CS81 Final Project Paper:
Using NEAT + ES to Play Games

Do June Min(dmin1)

December 18, 2017

Abstract

Evolutionary Strategy(ES) is a family of evolutionary algorithms that can be used to find a solution to an optimization problem. When the problem is solving a reinforcement problem such as finding a controller to play a game, ES can be used to find parameters for a given controller, where the candidates are vectors of parameter values. However, this approach has the disadvantage that it is not trivial to provide the "right" topology and dimension for the controller, and this might take significant amount of time. When the controller is a neural network, Neuroevolution of augmenting topologies(NEAT) can be used in advance to search for a promising neural architecture to mitigate the problem. After a neural architecture is evolved through NEAT, one may apply ES to the parameters of the evolved network to optimize the controller. We dub this process of evolving a controller NEAT + ES. In this study, we test the performance of NEAT, ES, and NEAT + ES by comparing the fitness of controllers evolved through each algorithm. Although we note that the high variance of each method prevents us from making conclusive statements about the relative performances of each algorithm, we remark that NEAT + ES is still a viable choice for reinforcement learning problems, especially when the dimension of the optimal solution is not clear.

1 Introduction

Evolution Strategy(ES) refers to a class of evolutionary algorithms that maintains a pool of candidate solutions, whose fitness is used to generate the next pool of candidates. Given an initialized parametric policy(usually can represented as a vector of numbers), ES creates a population of perturbed copies of the original solution. This perturbation is achieved by adding some noise(*mutation*) to the parameter vector of the original vector, and then the fitness of each perturbed policy is evaluated. Finally, high-scoring candidates are combined into the policy(*selection*). Note that ES only performs cross-over in an indirect manner, by combining the policy vectors of the parent and top fit candidates. If used as a black-box optimization method, it can be used to "train" neural networks in a similar manner to how policy gradient searches policy space.

However, this similarity of ES to traditional gradient search techniques suggests that there is a shared downside of the two techniques. Despite efforts[1] [2] to answer the question of tuning hyperparameters for neural networks, there are simply intractable number of possible combinations to explore, especially with the rise of very deep neural architectures, such as ResNet[3] or Densely Convoluted Networks[4]. Conducting a grid search of hyperparameters is hence usually impractical, and researchers and practitioners often rely on domain experts, heuristics, or even trial-and-error to settle on an architecture.

In this paper, we experiment using Neuroevolution of augmenting topologies(NEAT) to deal with this problem, motivated by suggestions [5] [6] that genetic algorithms can be used to optimize

neural network architectures. The intuition behind our method is to evolve a network topology first using NEAT, and then converge to an optimal controller for the evolved architecture using ES.

1.1 Related Work

OpenAI released a paper[7] claiming that several problems typically tacked using reinforcement learning techniques can be successfully solved using ES. Their findings are that ES is (1) competitive with traditional reinforcement learning techniques in terms of finding optimal policy, (2) robust to small changes in the environment, (3) often better at exploring different situations without needing stochastic policy, (4) better at dealing with the credit assignment problem, (5) highly parallelizable. Although it is not our main result, this study demonstrates that ES can be used to solved problems that have been traditionally tacked by reinforcement learning methods.

Schumiduber also suggests that neural networks for reinforcement learning problems can be evolved through evolutionary algorithms[8]. Moreover, Wiestra's article on natural evolutionary strategy[9] and Hansen's tutorial on covariance matrix adaptation ES(CMA-ES) discuss different implementations of ES in detail. For a more broad overview of genetic algorithms, refer to [10].

Zhang et al[5] suggest using an Occam's razor inspired fitness function to evolve both the topology and weights of a neural network simultaneously, but instead of following this approach we use NEAT as detailed in [11] to obtain a neural architecture first. A recent work from Google's DeepMind [1] also suggests maintaining a pool of candidates to tune hyperparameters, but they also follow the simultaneous approach and seem to focus on hyperparameters other than network topology.

2 Evolutionary Algorithms Used

Evolutionary algorithms refer to a group of metaheuristic methods inspired by the process of evolution[10]. Typically, the process consists of initialization, selection, mutation, and optionally, crossover of candidate solutions. The pool of candidates is referred to as population.

Initialization of population can be done by simply assigning random values for each candidate. Then, each candidate is evaluated, yielding corresponding fitness values. These fitness values are used in the selection process, where a subset of the current population is chosen. The chosen candidates are then mutated, yielding a new set of candidates. Then, the new candidates can be combined(crossover) to yield offsprings. Note that the order mutation and crossover can be exchanged. This process is repeated until a goal fitness is reached or allotted resources are used up.

2.1 Evolutionary Algorithms for Reinforcement Learning Problems

Evolutionary algorithms can be used for a wide array of problems, but two characteristics make them particular useful and novel approaches to reinforcement learning tasks.

2.1.1 Parallelization

Recent works in deep reinforcement learning techniques have been focused on accelerating the speed of finding good solutions by using specialized architecture designed for parallel evaluation of different agents. One example is asynchronous advantage actor-critic method (A3C)[12]. However, several evolutionary algorithms can be easily adapted to parallelization without needing a specialized architecture, since each candidate in the population can be evaluated independently from others.

This is not the case of deep reinforcement learning techniques which uses backpropagation, since a network has to be adjusted with reference to the previous iteration's network.

2.1.2 Credit Assignment Problem

A significant challenge in solving reinforcement learning tasks is determining which action can be attributed to observed reward(or punishment). Deep reinforcement techniques typically assign a value to an action(or an action-state pair), but there is no guarantee that the association of action and value is correct. For instance, in the game of Chess, a move might have long-lasting consequences. Thus, standard $Q-$table or $Q-$approximation approaches might require a lot of simulations.

Evolutionary algorithms sidestep this problem by ignoring the history of actions and rewards. All they require to work is the final accumulated result, which is taken as fitness of the candidate.

2.2 Evolutionary Strategies

Evolutionary strategy is distinguished from other EAs by the way it generates the offspring population. Instead of doing mutation and crossover, it selects a subset of the population and applies noise(typically Gaussian) to generate the next population.

Algorithm 1 A Generic ES Algorithm

1: candidates $\leftarrow$ randomized vectors
2: **while** Termination criterion is not met **do**
3:     fitnesses $\leftarrow \emptyset$
4:     **for** candidate in candidates **do**
5:         fitness = ***evaluate***(candidate)
6:         fitnesses.append(fitness)
7:     candidates = ***sample_next***(fitnesses, candidate)
8: **return** candidates[argmax(fitnesses)]

There are two subroutines in the above pseudocode, evaluate and sample_next. For reinforcement learning tasks, evaluate is simply simulating a play with the candidate solution and taking the accumulated reward. The choice of sample_next may vary, but we follow that of CMA-ES[13], where the covariance matrix of solution is used to adaptively sample the solution space.

CMA-ES often converges faster than other variants of ES, because it is able to adaptive adjust the distribution of the next set of candidates. This can be likened to casting a wider net, if the good candidates in the previous population were scattered around. Conversely, a smaller net would be cast if the top candidates were relatively clustered.

2.3 NEAT

We skip a detailed discussion of Neuroevolution of augmenting topologies(NEAT) in this paper, but remark that NEAT optimizes both the network topology and the connection weights simultaneously, since the mutation involves perturbing the weights and adding/deleting new nodes and connections. Thus, when an optimal dimension is already found, the optimization of weights will be slower for NEAT than other genetic algorithms that exclusively optimize weights.

2.4 NEAT + ES

NEAT + ES is simply a sequential process of running ES after NEAT.

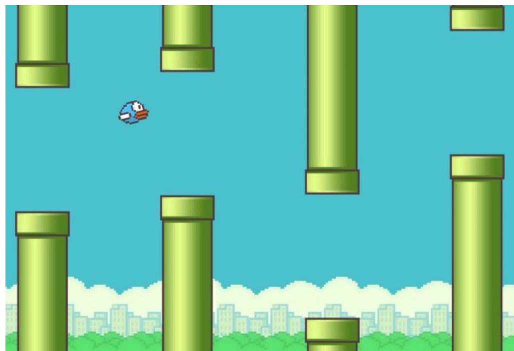
1. Run NEAT until the *termination criterion* is met.
2. Extract the evolved fittest genome's architecture.
3. Do ES on the weights of the evolved network(Candidate vectors are weights to the NEAT-developed network).

However, we note that determining when to stop NEAT(the termination criterion) is not trivial and the timing might have a significant impact on the optimality and convergence speed of the resulting solution. One possible option is to terminate NEAT when all candidates stagnate for a preset amount of generations, but this might result in suboptimal controllers stuck in deceptive optima.

3 Experiment

3.1 Test Environment: Flappy Bird

To test the three evolutionary algorithms, we used the game of Flappy Bird, as implemented in PyGame Learning Environment(PLE)[14] python library. In Flappy Bird, the user controls a bird-like character that is automatically moving to the right and is evaluated by how far the bird has progressed without hitting pipe-shaped obstacles. At any given moment, the user can either (1) do nothing (2) or press 'Up', which will accelerate the bird upwards. The difficulty of the game can be adjusting by changing the amount of gap between the upper and lower pipes. In this experiment, the gap was set to 100, making the game quite difficult for human players.



(a) Screenshot of a Flappy Bird game instance

	Game States
1	player y position
2	player velocity
3	next pipe distance to player
4	next pipe top y position
5	next pipe bottom y position
6	next next pipe distance to player
7	next next pipe top y position
8	next next pipe bottom y position

(b) List of Game States

We chose Flappy Bird as the testing bed for the evolutionary algorithms since Flappy Bird has already been successfully played using both deep reinforcement technique(DQN) and evolutionary algorithm(NEAT). Hence, we could compare the speed of convergence can be made, without having to worry about whether any of the algorithm will be able to find a good solution. However, it remains a new result to determine if ES can also find players on par with DQN or NEAT evolved controllers.

To supply the controller with information about game states, internal game state variables were directly accessed and fed as inputs to the neural network. There are 8 relevant variables at any given moment of the game, as provided in the following table. However, in the following experiments, we

fed not only the current game states but also the previous time step's information. This is due to reduce the effect of perceptual aliasing.

3.2 Perceptual Aliasing

By perceptual aliasing, we refer to the phenomenon of an agent unable to distinguish two different game states solely from the information it is provided. In Flappy Bird, it is possible that the 8 game states are not sufficient for the controller to determine if the bird is in an upward acceleration or downward acceleration, if only the current states are provided. This information is crucial for a good controller, since it might results in different optimal actions, even when the location of the bird is identical at time step t.

To prevent the existence of possible alias states, we feed game states of two consecutive steps as inputs for the controller. If the agent has access to the game states at both the current time step t and previous step $t-1$, the agent can determine upward or downward motion, by comparing its altitude at t and $t-1$. Empirically, the agents with access to two consecutive time state variables converged to a better solution faster. This is why the neural architectures have input dimension of 16.

3.3 Algorithm Configuration

The following table lists the parameters used for each of the evolutionary algorithms used. NEAT_POP and ES_ each refers to the fixed size of the population for NEAT and ES, while NEAT_GEN and ES_GEN each denotes the number of generations NEAT or ES was ran. Default values were used for parameters not listed in the table.

Algorithm	NEAT_POP	NEAT_GEN	ES_POP	ES_GEN
NEAT	150	20	N/A	N/A
ES	N/A	N/A	20	150
NEAT + ES	150	10	10	150

Figure 2: List of Parameters Used

Although we have not conducted a systematic grid search of parameters, the parameters were chosen in a heuristic effort to maximize the efficiency of each algorithm. For NEAT, the size of population was set large(150) to maximize the diversity of candidates in order to cast a "wide" net to search the space of possible topologies.

For ES, we empirically found out that the number of generations are somewhat similar to training epochs for standard neural network training, as the performance seemed to increase with the number of iterations. Thus, we used a larger value(150) for the number of generations.

Finally, NEAT + ES parameters were chosen to make sure that it had the same number of simulations, making the comparison fair. This choice involved implicitly setting a transition point from NEAT + ES. Note that this transition is made arbitrarily, without consideration of available information, such as stagnation, or number of species.

3.4 Dealing with Random Noise: Repeated Experiments

One challenge in measuring the performance of these evolutionary algorithms is that they are susceptible to the initial starting points, as each of the algorithm involves stochastic sampling over the solution space. Moreover, all three algorithms are not guaranteed to find optimal solutions. To

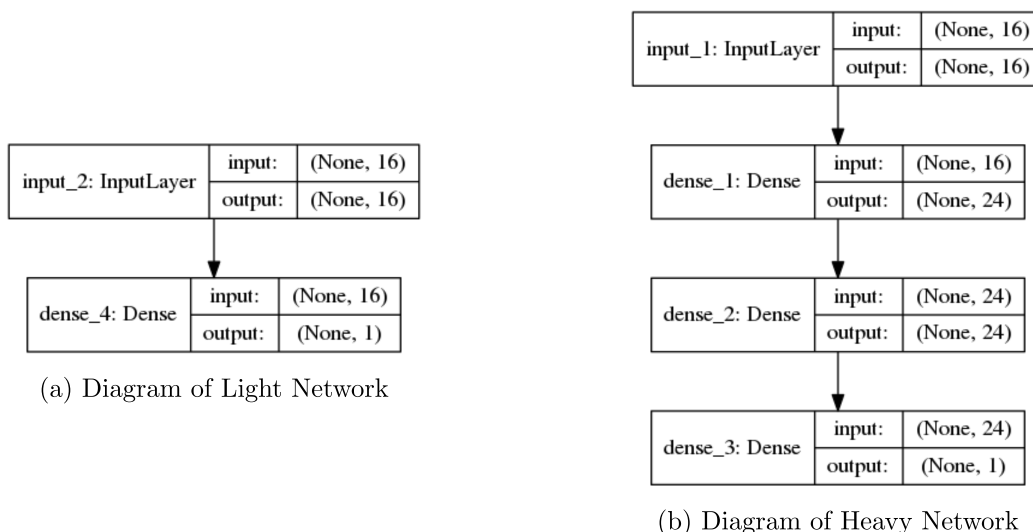
exacerbate the problem, Flappy Bird is also a stochastic environment, in that for each game, the pipes are randomly distributed around the environment.

To mitigate the variance in observed fitness resulting from this multi-fold stochasticity, we used the average of 5 experiments for each algorithm. Thus, the total number of evaluation is constant at 15000.

$$\text{Total number of Evaluations} = 150 \times 20 \times 5 = 15000$$

3.5 Small vs Large Topology for ES

As mentioned earlier, the success of ES is dependent on the choice of the solution dimension. Thus, we use two different architectures as templates for ES, in order to examine the impact of different topology choices.



(a) Diagram of Light Network

(b) Diagram of Heavy Network

Figure 3: Two network topologies for ES

The network on the left represents the "light" network with 17 parameters, while the right network has 1033 parameters. As we will see later in the paper, the "light" network is likely more closer to the optimal network, suggesting that Flappy Bird is a relatively easy game.

3.6 Hypotheses

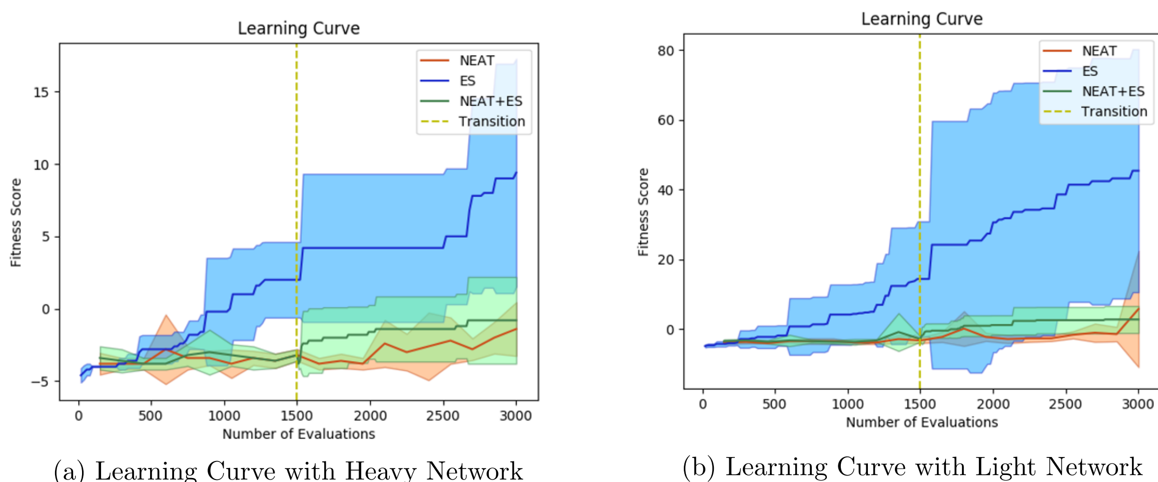
Our first hypothesis is that if the number of evaluations is fixed, NEAT+ES will converge to a better solution than ES alone or NEAT alone. We base our hypothesis on the following logic.

For ES, there is no room for adjustment in the dimension of the parameters. On the other hand, NEAT is a neuroevolution technique that aims to explore policy space of different dimensions by incrementally adding neurons to the network. At the same time, NEAT can but is slower to optimize weights for a given network, since it conducts a simultaneous search of weights and neural architecture. Thus, it is possible that first employing Neuroevolution of Augmenting Topologies(NEAT) to find optimal, or reasonably good neural architecture and then using ES to converge on optimal weights may outperform using ES or NEAT alone.

Moreover, our second hypothesis that ES to be able to find controllers that match or outperform the controllers evolved through NEAT.

3.7 Results

For the following learning graphs, the shaded region correspond to the area within the standard deviation of the mean fitness, which are denoted by the line curves. Also, the dotted yellow line marks the transition from NEAT to ES for the NEAT+ES experiment.

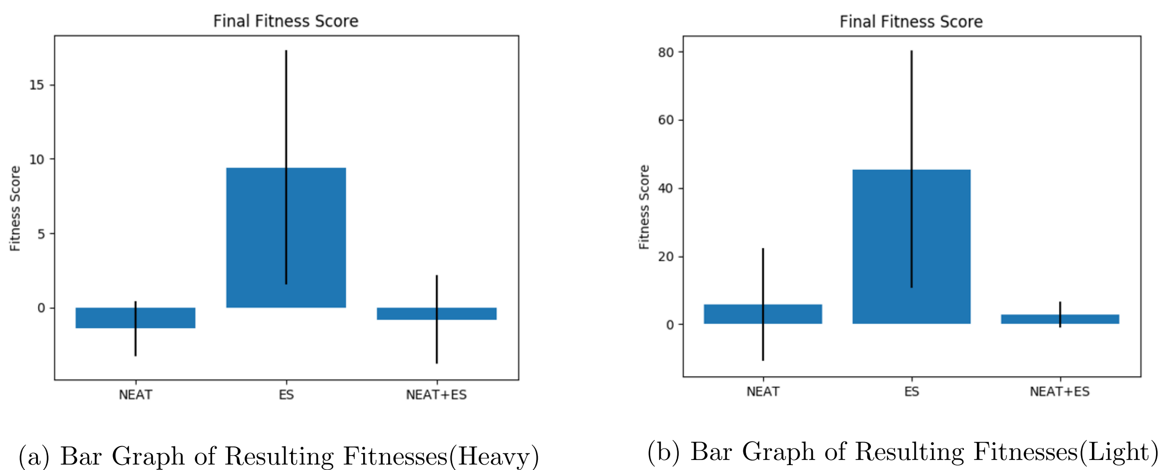


(a) Learning Curve with Heavy Network

(b) Learning Curve with Light Network

Figure 4: Resulting Learning Curves

For both sets of experiments, we observe from the bar graphs that ES seems to result in Flappy Bird controller networks with the highest mean fitness. The vertical lines in the graphs correspond to the standard deviations. Thus, our second hypothesis on the success of ES algorithm is confirmed.



(a) Bar Graph of Resulting Fitnesses(Heavy)

(b) Bar Graph of Resulting Fitnesses(Light)

Figure 5: Resulting Bar Graphs

However, we note that the light network reached much higher average fitness score than its heavy counterpart. This suggests that the optimal dimension of neural network controllers for Flappy Bird is closer to that of the light network. Also, this strengthens our observation that the performance of ES is heavily dependent on the choice of the network dimension.

Also, from the learning curves, we note that after the transition from NEAT to ES, the mean fitness score achieved by NEAT+ES rises faster than in the NEAT phase. This confirms our

intuition that ES is able to optimize the weights faster than NEAT.

More importantly, we remark that both experiments fail to confirm our hypothesis, since ES alone outperformed both NEAT and NEAT+ES in average fitness. While this confirms our minor hypothesis that ES can find a good solution, there is a very high variance in the fitness scores for all three algorithms, as the following bar graphs show. For the "heavy" experiment, NEAT+ES may have achieved higher terminal fitness score than ES alone in some iterations, while the same is true for NEAT alone for the "light" experiment.

Last but not least, the variance is most pronounced for ES, suggesting that ES is most susceptible to the choice of the initial population.

3.8 Flappy Bird as an easy task

The following diagrams represent the architectures of the evolved neural networks resulting from NEAT or NEAT+ES. Even though the network of NEAT+ES algorithm is basically the product of NEAT cut-short, their topology and connections are highly similar.

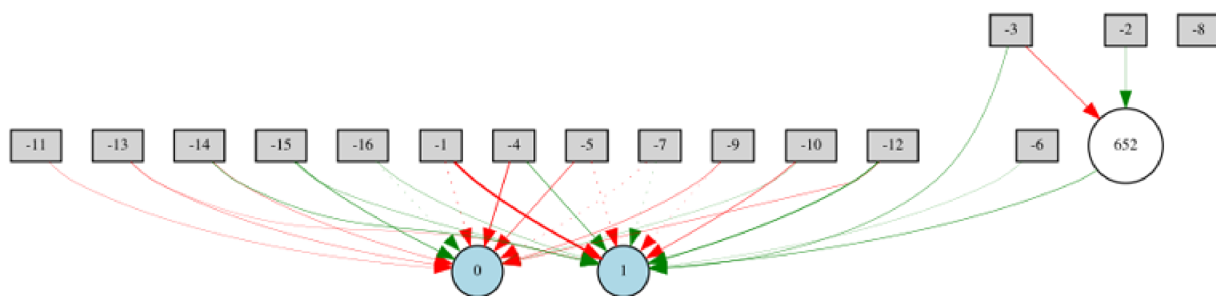


Figure 6: Resulting Neural Network from NEAT algorithm

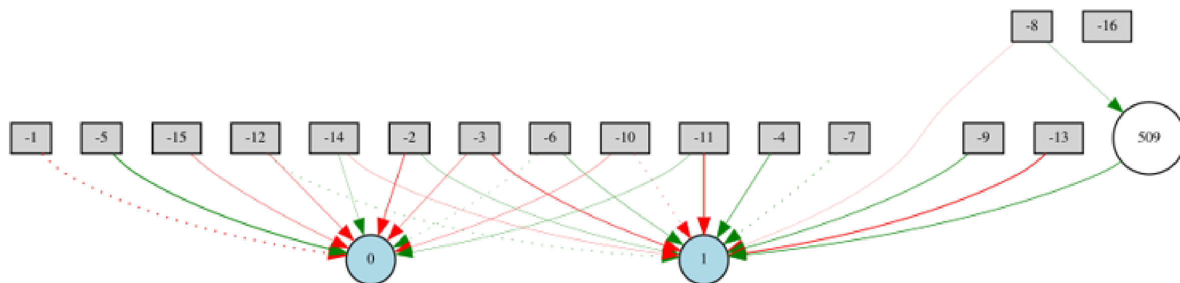


Figure 7: Resulting Neural Network from NEAT+ES algorithm

The similarity of the NEAT networks to that of the ES light network suggest that Flappy Bird can be optimally played by a relatively small, basic controller. Thus, there is a possibility that ES outperformed its competitors as a result of starting with a near-optimal neural network. If that is the case, it is natural that ES converges to a better controller faster, since weight optimization is more important.

4 Conclusion and Future Work

The experiments conducted in this paper fail to provide a conclusive result on the performance of ES. To achieve this, a larger experiment with additional measures to reduce the variance of each algorithm is required.

One way to achieve this is to use average fitness instead of 1-shot fitness. Rathern than simulating one game with a given candidate and taking the resulting score, one could play multiple random games and use the average score as the fitness for that candidate. Hopefully, this will reduce the variance resulting from the stochasticity of the game. Another remedy suggested by Henry Feinstein(student of CS81), is to use a fixed initial state for all three algorithms. However, it will be a difficult task to sample "representative" initial points, so that the results reflect the properties of the solution space. Moreover, it is not clear how to fix the starting points when each algorithm might use different architectures. Lastly, we conclude that a longer, larger experiment is needed. However, an example experiment with 10 repetitions, each with 150,000 evaluations ran for 2.5 days, and thus time remains the largest obstacle. Theoretically, parallelization can mitigate the problem by reducing the time complexity of the task.

Despite the inconclusive empirical results, NEAT+ES might still be a viable heuristic process for finding a solution for optimization tasks, especially when there is only little to no information about the optimal dimension of controllers. While NEAT can be used to explore the dimension space, it may be slow in converging to the optimal weights, even after converging to the optimal dimensions, because NEAT does both weight and topology optimization at the same time.

However, a more useful NEAT+ES process would involve an informed way of testing if NEAT has converged to at least a "good enough" network topology. One key weakness, and possibly the reason for the inconclusive result of the experiments, is that the transition from NEAT to ES was made arbitrarily.

References

[1] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu. Population Based Training of Neural Networks. *ArXiv e-prints*, November 2017.

[2] D. Stathakis. How many hidden layers and nodes? *International Journal of Remote Sensing*, 30(8):2133–2147, 2009.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *ArXiv e-prints*, December 2015.

[4] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely Connected Convolutional Networks. *ArXiv e-prints*, August 2016.

[5] Byoung-Tak Zhang and Heinz Mhlenbein. Evolving optimal neural networks using genetic algorithms with occam's razor. 7, 02 1995.

[6] A Fiszelew, Paola Britos, A Ochoa, Hernn Merlino, E Fernandez, and R Garcia-Martinez. Finding optimal neural network architecture using genetic algorithms. 27:15–24, 01 2007.

[7] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *ArXiv e-prints*, March 2017.

[8] J. Schmidhuber. Deep Learning in Neural Networks: An Overview. *ArXiv e-prints*, April 2014.

[9] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, and J. Schmidhuber. Natural Evolution Strategies. *ArXiv e-prints*, June 2011.

[10] Kenneth de Jong. Evolutionary computation: a unified approach. In *Genetic and Evolutionary Computation Conference, GECCO '14, Vancouver, BC, Canada, July 12-16, 2014, Companion Material Proceedings*, pages 281–296, 2014.

[11] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[12] V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. *ArXiv e-prints*, February 2016.

[13] Nikolaus Hansen. The CMA evolution strategy: A tutorial. *CoRR*, abs/1604.00772, 2016.

[14] Norman Tasfi. Pygame learning environment. https://github.com/ntasfi/PyGame-Learning-Environment, 2016.