2023 IEEE Symposium Series on Computational Intelligence (SSCI)
Mexico City, Mexico. December 5-8, 2023

Connectivity Schemas in NeuroEvolution: What Neural Architectures does GEPNN evolve?

Mwaura, Jonathan
Khoury College of Computer Sciences
Roux Institute At Northeastern University
Portland, ME, USA
j.mwaura@northeastern.edu

Heminway, Ryan
Khoury College of Computer Sciences
Northeastern University
Boston, MA, USA
heminway.r@northeastern.edu

***Abstract*—In recent years, there has been a rise in popularity of using evolutionary algorithms (EA's) in conjunction with artificial neural networks (ANN's). This approach is commonly known as NeuroEvolution (NE). NeuroEvolutionary approaches typically optimize just the weights of an ANN or optimize the architecture, learning rates, thresholds, and weights together. Algorithms capable of the latter are known as Topological and Weight Evolving ANN (TWEANN). One such TWEANN is Gene Expression Programming for Neural Networks (GEPNN). This paper presents an empirical investigation of the network topologies that arise when GEPNN is used and whether evolved architectures have any relation to state of the art architectures. Results show that GEPNN naturally discovers powerful structural motifs such as shortcut connections and also creates sparse networks. Both these schemas have been shown to be advantageous in deep learning techniques. As an additional contribution from this work, we provide an open source library for developing GEPNN solutions in Python.**

***Index Terms*—Evolutionary computation, intelligent systems, neural networks, NeuroEvolution, architectures**

I. Introduction

Artificial neural networks (ANN's) are mathematical models of neuron connectivity and learning in systems biology. Although ANN's depict a simplified version of what happens in systems biology, they have proven to be powerful models for solving complex problems [1], [2]. Performance of these models relies on optimized weights and a network topology suited for the problem. It is common for the topology to be human designed based on mathematical formulations and for the weights to be optimized through a gradient-based method such as gradient descent.

Evolutionary Algorithms (EAs) are techniques that mimic Darwinian evolution to create models for heuristic search. These techniques have been shown to provide an alternative method to neural network weights optimization. In addition, EA techniques that evolve ramified tree-like structures (e.g Gene Expression Programming) have shown great capabilities at evolving both the architecture and the weights of the neural network.

The work presented here investigates network topologies that occur when EAs are used to evolve the structure and weights of an ANN. The motivation is to compare the patterns of connectivity created by evolutionary methods with those seen in existing architectures that have provided state-of-the-art results. Specifically, this work experiments with Gene Expression Programming for Neural Networks (GEPNN)[3] to observe the patterns of connectivity it creates. We view our contribution as a stepping stone to understanding the broader applications of EAs for topological innovation.

The rest of this paper is arranged as follows: First, a literature review discussing ANNs, deep learning and GEPNN is given. Second, the experimental setup is shown specifying all aspects of the experimentation. Thirdly, the results showing both performances and evolved structures is shown. Fourthly, a discussion highlighting insights to the results is presented and finally, a conclusion and motivation for further work is drawn.

II. Literature Review

A. Artificial Neural Networks

Artificial neural networks, at their core, are connectionist models for computation. Each fundamental unit in an ANN is a node or neuron that calculates the weighted sum of its inputs and applies an activation function to produce output. The mathematical formulation for this is:

$$a = f(b + \sum_{i=1}^{n} w_i x_i) \tag{1}$$

where a represents the output of the node, f represents the activation function used, b represents the bias term for the given node, w_i corresponds to the weight of the connection from the i^{th} input, and x_i represents the value of the i^{th} input. The combination of these basic units of computation, with non-linear activation functions, into a layered structured yielded the neural network architecture known as the Multilayer Perceptron (MLP) [4]. The MLP architecture, visualized in Fig. 1, is identifiable as having at least three layers corresponding to an input, hidden, and output layer of densely connected nodes.

Furthermore, the MLP architecture provides the basis for feedforward neural network (FFNN) architectures. FFNN's map a fixed-size input to a fixed-size output using layers of nodes which have incoming connections from all nodes of the previous layer [5]. With each successive layer of non-linear nodes, a FFNN is able to extract hierarchical relationships in

the inputs to model complex functions. This capability creates a general processing schema that is well suited for classical problems such as classification or regression. As powerful as a feedforward network can be, its simplistic formulation also leads to drawbacks. Notably, the architecture assumes independence between training samples and constrains all inputs and outputs to a fixed size. These drawbacks have major impacts on tasks such as machine translation where input sentences are often different lengths than the desired output and modelling relationships between words in a sample is critical to effective learning.

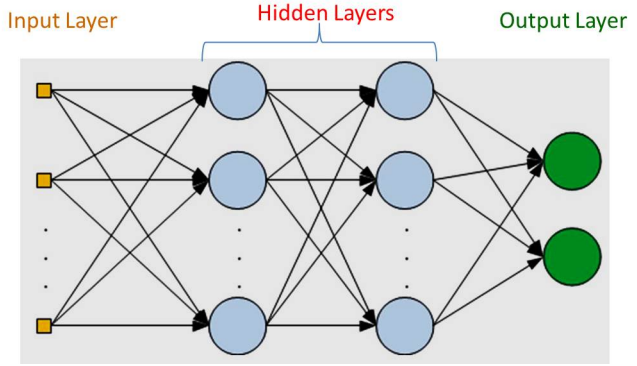

Fig. 1. Basic feedforward architecture of a Multilayer Perceptron[1].

B. Deep Learning

The capabilities of the first neural networks spurred years of machine learning (ML) research around these models including the creation of different architectures, training approaches, and optimization strategies. Deep learning (DL) refers to the field of ML interested in computational models created with multiple layers of processing units such that the system is able to extract features at multiple levels of abstraction. There are many DL architectures in literature. However, this work reviews topical developments among two popular variants: Convolutional Neural Networks (CNN's) and Recurrent Neural Networks (RNN's). For a comprehensive review of deep learning and its history, please see [5].

1) Convolutional Neural Networks: CNNs, also known as ConvNets, are one of the most popular and successful ANN architectures that depart from the archetype set forth by the MLP. CNNs are specifically designed for processing data that is image-like, structured into one or two dimensional arrays [5]. A simple example of CNN architecture can be seen in Fig. 2. The critical innovation of the first CNN was the use of a single neuron with a "local receptive field" such that the activation value of the neuron can be viewed as the application of a convolutional kernel to a part of the input image [6]. This pattern of connectivity is referenced as the "convolutional layer" in works surrounding these topics [5]. This architectural shift away from neurons with dense input connections unlocks key processing capabilities. First, the application of convolution to create activation maps at each layer provides the model with a natural way to learn of how local features in an image are related. Second, since dense connections are not used in convolutional layers, and the weights for all units in such a layer are shared, the number of free parameters that must be optimized during training is dramatically reduced [6], this leads to ease in training.

Since CNN's first use in 1989, there have been a number of notable variations in architectural composition that demonstrate further improvements in training efficiency and computational capabilities. For more detail on these innovations such as pooling or dropout layers, please refer to [7].

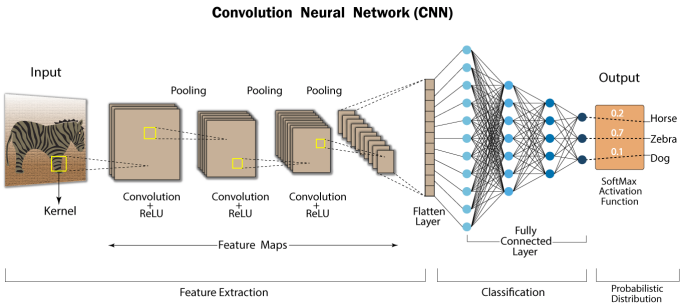

Fig. 2. Architecture of a basic Convolutional Neural Network, with labeled patterns of connectivity[3].

Another innovative connection schema related to CNNs is the "shortcut" or "skip" connection [8]. This schema provides a connection between nodes that skips one or more layers as visualized in Fig. 3. Although simplistic, this non-standard connection has been used extensively with results that demonstrate its flexibility and utility [9]–[12]. For instance in [9], shortcut connections are used across stacked convolutional layers such that the layers are forced to learn residual functions rather a true underlying mapping. In addition, [13] argues that shortcut connections enable construction of deeper networks in the face of vanishing gradient when training deep networks with gradient based optimization methods.

Further benefits of shortcut connections are shown in [14]–[16]. For example, [15] shows that skip connections avoid "singularities" which cause non-identifiability in the model which hinders optimization during training and also leads to favourable initialization of weight parameters [15]. These findings highlight the value of shortcut connections in designing deep neural network architecture as well as the fact that many of these topological motifs may not be fully understood.

2) Recurrent Neural Networks: A different, yet simple, shift in connectivity gave rise to another popular class of DL models: Recurrent Neural Networks (RNN). These models break the concept of FFNN connections to enable enhanced processing of sequential data. As seen in Fig. 4, RNNs are identified by "recurrent" connections which allow the transfer

[1]Image adapted from https://kindsonthegenius.com/blog/basics-of-multilayer-perceptron-a-simple-explanation-of-multilayer-perceptron/

[3]Image adapted from https://www.analyticsvidhya.com/blog/2021/05/20-questions-to-test-your-skills-on-cnn-convolutional-neural-networks/

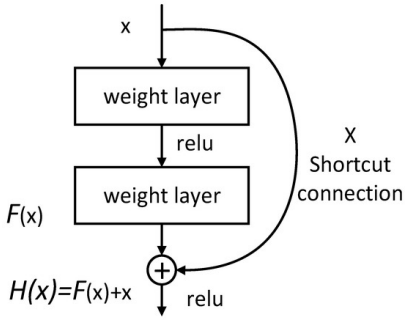

Fig. 3. Example of a shortcut connection commonly used in deep neural network architectures. Image adapted from [9].

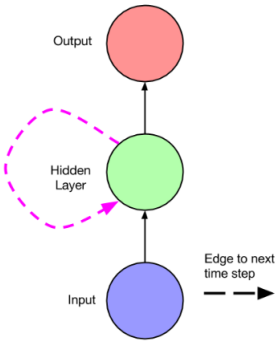

Fig. 4. Architecture of a simple RNN with a single hidden node. Image adapted from [17].

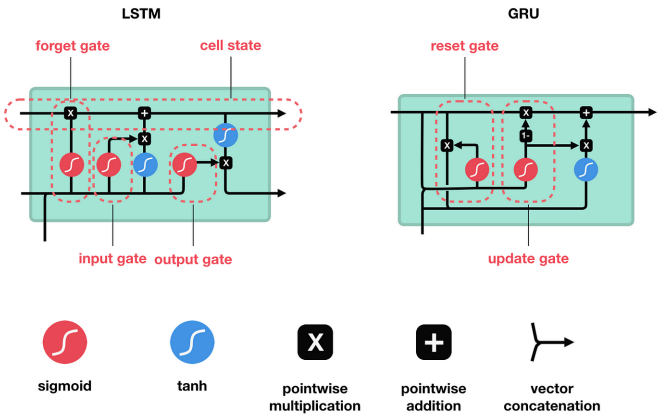

Fig. 5. Architectures of LSTM and GRU cells, with labeled structural components[5].

of information across sequence steps via a ”hidden state” [17]. With its recurrent formulation, RNN models explicitly solve some of the shortcomings previously mentioned about FFNN. Principally, RNNs do not assume independence between sequence elements and can be used for dynamically sized inputs and outputs.

Continuing with simple shifts in connectivity, bidirectional RNNs (BRNN’s) add an additional layer of hidden nodes that support recurrent connections to both past and future elements in the sequence [18]. While useful, the basic RNN models still suffer from the same problems such as vanishing gradients [13]. To overcome this, researchers created new ”memory cells” to replace traditional nodes in the hidden layer of a network [19]. The most popular architectures for these memory cells are called the Long Short-Term Memory unit (LSTM) and the Gated Recurrent Unit (GRU) [17] (See in Fig. 5). One way to understand these memory cells is as a more complex activation function for a typical neuron, with multiple functional components within it. The added functional components are termed ”gates” which control the flow of data throughout the cell to conditionally ”remember” or ”forget” information within the hidden state [20]. Important to our discussion on structures, [17] emphasize that the memory cell is really a ”composite unit, built from simpler nodes in a specific connectivity pattern”. Apart from its structure, memory cells are imperative to modern RNN architectures because their recurrent gated connections are able to propagate errors across time steps in such a way that avoids exploding or vanishing gradients [17].

In another feat of topological innovation, [21] combined shortcut connections with RNNs for an architecture they deemed a Dilated Recurrent Neural Network (dRNN). The addition of a shortcut connection to an RNN was able to produce state-of-the-art performance while improving training efficiency ”even with standard RNN cells”.

This short review of architectural components present in CNNs and RNNs showcases that although these are two different models of connectivity, they both share a common theme of using skip connections. In addition, skip connections seem to lead to ease of training as well as improved performance. The rest of this work investigates whether skip connections and other noteworthy connectivity schemas will arise with use of GEPNN.

[5]Image adapted from https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

C. Gene Expression Programming for Neural Networks (GEPNN)

Gene Expression Programming (GEP) [22] is an evolutionary algorithm (EA) similar to Genetic Algorithms and Genetic Programming. GEP combines the simplicity of Genetic Algorithm (linear chromosome representation) with the capabilities of Genetic Programming (ramified tree structures), to create a genotype/phenotype mapping that allows it to outperform its predecessors on complex problems.

A GEP individual, the genome, is made of a fixed length string of symbols that corresponds to one or more genes [22]. Symbols in the string correspond to arithmetic functions such as addition or multiplication as well as to input variables. The value of this representation is its expression into a phenotype which represents a mathematical expression tree. The expression tree is what is ultimately used for fitness evaluations of the individual. While each gene does have a fixed length, the open reading frame (ORF) construction of the phenotype means that

a genome is capable of representing expression trees of varying sizes. This provides a wider search space and the propensity to discover optimally sized expression trees for a given task.

An important extension to GEP is the capability to be utilised in NeuroEvolution, that is, the application of an evolutionary algorithm for optimization of one or more aspects of ANNs. In the case of GEP, the ramified tree-like structures of an individual phenotype is interpreted as a neural architecture. This is referred to as Gene Expression Programming for Neural Networks (GEPNN) [3]. The GEPNN evolves not only the weights of a neural network but also its topology, activation thresholds and learning rates.

In order to enable NeuroEvolution in the basic GEP linear encoding, new domains for weights and thresholds are appended to the end of the genome, with the boundaries of each domain determined by the maximum arity of the function set used [3]. Typical genetic operators such as mutation and recombination are easily modified to support these new domains. Figure 6 shows an example of a GEPNN chromosome and neural architecture.

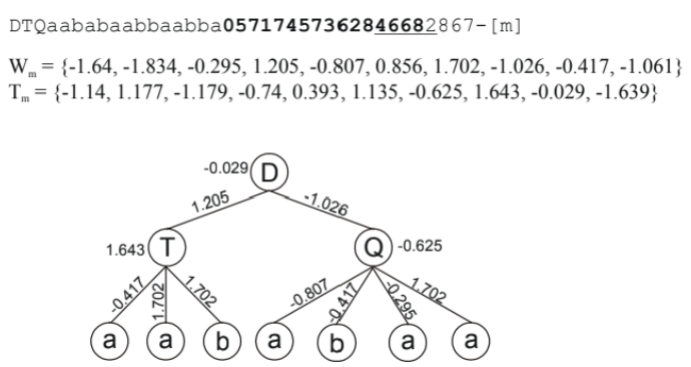

Fig. 6. Example of the linear genome of a GEPNN individual and its expression as a neural network phenotype, including weights and thresholds for connections and nodes respectively. Adapted from [3].

NeuroEvolution (NE) is not exclusive to GEPNN. In various works, genetic algorithms and evolution strategies have been used in optimization of neural network weights as an alternative to backpropagation [23], [24]. For instance in [25], a GA proved to be competitive and even advantageous in optimizing the weights of deep neural networks with fixed topologies .

EAs such as GEPNN, that are built for the capability to evolve the architecture, weights, learning thresholds and activate weights of ANN are known as Topological and Weight Evolving Artificial Neural Networks (TWEANNs) [23]. TWEANN's have been shown to perform comparably to state of the ANN/DL models. For instance, [26], [27] demonstrate the use of NeuroEvolution of Augmenting Topologies (NEAT) to evolve the topology of memory cells for sequence learning problems with RNNs. The results were memory cells that were comparable in structural components and performance to that of state-of-the-art LSTM cells. In another work, Evolutionary Acquisition of Neural Topologies (EANT) was shown to yield successive performance on an XOR and Double Pole balancing problem [28]

NEAT and EANT provide an alternative to GEPNN and utilise different encoding and optimization strategies to achieve NE. Notable to our discussion on structures, NEAT uses an individual encoding which includes unique identifiers for nodes in the resultant network that allow for connections between any two nodes [29]. EANT takes this a step further by additionally encoding the type of connection between nodes [28]. It is valuable to keep these differences in mind throughout the remaining analysis. The work reported here investigates GEPNN due to its simplicity and the capabilities to mutate the head section of the genome which we posit could lead to capabilities in evolving skip connections.

III. Experimental Setup

In this work, the aim was to apply GEPNN to classical machine learning problems as a medium for analyzing the patterns of connectivity that arise in the evolved neural networks. Similar to [30] where the capabilities for GEPNN in ML is investigated, the work reported here use two classification tasks and one regression problem.

The classification problems used were (a) Iris data set [31]: a small but well studied classification task that has 150 samples, 4 features per sample, and 3 possible classes and (b) the Glass data set [32]: a slightly more complex with 214 samples, 9 features per sample, and 6 possible classes. All experiments used an 80/20 train and test split where test data was used to evaluate the model's ability to generalize to unseen samples after it was finished training. The regression problem chosen uses the polynomial function of $y = 2.718a^2 + 3.146a$ which provide a relatively difficult function finding task. Note that these ML problems are investigated in [30] and their usage in the current work is to act as a benchmark to verify that the implementation yielded expected results.

A. Algorithm Parameters

Parameters that are shared between all experiments can be found in Table I. Parameters that are unique for the regression, iris classification, and glass classification experiments can be found in Table II. To clarify notation in the listed function sets, typical activation functions used for neural network construction are described using a formula of A_F where A describes the arity of the function, and F describes the type of activation function used. Following the convention set in [3], the arity descriptor D refers to a function with arity of two. The activation functions are one of [ReLu, sigmoid, tanh] and are denoted in the notation by the first letter of their name. A function set entry of D_s thus refers to a sigmoid function with an arity of two.

B. Experiment Variations

To support the focus on connectivity in GEPNN, two GEPNN variations of each task described above were run. The first variation uses the canonical description of GEPNN where there are no limitations on the type of networks that can be

TABLE I
COMMON PARAMETERS FOR ALL EXPERIMENTS

Number of runs	100	Gene length	41
Tournament size	3	Mutation rate	0.044
Population size	20	One point crossover rate	0.6
IS transposition rate	0.1	RIS transposition rate	0.1
Head length	7	Extra domains transposition rate	0.1
Number of Elites	1		

TABLE II
EXPERIMENT-SPECIFIC PARAMETERS

	Regression	Iris	Glass
Generations	5000	1000	
Function Set	*,+,/	-,*,+,D_r,D_t,D_s	
Terminal set	a	sl,pl,sw,pw	Si,K,Fe,Al,RI,Ca,Ba,Mg,Na
Gene Count	1	3	7
Max arity	2	4	

evolved other than those implicit in the encoding itself. The second variation, denoted henceforth as "guided", restricts the head section of every individual to a length of seven rather than eight and restricts it to contain only functions. In this variation, genetic operators are modified or disabled to maintain this invariant. Elements within the head can still be mutated to other functions within the function set. All other parameters used for this experiment match those listed for the canonical GEPNN experiment.

C. Fitness Functions

The fitness functions used match those used in [30] to provide a fair comparison. Specifically, the fitness function used for the regression experiment is:

$$f_i = \sum_{i=1}^{C_t} (M - |C_{(i,j)} - T_j|) \tag{2}$$

where M is the range of selection, $C(i,j)$ the value returned by the individual i for fitness case j (out of C_t fitness cases) and T_j is the target value for fitness case j. For this problem, $M = 100$ was used which provides for $f_{max} = 1000$.

For the classification problems, a summed Cross Entropy Loss between label predictions and true labels for the entire training set is used. The output of a GEPNN individual, when evaluating a specific input, is a vector of logits where each individual logit is the output of a gene in the individual. During the fitness evaluation of an individual, the logits are passed through a Softmax function to create a probability distribution over the set of possible classes. The Softmax plays the role of the linking function as originally described in [3]. True labels are formatted in a one-hot vector notation.

D. Programming Framework

The work described here extended the open-source Geppy [33] library to create a new library for the flexible application of GEPNN. In addition, the library is constructed with the flexibility to create PyTorch compatible neural networks. This new GEP library, GeppyNN, has been made publicly available to enable reproducibility of the reported results.

IV. RESULTS

A. Performance Analysis

The results of the regression and classification tasks are summarized in Tables III and IV respectively. All the results represent an average across the one hundred runs conducted for each task. Both variations of the GEPNN implementation performed quite well on these tasks as can be seen in comparisons with the results found in [30].

TABLE III
REGRESSION RESULTS

Implementation	Variation	Average Max Fitness
Wang et al. [30]	GEPNN	995.70
This paper	GEPNN	995.05
This paper	GEPNN (Guided)	996.26

TABLE IV
CLASSIFIER RESULTS

Implementation	Data Set	Variation	Test Accuracy
Wang et al. [30]	Iris	GEPNN	94.00%
This paper	Iris	GEPNN	96.16%
This paper	Iris	GEPNN (Guided)	92.04%
Wang et al. [30]	Glass	GEPNN	57.94%
This paper	Glass	GEPNN	62.16%
This paper	Glass	GEPNN (Guided)	63.31%

The aim of showing these performances is to ascertain and reproduce GEPNN performance on typical tasks applied to machine learning. These results mirror those attributed to[30]. Of importance is that the performance of the guided GEPNN is not hampered by the limitations imposed on it.

B. Structural Analysis

The main focus of this work is on analyzing the neural architectures created by GEPNN. Figures 7,8,9,10, and 12 show the final neural architecture discovered by the best individual among all runs for the regression (standard GEPNN), regression (guided GEPNN), Glass (standard GEPNN), Glass (guided GEPNN), and Iris (guided GEPNN) experiments respectively. Connection weights are omitted from all images for readability. Nodes are labeled with the activation function used, corresponding to the provided function set for a given experiment. Note also that each input can appear multiple times as unique nodes in the image, due to the drawing library used.

As a general observation across all experiments, GEPNN produces FFNN of various sizes. Notably, all neural architectures are sparsely connected and a common motif, most obvious in Figures 7 and 9, is the presence of shortcut connections. As expected, the algorithmic restrictions for the guided experiments cause the neural architecture of each gene in an individual to be characterized by a structure that resembles a full binary tree in the hidden layers. This is easily seen

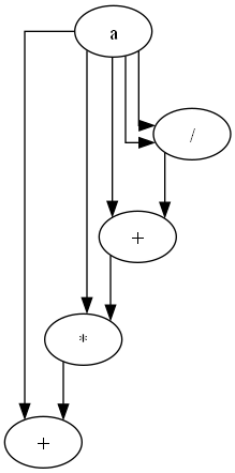

Fig. 7. Neural architecture of the best individual found while solving the regression experiment using standard GEPNN. This solution achieved a fitness of 999.57.

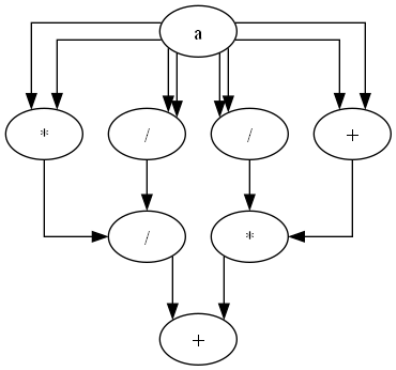

Fig. 8. Neural architecture of the best individual found while solving the guided regression experiment using guided GEPNN. This solution achieved a fitness of 999.87.

in Figure 10. Although not included here, a visualization of the individuals throughout the evolutionary path for the guided experiments would show that the structure of the hidden layers remains static throughout. The only perceived changes to the neural architecture, for the guided experiments, come in

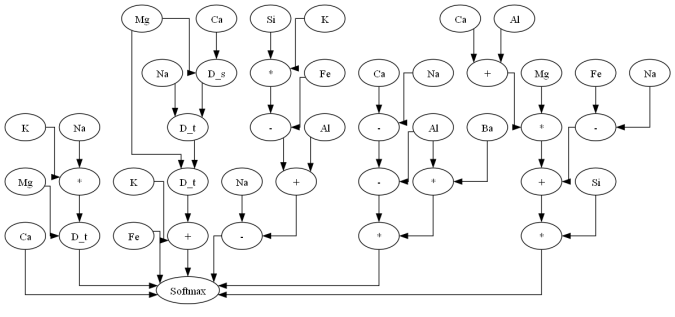

Fig. 9. Neural architecture of the best individual found while solving the Glass classification experiment using standard GEPNN. This solution achieved a fitness of 253.96 and a final test accuracy of 74.42%.

the form of changes to the activation functions, input layer connections, and connection weights. As evident in Tables III and IV, this restriction did not limit the effectiveness of the solutions.

Taking a closer look at how these final neural architectures evolve, Figure 12 depicts the neural architecture of the best individual in the population at various points along the evolutionary path for the Iris experiment using standard GEPNN. The figure in the top left of Figure 12 depicts a very simple neural architecture that provided the best fitness in the population beginning at generation 10 and lasting to generation 29. In the top center of Figure 12, the neural architecture is representative of a trend seen throughout the experiments: improvements in the fitness of an individual are often made by incrementally building out one sub-structure at a time. In the case of the individual from generation 500, this was realized by building out the central gene between generation 29 and generation 500. The individual from generation 550 captures the moment where the system begins expanding the right gene structure as well. By generation 1000, the best neural architecture found for the standard Iris classification experiment is reminiscent of the structures found in other experiments. Interestingly, we observe that the structure built up in the middle gene for the individual at generation 500 has now shifted to the left gene (likely through transposition) by generation 1000.

V. Discussion

The results across all figures demonstrate the capabilities of GEPNN to evolve sparse connectivity schema as well as shortcut (skip) connections. As earlier discussed these two aspects have led to an increase in performance in both CNNs and RNNs. In GEPNN, the sparse connectivity and the skip connections can be explained by the representation of GEPNN individuals. Principally, GEPNN uses a direct linear representation where the phenotype is formulated by an Open Reading Frame (ORF) expansion. This alone prohibits recurrent connections from ever being discovered, since connections are implicit in the representation so there is no way to specify a type for the connection. Just as important, the representation does not provide unique identifiers for nodes. This is quite limiting and is the reason why dense connections are simply impossible. Beneficially, though, the flexibility surrounding placement of functions and terminals allows the incorporation of diverse activation functions and natural discovery of shortcut connections.

As demonstrated by the guided experiments, GEPNN provides a flexible tool for designers to constrain the types of architectures discovered. This work presents a single way to do this, but the authors posit that many different models are possible. These algorithmic constraints can be viewed as tool for designers to incorporate a priori knowledge about a problem domain.

Finally, unlike most human designed architectures, or even those generated by other evolutionary algorithms, GEPNN algorithmically incorporates a set of diverse activation functions.

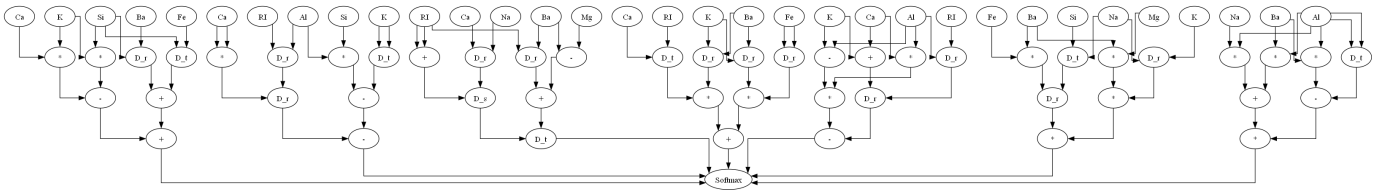

Fig. 10. Neural architecture of the best individual found while solving the Glass classification experiment using guided GEPNN. This solution achieved a fitness of 260.08 and a final test accuracy of 74.42%.

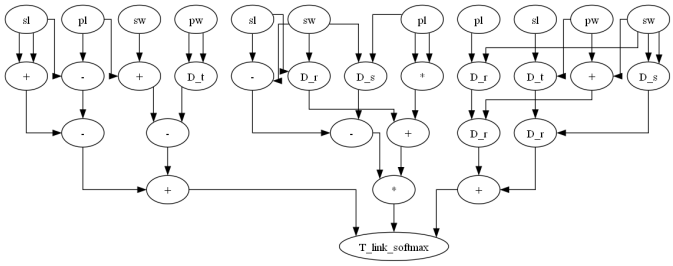

Fig. 11. Neural architecture of the best individual found while solving the guided Iris classification experiment using guided GEPNN. This solution achieved a fitness of 72.35 and a final test accuracy of 96.15%.

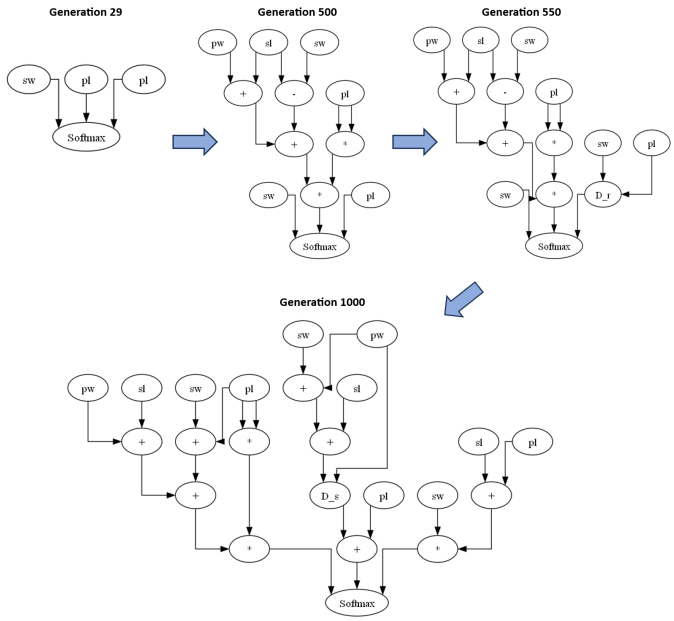

Fig. 12. Neural architecture of the best individual found at generations 29 (top left), 500 (top center), 550 (top right), and 1000 (bottom) when solving the Iris classification experiment using standard GEPNN. The fitnesses of these individuals were 106.31, 84.02, 82.64, and 73.12 respectively. The final test accuracy of the individual from generation 1000 was 100%.

The upside of this feature is worth exploring more in the future as it is unique compared to other approaches and, as far as we know, not well studied.

VI. Conclusion

This work explored the neural connectivity schemas discovered by the GEPNN algorithm. Whereas not exhaustive, the work carried out a literature review that shows that skip connections contribute a lot of advantages in deep learning techniques such as RNN and CNN. The review further concentrated on GEPNN, which is a TWEANN technique. The experimentation created a variation of GEPNN, the so called guided version, where the head section of the GEP could only contain functions. This guided GEPNN created structures that somewhat represented full binary trees; the intention being to investigate the possibility of creating dense networks using GEPNN.

The experimental results demonstrate the type of structures that GEPNN is capable of producing. Results show GEPNN naturally produces skip connections and sparse networks. These results showcase that GEPNN can be utilised in tasks that deep neural networks would be used. Furthermore, a lightweight library for developing GEPNN solutions in Python has been published to enable research repeatability.

A future work in this direction is to investigate capabilities of other TWEANN technique such as EANT and NEAT and create an empirical comparison with GEPNN. It is immediately possible to theorize about the different patterns of connectivity possible due to choices in the encoding representations of these other TWEANNs, and we hope to explore this experimentally in the future.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012.

[2] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017.

[3] C. Ferreira, "Designing neural networks using gene expression programming," in *Applied Soft Computing Technologies: The Challenge of Complexity*, A. Abraham, B. de Baets, M. Koppen, and B. Nickolay, Eds., Springer-Verlag, 2006, pp. 517–536.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, D. E. Rumelhart and J. L. Mcclelland, Eds., MIT Press, 1986, pp. 318–362.

[5] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[6] Y. LeCun, B. Boser, J. Denker, *et al.*, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2, Morgan-Kaufmann, 1989.

[7] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, 1998, pp. 2278–2324.

[8] C. Bishop, *Neural networks for pattern recognition*. Oxford University Press, USA, 1995.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '16, IEEE, 2016, pp. 770–778.

[10] B. Ripley, *Pattern Recognition And Neural Networks*. Jan. 2008, vol. 11.

[11] C. Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-Supervised Nets," *ArXiv e-prints*, vol. stat.ML, 2014.

[12] C. Szegedy, W. Liu, Y. Jia, *et al.*, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[13] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[14] O. K. Oyedotun, K. A. Ismaeil, and D. Aouada, "Training very deep neural networks: Rethinking the role of skip connections," *Neurocomputing*, vol. 441, pp. 105–117, 2021.

[15] E. Orhan and X. Pitkow, "Skip connections eliminate singularities," in *International Conference on Learning Representations*, 2018.

[16] E. T. B. Lundby, H. Robinsson, A. Rasheed, I. J. Halvorsen, and J. T. Gravdahl, *Sparse neural networks with skip-connections for identification of aluminum electrolysis cell*, 2023.

[17] Z. C. Lipton, "A critical review of recurrent neural networks for sequence learning," *CoRR*, vol. abs/1506.00019, 2015.

[18] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[20] K. Cho, B. van Merrienboer, C. Gulcehre, *et al.*, *Learning phrase representations using rnn encoder-decoder for statistical machine translation*, 2014.

[21] S. Chang, Y. Zhang, W. Han, *et al.*, "Dilated recurrent neural networks," Oct. 2017.

[22] C. Ferreira, "Gene expression programming: A new adaptive algorithm for solving problems," *Complex Systems*, vol. 13, no. 2, pp. 87–129, 2001.

[23] E. Papavasileiou, J. Cornelis, and B. Jansen, "A Systematic Literature Review of the Successors of "NeuroEvolution of Augmenting Topologies"," *Evolutionary Computation*, vol. 29, no. 1, pp. 1–73, Mar. 2021.

[24] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[25] 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/1712.06567, 2017.

[26] J. Bayer, D. Wierstra, J. Togelius, and J. Schmidhuber, "Evolving memory cell structures for sequence learning," in *Artificial Neural Networks – ICANN 2009*, C. Alippi, M. Polycarpou, C. Panayiotou, and G. Ellinas, Eds., Springer Berlin Heidelberg, 2009, pp. 755–764.

[27] A. Rawal and R. Miikkulainen, "From nodes to networks: Evolving recurrent neural networks.," *CoRR*, vol. abs/1803.04439, 2018.

[28] Y. Kassahun and G. Sommer, "Efficient reinforcement learning through evolutionary acquisition of neural topologies," in *The European Symposium on Artificial Neural Networks*, 2005.

[29] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[30] W. Wang, Q. Li, and X. Qi, "Gene expression programming neural network for regression and classification," in *Advances in Computation and Intelligence*, L. Kang, Z. Cai, X. Yan, and Y. Liu, Eds., Springer Berlin Heidelberg, 2008, pp. 212–219.

[31] R. A. Fisher, *Iris*, UCI Machine Learning Repository, 1988.

[32] B. German, *Glass Identification*, UCI Machine Learning Repository, 1987.

[33] S. Gao, *geppy: a Python framework for gene expression programming*, version 0.1, 2020. [Online]. Available: https://github.com/ShuhuaGao/geppy.