Should I Stay or Should I Grow? A Dynamic Self-Governed Growth for Determining Hidden Layer Size in a Multilayer Perceptron

Matt Ross
School of Psychology
University of Ottawa
Ottawa, Canada
mross094@uottawa.ca

Nareg Berberian
School of Psychology
University of Ottawa
Ottawa, Canada
nberb062@uottawa.ca

Sylvain Chartier
School of Psychology
University of Ottawa
Ottawa, Canada
sylvain.chartier@uottawa.ca

***Abstract*— A novel dynamic self-governed growth algorithm inspired from population dynamics is introduced in a Multi-Layer Perceptron (MLP). This allows the inclusion of a carrying capacity, which is the maximum population of hidden units that can be sustained in a single hidden layer. The inclusion of this constraint in combination with population dynamics provides a built-in mechanism for a dynamic growth rate. The proposed approach is used in parallel with direct performance feedback from the network to modulate the growth rate of the hidden layer. This algorithm incrementally adds units to the hidden layer up to a point where the complexity of the task no longer requires further addition. The MLP is extended with the growing algorithm and its adaptability is tested by subjecting the network to increasing levels of task complexity for the *n*-bit problem. Using fixed rules that dictate both the size of a fixed layer MLP (fMLP) and the upper bound carrying capacity of the growing MLP (gMLP), the resulting topologies are directly compared for the *n*-bit problem. In short, the results suggest that even if an upper boundary of the carrying capacity is set by a fixed rule, the growing algorithm is capable of converging to less than the predicted number of units required for solving the given task. With the majority of trials growing to the same number of hidden units regardless of the rule used. This effect is consistent across the specified rules and levels of task complexity for the *n*-bit problem.**

Keywords—*multilayer perceptron, growing, constructive algorithm, population dynamics, artificial neural networks, parity n-bit*

I. INTRODUCTION

In multilayered artificial neural networks (ANNs), deciding on the number of hidden units within the architecture can be crucial for properly learning and solving a given task. Using too few units can result in underfitting whereby the input data may not be adequately captured by the network preventing proper learning or generalization. In contrast, too many units can lead to overfitting whereby the input data is not enough to properly train all the hidden units and can therefore increase training time [1]–[4]. Unfortunately, there is no methodological consensus for determining the number of hidden units in a hidden layer. There are several rule-of-thumb methods [1], however typically this number is found by extensive time consuming trial and error [3], [5].

This problem is particularly important when multi-layered feedforward networks are used. The most popular of which is the multi-layered perceptron (MLP) that uses the backpropagation algorithm developed by Rumelhart [6]. The MLP includes internal hidden units which are not part of the input or output, yet come to represent important features that capture task-specific regularities. As a result of the hidden layer extension of the standard perceptron, the MLP has been shown to be capable in solving non-linearly separable tasks such as the XOR problem, as well as its higher dimensional extension known as the *n*-bit Parity problem [7]. The *n*-bit problem is one of the most widely used problems for testing the efficacy of training algorithms for ANNs [8], [9]. If the number of ones in the vector is even, then the output is 1, and if the number of ones is odd, then the output is 0 [8], [9]. The number of input vectors of an *n*-bit problem is given by the dimension parameter *m,* where *m* is given by 2^n, with $n \in \{0,1\}$ being the number of bits.

Previous approaches have formulated strict rules on the minimum number of hidden threshold units required to solve the *n*-bit Parity problem. For instance, in the case of a single hidden layer of the MLP, simulation experiments have shown that the minimum number of hidden units needed is equal to *n* in order to solve the *n*-bit problem [10], [11], whereas others have proposed that the general solution for solving the *n*-bit problem is $n+1$ [12]. These formulations require that the architecture of the MLP remains purely feedforward, with no direct connections between the input layer and the output layer. In contrast, if there are additional connections between the input and the output layers, as in the Bridged MLP, a generalized solution is $2n+1$ [12]. Although these rules allow the characterization of the number of hidden units required in the hidden layer of an MLP, these established rules require a prior specification of the number of hidden units and are only applicable if certain conditions are met.

A potential solution is to have the topology grow as the network learns through the use of constructive algorithms, that start with few units and incrementally add units and connections (for a detailed reviews see [4], [13]). This class of algorithms offer flexible economical topologies that can match task complexity [4]. The most popular amongst them include Cascade Correlation [14] which freezes the existing network and trains the newly added unit, and Dynamic Node Creation (DNC) [5] which retrains the entire network. Previous work, that falls under the category of DNC, has shown that the backpropagation algorithm can be used with varying number of hidden units to approach the XOR problem [15]. To accomplish incremental growth the total error was checked every 100 epochs and a new hidden unit was added if the total error remained higher than one percent. As a result, the size of the network increased outbound and therefore required intervention via removing a hidden unit and retraining until obtaining a more reasonable number of hidden units [15].

When should a constructive algorithm add a new unit? Adding units randomly or on a fixed time schedule could lead to unnecessarily large topologies. If a user defines a maximum

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

capacity size it could prevent excessive growth. Nonetheless, hidden units that are added may not have enough time to be trained to have an impact on the training error and could still lead to the addition of unnecessary units. The aforementioned constructive algorithms employ a user defined parameter whereby if the average training error drops below or stops changing by a preset amount, only then will a new unit be added [5], [14], [15]. While very effective, this requires some *a priori* parameter setting.

We propose an extension of the MLP using a flexible growing algorithm inspired by population dynamics [16]. From a theoretical viewpoint, a neural network or even each layer that comprises the network can be viewed as a set of populations. From this perspective, a hidden layer can be considered as the environment that neuronal units exist in. This would endow the hidden layer with a carrying capacity. Put simply, the carrying capacity is the maximum population size that the environment can sustain, or in terms of ANNs it is an upper bound on the possible number of hidden units. Giving the hidden layer this constraint allows us to apply population dynamics when calculating the growth of the hidden unit population. This provides a built-in self-governed method for preventing the network from growing too large. In this version, we are only concerned with the carrying capacity of the hidden layer and no other environmental factors, such as resource availability [16]. As such, we are considering this population as a "single-species" of neuronal unit within the hidden layer environment. For a given task, the algorithm allows the network to incrementally add units in the hidden layer up to a point where the complexity of the task no longer requires further addition in the number of hidden units.

The adaptability of the growing network (gMLP) is tested using increasing levels of task complexity for the *n*-bit problem, and compared to an identical MLP network that has a fixed hidden layer size (fMLP). In doing so, we show that fixed rules for the number of hidden units needed to solve *n*-bits are not ideal, and that dynamic flexible growth will lead to better generalized approximations in size irrespective of the rule used. As such, we do not focus on the total number of training epochs that the network takes, but the total number of hidden units used by the network for each rule across different *n*-bits.

The remainder of the paper is divided as follows: Section II introduces the model describing the network's architecture, activation function, learning algorithm, and learning procedure for a standard MLP with a single hidden-layer. We then described the growing algorithm inspired from population dynamics. Section III describes the results of the *n*-bit Parity problem using a standard fixed hidden-layer size (fMLP) and extended version of the dynamic self-governed growth algorithm (gMLP). In section IV, we discuss and conclude the overall findings of our work.

II. METHODS

A. Architecture

The MLP used is a simple feedforward network and is comprised of three layers: an input layer, a single hidden layer, and output layer containing a single unit (Fig.1). If the size of the hidden layer, *j*, is fixed then its size is dependent on the input vector dimension, *m,* and the selected rule. In this condition all the weight connections are present from the beginning. If the size of the hidden layer, *j*, is permitted to change across time as with growing, then new units are added to the end of the hidden layer. The weight connections associated to the new hidden unit are added from the input layer to the hidden and from the hidden to the output layer.

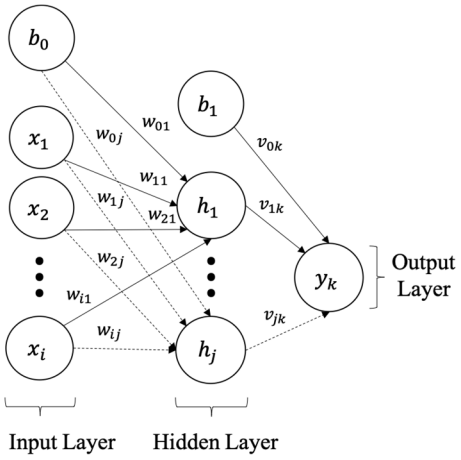

Fig. 1. MLP Architecture. In the above architecture *x* is a bit vector of length *i*, where *i* is equal to *n* the number of bits, *j* is the number of hidden units, and the dashed lines represent weight connections that are added during growing.

B. Activation Function

The activation function used for the MLP is the bipolar sigmoid function (1). This function has a range from -1 to 1, allowing a bipolar representation of the data (Fig. 2). It's derivative (2) is used when calculating the backpropagated error [17].

$$f(x) = \frac{2}{1 + \exp(-x)} - 1 \tag{1}$$

$$f'(x) = \frac{1}{2}(1 + f(x))(1 - f(x)) \tag{2}$$

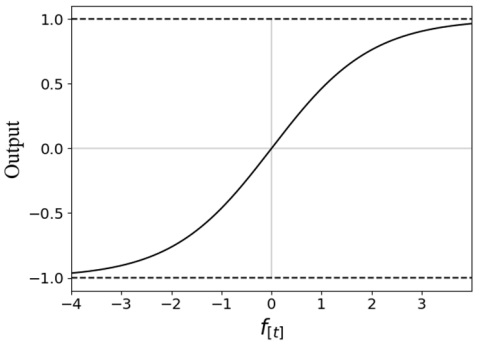

Fig. 2. Bipolar Sigmoid Activation Function.

Forward pass of the inputs through the hidden layer of the MLP network is derived according to:

$$h_j^{in} = b_0 + \sum_{i=1}^{n} x_i w_{ij} \tag{3}$$

$$h_j = f\left(h_j^{in}\right) \tag{4}$$

where x_i is the input vector, w_{ij} are the weight connections between the input and the hidden layer, b_0 is the bias on the hidden unit j, h_j^{in} is the activation of the hidden

unit j, h_j is the output from the hidden unit, and f the bipolar activation function defined in (1).

Once the output from the hidden layer is obtained, this information is passed forward to the output layer according to:

$$y_k^{in} = b_1 + \sum_{j=1}^{n} h_j v_{jk} \tag{5}$$

$$y_k = f(y_k^{in}) \tag{6}$$

where v_{jk} is the weight connections between the hidden layer and the output unit, b_1 is the bias on the output unit, y_k^{in} is the activation of the output unit and y_k the output signal.

C. Learning Algorithm

The learning for the MLP network is based on standard backpropagation. Once an output is obtained from the MLP network, the error term (δ_k) is computed according to (7) from the output layer and back-propagated to the hidden layer.

$$\delta_k = (t_k - y_k) f'(y_k^{in}) \tag{7}$$

where f' is the derivative of the activation function (2), and t_k is the associated target pattern. To improve convergence, targets were not set at the asymptotes, but at 0.8 and -0.8 respectively [17]. The error term from the output layer is then used to calculate the error term from the hidden layer (δ_j) according to:

$$\delta_j^{in} = \sum_{k=1} \delta_k v_{jk} \tag{8}$$

where δ_j^{in} is the summation of delta inputs from the output.

$$\delta_j = \delta_j^{in} f'\left(h_j^{in}\right) \tag{9}$$

Once the error terms have been calculated, the output unit updates its weight connections (10) and bias (11) by adding correction terms according to:

$$v_{jk}^{new} = v_{jk}^{old} + \eta(\delta_k h_j) \tag{10}$$

$$b_1^{new} = b_1^{old} + \eta\delta_k \tag{11}$$

where v_{jk}^{old} is the weight connections from the previous epoch, η the learning rate set to 0.1, v_{jk}^{new} represents the updated weight connections, b_1^{old} is the bias on the output layer from the previous epoch, and b_1^{new} is the updated bias term.

This process is then repeated by the hidden layer that uses its error term (δ_j) to create correction terms to update its own weights (12) and bias (13) according to:

$$w_{ij}^{new} = w_{ij}^{old} + \eta(\delta_j x_i) \tag{12}$$

$$b_0^{new} = b_0^{old} + \eta\delta_j \tag{13}$$

where w_{ij}^{old} is the weight connections from the previous epoch, w_{ij}^{new} is the updated weight connections, b_0^{old} is the bias on the hidden layer from the previous epoch, and b_0^{new} is the updated bias term.

The MLPs performance is calculated at the end of each epoch according to:

$$\text{MSE} = \frac{1}{m}\sum_{k=1}^{m}(t_k - y_k)^2 \tag{14}$$

where m is the input dimension given by 2^n, with $n \in \{-1,1\}$ being the number of bits, and MSE is the mean-squared error of the MLP network.

D. Learning Procedure

Learning for both MLPs is conducted in the same manner using batch training outlined by the following steps:

1) Initialization of weight connections at random values between -0.5 and 0.5.
2) Forward propagation where the input vector is broadcasted to all hidden units that sum their weighted input signals according to (3), and apply the bipolar activation function (1) to compute each of their respective outputs (4).
3) Continuation of forward propagation where the output unit receives the summation of weighted input signals according to (5), over which the bipolar activation function is applied (1) to compute its output (6).
4) Each output unit receives the associated target patterns for each respective input training patterns and calculates its error term (δ_k) which is then back-propagated to the hidden layer (7).
5) The hidden layer uses the back-propagated error and calculates its own error information term δ_j according to (8) and (9).
6) Weights are then updated according to (10) for the output layer and (12) for the hidden layer. Biases are also updated according to and (11) and (13).
7) Training continues until the maximum number of epochs is reach (set to 10^6) or if the MSE (14) falls below a minimum value set at 0.001.

The only exception between the fMLP and the gMLP learning procedures is the inclusion of the growing algorithm. The gMLP is only permitted to add units at the very end of each epoch after the network has updated its weight connections. When a new unit is added, its new connections are also initialized at random values between -0.5 and 0.5 and are then updated along with all the other weight connections during the next epoch.

E. Growing Algorithm

The growth of the population of hidden units is dictated by the growing algorithm (15). This algorithm is adapted and modified from the Single-species model with Allee effect outlined in [16]. This algorithm is used during each epoch to calculate the change in the size of the hidden unit population (i.e. growth rate). The addition of a new unit to the hidden layer only occurs when the growth rate reaches an integer value, as adding a proportion from a single unit is not plausible. The dynamics of the growing algorithm are characterized by:

$$\frac{dh_s}{dt} = MSE\left(1 - \frac{h_s}{c}\right)(h_s - \beta) \quad (15)$$

where c is the carrying capacity of the hidden layer environment (upper bound) determined by the fixed rule used, h_s is the current size of the hidden unit population, and β is a constant set to 0.2.

As the hidden unit population grows within the hidden layer, it receives direct performance feedback from the gMLP network in the form of the MSE. The MSE modulates the rate at which the hidden unit population can grow (Fig. 3). The MSE is inversely proportional to the growth rate. In this regard, as the gMLP learns, its performance can slow the rate of growth of the hidden layer. With respect to ANNs, during the learning process the error is reduced and should approach zero. With a gradually declining error, the growth rate is in a state of constant change that is dependent on the network's performance. In this context, even though the hidden layer has a maximum carrying capacity, the hidden unit population will grow towards the optimal number of hidden units required to reach an error close to zero. This is reflected in Fig. 4, where the curves: 10, 8, 6, 4, 2; depict hypothetical situations where the MSE is initiated at a value of 1 and converges to 0.001 at different numbers of hidden units within the hidden layer. These curves reflect the fluctuation in growth rate as a result of error feedback from the network's performance. Hence, the minimum error may either be reached at the carrying capacity or at a significantly lower hidden layer sizes, depending on task complexity or the defining characteristics of the network itself, such as the architecture or learning rule.

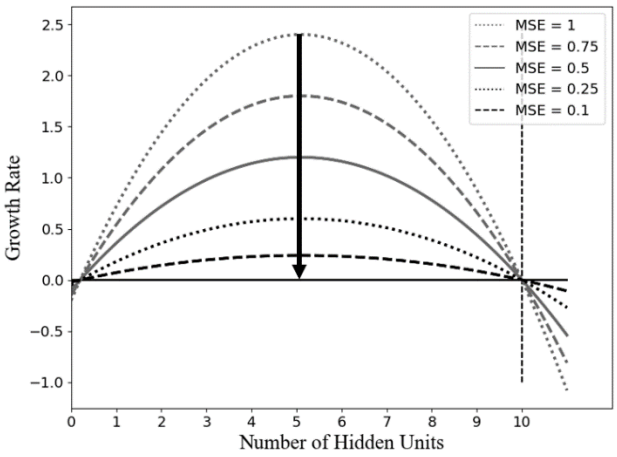

Fig. 3. Hidden Layer Growth Rate Function with Constant Error. In the above figure the growth rate of the hidden unit population is shown with respect to the gMLP network's MSE. Here the carrying capacity is fixed at 10 hidden units, and the MSE is held constant at values varied between values of 1 and 0.1.

F. Simulation: n-Bit Partity Problem

The *n*-bit parity vectors used in the following simulations were given a bipolar representation of -1 and 1. This set of simulations tested rules: 2^n, $2n$+1, and n+1; for determining hidden layer size for parity 2 to 7-bit problems, where n is the number of bits.

These rules were implemented in two identical MLP networks. The fMLP with a fixed number of hidden units given by the rules and the gMLP using the number of hidden units predicted by the rules as the carrying capacity.

III. Results

The effectiveness of the fMLP and the gMLP was evaluated for solving the n-bit parity problem for 2 to 7-bits across 100 trials per bit. To obtain an idea of just how the

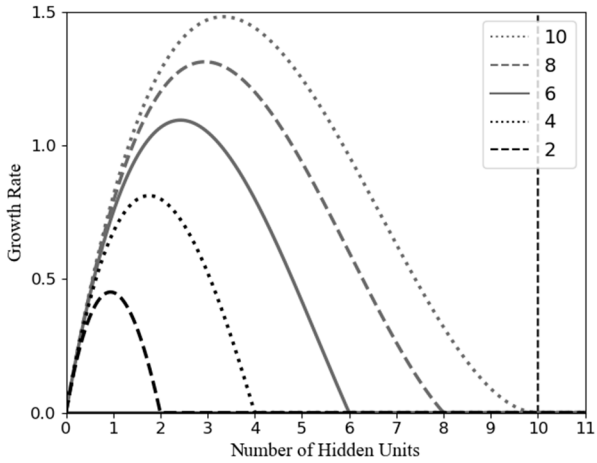

Fig. 4. Hidden Layer Growth Rate Function with Declining Error. In the above figure the growth rate of the hidden unit population is shown with respect to the gMLP network's MSE. Here the carrying capacity (c) is fixed at 10 hidden units, β is set to 0, and the MSE is declining from 1 to 0.001.

gMLP compares to the fMLP, a single trial of the 4-bit problem using the rule 2^n is examined in detail. Using the rule 2^n, the fMLP is set to 16 units. This means that this network has a total of 65 weight connections. As can be seen in Fig. 5a, the MSE of the fMLP decreases gradually until reaching the preset minimum error of 0.001. In contrast, the gMLP uses the rule 2^n to set the carrying capacity to a value of 16 for the 4-bit problem. This network starts with a single hidden unit and grows in an ascending step-like fashion before converging at a total of 5 hidden units (Fig. 6). Therefore, this network has a total of only 21 weight connections. As can be seen in Fig. 5b, the MSE of the network decreases in descending step-like fashion until it reaches the preset minimum error of 0.001. The plateaus seen in this figure indicate that the error is decreasing less and less between epochs and a new unit is needed. The spikes in error seen in Fig. 5b at the end of the plateaus occur simultaneously with the addition of a new unit to the network (see Fig. 6) and consequently the addition of new randomized weight connections. This initial spike in error is quickly reduced and allows the network to continue to train and further lower the MSE. To summarize, using the same rule, 2^n, the fMLP uses more hidden units and weight connections compared to the gMLP, but takes less training epochs.

To verify that this result was consistent, both networks were subjected to 100 trials of each bit problem from 2 to 7-bits. This process was repeated across the rules: 2^n, $2n$+1, and n+1; for a total of 21 conditions per MLP network. The results of the fMLP are shown in Table I, and the results of the gMLP are shown in Table II. Where the "*Predicted*" column gives the number of units that the rule indicates should be used and the "*Actual*" column gives the number of hidden units the gMLP network actually used across 100 trials. Due to the variability in gMLP performance, the number of hidden units in the hidden layer varies from trial to trial. As such, a range of hidden units is reported per bit in Table II from the least to most hidden units used across the 100 trials. To ensure full transparency, the frequency (percentage) of each hidden unit

size with respect to the 100 trials is reported across all the bit problems tested (Table III).

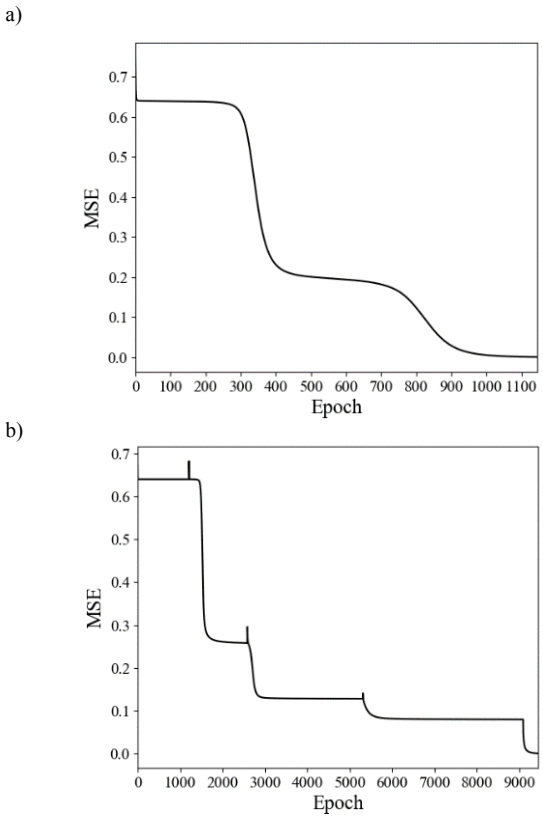

Fig. 5. Mean Squared Error (MSE) across MLP Training Epochs for a 4-bit Trial. a) MSE across fMLP training epochs when the rule 2^n determines a fixed hidden layer of 16 units. b) MSE across gMLP training epochs when the rule 2^n sets the carrying capacity (c) at 16, but the hidden layer only grows to 5 units.

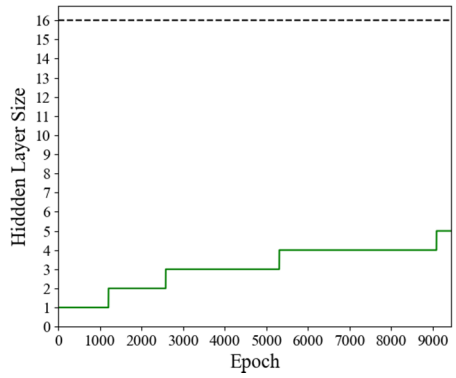

Fig. 6. Growth of the Hidden Layer Across Epochs for a 4-bit Trial. In the above figure the size of the hidden unit population is shown growing across gMLP training epochs. Here the carrying capacity, depicted by a dashed line, is determined by the rule 2^n to be 16 hidden units.

Dependent on the rule, the results of the fMLP for both 2 and 3-bit problems use between 2-8 hidden units (Table I). In contrast the gMLP uses between 2 and 3 hidden units regardless of the rule used (Table II). The percentages of each hidden unit size used across the 100 trials per each rule is shown in Table III. These results show that 80-100% of the time only two units are used for both the 2 and 3-bit problems. A visual of the quality of the classification for a single trial of the 2-bit problem is shown in Fig. 7. As can be seen in Fig. 7a-c, the fMLP has clearly defined classification across the three rules. With the fMLP using: 4 units for 2^n, 5 units for $2n$+1, and 3 units for n+1. In contrast the gMLP uses only 2 hidden units for all three rules. However, as can be seen in Fig. 7d-f, the gMLP is still able to maintain clear distinct classification borders.

Examining the results of the 4-bit problem in depth using the rule 2^n, the fMLP used a total of 16 hidden units (Table I). While the gMLP varied from 3-13 units across the 100 trials (Table II). However, examining these numbers closely 99% of the time the number of hidden units was between 3-6 and only 1% of the time was the hidden layer size 13 (Table III). This is reflected in Table III with a mode at 4 hidden units. Continuing to both $2n$+1 and n+1 rules, the fMLP used 9 and 5 units respectively (Table I). The gMLP varied between 3-7 units, but closer examination reveals that only 1% of the time was 7 units used. Additionally, both distributions have a mode at 4 (Table III).

At 5-bit and higher, the number of predicted units begins to show distinct variation between rules. At 2^n the fMLP used 32 units, 11 units at $2n$+1, and 6 units at n+1. Contrary to this, similarity can be seen across rules for the gMLP with a mode at 4 for all three distributions (Fig. 8a-c). For 2^n, 98% of the trials used 7 units or less. Both of the 11 and 25 units occur only 1% each. This effect is seen using the $2n$+1 rule, with 96% of the trials using 7 units or less. These results are again continued with the n+1 rule, where 100% of results occurring between 4-5 units. Similar results are observed for the 6-bit problem. Finally, for the 7-bit problem the fMLP, dependent on the rule, used a total of: 128, 15, and 8 hidden units. However, using the rule 2^n, which leads to using 128 hidden units, the network is unable to converge during learning for many of the 100 trials. The gMLP showed variation compared to previous results with respect to the rules 2^n and $2n$+1. For 2^n in the gMLP, 65% of trials used between 7 and 13 hidden units, however 30% of trials used between 119 and 128 hidden units (Table III). Similar results are found using the rule $2n$+1, with 56% of trials between 7 and 12 and 44% of trials using 14 or 15 units. These two rules show evidence of bimodal distributions. In spite of this a mode of 7 hidden units is still observed across rules (Table III).

IV. Discussion

In this study we introduced a novel constructive algorithm inspired from population dynamics to incrementally add units to a MLP. This algorithm utilizes a carrying capacity parameter that prevents the network from growing outbound. This endowed the network with a growth rate that allowed more rapid growth when the hidden layer size was far from capacity and a slower growth rate when it approached capacity. This carrying capacity was used in parallel with direct performance feedback from the network in the form of the MSE. The MSE scaled the growth rate with regard to the networks performance. This allowed for a dynamic self-governed growth, the rate of which was modulated both by network performance and hidden layer size.

To test the adaptability of this algorithm, we subjected the gMLP to increasing levels of task complexity for the n-bit problem, and compared our results to an identical fMLP. Our aim was to show that by using fixed rules for hidden layer

TABLE I. fMLP HIDDEN LAYER SIZE (AVERAGED OVER 100 TRIALS)

Bit	Rule for Number of Hidden Units								
	2^n			$2n+1$			$n+1$		
	Preditcted	*Actual*	*Avg.Epochs*	*Predicted*	*Actual*	*Avg.Epochs*	*Predicted*	*Actual*	*Avg.Epochs*
2	4	4	789	5	5	721	3	3	918
3	8	8	315	7	7	334	4	4	434
4	16	16	1 174	9	9	1 041	5	5	5358
5	32	32	461	11	11	514	6	6	8 596
6	64	64	6 693	13	13	3 967	7	7	10 535
7	128	128	703 721	15	15	12 917	8	8	140 490

TABLE II. gMLP HIDDEN LAYER SIZE (AVERAGED OVER 100 TRIALS)

Bit	Rule for Number of Hidden Units								
	2^n			$2n+1$			$n+1$		
	Preditcted	*Actual*	*Avg.Epochs*	*Predicted*	*Actual*	*Avg.Epochs*	*Predicted*	*Actual*	*Avg.Epochs*
2	4	2-3	2 685	5	2-3	2 463	3	2	7 674
3	8	2-3	3 173	7	2-3	3 276	4	2-3	3 888
4	16	3-13	25 678	9	3-7	14 084	5	3-5	26 056
5	32	4-25	22 941	11	4-10	50 334	6	4-5	103 412
6	64	5-63	60 598	13	5-13	93 686	7	5-7	261 749
7	128	7-128	321 232	15	6-15	527 059	8	6-8	631 603

TABLE III FREQUENCY OF HIDDEN UNITS IN gMLP ACROSS 100 TRIALS

Bit	Number of Hidden Units Across 100 Trials		
	Rule for Number of Hidden Units		
	2^n	$2n+1$	$n+1$
2	**2:99%**, 3:1%	**2:94%**, 3:6%	**2:100%**
3	**2:83%**, 3:17%	**2:89%**, 3:11%	**2: 93%**, 3:7%
4	3:14%, **4:57%**, 5:25%, 6:3%, 13:1%	3:23%, **4:63%**, 5:13%, 7:1%	3:27%, **4:69%**, 5:4%
5	**4:73%**, 5:19%, 6:4%, 7:2%, 11:1%, 25:1%	**4:71%**, 5:22%, 6:2%, 7:1%, 10:4%	**4:72%**, 5:28%
6	5:17%, **6:43%**, 7:18%, 8:11%, 9:2%, 10:1%, 12:1%, 13:1%, 17:1%, 19:1%, 22:1%, 60:1%, 63:1%	5:16%, **6:46%**, 7:21%, 8:7%, 9:1%, 10:2%, 11:1%, 12:5%, 13:1%	5:22%, **6:68%**, 7:10%
7	**7:38%**, 8:16%, 9:2%, 11:2%, 12:3%, 13:4%, 23:1%, 25:1%, 27:1%, 54:1%, 82:1%, 119:1%, 125:1%, 126:2%, 127:10%, 128:16%	**7:42%**, 8:10%, 9:1%, 10:1%, 11:3%, 14:12%, 15:32%	6:3%, **7:50%**, 8:47%

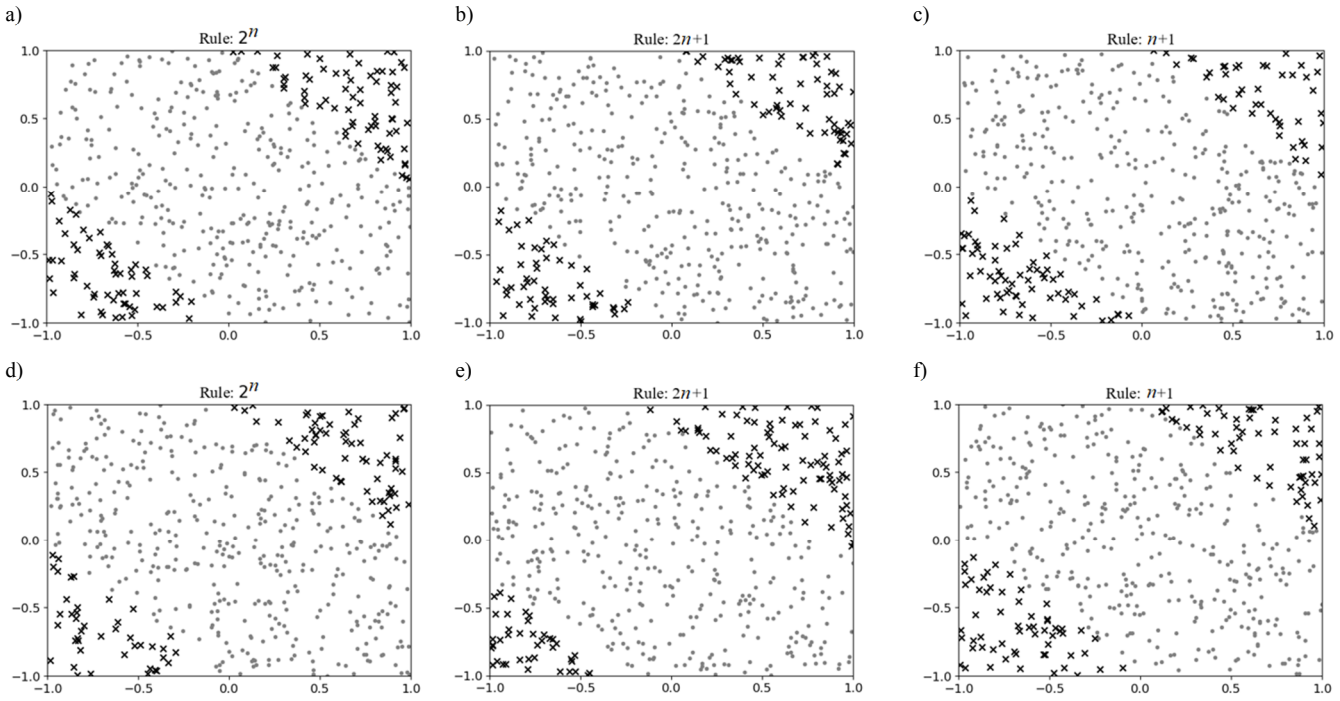

Fig. 7. Classification Borders of the MLP Networks for a Single Trial of the 2-bit Problem Across the Three Different Rules. a-c) fMLP classification quality across the three different rules for 500 random points. d-f) gMLP classification quality across the three different rules for 500 random points.

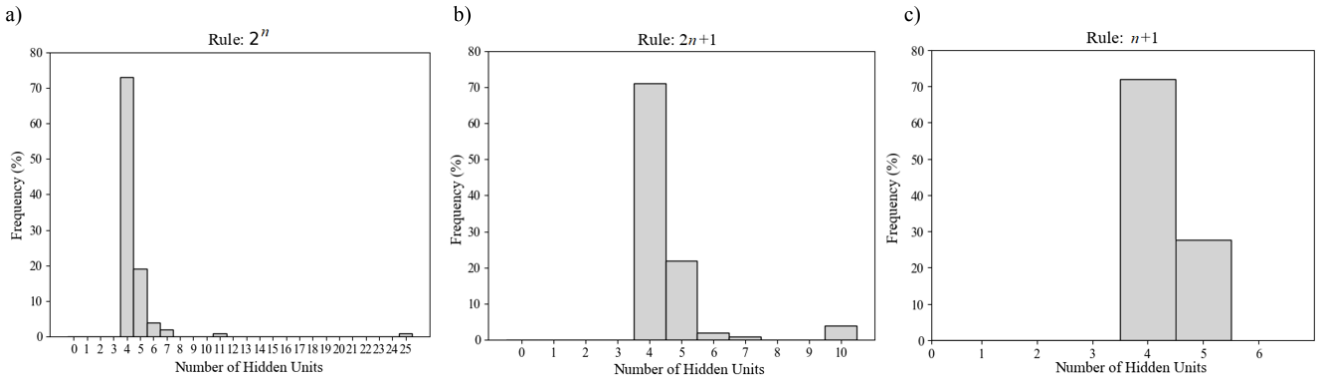

Fig. 8. Frequency of Hidden Units Needed by the gMLP across 100 Trials. a-c) Final number of hidden units from 100 trials for the 5-bit problem using three different rules to determine the hidden layer's carrying capacity (c).

size, flexible dynamic growth would lead to generalized approximations in hidden layer size regardless of the rule used. Even if an upper boundary of the carrying capacity is set by a fixed rule, the growing algorithm is capable of converging to a number of hidden units that would fall within the lower range of units required for solving the given task. This effect is consistent regardless of the rule used for deciding on the upper boundary of the carrying capacity.

As illustrated in our results, the gMLP was successful in showing a general hidden layer size regardless of the rule used. This is evident from the distributions for each of the three rules having the same mode for each bit problem. Additionally, as previously mentioned only one of these rules, n+1, is an actual generalized rule for solving the n-bit problem with a feedforward architecture. The other two rules 2^n and $2n$+1 were chosen to push both MLP networks to larger hidden layer sizes than the n-bit problem requires. Our results were clear for the fMLP. For the 7-bit problem, using too many hidden units, such as with the rule 2^n, caused the network during some trials to reach the maximum number of epochs set at one-million so as to prevent network convergence. Contrary to this, the gMLP showed more robust qualities and overall across 100 trials was able to converge to smaller topologies while still solving the task.

It wasn't until the 7-bit problem was reached where the gMLP experienced difficulties as evident from the two bimodal distributions for the rules 2^n and $2n$+1. This problem is not new to growing methods. One of the limitations of growing methods is that the networks can sometimes get trapped in local minima [13], [18]. This, as well as slow convergence are particularly pertinent problems with backpropagation. To escape the local minima adding more than one unit at a time could solve this issue. To address slow convergence, using a modified error term [19] or avoid using standard backpropagation and use faster training methods like Quickprop in Cascade Correlation [14]. Furthermore, for some of our results the final hidden layer size is sometimes less than the aforementioned minimum of n. For instance, with the 4-bit problem it can be seen that the gMLP uses only 3 hidden units. Previous work using analytical solutions has shown that the number of hidden units required to solve n-bit problems is governed by (n/2)+1 if even, and (n+1)/2 if odd [8]. According to this metric the 4-bit problem can be solved by a minimum of 3 hidden units.

Despite the success of the model for solving the n-bit problem, it is important to note the variation in training times between the fMLP and the gMLP. The gMLP took more training epochs, growing from a single hidden unit to the amount needed to perform the task. However, as previously mentioned optimizing training times for application was not the objective of this study. Rather, the primary objective was the proof of concept of a new growing algorithm inspired from populations dynamics and the demonstration of its effectiveness and plausibility. The main novelty of which is the inclusion of a carrying capacity and by extension a growth rate. Unlike previous methods such as with the DNC [5] that calculates the average error in relation to a user set minimum slope, the addition of new units to the hidden layer is not user determined. Rather, it is self-governed based on the growth rate which is directly scaled by the current size of the network in relation to the carrying capacity and simultaneously by the training performance of the network through the MSE. Even though a measure of error is used, there is no user set criterion for how much the error stops changing by to then introduce a new unit to the network.

A similar mechanism is seen using evolutionary algorithms for growing the network structure. For instance, the inclusion of a growth probability operator and an accompanying growth rate distribution to determine how many units the network grows by [20]. This has shown to exhibit higher growth probabilities at initial stages and lower growth probabilities at later stages, effectively speeding up the process of finding an optimal solution. Though this method is self-adaptive, it introduces randomness through mutations, requires an evaluation of fitness, and takes place across many networks with only the best performing ones are chosen to "survive" [20]. In the present algorithm, the growth rate is being utilized as a self-governed process, without making reference to processes governed by evolutionary dynamics. Additionally, only one network is grown and the population refers only to the hidden layer size. Therefore, the current dynamic growth algorithm introduced here can be seen as a hybrid between evolutionary and constructive approaches.

To validate the observed results, replication with other tasks is needed, such as the Double Moon problem. In this case, the gMLP would provide an estimate of the number of hidden units required to solve the task. This estimated value for the number of hidden units would provide an approximation of the general rule required to solve the Double Moon problem. The same way that our approach provided approximations close to rules mentioned for solving the n-bit Parity problem. If fixed rules that are task specific can be generalized to other tasks and still provide accurate results it would effectively validate the growing algorithm applied here. Additionally, it would be pertinent to test the

generalization with respect to classification of the growing network to over or under fitting the data. Such a validation could be achieved using a k-fold validation. Additionally, by combining this dynamic growth algorithm with more efficient learning approaches, such as Quickprop, we aim reduce training times to allow a more direct comparison between growing and fixed networks. Future work aims to refine this algorithm to incorporate neuronal pruning to allow the algorithm to have a negative growth rate to decay the neuronal population and obtain an optimal minimal topology. To govern such a method, one potential avenue would be to re-introduce the possibility of local extinction as outlined in the original population dynamics model [16]. Introduction of such a mechanism could create internal competition potentially allowing a synergistic effect to reach optimal topologies. Finally, it would be very interesting to explore how a multilayered network with two hidden layers could be governed by two of these algorithms, whereby growth of one layer could lead to pruning in another depending on the performance and needs of the network.

V. Conclusion

Dynamic growth inspired from population dynamics introduces a constructive approach that is self-governed. This approach can circumvent the need for arbitrary trial and error to determine optimal hidden topologies. Furthermore, it also avoids the problem of growing outbound by converging to smaller topologies on its own, according to how both the network performance and the hidden layer size modulate the growth rate. The self-governed dynamics of this growing algorithm merit further investigation.

Acknowledgment

The authors wish to acknowledge the support from the National Sciences and Engineering Research Council of Canada (NSERC).

References

[1] A. G. Y. P. K. and D. P. Gaurang Panchal, "Behaviour Analysis of Multilayer Perceptrons with Multiple Hidden Neurons and Hidden Layers," pp. 1793–8201, 2011.

[2] D. Liu, T.-S. Chang, and Y. Zhang, "A Constructive Algorithm for Feedforward Neural Networks With Incremental Training," *IEEE Trans. CIRCUITS Syst. Fundam. THEORY Appl.*, vol. 49, no. 12, 2002, doi: 10.1109/TCSI.2002.805733.

[3] R. Parekh, J. Yang, and V. Honavar, "Constructive neural-network learning algorithms for pattern classification," *IEEE Trans. Neural Networks*, vol. 11, no. 2, pp. 436–451, Mar. 2000, doi: 10.1109/72.839013.

[4] T. Y. Kwok and D. Y. Yeung, "Constructive algorithms for structure learning in feedforward neural networks for regression problems," *IEEE Trans. Neural Networks*, vol. 8, no. 3, pp. 630–645, 1997, doi: 10.1109/72.572102.

[5] T. Ash, "Dynamic Node Creation in Backpropagation Networks," *Conn. Sci.*, vol. 1, no. 4, pp. 365–375, Jan. 1989, doi: 10.1080/09540098908915647.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986, doi: 10.1038/323533a0.

[7] B. M. Wilamowski, D. Hunter, and A. Malinowski, "Solving Parity-N Problems with Feedforward Neural Networks," in *Proceedings of the International Joint Conference on Neural Networks*, 2003, vol. 4, pp. 2546–2551, doi: 10.1109/ijcnn.2003.1223966.

[8] R. Setiono, "On the solution of the parity problem by a single hidden layer feedforward neural network," *Neurocomputing*, vol. 16, no. 3, pp. 225–235, Sep. 1997, doi: 10.1016/S0925-2312(97)00030-1.

[9] Y. Lu, J. Yang, Q. Wang, and Z. J. Huang, "The upper bound of the minimal number of hidden neurons for the parity problem in binary neural networks," *Sci. China Inf. Sci.*, vol. 55, no. 7, pp. 1579–1587, Jul. 2012, doi: 10.1007/s11432-011-4405-6.

[10] H. K. Fung and L. K. Li, "Minimal feedforward parity networks using threshold gates," *Neural Comput.*, vol. 13, no. 2, pp. 319–326, Feb. 2001, doi: 10.1162/089976601300014556.

[11] R. C. Minnick, "Linear-Input Logic," *IEEE Trans. Electron. Comput.*, vol. EC-10, no. 1, pp. 6–16, Aug. 2009, doi: 10.1109/tec.1961.5219146.

[12] D. Hunter, H. Yu, M. S. Pukish, J. Kolbusz, and B. M. Wilamowski, "Selection of proper neural network sizes and architectures-A comparative study," *IEEE Trans. Ind. Informatics*, vol. 8, no. 2, pp. 228–240, May 2012, doi: 10.1109/TII.2012.2187914.

[13] S. Curteanu and H. Cartwright, "Neural networks applied in chemistry. I. Determination of the optimal topology of multilayer perceptron neural networks," *J. Chemom.*, vol. 25, no. 10, pp. 527–549, Oct. 2011, doi: 10.1002/cem.1401.

[14] S. E. Fahlman and C. Lebiere, "The Cascade-Correlation Learning Architecture," *Adv. Neural Inf. Process. Syst.*, pp. 524–532, 1990.

[15] Y. Hirose, K. Yamashita, and S. Hijiya, "Back-propagation algorithm which varies the number of hidden units," *Neural Networks*, vol. 4, no. 1, pp. 61–66, 1991, doi: 10.1016/0893-6080(91)90032-Z.

[16] G. Q. Sun, "Mathematical modeling of population dynamics with Allee effect," *Nonlinear Dynamics*, vol. 85, no. 1. Springer Netherlands, 01-Jul-2016, doi: 10.1007/s11071-016-2671-y.

[17] L. Fausett, "Fundamentals of Neural Networks — Architectures, Algorithms, and Applications," Prentice-Hall Inc., 1994, pp. 289–302.

[18] P. L. Narasimha, W. H. Delashmit, M. T. Manry, J. Li, and F. Maldonado, "An integrated growing-pruning method for feedforward network training," *Neurocomputing*, vol. 71, no. 13–15, pp. 2831–2847, 2008, doi: 10.1016/j.neucom.2007.08.026.

[19] W. Bi, X. Wang, Z. Tang, and H. Tamura, "Avoiding the local minima problem in backpropagation algorithm with modified error function," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. E88-A, no. 12, pp. 3645–3653, 2005, doi: 10.1093/ietfec/e88-a.12.3645.

[20] J. H. Ang, K. C. Tan, and A. Al-Mamun, "Training neural networks for classification using growth probability-based evolution," in *Neurocomputing*, 2008, vol. 71, no. 16–18, pp. 3493–3508, doi: 10.1016/j.neucom.2007.10.011.