Automated Gait Generation for Simulated Bodies using Deep Reinforcement Learning

Abhishek Ananthakrishnan
Computer Engineering Department,
K. J. Somaiya College of Engineering,
Vidyavihar, Mumbai
abhishek.aa@somaiya.edu

Vatsal Kanakiya
Computer Engineering Department,
K. J. Somaiya College of Engineering,
Vidyavihar, Mumbai
vatsal.kanakiya@somaiya.edu

Dipen Ved
Computer Engineering Department,
K. J. Somaiya College of Engineering,
Vidyavihar, Mumbai
dipen.ved@somaiya.edu

Grishma Sharma
Asst. Professor,
Computer Engineering Department,
K. J. Somaiya College of Engineering,
Vidyavihar, Mumbai
neelammotwani@somaiya.edu

Abstract—

A popular problem to solve these days is to propose an algorithm such that a body autonomously learns locomotion. Some of the most efficient and popular contemporary algorithms are Deep Reinforcement Learning algorithms. In this paper, we study three such algorithms from recent times, namely - Deep Deterministic Policy Gradient, Advantage Actor Critic, and Proximal Policy Optimization. We implement and compare the algorithms on the performance metric of average reward per epoch. Given our implementations, we then draw our conclusions on the efficiency of the algorithms proposed and rank the algorithms based on the same.

Keywords—Reinforcement Learning; Reward Function; Value Function; Policy; Action Space; Observation Space; Actor - Critic; Policy Gradient; Mujoco; PyTorch

I. INTRODUCTION

There have been huge advancements in the field of reinforcement learning in the past few years, with the creation of deep reinforcement learning. The achievement of high performance in games like Atari Breakout, Pong, Mario has proved to be revolutionary. More recently, DeepMind has used reinforcement learning to teach a computer to play the game Go, which demonstrated better performance as compared to an older model for Go trained using supervised learning. This marked a huge step in reinforcement learning. The scope of reinforcement learning is not restricted only to games. Using the concept of assigning rewards for performed actions to achieve a specific goal, models have been created for near human chatbots, and for robots that can learn to perform tasks such as walking, running autonomously, with no external input.

In this paper, we compare a few of the foremost reinforcement learning models proposed to teach robots to walk autonomously. We implement and compare three models, namely - Proximal Policy Optimization, Deep Deterministic Policy Gradient Algorithm, and Advantage Actor Critic. We compare their performance in terms of the cumulative reward gained after a series of episodes and epochs. The algorithms have been tested on different bodies and environments designed by the team at OpenAI [2] using the MuJoCo physics engine [8]. We have implemented the algorithms using PyTorch, an open source machine learning library.

II. BASIC TERMINOLOGIES USED

- *Reinforcement Learning*

Reinforcement Learning (RL) is learning what to do so as to maximize a numerical reward signal[1]. It deals with performing the action which would give maximum reward. Reward could be set taking in consideration multiple aspects. RL allows the simulated machines to learn their behaviour according to the reward received from the environment

- *Reward Function*

A reward function defines the goal in a reinforcement learning problem. On each time step, the environment sends to the reinforcement learning agent a single number called the reward. The agent's objective is to maximize the total reward it receives in the long run[1]. The reward is denoted by the symbols R or r.

- *Value Function*

A value function specifies what is good in the long run. The value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. Values indicate the long term desirability of states [1]. It is denoted by the symbol $V(s)$ where s is the current state.

- *Policy*

A policy is a mapping from perceived states of the environment to actions to be taken when in those states[1]. A policy is represented by the symbol π.

- *Action Space*

Action Space is the set of all actions that the model can take at any point of time. It is denoted by the symbol A.

- *Observation Space*

Observations are required to understand the effect of action on the environment. Observation Space describes the format of valid observations.

- *Actor - Critic*

The Actor Critic algorithms are a set of reinforcement learning algorithms which change the policy before the value settles. In the, the Actor is responsible for improving the policy and estimating how changes to the policy affect the reward. The Critic performs policy evaluation and evaluates the reward given the current policy is used as is. Using the two concurrently, one can ensure that any change to the policy by the Actor leads to an increase in the reward predicted by the Critic.

- *Policy Gradient*

Policy gradient methods learn a policy function from trajectories generated by the current policy by optimizing the parameters of the policy with respect to the expected long term reward by gradient descent.

- *MuJoCo*

MuJoCo stands for Multi-Joint dynamics with Contact. It is a physics engine that is used for model based control. It is primarily used for research purposes in animations, robotics, mechanics, etc.

- *PyTorch*

PyTorch is an open source machine learning library for Python. It is based on Torch and is used in various machine learning concepts and applications.

III. Algorithms

In the recent years there has a been a lot of research in the field of reinforcement learning toward building agents with autonomy. To achieve this autonomy in various tasks and environments, DeepMind Technologies has implemented several algorithms that have now become an important class of algorithms. Following are some of the popular algorithms :-

A. Deep Deterministic Policy Gradient

DDPG stands for Deep Deterministic Policy Gradient [5]. It is a policy gradient algorithm that evaluates a deterministic target policy, which is much easier to learn. It adopts a stochastic behaviour for the purpose of better exploration. Consider a reinforcement learning setting, where an agent is currently in state 's' at time step 't'. The agent performs an action 'a_t' at time step 't', receives a reward 'r_t' and transits to state 's_{t+1}'. The DDPG algorithm consists of a parameterized actor neural network $\mu(s|\theta^\mu)$ and a critic neural network $Q(s, a|\theta^Q)$. The neural network $\mu(s|\theta^\mu)$ performs mapping of states to a specific action and the critic neural network $Q(s, a|\theta^Q)$ learns using an off-policy algorithm such as Q-Learning and outputs the expected total reward given a state 's_t' and action 'a_t'. θ^μ and θ^Q act as the weight parameters. Also, there are additional copies of both networks viz. Actor copy $\mu'(s|\theta^\mu)$, Critic copy $Q'(s, a|\theta^Q)$, referred to as 'target networks', which are responsible for performing computation. When samples are generated by sequential online exploration, the assumption of identically distributed samples no longer holds. To solve this issue, DDPG relies on use of replay memory buffer. The transitions resulting from action 'a_t' are stored as a tuple (s_t, a_t, r_t, s_{t+1}) in this memory buffer. Then, transitions are sampled from the environment in the form of a random mini batch of N transitions (s_i, a_i, r_i, s_{i+1}). The expected return is then calculated by

$$y_i = r_i + \gamma Q' (s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$

where γ is the discount factor.

Then the critic network is updated by minimizing the loss function L observed with respect to the predicted total reward using $Q(s_t , a_t |\theta^Q)$ i.e.

$$L = (\Sigma_i(y_i - Q(s_i , a_i |\theta^Q))^2 /N$$

The actor network is updated with following gradient estimate.

$$\nabla_{\theta\mu} J \approx (\Sigma_i \nabla_a Q(s, a|\theta^Q)|_{s=si,a=\mu(si)} \nabla_{\theta\mu} \mu(s|\theta^\mu)|_{si})/N$$

The above updates did not lead to convergence. Hence there is a need for soft updates to the parameters of the target networks which led to better stability of learning. Following equations are followed for target updates.

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1 - \tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1 - \tau)\theta^{\mu'}$$

where τ is the soft update constant.

B. Advantage Actor Critic (A2C)

A2C stands for Advantage Actor Critic Algorithm. There are 2 parts to this algorithm - the Actor Critic Algorithm and the Advantage function.

Similar to DDPG, A2C also uses actor critic approach. In A2C, the network estimates both value function V(s) and the policy. The agent potentially uses a value estimated by the critic to update the policy estimated by the actor.
The second part of the algorithm is the usage of an advantage estimator. A2C uses fixed-horizon advantage estimator for the purpose of variance reduction. These estimators are used for updating the policy.
For each iteration in A2C, the agent acts for T timesteps. Since, we do not calculate Q values, here we evaluate expected returns as an estimate of the Q(s, a) value. The expected return is calculated using the following equation.

$$R_t = r_t + \gamma r_{(t)+1} + ... + \gamma^{T-t+1} r_{T-1} + \gamma^{T-\ t} V(s_t)$$

where R_t is net total return and γ is the discount parameter.
Now, the advantage obtained is framed as

$$A_t = R_t - V(s_t)$$

where A_t represents the estimated advantage function and $V(s_t)$ is the baseline value function.
To have better exploration, A2C makes use of entropy. The advantage loss function can be stated as[5]

$$\text{Loss Gradient} = \nabla_\theta \Sigma_t [\log \pi(a_t|s_t)\, A_t + \beta H(\pi(s_t))]$$

where β is a hyperparameter and H is the entropy. Using this loss function, one step gradient descent results in a gradient estimator, which is used to get a stochastic gradient descent algorithm.
The two parts form an intermediate level algorithm termed A2C. A paper by *V. Minh et. al.* [7] has suggested improvements to A2C by making it asynchronous. This entails training multiple models on the same environment and then updating just one central model.

C. **Proximal Policy Optimization**

PPO stands for Proximal Policy Optimization [6][9]. Without affecting the performance adversely between each policy update, PPO tries to update the policy conservatively. It is scalable to large models. It can have parallel implementations. It is data efficient and successful on a variety of problems where hyperparameter tuning is not used.

Consider the world as an environment. This environment is in a state 's' at a time 't'. The agent, which is the decision-making system, gets to choose an action from a set of actions. After choosing an action 'a', it gets executed. As a consequence, the environment changes its state. The environment emits a reward. The reward indicates the condition of current environment after execution of action 'a'. And the cycle repeats. The hope is to find an agent that can optimize expected sum of rewards in this process.

In policy optimization, this problem is dealt by defining a π_θ with a bunch of free parameters, for example, a big neural net with weights as the free parameters, that will map the states to actions. The objective is to find a set of parameters θ such that the expected sum of rewards is maximized under the policy π_θ in the environment that the agent is in i.e.

$$\max_\theta E[\Sigma_t \gamma^{t-1} r(s_t, a_t) \mid \pi_\theta] .$$

A stochastic policy class is chosen because it actually smooths out the problem. $\pi_\theta(a|s)$ is the probability distribution over all possible actions given their states. Actions can be discrete or continuous. γ being the discount factor.

The policy optimization should give a stable and monotonic improvement. On the other hand, the learning curve should contain fluctuations. For these, the policy must have a good way to choose step sizes. In supervised learning, if the agent takes a step too far, the subsequent updates fix it. While in reinforcement learning, the step too far, results in bad policy and the next batch will be collected from this bad policy which is a different state distribution and becomes hard to recover.

The key idea is to use surrogate objective. To find the expected return from policy π, data is collected from the old policy π_{old}. Some objective must be optimised to get a new policy π. $L_{\pi(old)}(\pi)$ is considered as the surrogate loss.

$$L_{\pi(old)}(\pi) = E_{\pi(old)} [\Sigma_t \gamma^{t-1} (\pi(a_t|s_t) / \pi_{old}(a_t|s_t))\, A^{\pi(old)} (a_t,s_t)]$$

Probability ratio of policies times the advantage under the old policy with expectation over old policy defines the surrogate loss. If π is too far from π_{old}, it can result in a downgrade. There is hence a need to limit the size of the update. In proximal policy optimization, unlike as in Trust Region Policy Optimization (TRPO) [10][11], penalty $KL[\pi_{old}|\pi_\theta]$ is considered rather than constraint to limit the update for first order. $KL[\pi_{old}|\pi_\theta]$ denotes the KL divergence hyperparameter. PPO minimizes the following function over θ.

$$\text{minimize}_\theta\ \Sigma_t (\pi_\theta(a_t|s_t) / \pi_{old}(a_t|s_t))\, A_t - \lambda.KL[\pi_{old}|\pi_\theta]$$

For each iteration in PPO, policy is run for T timesteps. Advantage function is estimated at all timesteps. On the above objective equation, stochastic gradient descent is done for some number of epochs and KL divergence penalty is adjusted adaptively. If the desired change in the policy per iteration $KL_{(target)}$ is too high, increase λ and if $KL_{(target)}$ is low then decrease λ. Due to this, the step size stays reasonable in the learning process.

IV. Experimental Setup

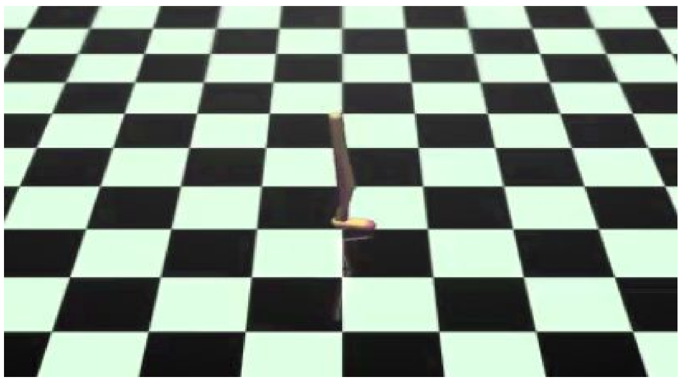
A. **Environment Setup**

We run the aforementioned algorithms on the environments provided by OpenAI Gym. The environments used are namely

'Walker2d-v2' and 'Humanoid-v2'. A description of each of the environments is as given below:

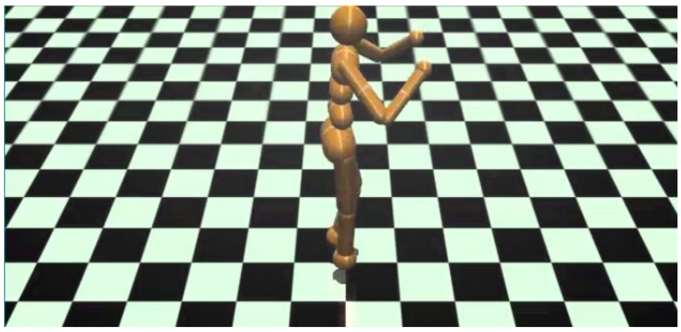
1. Walker2d-v2 : In this environment, the agent is a two dimensional bipedal robot. The goal here is to make the robot walk forward as fast as possible. The action space is described by a vector of 6 real numbers and observation space is described by a vector of 17 real numbers.

The following figure represents the rendered environment.



2. Humanoid-v2 : In this environment, the agent is a three dimensional bipedal robot. The goal here is to make the robot walk forward as fast as possible, without falling over. The action space is described by a vector of 17 real numbers and observation space is described by a vector of 376 real numbers.

The following figure represents the rendered environment.



B. Hardware Setup

The following is the hardware setup used on the training and rendering machine :

- 2.8GHz Intel Core i7-7700HQ
- 4 cores, 8 threads
- Nvidia GTX 1050 GPU with 4GB RAM
- 16GB 2400MHz DDR4 RAM

C. Experiment Setup

For PPO and A2C, the agent is trained for 1000 epochs while for DDPG, the agent is trained for 25 epochs as DDPG doesn't show much change improvement in the average reward obtained over time. In each epoch, a batch size of 5000 steps is performed by the agent. The agent learns for a maximum of 10,000 steps. These parameters are kept consistent throughout. The discount factor γ is kept between 0 and 1.

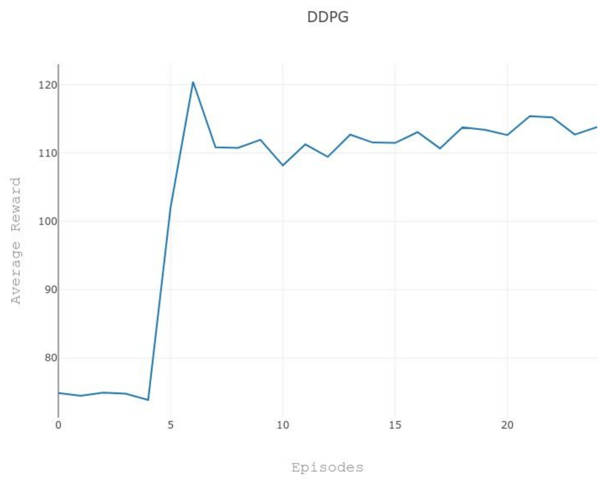
Each neural network has 2 hidden layers of size 64 each. The input layer and the output layer are sized according to the environment.

V. RESULTS

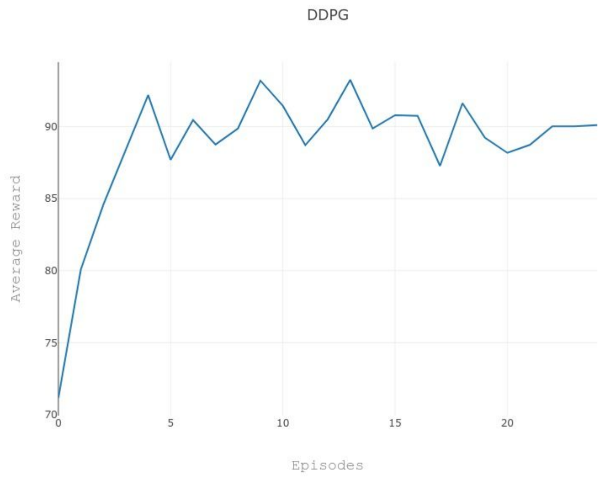
The simulation rendered by the engine describes the movement of the robot. The performance of DDPG, A2C, and PPO algorithms were tested on the environments 'Walker2d-v2' and 'Humanoid-v2' are displayed below graphically with episodes on the X-axis and the average reward obtained per episode on the Y-axis.

DDPG :

Walker2d-v2

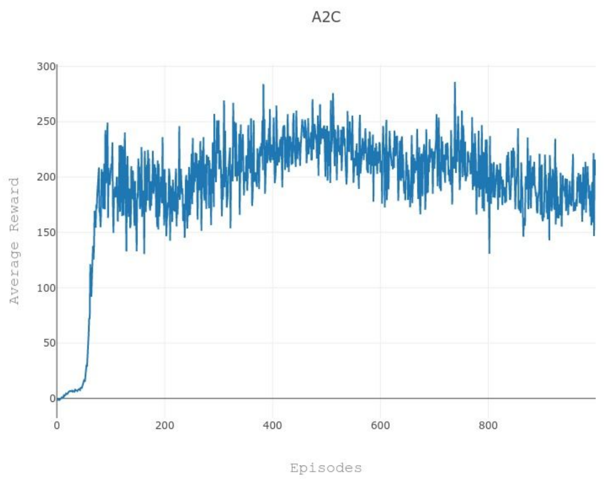


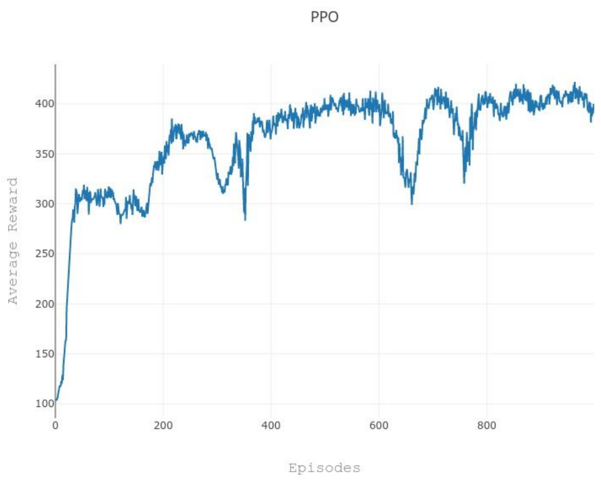
Humanoid-v2



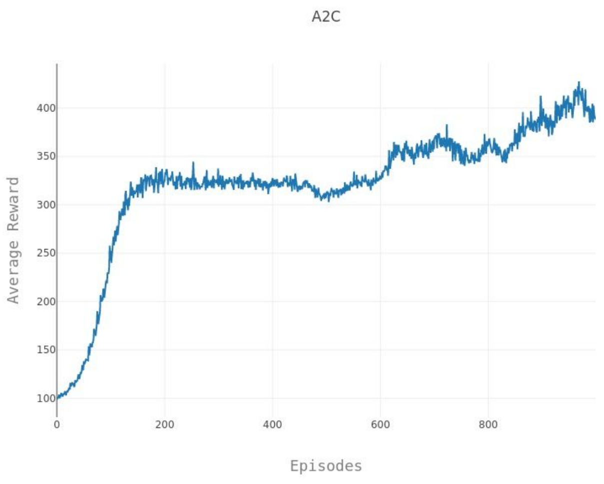
A2C :

Walker2d-v2



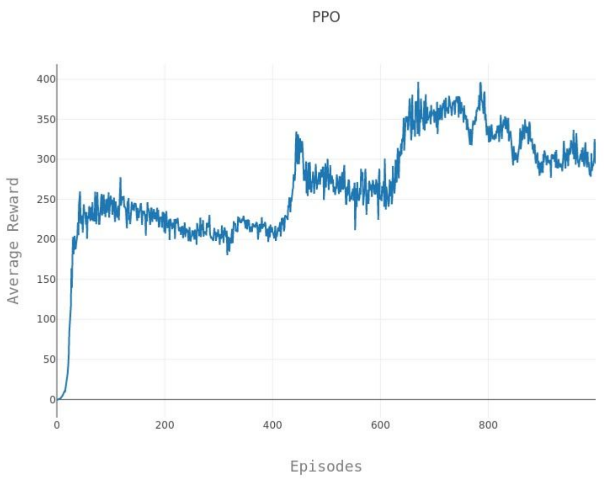


Humanoid-v2



PPO :

Walker2d-v2



Humanoid-v2

VI. CONCLUSION

In this work, we have implemented several policy based reinforcement algorithms to solve a continuous control task such as gait generation. Results show, that among the three algorithms, PPO gave the best average reward per episode at the end of simulation run. The performance of A2C was better than that of DDPG. A reason for poor performance in DDPG is due to use of a deterministic policy, which inhibits the possibility of better exploration. This work provides good interpretation of policy based reinforcement learning algorithms for locomotion tasks.

VII. FUTURE WORK

With PPO turning out to be relatively better algorithm in terms of performance, we plan to use PPO as our lower level controller in a hierarchical framework. The hierarchical framework will consist of a higher level controller that guides the lower level controller to take a step in an optimal direction.

Lower level controller controls how a step is taken by controlling each joint and limb while the higher level controller instructs the lower level controller with the step to be taken for a given terrain map, essentially path planning. The higher level controller will be using policy gradient with convolutional layers. Additionally, Generative Adversarial Networks (GANs) can be used which has two networks. One network generating 2D contour maps and the other verifies for the existence of a path for agent to traverse. These contour maps would be the input for the above hierarchical framework.

VIII. REFERENCES

[1] Sutton, Barto, “Reinforcement Learning: An Introduction”.

[2] OpenAI Gym https://github.com/openai/gym.

[3] Richard S. Sutton, David McAllester, Satinder Singh, Yishay Mansour, “Policy Gradient Methods for Reinforcement Learning with Function Approximation”.

[4] Vijay R. Konda John N. Tsitsiklis. “Actor-Critic Algorithms”.

[5] Lillicrap T., Hunt J., Pritzel A., Heess N., Erez T., Tassa Y., Silver D., and Wierstra D., “Continuous control with deep reinforcement learning”, arXiv:1509.02971, 2015.

[6] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin Riedmiller, David Silver, “ Emergence of Locomotion Behaviours in Rich Environments”, (2017).

[7] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu, “Asynchronous Methods for Deep Reinforcement Learning”.

[8] E. Todorov, “MuJoCo: Modeling, Simulation and Visualization of Multi-Joint Dynamics with Contact (ed 1.0)”. Roboti Publishing, 2015.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, “Proximal Policy Optimization Algorithms”.

[10] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, Pieter Abbeel, “Trust Region Policy Optimization”.

[11] Yuhuai Wu, Shun Liao, Roger Grosse, Jimmy Ba, “Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation”.