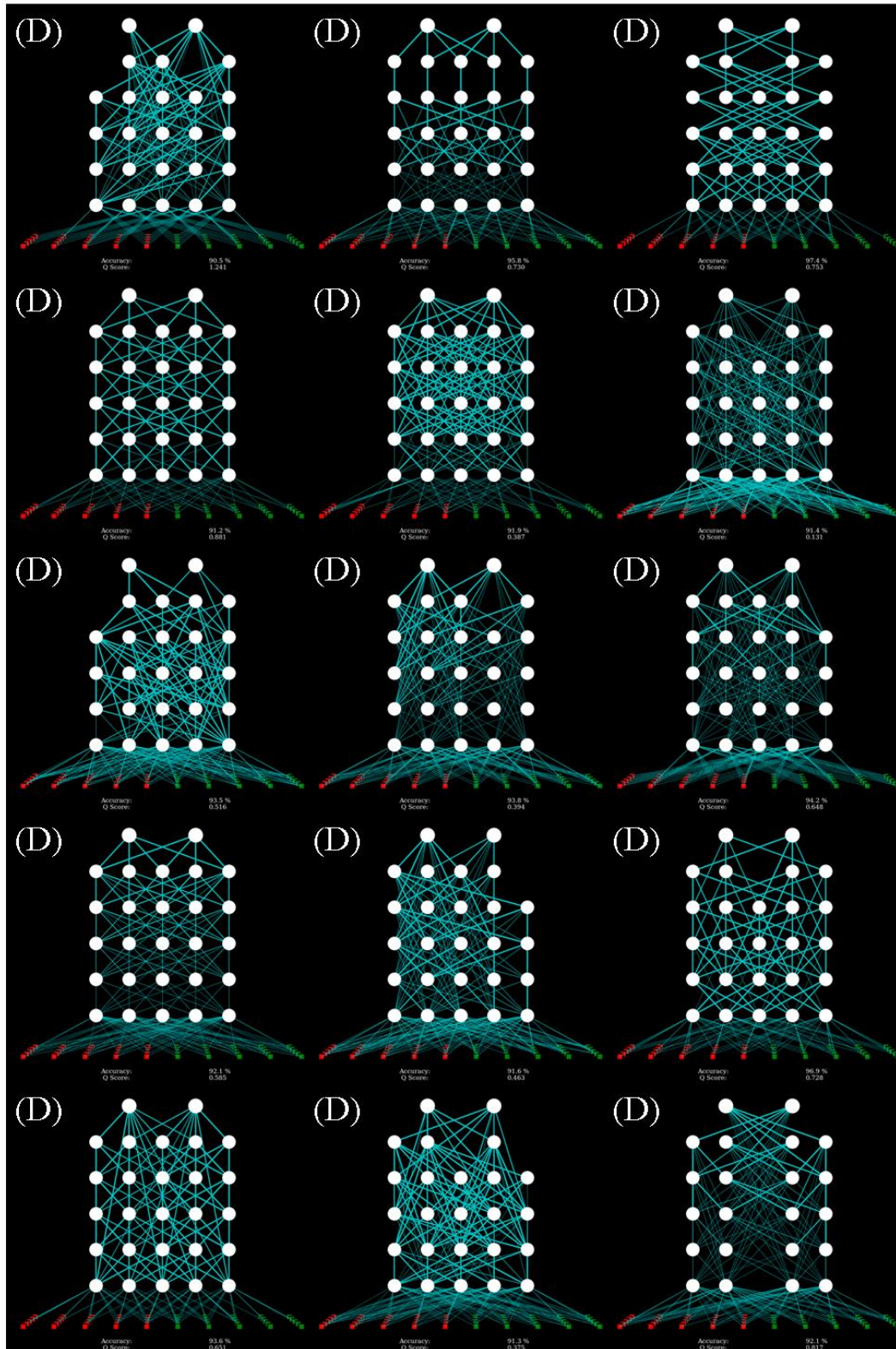


Figure 48. Experiment 1, geometrically separable task, EIC's final networks (1-15). The 'dense' motif (marked *D*) appears in 100% of runs.

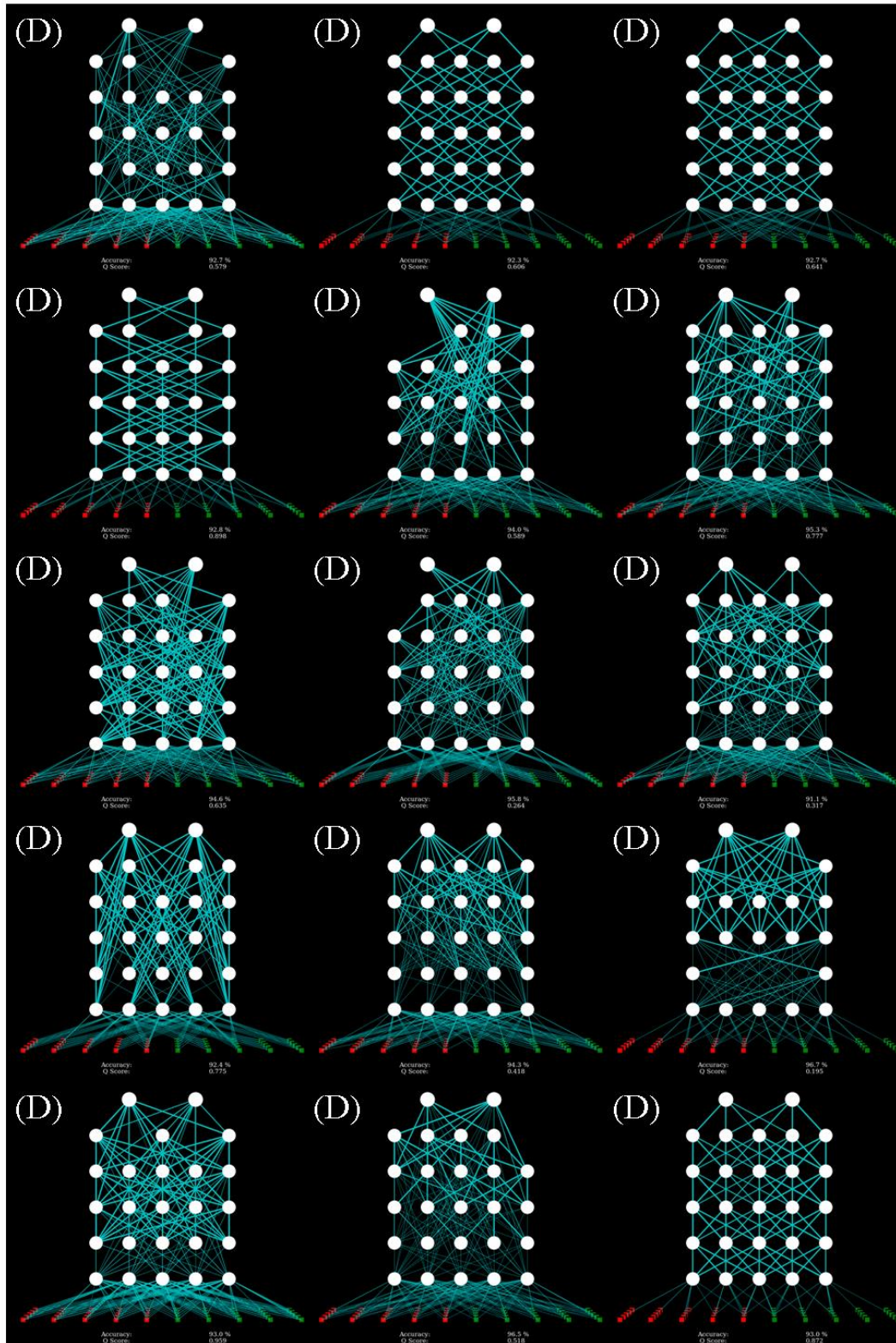


Figure 49. Experiment 1, geometrically separable task, EIC's final networks (16-30). The 'dense' motif (marked *D*) appears in 100% of runs.

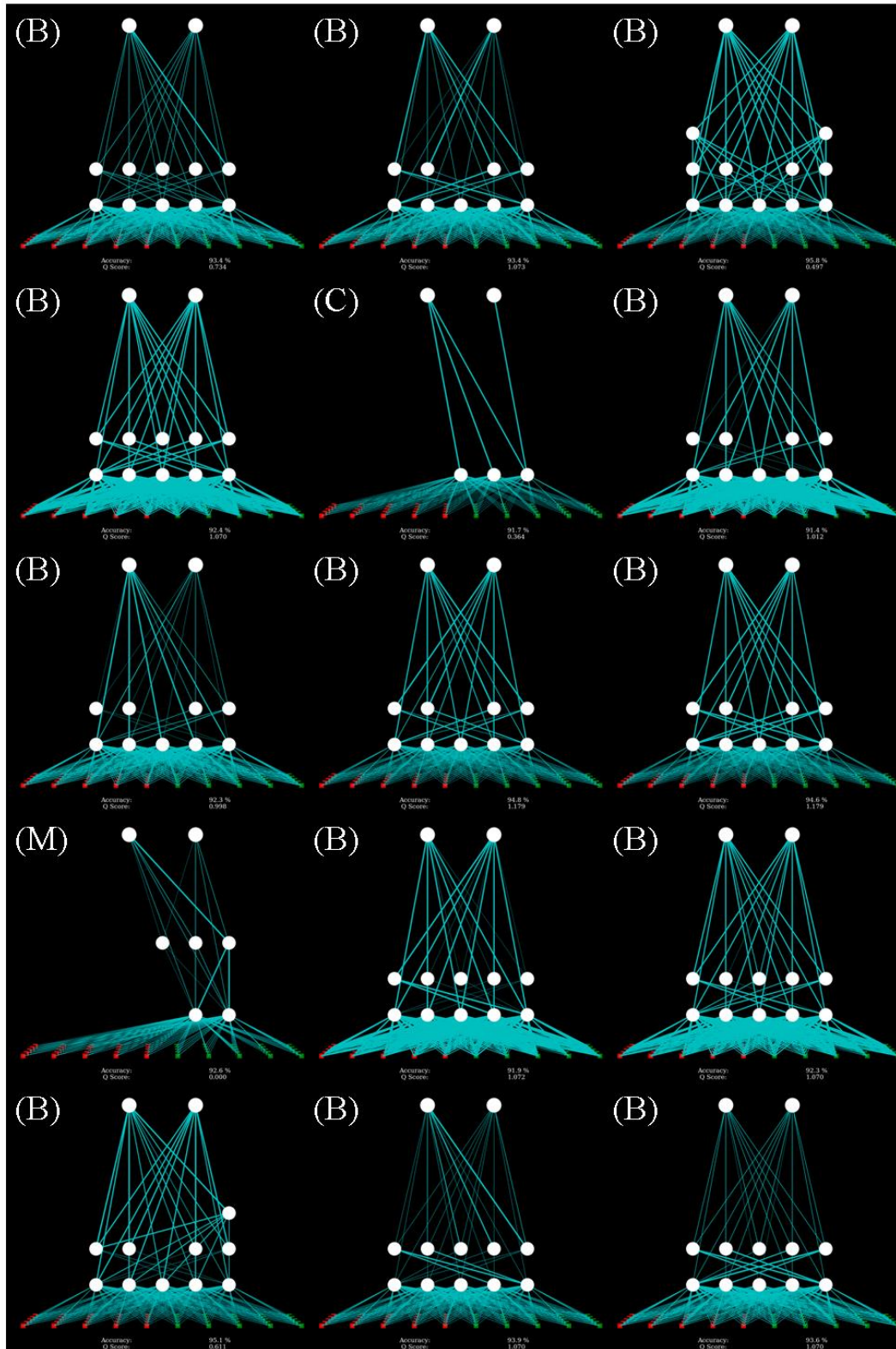


Figure 50. Experiment 1, geometrically separable task, GIC's final networks (1-15). The 'block' motif (marked *B*) appears in 83.3% of runs. The 'collapsed' motif (marked *C*) appears in 16.7% of runs.

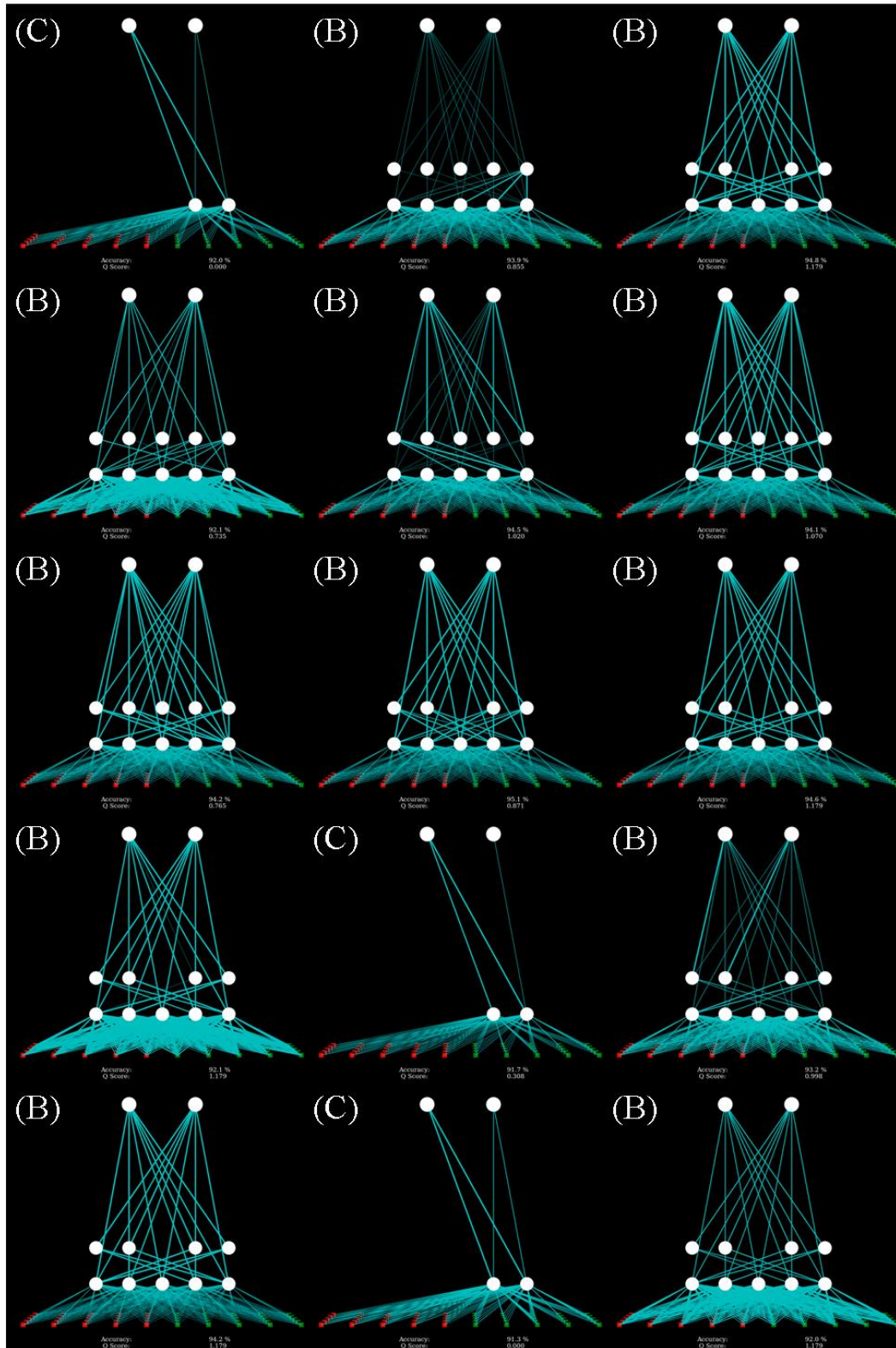


Figure 51. Experiment 1, geometrically separable task, GIC's final networks (16-30). The 'block' motif (marked *B*) appears in 83.3% of runs. The 'collapsed' motif (marked *C*) appears in 16.7% of runs.

B.2 Experiment 1 – Geometrically Inseparable Task

Table 6. Percentile data and Mann-Whitney U-Tests & P-values for experiment 1 (geometrically inseparable task).

	LCC		GCC		FIC		EIC		GIC	
	U	P	U	P	U	P	U	P	U	P
LCC	-	-	397	0.282	389	0.184	409	0.27	290	0.014
GCC	397	0.282	-	-	304	0.023	403	0.314	165	<0.001
FIC	389	0.184	304	0.023	-	-	320	0.027	353	0.107
EIC	409	0.27	403	0.314	320	0.027	-	-	279	0.009
GIC	290	0.014	165	<0.001	353	0.107	279	0.009	-	-
Q-score percentiles										
25%	0.613		0.953		0.501		0.685		0.308	
50%	1.131		1.131		0.675		0.917		0.864	
75%	1.330		1.324		1.092		1.526		0.864	
Task performance percentiles (%)										
25%	95.800		95.720		98.325		97.785		96.640	
50%	96.910		96.000		98.400		98.250		96.900	
75%	98.200		97.140		99.200		99.135		97.180	

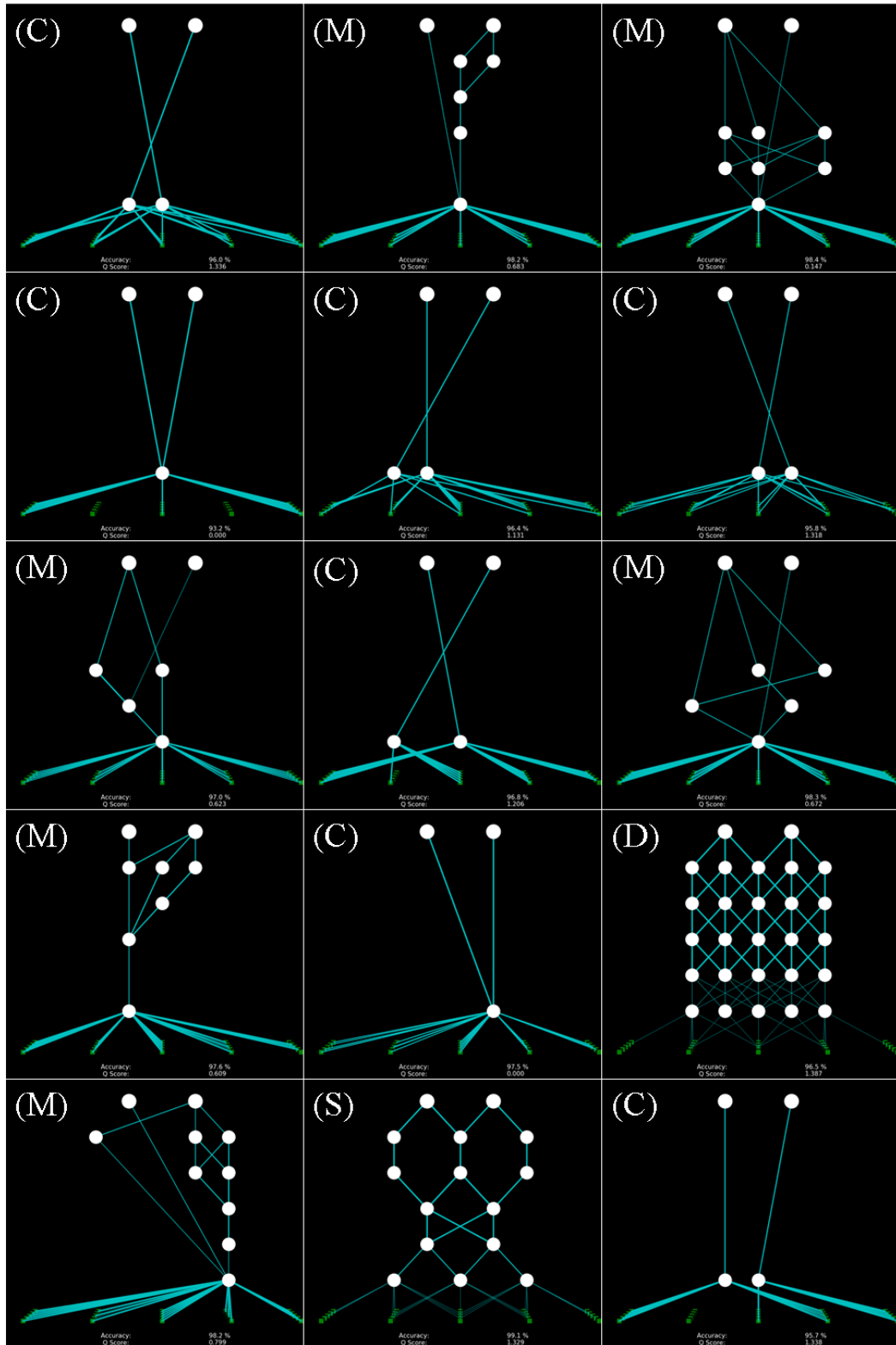


Figure 52. Experiment 1, geometrically inseparable task, LCC's final networks (1-15). The 'collapsed' motif (marked C) appears in 50% of runs. The 'messy' motif (marked M) appears in 36.7% of runs. The 'sparse' motif (marked S) appears in 10% of runs. The 'dense' motif (marked D) appears in 3.3% of runs.

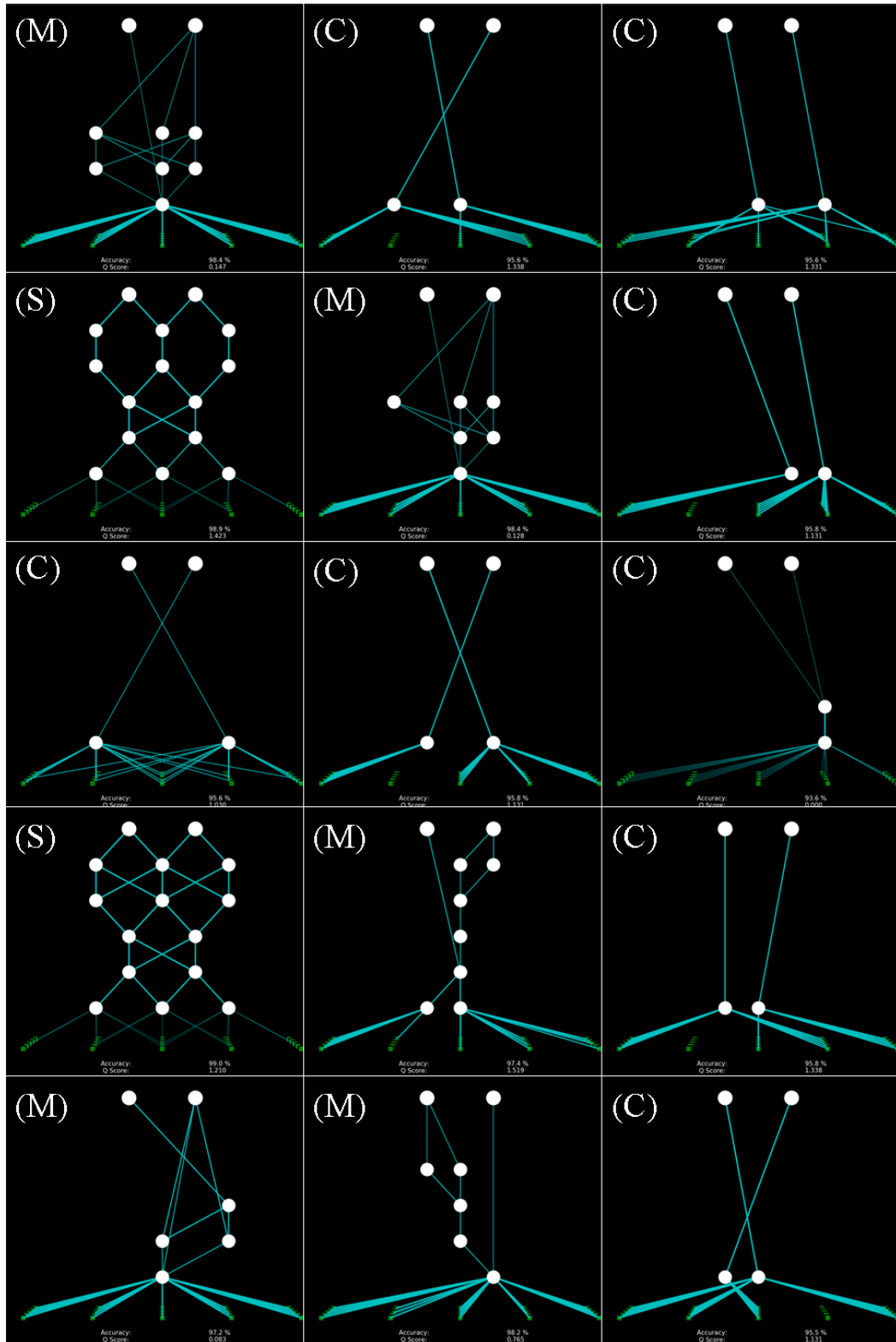


Figure 53. Experiment 1, geometrically inseparable task, LCC's final networks (1-15). The 'collapsed' motif (marked C) appears in 50% of runs. The 'messy' motif (marked M) appears in 36.7% of runs. The 'sparse' motif (marked S) appears in 10% of runs. The 'dense' motif (marked D) appears in 3.3% of runs.

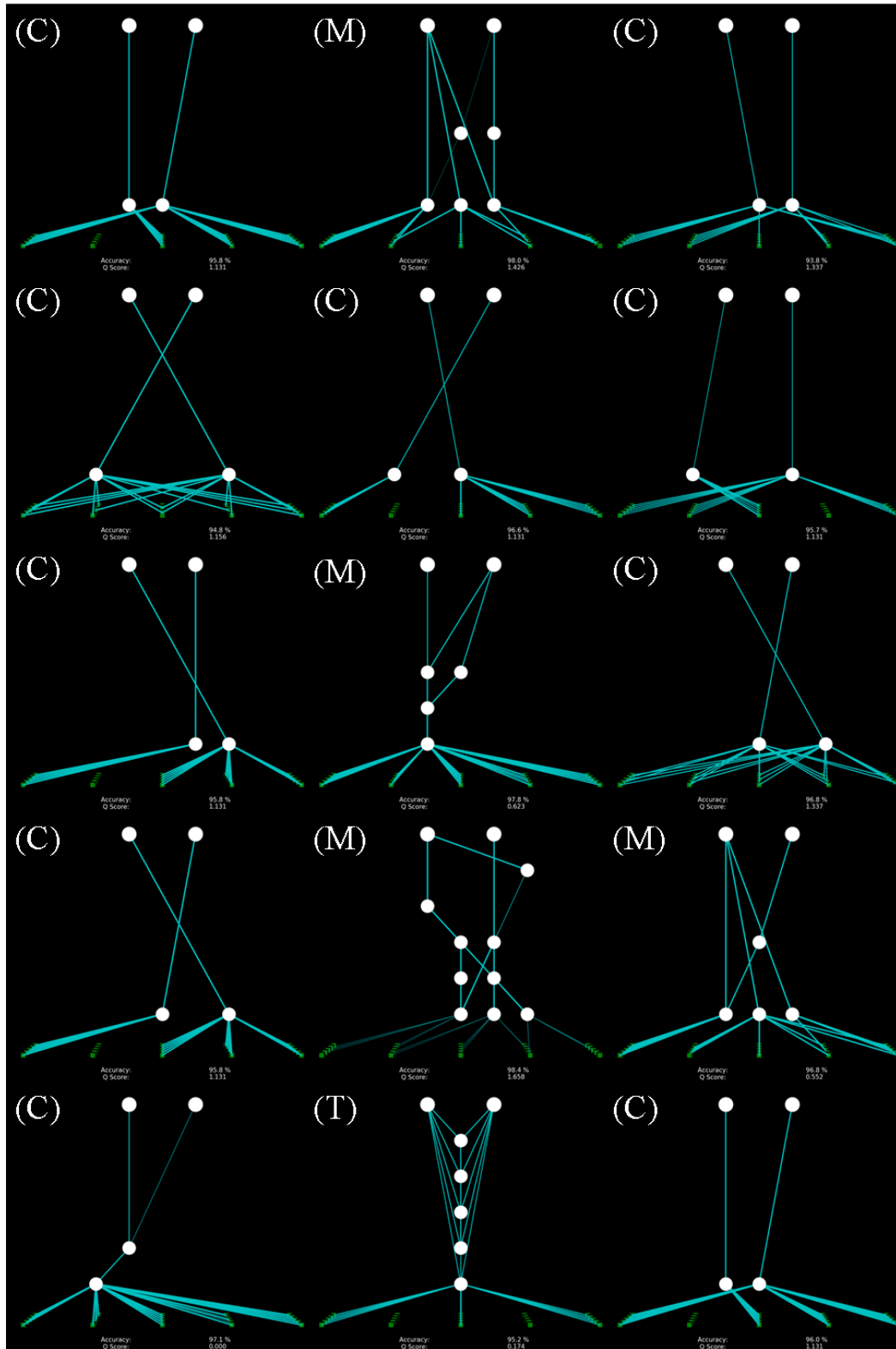


Figure 54. Experiment 1, geometrically inseparable task, GCC's final networks (1-15). The 'collapsed' motif (marked C) appears in 76.7% of runs. The 'messy' motif (marked M) appears in 20% of runs. The 'tower' motif appears in 3.3% of runs.

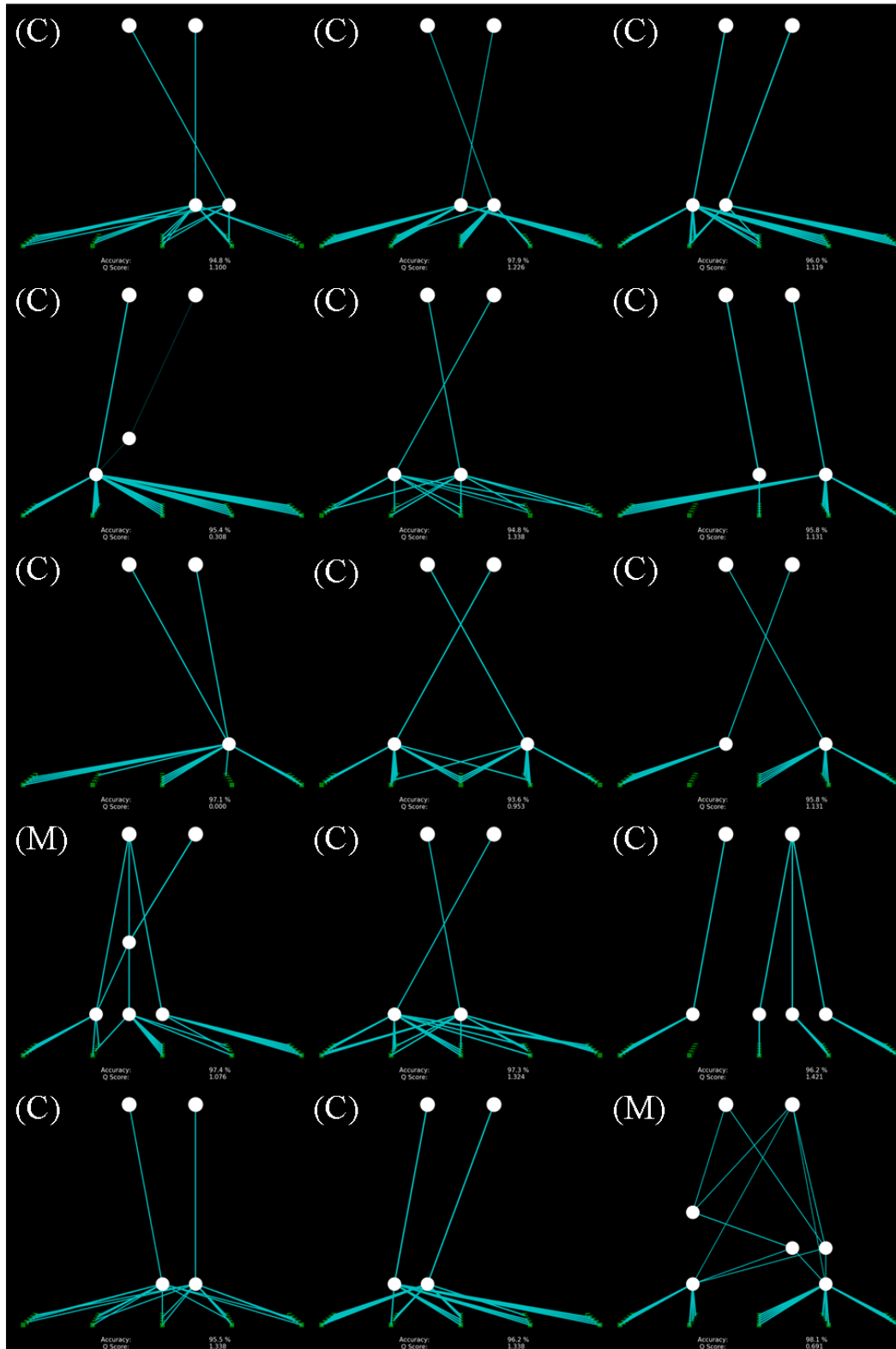


Figure 55. Experiment 1, geometrically inseparable task, GCC's final networks (16-30). The 'collapsed' motif (marked C) appears in 76.7% of runs. The 'messy' motif (marked M) appears in 20% of runs. The 'tower' motif appears in 3.3% of runs.

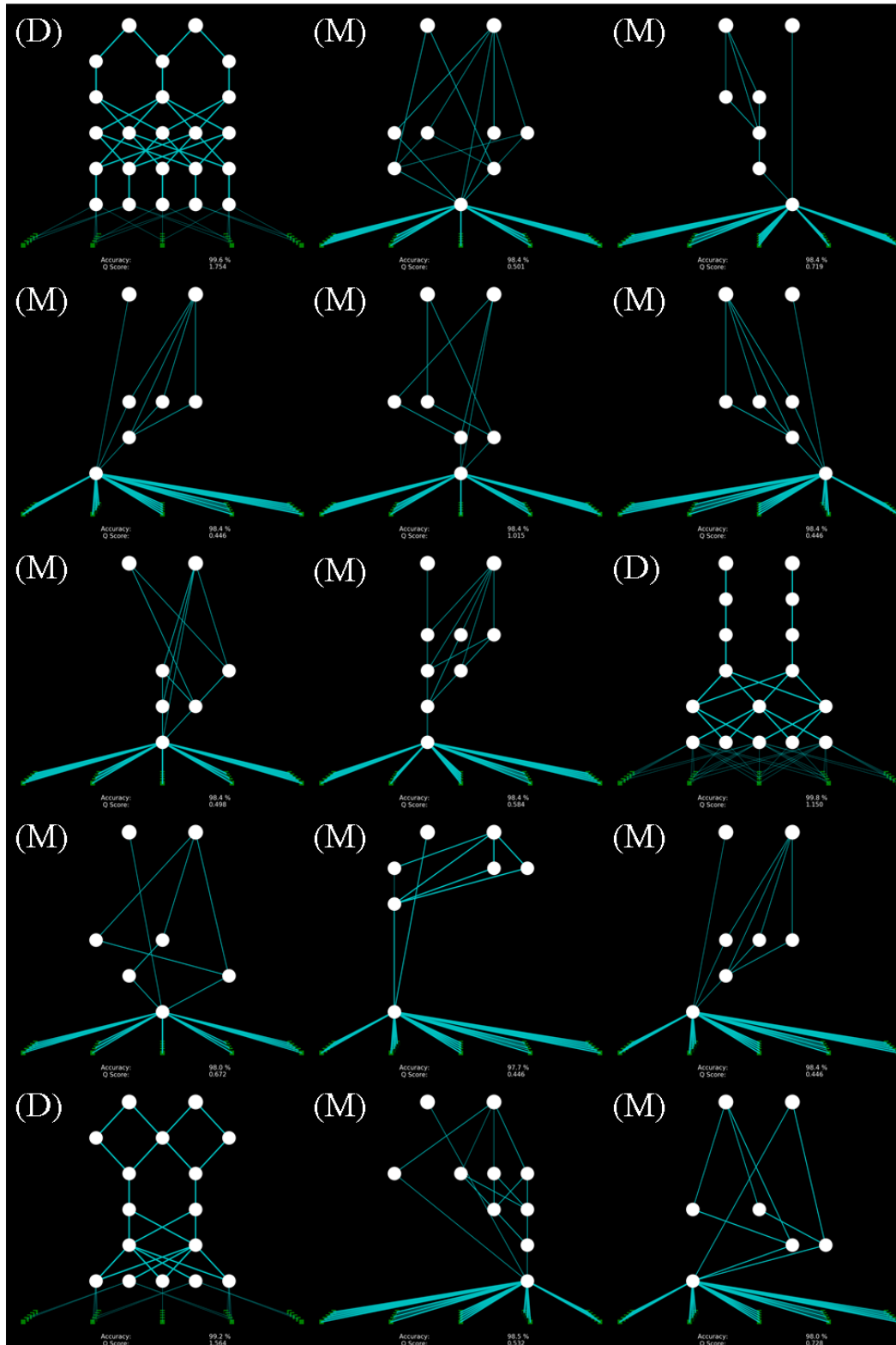


Figure 56. Experiment 1, geometrically inseparable task, FIC’s final networks (1-15). The ‘messy’ motif (marked *M*) appears in 73.3% of runs. The ‘dense’ motif (marked *D*) appears in 23.3% of runs.

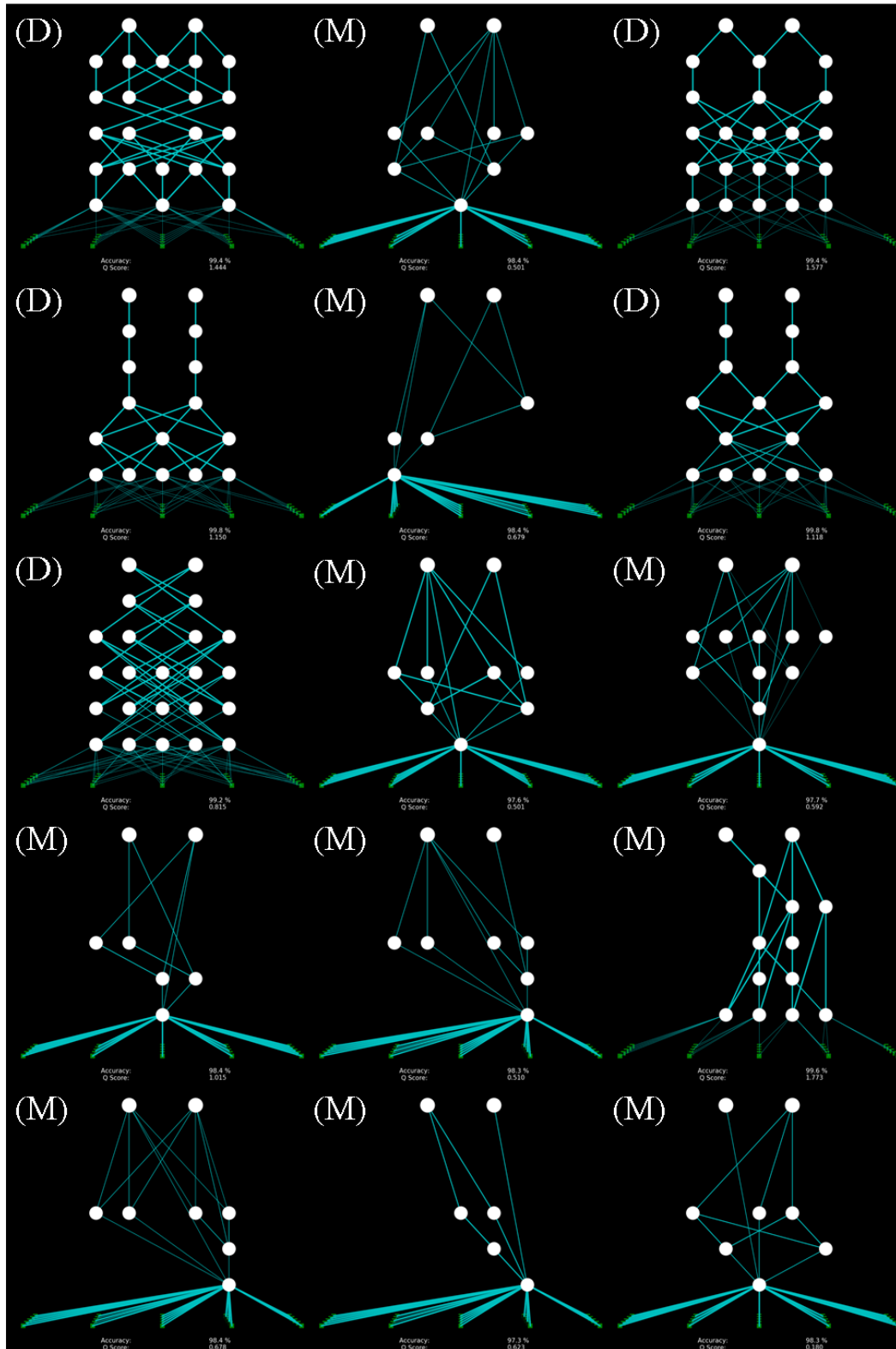


Figure 57. Experiment 1, geometrically inseparable task, FIC's final networks (16-30). The 'messy' motif (marked *M*) appears in 73.3% of runs. The 'dense' motif (marked *D*) appears in 23.3% of runs.

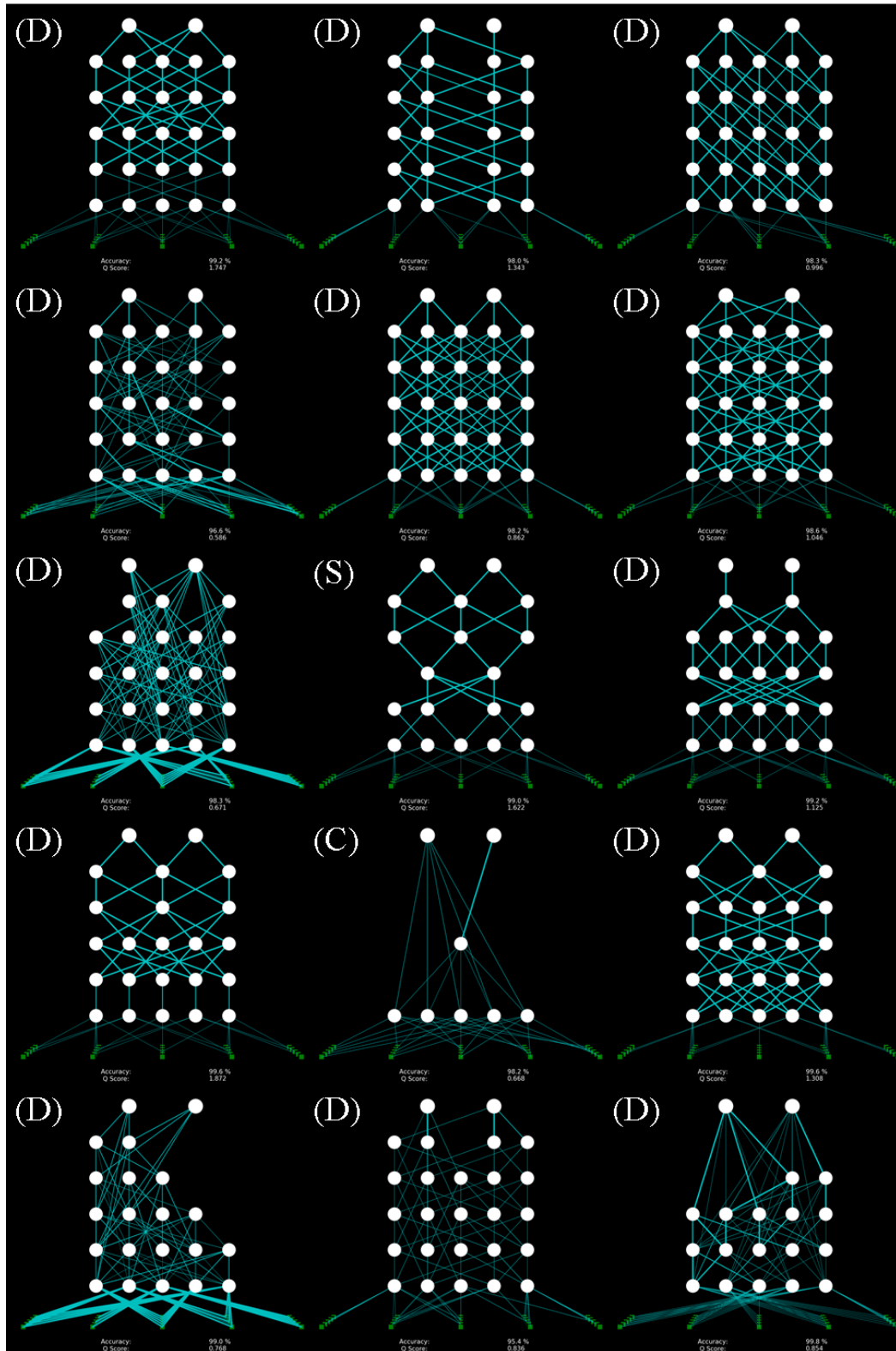


Figure 58. Experiment 1, geometrically inseparable task, EIC's final networks (1-15). The 'dense' motif (marked *D*) appears in 73.3% of runs. The 'messy' motif (marked *M*) appears in 10% of runs. The 'collapsed' motif (marked *C*) appears in 6.7% of runs. The 'sparse' motif (marked *S*) appears in 3.3% of runs. The 'block' motif (marked *B*) appears in 3.3% of runs.

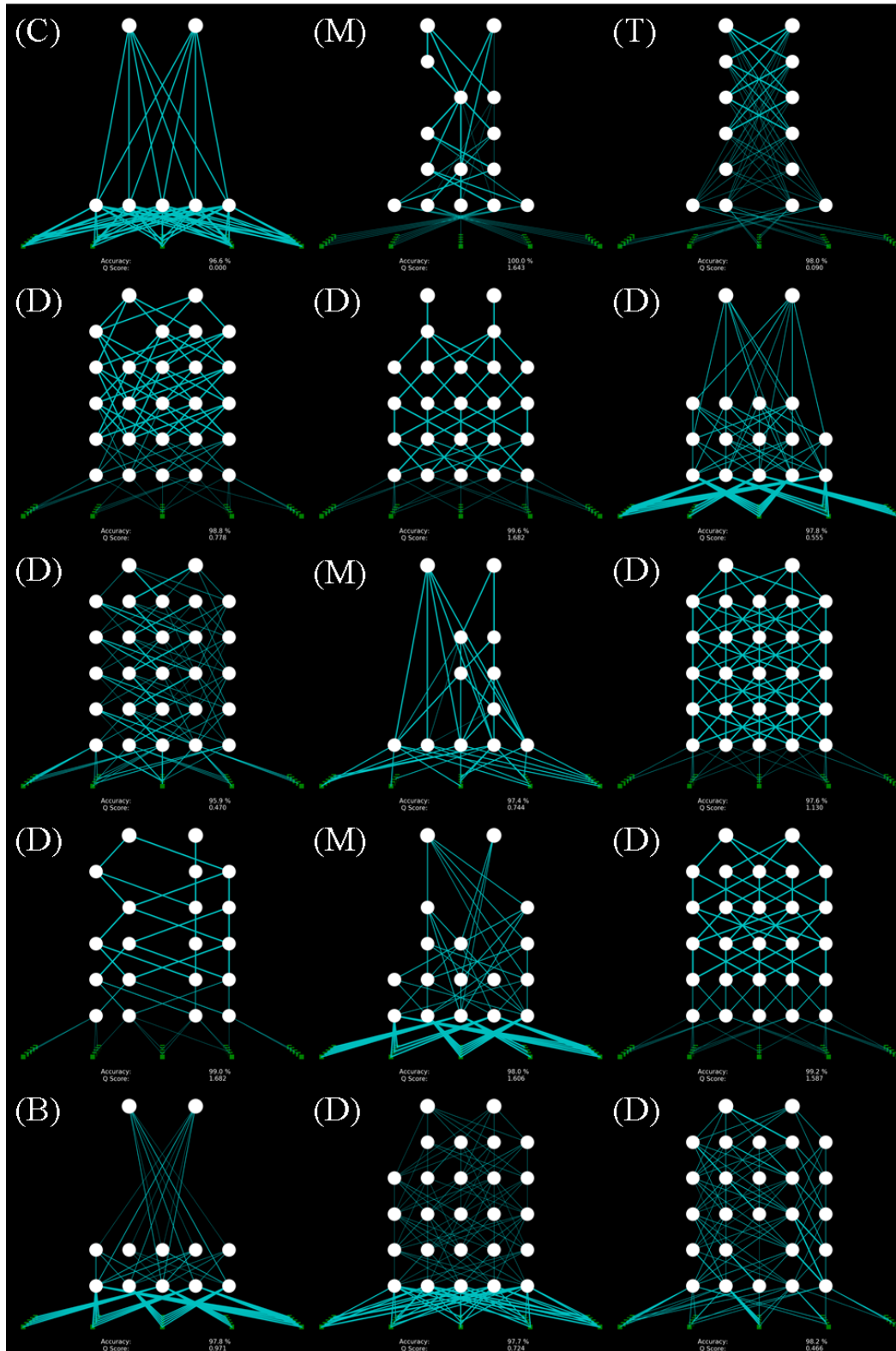


Figure 59. Experiment 1, geometrically inseparable task, EIC's final networks (16-30). The 'dense' motif (marked *D*) appears in 73.3% of runs. The 'messy' motif (marked *M*) appears in 10% of runs. The 'collapsed' motif (marked *C*) appears in 6.7% of runs. The 'sparse' motif (marked *S*) appears in 3.3% of runs. The 'block' motif (marked *B*) appears in 3.3% of runs.

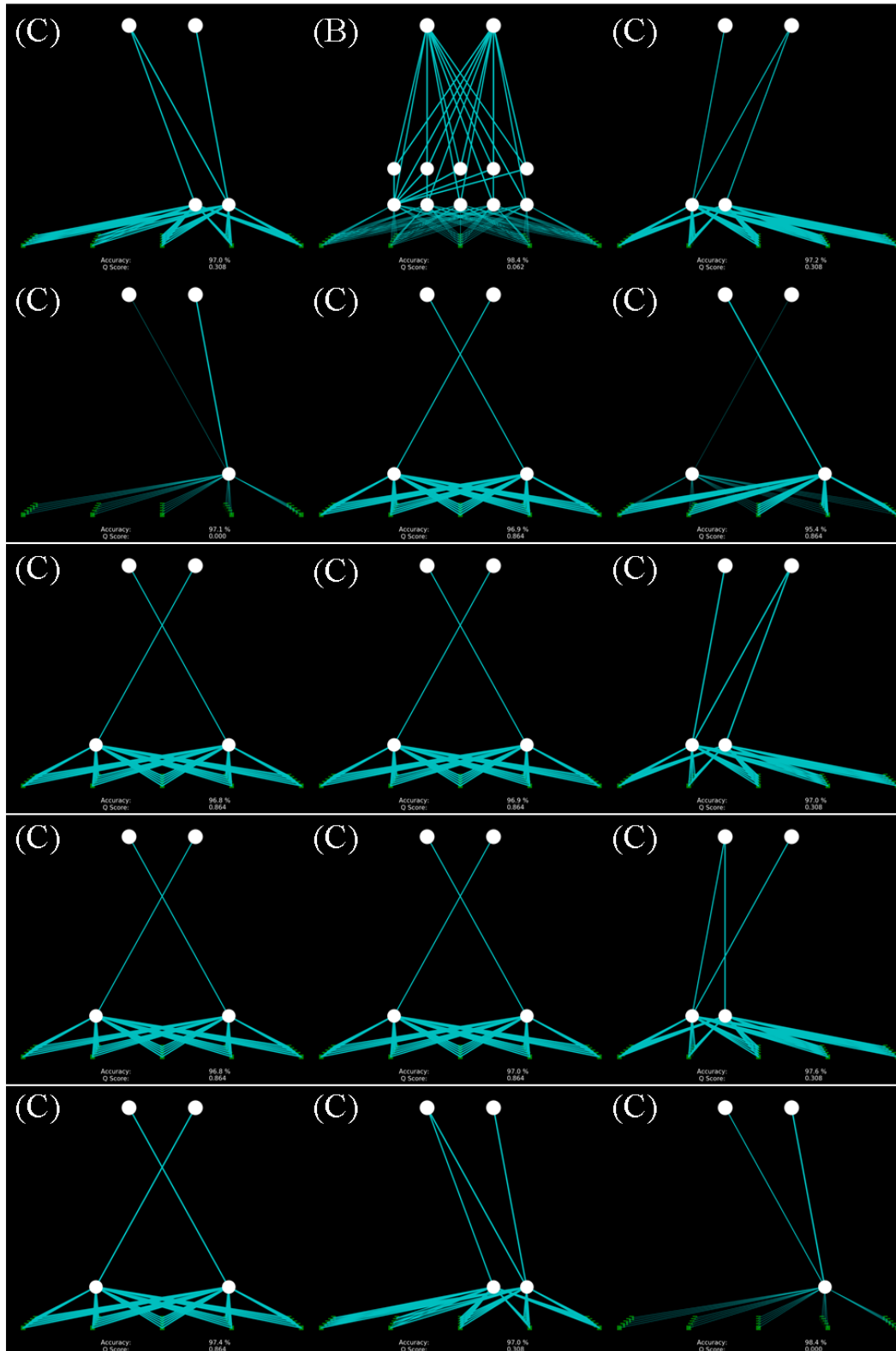


Figure 60. Experiment 1, geometrically inseparable task, GIC's final networks (1-15). The 'collapsed' motif (marked C) appears in 93.3% of runs. The 'block' motif (marked B) appears in 6.7% of runs.

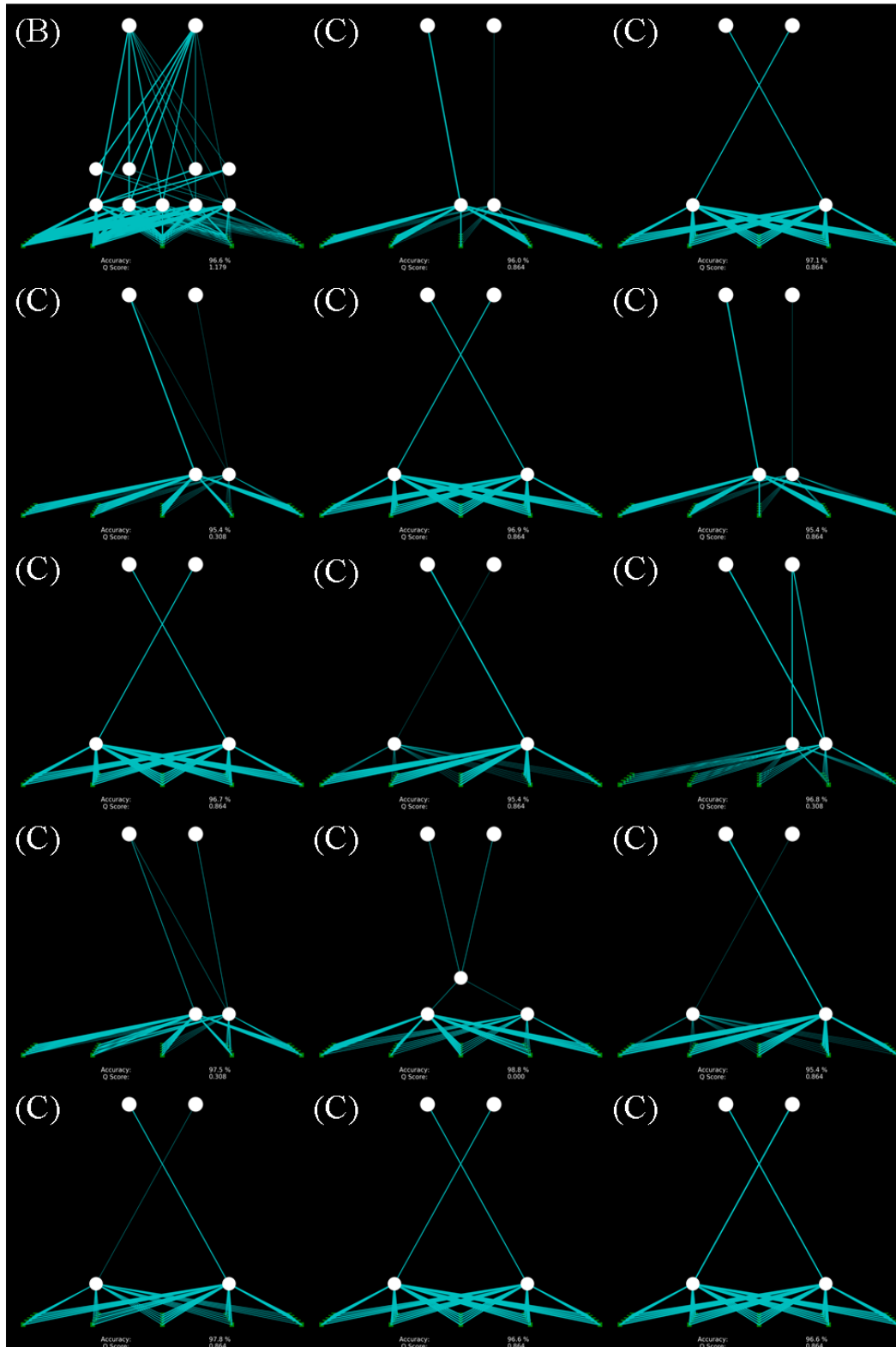


Figure 61. Experiment 1, geometrically inseparable task, GIC's final networks (16-30). The 'collapsed' motif (marked C) appears in 93.3% of runs. The 'block' motif (marked B) appears in 6.7% of runs.

B.3 Experiment 1 – Without Supralayer Connections

Table 7. Percentile data and Mann-Whitney U-Tests & P-values for experiment 1 (geometrically inseparable task, sans supralayer connections).

	LCC		GCC		FIC		EIC		GIC	
	U	P	U	P	U	P	U	P	U	P
LCC	-	-	402	0.308	197	<0.001	188	<0.001	399	0.223
GCC	402	0.308	-	-	210	0.001	191	<0.001	357	0.118
FIC	197	<0.001	210	0.001	-	-	400	0.462	194	<0.001
EIC	188	<0.001	191	<0.001	400	0.462	-	-	199	<0.001
GIC	399	0.223	357	0.118	194	<0.001	199	<0.001	-	-
Q-score percentiles										
25%	0.081		0.095		0.157		0.197		0.023	
50%	0.095		0.095		0.296		0.388		0.095	
75%	0.248		0.242		0.536		0.519		0.195	
Task performance percentiles (%)										
25%	86.610		59.240		88.250		90.020		92.805	
50%	88.480		87.140		90.240		91.200		94.180	
75%	94.570		88.760		91.805		92.520		94.555	

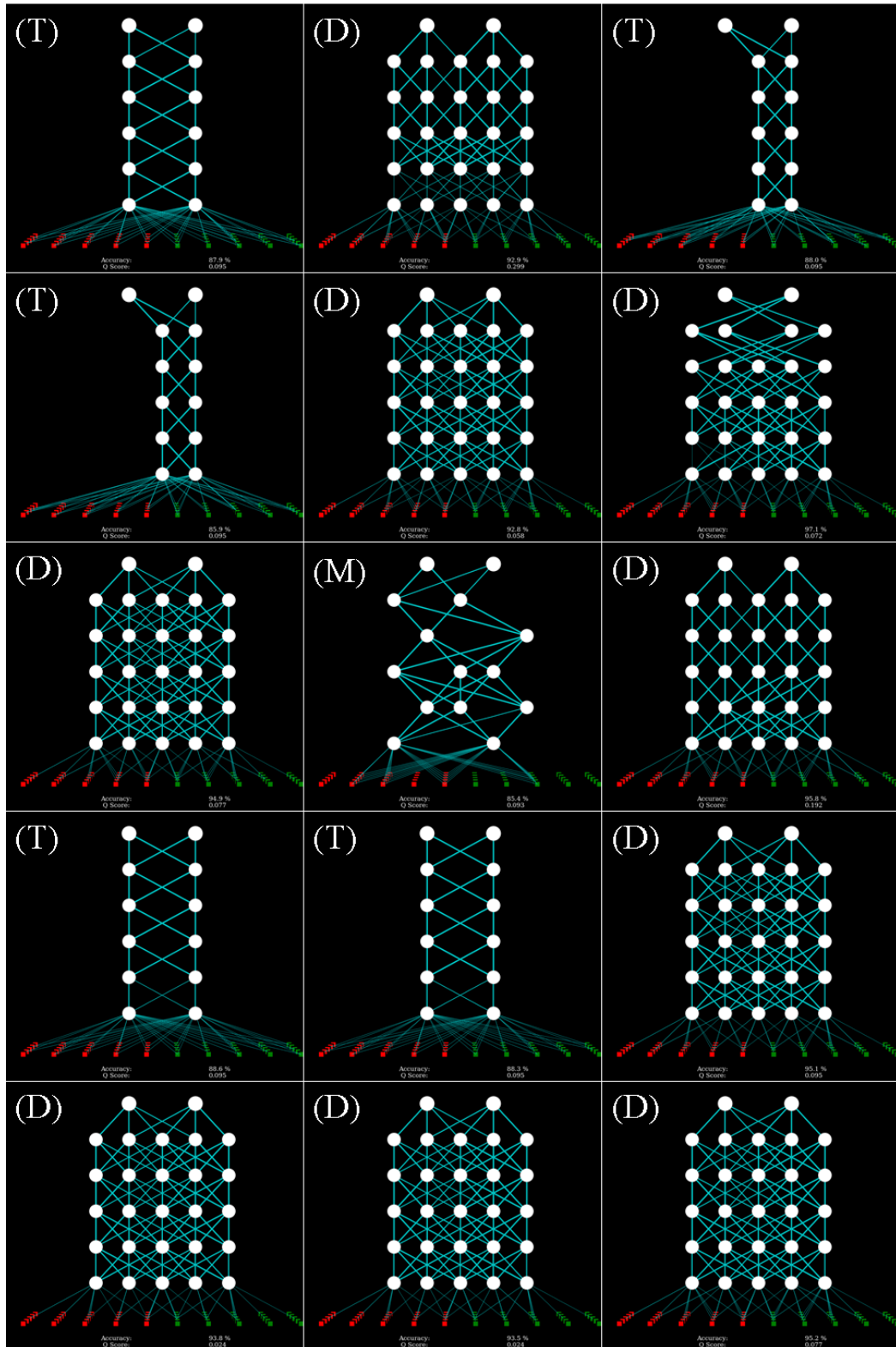


Figure 62. Experiment 1, geometrically inseparable task, without supralayer connections, LCC's final networks (1-15). The 'dense' motif (marked *D*) appears in 46.7% of runs. The 'tower' motif (marked *T*) appears in 26.7% of runs. The 'messy' motif (marked *M*) appears in 16.7% of runs. The 'null' motif (marked *N*) appears in 10% of runs.

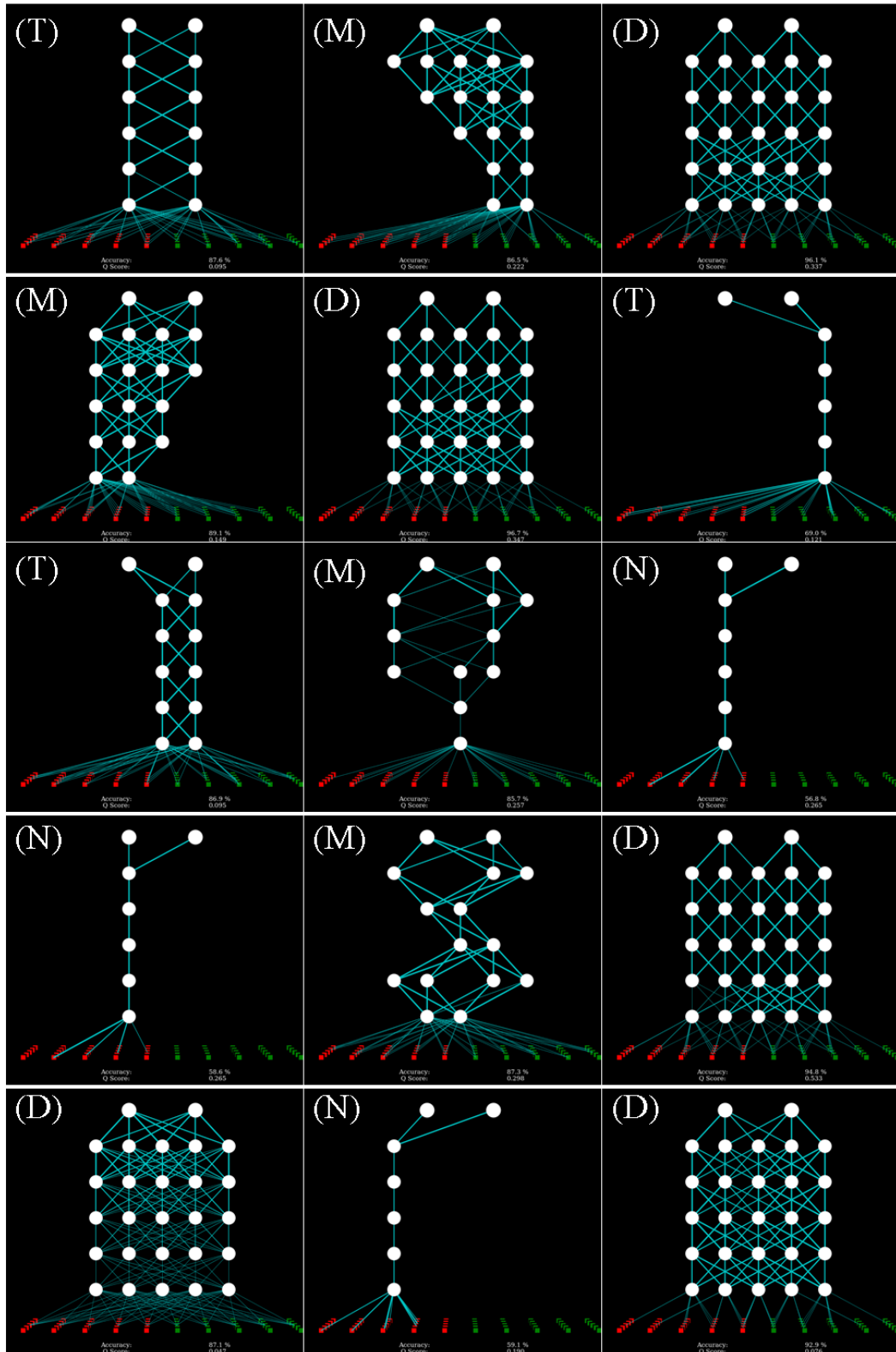


Figure 63. Experiment 1, geometrically inseparable task, without supralayer connections, LCC's final networks (16-30). The 'dense' motif (marked *D*) appears in 46.7% of runs. The 'tower' motif (marked *T*) appears in 26.7% of runs. The 'messy' motif (marked *M*) appears in 16.7% of runs. The 'null' motif (marked *N*) appears in 10% of runs.

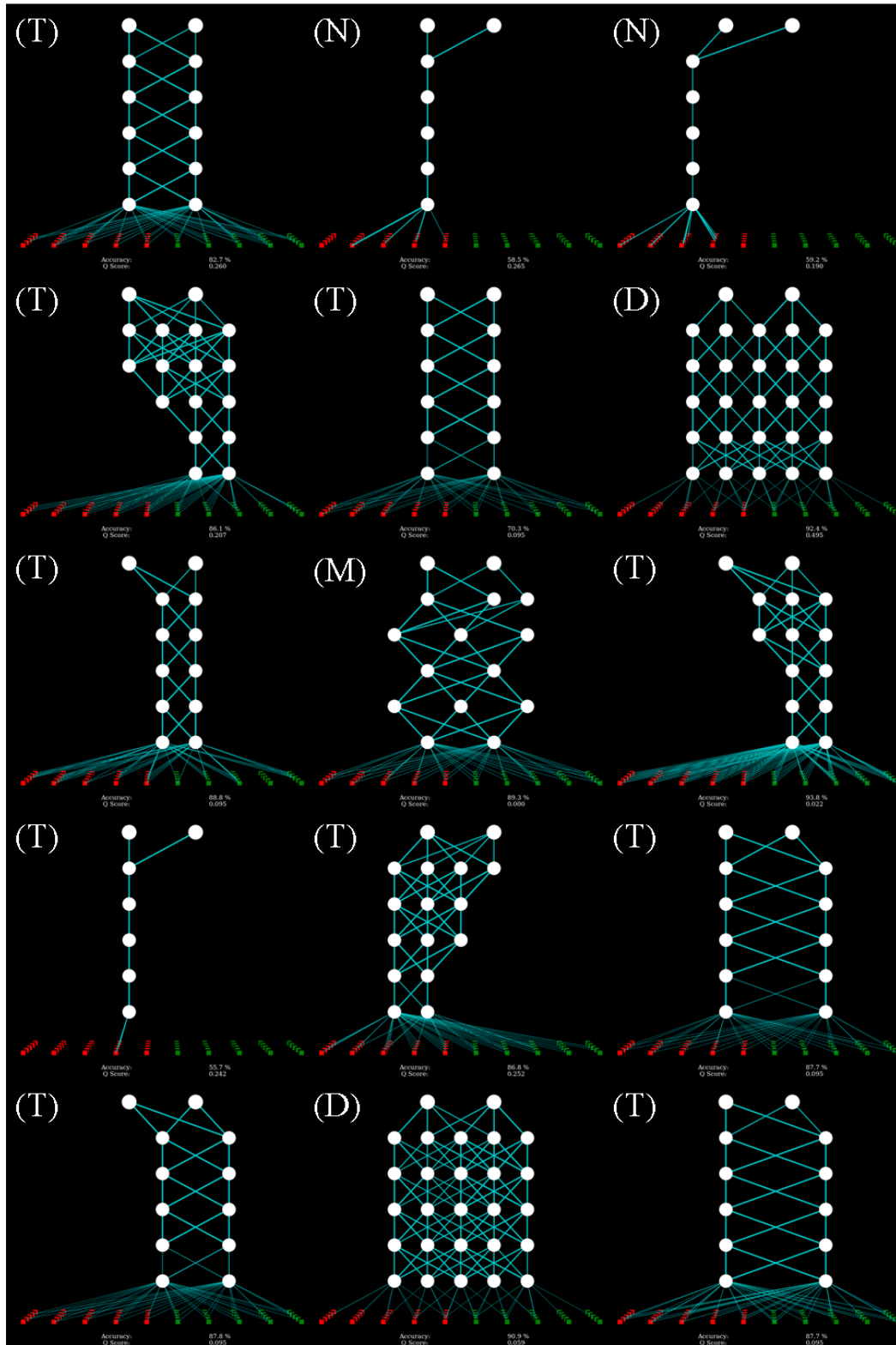


Figure 64. Experiment 1, geometrically inseparable task, without supralayer connections, GCC's final networks (1-15). The 'tower' motif (marked *T*) appears in 53.3% of runs. The 'null' motif (marked *N*) appears in 30% of runs. The 'dense' motif (marked *D*) appears in 10% of runs. The 'messy' motif (marked *M*) appears in 6.7% of runs.

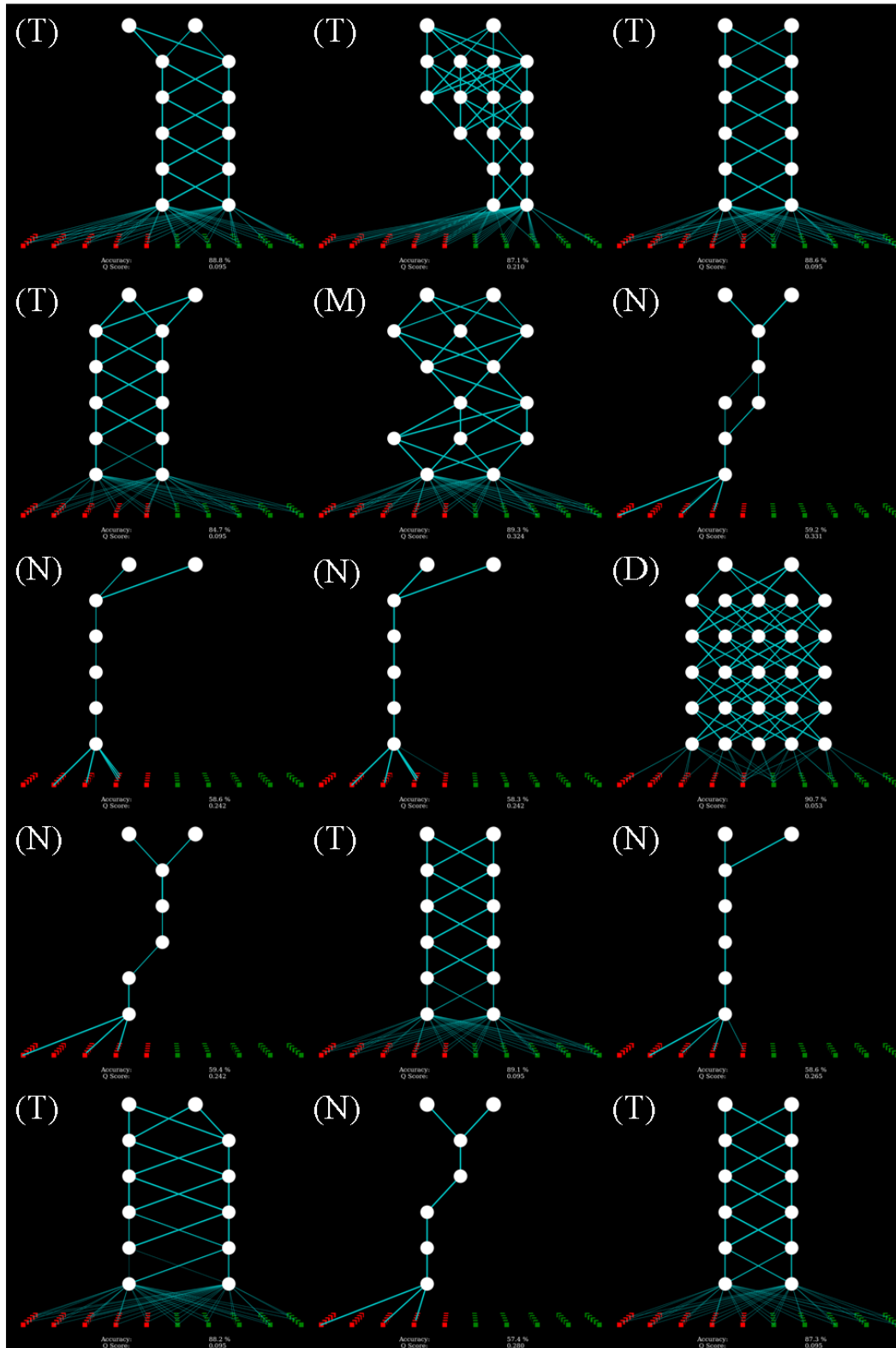


Figure 65. Experiment 1, geometrically inseparable task, without supralayer connections, GCC's final networks (16-30). The 'tower' motif (marked *T*) appears in 53.3% of runs. The 'null' motif (marked *N*) appears in 30% of runs. The 'dense' motif (marked *D*) appears in 10% of runs. The 'messy' motif (marked *M*) appears in 6.7% of runs.

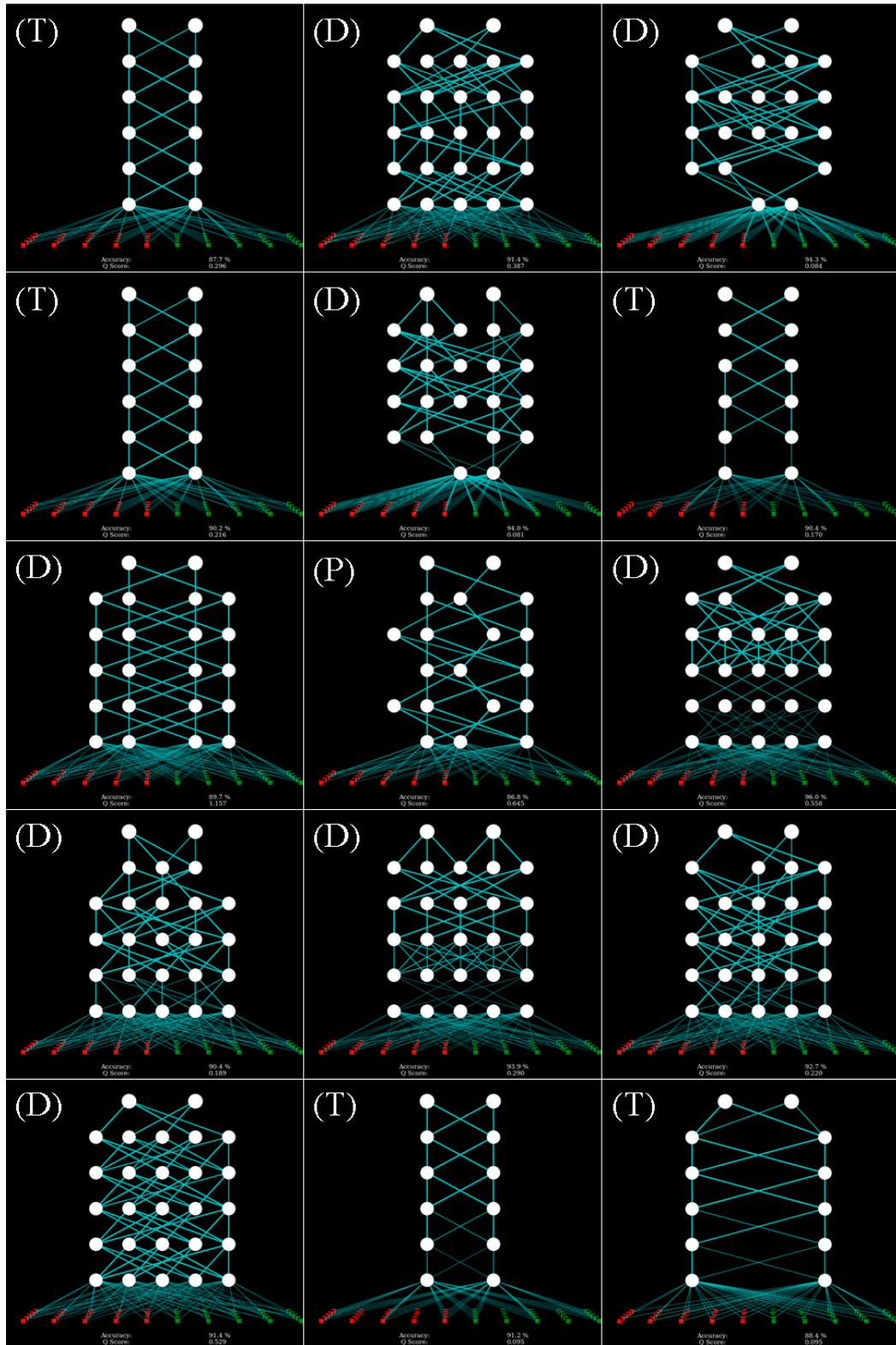


Figure 66. Experiment 1, geometrically inseparable task, without supralayer connections, FIC's final networks (1-15). The 'dense' motif (marked *D*) appears in 46.7%. The 'tower' motif (marked *T*) appears in 26.7%. The 'pleated' motif (marked *P*) appears in 16.7%. The 'null' motif (marked *N*) appears in 6.7%.

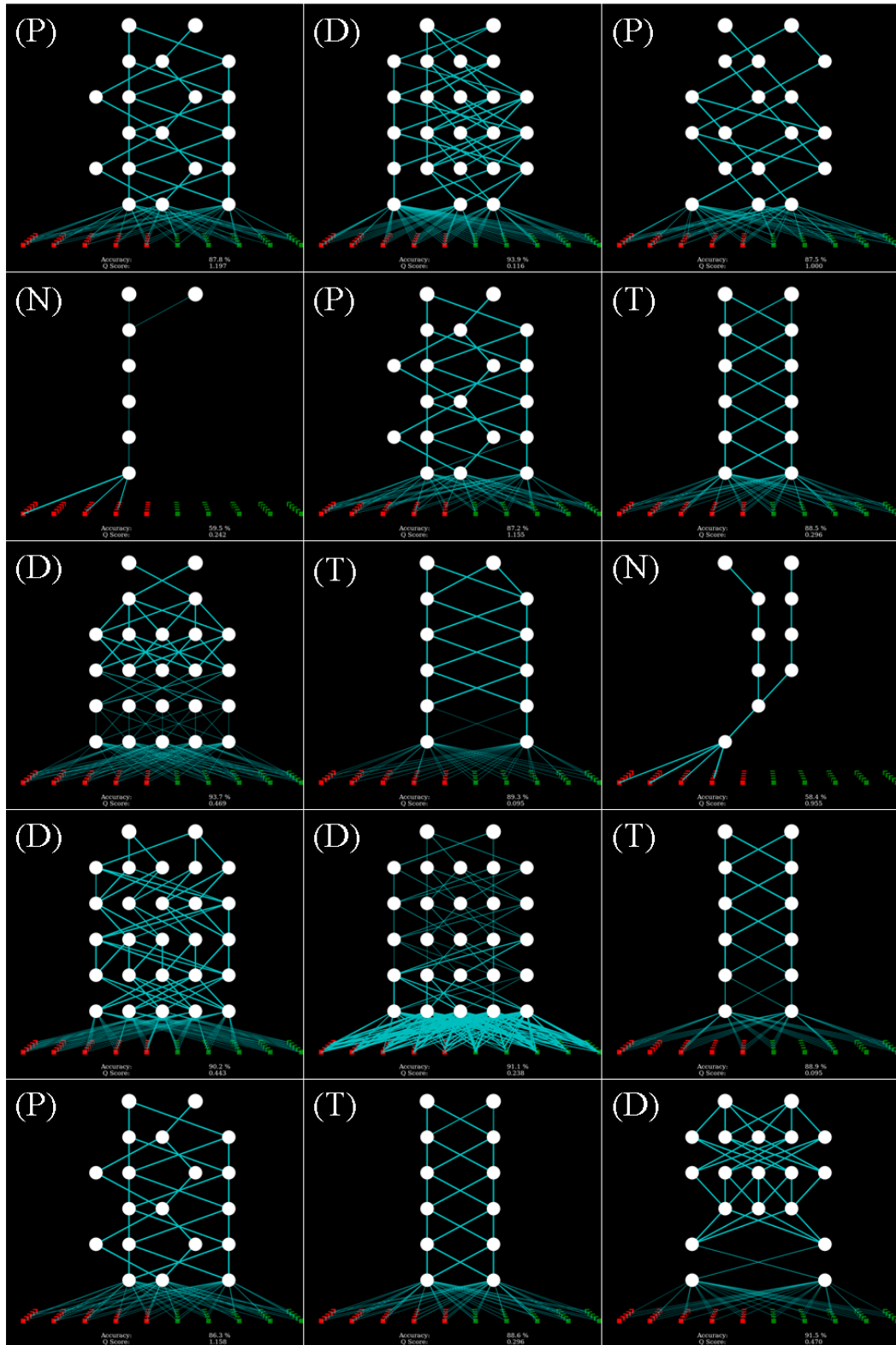


Figure 67. Experiment 1, geometrically inseparable task, without supralayer connections, FIC's final networks (16-30). The 'dense' motif (marked *D*) appears in 46.7%. The 'tower' motif (marked *T*) appears in 26.7%. The 'pleated' motif (marked *P*) appears in 16.7%. The 'null' motif (marked *N*) appears in 6.7%.

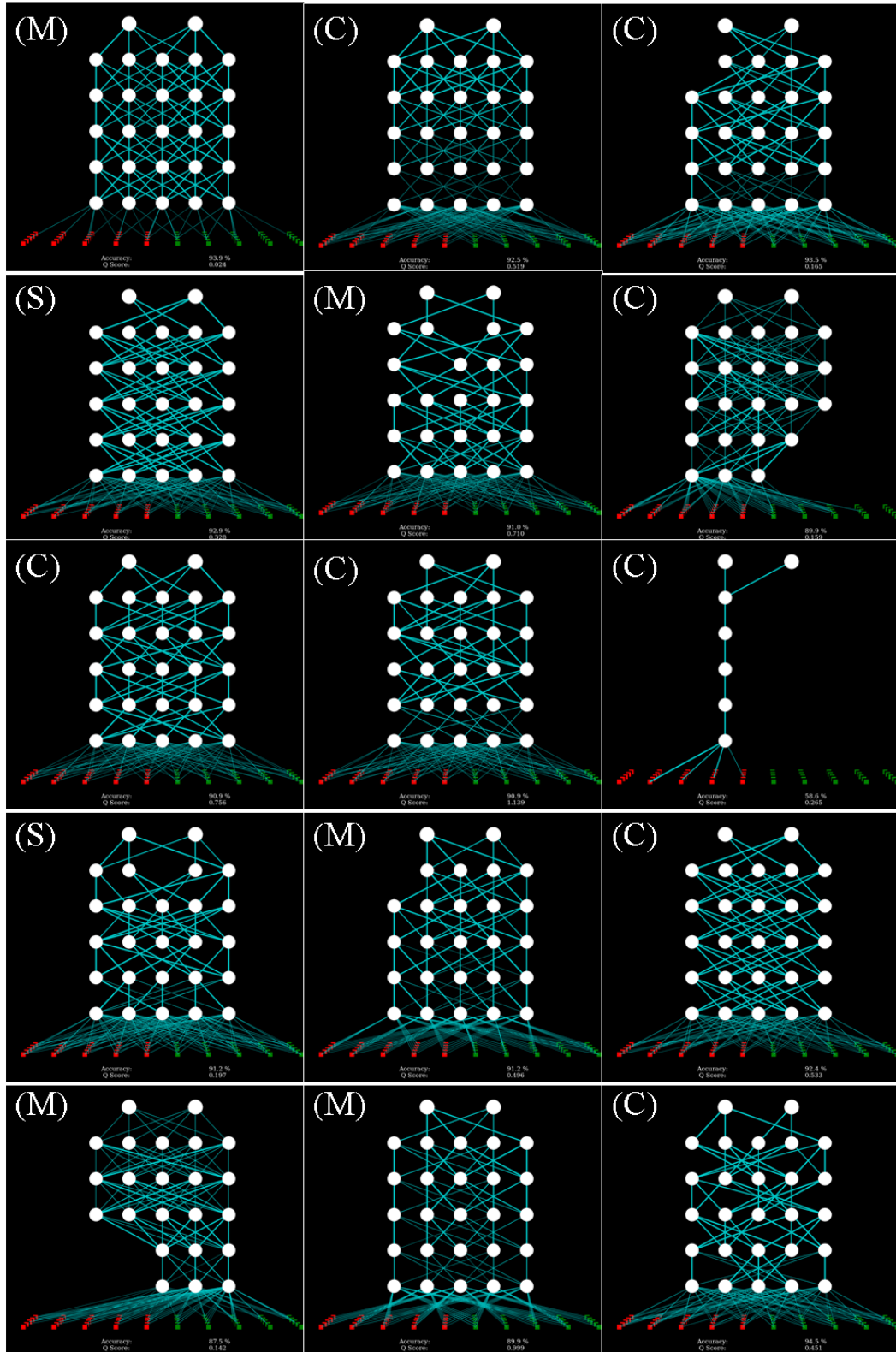


Figure 68. Experiment 1, geometrically inseparable task, without supralayer connections, EIC's final networks (1-15). The 'dense' motif (marked *D*) appears in 93.3% of runs. The 'null' motif (marked *N*) appears in 6.7% of runs.

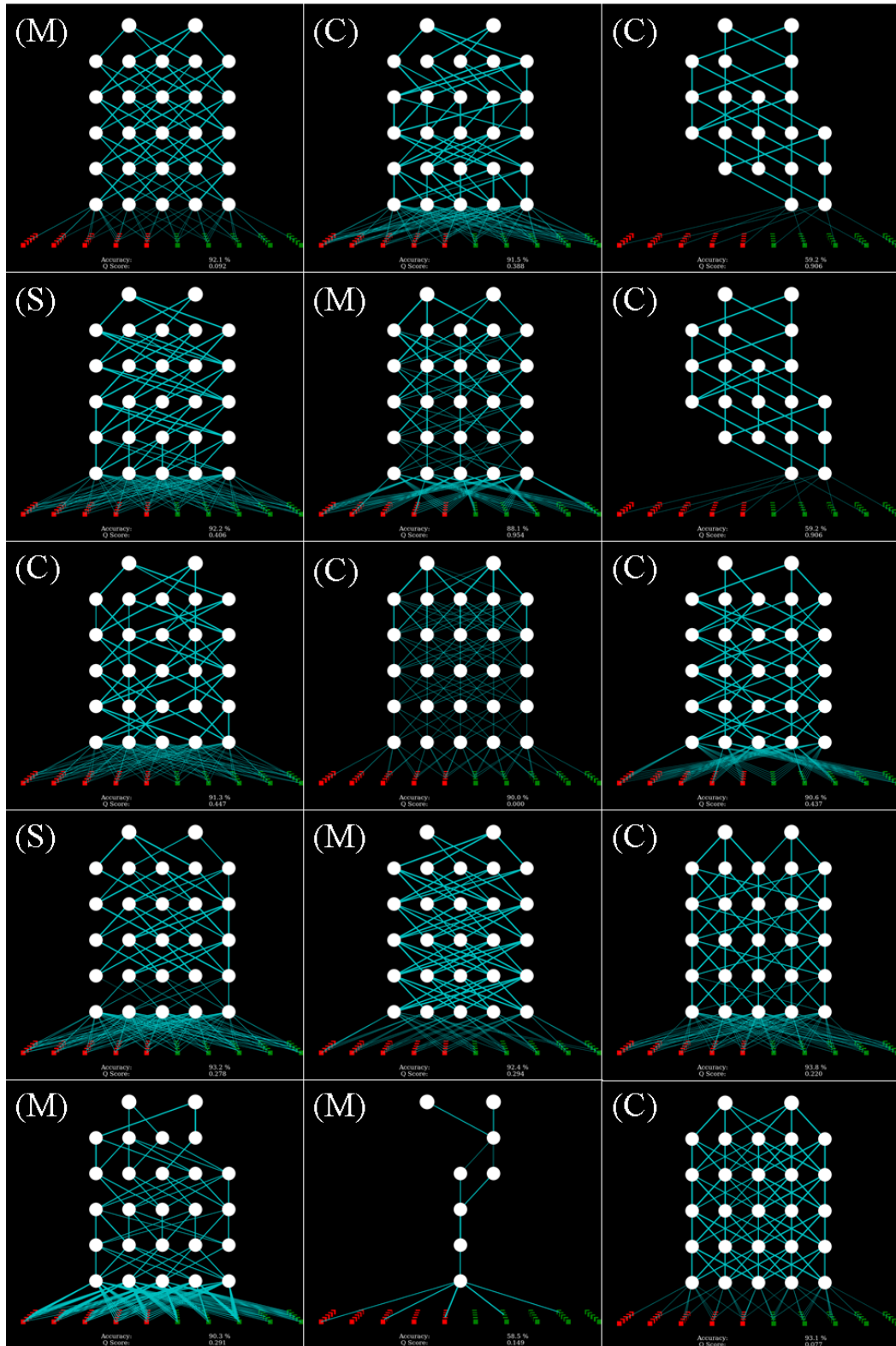


Figure 69. Experiment 1, geometrically inseparable task, without supralayer connections, EIC's final networks (16-30). The 'dense' motif (marked *D*) appears in 93.3% of runs. The 'null' motif (marked *N*) appears in 6.7% of runs.

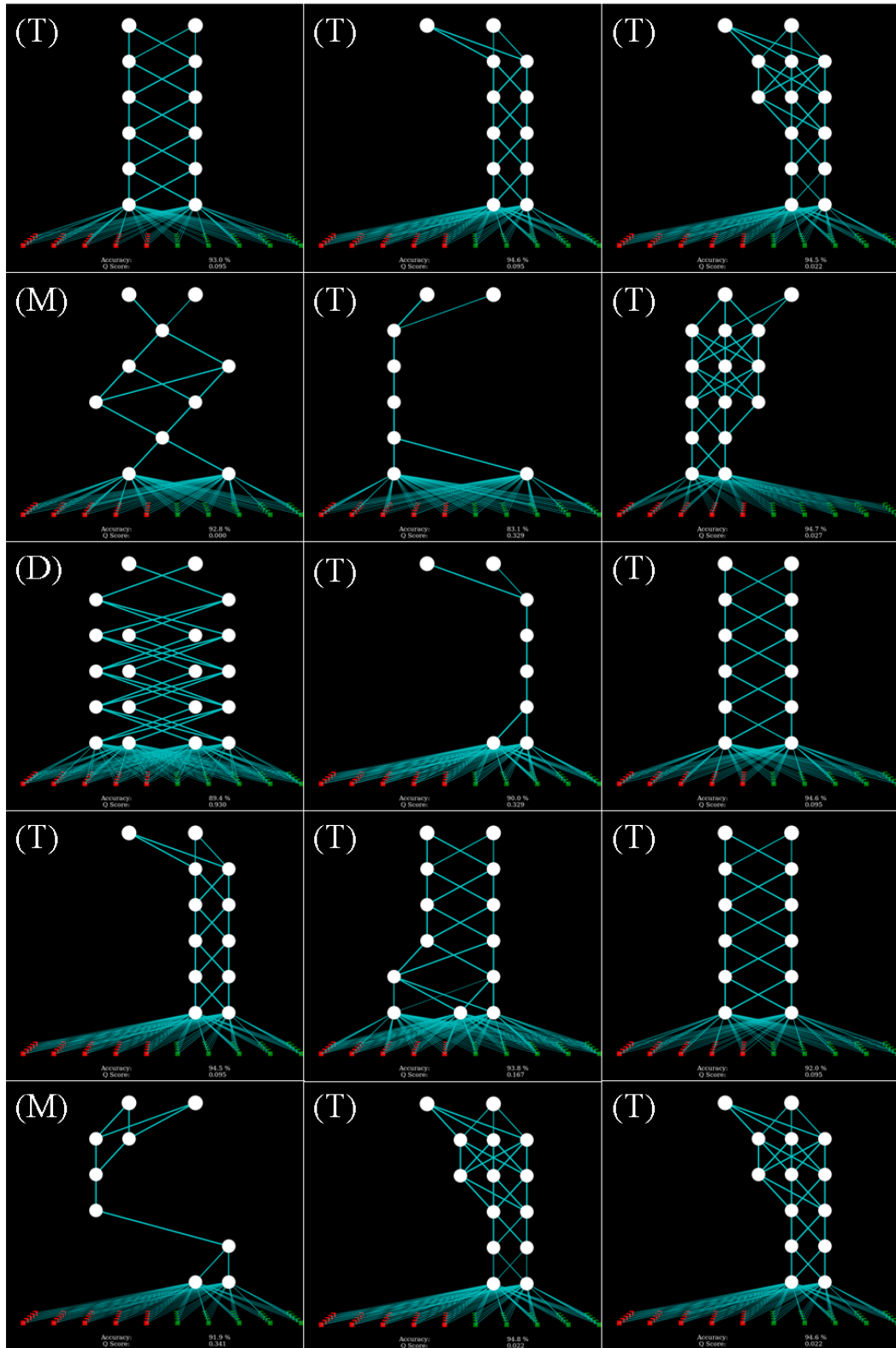


Figure 70. Experiment 1, geometrically inseparable task, without supralayer connections, GIC's final networks (1-15). The 'tower' motif (marked *T*) appears in 66.7% of runs. The 'messy' motif (marked *M*) appears in 23.3% of runs. The 'dense' motif (marked *D*) appears in 10% of runs.

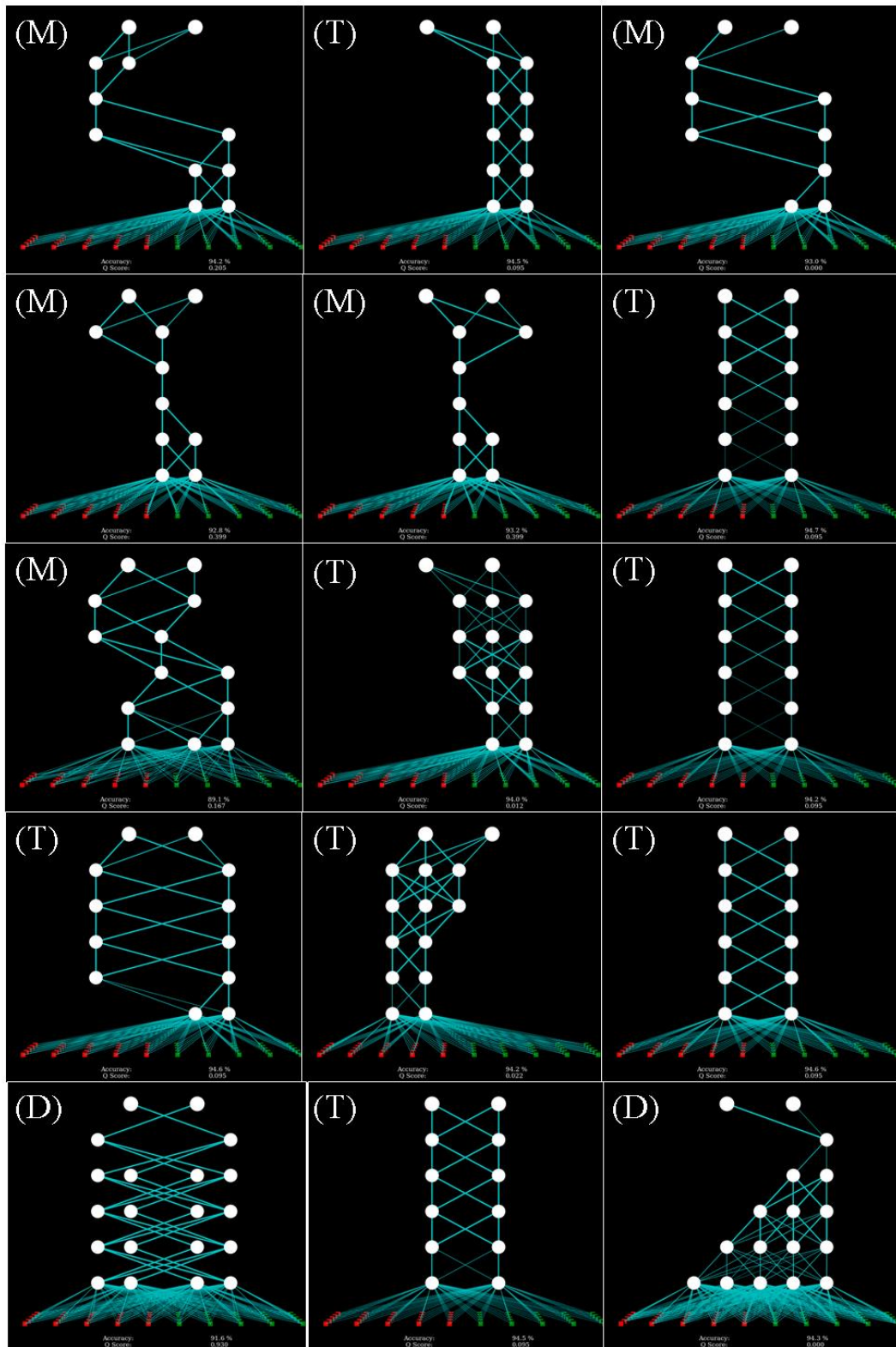


Figure 71. Experiment 1, geometrically inseparable task, without supralayer connections, GIC's final networks (16-30). The 'tower' motif (marked *T*) appears in 66.7% of runs. The 'messy' motif (marked *M*) appears in 23.3% of runs. The 'dense' motif (marked *D*) appears in 10% of runs

B.4 Experiment 2

Table 8. Percentile data and Mann-Whitney U-Tests & *P*-values.

	LCC		GCC		FIC		EIC		GIC	
	U	<i>P</i>	U	<i>P</i>	U	<i>P</i>	U	<i>P</i>	U	<i>P</i>
LCC	-	-	323	0.03	388	0.178	363	0.098	275	0.005
GCC	323	0.03	-	-	317	0.024	185	<0.001	429	0.375
FIC	388	0.178	317	0.024	-	-	266	0.003	246	0.001
EIC	363	0.098	185	<0.001	266	0.003	-	-	83	<0.001
GIC	275	0.005	429	0.375	246	0.001	83	<0.001	-	-
Q-score percentiles										
25%	0.0		0.0		0.115		0.370		0.0	
50%	0.599		0.0		0.320		0.608		0.051	
75%	0.837		0.489		0.541		0.964		0.229	
Task performance percentiles (%)										
25%	58.925		57.350		63.630		64.935		69.335	
50%	61.260		60.950		74.710		76.390		70.600	
75%	76.660		64.395		85.990		87.955		91.025	

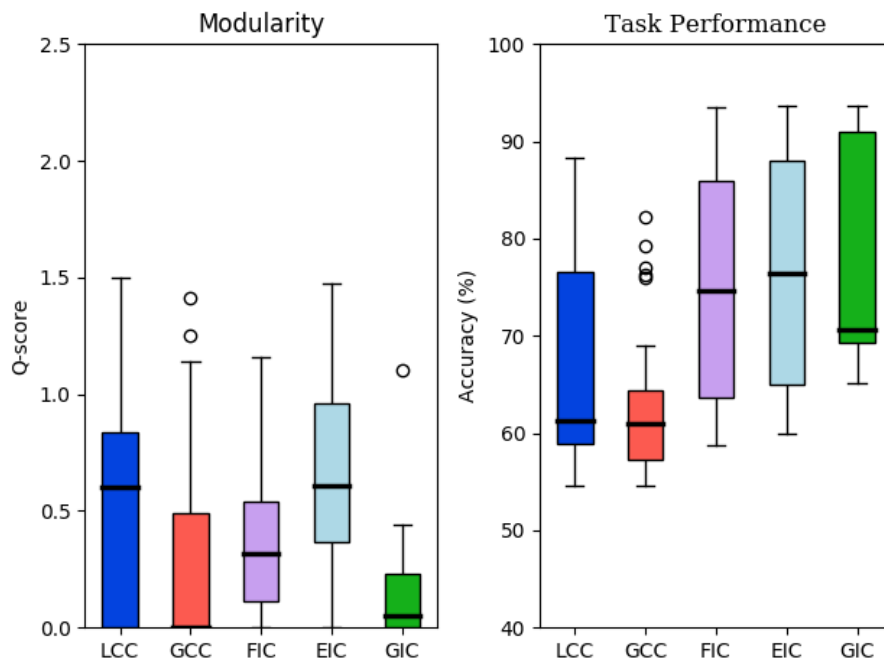


Figure 72. Comparison of Q-scores and task performance per connectivity constraint for Experiment 2 (with the geometrically separable task).

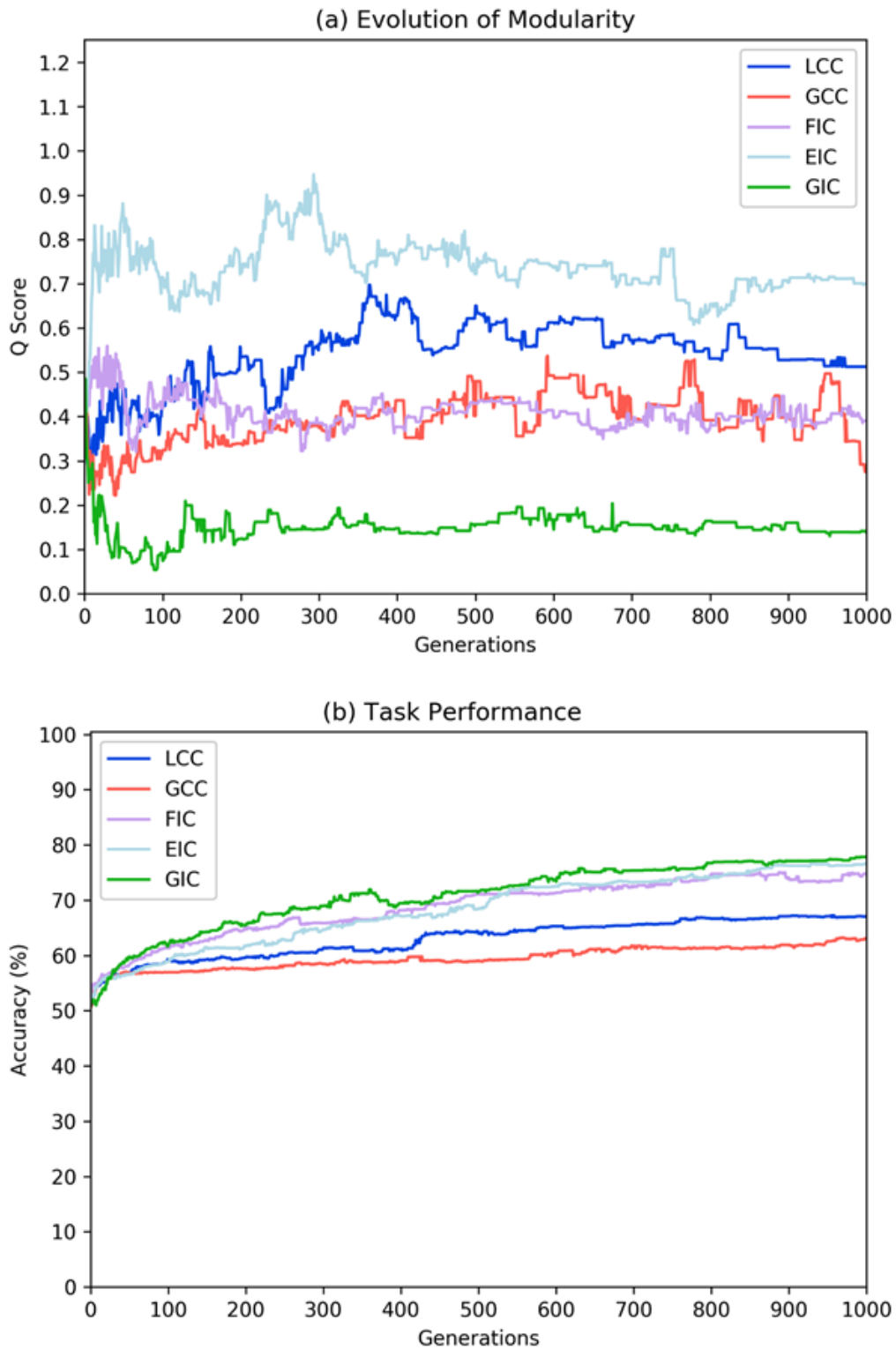


Figure 73. Convergence curves per connectivity constraint showing the evolution of (a) modularity and (b) task performance in Experiment 2 (with the geometrically inseparable task).

B.5 Experiment 3

Table 9. Experiment 3: no interaction.

	LCC		GCC		FIC		EIC		GIC	
	U	P	U	P	U	P	U	P	U	P
LCC	-	-	410	0.275	426	0.361	419	0.321	157	<0.001
GCC	410	0.275	-	-	380	0.149	433	0.398	181	<0.001
FIC	426	0.361	380	0.149	-	-	386	0.172	105	<0.001
EIC	419	0.321	433	0.398	386	0.172	-	-	149	<0.001
GIC	157	<0.001	181	<0.001	105	<0.001	149	<0.001	-	-
Q-score percentiles										
25%	0.767		0.234		0.575		0.364		0.0	
50%	0.893		0.903		0.905		0.774		0.086	
75%	1.253		1.126		1.432		1.349		0.308	
Task performance percentiles (%)										
25%	58.150		58.450		62.00		59.100		66.450	
50%	59.700		59.800		64.700		61.200		72.300	
75%	60.900		60.550		68.300		63.150		75.400	

Table 10. Experiment 3: weak explicit interaction.

	LCC		GCC		FIC		EIC		GIC	
	U	P	U	P	U	P	U	P	U	P
LCC	-	-	374	0.176	178	<0.001	216	<0.001	229	0.001
GCC	374	0.176	-	-	183	<0.001	208	<0.001	225	0.001
FIC	178	<0.001	183	<0.001	-	-	322	0.043	348	0.066
EIC	216	<0.001	208	<0.001	322	0.043	-	-	279	0.009
GIC	229	0.001	225	0.001	348	0.066	279	0.009	-	-
Q-score percentiles										
25%	0.033		0.014		0.472		0.340		0.445	
50%	0.136		0.035		0.699		0.651		1.034	
75%	0.581		0.549		0.914		0.748		1.097	
Task performance percentiles (%)										
25%	87.860		87.260		91.390		91.880		92.105	
50%	92.490		89.940		92.920		93.460		92.830	
75%	93.645		92.600		93.905		94.360		94.115	

Table 11. Experiment 3: strong explicit interaction.

	LCC		GCC		FIC		EIC		GIC	
	U	P	U	P	U	P	U	P	U	P
LCC	-	-	428	0.372	390	0.188	375	0.134	421	0.334
GCC	428	0.372	-	-	415	0.302	412	0.287	445	0.471
FIC	390	0.188	415	0.302	-	-	431	0.389	447	0.482
EIC	375	0.134	412	0.287	431	0.389	-	-	411	0.282
GIC	421	0.334	445	0.471	447	0.482	411	0.282	-	-
Q-score percentiles										
25%	0.312		0.315		0.291		0.215		0.245	
50%	0.631		0.587		0.404		0.408		0.602	
75%	0.866		0.857		0.739		0.811		0.998	
Task performance percentiles (%)										
25%	60.205		58.540		76.900		82.00		88.370	
50%	85.570		60.450		83.480		86.670		90.270	
75%	88.434		82.435		86.070		89.965		91.845	

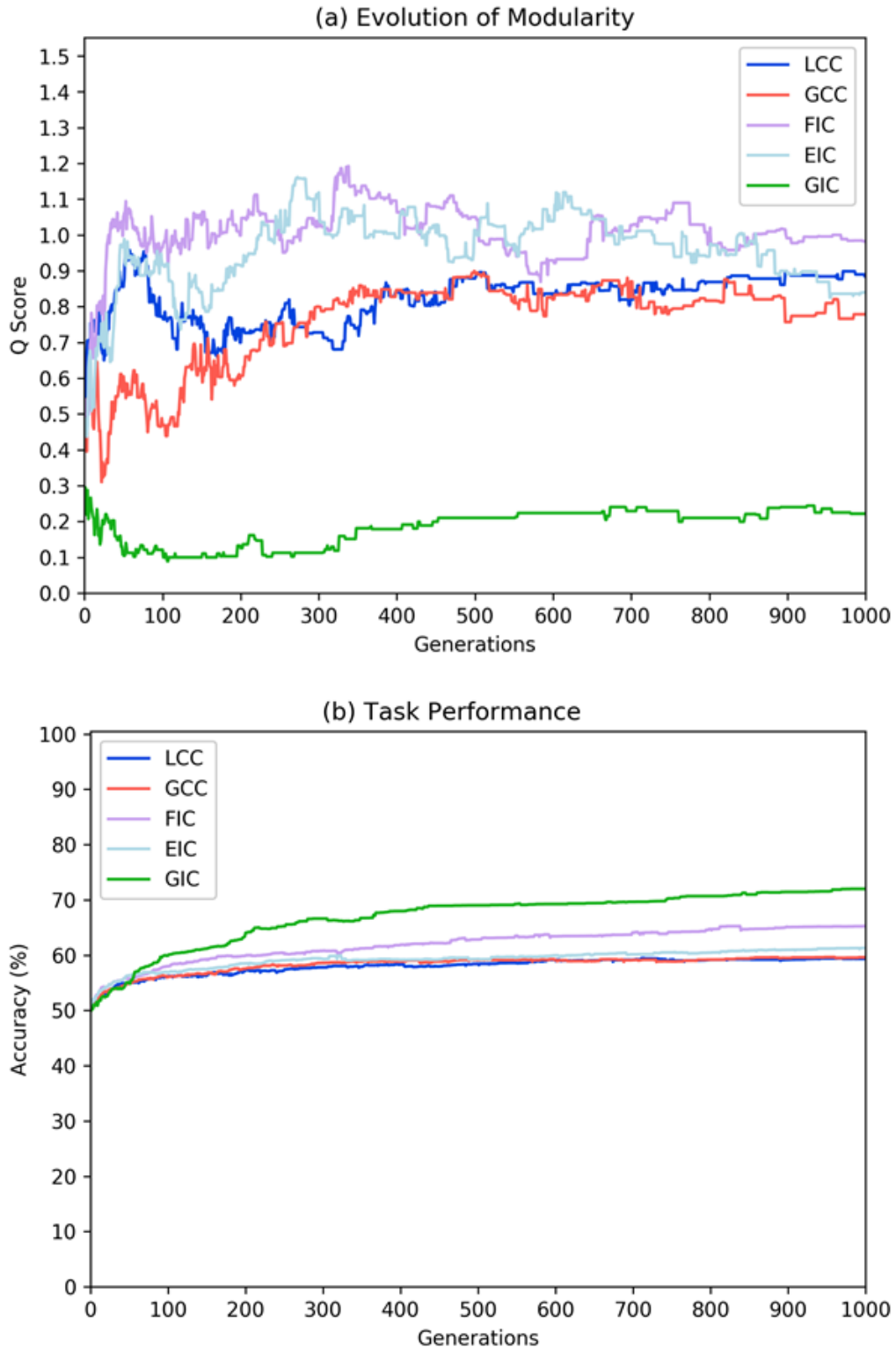


Figure 74. Convergence curves per connectivity constraint showing the evolution of (a) modularity and (b) task performance for experiment with no interaction between lifetime and evolutionary learning.

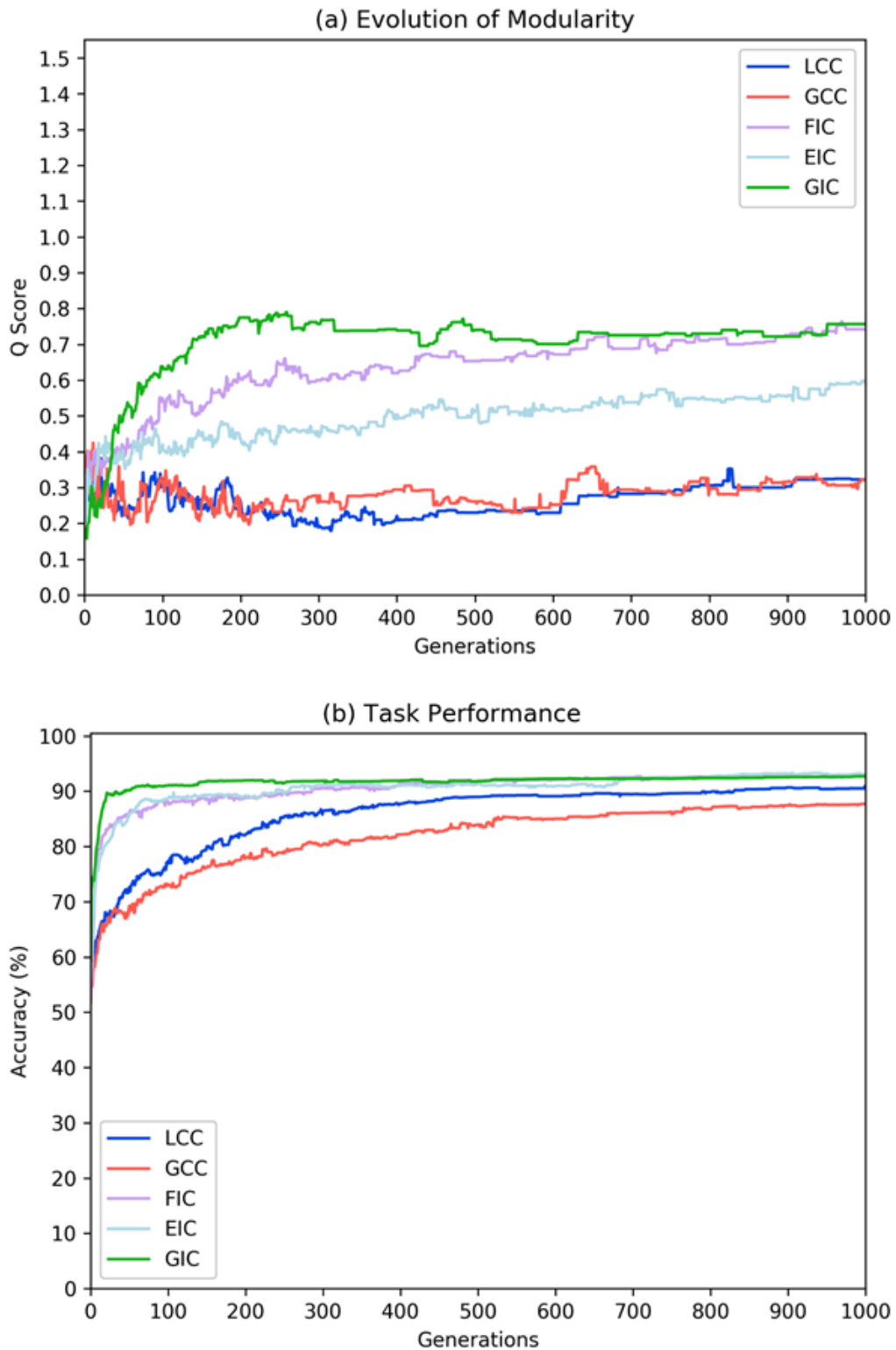


Figure 75 Convergence curves per connectivity constraint showing the evolution of (a) modularity and (b) task performance for experiment with weak interaction between lifetime and evolutionary learning.

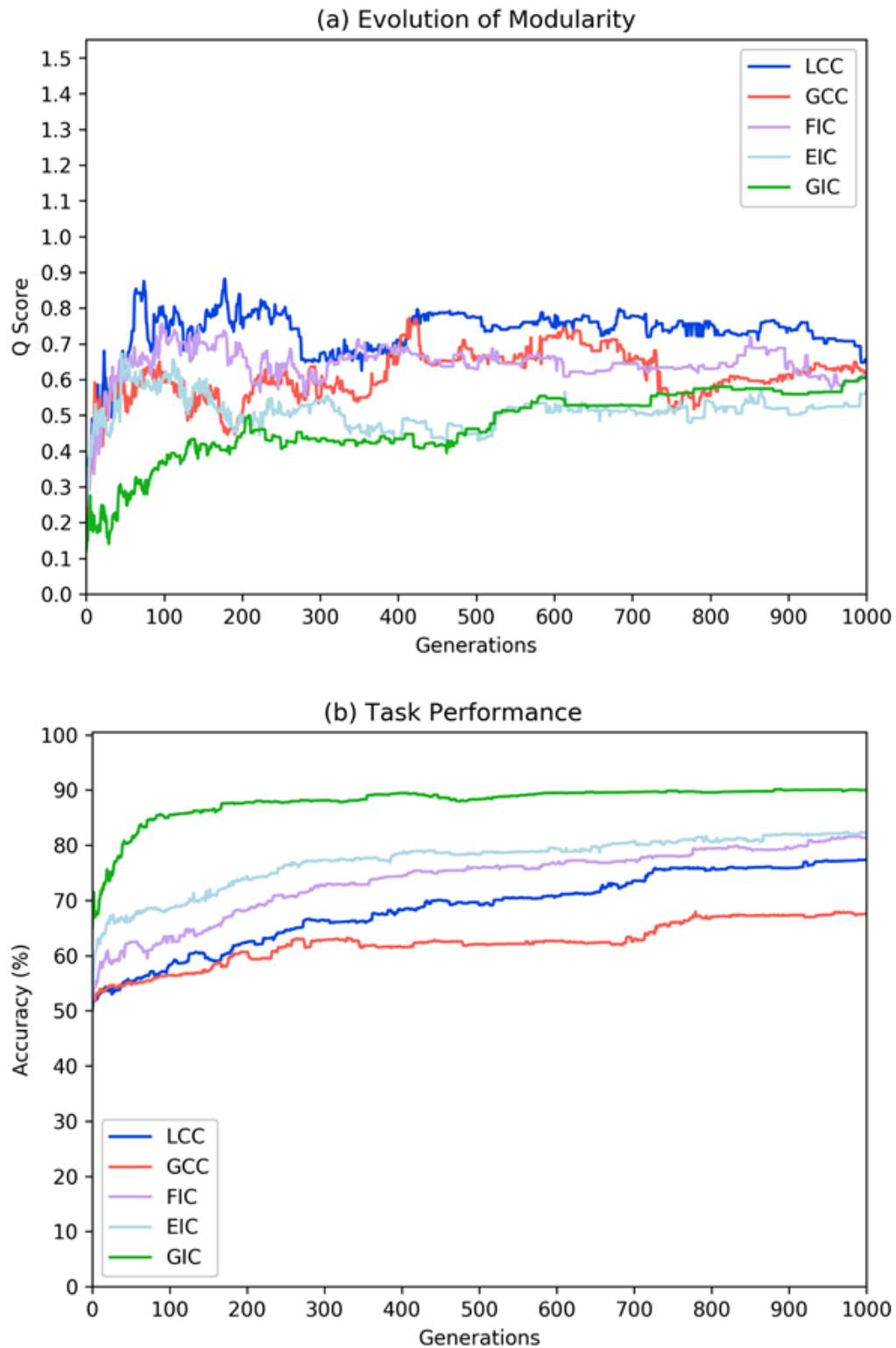


Figure 76. Convergence curves per connectivity constraint showing the evolution of (a) modularity and (b) task performance for experiment with strong interaction between lifetime and evolutionary learning.

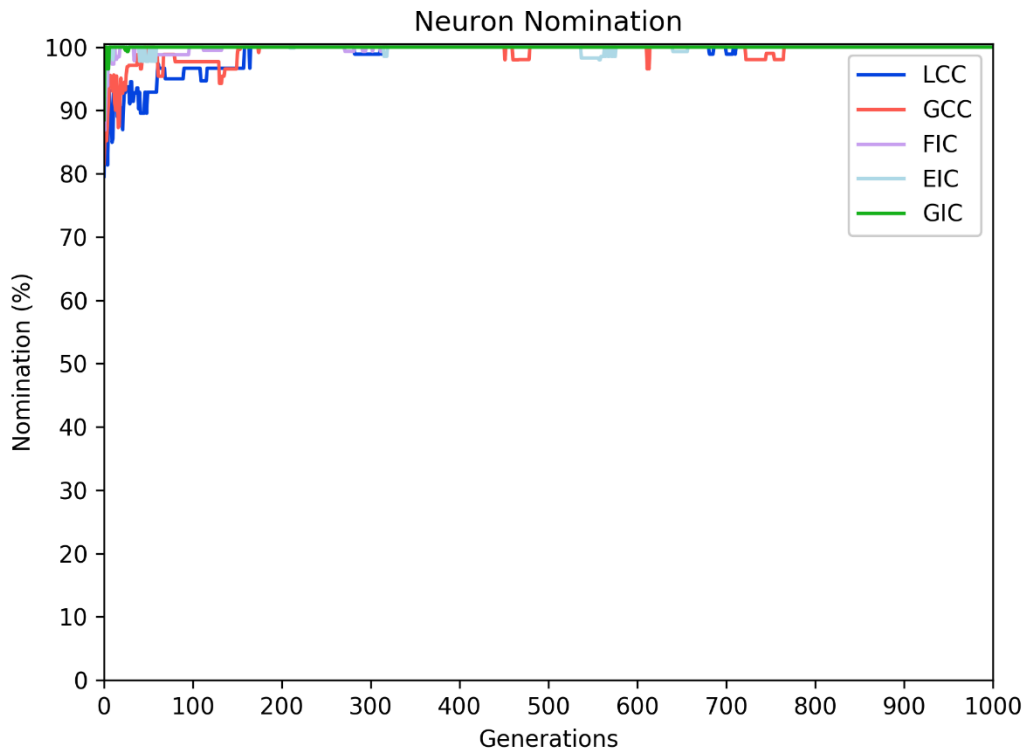


Figure 77. Neuron nomination values for the weak explicit interaction.

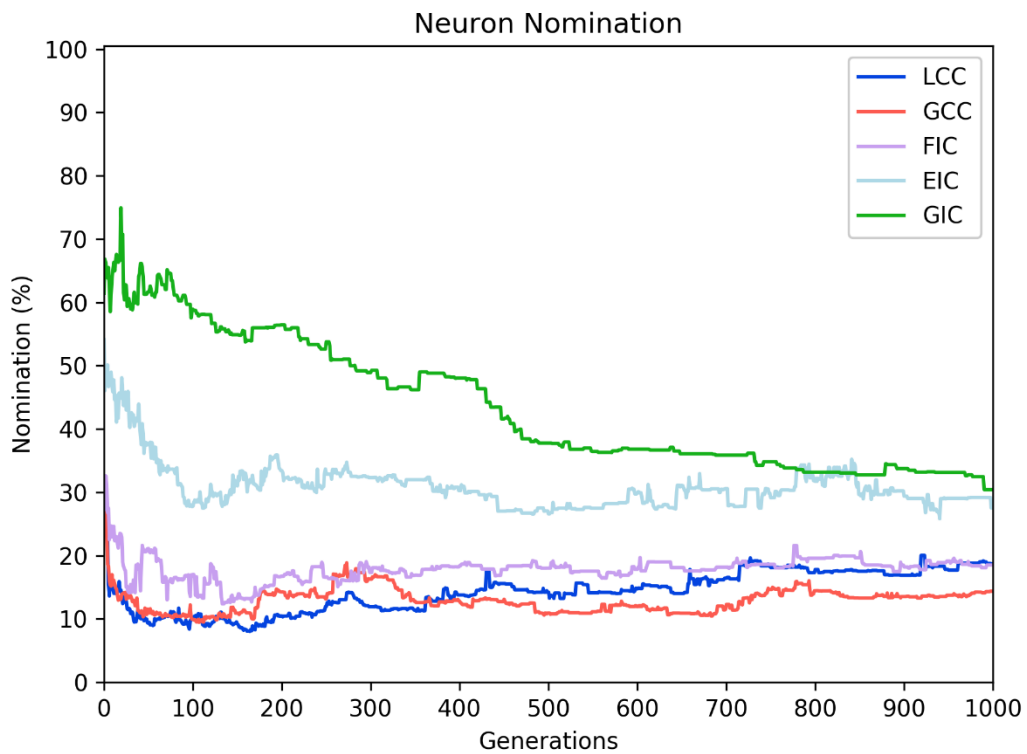


Figure 78. Neuron nomination values for the strong explicit interaction.

B.6 Experiment 4

Table 12. Experiment 4 on the geometrically separable task.

	LCC		GCC		FIC		EIC		GIC	
	U	P	U	P	U	P	U	P	U	P
LCC	-	-	389	0.182	291	0.009	245	0.001	291	0.009
GCC	389	0.182	-	-	257	0.002	234	0.001	280	0.006
FIC	291	0.009	257	0.002	-	-	450	0.497	390	0.188
EIC	245	0.001	234	0.001	450	0.497	-	-	400	0.230
GIC	291	0.009	280	0.006	390	0.188	400	0.230	-	-
Functional module overlap percentiles										
25%	0.343		0.333		0.136		0.222		0.25	
50%	0.463		0.5		0.303		0.327		0.348	
75%	0.549		0.667		0.476		0.44		0.417	
Task performance percentiles (%)										
25%	50.300		58.150		54.217		50.400		64.117	
50%	55.000		62.470		62.467		54.033		74.433	
75%	66.617		76.933		87.867		58.800		85.567	

Table 13. Experiment 4 on the geometrically inseparable task.

	LCC		GCC		FIC		EIC		GIC	
	U	P	U	P	U	P	U	P	U	P
LCC	-	-	338	0.07	336	0.046	247	0.002	420	0.407
GCC	338	0.07	-	-	223	0.001	151	<0.001	288	0.02
FIC	336	0.046	223	0.001	-	-	312	0.031	424	0.434
EIC	247	0.002	151	<0.001	312	0.031	-	-	273	0.011
GIC	420	0.407	288	0.02	424	0.434	273	0.011	-	-
Functional module overlap percentiles										
25%	0.0		0.0		0.115		0.154		0.0	
50%	0.118		0.0		0.143		0.269		0.250	
75%	0.333		0.167		0.242		0.407		0.250	
Task performance percentiles (%)										
25%	78.800		81.133		65.800		55.333		81.133	
50%	84.633		83.667		97.767		67.867		95.867	
75%	97.317		87.333		98.117		80.333		96.400	

