Re-Evolutionary Algorithms
(Combining Policy Gradient Methods in Reinforcement Learning with Evolutionary Algorithms)

Devang Agrawal
ICME Dept.
devang18

Kaushik Ram Sadagopan
MechE Dept.
kaushik7

Fatma Tlili
CS Dept.
ftlili

Abstract

We aim to develop a hybrid evolutionary reinforcement learning algorithm and apply it to a classic control problem to prove its superiority over the standalone algorithms. We implement a policy gradient algorithm (Advantage Actor Critic - A2C) and an evolutionary algorithm (ES) for the cartpole problem on OpenAI gym. Subsequently, we combine A2C with ES for the cartpole problem to show that it performs better than the standalone algorithms.

1 Introduction

Reinforcement Learning is a form of policy search algorithms which given an environment aims to find the actions to be taken to maximize cumulative rewards. Deep Reinforcement Learning (DRL) algorithms use neural networks to solve control problems which have high-dimensional state and action spaces by learning to map the state-action space to their corresponding rewards. While these algorithms proved to be effective in applications such as robotics, control and in games such as Go, they are known to suffer from some difficulties such as high sensitivity to hyper-parameter settings and limited feature space exploration. A class of search algorithms which addresses these problems of DRL is the Evolutionary Strategies (ES). These algorithms fall under black-box optimization, which estimate the parameters of the policy neural network to optimize the cumulative rewards with no regards to the given environment. While ES algorithms are more stable than DRL and can explore the feature space better, they suffer from low exploitation of the environment feedback signals and tend to have poor performances in most applications due to their high sample complexity.

1.1 Related Work

Some of the main challenges that deep reinforcement learning algorithms, including policy gradient algorithm tend to suffer from are temporal credit assignment with sparse rewards, lack of exploration, and extreme sensitivity to hyperparameters tuning. Evolutionary Strategies tend to overcome these challenges by probabilistically selecting promising candidate out of a population of candidates thus allowing more exploration.

(Khadka and Tumer, 2018) combines Evolutionary Strategies algorithm and a Deep RL agent by first training some RL actors and then periodically injecting gradient information into the EA population.

This paper (Houthooft et al., 2018) introduces an evolved differentiable loss function where the loss is parametrized via temporal convolutions over the agent‘s experience which uses the previous knowledge and experience of the agent for future tasks.

(Pourchot and Sigaud, 2018) combines simple cross-entropy method (CEM) and TD3 (a deep reinforcement learning algorithm) by first initializing a population of actors and dividing it into two groups. The first group is directly evaluated while the second group follows the direction of the gradient given by the critic in TD3.

Similarly, in the work of (Colas et al., 2018) a replay buffer is filled with exploratory trajectories and then DDPG is run on that data. While this approach doesn't use ES directly it is similar to ES in focusing on the diversity of the learned policies.

Finally (Maheswaranathan et al., 2018)‘s work uses the gradients from a gradient descent algorithm such as Q-learning to modify the covariance matrix of an ES algorithm to change the distribution from which the population of candidates is sampled.

1.2 Environment

A pole is attached to a cart which moves along a frictionless track. The system is controlled by moving the cart right or left. The pole starts upright and the goal is to prevent it from falling over. The objective of this task is to keep the cartpole upright continuously for 200 timesteps which corresponds to a reward of 200. (Fig: 1)

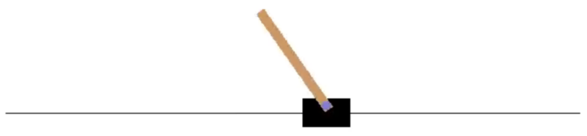

Figure 1: Cartpole problem on OpenAI Gym

2 Methods

2.1 Advantage Actor Critic

We implement an Advantage Actor Critic (A2C) policy gradient algorithm for the cartpole problem. We get a probability distribution for the actions for each state and we choose actions sampled from that distribution. The advantage is calculated by finding the difference between an estimated average future reward and the average current value of the state. These advantages are used to scale our current predictions directly into our policy gradient.

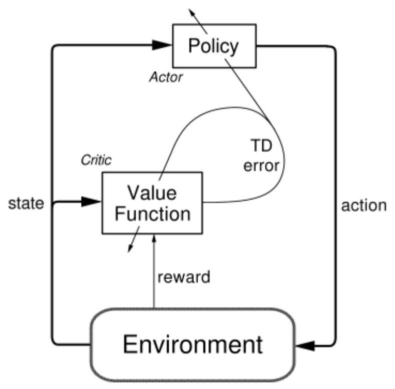

Figure 2: actor-critic architecture (Sutton and Barto, 1998)

2.1.1 Policy Gradient

For the policy gradient, we output a policy to take an action given a specific set of states. This policy gradient algorithm will learn a set of weights for each action based on the observations within the environment. We minimize the negative of the logarithm of probabilities of each action weighted by the advantages. This process is done using TensorFlow and the loss is minimized using AdamOptimizer for backpropagation.

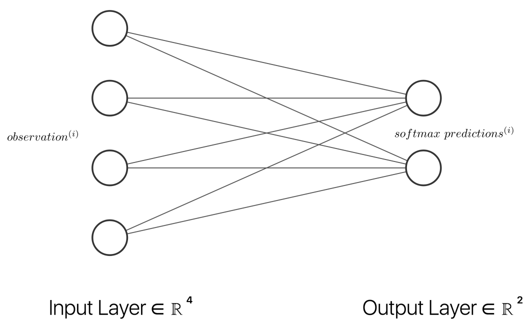
$observation^{(i)}$
$softmax\ predictions^{(i)}$
Input Layer $\in \mathbb{R}^4$
Output Layer $\in \mathbb{R}^2$

Figure 3: Policy Gradient architecture which outputs predictions of each action

2.1.2 Value Gradient

For the value gradient, we learn parameters that compute the advantage of taking a particular action given an observed state. We minimize the least squared error between future reward value estimations and current average reward value estimations, then update the weights for our value gradient neural network for calculating our current advantage estimations. New values are calculated by using a discounted Monte-Carlo simulation, which will place importance on short-term reward rather than long-term reward using a discount factor. Once we can compute an advantage, we can then feed this directly back into our policy gradient. For a single observation, we utilize a hidden layer with 10 neurons with the ReLU activation function, and then its output is subtracted from the DMC (Discounted Monte-Carlo) value of that state, and we obtain an output representing the advantage.

2.2 Evolutionary Strategies

We spawn instances of the parameters which are jittered by random noise. One episode of cartpole is run with each parameter instantiation and the total reward at the end of the episode is calculated. This is computationally efficient because we can parallelize the code to run each episode with its parameter instantiation.

One way of creating an offspring of parameters is to weight the noises (of each parameter instanti-

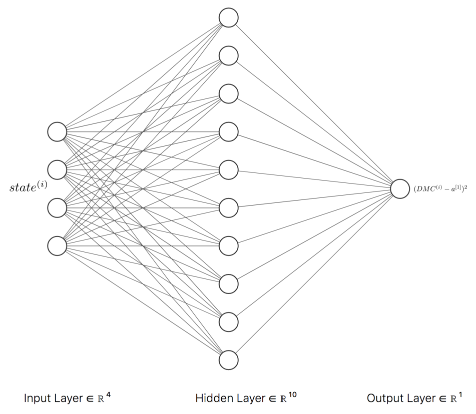
$state^{(i)}$
$(DMC^{(i)} - a^{[1]})^2$
Input Layer $\in \mathbb{R}^4$
Hidden Layer $\in \mathbb{R}^{10}$
Output Layer $\in \mathbb{R}^1$

Figure 4: Value Gradient architecture using a single hidden layer

ation) with a value proportional to the total reward obtained at the end of the episode with that parameter instantiation produced by the noise. We normalize the total rewards obtained for each parameter instantiation in our population. We linearly combine these noises (proportional to their total rewards), normalize this and we update the parameters accordingly.

Another way is to choose the parameters corresponding to the maximum reward obtained by that parameter instantiation at the end of that episode.

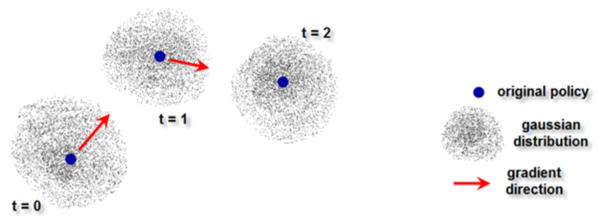

Figure 5: Evolutionary Strategies

In the first plot (Fig: 6) the candidate parameters of the population are weighted by their corresponding rewards. In the second plot (Fig: 7) only the candidate which corresponds to the maximum reward is chosen. The plots shows that taking the best candidate parameters is better than the weighted combination of the candidates.

3 Experiments

3.1 Vanilla E-A2C

We combine ES and A2C iteratively in a sequence. Each iteration of algorithm spawns parameters and makes an update by choosing the best candidate

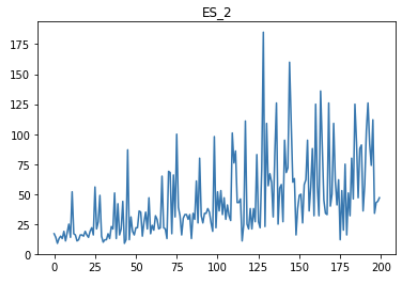

Figure 6: Weighted Combination of Candidates

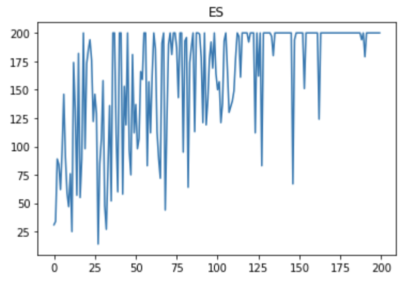

Figure 7: Maximum Candidate Parameters

or the weighted combination of candidates. The weights updated by the ES is passed on to the policy gradient function of the A2C algorithm which performs a gradient descent update. ES spawns a population of parameters from the gradient descent updated parameters from A2C and the sequence continues. (Fig: 8)

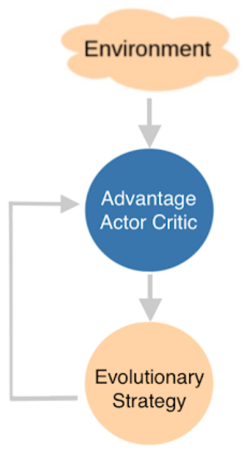

Figure 8: Vanilla E-A2C

3.1.1 Preliminary Experiments

We use the weighted combination of parameters proportional to the rewards obtained on the episode for this combination. The performance evaluation parameter we use for the comparison of these algorithms is the number of episodes it takes to reach a reward of 200 (which implicitly means that the weights have undergone sufficient training to produce a satisfactory total reward). The number of instances of parameters we spawn (noises we generate) is set to 50, we used a noise standard deviation of 0.1 and a learning rate of 0.001 for the ES algorithm. We expect the ES algorithm to take the longest time to reach a reward of 200 and expectedly so it takes 1160 episodes to reach a reward of 200 averaged over 100 different training sequences. The A2C algorithm takes 266 episodes to reach a reward of 200 averaged over 100 different training sequences. Our vanilla evolutionary A2C algorithm takes 226 episodes to reach a reward of 200 averaged over 100 different training sequences, and we observe that it performs better than both the standalone algorithms.

3.2 Evolutionary A2C

In our final combination, ES spawns a population of parameters and A2C updates each member of the population by performing a series of gradient descent updates. Finally ES chooses the best parameter vector (based on the rewards obtained) in the population, and injects noise onto this parameter vector to generate the new population.

We inject noise in the parameters of the policy gradient function which contain the information of the action choices for all of the 200 states for a given episode. A2C trains for 5 episodes in each epoch we have plotted the rewards for. (Fig: 9)

3.3 Mountain Car Environment

We extended our algorithm to the mountain car environment (Fig: 10) but our evolutionary A2C algorithm failed to learn meaningful value to the parameters since there is an exploration issue in the task. Policy gradient methods face an issue in this environment because the rewards are very sparse. So even if the car explores the correct side of the track, there is no gradient descent update made by A2C to the policy since this is an on-policy algorithm and does not re-use the data to a later state when it actually reaps the reward for that exploratory action. Hence, for this problem off-policy algorithms like DQN are effective.

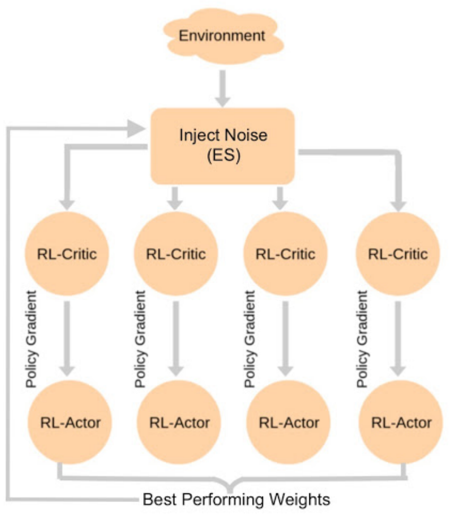

Figure 9: Evolutionary A2C

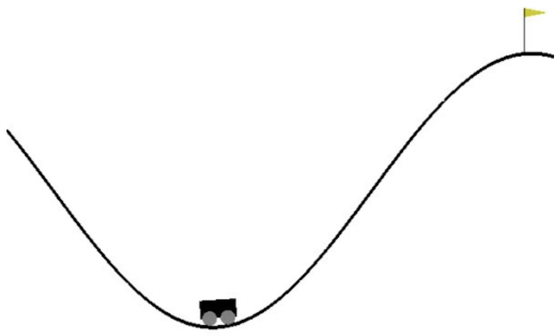

Figure 10: Mountain Car Problem

4 Results

The training converges when the average reward reaches 200 consistently. A2C reaches this state at around 75 epochs (Fig: 11) but it has a lot of variation due to the stochasticity in the selection of actions. ES has a lot of variation at the beginning but stabilizes after 150 epochs (Fig: 12). Vanilla E-A2C reaches this state after 125 epochs (Fig: 13). The evolutionary A2C (Fig: 14) is clearly superior to the other three algorithms in the sense that it is the quickest to converge and the variations in the reward are minimal after reaching this state.

The code for each of these algorithms is attached herewith:
https://drive.google.com/open?id=19OUh2 Rywr97RsLUmFC$_tLaBIRB-ytey5$

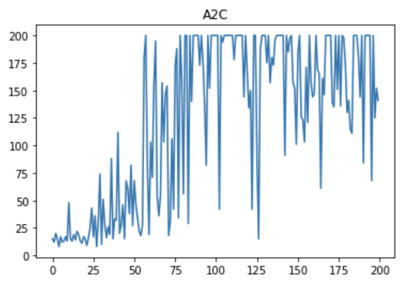

Figure 11: A2C

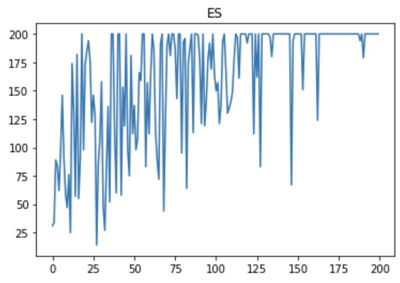

Figure 12: ES

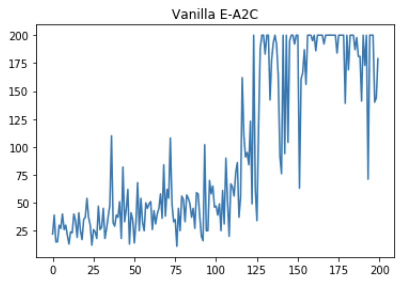

Figure 13: Vanilla E-A2C

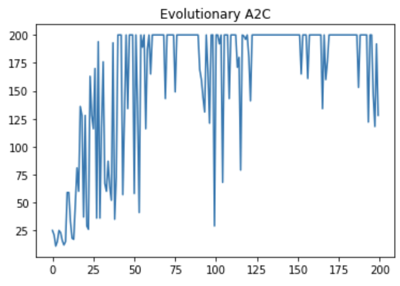

Figure 14: Evolutionary A2C

5 Conclusions and Future Work

We conclude that our combined evolutionary actor critic algorithm is most efficient at the cartpole task compared to the standalone algorithms we've chosen. We observe that ES instantiating a population of policy gradients is effective in that it explores better and uses the environment signals to arrive at the optimal solution quickly. We could explore combinations of Proximal Policy Optimization with a similar evolutionary strategy to solve tasks on MuJoCo.

6 Contributions

Literature study of previous work on combining policy gradient algorithms with ES - Fatma
Implementing a standalone A2C - Kaushik Ram
Implementing a standalone ES - Devang
Vanilla combination of A2C and ES - Devang and Fatma
Final combination of A2C and ES - Devang and Kaushik Ram
Extending combined algorithm to the mountain car problem - Devang
Writing the report - Kaushik Ram and Fatma

References

Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. 2018. Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms. *arXiv preprint arXiv:1802.05054* .

Rein Houthooft, Richard Y Chen, Phillip Isola, Bradly C Stadie, Filip Wolski, Jonathan Ho, and Pieter Abbeel. 2018. Evolved policy gradients. *arXiv preprint arXiv:1802.04821* .

Shauharda Khadka and Kagan Tumer. 2018. Evolutionary reinforcement learning. *arXiv preprint arXiv:1805.07917* .

Niru Maheswaranathan, Luke Metz, George Tucker, and Jascha Sohl-Dickstein. 2018. Guided evolutionary strategies: escaping the curse of dimensionality in random search. *arXiv preprint arXiv:1806.10230* .

Aloïs Pourchot and Olivier Sigaud. 2018. Cem-rl: Combining evolutionary and gradient-based methods for policy search. *arXiv preprint arXiv:1810.01222* .

Richard S. Sutton and Andrew G. Barto. 1998. Reinforcement learning: An introduction. http://www.incompleteideas.net/book/the-book.html.