Automated Generation of Environments to Test the General Learning Capabilities of AI Agents

Oliver J. Coleman
School of Computer Science and Engineering
University of New South Wales
Sydney, 2052, NSW, Australia
ocoleman@cse.unsw.edu.au

Alan D. Blair
School of Computer Science and Engineering
University of New South Wales
Sydney, 2052, NSW, Australia
blair@cse.unsw.edu.au

Jeff Clune
Evolving AI Lab
Computer Science Dept.
University of Wyoming
Laramie, WY, USA
jeffclune@uwyo.edu

ABSTRACT

Algorithms for evolving agents that learn during their lifetime have typically been evaluated on only a handful of environments. Designing such environments is labour intensive, potentially biased, and provides only a small sample size that may prevent accurate general conclusions from being drawn. In this paper we introduce a method for automatically generating MDP environments which allows the difficulty to be scaled in several ways. We present a case study in which environments are generated that vary along three key dimensions of difficulty: the number of environment configurations, the number of available actions, and the length of each trial. The study reveals interesting differences between three neural network models – Fixed-Weight, Plastic-Weight, and Modulated Plasticity – that would not have been obvious without sweeping across these different dimensions. Our paper thus introduces a new way of conducting reinforcement learning science: instead of manually designing a few environments, researchers will be able to automatically generate a range of environments across key dimensions of variation. This will allow scientists to more rigorously assess the general learning capabilities of an algorithm, and may ultimately improve the rate at which we discover how to create AI with general purpose learning.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning-Concept Learning, Connectionism and Neural Nets

General Terms

Experimentation, Algorithms

Keywords

Evolutionary robotics, Generative encodings, Neural networks, Learning, Neuromodulation

GECCO'14 July 12-16, 2014, Vancouver, BC, Canada
ACM Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2662-9/14/07 $15.00.
http://dx.doi.org/10.1145/2576768.2598257.

1. INTRODUCTION

A long-standing goal in artificial intelligence is producing agents that learn throughout their lifetime [24]. One technique for doing so is to combine evolutionary algorithms with algorithms for intra-life learning [1, 9, 19]. A challenge for scientists in this field is evaluating agents over more than a few environments [2, 6, 15, 17, 21–23, 26, 28, 33].

Designing environments is labour intensive, and any single given environment is potentially open to bias towards or against certain kinds of algorithms and models [9, 18]. Testing on only a few environments provides only a small sample size that may prevent accurate general conclusions from being drawn [9, 18].

To address these problems, we present a method for automatically generating environments for which the difficulty can be scaled in several key ways. This automated environment generation method allows a more rigorous assessment of the learning capabilities of new models and algorithms. In what follows, we first introduce the concepts behind automatically generating environments. We then present a brief case study that demonstrates how this technique can generate environments for which the difficulty can be varied over three dimensions. The study reveals differences between three neural network models that would not have been obvious without sweeping across these difficulty dimensions.

The overall goal of our paper is to encourage research into ways to automatically test algorithms across many different, important dimensions of variation. Doing so may ultimately improve the rate at which we discover how to create AI capable of general purpose learning.

2. PREVIOUS WORK

While the technique in this paper is potentially beneficial for any type of learning agent, in this paper we focus on its benefits in the field of evolving *plastic* neural networks, i.e. those that *learn within their lifetimes*. Evaluating the fitness of such agents requires testing the agent's ability to learn and exploit information about its environment to achieve goals. Such evaluations require producing environments that are different in some way each generation or that change during a fitness evaluation [6, 17, 26, 28].

Initial studies on evolving plastic neural networks that perform intra-life learning focused on supervised learning [2, 6, 12]. Nearly all subsequent research has focused on reinforcement learning [15, 17, 21–23, 26, 28, 33]. In most reinforcement learning experiments the fitness of an agent is the amount of reward it receives in its lifetime [17, 26, 28] or is strongly correlated with the reward received [15, 22].

To maximize rewards, agents typically must learn an association that changes during their lifetime, such as which colour flower provides the most nutrition in a simple bee foraging task [17, 20, 27], which object types are food or poison in a more complex foraging task [28], or which arm of a T-maze contains the highest-value reward [20, 21, 26].

There have been variations on these kinds of environments designed to increase their difficulty. For example variations on the T-maze include the double T-maze, which has four arms instead of two [20, 26], a version requiring the agent to learn a possibly non-linear association between the perception of objects and their reward value [21], and a continuous version where the only input to the controller, besides a reward signal, comes from 5 range-finder sensors [22].

Possibly one of the most difficult tasks studied to date, in terms of the size of the exploration space and the number of associations to memorise, is a purely associative task in which agents learn associations between each possible input and output pattern [33, 34]. Aside from an input reward signal, the input and output vectors were four binary values. To simplify the task only one bit of either was set to 1, the rest were 0. The number of possible combinations to learn is 16, or four times as many as the double T-maze.

Over all previous work on evolving plastic neural networks that perform intra-life learning, only a handful of environments or tasks have been considered. The tasks have typically required a simple exploration strategy to determine which of a small set of behaviours should be conducted.

In any given study a newly introduced algorithm or model, such as an encoding scheme or neural network model, is typically assessed on only one or two environments. Furthermore, the variations of environments typically alter the environment in a single way. While it is understandable that few environments are tested, because designing environments is labour-intensive, the unfortunate result is that we problematically attempt to draw general conclusions about new algorithms and models from a very small sample size.

A method for automatically generating a broad range of environments would be beneficial in order to more rigorously assess the learning abilities of new models and algorithms.

3. METHODS

In the field of reinforcement learning, environments are often represented by a Markov Decision Process (MDP) that consists of a finite set of discrete states [24]. To move from one state to another in the environment an agent must perform one of a finite number of actions (with some actions possibly having an effect only in some states). Some state transitions confer a reward value. The task for the agent is to maximise the reward it receives over its lifetime, usually by performing an exploration of the environment to determine the expected future reward for each action in each state and then exploiting this knowledge.

A key insight of this paper is that this traditional way of representing reinforcement learning environments permits us to algorithmically generate novel environment instances with tunable difficulty over multiple dimensions, hereafter referred to as *difficulty dimensions*. To generate an environment, each state is assigned a random mapping of actions to transitions to other states, and each state transition is assigned a randomly selected reward value.

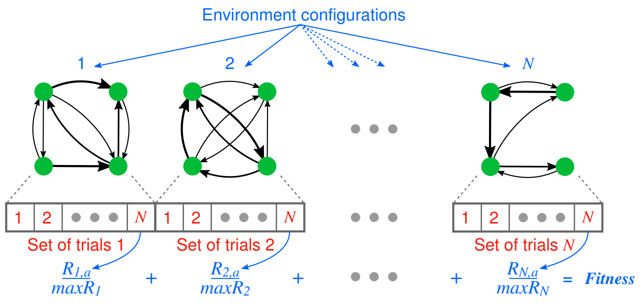

Environment configurations
1
2
N
1 2 ● ● ● N
Set of trials 1
Set of trials 2
Set of trials N
$\frac{R_{1,a}}{maxR_1}$ + $\frac{R_{2,a}}{maxR_2}$ + ● ● ● + $\frac{R_{N,a}}{maxR_N}$ = Fitness

Figure 1: **Automatically Generated Environments.** For a single evolutionary run we generate N MDP environments that all have the same number of states and number of possible actions. We refer to these as *environment configurations*. A fitness evaluation consists of multiple *sets of trials*, with each environment configuration being presented for several consecutive trials – a set of trials – before the next environment configuration is presented for a set of trials. This process is repeated until all N environment configurations have been presented. Following [34], we reset the network at the beginning of each set of trials because we are testing the ability to learn, not the ability to forget and learn. Thus each set of trials simulates a lifetime for the agent, providing a more statistically informed evaluation of the general learning capabilities of the agent by avoiding the noise introduced by agents that just happen to work well for a particular ordering of the environment configurations. N equals the number of environment configurations, allowing an agent that can perform a good behaviour for each environment configuration to try each behaviour in turn within a set of trials until it discovers the best behaviour for the current configuration. The reward $R_{x,a}$ received by agent a in the last trial of each set of trials x, is the proportion of the maximum possible reward $maxR_x$ for the environment configuration presented in that set of trials. The fitness is calculated as the sum of these proportions over all sets of trials.

3.1 Assessing Learning Capabilities

As noted previously, assessing the learning capabilities of an agent in an evolutionary framework requires producing environments that are either different each generation or that change in some way during the agent's lifetime. Following from previous work, for the case study in this paper we adopt the latter approach.

Figure 1 describes how an agent's fitness is evaluated. The task of the agent is to determine, over a given set of trials, the current *environment configuration* it is in and adopt the optimal behaviour for that configuration. Switching between environment configurations during the agent's lifetime, with each configuration being presented for several consecutive trials before the switch, is similar in spirit to most environments from previous work on evolving plastic neural networks. For example, the T-maze environment has two configurations, one for each reward location. The agent must determine which configuration it is in and adopt the appropriate behaviour until the environment configuration changes.

One strategy an evolving agent can take, which frequently emerged in previous work [17, 21–23, 26, 28, 33], is for an agent to perform multiple fixed behaviours, one for each environment configuration, and iterate over those behaviours until the correct one is found in a given set of trials. The minimum number of trials per set of trials required for an agent to try each fixed behaviour in turn until it finds the correct one for the environment configuration being presented is thus the number of environment configurations. Thus in this case study the number of trials in a set of trials is the same as the number of environment configurations.

Only the reward received in the last trial of each set of trials contributes to the fitness. Including reward only from the last trial promotes agents that are able to determine which environment configuration they are operating in and to then adopt the best behaviour for that configuration, thereby collecting the highest reward in the last trial. Specifically, the fitness of an agent is calculated as the sum of the proportion of the maximum possible reward that can be earned in the last trial of each environment configuration the agent encounters in a lifetime. The agent – here, a neural network – is reset to its initial state between each set of trials.

Some of the dimensions over which the difficulty of this task can be varied are: the number of states; the number of available actions; the number of environment configurations; whether state transitions are probabilistic; whether the agent can directly observe the state of the environment; how regular the structure of the state transitions is (e.g. constraining transitions such that the states form a grid); and the length of a single trial.

In this case study we demonstrate how environments can be automatically generated along three different dimensions of variation. Where not specified, the number of actions and the length of trials are set to 4 and the number of environment configurations per run is 8. For any given state, the proportion of actions that have an effect – for which a state transition will occur – is 0.75, and the proportion of state transitions that provide a reward value is 0.5. When generating environments these proportions are evaluated probabilistically. All environments are deterministic, the environment state is directly observable by the agent and the structure of the environments is unconstrained.

For these environments, the complexity of the optimal behaviour for a configuration increases as the *available actions* and *trial length* difficulty parameters are increased.

3.2 Neural Network Models

Our case study examines three different neural network models, which control agents operating in MDP environments. We test these three treatments across three dimensions of environmental difficulty to demonstrate that automatically generated environments can reveal differences between neural models. The three neural network models are (1) non-plastic, fixed-weight, (2) Hebbian learning, and (3) neuromodulated Hebbian learning.

Fixed-Weight

The *Fixed-Weight* model is a typical fixed-weight, recurrent neural network in which the neurons have a sigmoidal, rate-based activation function [16].

Plastic-Weight

The *Plastic-Weight* model is identical to the *Fixed-Weight* model, but instead of fixed weights it has a more general form of Hebbian learning [13] introduced by Niv et al. [17]. Whereas Hebbian learning increases the strength of a connection if both the pre-synaptic and post-synaptic neuron fire together, the parametrised model allows connections to both increase or decrease, and do so in more situations. The model consists of four terms: *(A)* a correlation term that only adjusts the weight when the pre- and post-synaptic activation signals are correlated; *(B)* a pre-synaptic term that adjusts the weight based only on the pre-synaptic signal; *(C)* a post-synaptic term that adjusts the weight based only on the post-synaptic signal; and *(D)* a constant term that adjusts the weight only as a function of time, independently of pre- or post-synaptic activation signals.

The coefficients of these terms, as well as overall "learning rate" η, are the parameters of the model:

$$\Delta w = \eta(Axy + Bx + Cy + D) \tag{1}$$

where x and y are the pre- and post-synaptic activation signals respectively. A, B, C, D, and η are evolvable.

Modulated Plasticity

The *Modulated Plasticity* model is identical to the *Plastic-Weight* model, but adds a model of neuromodulation [22, 26], which allows learning to be turned on and off for a subset of connections based on environment stimuli and the state of other neurons. Thus the neural network can selectively turn learning on or off in a subset of connections depending on the situation that the agent is in. Our implementation of neuromodulation is identical to that of Risi and Stanley [22].

In this model some connections carry a modulatory signal rather than a normal activation signal. Every neuron has a modulatory activation level as well as the standard activation. The modulatory activation level is calculated similarly to that of the normal activation level:

$$m_i = \sum_{w_{ji} \in \text{Mod}} w_{ji} o_j \tag{2}$$

where m_i is the modulatory activation for neuron i, w_{ji} is the weight of the synapse connecting neuron j to i, o_j is the output of neuron j and Mod is the set of modulatory synapses. The modulatory activation level m_i modulates the plasticity of regular (i.e. non-modulatory) connections leading into neuron i. Based on the plasticity model from Niv et al. [17] this gives the updated plasticity rule:

$$\Delta w = \tanh(m_i/2)\eta(Axy + Bx + Cy + D) \tag{3}$$

3.3 HyperNEAT

This section is from [8] with minor modification. In 2007 an encoding was introduced called Compositional Pattern Producing Networks (CPPNs), which abstracts the process of natural development without requiring the simulation of diffusing chemicals [29]. When CPPNs encode ANNs, the algorithm is called HyperNEAT [30], which is described below. A key idea behind CPPNs is that complex patterns can be produced by determining attributes of their phenotypic components as a function of their geometric location. This idea is based on the belief that cells (or higher-level modules) in nature often differentiate into their possible types as a function of where they are situated in geometric space. For example, for some insects, a segment at the anterior pole should produce antennae and a segment at the posterior pole should produce a stinger.

Components of natural organisms cannot directly determine their location in space, so organisms have evolved developmental processes that create chemical gradients that organismal components use to figure out where they are and, thus, what to become [5]. For example, early in the development of embryos, different axes (e.g., anterior-posterior) are indicated by chemical gradients. Additional gradients signaled by different proteins can exist in the same area to represent a different pattern, such as a repeating motif. Downstream genes, such as Hox genes, can then combine repeated and asymmetric information to govern segmental differentiation [5]. Further coordinate frames can then be set up within segments to govern intra-module patterns.

3.3.1 CPPNs

One of the key insights behind CPPNs is that cells *in silico* can be directly given their geometric coordinates. The CPPN genome is a function that takes geometric coordinates as inputs and outputs the fate of an organismal component. When CPPNs encode two-dimensional pictures, the coordinates of each pixel on the canvas (e.g., $x = 2$, $y = 4$) are iteratively passed to the CPPN genome, and the output of the function is the color or shade of the pixel (Figure 2).

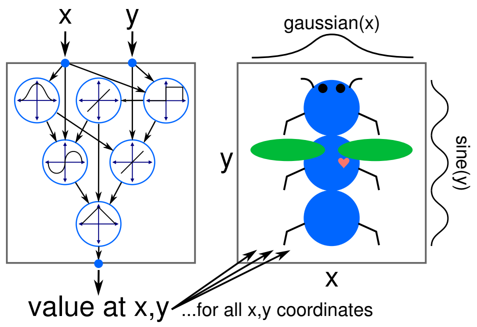


Figure 2: **Compositional Pattern Producing Networks.** CPPNs compose mathematical functions to generate regularities, such as symmetries and repeated modules, with and without variation. This figure is adapted from [29].

Each CPPN is a directed graph in which every node is itself a single function, such as sine or Gaussian. The nature of the functions can create a wide variety of properties, such as symmetry (e.g., a Gaussian function) and repetition (e.g., a sine function) that evolution can exploit. Because the genome allows functions to be comprised of other functions, coordinate frames can be combined and hierarchies can develop. For instance, a sine function early in the network can create a repeating theme that, when passed into the symmetrical Gaussian function, creates a repeating series of symmetrical motifs (Figure 2). This procedure is similar to the natural developmental processes described above [5].

The links that connect and allow information to flow between nodes in a CPPN have a weight that can magnify or diminish the values that pass along them. Mutations that change these weights may, for example, give a stronger influence to a symmetry-generating part of a network while diminishing the contribution from another part.

When CPPNs are evolved artificially with humans performing the selection, the evolved shapes look complex and natural (Figure 3) [25]. Moreover, these images display the features in natural organisms that indirect encodings were designed to produce, namely, symmetries and the repetition of themes, with and without variation.

3.3.2 Encoding ANNs with CPPNs

In the HyperNEAT algorithm, CPPNs encode ANNs instead of pictures, and evolution modifies the population of CPPNs [11, 30]. HyperNEAT evolves the weights for ANNs with a fixed topology. The ANNs in the experiments in this paper feature a two-dimensional, $m \times n$ Cartesian grid of inputs and a corresponding $m \times n$ grid of outputs. If an experiment uses an ANN with hidden nodes, the hidden nodes are placed in their own two-dimensional layer between the input and output grids. Recurrence is disabled, so each of the $m \times n$ nodes in a layer has a link of a given weight to each of the $m \times n$ nodes in the proximate layer, excepting output nodes, which have no outgoing connections. Link weights can be zero, functionally eliminating a link.

The inputs to the CPPNs are a constant bias value and the coordinates of both a source node (e.g., $x_1 = 0, y_1 = 0$) and a target node (e.g., $x_2 = 1, y_2 = 1$) (Figure 4). The CPPN takes these five values as inputs and produces one or two output values, depending on the ANN topology. If there is no hidden layer in the ANN, the single output is the weight of the link between a source node on the input layer and a target node on the output layer. If there is a hidden layer, the first output value determines the weight of the link between the associated input (source) node and hidden-layer (target) node, and the second output value determines the weight of the link between the associated hidden-layer (source) node and output-layer (target) node. All pairwise combinations of source and target node coordinates are iteratively passed as inputs to the CPPN to determine the weight of each ANN link. HyperNEAT can thus produce patterns in weight space similar to the patterns it produces in two-dimensional pictures (Figure 3).

An additional novel aspect of HyperNEAT is that it is one of the first neuroevolutionary algorithms capable of exploiting the geometry of a problem [7, 10, 11, 30]. Because the connection weights between nodes are a function of the geometric positions of those nodes, if those geometric positions represent aspects of the problem that are relevant to its solution, HyperNEAT can exploit such information. For example, when playing checkers, the concept of adjacency (on the diagonals) is important. Connection weights between

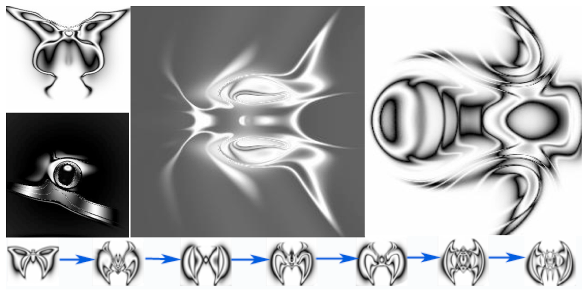


Figure 3: **Images Evolved with CPPNs.** Displayed are pictures from picbreeder.org [25], a website where visitors select images from a population evolved with the CPPN indirect encoding, which is also used in HyperNEAT. The bottom row shows images from a single lineage. Arrows represent intermediate forms that are not pictured.

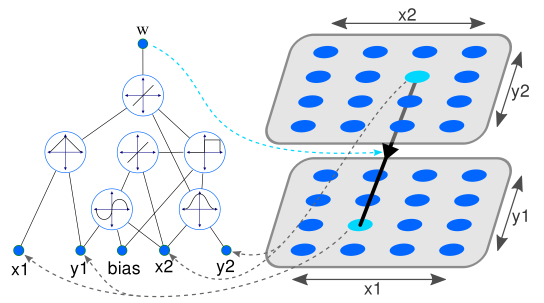


Figure 4: **HyperNEAT Produces ANNs from CPPNs**. Weights are specified as a function of the geometric coordinates of the source node and the target node for each connection. The coordinates of these nodes and a constant bias are iteratively passed to the CPPN to determine each connection weight. If there is no hidden layer, the CPPN has only one output, which specifies the weight between the source node in the input layer and the target node in the output layer. If there is a hidden layer in the ANN, the CPPN typically has two output values, which specify the weights for each connection layer. Alternatively inputs may be added to designate the z coordinates for the source and target nodes.

neighbouring squares may need to be different than weights between distant squares. HyperNEAT can create this kind of connectivity motif and repeat it across the board [10, 11]. Producing such a regularity would be more difficult with an encoding that does not include geometric information, because there would be no easy way for such an algorithm to identify which nodes are adjacent.

Variation in HyperNEAT occurs when mutations or crossover change the CPPNs. Mutations can add a node, which results in the addition of a function to the network, or change its link weights. Typical functions for CPPNs are sine, sigmoid, Gaussian, and linear. The evolution of the population of CPPN networks in HyperNEAT occurs according to the principles of the widely used NeuroEvolution of Augmenting Topologies (NEAT) algorithm [31]. NEAT, which was originally designed to evolve ANNs, can be fruitfully applied to CPPNs because a population of CPPNs is similar in structure to a population of ANNs.

The NEAT algorithm contains three main components [31, 32]. First, it starts with small genomes that encode simple networks and slowly complexifies them via mutations that add nodes and links to the network. This complexification enables the algorithm to evolve the network topology in addition to its weights. Second, NEAT implements a fitness-sharing mechanism via a speciation method that preserves diversity in the population and allows new innovations time to be tuned by evolution before forcing them to compete against rivals that have had more time to mature. Finally, historical information stored in genes helps to perform crossover in a way that is effective, yet avoids the need for expensive topological analysis. A full explanation of NEAT can be found in Stanley & Miikkulainen [31].

Our implementation of HyperNEAT differs in three ways from the original. First, speciation is implemented with a K-means clustering method. Second, in the original NEAT algorithm nodes are only added by splitting an existing connection, but in our implementation nodes could also be added anywhere along with two new connections to connect it to the existing network. Third, in the original formulation of HyperNEAT a connection is not expressed if the magnitude of the weight output of the CPPN is below a threshold, but in an extension called Link Expression Output (LEO) a separate output is used to determine whether a connection should be expressed. This method is employed in this case study. The key insight of LEO is that it makes easier to evolve the pattern of which connections should exist independently from the pattern of weight values [35].

3.4 Synaptic Plasticity Parameter Classes

Rather than encoding each parameter of the synaptic plasticity model for each individual connection, we take the same approach as Stanley et al. [28], in which the genome defines the parameter values for several plasticity rules, or classes, and then encodes which rule each connection references. Two benefits of this approach are that it reduces the overall search space and the classes can be optimised separately from the genes that determine which class a connection belongs to. Additionally, this approach bears some resemblance to natural neural networks, which tend to have numerous, but specific, types of neurons, neurotransmitters, and synapses that perform different functions [3, 4, 14, 36].

Unlike the model in [28], which is based on a direct encoding, we adapt the approach to HyperNEAT by evolving CPPNs that have $n+1$ outputs that specify the class for a connection or a special no plasticity class, where the number of classes is n. This approach is employed for all plastic neural network models considered in this case study.

3.5 Experimental Details

For each treatment 50 runs of evolution are performed, with each run lasting 2000 generations. At the start of a run multiple environment configurations are generated – as determined by the environment configurations difficulty dimension setting – which are used throughout that run.

Each run starts with a different random seed, however the random seed used to generate environment configurations at the start of a run varies only as a function of the run number within a treatment. Thus for a given set of environment generation parameters, such as the number of states and actions, the same environment configurations are generated for the same run number for each of the different neural network model treatments. This allows a more direct and fair comparison between treatments.

The parameter settings for HyperNEAT are similar to those in [30] except for the following differences: the population size is 1000; the K-means speciation method uses 32 clusters; the available functions for CPPNs are Sigmoid, Gaussian, Sine, absolute, ramp, linear, and sign; the probabilities of adding a CPPN node by replacing an existing connection or adding a node with two new connections are both 0.02; and CPPN weights are mutated with 0.15 probability by perturbation from a normal distribution with σ=0.3.

Evolved neural networks consist of an input layer, a hidden layer and an output layer, with the input and output layer sizes based on the number of states and number of actions that the generated environments are composed of (Figure 5). The networks are potentially fully recurrent within and between all layers, except the input layer, with LEO determining whether a potential connection is expressed.

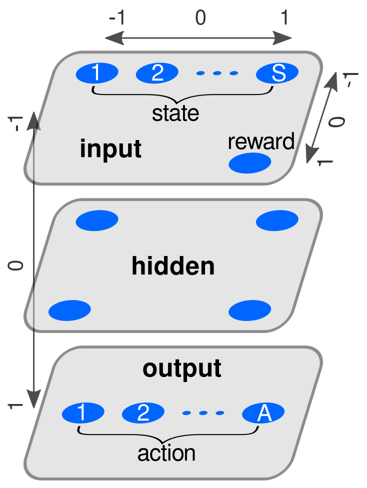


Figure 5: **Neural Network Topology for All Experiments.** S and A are the number of states and actions in the environment respectively.

All software, experiment settings, and data are available upon request or at *http://ojcoleman.com/publications*.

4. RESULTS AND DISCUSSION

We test the performance of three neural network models over automatically generated environments that sweep across three difficulty dimensions. Each dimension is scaled from easier to harder (Figure 6). We compare how the network models respond to changes in each difficulty dimension and across the difficulty dimensions. All statistical analyses in this case study are performed with R's Mann–Whitney–Wilcoxon rank sum test.

Across all difficulty dimensions, when the difficulty parameter is greater than or equal to 4, the Plastic-Weight and Modulated Plasticity models significantly outperform the Fixed-Weight model in nearly all cases ($p < 0.01$).

Interestingly, the Modulated Plasticity model does not typically perform significantly better than the Plastic-Weight model, and in fact the Plastic-Weight model significantly outperforms it when the number of actions is increased to 8 and beyond ($p < 0.001$; Figure 6b). The only case where the Modulated Plasticity model significantly outperforms the Plastic-Weight model ($p < 0.05$) is when the number of environment configurations is 16 (Figure 6a).

The results reveal differences between the neural network models that are not obvious without sweeping across these dimensions of difficulty. For example, as the number of actions is increased, the performance of the Modulated Plasticity model and the Plastic-Weight model diverge drastically, whereas these two models perform almost identically when sweeping across the other two dimensions of difficulty.

If we were to compare these neural network models over any one difficulty dimension, then very different conclusions could be drawn from the results. For example, if the number of actions is not considered then we may have drawn the general conclusion that the Modulated Plasticity and Plastic-Weight models are very similar in performance, and alternatively if only the number of actions are considered then we could draw the conclusion that, given a sufficient number of actions, the Plastic-Weight model outperforms the Modulated Plasticity model.

Similarly, if we compare these neural network models over any one difficulty parameter value then very different conclusions could be drawn from the results. For example, if only a difficulty parameter value of 4 is studied, we may draw the general conclusion that the Plastic-Weight and Modulated Plasticity models do not perform significantly differently. However if we study a difficulty parameter value of 8 only, then we may draw the conclusion that, at least in some cases, the Plastic-Weight model significantly outperforms the Modulated Plasticity model. More generally, without a wide sweep beyond 4 available actions the drastic difference between the Plastic-Weight and Modulated Plasticity models is not evident. Even a seemingly wide sweep of 2, 3, 4 and possibly even 5 available actions may not show the difference between the two models.

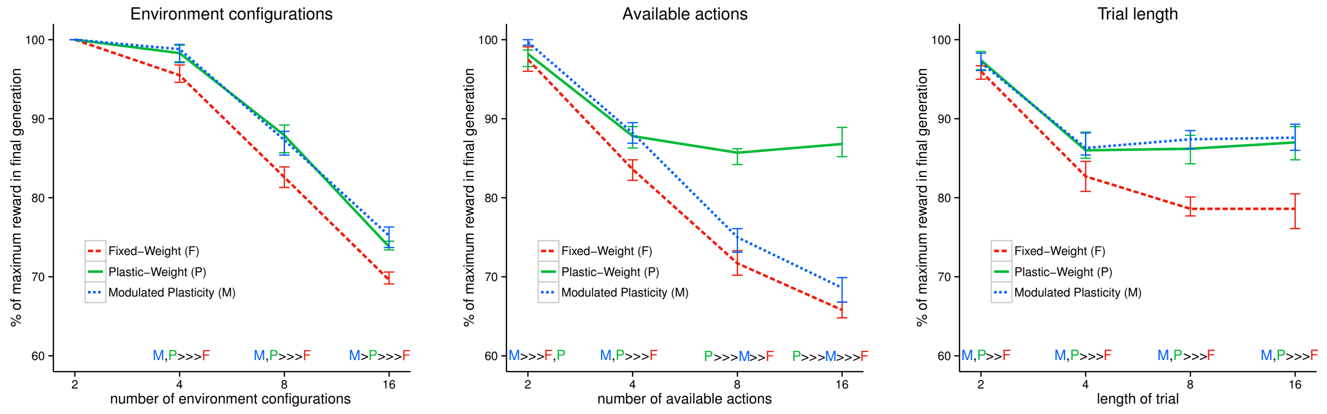
Another interesting performance difference that emerges between the different environment difficulty dimensions is that increasing the trial length beyond 4 does not affect the performance of any of the neural network models very much. On the other hand, increasing the number of environment configurations beyond 4 greatly reduces the performance of every neural network model. Thus if we do not sweep over all of these environment difficulty dimensions we may arrive at very different conclusions about how the performance of all neural network models are affected by increases in the difficulty of the environments.

These kinds of difficulty dimensions and difficulty parameter settings are reminiscent of those chosen in previous research on evolving plastic neural networks that learn during their lifetime [2, 6, 15, 17, 21–23, 26, 28, 33] in terms of the size of the memory requirements, kinds of environment exploration strategies, and complexities of behaviour required. For example, increasing the T-maze environment from a single T-maze, which can be solved with 2 fixed behaviours, to the double T-maze [26], which requires 4 fixed behaviours. Thus the results from the case study presented in this paper are relevant to the kinds of tasks and environments employed in our field to assess, and draw conclusions about, the learning capabilities of new models and algorithms. The results of this case study strongly suggest that a more rigorous assessment of learning capabilities, such as that provided by the automated environment generation method introduced in this paper, is not only useful but perhaps necessary to gain a more accurate picture of learning capabilities.

5. FUTURE WORK

While the scope of this paper is to introduce the automated environment generation method, in future work we will use this method to rigorously assess and analyze the performance characteristics of existing models and algorithms. For example, in the case study in this paper, the Modulated Plasticity model did not outperform the Plastic-Weight model. Testing many variants of these two models across different difficulty dimensions will allow us to better understand why.

The case study in this paper sweeps across only three of many possible difficulty dimensions, and studies only three neural network models. Future work will explore other difficulty dimensions and include more models and algorithms. We will also study not just environments that change *within*



(a) (b) (c)

Figure 6: **Performance of the three neural network models over sweeps across three difficulty dimensions in automatically generated environments.** The median performance (±95% bootstrapped confidence intervals) of the best performing neural network from the final generation over all runs are shown. Statistical significance is indicated by $>$ ($p < 0.05$), $>>$ ($p < 0.01$) and $>>>$ ($p < 0.001$).

a generation, but also environments that change *across* generations.

The discrete MDP environment in this paper cannot represent many real-world tasks. Two reasons are the inability to accurately represent continuous values and a lack of environment *dynamics*: environment states that change over time due to the laws of physics. Another direction for future work is adding the ability to automatically generate environments with continuous values and such physical dynamics. More generally, being able to automatically generate a broader class of environments will further increase our ability to assess the general properties of models and algorithms.

6. CONCLUSION

In this paper we have introduced a method for automatically generating environments for which the difficulty can be scaled in many dimensions. This method makes it easier to conduct a more rigorous assessment of the learning capabilities of agents that perform intra-life learning. The automated environment generation method addresses three key issues: (1) any single environment is potentially open to bias towards or against certain kinds of models and algorithms; (2) manually creating environments is labour intensive, at best taking time away from focusing on other aspects of research and, at worst, meaning that sweeps across key difficulty dimensions are not performed; and, most significantly (3) testing new models and algorithms on one or a few environments provides only a small sample size that may prevent accurate general conclusions from being drawn.

The utility of automated environment generation is not limited to assessing the learning capabilities of evolved agents or agents based on neural networks. Rather, the automated environment generation method could be applied to assessing the learning capabilities of any kind of agent that is able to operate in MDP environments, regardless of how the agent is developed or what kind of learning model the agent is based on. Thus the automated environment generation method presented in this paper has a potentially broad applicability to the field of artificial intelligence, and may ultimately accelerate the rate at which we discover how to create general purpose AI.

References

[1] D. Ackley and M. Littman. Interactions between learning and evolution. *Artificial life II*, 10, 1991.

[2] J. Baxter. The evolution of learning algorithms for artificial neural networks. *Complex Systems*, 1993.

[3] G. Bi and M. Poo. Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.*, 18(24):10464–10472, Dec. 1998.

[4] R. Cantrup, R. Dixit, E. Palmesino, S. Bonfield, T. Shaker, N. Tachibana, D. Zinyk, S. Dalesman, K. Yamakawa, W. K. Stell, R. O. Wong, B. E. Reese, A. Kania, Y. Sauvé, and C. Schuurmans. Cell-type specific roles for PTEN in establishing a functional retinal architecture. *PLoS ONE*, 7(3), Mar. 2012.

[5] S. Carroll. *Endless forms most beautiful: The new science of evo devo and the making of the animal kingdom.* WW Norton & Company, 2005.

[6] D. Chalmers. The evolution of learning: An experiment in genetic connectionism. In *Proc. of the 1990 connectionist models summer school*, 1990.

[7] J. Clune, C. Ofria, and R. T. Pennock. The sensitivity of HyperNEAT to different geometric representations of a problem. In *Proc. of the 11th Annual Genetic and Evolutionary Computation Conference*, 2009.

[8] J. Clune, K. Stanley, R. Pennock, and C. Ofria. On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation*, 15(3):346–367, 2011.

[9] D. Floreano and C. Mattiussi. *Bio-inspired artificial intelligence: theories, methods, and technologies.* The MIT Press, 2008.

[10] J. Gauci and K. Stanley. A case study on the critical role of geometric regularity in machine learning. In *Proc. of the 23rd AAAI Conference on Artificial Intelligence. AAAI Press*, 2008.

[11] J. Gauci and K. Stanley. Autonomous evolution of topographic regularities in artificial neural networks. *Neural Computation*, 2010.

[12] F. Gruau and D. Whitley. Adding learning to the cellular development of neural networks: Evolution and the baldwin effect. *Evolutionary Computation*, 1(3):213–233, 1993.

[13] D. O. Hebb. *The Organization of Behavior: A Neuropsychological Theory.* Wiley, New York, NY, USA, 1949.

[14] E. M. Izhikevich. Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, 15(5):1063–1070, 2004.

[15] G. M. Khan, J. F. Miller, and D. M. Halliday. Evolution of cartesian genetic programs for development of learning neural architecture. *Evolutionary Computation*, 19(3):469–523, 2011.

[16] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of mathematical biology*, 5(4):115–133, 1943.

[17] Y. Niv, D. Joel, I. Meilijson, and E. Ruppin. Evolution of reinforcement learning in uncertain environments: A simple explanation for complex foraging behaviors. *Adaptive Behavior*, 10(1):5–24, 2002.

[18] S. Nolfi and D. Floreano. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines.* MIT press, Cambridge, MA, 2000.

[19] S. Nolfi, D. Parisi, and J. L. Elman. Learning and evolution in neural networks. *Adaptive Behavior*, 3(1):5–28, June 1994.

[20] S. Risi, C. E. Hughes, and K. O. Stanley. Evolving plastic neural networks with novelty search. *Adaptive Behavior*, 18(6):470–491, Dec. 2010.

[21] S. Risi and K. O. Stanley. Indirectly encoding neural plasticity as a pattern of local rules. In *11th International Conference on Simulation of Adaptive Behavior (SAB 2010)*, page 533–543, 2010.

[22] S. Risi and K. O. Stanley. A unified approach to evolving plasticity and neural geometry. In *Proc. of the International Joint Conference on Neural Networks (IJCNN).* IEEE, 2012.

[23] S. Risi, S. D. Vanderbleek, C. E. Hughes, and K. O. Stanley. How novelty search escapes the deceptive trap of learning to learn. In *Proc. of the 11th Annual Genetic and Evolutionary Computation Conference*, page 153–160, 2009.

[24] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards. *Artificial Intelligence: A Modern Approach*, volume 74. Prentice Hall, 1995.

[25] J. Secretan, N. Beato, D. D Ambrosio, A. Rodriguez, A. Campbell, and K. Stanley. Picbreeder: evolving pictures collaboratively online. In *Proc. of the Computer Human Interaction Conference*, page 1759–1768. ACM, 2008.

[26] A. Soltoggio, J. A. Bullinaria, C. Mattiussi, P. Dürr, and D. Floreano. Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios. *Artificial Life*, 11:569, 2008.

[27] A. Soltoggio, P. Dürr, C. Mattiussi, and D. Floreano. Evolving neuromodulatory topologies for reinforcement learning-like problems. In *Proc. of the IEEE Congress on Evolutionary Computation*, page 2471–2478, 2007.

[28] K. O. Stanley. Evolving adaptive neural networks with and without adaptive synapses. In *Proc. of the IEEE Congress on Evolutionary Computation*, volume 4, 2003.

[29] K. O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162, 2007.

[30] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009.

[31] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002.

[32] K. O. Stanley and R. Miikkulainen. Competitive coevolution through evolutionary complexification. *J. Artif. Intell. Res. (JAIR)*, 21:63–100, 2004.

[33] P. Tonelli and J. Mouret. Using a map-based encoding to evolve plastic neural networks. In *2011 IEEE Workshop on Evolving and Adaptive Intelligent Systems (EAIS)*, pages 9–16. IEEE, Apr. 2011.

[34] P. Tonelli and J.-B. Mouret. On the relationships between synaptic plasticity and generative systems. In *Proc. of the 13th annual Genetic and Evolutionary Computation Conference*, page 1531–1538, 2011.

[35] P. Verbancsics and K. O. Stanley. Constraining connectivity to encourage modularity in HyperNEAT. In *Proc. of the 13th annual Genetic and Evolutionary Computation Conference*, page 1483–1490, 2011.

[36] D. Weinreich and W. F. Wonderlin. Inhibition of calcium-dependent spike after-hyperpolarization increases excitability of rabbit visceral sensory neurones. *J. Physiol.*, 394(1):415–427, 1987.