CS230

Improving the Performance of Evolutionary Algorithms via Gradient-Based Initialization

Chris Waites
Department of Engineering
Stanford University
waites@stanford.edu

Matthew Prelich
Department of Engineering
Stanford University
mprelich@stanford.edu

Andrew Shoats
Department of Engineering
Stanford University
ashoats@stanford.edu

Abstract

In this work, we propose the use of gradient-based optimization as a means for initializing an evolutionary algorithm population to approach deep reinforcement learning problems. We investigate the efficacy of this method on several standard reinforcement learning benchmarks by evaluating improvements in the policies of the resulting models and their sample complexity. We find that our hybrid approach surpasses DQN and GAs alone in common RL bechmarks, and show that a population of five agents per generation with good initialization could be used to outperform a population of five hundred randomly initialized agents.

1 Introduction

Our work is motivated from the following two claims, which many would argue are commonly held observations in practice. The first is that the benefits of gradient-based optimization methods are often pronounced during the beginning of training, but are often suppressed as one attempts to move from locally optimal solutions to those which approach the global optimum. That is, gradient-based optimization methods are often quite capable of moving quickly from their region of initialization to broad regions of lower loss, but often struggle or are completely incapable in their standard formalization of escaping locally optimal solutions towards the end of training. Alternatively, it's reasonable to claim that in the general case, evolutionary algorithms effectively behave in the reverse manner. That is, evolutionary algorithms are generally quite effective in reaching solutions which are closer to the global optimum by virtue of the fact that they place a heightened emphasis on global search as compared to gradient-based optimization. Although, this property is only observable in the case that the algorithm is able to get off the ground from its random initialization to somewhat reasonable candidate solutions, which is generally considered to be quite time consuming. That is, given the operations available during an evolutionary algorithm, namely random perturbation and crossover, the algorithm generally has a much harder time finding reasonable candidate solutions via these operations from random noise than combining already good solutions to find better solutions.

It's worth noting that there has been recent success in applying both of these methods to a variety of reinforcement learning benchmarks. Although, acknowledging the previously outlined observation, it begs the question as to whether one could benefit from exploiting the upsides of both algorithms, noting their asymmetric benefits. In response to this, we propose investigating the degree to which gradient-based initialization of evolutionary algorithms is able to improve the training of reinforcement learning agents, be it in terms of improved performance of the resulting models or an overall reduction in necessary computational expenditure. So specifically, our hypothesis is that training a population of models via gradient-based methods and then transitioning to an evolutionary algorithm would simultaneously yield better performing models and the time to reach such solutions would be reduced in comparison to using either method used in isolation. On a high level the ideal outcome of this would be, if we were able to train a gradient-based model for some small number of iterations and use an evolutionary algorithm to completion, that the overall training time would be significantly reduced and the computed policy would generalize better. Surprisingly, to the best of our knowledge, a thorough investigation of the degree to which this initialization improves upon the aforementioned metrics has not been released and is the overarching goal of this work.

2 Related Work

The application of deep learning to the task of reinforcement learning was first proposed in "Human-Level Control Through Deep Reinforcement Learning" [8], in which Mnih et al. propose the original deep Q-network, capable of learning policies directly from high-dimensional sensory input. The method was evaluated on the notoriously difficult domain of Atari 2600 games, and they were able to show that the deep Q-network, receiving only pixels and game score as inputs, was able to surpass the performance of all previous benchmarks and achieve professional human ability across 49 games. The deep Q-network now acts as a foundational algorithm from which the majority of gradient-based approaches to solving reinforcement learning today stem from.

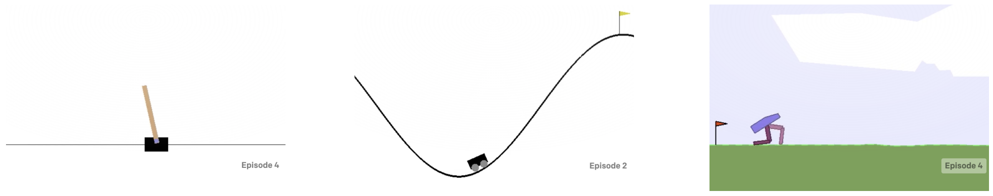
Evolutionary algorithms have been shown in recent years to be competitive alternatives to gradient-based optimization methods for deep reinforcement learning. For example, in "Evolution algorithms as a Scalable Alternative to Reinforcement Learning" [9], Salimans et al. explore the use of evolutionary algorithms, a class of black box optimization algorithms, as an alternative to popular gradient-based methods such as deep Q learning and policy gradients. They perform several experiments on MuJoCo and Atari environments to show that evolutionary algorithms are a viable solution that scale well with the number of CPUs available.

In "Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning" [10], Such et al. propose a similar approach of tackling reinforcement learning problems via evolutionary algorithms, but further develop the previous results to reach the state of the art achieved by gradient-based methods. This expanded the previously held notion of the scale at which genetic algorithms could operate and, by exploiting parallelization, showed significant improvements in training time, being able to solving Atari in approximately 4 hours on a modern desktop.

There has been previous work in incorporating gradient-based information into evolutionary algorithms to approach deep reinforcement learning problems. For example, the most similar work in regard to our approach is "Evolutionary Reinforcement Learning" [7], in which Khadka and Tumer propose a hybrid algorithm proceeding as follows. First, a population of actor networks is initialized with random weights. In addition to the population, one additional actor network (referred to as rl_{actor}) is initialized alongside a critic network. The population of actors (rl_{actor} excluded) are then evaluated in an episode of interaction with the environment. The fitness for each actor is computed as the cumulative sum of the reward that they receive over the timesteps in that episode, and then a relatively vanilla genetic algorithm approach follows. Our work differs in theirs in several ways, but most significantly in the complexity and assumptions made about the training context with a focus on proper initialization. That is, in a similar manner in which Dropout [5] is a rather simple, empirically driven method which can be applied to the majority of conceivable neural network architectures to improve their generalization properties, we wish to propose a similarly generic augmentation with strong empirical promise. Several benefits stem from this, in particular the avoidance of any dependency on any particular choice in gradient-based/evolutionary algorithm, an invariance to the source of the initilizations, as well as ease of implementation.

3 Dataset and Features

Gym [1] is a reinforcement learning toolkit for researchers proposed by OpenAI. It includes a collection of common benchmark problems that expose a common interface. To be precise, the following Gym environments were considered: `CartPole-v1`, `MountainCar-v0`, and the Atari games `Breakout` and `Frostbite`. For reference, a full detailing of the following values for each environment can be found at `https://gym.openai.com/envs/`. For `CartPole-v1`, a pole is attached by an unactuated joint to a cart, which moves along a frictionless track. Hence, the



state space of the environment are vectors of the form $\{x, v, \theta, \omega\} \in [-2.4, 2.4] \times (-\infty, \infty) \times [-41.8°, 41.8°] \times (-\infty, \infty)$ where x denotes the position of the cart, v denotes the velocity of the car, θ denotes the angle of the pole, and ω denotes the velocity of the tip of the pole. The system is controlled by applying a force of +1 or -1 to the cart, and hence the action space is $\{-1, +1\}$ for all states $s \in S$. The pendulum starts upright, and the goal is to prevent it from falling over. So a reward of +1 is provided for every timestep that the pole remains upright, and the episode ends when any of the following criteria are met: the pole is more than $15°$ from the vertical, the cart moves more than 2.4 units from the center, or the pole has been balanced for 200 total time steps.

For `MountainCar-v0`, a car is on a one-dimensional track positioned between two smooth hills. The goal is to drive up mountain by only driving back and forth to build up momentum. The state space of the environment are vectors of the form $\{x, v\} \in [-1.2, 0.6] \times [-0.07, 0.07]$ where three actions are available at each state, namely $a \in \{-1, 0, 1\}$ denoting the direction in which to push the cart. A reward of -1 is yielded at each time step until the goal position of 0.5 is reached. The agent is initialized at a random state $s_0 \in [-0.6, -0.4]$ and the episode terminates once the goal position is reached or 200 time steps have passed.

4 Methods

For background, when referring to training a model via gradient-based method, in our implementation and approach, we are specifically referring to training a deep Q-network as opposed to alternative gradient-based methods such as policy gradient methods. In training a deep Q-network we iteratively apply the following update rule, in which at iteration t, θ_t denotes the parameters of the network, s_t denotes the state of the model within the environment, a_t denotes the action taken at that state, s'_t denotes the following observed state, r_t denotes the observed reward, and γ is the discount factor. Naturally we reassign each state s_t to s'_t for the next stage of optimization, and sample an action to take at each step in proportion to the outputs of the Q-network at the current state.

$$\theta_{t+1} = \theta_t - \nabla_{\theta_t} \left(Q_{\theta_t}(s_t, a_t) - (r_t + \gamma \max_{a'_t} Q_{\theta_t}(s'_t, a'_t)) \right)^2$$

A genetic algorithm (GA) is a specific class of evolutionary algorithms which function as gradient-free population-based metaheuristic optimization algorithms. A GA works by evolving a population of N individuals through what are called generations. In this process the weights of the network are not updated using backpropagation (i.e. gradient descent). Instead, GAs merely require forward propagation to evaluate the networks, making GAs ideal for parallelization since network evaluations are independent and, consequently, entire populations can be evaluated at once. In order to ensure standardization, we followed the same genetic algorithm implementation as Uber AI in [10] which demonstrated a basic mutation genetic algorithm outperforming alternative RL algorithms such as DQN, A3C, ES, etc. At each generation, the parameters are changed through a mutation function which adds Gaussian noise to the parameter vector. After each individual network is evaluated, the best performing network in this generation is preserved for the next generation, referred to as elitism. Subsequently, a parameter selected uniformly at random from the top T performing networks is mutated by adding Gaussian noise. In effect, the best performing networks are passed down by generation and thus, in theory, imply a monotonically decreasing average loss as a function of generations. The mutation function is crucial as it introduces exploration of the feature space.

For our hybrid algorithm, we utilize a gradient-based algorithm (in this case, DQN) to train a population of learned parameter models for initialization (i.e. the first generation) to a gradient-free genetic algorithm. The degree to which the DQN models are trained before passing as initialization to the GA is a tunable hyperparameter explored in subsequent sections. In theory, less difficult RL

Table 1: Expected cumulative reward for given episode limit, averaged over 20 algorithm initializations

CartPole-v1										
Algorithm	50	100	150	200	250	300	350	400	450	500
DQN	51.94	138.55	162.73	177.85	181.92	229.35	320.41	393.59	435.27	483.21
EA	21.91	26.29	27.73	31.545	35.205	36.835	41.635	47.65	52.27	57.725
Hybrid	70.16	142.21	183.19	223.53	275.69	380.71	480.12	500.00	500.00	500.00

Table 2: Expected number of episodes to achieve given reward threshold, averaged over 20 algorithm initializations

CartPole-v1										
Algorithm	50	100	150	200	250	300	350	400	450	500
DQN	48	84	121	284	327	340	376	405	478	531
EA	970	1875	2490	3000	3614	4012	4211	4397	4591	4794
Hybrid	21	23	27	29	30	32	34	37	40	41

tasks such as the inverted pendulum can utilize the partial degree of training for saving important resources (CPU run-time, number of genotypes/individuals per generation, number of generations, etc.) whereas more difficult or computationally expensive games such as Atari ones can utilize fully trained DQN models for initialization in order to obtain better optima.

5 Experiments, Results, and Discussion

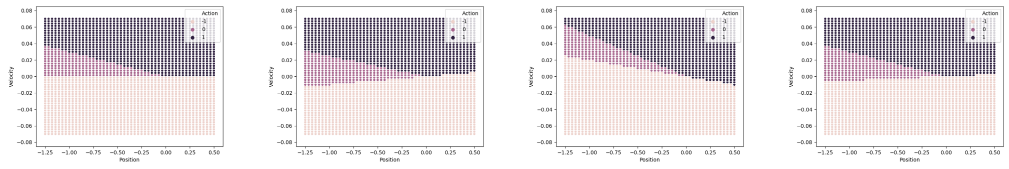
5.1 Local Gym Environments

To quantitatively assess our methods, we see the evaluation of a given reinforcement learning algorithm to be measured along two axes, in particular, the quality of the resulting models and the sample complexity. To be concrete about model performance, we quantify the quality of the resulting models for a given reinforcement learning algorithm as the expected cumulative reward when trained for a predefined, fixed number of episodes, averaged across training initializations and evaluation trajectories. That is, we train several models with different initializations, feed them a fixed number of episodes, and observe the average reward they achieve when we observe their trajectories during evaluation. We report these values for different environments across multiple algorithms and episode limits in 1. Similarly, we define the sample complexity as the expected number of episodes required to reach a candidate agent which is capable, in expectation, of exceeding some set reward threshold. That is, we train several models with different initializations and observe the average number of episodes the model needs to see until, on average, the model exceeds some fixed reward threshold during evaluation. We report these values for several different environments across multiple algorithms and reward thresholds in 2, as well as the total CPU time in 3.

To qualitatively assess our methods, we opt to perform policy visualization. That is, we compare the learned policies of our agents against an expert and allow for subjective visual assessment of their similarity. Although, the state spaces of the environments are often of reasonably high dimension, and hence visualizing the policy directly is not tractable. Hence, we opt to train agents on the

Table 3: Expected CPU runtime to achieve given reward threshold, averaged over 20 algorithm initializations

CartPole-v1										
Algorithm	50	100	150	200	250	300	350	400	450	500
DQN	2.7	5.1	9.4	17.4	22.7	45.8	60.1	93.0	123.2	162.5
EA	3.3	6.7	13.9	28.1	53.5	94.1	130.5	153.4	172.9	210.0
Hybrid	0.9	1.1	4.1	4.5	6.8	8.3	10.2	12.1	13.9	15.6



MountainCar-v0 environment which happens to have a two dimensional state space for this task, rather than potentially losing state information during some dimensionality reduction technique such as PCA. We linearly interpolate through the valid state space and query a model trained on exactly five hundred episodes (split half and half between DQN and EA for the hybrid approach) using each of the proposed methods. A figure of an expert model, a model trained as a DQN, a model trained via an evolutionary algorithm, and a model trained via our hybrid approach is shown in 5.1. In terms on analysis of the results of this figure, it intuitively reaffirms that our method has a slight advantage over other approaches in converging to an optimal policy in fewer samples to the environment.

5.2 Atari

"Deep Neuroevolution" [10] showed a compression technique to evolve a 4 million+ parameter network by passing random number generator seeds for the initialization and evolutions to the workers instead of passing the large parameter network each time (effectively passing in a compressed model). This presents a difficulty for us since we can't just pass an initialization seed to the network (they have to be the trained DQN models). Thus, the group proposed having each model class contain an ID attribute which corresponds to any DQN trained model. The master sends the compressed network with the ID attribute to the worker (thereby only sending a few bytes of data). Afterwards, the worker looks up the DQN initialization corresponding to this model's ID in O(1) time by us defining a hash table with ID's to the DQN trained models. Effectively, this allows us to bypass the previous constraint that weights cannot be customly initialized when utilizing this compression technique. For our Atari results, the group ran into issues with cost overruns as we needed to run 720 cores on 10 AWS EC2 instances. We used over 80% of our AWS course credits so did not wish to continue running in costs go over. For future work, we hope to acquire more AWS credits and run with less machines to reduce costs.

6 Conclusion and Future Work

Overall, our work introduces a hybrid algorithm which effectively surpasses DQN and GAs alone in the popular RL bechmarks of `CartPole-v1` and `MountainCar-v0` in terms of both expected reward and time complexity. Moreover, the results also showed that a population of a mere 5 individuals per generation could be used to achieve better expected rewards in faster time than using a population of 500 individuals. As a further note, our analysis represents performance without the use of distributed computing. This means we essentially ran the DQN initializations and forward propagations in series, which is sub-optimal. Since gradient based techniques (e.g. DQN, DDQN, A3C, etc.) require forward and backward passes on a single network, they are great at leveraging the power of single GPUs. However, with our hybrid algorithm, one can take notable advantage of burgeoning distributed computing services to run the DQN initializations and GA forward propagations on an enormous number of CPUs instead of single GPUs.

With respect to future work, we would like to evaluate our proposed method on a broader class of problems of varying difficulty, including MuJoCo and Atari. Another promising direction would be to acknowledge that our proposed method could be interpreted as a special case of a potentially more general methodology. Alternatively, one could see this as randomly sampling a model update method from either of the two algorithms at each episode during training, selecting the gradient-based approach with probability 1 until iteration t, and then selecting the evolutionary update with probability 1 from t until convergence. A more general methodology would be to have this probability between the two methods decay over time according to some update rule. That is, let the probability of selecting the evolutionary update be $P_t(update = evo) = \frac{1}{1+e^{-(at+b)}}$, with hyperparameters a and b designating the emphasis on gradient-based versus evolutionary based and the sharpness of the transition between the two.

7 Contributions

To preface the following section, it's worth noting that the authors together agree that they were each comfortable with the level of effort put forth by their peers and would consider the project done via roughly equal contribution. Although, we acknowledge that this is up to the discretion of the course advisors.

Chris was the primary contributor to the final paper and poster presentation, writing the majority of the content aside from portions from the dataset overview section and the methods section. He was also responsible for designing and executing on the qualitative assessment and formalizing the quantitative evaluation metrics contained within this report. Additionally, he contributed to the implementation of the models and collection of the empirical results, running many of the experiments. Chris will also be available for the presentation of the poster.

Andrew was the primary lead for the experimental results and implementations presented within this report, being responsible for sifting through public repositories and augmenting them fit our needs. In particular, he gathered the quantitative results for evaluating model performance and managed our implementations of DQN and EA. Andrew will also be available for presentation of the poster. He also suggested to use the CartPole problem as a proof-of-concept instead of going straight to the Atari games.

Matthew was primarily responsible for experimenting with the Atari "Deep Neuroevolution" repository, augmenting its implementation for custom initialization with compressed models, and working with AWS. He also formulated the original hypothesis of using gradient-based methods for initialization to gradient-free algorithms (originally proposed with either Nelder-Mead or GA) based on the idea of saddle points (stuck from gradient methods) being optimized wrt to certain dimensions/parameters and not others. Thus, in theory, this optimized subset of the feature space which was found through gradient based methods will tend to be retained through generations in a genetic algorithm while simultaneously allowing the non-optimal (likely corresponding to relative maximums in its corresponding dimension) weights to be varied. This should beget a continually improving population of parameters. He also contributed heavily to the literature review, final report, the process of verifying that our contribution was novel, and suggested the idea of using reinforcement learning instead of strongly supervised learning tasks like image classification.

References

[1] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.

[2] Paras Chopra. *Reinforcement Learning Without Gradients: Evolving Agents Using Genetic Algorithms*. Jan. 2019. URL: `https://github.com/paraschopra/deepneuroevolution`.

[3] Ahmed Fawzy Mohamed Gad. *NeuralGenetic*. 2018. eprint: `10.1007/978-1-4842-4167-7`.

[4] John J. Grefenstette, David E. Moriarty, and Alan C. Schultz. "Evolutionary Algorithms for Reinforcement Learning". In: *CoRR* abs/1106.0221 (2011). arXiv: `1106.0221`. URL: `http://arxiv.org/abs/1106.0221`.

[5] Geoffrey E. Hinton et al. "Improving Neural Networks by Preventing Co-adaptation of Feature Detectors". In: *CoRR* abs/1207.0580 (2012). arXiv: `1207.0580`. URL: `http://arxiv.org/abs/1207.0580`.

[6] Max Jaderberg et al. "Population Based Training of Neural Networks". In: *CoRR* abs/1711.09846 (2017). arXiv: `1711.09846`. URL: `http://arxiv.org/abs/1711.09846`.

[7] Shauharda Khadka and Kagan Tumer. "Evolutionary Reinforcement Learning". In: *CoRR* abs/1805.07917 (2018). arXiv: `1805.07917`. URL: `http://arxiv.org/abs/1805.07917`.

[8] Volodymyr Mnih et al. "Human-Level Control Through Deep Reinforcement Learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 00280836. URL: `http://dx.doi.org/10.1038/nature14236`.

[9] Tim Salimans et al. *Evolution Strategies as a Scalable Alternative to Reinforcement Learning*. 2017. arXiv: `1703.03864 [stat.ML]`.

[10] Felipe Petroski Such et al. "Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning". In: *CoRR* abs/1712.06567 (2017). arXiv: `1712.06567`. URL: `http://arxiv.org/abs/1712.06567`.