2020

Co-optimization of a Robot's Body and Brain via Evolution and Reinforcement Learning

Jack Felag
University of Vermont

Co-optimization of a Robot's Body and Brain via Evolution and Reinforcement Learning

A Thesis Presented

by

Jack Michael Felag

to

The Faculty of the Graduate College

of

The University of Vermont

In Partial Fulfillment of the Requirements
for the Degree of Master of Science
Specializing in Complex Systems & Data Science

May, 2020

Defense Date: March 16th, 2020
Dissertation Examination Committee:

Joshua Bongard, Ph.D., Advisor
Nicholas Allgaier, Ph.D., Chairperson
Nicholas Cheney, Ph.D.
Safwan Wshah, Ph.D.
Cynthia J. Forehand, Ph.D., Dean of Graduate College

Abstract

Agents are often trained to perform a task via optimization algorithms. One class of algorithms used is evolution, which is "survival of the fitness" used to pick the best agents for the objective, and slowly changing the best over time to find a good solution. Evolution, or evolutionary algorithms, have been commonly used to automatically select for a better body of the agent, which can outperform hand-designed models. Another class of algorithms used is reinforcement learning. Through this strategy, agents learn from prior experiences in order to maximize some reward. Generally, this reward is how close the objective is to being complete, or otherwise some stepping stone towards its completion. Evolution and reinforcement learning can both train agents to the point where a task can be successfully completed, or an objective met. In this thesis, we outline a framework for combining evolution and reinforcement learning, and outline experimental designs to test this method against traditional reinforcement learning. Finally, we show preliminary results as a proof of concept for the methods described.

Acknowledgements

First off, I would like to thank my advisors: Josh Bongard and Nick Cheney. Josh initially got me into evolutionary robotics during my undergrad at UVM, and has been advising me in research for nearly four full school years so far. Around that beginning time, I met Nick while he was a visiting PhD student in the lab, and he was a fountain of knowledge both then and now. I am extremely appreciative of the time and advice the both of them have given me. I would also like to thank many members of the Complex Systems Center, including:

-Sam Kriegman, for always ~~arguing over~~ discussing details and ideas,
-Colin Van Oort, Sida Liu, and David Matthews, for their help in understanding the inner-workings of Tensorflow and the VACC, and
-Joshua Powers, for answering questions about RL and Pybullet when the documentation was scarce.

I am also grateful for the VACC for providing computational resources for the work done here.

Finally, I would like to thank my thesis committee, Nick Allgaier and Safwan Wshah, for their time and effort in reviewing this thesis, and in all other parts of this process.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Traditionally in the field of reinforcement learning, a controller is trained for a hand-designed body. An exception to this is the work of Ha, where body parts of the agent were parameterized and those parameters were learned in addition to the controller [3] (and later, the work of Luck et al [5]). This method beat out only optimizing the controller because the agent "learned" a better body for the task. Updating the body, or morphology, of an agent has also been explored extensively in the field of evolutionary computation. The body, while used as a vector for computation, also affects how an agent learns [12, 13]. The traditional way of optimizing a controller for a fixed body is outdated, and methods to co-optimize the body and brain of an agent should be explored further. Therefore in this thesis, we extend the work of Ha to allow for body parts to be added and removed from the morphology of an agent.

1.1 Morphology

Morphology is the body of the agent. When considering an agent like a robot, morphology has four distinct, yet interconnected, parts: body segments, joints, motors, and sensors. The body segments are what the physical realization of the agent looks like. As an example, humans have a head, torso, arms, legs, etc. Joints dictate what body segments are attached, how they are allowed to move relative to one another, and their range of motion. Motors provide movement to the joints, and sensors allow the agent to perceive both itself and its environment.

1.2 Control

Control relates to how the agent moves in its environment. Examples of control can be simple rules, functions given to each motor, or a neural network. For the work in this thesis, a neural network controller was used.

1.2.1 Neural Networks

Artificial neural networks (ANNs), can be represented by neurons and synapses. Neurons hold certain values (they can be inputs, outputs, or used for processing (called "hidden"), and synapses dictate how the neurons are connected, and how strongly a signal will propagate across it. This strength is called a weight, with negative values being called inhibitory, positive are excitatory, and zero represents a nonexistent synapse. In Figure 1.1, an example ANN is shown: neurons are represented as circles, and synapses as lines. When working with an agent, the inputs are generally infor-

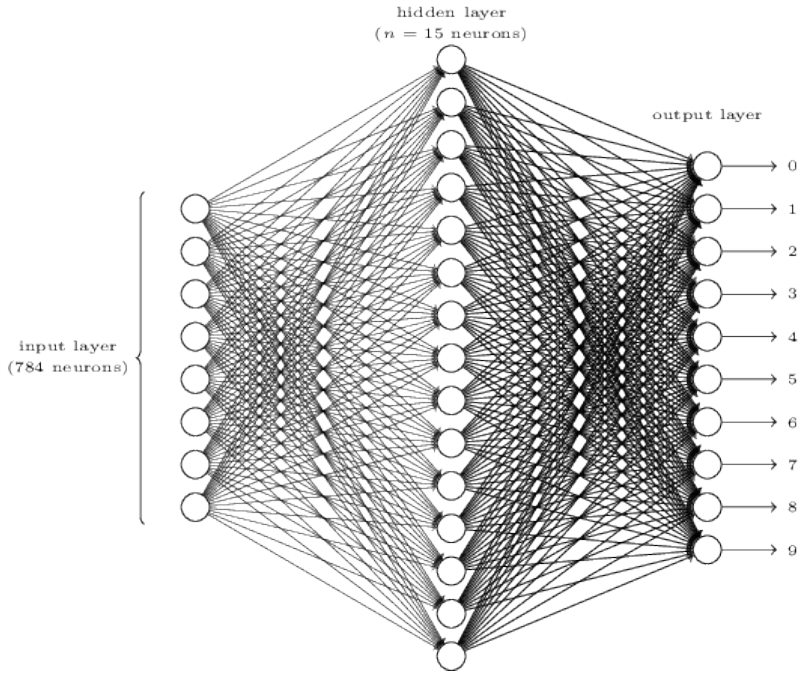

Figure 1.1: A neural network with three layers: one input, one hidden, one output [14].

mation from its sensors, and its outputs control its motors. ANNs can be simplified greatly with matrix multiplication. The simplest form of which is:

$$\mathbf{m} = \sigma(\mathbf{W}\mathbf{s} + \mathbf{b}) \tag{1.1}$$

Where $\mathbf{m}$ is the vector containing the values of the agent's motor control, $\mathbf{s}$ is the vector containing the agent's sensory information, $\mathbf{W}$ is a matrix holding all the synapse weights, and $\mathbf{b}$ is a bias vector. The function, σ, is some non-linear function, usually hyperbolic tangent ($\sigma = \tanh$). This non-linearity is necessary to add complexity since matrix multiplication only allows for linear transformations of variables.

1.3 Heuristic Search Techniques

Heuristic search is a term that describes a general class of algorithms used to solve an optimization problem. The basic principle around these algorithms is using stochastic methods or a heuristic to solve an otherwise intractable problem. For some optimization problems, polynomial-time algorithms exist that result in the optimal solution in a relatively short amount of time. However, these nice solutions do not exist for all problems, so certain random steps or approximations need to be made in order to find a solution that works good enough. This thesis employs two such techniques, reinforcement learning, and evolutionary algorithms, which both in turn describe a general class of algorithms.

1.3.1 Reinforcement Learning

Reinforcement learning (RL) is a way for an agent to learn from its past experiences. As an example, consider training a dog. When a dog is first learning how to sit, a human will tell the command to the dog, and might provide some kind of physical feedback (hand pointing down or gently pressing on the dog's back). If the dog successfully sits down, it gets a treat! If not, the process is repeated. An agent will learn in a similar way, using observations (in the example, voice and hand signals), actions (trying to sit down), and rewards (treats).

Observations are generally set up in what's called an observation space. This contains the set of allowed perceptions from both itself and its environment. While dogs have plenty to observe (sight, smell, sound, temperature, feel of the ground, etc), we can

define an observation very precisely in simulation. For an agent, observations can be limited to just the sensors it's been given. This allows for a simpler problem (reducing the size of the input layer in the ANN).

Similar to observations, actions are set up in action spaces. These refer to the set of all possible actions an agent can take. Also like observations, we can limit these to simplify the problem. Generally, actions will be limited to control of the agent's motors. This has the same effect as making the output layer smaller (in the dog example, barking isn't necessarily related to sitting down).

Finally, there are rewards. These tell the agent how well it's doing for the given task. Rewards can take the form of a human-in-the-loop (manual feedback), or automated via a reward function. A dog relies on a human-in-the-loop method: the human verifies sitting or not, and rewards accordingly. However, automating the method is much faster and thus convenient (assuming there exists a simple function to determine how the agent is performing).

With these three pieces, RL comes together in a loop of observation-action-reward. Initially, the agent will observe and perform an action (feeding the observation through its ANN). Based on the ANN's synaptic weights, an action will be determined and the agent will go through with that. After the action, a reward will be assigned by some method, and the agent will "learn" accordingly. Learning in this case is updating the synaptic weights of the ANN in order to receive a higher reward. The agent is then reset and iterated through this process again and again, with each iteration as one

"episode". The general control here is referred to as a policy, where a policy attempts to maximize a reward by performing an action based on the current observation.

1.3.2 Evolutionary Algorithms

Evolutionary algorithms take their main inspiration from biological evolution. An individual that is highly fit is able to produce more offspring than others. While simulating evolution, fitness can mainly comes down to one purpose: selecting an agent that can perform a task well. To do this, a population of agents is initialized, then three phases are repeated: evaluation, selection, and mutation. One iteration of these three phases make up a single generation in the algorithm.

Evaluation is the phase where agents receive a fitness score. "Fitness" itself can vary from simple metrics, percent completion of a task, novelty [8], or some combination of metrics [9]. Selection is where "survival of the fittest" comes in. The mechanism will keep some agents, and remove the others, based on fitness and possibly other metrics like age [10]. The remaining agents are now the parents that will produce mutant offspring to fill the population back up. These mutants can be slight modifications to copies of the parents, or a combination of two parents in an operation known as crossover [11]. When the population is filled, the next generation then begins and the cycle repeats until the amount of generations specified is complete. Over evolutionary time, the small changes in agents along with selecting for fit ones results in an upward trend of fitness.

Chapter 2

Methods and Results

This chapter presents the results from each experiment and the methods used to produce the experiments. All code was written in `Python 3.x` and is available on GitHub.

2.1 Framework

2.1.1 Components

As the setup to the experiments in this thesis, a framework was created to integrate evolution and RL together. The framework is made up of three main parts: a physics simulator, an environment, and a population.

For the physics simulator, we use `PyBullet` [6]. This simulator is used mainly for its ability easily add reinforcement learning to agents within it. An agent for this simulator is described by an XML file, which has tags for body segments, joints, motors, etc.

The environment is used to set up observations, actions, and rewards for RL. Here, we use `Open AI Gym` [7]. The Gym environment was used mainly for its ease of integrating RL with a physics simulation.

Finally, the population is a data structure that contains the set of agents that need to be evaluated. Each agent has a few parts themselves: a filename pointing towards the XML file that defines its morphology, a filename that points to its controller, an editor to make changes to the XML file, along with information about the agent like fitness, age, etc. The editor can change morphology by automatically adding or removing tags that define body segments, then adding or removing associated joints and motors. This comes down to parsing the XML file, which represents it as a tree: tags are nodes and child tags are child nodes. This makes changing the XML file as simple as adding or removing nodes from a tree.

2.1.2 Training

To train agents in this framework, the population goes through the three main stages of an evolutionary algorithm. Selection will choose agents with a high fitness to produce mutant offspring. These will have either one additional body part, or one body part removed (with any "downstream" limbs removed as well to prevent disjoint bodies). The important difference in our method is a loop of reinforcement learning that occurs during the evaluation phase. Instead of the agent simply being dropped in the simulator once, it is repeatedly simulated while RL updates the agent's policy (ANN controller) attempting to maximize some reward. The fitness value that comes

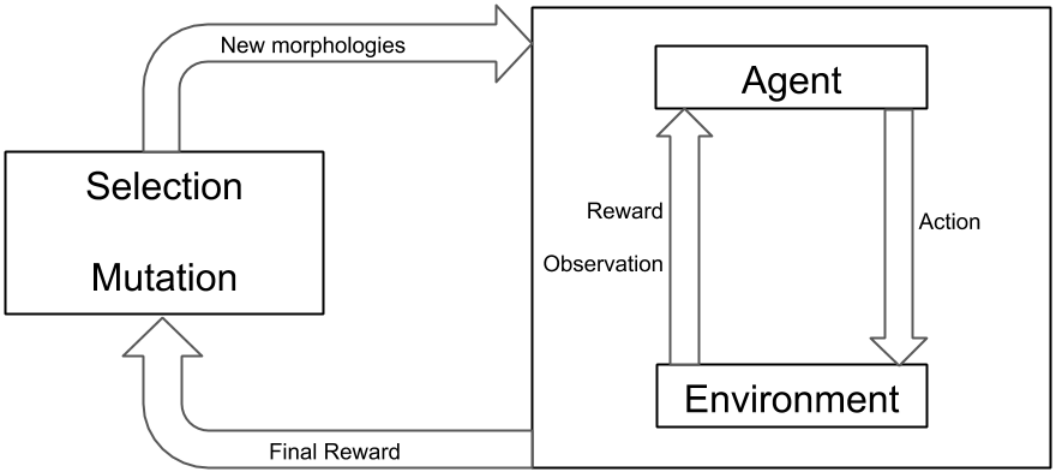

Figure 2.1: Flow of individuals in ERL.

out of this evaluation is the final x-position of the agent during its last episode for that evaluation. We then have two repeated processes here: and "inner loop" of RL optimizing the controller for the fixed morphology of the agent, and an "outer loop" of evolution selecting for morphologies that facilitate learning. Figure 2.1 visually shows this feedback.

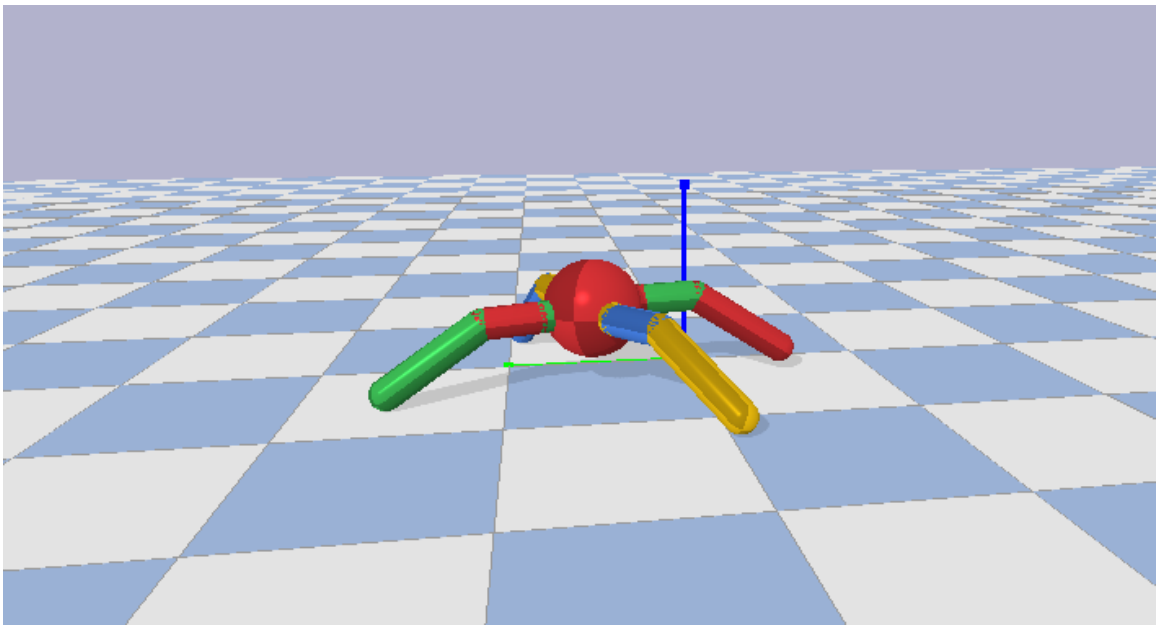

Figure 2.2: `AntWalker` *morphology.*

2.2 Experiments

2.2.1 Combining Reinforcement Learning and Evolutionary Algorithms

Parameters and Initialization

The main experiment is combining PPO and morphological innovation protection. This experiment will be referred to as *ERL*. For the outer loop of evolution, a population of 10 individuals were initialized. These all started as the baseline `AntWalker` morphology (Figure 2.2), but each with an independently random neural network controller. This baseline morphology has 13 body parts: a spherical torso, and four legs each containing three segments. The segment attaching each leg to the torso is

fixed, so only the two most distal segments are attached by motorized joints.

Evaluation

Evaluation is made up of 10 reinforcement learning episodes, which each lasted 3000 time steps in the physics simulator. The agents were being evaluated for movement in the positive x-direction in the simulation. Each observation was:

$$O_t = [\vec{p}(j)_t,\ \vec{v}(j)_t,\ A_{t-1}] \tag{2.1}$$

Where t is the current time step, $\vec{p}(j)_t$ is the vector containing all joint positions at the current time, and $\vec{v}(j)_t$ are the joint velocities. and A_{t-1} is the action of the previous timestep. The action is defined as the target velocity assigned to each motor, and note that $\vec{v}(j)_t \neq A_{t-1}$, as each joint may not reach its target from the previous action. Finally, we have the reward function, which is defined as:

$$r_t = v_t - 10^{-6} \cdot ||\mathbf{u}_t||^2 + p_z \tag{2.2}$$

Where v_t is the velocity of the agent along the x-axis in simulation, $\mathbf{u}_t$ is the vector containing the motor torques applied to all the joints at time t, and p_z is the height of the center of the agent's torso (spherical body). The position was added to promote standing of the agents.

The RL algorithm used was PPO [4]. This algorithm was selected since it has both good performance and high popularity.

Finally, the fitness value assigned to the agent was the agent's final x position.

Selection

Selection occurred by selecting agents that had a high fitness and low age [2]. To do this, an agent would be removed (dominated) if another agent had both a higher fitness and a lower age. If only one of these criteria were met, the agent would be kept. After all dominated agents were removed, the remaining agents had their ages all incremented by one. Agents that stayed in the population also kept their controller, allowing them to further optimize it during the next evaluation. Balancing the age and fitness of each agent allowed for newer morphologies to get an opportunity to train their controllers.

Mutation

To fill the population, a one of the agents still in the population were selected completely at random to be the parent. A mutant was made by copying the parent, and either removing or adding a body segment both with 0.5 probability. These mutants were considered as random agents, so their age would be set to 0 and their controller would be randomized. All of this was repeated for 50 generations.

2.2.2 Control

Reinforcement Learning Only

The setup for this control is the same as *ERL*: 10 individuals were initialized as just the baseline morphology with random neural networks. However, instead of

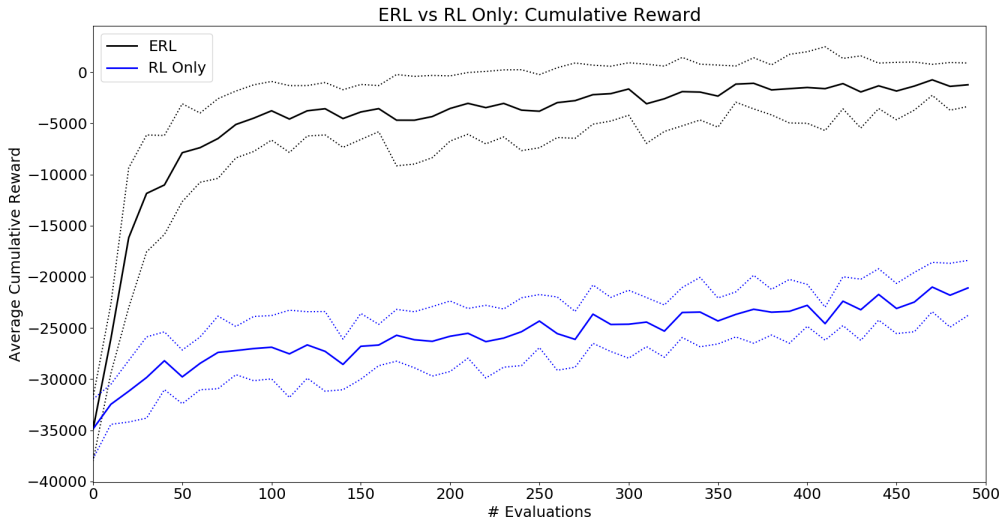

Figure 2.3: Comparing ERL to RL only.

going through 100 generations each with 10 reinforcement learning episodes, these individuals went under $50 \cdot 10 = 500$ reinforcement learning episodes on just the baseline morphology. Note that each robot went under the same number of total evaluations in this experiment compared to the main method. This control experiment was selected based on its ability to compare the main method's combined technique with purely optimizing the controller of a known good morphology.

2.3 Results

With the experiments run, there is currently some evidence to support this experimental design. As shown in Figure 2.3, the main method outlined here outperformed RL only on the starting baseline morphology. Each point in one of the curves repre-

Method	Fitness ± StDev
ERL	-1234.495 ± 2123.830
RL Only	-21089.835 ± 2688.919

Table 2.1: Summary of final results.

sents the cumulative sum of rewards over the best agent's[1] last episode, averaged over 20 independent trials. The dotted curves around each solid curve represent plus/minus one standard deviation. The final scores are summarized in Table 2.1.

It is worth noting that both experiments start at the same point with rewards. Since *ERL* initializes the population with the baseline morphology, no changes are made until generation one.

[1]Here, the best agent refers to the agent at each generation that scored the highest fitness value.

Chapter 3

Discussion

For the work described so far, these results are only preliminary, as more testing needs to be done. The kind of work done requires more computation than just 50 generations of 10 individuals each. In addition, we used just the `AntWalker`, but other morphologies taken from the `Open AI` baselines would need to be included before any hard claims can be made about this method. However, the baseline morphology used in this thesis did provide a proof of concept that this method is more powerful than only optimizing a controller.

The experimental design in this thesis provides: a main experiment that co-optimizes the body and brain of an agent, a control experiment showing the traditional method (control for total number of evaluations), and a framework that handles the pipeline of this inner and outer loop. The framework can be used with other evolutionary algorithms by implementing new selection and mutation functions for the population data structure, and other RL algorithms can be used thanks to the simplicity of `Open AI`'s implementation.

3.1 Future Work

To take this work further, the obvious route to take is adding more computation time to more broadly cover the search space of both the morphology and controller. The morphology space and controller space constantly change sizes due to the mutation adding and removing body parts, so more time to explore these dimension-changing spaces may provide more insight in how these agents are using their new bodies. More computation would also allow PPO (or any other RL algorithm) to simulate multiple controllers for a given agent at once, allowing for better policies to be found.

Additionally, one could also add the work of Ha to this and include parameterizations of the body parts as learnable parameters [3]. The work done here simply added a copy of one of the leg segments and left it as is, but changing the segment to be longer or thinner was not possible. This was omitted mainly for simplicity, but allowing for more computation could make this possible.

Outside of more computation for faster learning, there are three clear paths to take. The first is giving the reward function more engineering. The one used was created mainly for quadrupedal locomotion, thus changing the amount of limbs can disrupt learning. Second, body parts other than the capsules could be added to expand the possible morphologies. And third, more pressure for learning can be placed on the agents through changing the world they are evaluated in.

Bibliography

[1] Bongard, J. C., Bernatskiy, A., Livingston, K., Livingston, N., Long, J., & Smith, M. (2015, July). Evolving robot morphology facilitates the evolution of neural modularity and evolvability. In Proceedings of the 2015 annual conference on genetic and evolutionary computation (pp. 129-136).

[2] Cheney, N., Bongard, J., SunSpiral, V., & Lipson, H. (2018). Scalable co-optimization of morphology and control in embodied machines. Journal of The Royal Society Interface, 15(143), 20170937.

[3] Ha, D. (2019). Reinforcement learning for improving agent design. Artificial life, 25(4), 352-365.

[4] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

[5] Luck, K. S., Amor, H. B., & Calandra, R. (2019). Data-efficient Co-Adaptation of Morphology and Behaviour with Deep Reinforcement Learning. arXiv preprint arXiv:1911.06832.

[6] Coumans, E., & Bai, Y. (2016). Pybullet, a python module for physics simulation for games, robotics and machine learning. GitHub repository.

[7] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. arXiv preprint arXiv:1606.01540.

[8] Lehman, J., & Stanley, K. O. (2011, July). Evolving a diversity of virtual creatures through novelty search and local competition. In Proceedings of the 13th annual conference on Genetic and evolutionary computation (pp. 211-218).

[9] Deb, K. (2001). Multi-objective optimization using evolutionary algorithms (Vol. 16). John Wiley Sons.

[10] Schmidt, M., & Lipson, H. (2011). Age-fitness pareto optimization. In Genetic programming theory and practice VIII (pp. 129-146). Springer, New York, NY.

[11] Spears, W. M. (1995, March). Adapting crossover in evolutionary algorithms. In Evolutionary programming (pp. 367-384).

[12] Powers, J., Grindle, R., Kriegman, S., Frati, L., Cheney, N., Bongard, J. (2019). Embodiment dictates learnability in neural controllers. arXiv preprint arXiv:1910.07487.

[13] Hauser, H., Fuechslin, R. M., & Nakajima, K. (2014). Morphological computationâThe physical body as a computational resource. Opinions and Outlooks on Morphological Computation, 226-244.

[14] Nielsen, M. Neural Networks and Deep Learning. 2015.