# RAINBOW POLICY GRADIENT

**Siddharth A. Dinkar**
Bellarmine College Preparatory
San Jose, CA 95135, USA
{s.dinkar23@bcp.org, siddharth.dinkar@gmail.com}

## ABSTRACT

The scientific literature in the field of reinforcement learning has provided many influential policy gradient algorithms and improvements. However, each of these algorithms address a singular problem, which leaves room for integration of these component algorithms to build a comprehensive agent that excels in high dimensional action spaces. In this paper, we examine major issues that arise in continuous action spaces for policy gradient algorithms and subsequently discuss four extensions to the Deterministic Policy Gradient algorithm (DPG, Silver et al. 2014) to build an integrated algorithm. We provide empirical comparisons between our algorithm and popular policy gradient algorithms, display the results of our hyperparameter tuning, and perform detailed ablation studies to outline the effect of each extension on our algorithms performance. Our integrated algorithm, which we call Rainbow Policy Gradient (RPG), delivers state-of-the-art performance on 6 locomotion and control tasks on OpenAI Gym environments, surpassing the performance of several popular policy gradient algorithms.

## 1 INTRODUCTION

As the available computational power of machines has grown over the past couple of decades, high capacity neural networks have become the standard function approximation techniques in various reinforcement learning (RL) algorithms, in areas such as predicting action-value function, optimal policies, etc. The Deep Q-Network (DQN) algorithm (Mnih et al., 2015) was the most fundamental success utilizing neural network function approximators to represent and learn action-value functions. Before the advent of this algorithm, it was generally believed that large nonlinear neural network function approximators were too unstable for such control problems. This algorithm surpassed human-level performance in Atari games and prompted new research into aspects of deep RL such as overestimation bias (DDQN; van Hasselt, Guez, and Silver 2016). A major issue stands out with this algorithm, however. Though DQN can handle complex, high-dimensional observation spaces, it cannot easily be adapted to continuous, high-dimensional action spaces (Lillicrap et al., 2015), and thus is constrained to discrete, low-dimensional action spaces. Accordingly, many alternatives and their extensions have been proposed to overcome this constraint.

The most promising alternative family of algorithms are the policy gradient algorithms, which optimize the policy directly. The policy gradient theorem establishes the proportional relationship between the gradient of the reward and the gradient of the natural log of policy (Sutton & Barto, 2017; Sec. 13.1), which dictates updates to the policy. The recent breakthroughs in policy gradient deep RL for complex, high-dimensional, robotic control first began at the introduction of the Deep Deterministic Policy Gradient (DDPG) algorithm (Lillicrap et al., 2015), based off an actor-critic approach with and deterministic actor method and a Q-network optimized toward a frozen target network. Subsequent improvements to DDPG include proportional prioritization with Prioritized Experience Replay (PER), first introduced in Schaul et al. (2015) for Double DQN but later adapted to DDPG in Hou et al. (2017). PER works by ranking samples from the replay based on absolute TD error. Essentially, this algorithm prioritizes replaying high error 'unexpected' transitions in order to learn on them more effectively. Twin Deep Delayed Deterministic Policy Gradient (TD3, Fujimoto et al. 2018) extends DDPG by correcting overestimation bias property that arises from Q-learning. Another alternative to the Deterministic Policy Gradient algorithm is the Soft Actor-Critic (SAC, Haarnoja et al. 2017), which utilizes stochastic actors rather than deterministic actors. SAC aims

to obtain the maximum reward while also maintaining maximum entropy, ie. acting as randomly as possible. This allows the actor to acquire more experience and leads to more stable learning updates. Similarly, research into neural network architectures such as Deep Dense Reinforcement Learning (D2RL, Sinha et al. 2019) have been shown to enhance RL algorithms.

These algorithms each address a singular solution that arises in policy gradient algorithms when learning complex high dimensional environments. In this paper, we build a comprehensive algorithm that implements many of the improvements discussed above. These improvements work in unison to alleviate prevalent issues in each respective policy gradient algorithm to allow our algorithm to achieve superior results compared with its component algorithms in 6 continuous control environments from the OpenAI Gym suite.

This paper begins with a brief background of reinforcement learning terminology and notation. We then continue by establishing the prevalent issues in policy gradient algorithms in a continuous control setting; specifically issues relating to the DDPG algorithm. Then, we explore various solutions, improvements, and their proofs provided in the literature of this field, and discuss how we can combine them to build an integrated algorithm. Finally, we showcase our algorithm by providing state of the art results from our experiments, explain our methods of hyperparameter tuning, and discuss the utility of our new algorithm.

## 2 BACKGROUND

Reinforcement learning is the study of an agent interacting with its environment to make the optimal decisions to achieve the highest scalar reward. The agent receives no prior knowledge of the environment and learns solely through its actions.

### 2.1 MARKOV DECISION PROCESSES

Markov Decision Processes (MDPs) represent the trajectory of an agent through the environment. In its conventional form for continuous action spaces, an agent begins with an initial observation $S_0 \in \mathbb{S}$ at time $t = 0$. As the agent progresses through the environment by acting with action $A_t \in \mathbb{A}$, it receives subsequent observations $S_{t+1} \sim p\ (\cdot \mid S_t,\ A_t)$, reward $R_{t+1}$, and discount rate $\gamma_{t+1}$ for every discrete time step $t$, where $p\ (S_{t+1} \mid S_t,\ A_t)$ is the stochastic transition function intrinsic to the environment. The MDP for can be formalized as the tuple $(\mathbb{S},\ \mathbb{A},\ p,\ r,\ \gamma)$, where the state space $\mathbb{S}$ and action space $\mathbb{A}$ are each continuous sets, $p$ represents the state transition function $p) : \mathbb{S} \times \mathbb{S} \times \mathbb{A} \to [0, \infty)$, $r$ represents the reward function $r : \mathbb{S} \times \mathbb{A} \to [r_{min},\ r_{max}]$, and represents the discount factor for future rewards. A policy $\pi : \mathbb{S} \to p(a)$ is the function attributed to the agent that maps a probability mass to actions given an observed state $S_t$. In this paper, we discuss uses and implementations of stochastic and deterministic policies. Simply, stochastic policies return action distributions $p(\mathrm{a})$, while deterministic policies return a single action $A_t$. $J(\phi)$ represents the expected cumulative reward of a policy $\pi_\phi$ with parameter $\phi$, where

$$J(\phi) = \mathbb{E}_{\phi,(s_t,a_t)\sim\rho_\pi}[r(s_t, a_t)] \tag{1}$$

$r(s_t, a_t)$ represents the expected cumulative discounted return $\sum_{k=0}^{T}(\gamma_t^k R_{t+k+1})$, where $T$ is the terminal time step. In the reinforcement learning problem, the goal is to find the optimal policy $\pi_\phi^* = argmax_\phi J(\phi)$, with paramters $\phi$ that maximizes the total discounted return in an episode.

### 2.2 PREDICTION AND CONTROL

Policy updates occur with respect to learned quantities in a reinforcement learning algorithm. These quantities are generally value-based estimations of the total discounted rates of a trajectory under an arbitrary policy $\pi_\phi$, with parameters $\phi$, such as $v_{\pi_\phi}(\mathrm{s}) = \mathbb{E}_{\mathrm{s}\sim\mathbb{S}}(G_t|S_t = \mathrm{s})$ for state value estimation or $q_{\pi_\phi}(\mathrm{s}, \mathrm{a}) = \mathbb{E}_{\mathrm{s}\sim\mathbb{S},\ \mathrm{a}\sim\mathbb{A}}(G_t|S_t = \mathrm{s}, A_t = \mathrm{a})$ for action value estimation, where s represents an arbitrary state and a represents an arbitrary action. Q-learning, which learns the aforementioned action value estimation $q_{\pi_\phi}(\mathrm{s}, \mathrm{a})$, is a popular standard for modern reinforcement learning algorithms. In

this paper, we primarily study RL algorithms that utilize Q-function approximation for prediction of action values.

The Q-function is a powerful prediction tool in RL. It can be more formally defined by the recursive relationship known as the Bellman equation (Lillicrap et al. 2015):

$$Q^{\pi}(\mathrm{s}_t, \mathrm{a}_t) = \mathbb{E}_{r_t, s_{t+1} \sim \mathbb{S}}[r(\mathrm{s}_t, \mathrm{a}_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^{\pi}(\mathrm{s}_{t+1}, \mathrm{a}_{t+1})]] \tag{1}$$

if the policy $\pi$ is non deterministic or

$$Q^{\mu}(\mathrm{s}_t, \mathrm{a}_t) = \mathbb{E}_{r_t, s_{t+1} \sim \mathbb{S}}[r(\mathrm{s}_t, \mathrm{a}_t) + \gamma Q^{\mu}(\mathrm{s}_{t+1}, \mathrm{a}_{t+1})] \tag{2}$$

for a deterministic policy $\mu : \mathbb{S} \rightarrow \mathbb{A}$ (Lillicrap et al. 2015). The aforementioned value-based estimation functions represent the prediction component of an RL algorithm.

Policies are structured around learned state-value or action-value estimates. Some simple policies include greedy policy or $\epsilon$-greedy policies with respect to a learned value estimation. A greedy policy consistently chooses the most rewarding action as estimated by the current iteration of the Q-function,

$$A_t = argmax_a \pi(S_t, a) \tag{3}$$

while an $\epsilon$-greedy policy chooses the action with the highest value with a probability $(1 - \epsilon)$ and chooses an action from an uniform distribution with probability $\epsilon$.

$$A_t = \begin{cases} argmax_a \pi(S_t, a) & \epsilon \\ a_t \sim p(A) & 1 - \epsilon \end{cases} \tag{4}$$

where $p(A)$ represents a uniform distrution of actions. These two policy constructions summarize the interdependence between exploration and exploitation for a given policy. While a greedy policy might seem to be the obvious choice, we must consider its implications. Since the agent receives only observations from the environment, the Q-function is just an estimation that is learned through interaction with the environment. This Q-function estimation gains accuracy as the agent acquires more experience in different trajectories through the environment. Thus, a purely greedy policy would base its decisions on inaccurate estimations and would not be able to explore separate trajectories that the faulty Q function would have deemed less rewarding. An $\epsilon$-greedy policy slightly alleviates this issue by requiring the agent to explicitly explore with probability $\epsilon$. These two policies only represent the simplest solutions to the exploration versus exploitation problem. More complex and efficient solutions have been proposed over the years to tackle this problem, some of which will be discussed in this paper.

In RL algorithms, prediction mechanisms and control mechanisms are updated simultaneously in a process called general policy iteration (GPI, Sutton & Barto 2017). In GPI, the control mechanism interacts with the environment epsilon-greedily to produce new experience that is used to update the estimation of action values. The policy is then updated to be greedy with respect to the policy gradient. Eventually, these functions converge to optimality. Current studies into the field of RL focus on improving convergence time to optimality. The interplay between the prediction and control mechanisms summarizes the family of actor-critic methods, where the prediction mechanism is represented by a critic function (i.e. Q-network) and the control mechanism is represented by a actor function (i.e. policy network).

### 2.3 Deep Reinforcement Learning

In continuous control RL problems, environments consist of high-dimensional observation and action spaces, making it impossible to represent the functions of state action pairs in tables or defined functions. Therefore, we represent these functions (i.e $\pi(\mathrm{s}, \mathrm{a})$ or $q(\mathrm{s}, \mathrm{a})$) as deep (i.e. multilayer) neural networks. These neural networks are parameterized by a weight vector $\theta$ with dimensionality $d \ll |\mathbb{S}|$, where $\mathbb{S}$ represents the continuous set of all observations (Sutton & Barto 2017; Section 9.1). The significance of using a relatively small dimensionality of the parameter vector compared

to the observation space is to reduce the computational and memory complexity of function approximators while still maintaining their efficacy. The neural network function approximators learn their respective functions through updates to their parameter vectors based on the gradient descent to minimize mean squared error loss function (MSE) $L(\theta)$ with respect to $\theta$.

$$L(\theta) = \mathbb{E}_{(S_t, A_t, R_{t+1}, S_{t+1}) \sim D}[(y_t - q_\theta(S_t, A_t))^2] \tag{5}$$

where $D$ represents the replay buffer from which state transition tuples are sampled.

### 2.4 DDPG

To discuss the basics of neural network approximation and its relation to prediction and control in continuous action spaces, we will consider the DDPG algorithm. The DDPG algorithm is the basis for the integrated algorithm we present in this paper. It is an off-policy algorithm that uses an exploration policy, $\beta = \mu_\phi(\mathrm{s}) + N(0, \eta)$ to gain experience and updates the true policy based on the gained experience. It consists of a deterministic actor function $\mu_\phi(\mathrm{s})$, with parameters $\phi$, which is updated through gradient ascent with respect to the gradient of the reward (i.e the policy gradient (Silver et al. 2014)),

$$\nabla_{\phi^\mu} J(\phi) = \mathbb{E}_{s_t \sim p^\beta, a_t \sim \mu(s_t)}[\nabla_a Q_\theta(s_t, a_t) \nabla_{\theta_\mu} \mu_\theta(s_t)] \tag{6}$$

and a critic function $q_\theta(\mathrm{s}, \mathrm{a})$, with parameters $\theta$, which is updated through gradient descent of the mean squared error loss function (Lillicrap et al. 2015) (See Equation 6 6).

The policy function and the Q-function each have a corresponding frozen target function with parameters $\phi'$ and $\theta'$ respectively, which provides a stable target for gradient updates. This target objective is represented by $y_t$ where

$$y_t = (R_{t+1} + \gamma max_{a'} q_{\theta'}(S_{t+1}, a') - q_\theta(S_t, A_t))^2 \tag{8}$$

The target networks are frozen with respect to their parameters in order to provide stable learning for the learned networks. To maintain a steady target, target networks are updated slowly via *polyak averaging*:

$$\theta' = \tau\theta' + (1 - \tau)\theta \tag{9}$$

$$\phi' = \tau\phi' + (1 - \tau)\phi \tag{10}$$

where $\tau \in [0, 1]$. Finally, DDPG utilizes a finite replay buffer $\mathbb{R}$ to store experience from the exploration policy. Experience is stored as a tuple of the form $(s_t, a_t, r_{t+1}, s_{t+1})$.

## 3 Related Works

The DDPG algorithm remains a popular algorithm used in control systems that yields state-of-the-art results in many reinforcement learning benchmarks. However, it presents many issues and limitations that inhibit its performance in real-world scenarios. In this section, we introduce these specific limitations and discuss how they emerge within the DDPG algorithm. Then we dive into the literature of the field and see how we can address these limitations. Finally, we propose the integration of several of these improvements and describe the resulting integrated reinforcement learning algorithm.

### 3.1 Double Delayed Q-Networks

The critic networks in DDPG have been shown to be affected by overestimation bias, a phenomenon that propagates small errors in target Q-function approximation due to bootstrapping, which results in poor learning for an RL agent. Fujimoto et al. (2018) establishes that the greedy temporal difference (TD) target $y_t = \mathrm{r} + \gamma(max_a Q(\mathrm{s}', \mathrm{a}'))$ is susceptible to a small positive error $\epsilon$, such that

$\mathbb{E}[max_a Q(\mathrm{s}', \mathrm{a}') + \epsilon] \geq max_a Q(\mathrm{s}', \mathrm{a}')$ (Thrun & Schwartz 1993), where the Q-function estimation of the value of the maximum action exceeds the true maximum. Though $\epsilon$ may be a small value, regular updates to critic network parameters compound this problem leading to massive value overestimation.

This issue was first identified for DQN in discrete action spaces and solved in the Double Deep Q Networks algorithm paper (van Hasselt, Guez, and Silver, 2016) by using decoupled double Q networks to alleviate overestimation of the temporal difference target. This paper presented the following modified temporal difference error that the Q-function aims to minimize:

$$(R_{t+1} + \gamma max_{a'} q_{\theta'}(S_{t+1}, argmax_{a'} q_{\theta'}(S_{t+1}, a')) - q_\theta(S_t, A_t))^2 \tag{11}$$

Fujimoto et al. (2018) adapts this idea of double Q networks to continuous action spaces. They argue for double clipped Q networks, which perform independent value estimations of the state-action space. The minimum of the two estimations is used in formulating the TD target, $y_t$, in the loss function:

$$y_t = r + \gamma min_{i=1,2} Q_{\theta_i{}'}(s', \pi_\phi(s'))] \tag{12}$$

We borrow this idea as an extension to our Rainbow Policy Gradient algorithm.

Fujimoto et al. (2018) also utilizes a policy update delay in their TD3 algorithm. Target networks are fundamental to producing convergent policy updates that help policy networks find optimality. In many applications they are frozen and updated slowly via polyak averaging to stabilize Q-learning. Frequently, however, Q-function loss with respect to the target network can be high, leading to inefficient and divergent policy network updates. Fujimoto et al. (2018) argues for delayed policy updates to allow for Q-function loss to be reduced. We also implement this extension in our integrated algorithm, reducing the number of divergent policy updates.

### 3.2 Proportional Prioritization

RL algorithms use experience replays to store state transitions for retrieval during the training step. Generally, these transition batches are sampled uniformly from the replay. Schaul et al. (2015) acknowledged the prominent inefficiency with this practice in their Prioritized Experience Replay (PER, Schaul et al. 2015). They introduce two methods of prioritization in their paper, proportional prioritization and stochastic prioritization. The more popular method, proportional prioritization uses absolute TD errors to rank state transitions in the experience replay. The transitions are then sampled with a probability, $p_t$, proportional to their weighted TD errors:

$$p_t = \frac{|(R_{t+1} + \gamma max_{a'} q_{\theta'}(S_{t+1}, a') - q_\theta(S_t, A_t))^2|^\alpha}{\sum_i p_i^\alpha} \tag{13}$$

where $\alpha$ is the prioritization hyperparameter. New transitions are also given greater priority. TD errors play an important role in the probability distribution of state transitions in the replay buffer. A high TD error signifies a transition that is "surprising" or has a high learning potential (i.e. the Q-function approximation for this state-action pair can be significantly improved). Thus it is more likely to be sampled in order to increase the accuracy of the value estimation. Further, learning on transitions with high learning potential needs to have a greater effect on Bellman updates on Q-functions for these networks to converge to optimality quicker. Thus, parameters of Q-networks are also updated proportionally to the accumulated weighted TD errors of the sampled transitions.

We find that a proportionally prioritized experience replay provides drastic performance increases for our integrated algorithm.

### 3.3 Maximizing Entropy

Reinforcement learning algorithms face the dilemma of exploration versus exploitation. Several solutions have been presented in scientific literature but the maximum entropy reinforcement learning framework (Ziebart et al., 2008; Toussaint, 2009; Rawlik et al., 2012; Fox et al., 2016; Haarnoja et

al., 2017) provides the most stable and efficient strategy to promote exploration. This framework, first successfully implemented in the Soft Actor-Critic (Harnooja et al. 2017), defines an agent that aims to maximize rewards while maximizing entropy (i.e. acting randomly to explore observation spaces). Maximum entropy learning augments the standard RL objective, which is to maximize cumulative reward $J(\phi)$. The maximum entropy objective adds a standard entropy term, defining a new objective:

$$J(\phi) = \sum_{t=0}^{T} \mathbb{E}_{\phi,(s_t,a_t)\sim\rho_\pi}[r(s_t, a_t) + \alpha\mathbb{H}(\cdot|s_t)] \tag{14}$$

where $\alpha$ represents the temperature parameter, which signifies the importance of the entropy term relative to the standard RL objective (Harnooja et al. 2017). The practical advantages of this entropy term include the capture of multiple optimalities, which is a symptom of incentivized exploration. Stochastic policies derived from the modified RL objective through gradient ascent assign equal weight to several optimal actions in continuous control applications.

We implement the maximum entropy objective in our Rainbow Policy Gradient algorithm to stabilize learning and maximize exploration.

### 3.4 Multi-Step Targets

Traditional Q-function updates use a singularly accumulated reward and greedily bootstrap using a target network to estimate the value of the optimal action (Hessel et al. 2017). This process is known as a one-step temporal difference method (Sutton & Barto 2017). In our Rainbow Policy Gradient algorithm, we utilize discounted $n$-step temporal difference updates to reduce dependency on Q-functions, which we have proven to be susceptible to overestimation bias. We found that $n = 3$ allowed our algorithm to best outperform its competitors.

The $n$-step TD method requires multiple batches of $n$ subsequent transitions to be sampled in order to produce a discounted $n$-truncated reward (Sutton & Barto 2017):

$$R_t^{(n)} = \sum_{k=0}^{max(T,n)} (\gamma_t^{(k)} R_{t+k+1}) \tag{15}$$

This truncated reward is then applied to the $n$-step TD error target to stabilize Q-learning due to further minimization of overestimation bias:

$$R_t^{(n)} + \gamma_t^{(n)} max_{a'} q_{\theta'}(S_{t+n}, a') - q_\theta(S_t, A_t))^2 \tag{16}$$

Sutton & Barto (2017) states that a properly tuned value of $n$ results in drastic performance increases and faster learning in reinforcement learning algorithms.

### 3.5 Deep Dense Neural Network Architectures

In our integrated agent, we introduce the idea of deep dense networks which enables our actor-critic method to better approximate complex high-dimensional observation and action spaces. Deep dense neural networks outperform traditional neural networks by enabling better optimization through smoother loss landscapes and by alleviating data-processing inequality (DPI, Huang et al. 2017).

The Deep Dense Reinforcement Learning paper (D2RL) by Sinha et al. (2019) provides a framework for deep dense networks in continuous control problems. Sinha et al. (2019) discusses the optimal depth of neural networks, claiming that a two layer neural network with residual connections provides the greatest performance benefits over the traditional linear network. Simply increasing depth and connectivity linearly does not lead to performance benefits; rather, it increases network instability through vanishing gradients (Hochreiter & Schmidhuber, 1997) and loss of mutual information between states (He et al. 2017a). Residual connections are a solution proposed in He et al. (2016a) to combat the issues of instability in a deep neural network.

Increasing the density of neural networks combats the problem of DPI. A common method of analysis of neural network architecture is to view a network as a Markov Chain of the form $X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow ... \rightarrow X_n$ (Sinha et al. 2019). The Data-processing inequality

$$MI(X_1 : X_2) \geq MI(X_1 : X_n) \tag{17}$$

for $n > 2$, shows that mutual information between the first layer and each consecutive layer is lost as the data propagates forward through the network. Huang et al. (2017) provides a method of increasing density of the networks: input concatenation, where the original input is simply concatenated onto each hidden layer of the network. See Huang et al. (2017) and Sinha et al. (2019) for proofs of quicker convergence.

We borrow these two ideas of increasing depth and density of our neural networks and implement the D2RL architecture in our integrated algorithm.

### 3.6 Integrated Agent

Our integrated agent consists of the above five extensions: Clipped Double Delayed Q-networks (Fujimoto et al. 2018), Proportionally Prioritized Experience Replay (Schaul et al. 2015), the Maximum Entropy framework (Harnooja et al. 2017), Multi-Step Bootstrapped Targets (Sutton & Barto 2017), and the Deep Dense Neural Network Architectures (Sinha et al. 2019). We call this integrated algorithm Rainbow Policy Gradient, a name influenced by its peer algorithm which operates in discrete action spaces, Rainbow DQN (Hessel et al. 2017). We replace a singular critic network with our double clipped networks for overestimation bias minimization and implement a policy update delay to reduce instances of divergent policy updates. Next, we replace a uniformly sampled replay buffer with a prioritized buffer to allow for better Q-function approximation of unexpected state transitions. Additionally, we introduce the maximum entropy framework to allow for the maximum possible exploration during training for our agent. Further, we utilize discounted $n$-step TD targets to reduce dependence on bootstrapping, which results in less overestimation bias. Finally, we modify our traditional neural network architecture to a dense network architecture which incorporates input concatenations to reduce data processing inequalities (DPI, Huang et al. 2017) as we propogate forward through our neural network.

## 4 Expiremental Methodologies

In this section, we present the learning curves and policy evaluations for our trained agents, describe our methods of evaluating the integrated algorithm, and discuss hyperparameter tuning.

### 4.1 Evaluation

We evaluate our integrated agent on 6 different continuous control tasks using the MuJoCo (Todorov et al. 2012) physics engine accessed through OpenAI Gym (Brockman et al. 2016). To promote reproducibility, we use the original environment and reward setup for each of these 6 control tasks as outlined in Brockman et al. (2016). For all of our experiments, we report performance statistics after 1 million training steps. Performance statistics are reported as the maximum average score obtained by the agent in any given environment over 5 trials.

In our experiments, we compare the learning curves and maximum returns of our integrated agent to 3 popular off-policy actor-critic algorithms as well as one on-policy actor-critic algorithm. The 3 off-policy reference algorithms include DDPG (Lillicrap et al. 2015), TD3 (Fujimoto et al. 2018), SAC (Harnooja et al. 2017). Additionally, we compare our integrated agent to one variation of one of the above algorithms, DDPG + PER (Schaul et al. 2016), which uses a prioritized replay to efficiently sample beneficial transitions from the memory. Finally, we compare our integrated Rainbow Policy Gradient algorithm the state-of-the-art on-policy algorithm, Proximal Policy Optimization (PPO, Schulman et al. 2017).

The actor and critic networks for the above algorithms utilize two layer feedforward neural networks with 256 hidden nodes each and rectified linear activation units (ReLU) between each layer. The actor network uses the hyperbolic tan activation unit for the final output. The critic network input

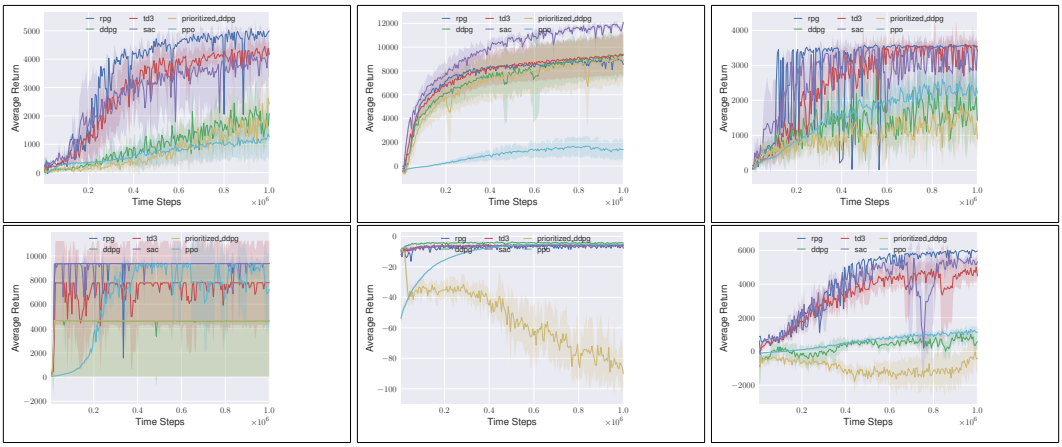


Figure 1: Learning curves for OpenAI Gym continuous control environments. Shaded regions represent one standard deviation.

Table 1: Max Average Return over 5 trials. Best performances are bolded.

| Environment | RPG | SAC | TD3 | DDPG | Prioritized DDPG | PPO |
|---|---|---|---|---|---|---|
| Walker2d-v2 | **4834.1304** | 4256.672 | 4517.9214 | 2666.4543 | 2867.37 | 2784.0571 |
| HalfCheetah-v2 | 8180.248 | **11715.94** | 9844.99 | 8170.5186 | 9797.448 | 3307.05 |
| Hopper-v2 | **3345.571** | 1849.25 | 3217.4575 | 2213.05 | 1748.51 | 1998.5316 |
| InvertedDoublePendulum-v2 | **9349.01** | 9113.73 | 9262.405 | 9203.275 | 9249.443 | 9174.663 |
| Reacher-v2 | -13.53 | -14.188 | -6.198 | **-4.057** | -86.70 | -4.850 |
| Ant-v2 | **5408.281** | 5366.183 | 4998.467 | 1246.7874 | -30.123 | 1486.192 |

layer takes both an observation and an action. All actor and critic networks are updated via gradient descent using the Adam optimizer (Kingma & Ba, 2014) with a learning rate of $10^{-3}$, while all target actor and critic networks are frozen and updated via polyak updating with $\tau = 0.995$. The algorithms accumulate experience before they begin training on mini-batches of 100 state transitions randomly sampled from the replay buffer. For the first $10^4$ steps, each algorithm's actions are sampled purely randomly in order to accumulate the necessary exploratory experience to begin policy training on unstable environments.

The deterministic off-policy algorithms (i.e. DDPG, TD3 and variations) use an exploration policy, $\beta$, derived from the true policy by adding a Gaussian exploration noise:

$$\beta = \mu_\phi(\mathrm{s}) + N(0, \eta) \tag{18}$$

We utilize Gaussian noise instead of a uniformly sampling policy to simplify our algorithm and avoid the process of importance sampling. Also, as stated in Fujimoto et al. (2019), sampling correlated exploration noise through the Ornstein-Uhlenbeck process (Ornstein & Uhlenbeck, 1930) provides no learning or performance benefits, so we do not implement this process to assist the exploration of any of our algorithms. Our stochastic policy algorithms (i.e. SAC, Integrated Algorithm, and variations) do not require crafted exploration strategies as the maximum entropy framework (Ziebart et al., 2008; Toussaint, 2009; Rawlik et al., 2012; Fox et al., 2016; Haarnoja et al., 2017) which these algorithms are built on already promote maximum exploration.

Each algorithm is trained on 5 random seeds each of 6 different OpenAI Gym continuous control environments for 1 million training steps. The trained models are then evaluated for 100 episodes on each environment with no exploration noise (optimal policy).

Our results are compiled in Figure 1 and Table 1. Our algorithm outperforms its competitors in terms of policy performance and learning rate.

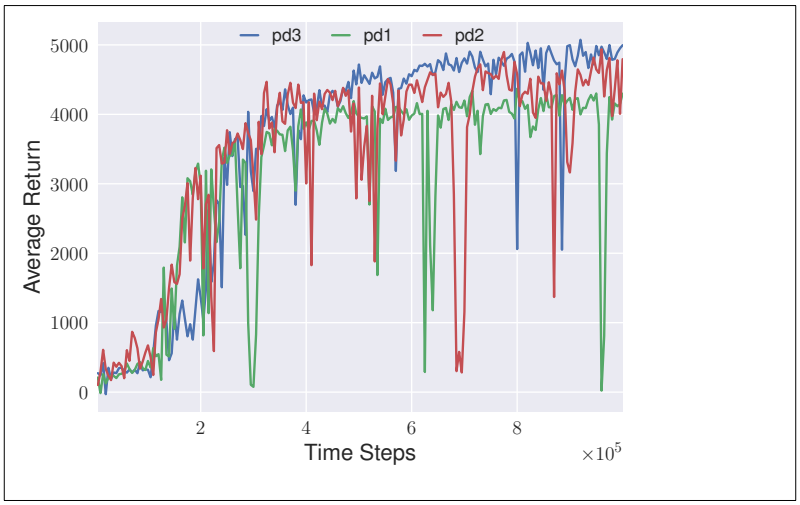


Figure 2: Hyperparameter search for policy update value. We are testing values of 1 (green), 2 (red), and 3 (blue). Learning curves are obtained from the Walker2d-v2 task.

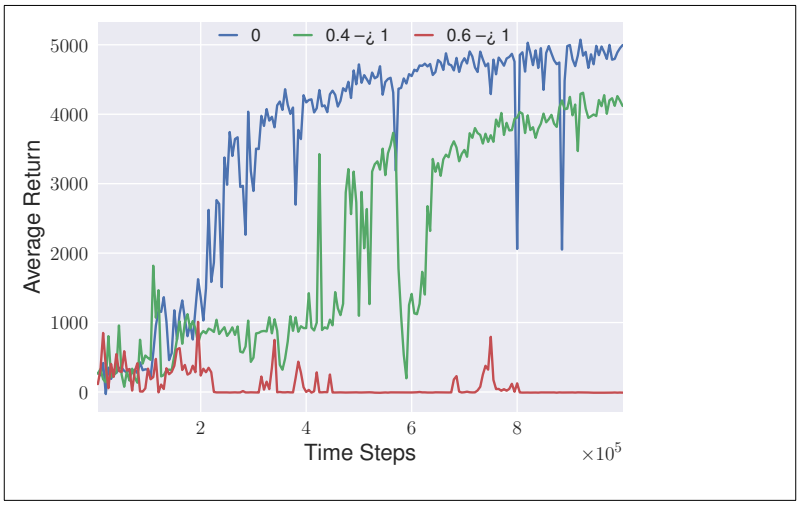


Figure 3: Hyperparameter serach for $\beta$ annealing schedule. We consider two linear schdules, $0.4 \rightarrow 1$ (green) and $0.6 \rightarrow 1$ (red), as well as one fixed value for $\beta$ of 0 (blue). Learning curves are obtained from the Walker2d-v2 task.

### 4.2 Hyperparameter Tuning

Our integrated algorithm is composed of many components which each have a number of hyperparameters that require tuning. We choose to perform hyperparameter tuning on two major components of our integrated agent, specifically the prioritized experience replay (Schaul et al. 2016) and the policy update delay. We begin our tuning with the hyperparameter values used in each component's paper and compare these results to new hyperparameter configurations.

Beginning with the prioritized experience replay (PER), we use the TD-error proportional prioritization recommended by Schaul et al. (2016). After extensive experimentation, we found that the priority exponent alpha offers no performance benefits at values greater than 0.6. Thus, we use the recommended $\alpha = 0.6$ for our priority exponent. For the importance sampling weight, $\beta$, we com-

pared three different annealing schedules. Our tuning results showed that a static $\beta = 0$ provided the greatest stability and performance improvements in the Walker2d-v2 environment in OpenAI Gym. The other $\beta$ annealing schedules we tested were $0.4 \rightarrow 1$, the recommended schedule in Schaul et al. (2016), and $0.6 \rightarrow 1$. Both of these annealing schedules resulted in unstable learning curves due to either extreme overestimation bias or stagnant target networks in our integrated algorithm. The performance of each of these $\beta$ values is presented in Figure 3 **??**.

Next, we tune the policy update delay in our integrated algorithm. Policy update delay is the method introduced in Fujimoto et al. (2018) which helps address the problem of overestimation bias and value estimation variance in Q-function approximators (See section 3.1). We test policy update delays of $1, 2, 3$, where a value of 1 denotes no policy update delay. We observed that a policy update delay of 3 offers the greatest performance benefits as shown in Figure **??**.

See Table **??** for all the hyperparameters of our integrated agent.

Table 2: Tuned Hyperparameter Values

| **Parameter** | **Value** |
|---|---|
| Minimum History | $10^4$ frames |
| Adam Learning Rate | $10^{-3}$ |
| Adam $\epsilon$ | $10^{-4}$ |
| Policy Update Delay | 3 |
| Prioritization type | proportional |
| Prioritization $\alpha$ | 0.6 |
| Prioritization $\beta$ Schedule | 0 |
| Multi-Step target $n$ value | 3 |

## 5 CONCLUSION

Over many years, literature in the field of continuous control and locomotion through reinforcement learning has produced many algorithms that each address a specific problem. In this paper, we unify the innovations of many papers to build an integrated algorithm that surpasses the performance of many state of the art algorithms used in industry today. This paper begins by establishing prominent issues in policy gradient algorithms, such as overestimation bias or inadequate exploration, which emerge in continuous control environments. We then discuss solutions presented in the scientific literature of this field, integrate them into one agent that limits all of the prevalent shortcomings, and present the relevant empirical data to prove that our algorithm can outperform modern reinforcement learning algorithms.

Our algorithm addresses four major problems for policy gradient algorithms learning continuous action spaces. First, we tackle the issue of overestimation bias, utilizing multiple clipped Q-network functions approximators and performing delayed policy updates (Fujimoto et al. 2018) to stabilize the policy iteration process (GPI, Sutton & Barto 2017). We then address inefficient replay buffers by introducing the proportional prioritization strategy (PER, Schaul et al. 2016), to increase state transition sampling efficiency for better value estimation of the observation space of a continuous control environment. Additionally, we introduce the maximum entropy framework (Harnooja et al. 2017) to address the problem of inadequate exploration, which leads to poor Q-function efficacy. Finally, we utilize multi-step temporal difference targets for our loss function, which allow for faster learning (Sutton & Barto, 2017).

The amalgamation of these components define our proposed integrated algorithm, which we call the Rainbow Policy Gradient algorithm (RPG). This algorithm significantly improves off-policy policy gradient algorithms in several locomotion and control tasks in 6 different MuJoCo (Todorov et al. 2012) environments. These generalized improvements are greatly influential and can be applied to many policy gradient algorithms to enhance performance.