SELF-CONSTRUCTING NEURAL NETWORKS THROUGH RANDOM MUTATION

Samuel Schmidgall
Department of Computer Science
George Mason University
sschmidg@gmu.edu

ABSTRACT

The search for neural architecture is producing many of the most exciting results in artificial intelligence. It has increasingly become apparent that task-specific neural architecture plays a crucial role for effectively solving problems. This paper presents a simple method for learning neural architecture through random mutation. This method demonstrates 1) neural architecture may be learned during the agent’s lifetime, 2) neural architecture may be constructed over a single lifetime without any initial connections or neurons, and 3) architectural modifications enable rapid adaptation to dynamic and novel task scenarios. Starting without any neurons or connections, this method constructs a neural architecture capable of high-performance on several tasks. The lifelong learning capabilities of this method are demonstrated in an environment without episodic resets, even learning with constantly changing morphology, limb disablement, and changing task goals all without losing locomotion capabilities.

1 INTRODUCTION

Biological brains learn as a product of synaptic and structural changes that occur over a single continuous lifetime. While structural developments in the brain are largely tailored by genetic factors, many of the genetic developments are dependent on experience for proper development (Berardi et al. (2015)). Early experimental evidence for this was shown in (Wiesel & Hubel (1963)) where cats were blindfolded in one eye during a critical developmental period. As a result, development of the neurons and neuronal connectivity responsible for visual processing was significantly favored in the unimpaired eye. This type of structural plasticity is not limited to critical developmental periods, and generally occurs throughout the lifetime of mature organisms (La Rosa et al. (2020), Holtmaat & Svoboda (2009), Johansen-Berg (2007)).

In nature, animals must learn without episodic boundaries or "environment resets," which is referred to as lifelong learning. When an animal makes a mistake it must face the consequences and be capable of learning additional skills to correct itself. This differs from the typical Reinforcement Learning (RL) setting, where the agent’s lifetime is composed of the same task repeated over a small time interval. Often in RL, if a mistake is made, the simulation just starts over again, sidestepping the need to learn a more general set of skills. Part of the difficulty with lifelong learning settings is that many learning algorithms experience a phenomenon known as catastrophic forgetting, which causes previously learned capabilities to degrade with the incorporation of new knowledge. Many successful attempts to overcome catastrophic forgetting involve localizing learning to a subset of the existing neural connections (Kirkpatrick et al. (2016), Masse et al. (2018)). However, this often requires knowledge of a discrete set of different tasks, as well as a centralized mechanism that switches between these regions in the right context. Additionally, it requires pre-defining the number of connections, hence the amount of learning that is expected to take place must be known prior to training. More often than not interesting learning tasks are not so simple.

In order to learn without episodic boundaries while constantly constructing new neurons and neural connections, a simple method for actively modifying neural architecture is proposed. This is accomplished by randomly modifying the neural architecture and incorporating changes based on a dynamic acceptance threshold. This method is demonstrated to be capable of lifelong learning in a
constantly changing environment, constructing networks capable of learning with only a small set of weight values, and performing competitively with traditional learning methods despite its simplicity.

2 RELATED WORK

While active self-construction itself is a less explored topic, the development of neural architecture emerged from a long history in evolutionary computation (Stanley & Miikkulainen (2002), Risi & Stanley (2011), [Stanley et al. (2009)], and has had success more recently with random search (Li & Talwalkar (2019), Sciuto et al. (2019)) and gradient-based methods (Zoph & Le (2017), Ha et al. (2016), [Li et al. (2020)). The goal of Neural Architecture Search (NAS) is to automatically find an architecture that, once trained, outperforms hand-designed architectures (Elsken et al. (2019)). As a consequence, many of these methods involve an expensive inner and outer loop search of optimizing architecture and weights. A classic example of architecture search which optimizes both weights and topology simultaneously is NeuroEvolution of Augmenting Topologies (NEAT) (Stanley & Miikkulainen (2002)). NEAT begins with a population of minimal neural network topologies and increasingly leads to a more complex and diverse population. These changes are made through random mutation on the evolutionary scale across generations.

A related strategy for NAS, network pruning, strategically removes connections from densely connected neural networks once their weight value shrinks below a certain value. This method has been shown to produce networks that contain only a small fraction of their original number of connections, while still retaining competitive performance with its fully-connected counterpart on the task they were trained for (Kusupati et al. (2020), Dong & Yang (2019)). Some results even indicate removing redundant connections may improve performance (Savarese et al. (2019), Frankle & Carbin (2018)). Recent work considered the capabilities of networks that were only pruned, using randomly initialized weights which were pruned down without weight optimization to produce networks that had far better than chance performance on several image classification tasks (Zhou et al. (2019)).

The importance of architecture itself independent of weight values was first explored in (Gaier & Ha (2019)). Here, a population of neural architectures are evolved using NEAT; however, to evaluate the population independent of weights, the performance of an architecture is collected using random weights. Since sufficient performance could not be attained using completely random weights, the performance of these networks are evaluated over a single weight which is shared among every connection. The overall performance of an architecture is computed by averaging performance over several of these shared weights taken from a discrete set. In this work, weight-agnostic neural networks were shown to provide far better than chance performance on several RL tasks indicating that neural architecture itself is may provide a solution to the learning problem.

3 RANDOM MUTATION SEARCH

Random Mutation Search (RMS) may be summarized in a few simple steps: (1) Initialize and evaluate a network of input and output neurons with no connections and set as current best, (2) randomly mutate the current best network structure, (3) evaluate network performance over a specified time interval, (4) if the observed performance is better than the previous best network (mutation acceptance threshold) then update best network and performance, (5) repeat algorithm starting at step 2.

The mutation operator has four mutation types: add connection, add neuron, change neuron activation, and remove connection (Figure 1). These operations provide the capability for both neural pruning and topology augmentation. The network is represented as a directed graph structure. This graph structure supports the development of feedforward, feedback, and self-recurrent connections. While feedforward and self-recurrent weights are commonly incorporated in neural networks, feedback weights are less utilized largely because it is unknown how and when to introduce these weights manually (Herzog et al. (2020)).

Add Connection: The add connection operator randomly selects two neurons and adds a connection between them. The connection’s weight is then assigned to a random value. This operation allows for any type of connection including: feedforward, feedback, lateral, and self-recurrent. Feedback weights connect one neuron with another whose activation occurs before it. Similar to self-recurrence, feedback connections receive a delayed signal from the activated neuron. Lateral
connections occur between two neurons in the same layer. While the "hidden" network structure does not have a strict definition, lateral connections are more straightforwardly seen as connecting two output or input neurons and are treated in the same way as feedback connections with a delayed signal.

**Add Neuron:** To add a neuron, a connection is selected at random, and split into two connections with a new neuron inbetween. As in (Stanley & Miikkulainen (2002)), the weight of the new neuron's incoming connection is set to 1, and the weight of the connection that was split is inherited by the new neuron's outgoing connection. The new neuron's activation function is then randomly selected.

**Change Activation:** The activation change operator is the most straightforward. A neuron is selected at random and its activation is randomly sampled from a list of defined activations. In this work a list of common (e.g. ReLU, hyperbolic tangent, sigmoid) and uncommon activation functions (e.g. Gaussian, sinusoid, step) following from (Gaier & Ha (2019)) are selected.

**Remove Connection:** Like the add connection operator, a connection is selected at random; however, actually removing the connection comes with additional difficulties. Once the connection is removed, the receiving neuron may end up without incoming connections. Additionally, the sending neuron may no longer have any outgoing connections. In either of these cases, a neuron deletion strategy must be employed. There are four primary strategies for deciding when to perform neuron deletion: when all incoming connections have been removed, when all outgoing connections have been removed, when both sets have, or when either set has been removed. In all but the final scenario, neurons and connections may exist in the networks without actually contributing to neural activity. However, these three methods preserve neural connections that were at one point useful which may be reused at a later time.

### 4 Experiments

The experimental section evaluates two distinct learning settings to provide an intuition for the expected performance, capabilities, and behavior of these self-constructing networks. In the performance comparison section, RMS is compared with Evolutionary Strategies (Salimans et al. (2017)), NEAT (Stanley & Miikkulainen (2002)), and the on-policy gradient method Proximal Policy Optimization (Schulman et al. (2017)) on several RL tasks in the traditional episodic setting. The lifelong learning section applies RMS in an environment without episodic resets all while constantly experiencing changes in morphology, limb impairment, and task goal.

For all of the experiments in this paper, neuron deletion occurs when both sets of incoming and outgoing connections have been removed. Upon creation, connection weights are randomly assigned to values in the set \( \{ \pm 1, \pm 0.5, \pm 0.25 \} \), which is referred to as the discrete set. In addition to the original four operators, a fifth is added, *change weight*, which randomly assigns a connection weight to a value sampled from the discrete set. In more practical settings this operation would be replaced...
by weight optimization techniques such as reinforcement learning, local plasticity rules, etc. To account for performance evaluation variance, as well as the changing environmental dynamics observed during lifetime learning problems, the current best network performance discussed in (4) is decayed exponentially over time. Additionally, in the episodic setting, RMS is penalized for having high variance among collected episodes; hence, the mutation acceptance threshold increases for that individual mutation to promote consistent behavior.

4.1 Comparison to other methods

The performance of three alternate learning paradigms are compared with RMS in the traditional episodic learning setting. The purpose of this section is provide a frame of reference for the learning capabilities of RMS. The first method, NEAT, provides the most straightforward comparison since the neural architecture does not need to be selected prior to training; it is constructed from scratch. The Evolutionary Strategies (ES) optimization, like NEAT, is a population based optimization method. However, unlike NEAT, ES does not construct its own topology; rather it is a numerical optimizer which follows from (Salimans et al. (2017)). As a result, the network topology must be hand-designed before training time. The final method uses stochastic gradient ascent via the on-policy RL algorithm Proximal Policy Optimization (PPO) (Schulman et al. (2017)). Like ES, this method is not self-constructing. The hyperparameters for each of these methods are described in the Appendix.

| Optimization Method          | Ant Locomotion | Swimmer | Swing-up |
|------------------------------|----------------|---------|----------|
| Random Mutation Search       | **1076 ± 106** | 1106 ± 39 | 326 ± 35 |
| NEAT                         | 788 ± 55       | 1177 ± 59 | 363 ± 14 |
| Evolutionary Strategies      | 737 ± 113      | **1318 ± 12** | 218 ± 3  |
| Proximal Policy Optimization | 962 ± 17       | 1286 ± 31 | **379 ± 12** |

Table 1: Performance of Random Mutation Search compared with three other algorithms on robotic learning benchmarks. Descriptions of these tasks are provided in the Appendix.

The performance results of each learning strategy is described in (Table 1). Despite having only a discrete set of possible weight values, and making changes based on randomly selected mutations, RMS is capable of performing competitively with ES, NEAT, and PPO.

For the Ant Locomotion task, RMS developed a total average of 283 connections (Table 1) which was 95% less than the 6336 connections that were used in PPO and ES, and 28% less than NEAT’s 393 connections. The ratio of feedback, feedforward, and self-recurrent weights were relatively consistent across tasks even as the number of connections increased. Self-recurrent connections did not develop frequently since sampling the same neuron in the add connection operation became less likely as more neurons were added. Comparing the difficulty of a task is challenging across different learning settings (e.g. Ant Locomotion, Swimmer, Swing-up), hence making it difficult to know the effect of task complexity on learned architecture. To appropriately assess this effect, two additional Ant Locomotion settings are considered of varying complexity. The limb impairment setting has the same goal as Ant Locomotion; however, a limb is randomly selected at the beginning of each episode, and all action output to that limb is rendered void (Figure 2 (A)). The dynamic morphology setting randomizes limb length, weight, and width (Figure 2 (B)). The results of training Ant Locomotion in these two settings, as well as in the non-impaired setting, indicate an increase in the number of connections across tasks based on task difficulty. The limb impaired architecture had 7% more connections than the non-impaired, and the dynamic morphology had 27% more. As a product of RMS co-developing weights and architecture without populations, it only required 10 million environment interactions compared to NEAT’s 150 million.

1The code is available at https://github.com/SamuelSchmidgall/RandomMutationSearch
Table 2: Average composition of learned neural architectures by connection type on Ant locomotion task over 10 million environment interactions in increasing order of connections.

| Quadruped Task          | Connections | Feedforward | Feedback | Self-Recurrence |
|-------------------------|-------------|-------------|----------|-----------------|
| Locomotion Only         | 283         | 94%         | 6%       | 0%              |
| Limb Impairment         | 302         | 92%         | 8%       | 0%              |
| Dynamic Morphology      | 362         | 92%         | 7%       | 1%              |
| Lifelong                | 483         | 93%         | 7%       | 0%              |

4.2 **Lifelong Learning**

In the lifelong learning scenario, a variation of Ant locomotion is introduced in a setting where episodic boundaries are dissolved. As a consequence, the learning problem changes fundamentally. In the episodic setting of this task, when the quadruped falls over, becomes disoriented, or positions its limbs poorly there are few consequences, as the episodic nature of the task simply restarts the quadruped’s position. This results in minimal incentive for learning a more general task solving skillset. Additionally, these tasks are simulated over a very short time-period which is comparatively only a small fraction of the lifespan of even the simplest creatures on earth. There is a significant difference in the way biological organisms must learn compared to traditional simulated machines.

![Variations of the original Ant locomotion task encountered in the lifelong learning setting.](image1)

(A) Impaired Limb (B) Randomized Morphology

Figure 2: Variations of the original Ant locomotion task encountered in the lifelong learning setting. (A) A random limb is chosen to be impaired (*in red*) rendering all action output to that limb void. (B) In the dynamic morphology task variant, morphological changes are randomly applied causing differences in limb length, width, and weight.

An important component of lifelong learning is that the task itself requires adaptation to new scenarios from which general knowledge can be developed. Here, this is introduced into the locomotion task in three ways: 1) the quadruped’s body is constantly changing (Figure 2B), 2) the quadruped’s limbs are constantly being impaired and unimpaired (Figure 2A), 3) the target direction for locomotion is constantly changing at random intervals all without environment resets. Importantly, the morphological changes are not incorporated into the state information, hence the self-constructing network must learn based on implicit context signals which limb is impaired and how the body was changed as well as produce a more general pattern of locomotion to account for these changes.

The lifelong learning networks were trained over 10 million environment interactions, just as the episodic networks were in (Section 4.1). However, self-construction led to the development of a network that was 71% larger than in the episodic setting, once again indicating there is an increase in network size based on task complexity.

The performance metric for this task is different than the episodic Ant Locomotion task to account for direction heading; however, to evaluate the efficacy and generalization of the lifelong learning neural architecture, we apply it to the Ant Locomotion task without dynamic morphology. Overall, there was little difference between the performance of the network trained in the episodic setting and the lifelong learning setting despite having been trained for different tasks; the lifelong learning network obtained a mean score of 856 compared to the episodic 1076. The similarity is non-trivial.
since generalization, in this case over morphology, impairment, and direction, is often accompanied by a decrease in specialization. There was only a slight decrease in performance for three of the four limbs that were disabled individually: the two front limbs obtained scores of 822, 789 and the back left limb, 658. Interestingly, with the back right limb disabled the policy obtained a low performance average of 22. This is because the self-construction mechanism generated a more greedy solution\(^2\) to provide limb robustness by, instead of relying on all four limbs equally, putting all of the locomotion capabilities into a single limb so that there was a lower probability of performance impairment. This is likely a result of selecting architecture based purely on maximum performance in a setting where the limb impairment is randomly sampled; episodes where the dominant limb is not disabled are likely to be preferred over those where it is.

There was little effect on morphological randomization and performance, obtaining an average of 844. Since the lifelong neural architecture provides similar performance despite the necessity of generalization, perhaps an enlargement of neural structure may provide both general and specialized learning capabilities.

5 DISCUSSION

Despite RMS proving successful on the task set presented in this work, environments with large observation or action spaces may likely prove challenging for RMS to effectively generate meaningful connections using only random mutations. This was a central concern around NEAT strategies. A potential alleviation of this challenge may come from using RMS to develop a network that serves to generate a separate network which acts as the environment policy, as in \cite{Stanley2009,Ha2016}. Additional performance improvements may come from combining weight-optimization techniques together with RMS, and could serve as a simple mechanism for both architectural and weight optimization that is capable of developing in lifelong learning settings.

This work introduces a simple method for learning neural architecture through random mutation. As a product of its simplicity, this method is capable of learning architecture during the agent’s lifetime. This innovation was shown to provide competitive performance with several standard learning paradigms, as well as being applied to a lifelong learning setting where episodic boundaries are dissolved and dynamic changes in morphology and task setting are constantly being experienced. Despite this, single weight optimizations were a product of random mutations sampled from a small discrete set of possible values. The learned architectures were marginally smaller than those developed by NEAT, and significantly smaller than the standard static architectures used in PPO and ES. In the lifelong learning setting, as well as the limb impairment and dynamic morphology settings, the number of neural connections were observed to increase as a product of task complexity.

REFERENCES

Nicoletta Berardi, Alessandro Sale, and Lamberto Maffei. Brain structural and functional development: genetics and experience. *Developmental Medicine & Child Neurology*, 57:4–9, 2015.

Rémi Coulom. *Reinforcement learning using neural networks, with applications to motor control*. PhD thesis, Institut National Polytechnique de Grenoble-INPG, 2002.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. [https://github.com/openai/baselines](https://github.com/openai/baselines) 2017.

Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search, 2019.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey, 2019.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

\(^2\)This strategy was visually reminiscent of cellular movement using a flagella, with the remaining limbs providing stability in the front.
Adam Gaier and David Ha. Weight agnostic neural networks. 2019. URL https://weightagnostic.github.io

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. CoRR, abs/1609.09106, 2016. URL http://arxiv.org/abs/1609.09106

Sebastian Herzog, Christian Tetzlaff, and Florentin Wörgötter. Evolving artificial neural networks with feedback. Neural Networks, 123:153–162, 2020. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2019.12.004. URL https://www.sciencedirect.com/science/article/pii/S089360801930396X

Anthony Holtmaat and Karel Svoboda. Experience-dependent structural synaptic plasticity in the mammalian brain. Nature Reviews Neuroscience, 10(9):647–658, Sep 2009. ISSN 1471-0048. doi: 10.1038/nrn2699. URL https://doi.org/10.1038/nrn2699

Heidi Johansen-Berg. Structural plasticity: Rewiring the brain. Current Biology, 17(4):R141–R144, 2007. ISSN 0960-9822. doi: https://doi.org/10.1016/j.cub.2006.12.022. URL https://www.sciencedirect.com/science/article/pii/S0960982206026200

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. CoRR, abs/1612.00796, 2016. URL http://arxiv.org/abs/1612.00796

Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity, 2020.

Chiara La Rosa, Roberta Parolisi, and Luca Bonfanti. Brain structural plasticity: From adult neurogenesis to immature neurons. Frontiers in Neuroscience, 14:75, 2020. ISSN 1662-453X. doi: 10.3389/fnins.2020.00075. URL https://www.frontiersin.org/article/10.3389/fnins.2020.00075

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. CoRR, abs/1902.07638, 2019. URL http://arxiv.org/abs/1902.07638

Liam Li, Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Geometry-aware gradient algorithms for neural architecture search, 2020.

Nicolas Y Masse, Gregory D Grant, and David J Freedman. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. Proceedings of the National Academy of Sciences, 115(44):E10467–E10475, 2018.

Sebastian Risi and Kenneth O Stanley. Enhancing es-hyperneat to evolve more complex regular neural networks. In Proceedings of the 13th annual conference on Genetic and evolutionary computation, pp. 1539–1546, 2011.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017.

Pedro Savarese, Hugo Silva, and Michael Maire. Winning the lottery with continuous sparsification. CoRR, abs/1912.04427, 2019. URL http://arxiv.org/abs/1912.04427

Samuel Schmidgall. Random mutation search. https://github.com/SamuelSchmidgall/RandomMutationSearch, 2021.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438, 2015.
A Training Details

A.1 Hyperparameters

This section provides an in-depth set of hyperparameter tables for reproducibility, which is also available in the code (Schmidgall (2021)). The code to train PPO in the performance comparison section was taken from OpenAI Baselines (Dhariwal et al. (2017)).

| RMS Hyperparameters                  |     |
|--------------------------------------|-----|
| Mutation per Update (Ant, Swimmer, Lifelong) | U(1, 20) |
| Mutation per Update (Swingup)        | 1   |
| Connection Penalty                   | 0.001 |
| Top Performance Decay Rate           | 0.995 |
| Initial Random Mutations             | 50  |
| Weight Type                          | \{±1, ±0.5, ±0.25\} |
| Neuron-deletion Strategy             | Both-sets |
| Remove Connection Probability        | 0.2  |
| Add Connection Probability           | 0.2  |
| Change Activation Probability        | 0.2  |
| Add Neuron Probability               | 0.2  |
| Change Weight Probability            | 0.2  |
| Performance Variance Penalty Coeff.  | 0.1  |
| Update Interval                      | 4000 |
Here are the ES Hyperparameters:

| Parameter                  | Value |
|----------------------------|-------|
| Population Size            | 128   |
| Learning Rate              | 0.04  |
| Learning Rate Decay        | 0.999 |
| Parameter Sampling Type    | Normal|
| Parameter Std              | 0.04  |
| Parameter Std Decay        | 0.999 |
| Weight Penalty             | 0.01  |
| Hidden Layer Width         | 64    |
| Hidden Layers              | 2     |
| Hidden Neuron Activation   | tanh  |
| Action Neuron Activation   | tanh  |
| Action Noise Std           | 0.01  |
| Update Iterations          | 512   |

Here are the NEAT Hyperparameters:

| Parameter                               | Value |
|-----------------------------------------|-------|
| Population Size                         | 192   |
| Parameter Sampling Type                 | Normal|
| Parameter Standard Deviation            | 0.005 |
| Optimization Type                       | Elite Selection |
| Total Elites                            | 24    |
| Unchanged Elites                        | 2     |
| Mutations Per Update                    | 1     |
| Add Neuron Probability                  | 0.2   |
| Add Connection Probability              | 0.4   |
| Change Activation Probability           | 0.4   |
| Performance Episode Average             | 3     |
| Update Iterations                       | 512   |

Here are the PPO Hyperparameters:

| Parameter                               | Value |
|-----------------------------------------|-------|
| PPO Gamma (\(\gamma\))                 | 0.99  |
| PPO Lambda (\(\lambda\))               | 0.95  |
| Clip Parameter                          | 0.2   |
| Optimizer Step Size                     | 3e-4  |
| Actor Critic Batch Size                 | 2048  |
| Optimizer Batch Size                    | 64    |
| Batch Epochs                            | 10    |
| Hidden Layer Width                      | 64    |
| Hidden Layers                           | 2     |
| Hidden Neuron Activation                | tanh  |
| Action Neuron Activation                | linear|
| Entropy Coeff.                          | 0.0   |
| Total Interactions                      | 1 Million |
| Optimizer                               | Adam (Kingma & Ba (2014)) |

**A.2 Environment Descriptions**

Here are descriptions provided for each environment used in this work. We note that these environments are also available in the paper code repository ([Schmidgall (2021)]).

**Ant Locomotion:**
The Ant Locomotion task inherits the robotic body of the Ant task which originally appeared in
The momentary task reward was modified to provide a positive reward signal proportional to velocity in any direction. This was to allow for an adequate comparison with the lifelong learning setting which restricted movement along the x and y plane, requiring a more general policy.

**Swimmer:**
The Swimmer task was originally used in (Coulom (2002)). This robot is a three-linked 'swimmer' which attempts to attain movement across a plane by extending and retracting its links, much like a snake or worm despite the deceptive environment name. In this task, much like the Ant Locomotion task, reward is based proportionally to velocity along the plane in any direction.

**CartPole-Swingup:** The CartPole-Swingup task requires swinging up and balancing a pole on a cart. Reward is proportional to both the cart’s distance from the center as well as the pole’s distance from being upright. This task requires much more fidelity and consistency than do the locomotion tasks.