8-2021

Effects of Saltatory Rewards and Generalized Advantage Estimation on Reference-Based Deep Reinforcement Learning of Humanlike Motions

Md Rysul Kabir
The University of Texas Rio Grande Valley

EFFECTS OF SALTATORY REWARDS AND GENERALIZED ADVANTAGE ESTIMATION ON REFERENCE-BASED DEEP REINFORCEMENT LEARNING OF HUMANLIKE MOTIONS

A Thesis

by

MD RYSUL KABIR

Submitted to the Graduate College of
The University of Texas Rio Grande Valley
In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2021

Major Subject: Computer Science

EFFECTS OF SALTATORY REWARDS AND GENERALIZED ADVANTAGE ESTIMATION ON REFERENCE-BASED DEEP REINFORCEMENT LEARNING OF HUMANLIKE MOTIONS

A Thesis
by
MD RYSUL KABIR

COMMITTEE MEMBERS

Dr. Dong-Chul Kim
Chair of Committee

Dr. Zhixiang Chen
Committee Member

Dr. Emmett Tomai
Committee Member

August 2021

ABSTRACT

Kabir, Md Rysul, Effects of Saltatory Rewards and Generalized Advantage Estimation on Reference-based Deep Reinforcement Learning of Humanlike Motions. Master of Science (MS), August, 2021, 53 pp., 7 tables, 21 figures, 50 references.

In the application of learning physics-based character skills, deep reinforcement learning (DRL) can lead to slow convergence and local optimum solutions during the training process of a reinforcement learning (RL) agent. With the presence of an environment with reward saltation, we can easily plan to magnify those saltatory rewards with the perspective of sample usage to increase the experience pool of an agent during this training process. In our work, we have proposed two modified algorithms. The first one is the addition of a parameter based reward optimization process to magnify the saltatory rewards and thus increasing an agent's utilization of previous experiences. We have added this parameter based reward optimization with proximal policy optimization (PPO) algorithm. What's more, the other proposed algorithm introduces generalized advantage estimation in estimating the advantage of the advantage actor critic (A2C) algorithm which resulted in faster convergence to the global optimal solutions of DRL. We have conducted all our experiments to measure their performances in a custom reinforcement learning environment built using a physics engine named PyBullet. In that custom environment, the RL agent has a humanoid body which learns humanlike motions, e.g., walk, run, spin, cartwheel, spinkick, and backflip, from imitating example reference motions using the RL algorithms. Our experiments have shown significant improvement in performance and convergence speed of DRL in this custom environment for learning humanlike motions using the modified versions of PPO and A2C if compared with their vanilla versions.

DEDICATION

This work is dedicated to my beloved parents without whose constant support and care I would not be able to successfully reach this current stage of my life.

ACKNOWLEDGMENTS

Throughout the duration of this work and writing this thesis, I have received an immense amount of support, encouragement, and assistance.

I would first like to thank my research advisor, this thesis committee chair Dr. Dong-chul Kim, whose expertise was invaluable in formulating the research questions and methodology. His insightful feedback and consistent support have elevated my thinking capability and brought this work to a higher level.

I would also like to thank Dr. Zhixiang Chen and Dr. Emmett Tomai for their constant support and for all the opportunities they have given me during my stay at UTRGV.

Finally, I would like to thank UTRGV for providing me with the Presidential Graduate Research Assistantship which played a very significant role in my life. Without that funding, I would have never thought about taking a risk and changing my major to computer science. Therefore, I am greatly thankful to Dr. Zhixiang Chen and the Graduate College of UTRGV for seeing my prospect and selecting me for that prestigious award.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER I

INTRODUCTION

This present world is built upon the abundance of the data all around us. Data not only inform us about our surrounding situations but also help us choosing a decision. Data can be collected and presented to us in the form of numbers e.g. the time, date, temperature, and number of wins or losses of an NFL team. Data can also be presented as symbols such as sunny or rain symbols of weather forecasts, or even pictographs. We may collect these data through our own observations, photography, audio and video recordings, surveys, sensors, or from internet of things. With the help of these data accumulated from the various sources, we can create statistical models to figure out the patterns within those collected data. As the pattern is already known, it can expedite and automate the procedure to solve problems in sectors like healthcare, manufacturing, marketing, business, security, education, and even on agriculture with the given values of the independent variables. This way of utilizing the data to learn from experience and automate human-like tasks with the help of machines is called Artificial Intelligence (AI).

Most AI examples we hear nowadays range from transportation in estimating the movement of traffic and suggesting the fastest routes by Google Map, ETA's of Uber, to fraud detection through the data gathered from transactions, and determining creditworthiness by FICO. These AI examples use different machine learning methods and algorithms in order to predict and help making a decision out of the acquired data. Machine learning is a section of artificial intelligence which is established on the idea that a machine's system can learn from collected data, identify patterns inherent to those data, and eventually take decisions with least possible human

interference on the machine by creating statistical models. Early works related to AI started in 1950s with neural networks which stirred the excitement for the machines which can think. From 1980s machine learning started to become quite popular using different statistical models like Linear Regression, Logistic Regression, Decision Trees, SVM, Clusturing, Random Forest. After 2010s deep learning became a trendy topic combining the aspects of both neural network and machine learning. Through deep learning, a deep neural network can approximate any function with the proper use of layers, nodes, and activation functions at the output of the neurons. On the other hand, in the last few years, Reinforcement Learning (RL) has emerged to be a powerful tool to solve complex sequential-decision making problems using Markov Decision Process (MDP).

Reinforcement Learning is a part of Machine Learning that talks about opting suitable actions to maximize the output, in a sense the reward for taking that action, in a particular circumstance. Reinforcement learning differs from the supervised learning methods (e.g. linear regression, decision tree, SVM) where the model has the input data and the corresponding labels as a whole training data set whereas no such dataset exists for RL. On the other hand, reinforcement learning is also different from unsupervised learning which actually finds out the hidden pattern in the collected unlabeled data. Actually a reinforcement learning agent decides what to do to perform a given task after learning from experience by itself. It initially performs poorly because of the lack of experience. However, with time, as it learns from the experience while exploring through different choices, the agent eventually learns to exploit the environment in order to acquire the highest reward and reach its goal eventually. Therefore, we can say reinforcement learning, in a way, simulates the learning process of a human being from his birth to the adulthood. Similar to a newborn, the RL agent initially knows nothing and starts from zero. However, while growing up, the child gradually experiences different aspects of the world through experience and eventually learns which steps to take to have a good consequence. Similarly, the RL agent takes the very same path–learning with experience–to maximize the reward. Therefore, we can say that reinforcement learning holds the true essence of artificial intelligence that

we computer scientists are working for. One of the challenges in reinforcement learning is the trade-off between this exploration to gather knowledge and exploitation maximize the prize. If an agent wants to obtain a lot of reward, it must prefer to perform actions that it has tried in the past and got good rewards as an outcome. However, in order to know which action provides better outcome the agent must explore other actions as well. Therefore, the agent must try different available actions to ensure all the possible states and actions available in the environment have been explored. This will guarantee that the reinforcement agent is actually learning by balancing the exploitation and exploration which is also required in human life.

There are abundant applications of reinforcement learning due to its non specific nature of the problem definitions. This very same theory can be applied to different domain specific problem with little to no effort. Practically, its application ranges from controlling robotic arms to robotic navigation where the collision avoiding behavior can be learnt from negative rewards if it bumps into obstacles. Logic games like Back-Gammon, Chess, Gomoku, Poker, and Go are well suited to reinforcement learning as they require a sequence of appropriate decisions to achieve victory. The world renowned AlphaGo created by a team of DeepMind was the first computer program to defeat Ke Jie who was ranked as number 1 back at that time in 2017. This AlphaGo program was trained by the reinforcement learning algorithm using a Monte Carlo tree search and artificial neural networks to predict and help the AlphaGo program to select appropriate moves to win a game [38]. On the other hand, reinforcement learning is also being used in healthcare for finding out the suitable treatment for health conditions and drug therapies. Moreover, most autonomous cars, trucks, and drones have been built upon having reinforcement learning algorithms at the center.

1.1 Previous Works

There is a long history of modeling complex movements of humanoid or other articulated models in fields like robotics, graphical animations, and even bio-mechanics. Researchers have

been working relentlessly to make movements of such models more natural and human-like and. Previously there have been a lot of works such modeling using numerous kinematic methods where others have been using physics-based methods. With the advent of machine learning, nowadays machine learning community is getting more into taking part in improving control of such models using various machine learning methods. Using machine learning, researchers are improving upon the control of such models using reinforcement learning and different motion imitation methods using reference motions.

1.1.1 Kinematic Models

Kinematic models are used for modeling complicated movements, specially in character animation, when there is a huge number of data available. Kinematic models can generate better quality models than simulation-based methods if we have high quality data. One way of using kinematic models is to use motion clips and use them to build a controller which will execute the relevant clip in a certain situation [2, 16, 33]. On the other hand, gaussian processes can also be used to learn the inherent representations to create motions in different situations [17, 49]. Extending the gaussian process, deep neural networks like autoencoders and phase functioned networks have been used to build generative models to build a controller for human motion [11, 12]. One problem with kinematic models is that they have limited possibilities to synthesize motions that the model can face for the first time in a situation. Moreover, if the goal and environment become complicated, to get rid of this limitation in kinematic models, we have to gather enough data to cover those possibilities. Collecting such high volume of data can become unattainable in complex environments. To circumvent the this problem with limited number of input data we can use prior knowledge from physical properties.

1.1.2 Physics-based Models

Physics-based methods help the models realize how physical properties will modulate the motions depending on various kinds of changes and variations in the environment. There

have been many works on developing controllers for locomotions of both human and non-human characters [5, 48, 50] using physics-based methods. These methods are built upon on a baseline model and an optimization process where the parameters of the optimization process are tuned in to achieve desirable behaviors [1, 8, 46]. What's more, such optimization can also be done using quadratic programming for developing several locomotion controllers [6, 14, 15]. While model-based methods built upon physics can find a way around to tackle the disadvantages of the kinematics-based models, it can be difficult for them to build controllers for more contact-rich and dynamical motions. One way of tackling this problem using model-based methods is using trajectory optimization to synthesize physics coherent motions for various goals and characters [24, 45]. This methods consider an extended horizon for an offline optimization process while using the equations of motions from physics as constraints during that optimization process. These methods can also be extended into model-predictive control methods [9, 43].

1.1.3 Reinforcement Learning

Most of the optimization of the simulated characters are done based on reinforcement learning. As already mentioned, sequencing motion clips for kinematic controllers can be done using value iteration, a reinforcement learning method, for a given task [16, 17]. In the similar way. the optimization for physics-based models are done using different reinforcement learing methods [4, 27]. Kinematic and physics-based models both lack one important aspect in building controllers for locomotions–generalizability. Evolution of deep reinforcement learning has added generalizability in building models that can perform numerous challengin tasks [3, 7, 18, 28, 31, 44]. Besides, value iteration methods, policy gradients methods have emerged as to solve many control problems in continuous space [34, 41]. Even though many reinforcement learning algorithms have added generalizability in building controllers, the resulting behaviors from them was found to be less fluid and natural than manually described motions [21, 35]. Optimization through reinforcement learning requires defining a reward function which is used to

optimize an objective function. What's more, defining a reward for making the motions natural is difficult with the absence of biomechanical models and objectives [15, 46]. Simple objectives like maintaining a certain velocity or like going in a straight path can optimize the model with gaits and artifacts. To get rid of these gaits and artifacts some penalties can be given discourage these unwanted extraneous limb movements. Crafting such objective functions can be quite challenging and requires a lot of human insight. On the other hand, current development of reinforcement methods like GAIL [10] can circumvent this problem by updating a parameterized reward function and in turn induce an objective function that is based on the imitation of motion capture. Even though this has shown significant improvements in the generated locomotion behaviors, the quality is still not comparable to standard methods in computer animation [21].

1.1.4 Motion Imitation

Performing computer animation by imitation using reference motion started with the development of controller for a bipedal locomotion with planar characters [37, 39] using policy search method. Moreover, methods based on models for imitating motions in 3D humanoid characters have also been introduced [14, 25, 50]. Reference motions can also be used in shaping the reward function used for deep reinforcement learning to develop more fluid locomotion behavior without any extraneous movements of limbs [29, 30]. One of the significant work in this field is Sampling-based Controller (SAMCON) [20, 19]. It has displayed numerous numbers of very dynamic and complex motions with simulated characters. The resulting controller can imitate the actual reference motions but it lacks the possibility of extending it for different task objectives. One of the most recent variations of SAMCON [20, 19] used DQN to train a policy that selects already computed SAMCON [20, 19] fragments. This incorporates a significant amount of versatility in the order of processing of controller fragments.

CHAPTER II

THEORETICAL BACKGROUND

This chapter will introduce us with several theoretical concepts required to understand and implement the basics of reinforcement learning in a scientific setting.

2.1 Reinforcement Learning

2.1.1 Theoretical Background

In reinforcement learning, an agent learns to control a process such that a long-term performance criterion is optimized which is defined in terms of rewards. In most of the reinforcement learning algorithms, this optimization problem is assumed to be a Markov Decision Process (MDP) i.e the future depends only on the current state and action. It is defined by 5-tuple (S, A, P, R, γ). Figure 2.1 describes the agent environment interaction in a Markov decision process.

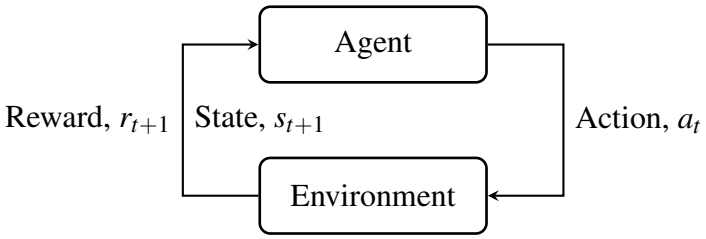


Figure 2.1: During a reinforcement learning process the agent and environment interact with each other at discrete time steps.

The goal of a RL agent is to maximize the cumulative reward it ultimately receives. If the sequence of rewards it received after time step t is denoted as $r_{t+1}, r_{t+2}, r_{t+3}, ...$, and so on, then we want to maximize the return G_t which can be defined as the sum of total rewards [41].

However, in order to control how much further an agent focuses in future from a state in terms of reward, we add a discounting factor (γ) within the range $0 \leq \gamma \leq 1$. We can see that the if $\gamma = 0$, then the agent emphasizes on the immediate reward. On the other hand, if it is 1, the agent considers the whole sequence of the rewards with same importance.

$$G_t = r_{t+1} + r_{t+2} + r_{t+3} + ... \tag{2.1}$$

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.2}$$

When the system model is available we either use dynamic programming methods (policy evaluation to calculate or approximate value/action functions for a policy, value iteration and policy iteration for finding an optimal policy) or resort to reinforcement learning. On the other hand, when there is no model, our only option is to use reinforcement learning methods.Right now, lets get familiar with some important terms frequently used in RL.

A state or action value function defines how good or bad a state or state-action pair is. The state value function is the expected return for following the policy $\pi(s|a)$ from state s.

$$v_\pi(s) = E_\pi[G_t|s_t = s] = \sum_{a \in A} q_\pi(s,a)\pi(a|s) \tag{2.3}$$

An optimal state value function,

$$v_*(s) = \max_\pi v_\pi(s) \tag{2.4}$$

which is the highest state value achievable by any policy for a certain state s which decomposes into the Bellman equation.

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) \sum_{s,s^{'} \in S} P(s^{'}|s,a)[r + \gamma v_\pi(s^{'})] \tag{2.5}$$

Moreover, the action value is the expected return for selecting an action a in state s while following the policy $\pi(s|a)$. On the other hand, an optimal action-value function can be obtained in the same way.

$$q_\pi(s,a) = E_\pi[G_t|s_t = s, a_t = a] \tag{2.6}$$

$$q_*(s,a) = \max_\pi q_\pi(s,a) \tag{2.7}$$

$$q_\pi(s,a) = \sum_{s,s^{'} \in S} P(s^{'}|s,a)[r + \gamma \sum_{a^{'} \in A} \pi(a^{'},s^{'}) q_\pi(a^{'},s^{'})] \tag{2.8}$$

Applying the optimal values into the Bellman equation, we can get the Bellman optimality backups.

$$v_*(s) = \max_a \sum_{s,s^{'} \in S} P(s^{'}|s,a)[r + \gamma v_*(s^{'})] \tag{2.9}$$

$$q_*(s,a) = \sum_{s,s^{'} \in S} P(s^{'}|s,a)[r + \gamma \max_{a^{'} \in A} q_*(a^{'},s^{'})] \tag{2.10}$$

2.2 Major Approaches and Classical Algorithms

2.2.1 Dynamic Programming

This is a method when the model is fully known i.e. dynamic programming assumes full knowledge of the transition and reward functions of the MDP. Following the Bellman equations, in dynamic programming, the policy iteration alternates between policy evaluation and policy improvement. In policy evaluation, the value function for the current policy is obtained by using Bellman expectation equation.

$$v_{n+1}(s) = \sum_{a \in A} \pi(a|s)[R(s,a) + \gamma \sum_{s,s^{'} \in S} P(s^{'}|s,a) v_n(s^{'})] \tag{2.11}$$

At each iteration $n+1$, for a state s, update $v_{n+1}(s)$ from value functions of its successor states $v_n(s^{'})$. The value function will converge to v_π for the current policy π. For policy improvement, a

better policy is generated $\pi' \geq \pi$ by taking action greedily with respect to the current value function v_π. Let's say we have a policy π, then we can generate an improved version π' by greedily taking action, $\pi'(s) = arg\max_{a \in A} q_\pi(s,a)$. This algorithm of iteratively evaluating and improving policy is called Generalized Policy Iteration (GPI).

2.2.2 Monte Carlo Methods

Monte Carlo methods learn directly from the raw experiences gathered from the episodic data. They do not need to model any environment dynamics (model free) but just computes the observed mean return as an approximation of the expected return. Therefore, Monte Carlo methods require to lo learn from the complete episodes $s_1, a_1, r_2, ..., s_T$ to compute the return Eq. 2.2. The empirical mean return for state s is:

$$v(s) = \frac{\sum_{t=1}^{T} 1[s_t = s] G_t}{\sum_{t=1}^{T} 1[s_t = s]} \tag{2.12}$$

Where $1[s_t = s]$ is a binary indicator function. This way of approximation can be easily extended to the action-value functions by utilizing (s,a) pair.

$$q(s,a) = \frac{\sum_{t=1}^{T} 1[s_t = s, a_t = a] G_t}{\sum_{t=1}^{T} 1[s_t = s, a_t = a]} \tag{2.13}$$

In order to learn optimal policy by Monte Carlo methods, we iterate the value functions by following the similar idea of GPI which are:

- Generating a new episode by following the policy π.
- Estimating the action-value function using the new episodes.
- Improving the policy greedily with respect to the current action-value function.

2.2.3 Temporal Difference Learning

Similar to Monte Carlo methods temporal difference learning is also a model free method. However, it can learn from incomplete episode which is different from Monte Carlo methods. It is one of the most important central ideas of reinforcement learning. It learns the state-value function $v(s)$

$$v(s) \leftarrow v(s) + \alpha[r + \gamma v(s^{'} - v(s))] \tag{2.14}$$

directly from the agent's experience with TD error and bootstrapping, in an online, model free way [40]. From Eq. 2.14, α is the learning rate and the term inside the square braces is the called TD error. This idea can also be extended towards action-value function $Q(s,a)$ which is called Q-learning [47].

$$q(s,a) \leftarrow q(s,a) + \alpha[r + \gamma \max_{a^{'} \in A} q(s^{'}, a^{'}) - q(s,a)] \tag{2.15}$$

Q-learning is an off-policy control method where the update is done greedily. On the other hand, the on-policy control alternative of Q-learning is called SARSA [32].

$$q(s,a) \leftarrow q(s,a) + \alpha[r + \gamma q(s^{'}, a^{'}) - q(s,a)] \tag{2.16}$$

In all of the above mentioned algorithms, the update was done using the data from the one step look ahead only. However, those update rules (Eq. 2.14-2.16) can incorporate n-steps by updating towards n-steps return defined as follows,

$$G_t \leftarrow r_t + \gamma r_{t+1} + ... + \gamma^{n-1} r_{t+n-1} + \gamma^n v(s_{t+n}) \tag{2.17}$$

2.2.4 Function Approximation

Previously mentioned methods have used a lookup table to store the value for the action-value function or state-value function. Those methods are not optimal when it comes to large

and continuous state-action spaces. Function approximation method approximates the value and action-values functions using the data acquired from the experience of an agent which is a concept similar to supervised learning. The most challenging part of this method is how we define the policy. Linear function approximation is a popular choice due to its easily applicable properties. However, deep neural network became the most popular choice after the work of deep Q-network [23] where authors utilized the universal approximation property of the deep neural network in their favor to approximate policy network. In this method the value functions become a function of both state s and the function parameter w i.e. state value function is defined a $v(s,w)$. Any optimization method can be used to get this parameter of the function. The most common optimization method is gradient descent. The update for the parameter of TD learning is given in the following formula where $\nabla v(s,w)$ is the gradient of the approximate value function with respect to the function parameter vector w and α is the update rate.

$$w \leftarrow w + \alpha[r + \gamma v(s^{'}, w) - v(s,w)]\nabla v(s,w) \tag{2.18}$$

This can be implemented by applying gradient descent on minimizing the mean squared error between the approximate value function $v(s,w)$ and the true value function $v(s)$.

2.2.5 Policy Gradient Methods

On the contrary to the value based methods like TD learning and Q-learning, policy based methods optimize the policy $\pi(a|s;\theta)$ directly utilizing the function approximation method with respect to the parameter vector theta. Policy based methods usually have better convergence properties and highly effective in high dimension or continuous state-action spaces. Moreover, we can also incorporate stochastic policies through policy gradient methods. However, this kind of methods generally converges to local optimum and have high variances.

If we define a policy objective function $J(\theta)$ for discrete and continuous state-action space respectively as the expected return and train the model with the aim to maximize $J(\theta)$,

$$J(\theta) = E_{\pi_\theta}[v] \tag{2.19}$$

$$J(\theta) = \sum_{s \in S} d_{\pi_\theta}(s) E_{\pi_\theta}[v] \tag{2.20}$$

we can follow the gradient ascent method to update the parameter θ to update the approximate policy function π_θ. Here, $E_{\pi_\theta}[v]$ for continuous space can be expanded as,

$$E_{\pi_\theta}[v] = \sum_{s \in S} (d_{\pi_\theta} \sum_{a \in A} \pi(a|s,\theta) q_\pi(s,a)) \tag{2.21}$$

According to the policy gradient theorem [42], the gradient of the reward function J_θ becomes(eq 18).

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \ln \pi(a|s;\theta) q_\pi(s,a)] \tag{2.22}$$

$$J^{clip}(\theta) = E_t[min(r_t(\theta)A_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t)] \tag{2.23}$$

Here $\nabla_\theta \ln \pi(a|s;\theta)$ is the score function or the likelihood ratio. In the similar way, by replacing $J(\theta)$ with $\pi(a|s;\theta)$, we can calculate the policy gradient analytically for a differentiable policy when it is not zero.

$$\nabla_\theta \pi(a|s;\theta) = \pi(a|s;\theta) \nabla_\theta \ln \pi(a|s;\theta) \tag{2.24}$$

Policy gradient method has different forms according to different values of $q_\pi(s,a)$.

- REINFORCE : $\nabla_\theta \ln \pi(a|s;\theta) G_t$
- Q actor-critic (AC): $\nabla_\theta \ln \pi(a|s;\theta) q(s,a,w)$

- Advantage AC: $\nabla_\theta \ln \pi(a|s;\theta)(q(a,s) - v(s))$
- TD AC: $\nabla_\theta \ln \pi(a|s;\theta)(r + \gamma v(s^{'}) - v(s)))$
- TD(λ) AC: $\nabla_\theta \ln \pi(a|s;\theta)(r + \gamma v(s^{'}) - v(s)))e$ where e is the eligibility trace which is used in the backward view TD online update.

If we subtract a learned function of state $b_t(s)$ which is also known as a baseline from $q(s,a)$ Eq. (2.6), the variances can be reduced while being unbiased in updating the gradients. A learned value function $v(s_t)$ is generally used a baseline. Therefore, the resultant gradient becomes $\nabla_\theta \ln \pi(a|s;\theta)(q_\pi(s,a) - v(s_t))$. In that gradient, the quantity $q_\pi(s,a) - v(s_t)$ is defined as the advantage which tells us how much better an action a would be when compared to the return acquired from an 'averaged' action (from $v(s_t)$). This approach of updating the gradient the policy gradient is called Advantage Actor(policy, π) Critic(baseline, b_t)(A2C) [22]. There are other policy optimization algorithms e.g. Trust Region Policy Optimization (TRPO) where the update rule ensures monotonic improvement of policy using a constraint on KL divergence [34] and Proximal Policy Optimization (PPO) in which a clipping function is used to restrain the update of the policy gradients to a certain ratio [36]. These methods ensure fast convergence while maintaining low variance in updating the policy.

CHAPTER III

CONTRIBUTION

Our first contribution on this thesis is to benchmark the performance of the humanoid AI agent on different state-of-the art reinforcement learning algorithm. In the previous study of building humanoid AI agents by human motion transfer, benchmarking the performances of those AI were neglected and proximal policy optimization (PPO) was used to measure the performance of those agents. For our benchmarking process we are going to use vanilla PPO [36], vanilla Advantage Actor Critic (A2C) [22], and A2C with generalized advantage estimation (GAE) [35] which is our own implementation of a modified version of A2C. This benchmark study will help us compare the performance of different famous reinforcement learning algorithms with regards to the convergence speed and reaching global optimal solutions through variance reduction and improved RL agent's performance.

3.1 Vanilla Advantage Actor Critic

As mentioned in the chapter II, the advantage for an actor critics can be calculated by subtracting a baseline $b_t(s)$ from the action value function $q(s,a)$. This advantage forces the AI agent to take an action a_t at state s_t which provides the maximum reward estimated from the state value function. For the vanilla A2C, this action value function appears to be the normally discounted reward whereas the baseline happens to be the parameterized value function $v(s_t,\theta)$ which is periodically updated using an objective function mentioned. Even though this subtraction of a learned function of state ensures a bit of reduction in variance, the extent of reduction is not as much as we observe from the addition of importance sampling into PPO.

3.2 Advantage Actor Critic with Generalized Advantage Estimation

Policy gradient method is a very straightforward method of directly optimizing the rewards while using a neural network to approximate nonlinear value functions or action-value functions. However, this approach requires a large number of samples to successfully go through a consistent and steady training process for a RL agent. This makes the the training time high and introduce large variance during training. One way of tackling this problem is to use an exponentially-weighted estimator of the advantage function similar to the temporal difference learning. This approach introduces some bias but dramatically reduce variance and improve the performance of a trained RL agent. This approach is called generalized advantage estimator (GAE) which have been previously used in policy estimates for PPO and TRPO. But it was never used with A2C. GAE is an estimation scheme parameterized by discount factor, γ and, weight decay factor λ.

GAE is concerned about producing an accurate estimate $\hat{A}_t$ of the discounted advantage function $A^{\pi,\gamma}(s_t, a_t)$ to construct a policy gradient estimator. This construction of $\hat{A}_t$ uses the calculation of TD residual of V, the approximate value function. Therefore, we can define the TD residual with discount γ as follows:

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{3.1}$$

This δ_t^V can be considered as an estimate of the advantage of action taken at time t, a_t. If we have the correct value function $V^{\pi,\gamma}$, i.e., $V = V^{\pi,\gamma}$, then δ_t^V is a λ-just unbiased estimator the advantage $V^{\pi,\gamma}$.

$$\begin{aligned} \mathbb{E}_{s_{t+1}}[\delta_t^{V^{\pi,\gamma}}] &= \mathbb{E}_{s_{t+1}}[r_t + \gamma V^{\pi,\gamma}(s_{t+1}) - V^{\pi,\gamma}(s_t)] \\ &= \mathbb{E}_{s_{t+1}}[Q^{\pi,\gamma}(s_t, a_t) - V^{\pi,\gamma}(s_t)] = A^{\pi,\gamma}(s_t, a_t) \end{aligned} \tag{3.2}$$

This estimator is only γ discounted for the correct value function $V^{\pi,\gamma}(s_t)$. Now, let's consider taking the sum of k of these TD residual terms, which we will denote by $\hat{A}_t^{(k)}$

$$\hat{A}_t^{(1)} = \delta_t^V = -V(s_t) + r_t + \gamma V(s_{t+1}) \tag{3.3}$$

$$\hat{A}_t^{(2)} = \delta_t^V + \gamma\delta_{t+1}^V = -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) \tag{3.4}$$

$$\hat{A}_t^{(3)} = \delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V = -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V(s_{t+3}) \tag{3.5}$$

$$\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V = -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k}) \tag{3.6}$$

Here we can see that $\hat{A}_t^{(k)}$ involves a k-step estimate of the returns, minus a baseline term $-V_{s_t}$. We can consider $\hat{A}_t^{(k)}$ as an estimator of the advantage function, which is only $\gamma-$just when $V = V^{\pi,\gamma}$. When $k \to \infty$, the bias becomes negligible as the term, $\gamma^k V(s_{t+k})$ becomes negligible at that case as it becomes more heavily discounted. As a result, we get the following expression of the advantage estimate which is simply the empirical returns minus the value function baseline.

$$\hat{A}_t^{(\infty)} = \sum_{l=0}^{\infty} \gamma^l \delta_{t+l}^V = -V(s_t) + \sum_{l=0}^{\infty} \gamma^l r_{t+l} \tag{3.7}$$

The generalized advantage estimator GAE (γ, λ) is defined as the exponentially-weighted average of these $k-$step estimators:

$$\begin{aligned} \hat{A}_t^{GAE(\gamma,\lambda)} &= (1-\lambda)(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + ...) \\ &= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V \end{aligned} \tag{3.8}$$

Equation 3.8 shows that the construction of the estimate of the advantage function is closely analogous to the TD(λ) definition from sutton and barto [41] where TD(λ) estimates value function.

3.3 Proximal Policy Optimization

The most significant contribution of PPO in terms of improving the learning process and performance of a RL agent, was introduced by incorporating a clipped surrogate objective. For PPO specifically the action value function $q(s,a)$ term of the policy gradient objective eq. 2.6 is replaced with probability ratio $r_t(\theta)$, a ratio between the new policy and the old policy for taking an action, multiplied with the advantage of that action.

Reinforcement learning algorithm proposed before PPO, such that A2C and TRPO, lacked the fact that when during an update of the policy π it would create a huge deviation from the previous policy, there was no constraint to avoid such kind of updates. PPO uses a clipping factor to clip the gradients according to the probability ratio $r_t(\theta)$. This ensures that when there is a large change in the policy update, the action value function $q(s,a)$ term of the policy gradient objective eq.2.6 does not have a value which slows down the training process of the RL agent and consistently updating the policy. As a result, this will help the agent reach its optimal solution faster. Without this constraints maximization process of the objective function eq.2.6, there can be an excessively large policy update which could deviate us from reaching the global optimal result.

The idea of using the probability ratio between the new policy and the old policy is not a original idea of PPO. It comes from the idea of importance sampling used in TRPO. Importance sampling is an statistical estimation method where using a known probability density function we can estimate an unknown probability density function. One of the important effects of using importance sampling is it reduces the variance in estimation. Thus, by using it in the PPO, we can also observe a narrower performance variation during the training process of a RL agent if compared with other algorithms like A2C. Therefore, we can observe, PPO converges the training process faster with lower about of variance.

3.3.1 Importance Sampling

Importance sampling is a statistical technique which we can use for estimating properties of a certain distribution using only the samples generated from a different distribution than the distribution we are interested about. In one way, we can say importance sampling is an approximation method instead of a sampling method. Let's consider a situation where we are trying to calculate the expectation of function $f(x)$ where we are sampling random variables x from the distribution $p(x)$. Then we have the following formula for estimating $f(x)$:

$$\mathbb{E}[f(x)] = \int f(x)p(x)dx \approx \frac{1}{n}\sum_i f(x_i) \tag{3.9}$$

Normal Monte Carlo sampling method is just sampling i random variable x's from the distribution $p(x)$ and taking the average of all those i samples to get an estimate of the expectation. However, what if the distribution $p(x)$ is very hard to sample? In many cases when we want to calculate the expectation $\mathscr{E}[f(x)]$ where $f(x)$ is almost zero outside region S for which $p(x)$ is very small. If we apply a normal Monte Carlo method in this case, samples from the distribution of random variable x might fail to have ample points inside region S.

In this situation importance sampling comes in handy. This comes from a simple transformation of the formula of expectation.

$$\mathbb{E}[f(x)] = \int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx \approx \frac{1}{n}\sum_i f(x_i)\frac{p(x_i)}{q(x_i)} \tag{3.10}$$

From eq.3.10, we can estimate the expectation $\mathbb{E}[f(x)]$ by sampling from the distribution $q(x)$ where $q(x)/p(x)$ is called the sampling weight, which is used as the correcting factor to offset the sampling from a different distribution $q(x)$.

In this way, importance sampling provides us a way with controlling the variance of the estimation as we have the known policy distribution in our control. We know the definition of the variance as follow,

$$Var(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2 \tag{3.11}$$

In eq.3.11, X is $f(x)p(x)/q(x)$. Therefore, by selecting a proper $q(x)$, the distribution to sample from, we can effective control the variance and eventually infer a distribution with low variance.

3.4 Reward Saltation

In reinforcement learning an agent obtain reward r_t at state s_t for taking an action a_t. In general, if we compare this reward r_t with the previous reward obtained r_{t-1} at the previous state s_{t-1} after taking action a_{t-1}, there can be a significant difference between the values of r_{t-1} and r_t, specially at the initial stage of the training process. This phenomenon is defined as reward saltation. When $r_t > r_{t-1}$, we say we have encountered a positive saltation. This situation happens when the agent finds a shortcut for reaching the goal. On the other hand, if $r_t < r_{t-1}$, the phenomenon is called negative saltation. Negative saltation occurs when the agent encounters an unexpected situation like facing a new obstacle in the road on which the agent is trying to walk. This idea has been implemented with various vesions of DQN [13].

Similar to the above situation, during the lifetime of a normal human being when he/she faces an unexpected situation that obstructs their success that he/she is aiming for, he/she tends to become very cautious at that stage where they have faced unexpected turns before. That person can even stay at that state out of fear because the cause of their response at that stage can be disastrous for their goal. As the goal of AI is to make machine more humanlike, we would like to transfer this idea of taking shortcuts when available to obtain very high rewards and be cautious when there is a potential danger in future steps. This idea can be implemented by magnifying the current reward, r_t , when $r_t > r_{t-1}$. On the other hand, when $r_t < r_{t-1}$, narrow down the current reward r_t.

This idea of saltation was never implemented with PPO specially with 3D locomotion tasks before. Shaping a reward function for a RL problem is difficult for different environments. However, this method we will use a parameter-adjustable optimization method for the reward function only. This method will not interfere with the interaction between the agent and the environment. What's more, we will not need to modify the already defined reward functions for human motion imitation [26]. Depending on the adjusted parameter, the current reward will be positively salted, negatively salted, or not salted, and will be stored within the replay buffer for constructing the policy gradient objective function.

3.4.1 Mathematical Formulation

The major concern to implement this idea of magnifying saltation is when,how, and to what extent we should magnify the reward. If the reward is amplified significantly, the RL agent might suffer from the low degree of exploration at those states. Therefore, the convergence process of the neural network will lead to local optimal position instead of a global one. On the other hand, if the extent of magnification is not enough, the experience of reward will not be realized by the RL agent. For this reason we will define a parameter ρ which is defined as below:

$$\rho = \frac{r_t}{r_{t-1}} \tag{3.12}$$

where r_t is the reward for taking the current action a_t, whereas r_{t-1} is the reward for taking the previous action a_{t-1}. When the value of ρ is greater than a certain parameter η, reward saltation parameter, the reward will be magnified. However, this way of controlling the magnification can cause two problems: i) if the current reward is zero, it will cause ρ to be meaningless and ii) when r_{t-1} is too low, the value of the current reward will have non significant effect on ρ. As a result, we can modify ρ as follows:

$$\rho = \frac{r_t^{'} - r_{t-1}^{'}}{min(r_t^{'}, r_{t-1}^{'})} \tag{3.13}$$

$$r_t^{'} = r_t + \sigma \tag{3.14}$$

$$r_{t-1}^{'} = r_{t-1} + \sigma \tag{3.15}$$

here, σ is a number that is close to zero. This provides the solution for the question of 'when' to perform reward saltation. The next question is to 'how' and 'what' extent we should magnify the current reward.

For this, we are going to use a monotonically increasing bounded function arc *tan* function, i.e. $f(x). = arctan(x)$, which ranges from $-\pi/2$ to $\pi/2$. Therefore, when $\rho > \eta$ we wish to magnify the current reward r_t such that when $\rho = \eta$, $f(\rho) = 1$ and when $\rho > \eta$, $f(\rho) > 1$. With the introduction of a sign function to incorporate both positive and negative saltation formula, we obtain the final version of $f(x)$ as follows:

$$f(x) = arctan(x * \frac{\pi}{2} * \frac{1}{\eta}) \tag{3.16}$$

where

$$x = \rho + \lambda \tag{3.17}$$

$$\lambda = sgn(r_t - r_{t-1}) \tag{3.18}$$

Finally, The magnified salted reward, r_t^* will be

$$r_t^* = r_t^{'} + (f(x) - \lambda) * |r_t^{'}| \tag{3.19}$$

CHAPTER IV

EXPERIMENTAL SETUP

The policies for our desired motions will be optimized using both proximal policy optimization (PPO) [36] and advantage actor critic (A2C) [22] method. What's more, there will be two variations for both of these methods. PPO will have a modification of reward saltation to compare its effect with the vanilla PPO. In both of those cases, the policy gradient advantages will be calculated using (GAE) [35]. On the other hand, one version of A2C will compute the advantages with GAE to compare with the vanilla version of A2C. All of these four versions will train the value function in the same way, that is by using multi-step temporal difference learning.

The experiments use a humanoid model (Fig. 4.1) built by using the PyBullet physics engine. Links comprising the humanoid model is described in Tab. 4.1 whereas the joints connecting those links and their types are described in Tab. 4.2. All the locomotions in consideration for experiments have state and action dimension of 197 and 36 respectively Tab. 4.3.

The deep mimic pipeline receives a set of kinematic reference motions (demonstration) along with the character model as inputs. It then uses reinforcement learning to optimize a controller that will imitate the reference motion as well as task objectives by optimizing the rewards which include imitation objective terms [26].

Each input reference motion is comprised of a set a of target poses $\{\hat{q}_t\}$. The controller can be defined a policy $\pi(a_t|s_t, g_t)$ where it provides action a_t when the character model is in a state s_t.

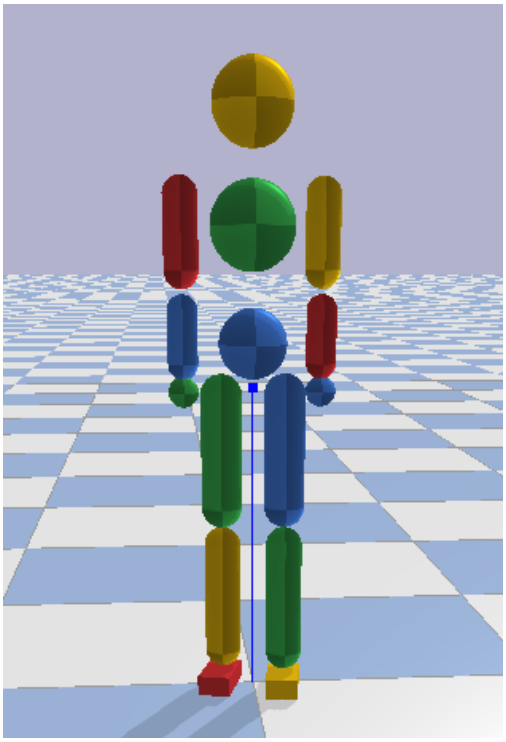


Figure 4.1: Humanoid model built using the PyBullet physics engine.

Table 4.1: Links and corresponding indices of the humanoid model used for the experiments.

Index	Link Name	Index	Link Name
-1	Base	7	Right elbow
0	Root	8	Right hand
1	Chest	9	Left hip
2	Neck	10	Left knee
3	Right hip	11	Left foot
4	Right knee	12	Left shoulder
5	Right foot	13	Left elbow
6	Right shoulder	14	Left hand

Table 4.2: Joints and corresponding joint types of the humanoid model used for the experiments.

Joint Name	Joint Type	Joint Name	Joint Type
Root	Fixed	Right hand	Fixed
Chest	Spherical	Left hip	Spherical
Neck	Spherical	Left knee	Revolute
Right hip	Spherical	Left foot	Spherical
Right knee	Revolute	Left shoulder	Spherical
Right foot	Spherical	Left elbow	Revolute
Right shoulder	Spherical	Left hand	Fixed
Right elbow	Revolute		

Table 4.3: State-action dimension of the motions in consideriation along with the indices of allowed links of the humanoid model to contact the floor.

Motion	State Dimension	Action Dimension	Fall Contact Links
Walk	197	36	5 and 11
Run	197	36	5 and 11
Backflip	197	36	5 and 11
Cartwheel	197	36	5, 8, 11 and 14
Spinkick	197	36	5 and 11
Spin	197	36	5 and 11

State s_t depicts the configuration/features of the character model at time t. This configuration includes the relative position, rotations (quaternions), linear and angular velocities of the links. What's more there is a phase variable ϕ which becomes 1 when the link reaches the target pose while it is 0 at the beginning of the motion. All of these state features are measured with respect the pelvis (root) of the character model.

This action a_t specifies the target angles/orientations (spherical joints: axis-angle form and revolute joints: scalar rotation form) of the proportional-derivative (PD) controller which in turn computes the torques to be applied in the character's link joints.

This controller policy is defined as a neural network where the network parameters are updated by maximizing the rewards defined in terms of imitation and task-specific objective. The updating process follows a state-of-the-art reinforcement learning algorithm PPO[36]. As

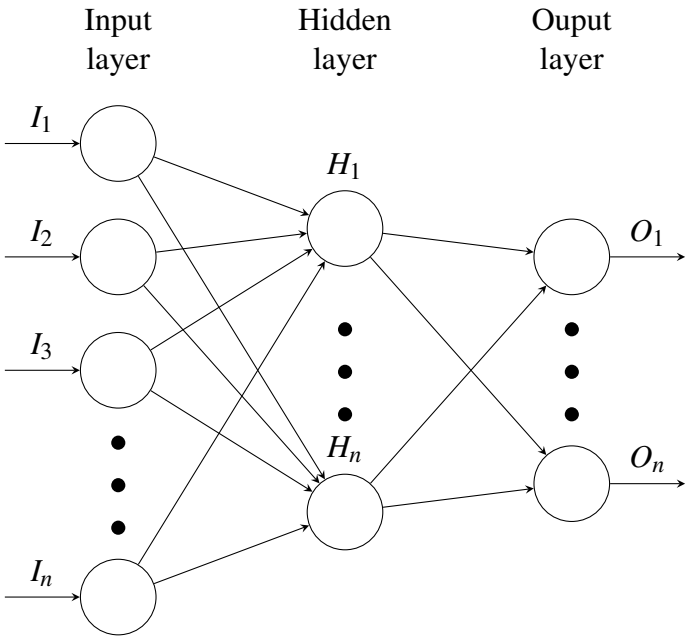


Figure 4.2: Neural network structure for the policy network.

already mentioned in a generalized form of an RL objective function eq. (2.19), the ppo agent's goal is to learn the optimized θ's that maximized the expected return defined by the Bellman equation. PPO uses policy gradient method where policy is updated using the gradient of the objective function eq. (2.22), where $q_\pi(s,a)$ is replaced with $R_t - v(s_t)$. Here, R_t denotes the return acquired by a trajectory when starting from state s_t at time t and $v(s_t)$ is the value function which estimate of average return when the agent starts from the state s_t and taking the subsequent steps by following policy π_θ. For the policy network (Fig. 4.2) output action is modeled as a Gaussian (continuous, $\mu(s)$) where the input features are processed with two fully connected hidden layer with 1024 and 512 nodes respectively. All of the hidden layer nodes have ReLU activation function while the output layer has linear activation function.

This value function $v(s_t)$ will be trained using TD(λ) while the advantage($q_\pi(s,a)$ of (2.22) will be calculated using both GAE(λ) [35] and from the difference of a learned function from a baseline. Value function estimation process uses a network (Fig.4.3) similar to the policy network but with only a single linear output unit.

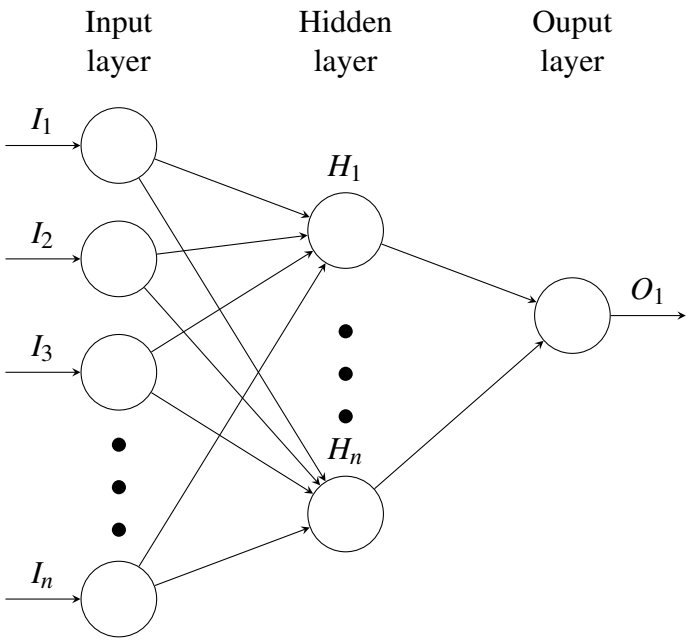


Figure 4.3: Neural network structure for the value network.

The overall process of reinforcement learning for updating the policy network objective Eq. (2.22) requires calculation of the total return. On the other hand, the return is calculated on the rewards an agent gets from taking a certain step. As the agent is forced to imitate reference motions, the reward is defined in such a way that the update of the policy network happens in such a way that it forces the agent to mimic the reference motion. This imitation reward, r_t is divided into four parts:

- Pose reward, r_t^p: This motivates the agent model to the match joint orientation of the reference motions. This reward is calculated from difference between the jth joint quaternions of the simulated agent model q_t and those of the demonstrated motion ($\hat{q}_t$).

$$r_t^p = exp\left[-2\left(\sum_j ||\hat{q}_t{}^j \ominus q_t^j||^2\right)\right] \tag{4.1}$$

- Velocity reward, r_t^v: This reward is calculated from the difference between the angular velocities of the jth joint of the simulated agent character ($\dot{q_t}^j$) and the ones ($\hat{\dot{q}}_t^j$) calculated by finite difference method from the reference motion data.

$$r_t^v = exp\left[-0.1\left(\sum_j ||\hat{\dot{q}}_t^j - \dot{q_t}^j||^2\right)\right] \tag{4.2}$$

- End-effector reward, r_t^e: This forces the agent character end-effectors (p_t^e) to match with the ones of the reference motion ($\hat{p_t}^e$). Left foot, right foot, left hand, and right hand are the end-effectors of the character model.

$$r_t^e = exp\left[-40\left(\sum_j ||\hat{p_t}^e - p_t^e||^2\right)\right] \tag{4.3}$$

- Center-of-mass reward, r_t^c: This punishes the agent if its center-of-mass (p_t^c) deviates from the reference motion ($\hat{p_t}^c$).

$$r_t^c = exp\left[-10\left(\sum_j ||\hat{p_t}^c - p_t^c||^2\right)\right] \tag{4.4}$$

These r_t^p, r_t^v, r_t^e, and r_c^p together comprise the total reward at time t, i.e., r_t. However, as not all of the rewards have the same effects on the imitation tasks, we are going to use weights for each of those separate rewards. Therefore, the total rewards for our imitation objective is defined as follows:

$$r_t = w^p r_t^p + w^v r_t^v + w^e r_t^e + w^c r_t^c \tag{4.5}$$

where w_p, w_v, w_e, and w_c have values 0.65, 0.1, 0.15, and 0.1 respectively. Higher weight for the pose reward r_t^p indicates that we are prioritizing on the humanoid model to match the reference motion orientation. Maginification on the saltatory reward in our proposed method will occur at time t, r_t, if it satisfies the given condition when compared with the reward at time $t-1$, r_{t-1}.

4.1 Training

As we have already mentioned, our policies will be trained using both PPO and A2C. In both of those cases, there will be two networks: one for the policy function $\pi_\theta(a|s)$ and the other one is for value function $v_\psi(s)$ where we update the parameters θ and ψ periodically. The overall training happens episodically where the the initial state s_0 is chosen uniformly in a random manner from the reference motion. On the other hand, the rollouts of the trajectory are acquired from sampling the actions using the policy $\pi_\theta(a|s)$ at each step. Each of the episodes continues until the agent reaches a certain time horizon or until it reaches the termination conditions.

In our experiments, all of these are done in parallel using multiprocessing method. In this multiprocessing method, we have used eight separate RL agents to run in eight different threads of our processor to generate ample amount of rollouts to form a batch of data. From that batch of data, minibatches are samples from the dataset to update the policy $\pi_\theta(a|s)$ and value function $v_\psi(s)$. In this way, we can generate a lot of data from rollouts which leads the networks converge quicker than the case when we run try to train our agent with single thread.The value function $v_\psi(s)$ is updated by target values calculated from TD(λ) using mean square error. On the other hand, the policy function $\pi_\theta(a|s)$ is updated from the policy gradient objective function with advantage calculated from both normally and using GAE.

Detailed pseudocodes of our implemented algorithms used for the experiments with custom RL environment are depicted below:

Algorithm 1: Vanilla Proximal Policy Optimization (PPO)

$\theta \leftarrow$ random weights
$\psi \leftarrow$ random weights
while *not done* **do**
    $s_0 \leftarrow$ sample initial state from reference motion
    Initialize character to s_0
    for *step = 1, ..., m* **do**
        $s \leftarrow$ start state
        $a \sim \pi_\theta(a|s)$
        Apply a and simulate forward one step
        $s^{'} \leftarrow$ end state
        $r \leftarrow$ reward
        store $(s, a, r, s^{'})$ into memory D
    end for
    $\theta_{old} \leftarrow \theta$
    for *each update step* **do**
        Sample from minibatch of n samples $(s, a, r, s^{'})$ from memory D
        `// Update value function`
        for *each* $(s_i, a_i, r_i, s_i^{'})$ **do**
            $y_i \leftarrow$ compute target values using TD(λ)
        end for
        $\psi \leftarrow \psi + \alpha_v(\frac{1}{n}\sum_i \nabla_\psi V_\psi(s_i)(y_i - V(s_i)))$
        `// Update policy function`
        for *each* $(s_i, a_i, r_i, s_i^{'})$ **do**
            $A_i \leftarrow$ compute advantage using V_ψ and GAE
            $w_i(\theta) \leftarrow \frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_{old}}(a_i|s_i)}$
        end for
        $\theta \leftarrow \theta + \alpha_\pi(\frac{1}{n}\sum_i \nabla_\theta min(w_i(\theta)A_i, clip(w_i(\theta), 1-\varepsilon, 1+\varepsilon))A_i)$
    end for
end while

Algorithm 2: Proximal Policy Optimization with Magnify Saltatory Reward (PPOMSR)

$\theta \leftarrow$ random weights
$\psi \leftarrow$ random weights
while *not done* **do**
    $s_0 \leftarrow$ sample initial state from reference motion
    Initialize character to s_0
    for *step = 1, ..., m* **do**
        $s_t \leftarrow$ start state
        $a_t \sim \pi_\theta(a|s)$
        Apply a_t and simulate forward one step
        $s_t^{'} \leftarrow$ end state
        $r_t \leftarrow$ reward
        $r_t^{'} \leftarrow r_t + \sigma$
        $\rho \leftarrow \frac{r_t^{'} - r_{t-1}^{'}}{min(|r_t^{'}|, |r_{t-1}^{'}|)}$
        $x \leftarrow \rho + \lambda$
        `// Checking saltation`
        if $x > \eta$ ***or*** $x < -\eta$ **then**
            $r_t^* \leftarrow r_t^{'} + (arctan(x * \frac{\pi}{2} * \frac{1}{\eta}) - \lambda) * |r_t^{'}|$
        else
            $r_t^* \leftarrow r_t$
        end if
        store $(s_t, a_t, r_t^*, s_t^{'})$ into memory D
    end for
    $\theta_{old} \leftarrow \theta$
    for *each update step* **do**
        Sample from minibatch of n samples $(s, a, r, s^{'})$ from memory D
        `// Update value function`
        for *each* $(s_i, a_i, r_i^*, s_i^{'})$ **do**
            $y_i \leftarrow$ compute target values using TD(λ)
        end for
        $\psi \leftarrow \psi + \alpha_v(\frac{1}{n}\sum_i \nabla_\psi V_\psi(s_i)(y_i - V(s_i)))$
        `// Update policy function`
        for *each* $(s_i, a_i, r_i, s_i^{'})$ **do**
            $A_i \leftarrow$ compute advantage using V_ψ and GAE
            $w_i(\theta) \leftarrow \frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_{old}}(a_i|s_i)}$
        end for
        $\theta \leftarrow \theta + \alpha_\pi(\frac{1}{n}\sum_i \nabla_\theta min(w_i(\theta)A_i, clip(w_i(\theta), 1-\varepsilon, 1+\varepsilon))A_i)$
    end for
end while

Algorithm 3: Advantage Actor Critic (A2C)

$\theta \leftarrow$ random weights
$\psi \leftarrow$ random weights
while *not done* **do**
    $s_0 \leftarrow$ sample initial state from reference motion
    Initialize character to s_0
    for *step = 1, ..., m* **do**
        $s \leftarrow$ start state
        $a \sim \pi_\theta(a|s)$
        Apply a and simulate forward one step
        $s^{'} \leftarrow$ end state
        $r \leftarrow$ reward
        store $(s, a, r, s^{'})$ into memory D
    end for
    $R \leftarrow \begin{cases} 0, & \text{if } s \text{ is a terminal state} \\ V_\psi(s), & \text{otherwise} \end{cases}$
    for *each update step* **do**
        Sample from minibatch of n samples $(s, a, r, s^{'})$ from memory D
        `// Update value and policy function`
        for *each* $(s_i, a_i, r_i, s_i^{'})$ **do**
            $R_i \leftarrow r_i + \gamma R_{i-1}$
            $A_i \leftarrow R_i - V(s_i)$
        end for
        $\psi \leftarrow \psi + \alpha_v(\frac{1}{n}\sum_i \nabla_\psi V_\psi(s_i)(R_i - V(s_i)))$
        $\theta \leftarrow \theta + \alpha_\pi(\frac{1}{n}\sum_i \nabla_\theta ln_\pi(a_i|s_i)A_i)$
    end for
end while

Algorithm 4: Advantage Actor Critic with Generalized Advantage Estimation

$\theta \leftarrow$ random weights
$\psi \leftarrow$ random weights
while *not done* **do**
    $s_0 \leftarrow$ sample initial state from reference motion
    Initialize character to s_0
    for *step = 1, ..., m* **do**
        $s \leftarrow$ start state
        $a \sim \pi_\theta(a|s)$
        Apply a and simulate forward one step
        $s^{'} \leftarrow$ end state
        $r \leftarrow$ reward
        store $(s, a, r, s^{'})$ into memory D
    end for
    $R \leftarrow \begin{cases} 0, & \text{if } s \text{ is a terminal state} \\ V_\psi(s), & \text{otherwise} \end{cases}$
    for *each update step* **do**
        Sample from minibatch of n samples $(s, a, r, s^{'})$ from memory D
        `// Update value and policy function`
        for *each* $(s_i, a_i, r_i, s_i^{'})$ **do**
            $R_i \leftarrow r_i + \gamma R_{i-1}$
            $A_i \leftarrow$ compute advantage using V_ψ and GAE
        end for
        $\psi \leftarrow \psi + \alpha_v(\frac{1}{n}\sum_i \nabla_\psi V_\psi(s_i)(R_i - V(s_i)))$
        $\theta \leftarrow \theta + \alpha_\pi(\frac{1}{n}\sum_i \nabla_\theta ln_\pi(a_i|s_i)A_i)$
    end for
end while

CHAPTER V

RESULTS AND DISCUSSION

This chapter demonstrates the effects of generalized advantage estimation (GAE) and magnifying salted rewards on the performance and training convergence speed of RL agents. What's more, it also compares the performance of the two state of the art RL algorithms PPO and A2C [36, 22] with respect to those parameters as well. Snapshots of all the six locomotions are available from Figs. 5.12 to 5.17. Theses simulated characters are trained using vanilla PPO and the interface is from PyBullet physics engine.

5.1 Effects of Generalized Advantage Estimation

We can easily observe the performance difference due to the presence of GAE from Figs. 5.1-5.6. Those figures depict the comparison among PPO, A2C, and A2C with GAE with respect to the average return we calculated during the training of the RL agents for six different motions: i) backflip, ii) cartwheel, iii) spinkick, iv) spin, v) walk, and vi) run. These figures help us come to a conclusion that except for the running motion, PPO is the most superior algorithm by a good margin. In those except running, the RL agent learned the motions not only quicker but also obtained higher average returns using the vanilla PPO algorithm. Table 5.1 displays the sample required on the scale of 10^8 to converge the training for an agent. From Tab. 5.1 we can say training of a RL agent to learn backflip converges 1.5 times faster (lesser amount of samples) if compared with vanilla A2C. This amount is comparatively huge as we are calculating on the scale of 10^8 and thus saving significant amount of computational time and power. On the other hand, RL agents, using vanilla PPO, learn all other motions quicker as well if compared with

vanilla A2C. Similarly, if we consider the average returns of the last hundred episodes during training (Tab. 5.3), PPO attains significantly higher average return for learning all of the motions in concern when compared with A2C.

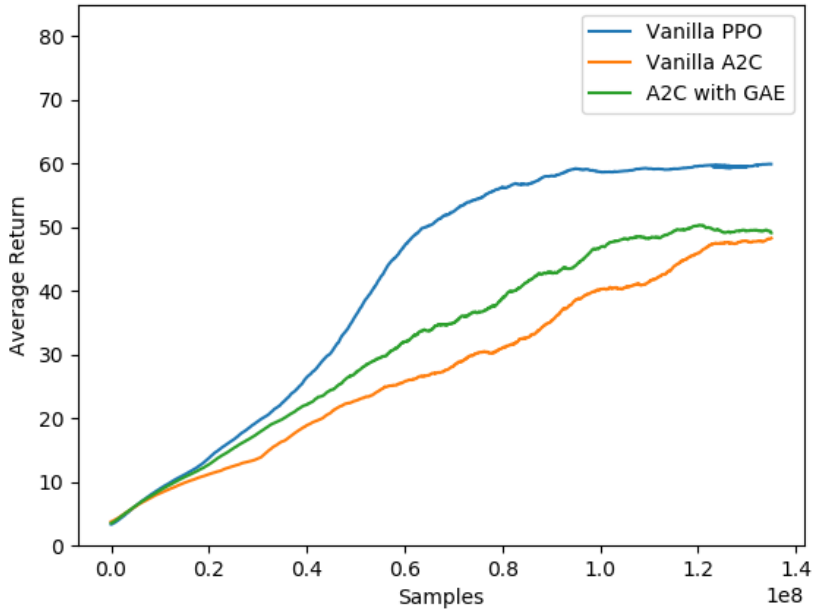


Figure 5.1: Comparison of the performance of a DRL agent of Backflip motion for Vanilla PPO, Vanilla A2C, and A2C with GAE.

As vanilla PPO uses an objective function with constraints where the gradients are clipped, the optimization does not use the gradient when there is an excessively large policy update which can deviate the optimization from the global maximum. This results in a stable and quicker convergence of the training of a RL agent. What's more, as GAE provides an estimate of the advantage by considering future advantages, that helps in deciding which action is the most advantageous one while keeping possible future actions in consideration at a certain state thus reducing the variances. This results in a significant overall gain in average returns as well as faster convergence for the training.

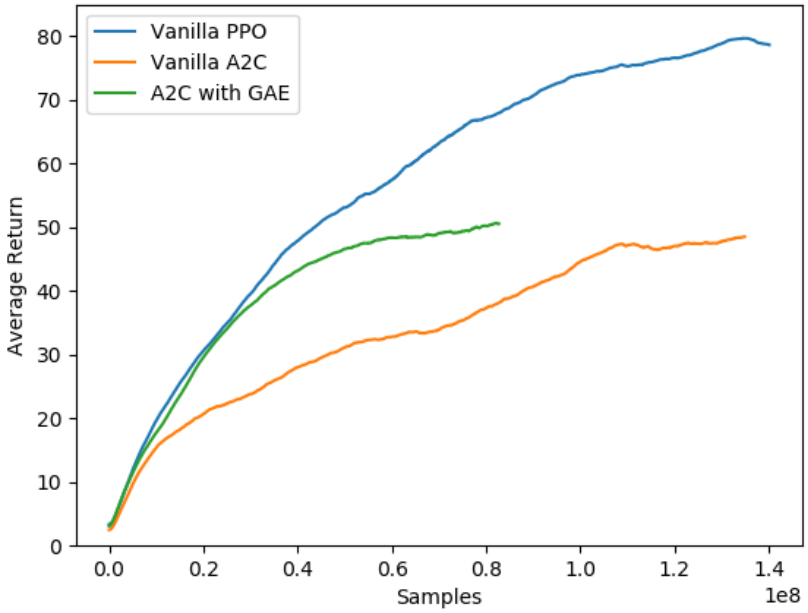


Figure 5.2: Comparison of the performance of a DRL agent of Cartwheel motion for Vanilla PPO, Vanilla A2C, and A2C with GAE.

On the other hand, after the inclusion of GAE with vanilla A2C, modified A2C appears to be performing well regarding average returns and convergence speed in comparison with the vanilla A2C. However, the performance of the modified A2C could not reach to the level of vanilla PPO in regards to learning the locomotions from references, Figs. 5.1 to 5.5, except learning running motion where the modified A2C matches level of performance of vanilla PPO displayed in Fig. 5.6. From Tab. 5.1 we can observe that the convergence speed of the training of a RL agent using the modified A2C for learning backflip motion is 1.2 times faster than vanilla A2C whereas it is a little bit slower than the agent using vanilla PPO. Similarly, while learning the motions from references, agents using A2C with GAE outperforms the agents using vanilla A2C (Tab. 5.3) with regards to the average returns of the last hundred episodes.

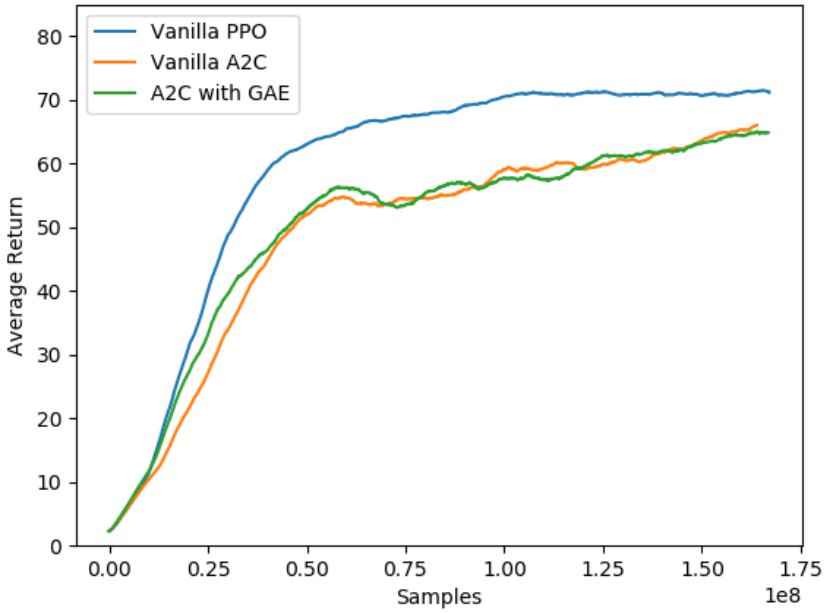


Figure 5.3: Comparison of the performance of a DRL agent of Spinkick motion for Vanilla PPO, Vanilla A2C, and A2C with GAE.

5.2 Effects of Magnifying Salted Rewards

Figures 5.7 to 5.8 display the effect of our proposed modified PPO with the presence of a parameter based reward function optimization process. Figures 5.7 to 5.8 compares the performance of the RL agent during training for different values of η along with the vanilla PPO. It is clear from Fig. 5.7 that the RL agent using PPOMSR with $\eta = 2$ has obtained the best result for learning backflip. From the values of Tab. 5.2, we can say, for all the values of η the training process converges faster than the vanilla PPO but in a decreasing order with the increment of η. However, that is different for cartwheel where only $\eta = 2$ obtains better convergence speed for the RL agent and other values of η performs a little worse than vanilla PPO.

For backflip motion, similar to the convergence speed, the average returns of PPOMSR (Tab. 5.4) decreases with the increment of η. For PPOMSR with $\eta = 2$ obtains a little higher average return for the last hundred episode than the vanilla PPO. On the contrary, for cartwheel,

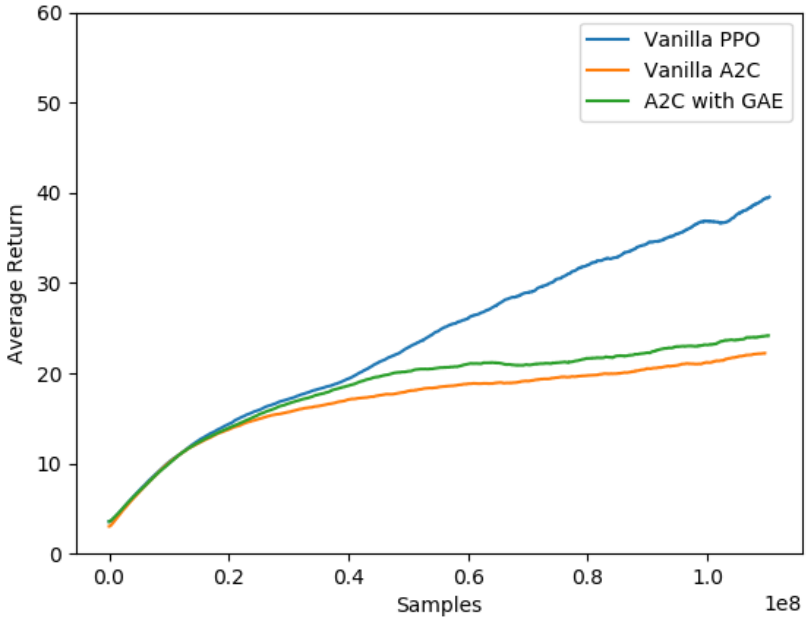


Figure 5.4: Comparison of the performance of a DRL agent of Spin motion for Vanilla PPO, Vanilla A2C, and A2C with GAE.

PPOMSR could not reach to a higher average return for any values of η than the vanilla PPO.

Snapshots of the humanoid at different time sequence for all the six motions are depicted from Fig. 5.12 to 5.17.

Table 5.1: Required sample size ($x10^8$) for completing the training the DRL agent with Vanilla PPO, Vanilla A2C, and A2C with GAE.

Motion	Vanilla PPO	Vanilla A2C	A2C with GAE
Backflip	0.90	1.40	1.18
Cartwheel	1.30	1.39	0.80
Spinkick	0.37	0.54	0.52
Spin	0.39	0.85	0.61
Run	0.53	0.62	0.57
Walk	0.61	0.63	0.45

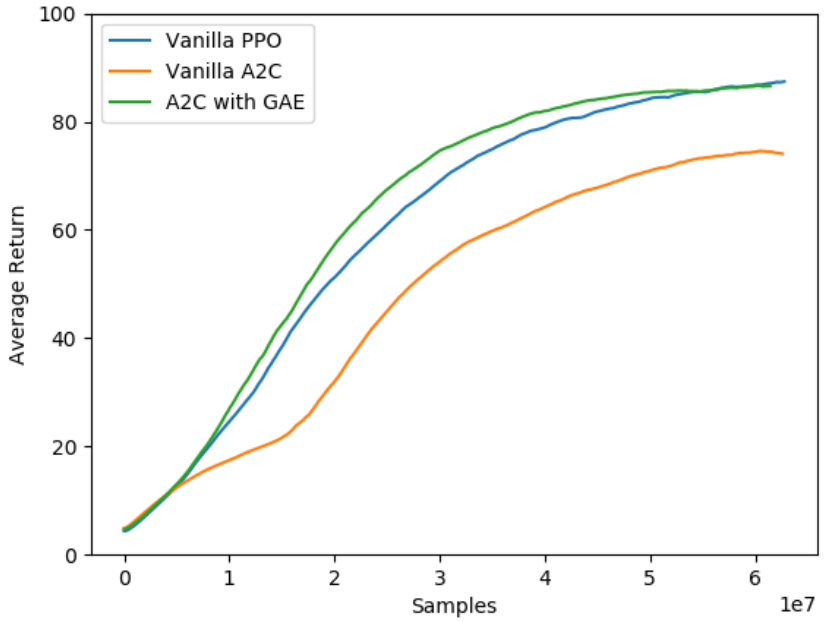


Figure 5.5: Comparison of the performance of a DRL agent of Walk motion for Vanilla PPO, Vanilla A2C, and A2C with GAE.

Table 5.2: Required sample size ($x10^8$) for completing the training the DRL agent with Vanilla PPO and PPOMSR.

Motion	Vanilla PPO	PPOMSR $\eta = 2$	PPOMSR $\eta = 3$	PPOMSR $\eta = 4$
Backflip	0.9	0.76	0.71	0.82
Cartwheel	1.30	1.20	1.39	1.39
Spinkick	0.37	0.55	0.70	
Spin	0.39	0.28	0.40	
Run	0.53	0.48	0.50	
Walk	0.61			

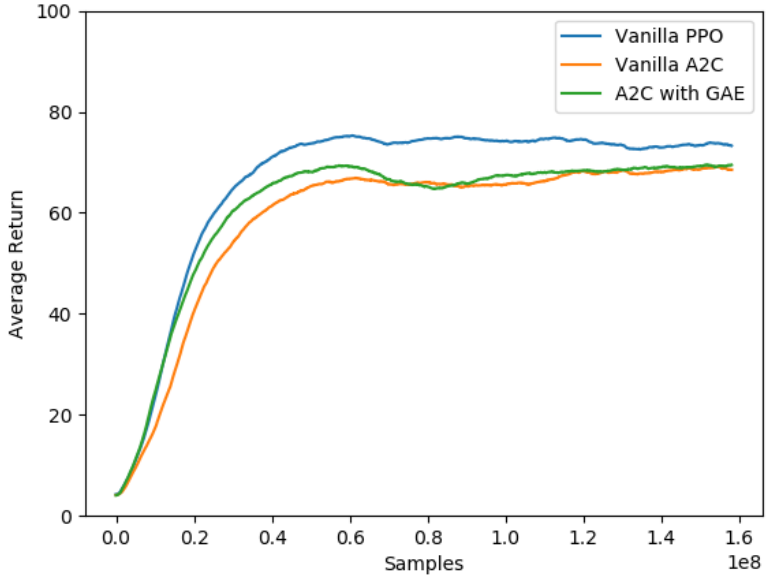


Figure 5.6: Comparison of the performance of a DRL agent of Run motion for Vanilla PPO, Vanilla A2C, and A2C with GAE.

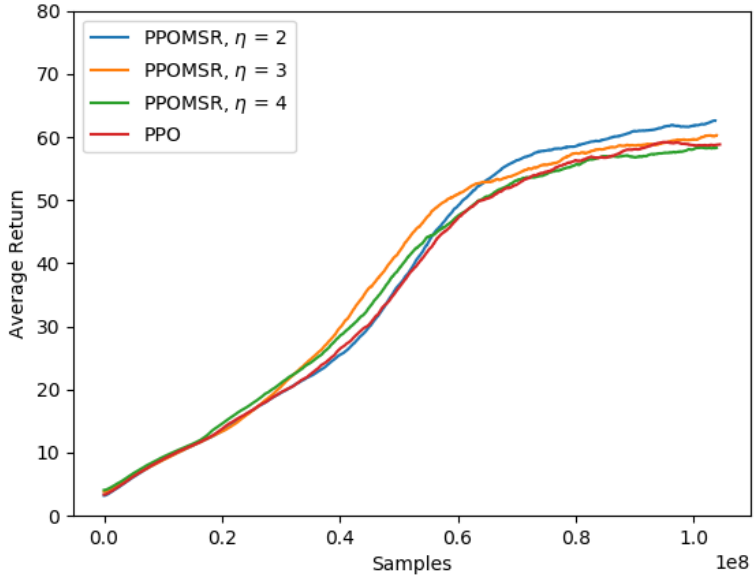


Figure 5.7: Comparison of the performance of a DRL agent of Backflip motion for Vanilla PPO and magnified saltatory rewarded PPO with $\eta = 2$, 3, and 4.

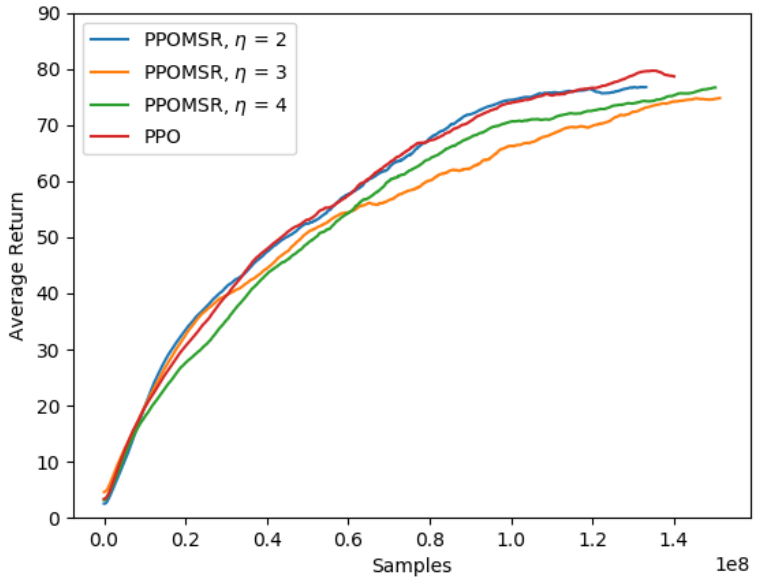


Figure 5.8: Comparison of the performance of a DRL agent of Cartwheel motion for Vanilla PPO and magnified saltatory rewarded PPO with $\eta = 2$, 3, and 4.

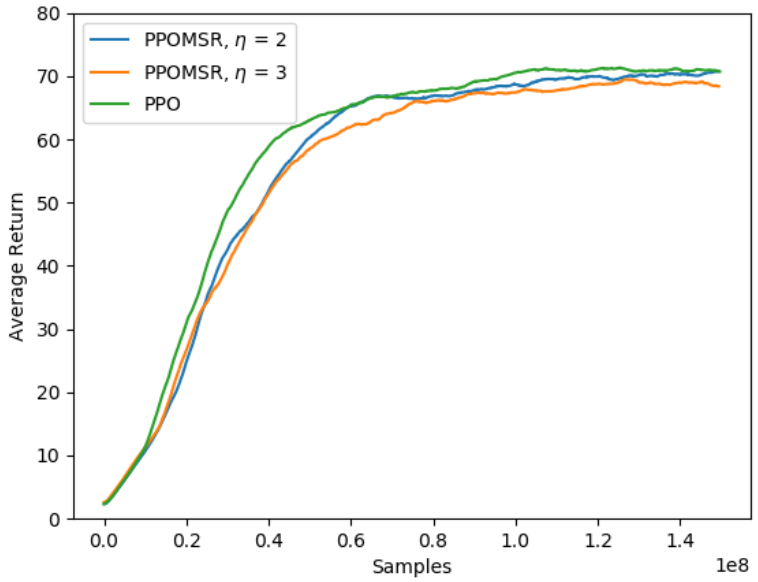


Figure 5.9: Comparison of the performance of a DRL agent of Spinkick motion for Vanilla PPO and magnified saltatory rewarded PPO with $\eta = 2$, 3, and 4.

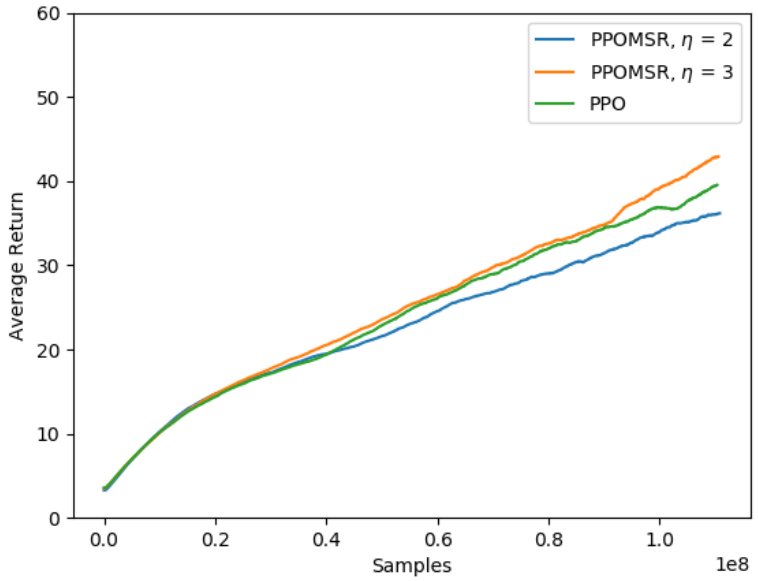


Figure 5.10: Comparison of the performance of a DRL agent of Spin motion for Vanilla PPO and magnified saltatory rewarded PPO with $\eta = 2$, 3, and 4.

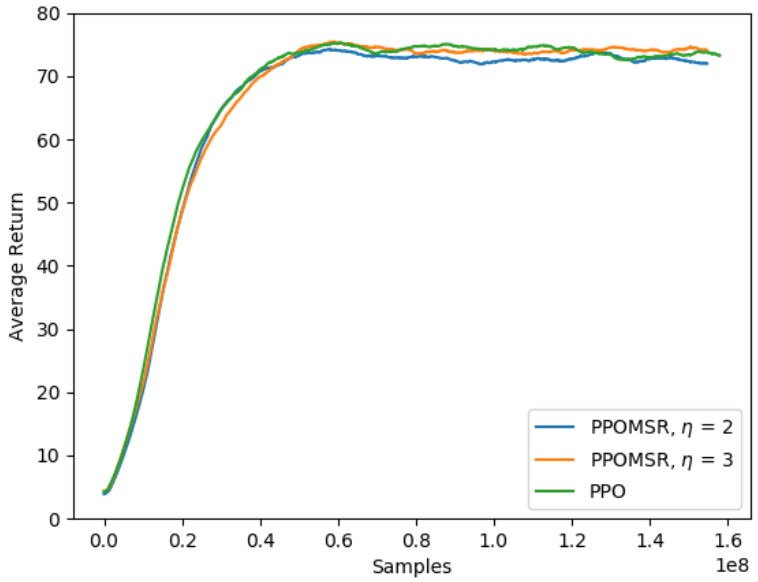


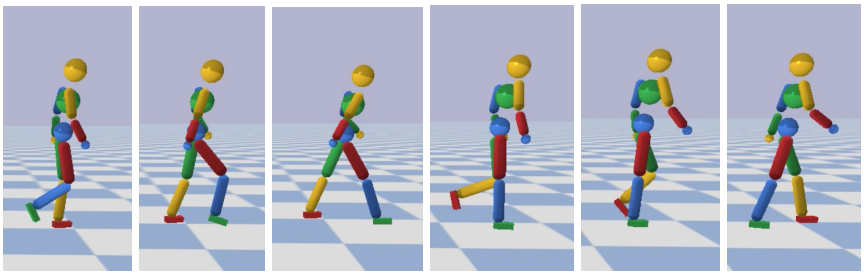
Figure 5.11: Comparison of the performance of a DRL agent of Run motion for Vanilla PPO and magnified saltatory rewarded PPO with $\eta = 2$, 3, and 4.

Table 5.3: Average returns of the DRL agent during training for last 100 episodes for Vanilla PPO, Vanilla A2C, and A2C with GAE

Motion	Vanilla PPO	Vanilla A2C	A2C with GAE
Backflip	76.651	46.232	47.839
Cartwheel	58.134	43.477	48.292
Spinkick	70.983	62.315	62.726
Spin	33.995	20.438	22.438
Run	76.281	61.838	78.316
Walk	73.490	68.247	68.777

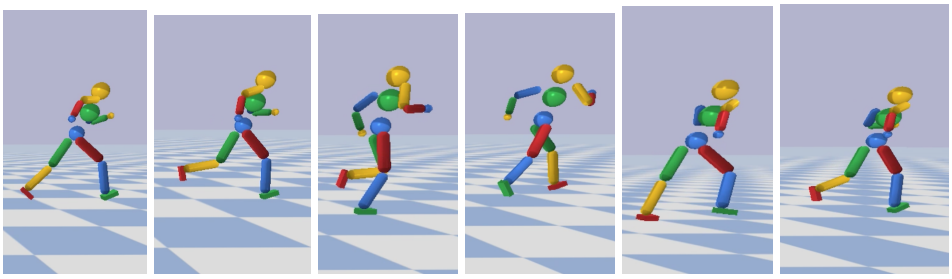
Table 5.4: Average returns of the DRL agent during training for last 100 episodes for Vanilla PPO and PPOMSR.

Motion	Vanilla PPO	PPOMSR $\eta = 2$	PPOMSR $\eta = 3$	PPOMSR $\eta = 4$
Backflip	58.134	58.562	56.934	55.244
Cartwheel	76.651	75.262	71.976	73.736
Spinkick	70.983	69.965	68.634	
Spin	33.995	31.477	35.573	
Run	76.281	72.759	74.118	
Walk	73.490			



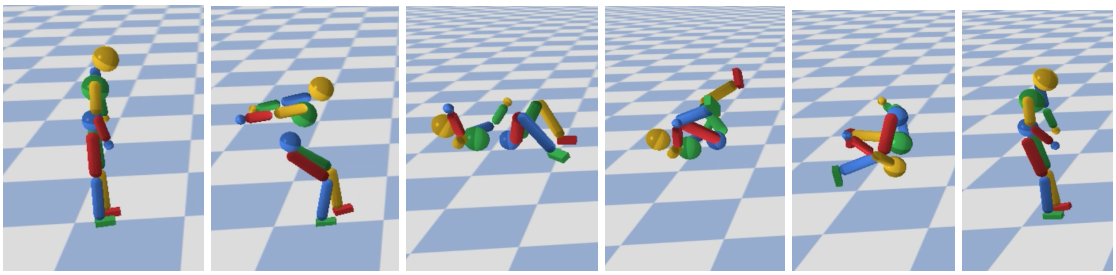
(a) (b) (c) (d) (e) (f)

Figure 5.12: Snapshots of walking motion from the trained policies.



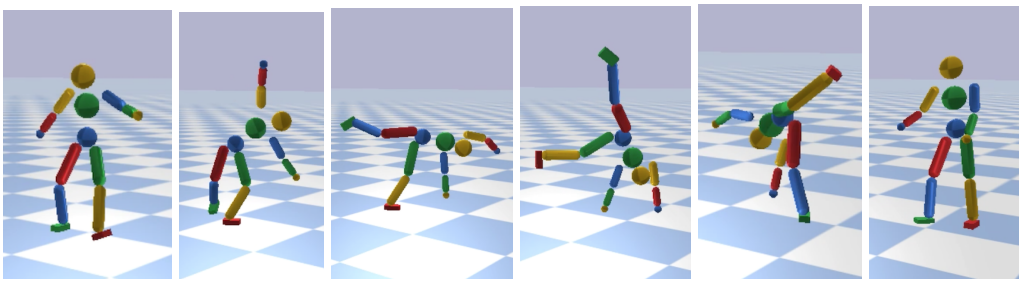
(a) (b) (c) (d) (e) (f)

Figure 5.13: Snapshots of running motion from the trained policies



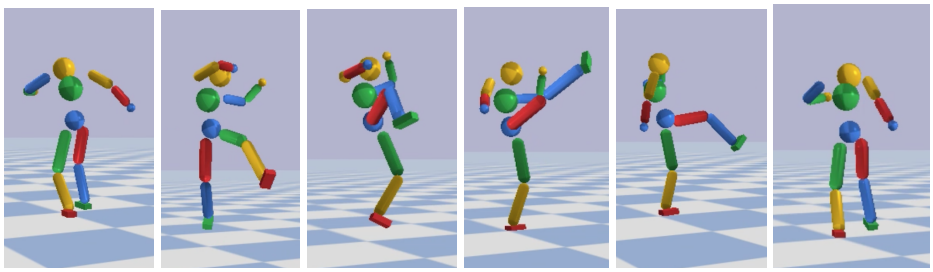
(a) (b) (c) (d) (e) (f)

Figure 5.14: Snapshots of backflip motion from the trained policies.



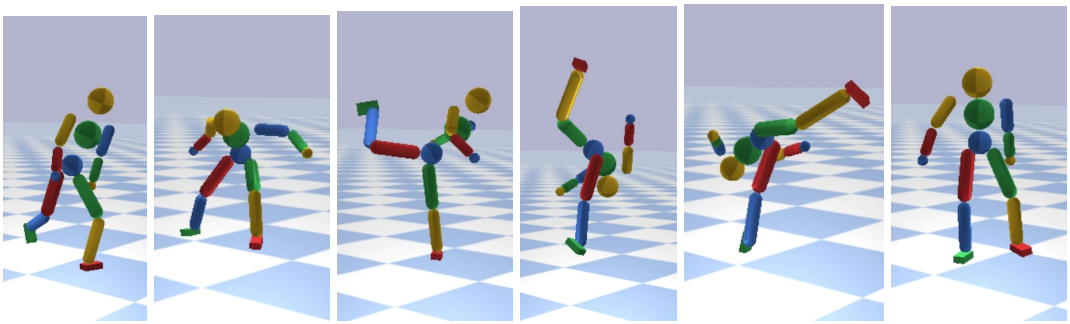
(a) (b) (c) (d) (e) (f)

Figure 5.15: Snapshots of cartwheel motion from the trained policies.



(a) (b) (c) (d) (e) (f)

Figure 5.16: Snapshots of spinkick motion from the trained policies.



(a) (b) (c) (d) (e) (f)

Figure 5.17: Snapshots of spin motion from the trained policies.

CHAPTER VI

CONCLUSION

In deep reinforcement learning, slow convergence of the training process and local optimal results are the main concerns of improvement nowadays. Because, if an agent can learn an environment with high-dimensional state-action space quicker, that will lead to a better way of using computational power. On the other hand, reaching a global optimal solution will also help the agent follow the perfect path to learn the environment. In our work, we have proposed two modified algorithms, for faster convergence and better average return during the training of a RL agent for learning high-dimensional continuous environments.

Policy gradient methods provides a way to approximate the policy gradients for policy updates. For this process we require to calculate advantage of an action. Estimating that advantage in a proper way can reduce the variance of the policy gradient optimization process and thus result in a quicker convergence and better average return of the RL training process. As a reusult, introducing generalized advantage estimation with vanilla A2C has proven to be improving the convergence speed and average return of the RL training process. In all of our experiment, we have discovered when the advantage is calculated with multi-step returns, it provides better performance than the vanilla A2C where the advantage is calculated using one-step return.

Adding the concept of magnifying salted reward with PPO adjusts the already defined reward function of imitation from experience and enriches the experience pool with salted rewards. This accelerates the convergence speed of the training and it also leads to the global optimal solution, i.e., higher average return during the training process. Our experiments with learning

different humanlike movements from reference motion have proven this. What's more, as this reward optimization is parameterized, this can also be adaptable to any other high dimensional RL environements.

BIBLIOGRAPHY

[1] S. AGRAWAL, S. SHEN, AND M. VAN DE PANNE, *Diverse motion variations for physics-based character animation*, in Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '13, New York, NY, USA, 2013, Association for Computing Machinery, p. 37–44.

[2] S. AGRAWAL AND M. VAN DE PANNE, *Task-based locomotion*, ACM Trans. Graph., 35 (2016).

[3] G. BROCKMAN, V. CHEUNG, L. PETTERSSON, J. SCHNEIDER, J. SCHULMAN, J. TANG, AND W. ZAREMBA, *Openai gym*, CoRR, abs/1606.01540 (2016).

[4] S. COROS, P. BEAUDOIN, AND M. VAN DE PANNE, *Robust task-based control policies for physics-based characters*, ACM Trans. Graph., 28 (2009), p. 1–9.

[5] S. COROS, P. BEAUDOIN, AND M. VAN DE PANNE, *Generalized biped walking control*, ACM Transctions on Graphics, 29 (2010), p. Article 130.

[6] M. DA SILVA, Y. ABE, AND J. POPOVIĆ, *Simulation of human motion data using short-horizon model-predictive control*, Computer Graphics Forum, 27 (2008), pp. 371–380.

[7] Y. DUAN, X. CHEN, R. HOUTHOOFT, J. SCHULMAN, AND P. ABBEEL, *Benchmarking deep reinforcement learning for continuous control*, CoRR, abs/1604.06778 (2016).

[8] S. HA AND C. K. LIU, *Iterative training of dynamic skills inspired by human coaching techniques*, ACM Trans. Graph., 34 (2015).

[9] P. HÄMÄLÄINEN, J. RAJAMÄKI, AND C. K. LIU, *Online control of simulated humanoids using particle belief propagation*, ACM Trans. Graph., 34 (2015).

[10] J. HO AND S. ERMON, *Generative adversarial imitation learning*, CoRR, abs/1606.03476 (2016).

[11] D. HOLDEN, T. KOMURA, AND J. SAITO, *Phase-functioned neural networks for character control*, ACM Trans. Graph., 36 (2017).

[12] D. HOLDEN, J. SAITO, AND T. KOMURA, *A deep learning framework for character motion synthesis and editing*, ACM Trans. Graph., 35 (2016).

[13] Z. HU, K. WAN, X. GAO, AND Y. ZHAI, *A dynamic adjusting reward function method for deep reinforcement learning with adjustable parameters*, Mathematical Problems in Engineering, 2019 (2019), p. 7619483.

[14] Y. LEE, S. KIM, AND J. LEE, *Data-driven biped control*, in ACM SIGGRAPH 2010 Papers, SIGGRAPH '10, New York, NY, USA, 2010, Association for Computing Machinery.

[15] Y. LEE, M. S. PARK, T. KWON, AND J. LEE, *Locomotion control for many-muscle humanoids*, ACM Trans. Graph., 33 (2014).

[16] Y. LEE, K. WAMPLER, G. BERNSTEIN, J. POPOVIĆ, AND Z. POPOVIĆ, *Motion fields for interactive character locomotion*, ACM Trans. Graph., 29 (2010).

[17] S. LEVINE, J. M. WANG, A. HARAUX, Z. POPOVIĆ, AND V. KOLTUN, *Continuous character control with low-dimensional embeddings*, ACM Trans. Graph., 31 (2012).

[18] L. LIU AND J. HODGINS, *Learning to schedule control fragments for physics-based characters using deep q-learning*, ACM Trans. Graph., 36 (2017).

[19] L. LIU, M. V. D. PANNE, AND K. YIN, *Guided learning of control graphs for physics-based characters*, ACM Trans. Graph., 35 (2016).

[20] L. LIU, K. YIN, M. VAN DE PANNE, T. SHAO, AND W. XU, *Sampling-based contact-rich motion control*, ACM Trans. Graph., 29 (2010).

[21] J. MEREL, Y. TASSA, D. TB, S. SRINIVASAN, J. LEMMON, Z. WANG, G. WAYNE, AND N. HEESS, *Learning human behaviors from motion capture by adversarial imitation*, CoRR, abs/1707.02201 (2017).

[22] V. MNIH, A. P. BADIA, M. MIRZA, A. GRAVES, T. LILLICRAP, T. HARLEY, D. SILVER, AND K. KAVUKCUOGLU, *Asynchronous methods for deep reinforcement learning*, in Proceedings of The 33rd International Conference on Machine Learning, M. F. Balcan and K. Q. Weinberger, eds., vol. 48 of Proceedings of Machine Learning Research, New York, New York, USA, 20–22 Jun 2016, PMLR, pp. 1928–1937.

[23] V. MNIH, K. KAVUKCUOGLU, D. SILVER, A. A. RUSU, J. VENESS, M. G. BELLEMARE, A. GRAVES, M. RIEDMILLER, A. K. FIDJELAND, G. OSTROVSKI, S. PETERSEN, C. BEATTIE, A. SADIK, I. ANTONOGLOU, H. KING, D. KUMARAN, D. WIERSTRA, S. LEGG, AND D. HASSABIS, *Human-level control through deep reinforcement learning*, Nature, 518 (2015), pp. 529–533.

[24] I. MORDATCH, E. TODOROV, AND Z. POPOVIĆ, *Discovery of complex behaviors through contact-invariant optimization*, ACM Trans. Graph., 31 (2012).

[25] U. MUICO, Y. LEE, J. POPOVIĆ, AND Z. POPOVIĆ, *Contact-aware nonlinear control of dynamic characters*, ACM Trans. Graph., 28 (2009).

[26] X. B. PENG, P. ABBEEL, S. LEVINE, AND M. VAN DE PANNE, *Deepmimic: Example-guided deep reinforcement learning of physics-based character skills*, ACM Trans. Graph., 37 (2018), pp. 143:1–143:14.

[27] X. B. PENG, G. BERSETH, AND M. VAN DE PANNE, *Dynamic terrain traversal skills using reinforcement learning*, ACM Trans. Graph., 34 (2015).

[28] X. B. PENG, G. BERSETH, AND M. VAN DE PANNE, *Terrain-adaptive locomotion skills using deep reinforcement learning*, ACM Transactions on Graphics (Proc. SIGGRAPH 2016), 35 (2016).

[29] X. B. PENG, G. BERSETH, K. YIN, AND M. VAN DE PANNE, *Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning*, ACM Transactions on Graphics (Proc. SIGGRAPH 2017), 36 (2017).

[30] X. B. PENG AND M. VAN DE PANNE, *Learning locomotion skills using deeprl: Does the choice of action space matter?*, in Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation, SCA '17, New York, NY, USA, 2017, ACM, pp. 12:1–12:13.

[31] A. RAJESWARAN, V. KUMAR, A. GUPTA, J. SCHULMAN, E. TODOROV, AND S. LEVINE, *Learning complex dexterous manipulation with deep reinforcement learning and demonstrations*, CoRR, abs/1709.10087 (2017).

[32] G. A. RUMMERY AND M. NIRANJAN, *On-line q-learning using connectionist systems*, tech. rep., 1994.

[33] A. SAFONOVA AND J. K. HODGINS, *Construction and optimal search of interpolated motion graphs*, in ACM SIGGRAPH 2007 Papers, SIGGRAPH '07, New York, NY, USA, 2007, Association for Computing Machinery, p. 106–es.

[34] J. SCHULMAN, S. LEVINE, P. ABBEEL, M. JORDAN, AND P. MORITZ, *Trust region policy optimization*, in Proceedings of the 32nd International Conference on Machine Learning, F. Bach and D. Blei, eds., vol. 37 of Proceedings of Machine Learning Research, Lille, France, 07–09 Jul 2015, PMLR, pp. 1889–1897.

[35] J. SCHULMAN, P. MORITZ, S. LEVINE, M. I. JORDAN, AND P. ABBEEL, *High-dimensional continuous control using generalized advantage estimation*, in 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings, Y. Bengio and Y. LeCun, eds., 2016.

[36] J. SCHULMAN, F. WOLSKI, P. DHARIWAL, A. RADFORD, AND O. KLIMOV, *Proximal policy optimization algorithms*, CoRR, abs/1707.06347 (2017).

[37] D. SHARON AND M. VAN DE PANNE, *Synthesis of controllers for stylized planar bipedal walking*, in Proc. of IEEE International Conference on Robotics and Animation, 2005.

[38] D. SILVER, A. HUANG, C. J. MADDISON, A. GUEZ, L. SIFRE, G. VAN DEN DRIESSCHE, J. SCHRITTWIESER, I. ANTONOGLOU, V. PANNEERSHELVAM, M. LANCTOT, S. DIELEMAN, D. GREWE, J. NHAM, N. KALCHBRENNER, I. SUTSKEVER, T. LILLICRAP, M. LEACH, K. KAVUKCUOGLU, T. GRAEPEL, AND D. HASSABIS, *Mastering the game of go with deep neural networks and tree search*, Nature, 529 (2016), pp. 484–489.

[39] K. W. SOK, M. KIM, AND J. LEE, *Simulating biped behaviors from human motion data*, in ACM SIGGRAPH 2007 Papers, SIGGRAPH '07, New York, NY, USA, 2007, Association for Computing Machinery, p. 107–es.

[40] R. S. SUTTON, *Learning to predict by the methods of temporal differences*, Machine Learning, 3 (1988), pp. 9–44.

[41] R. S. SUTTON AND A. G. BARTO, *Reinforcement Learning: An Introduction*, The MIT Press, second ed., 2018.

[42] R. S. SUTTON, D. MCALLESTER, S. SINGH, AND Y. MANSOUR, *Policy gradient methods for reinforcement learning with function approximation*, in Proceedings of the 12th International Conference on Neural Information Processing Systems, NIPS'99, Cambridge, MA, USA, 1999, MIT Press, pp. 1057–1063.

[43] Y. TASSA, T. EREZ, AND E. TODOROV, *Synthesis and stabilization of complex behaviors through online trajectory optimization*, 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, (2012), pp. 4906–4913.

[44] Y. W. TEH, V. BAPST, W. M. CZARNECKI, J. QUAN, J. KIRKPATRICK, R. HADSELL, N. HEESS, AND R. PASCANU, *Distral: Robust multitask reinforcement learning*, CoRR, abs/1707.04175 (2017).

[45] K. WAMPLER, Z. POPOVIĆ, AND J. POPOVIĆ, *Generalizing locomotion style to new animals with inverse optimal regression*, ACM Trans. Graph., 33 (2014).

[46] J. M. WANG, S. R. HAMNER, S. L. DELP, AND V. KOLTUN, *Optimizing locomotion controllers using biologically-based actuators and objectives*, ACM transactions on graphics, 31 (2012), p. 25. 26251560[pmid].

[47] C. J. C. H. WATKINS AND P. DAYAN, *Q-learning*, Machine Learning, 8 (1992), pp. 279–292.

[48] Y. YE AND C. K. LIU, *Optimal feedback control for character animation using an abstract model*, ACM Trans. Graph., 29 (2010).

[49] Y. YE AND C. K. LIU, *Synthesis of responsive motion using a dynamic model*, Comput. Graph. Forum, 29 (2010), pp. 555–562.

[50] K. YIN, K. LOKEN, AND M. VAN DE PANNE, *Simbicon: Simple biped locomotion control*, in ACM SIGGRAPH 2007 Papers, SIGGRAPH '07, New York, NY, USA, 2007, Association for Computing Machinery, p. 105–es.

BIOGRAPHICAL SKETCH

Md Rysul Kabir graduated with the Masters of Science in Computer Science program from the University of Texas Rio Grande Valley (UTRGV) in the August 13th of 2021. He started his journey at UTRGV as an awardee of the prestigious Presidential Graduate Research Assistantship from 2019. Since the beginning of his journey at the UTRGV, he has been performing research on machine learning and artificial intelligence under Dr. Dong-chul Kim–the PI of the Machine Intelligence lab of the computer science department. During his stay, he has not only worked as a research assistant but also worked as a teaching assistant for several courses in the computer science department.

After completing his degree he will be joining Indiana University Bloomington for this Doctor of Philosophy in Computer Science program starting from Fall 2021. His research interests include reinforcement learning, computer vision, natural language processing, and statistical inference.

Being born and raised in Bangladesh, he obtained his Bachelors of Science from the Bangladesh University of Engineering and Technology. He directly started his Doctor of Philosophy in Mechanical Engineering program at Missouri University of Science and Technology in 2018 where he stayed for a year before changing his major to computer science and joined UTRGV. For any help or information anyone can reach out to him using the following email address: rysulbuet12@gmail.com.