UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

REINFORCEMENT LEARNING FOR CONTINUOUS CONTROL

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

BY

Shyam Sundar Murali Krishnan
Norman, Oklahoma
2020

REINFORCEMENT LEARNING FOR CONTINUOUS CONTROL

A THESIS APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY THE COMMITTEE CONSISTING OF

Acknowledgements

I would like to express my sincere appreciation and gratitude to my thesis advisor Dr. Dean Hougen for his continuous guidance, valuable comments and feedback at various stages of the thesis work. I would also thank him for believing in me and introducing and motivating me to this thesis topic.

I would also like to thank Dr. Sridhar Radhakrishnan and Dr. Christan Grant for their willingness to serve on my thesis committee. I would like to acknowledge the Director of Computer Science Dr. Sridhar Radhakrishnan, faculty and staff for their support during my graduate studies.

I would like to specially thank Dr. Syed Naveed Hussain Shah for sharing his work and helping me understand the basics of my thesis.

I would like to specifically thank my family members for their constant support and motivation. I would like to thank my mother Nithya, my father Murali Krishnan, and my brother Sai Ramanan. I would like to thank my friends for motivating and supporting me.

I would like to thank OU Supercomputing Center for Education and Research (OSCER) team for providing computation resources for this thesis.

Abstract

Reinforcement learning is an area in machine learning that concerns how agents should take actions in an environment to maximize its reward. Earlier works on reinforcement learning algorithms work well for discrete action and discrete observations. In the continuous environment the actions are infinite in number and observations are also infinite. So certain reinforcement algorithms were developed in order to learn on continuous spaces. Among those, two algorithms are Stochastic Synapse Reinforcement Learning (SSRL) and Deep Deterministic Policy Gradient (DDPG). Previous works used these algorithms on different environments separately in order to understand the behaviour of the algorithms. In this thesis, a comparison study is made on these algorithms by using them on the same continuous environments and observing how each algorithm behaves on each environment and what kind of strengths and weaknesses can be inferred by comparing the algorithms. The algorithms are made to run on two continuous environments, namely mountain car continuous and pendulum. They are run 10 times for each set of time steps like 2000, 3000, 4000 for 1000 episodes each and the cumulative reward at the end of each episode is found. The episode is the length of the simulation at end of which the algorithm ends in a terminal state. The average and standard deviation of cumulative rewards across 10 repetitions for each time step and all the repetitions across different time steps are also collected. The results shows different trends across different experiments. Based on the results it can be inferred that overall SSRL performs consistently even though it does not gains rewards like DDPG whereas DDPG performs inconsistently but certain rewards it earns are higher than those of SSRL. Also in the case of the delayed-reinforcement pendulum environment both algorithms do not learn well, showing their weakness towards environments whose terminal state is not definite.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Reinforcement learning is an area in machine learning which concerns with how the agents would take actions in an environment to maximize its reward. *Reward* is the measure of how successful the action is with respect to completing the task. Reinforcement learning algorithms has been particularly useful and productive in learning the discrete environment but had less success in continuous environments. An *environment* is everything in the world that surrounds the agent; in other words, an environment can be described as a surroundings in which an agent can be present. The environment is the place where the agents resides and operate. The environment can have both discrete states and actions, discrete states and continuous actions, continuous states and discrete actions or both continuous actions and states. *States* are the representation of the world and *actions* are something the agent can do to change the states. If the environment is discrete then it will have finite number of actions or states or both. If the environment is continuous then it will have infinite number of actions or states or both. In this thesis, we work on environments having continuous states and continuous actions. It is regarded as one of the challenging environments because it is impossible for an agent to encounter an infinite number of states and impossible for an agent to take an infinite number of actions. This is part of what makes reinforcement learning in continuous state and action spaces difficult. In our computer simulations, we don't actually have continuous values. Instead, we use finite approximations, so the number of possible states and actions isn't literally infinite, but it is still far too large to possibly try all values.

Another challenge faced in the research of reinforcement learning is to handle delayed reward structure (Lee et al., 2019). *Delayed reward* is a kind of rewarding structure where the terminal reward is given only at the end of the episode. *Episode* is the length of the simulation at end of which the algorithm ends in a terminal state. The reason for delayed reward structure being a challenge is because the agent's actions determine not only its immediate reward, but also the next state of the environment. In normal case, the agent takes the action to go to the next state which is validated by giving a reward. But due to the delayed reward structure, where the reward is given only after a series of actions, the next states may not be effective because there is no rewards to validate the action to go to a better state. Such environments can be thought of as network of problems, but the agent must also take into consideration the next state as well as immediate reward when it decides which action must be taken.

This chapter gives a brief introduction about the challenges faced by the reinforcement learning and how to solve it, the research objectives of the thesis, overview on the results, contribution to the research community and the organization of the thesis.

1.1 Overview

The major difficulty in dealing with continuous action spaces and continuous state spaces is because it is impossible for an agent to encounter an infinite number of states and impossible for an agent to take an infinite number of actions. It becomes hard to find a policy because it has infinite possibility of actions that makes hard to select a particular action. *Policy* is the mapping that selects actions based on the observations from the environment. A way to deal with continuous environments is to use *actor- critic* approaches (Konda and Tsitsiklis, 2000) where *actor* outputs the action for a given state and it controls how the agent behaves by learning the optimal policy. *Critic* outputs the action value which is used to estimate how good is the action for a given state. The way both interact is that the the actor receives as input the observation from the environment and outputs a suitable action for the state. The critic receives as input the observation from the environment and the action given by the actor and outputs the action value for the action taken. This approach deals well with continuous environments because the actor network optimizes the policy function which is essentially a conditional probability distribution of actions given a state.

To work well on continuous action spaces the reinforcement learning could use an artificial neural network which works as an approximation mechanism. Another issue that occurs in continuous spaces is that for attaining the reward multiple synapses may be responsible for it and such synapses should be recognized properly. This is called *structural credit assignment problem* (Gullapalli, 1992). One approach used in continuous action spaces is to sample random noise using a zero-mean normal distribution and add this noise to the continuous action value which is deterministically provided by the neural network. This approach allows for stochasticity in the neural network which can yield better policy.

In case of a delayed reward, to achieve long run optimality the agent should also take the value of likely future rewards into account. That is, the agent should be able to learn from the delayed reinforcement also. The process may take a long sequence of actions, initially resulting in insignificant reinforcement but in the future ending up receiving very high reinforcement. The agent must be able to learn which of its actions are desirable based on the reward that can take place in the future.

The problem that delayed reward gives is called *temporal credit assignment problem* (Sutton, 1984) where it is that it is hard to determine which action is responsible for the reward. One approach is to consider those elements necessary for actions to be eligible for change by the process called *eligibility traces*. The eligibility traces keep track of which parameters have been mostly updated in addition to the states which are most recently updated. The reason behind using the eligibility traces is that while calculating the traces it considers the past actions too which also may consists of traces where certain actions provided better rewards and these traces may be very useful while dealing with delayed rewards.

Another way of dealing with delayed reward is by using a *replay buffer* (Moore and Atkeson, 2004; Lee et al., 2019) where the agent's experience which has the observation, action, reward and observation to the next step is stored at each time step. During learning, updates are done on samples of experience, drawn uniformly at random from the pool of stored experience. The reason why they work is that the replay buffer will have pool of experience stored in the buffer which may give an idea to select which kind of actions are needed to provide better rewards.

Based on the ideas to deal with the delayed reward environment and continuous environment two algorithms were proposed namely *Stochastic Synapse Reinforcement Learning (SSRL)* (Shah and Hougen, 2017, 2020) and *Deep Deterministic Policy Gradient (DDPG)* (Lillicrap et al., 2015). This thesis presents a comparative study these algorithms and inspects the advantages and disadvantages of each algorithm on environments with continuous state and action spaces and delayed rewards.

1.2 Research Objective

The objective of this research is to present a comparative study on Stochastic Synapse Reinforcement Learning and Deep Deterministic Policy Gradient by applying them to environments that have continuous actions and continuous observations as well as delayed rewards and observe the advantages and disadvantages of these algorithms.

The motivation for conducting this research is that these two algorithms represent fundamentally different approaches to the problem. SSRL uses stochastic noise which is used for learning policy and DDPG uses deterministic policy. In early works these two algorithms have been applied to different environments separately but never been compared under same set of environments and observed the behaviour, which is done in this thesis.

The comparative study on these algorithms are made by collecting the cumulative rewards for various repetitions, the average of cumulative rewards and standard deviation of cumulative rewards for different time steps and the overall average and standard deviation of cumulative rewards for various environments.

1.3 Overview of Results

Based on the comparative studies it is observed that SSRL gives consistent performance where the algorithm manages not to perform poorly whereas in case of DDPG, the algorithm occasionally gives better rewards compared to SSRL but does not manage to perform consistently. Also, in the delayed-reinforcement pendulum environment both algorithms struggles in learning.

1.4 Contribution to the Research Community

The knowledge that is obtained by conducting this research is that SSRL gives relatively consistent performance where the algorithm manages not to perform poorly whereas DDPG occasionally gives better rewards compared to SSRL but does not manage to perform consistently. Also, in the delayed-reinforcement pendulum environment both algorithms struggle in learning. This knowledge are important because if consistency is the priority then SSRL should be used or maximum performance is the priority then usage of DDPG could be suggested. Also, because both algorithms do not perform well in delayed-reinforcement pendulum environment it leads to a open question where there are algorithms that perform better on environments where the terminal state of the environment is indeterminate.

1.5 Organization of Thesis

The rest of this thesis is organized into six chapters. Chapter 2 covers background concepts needed to understand the algorithms. Chapter 3 is a literature survey about the algorithms used. Chapter 4 covers the methodology. Chapter 5 covers the results and observations. Chapter 6 is a discussion of the results and Chapter 7 is the conclusions and future work.

Chapter 2

Related Work

In this chapter, the basic building blocks for constructing Stochastic Synapse Reinforcement Learning and Deep Deterministic Policy Gradient are presented. The chapter discusses basic concepts and the work of artificial neurons, feedforward artificial neural networks, reinforcement learning and deterministic policy gradient.

The major motivation for discussing the artificial neural network is that in case of SSRL it is used as a approximation mechanism which is required for the reinforcement learning to learn well on continuous action spaces and in case of DDPG it is used as the actor and critic networks which also act as approximation mechanisms where the actor network gives the action for the given state and the critic network gives action values as output which estimate how good it is to take the action given from the actor network at a particular state. The major motivation for discussing reinforcement learning is that it is the major paradigm of machine learning which is employed on both the algorithms and the motivation for discussing deterministic policy gradient is that DDPG is built based on this concept.

2.1 Artificial Neural Networks

In this section an overview of artificial neural network is presented. This section discusses the artificial neuron and how it works, and this section also discusses a widely used algorithm which is the feedforward neural network. The concept of neural network was first proposed in 1943 (Mcculloch and Pitts, 1943). The model was specifically created as a computational model of the nerves in the brain.

2.1.1 Artificial Neuron

The structure of an artificial neuron is shown in Figure 2.1. *Artificial neuron* is a mathematical function which is elementary units in an artificial neural network. The neuron receives one or more inputs and does a suitable mathematical calculation to produce an output. The mathematical calculation differs from application to application. Each input is weighted and sent to a function called the *activation function* where the mathematical occurs.

Activation function defines the output of the node given an input. It can take different forms of linear or non-linear functions. For a given artificial neuron, consider $n + 1$ input which has input units ranging from x_0 to x_n and weights from w_0 through w_n. The weight shows the effectiveness of a particular input. More the weight, greater the impact on the network. Usually the x_n is assigned to a bias value $+1$ which makes it a bias input. The *bias value* is an additional parameter that is used to adjust the output along with the weighted sum of inputs to the artificial neuron. Therefore, the bias value is a constant that helps the neural network in a way that it can fit best for the given inputs. The output

O_k of the k^{th} neuron is

$$O_k = \Omega(\sum_{j=0}^{n} w_{kj} x_j)$$

where Ω is the activation function.

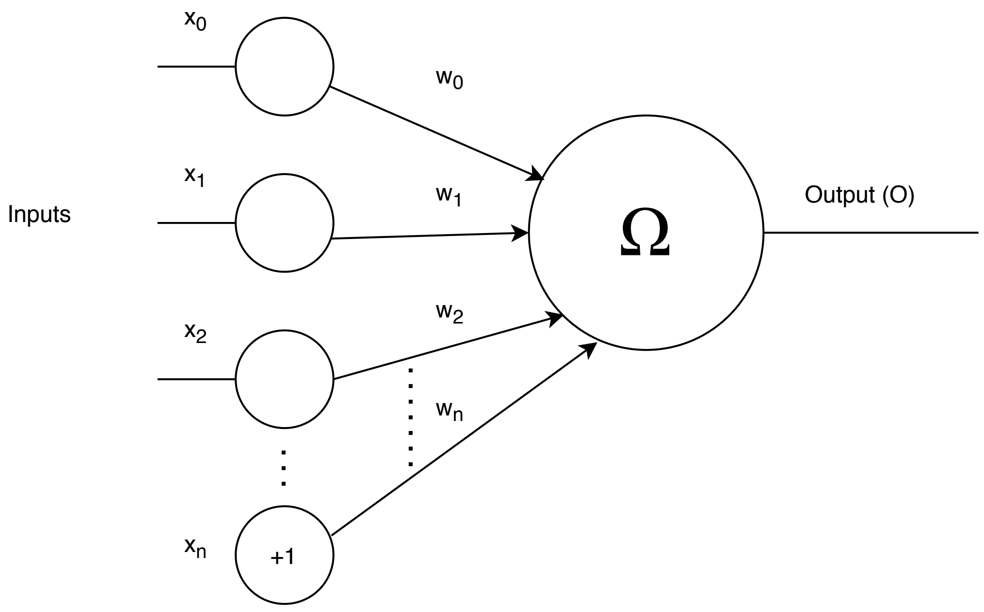


Figure 2.1: An Artificial Neuron

2.1.2 Feedforward Artificial neural Network

The structure of artificial neural network is shown in Figure 2.2. The *feedforward neural network* is a simple neural network types. It is a kind of neural network that uses supervised learning. *Supervised learning* is a machine learning task of learning a function that maps an input to output based on example input-output pairs. It was named feedforward neural network because the information moves in one direction that is in forward direction. It can move through the hidden layer that is present between the input and the output layer and it performs non-linear transformations of inputs and finally from the hidden layer to the output layer.

This type of neural network does not have any loops or cycles. The feedforward neural network is of different types based on the number of layers it has in the network. In this section the feedforward neural network is explained in terms of supervised learning because the concept of feedforward neural network was first introduced for supervised learning and the concept can be explained more clearly in terms of supervised learning. The neural network in terms of reinforcement learning is explained in Chapter 4.

The *single-layer perceptron network* consists of a single layer which has only the input nodes and the output nodes. There is no hidden layer in between. The inputs are fed to the outputs through a series of randomly initialized weights. These inputs and the weights are sent to the activation function. The advantages of single-layer perceptron are it is quite easy to set up and train. This perceptron works well for linearly separable data. The *multi-layer perceptron network* consists of hidden layers in between the input and the output layer. The weights are initialized between each layer and activation function is implemented in the hidden layers and the output layer. The advantages of multi-layer perceptron are that it has ability to learn how to do tasks based on data given for training or initial experience. Also, the multi-layer perceptron yields the required decision function directly via training. Also, the multi-layer perceptron with sufficient hidden nodes has been proven to be universal approximator (Hornik et al., 1989; Cybenko, 1989). The function can be same throughout or different at each layer. The working of feedforward neural network with single hidden layer (multi-layer perceptron) is shown as equations below.

The activation function $\Omega(x)$ (sigmoid function) is defined as

$$\Omega(x) = \frac{1}{1+e^{-x}}.$$

The reason for using *sigmoidal activation function* is that because of its non-linear nature. The non-linear function allows the model to create complex mappings between the network input and output. The sigmoidal function has a smooth gradient.

The interior value net_j of each unit j in the hidden layer is computed as

$$net_j = \sum_{i=1}^{I} x_i w_{ij}$$

where I is the total number of input units, i represents each unit in the input layer, x represents the inputs and w_{ij} represents the weights between the input layer and the hidden layer.

The value of each hidden unit H_j is calculated as

$$H_j = \Omega(net_j).$$

The interior value net_k of each unit k in the output layer is computed as

$$net_k = \sum_{j=1}^{J} H_j w_{jk}$$

where J is the total number of hidden units, j represents each unit in the hidden layer and w_{jk} represents the weight between the hidden layer and the output layer.

The value of each hidden unit O_k is calculated as

$$O_k = \Omega(net_k).$$

For the networks to learn, the method of *back propagation* is used. Here the actual output obtained from the output layer is compared with the target output and the error is identified. The gradient of the error function is calculated with respect to the neural network weights through the method *gradient descent*. This method is continued several times until the error between the actual output and the target output is small. This means that the network can predict well for different values in the future. The working of backpropagation with single hidden layer is shown as equations below.

The derivative (sigmoid derivative) $d(x)$ is calculated as

$$d(x) = \Omega(x)(1 - \Omega(x)).$$

The error in output units of the output layer $\delta_k^{(2)}$ is calculated as

$$\delta_k^{(2)} = O_k - Y_k.$$

Where Y is the target output, O is the actual output obtained from the network.

The change in the values of output units in output layer $\Delta_k^{(2)}$ is calculated as

$$\Delta_k^{(2)} = \delta_k^{(2)} d(O_k).$$

The error in hidden units of the hidden layer $\delta_j^{(1)}$ is calculated as

$$\delta_j^{(1)} = w_{jk} \Delta_k^{(2)}.$$

The change in the values of hidden units in a hidden layer $\Delta_j^{(1)}$ is calculated as

$$\Delta_j^{(1)} = \delta_j^{(1)} d(H_j).$$

The change in weights between hidden layer and the output layer Δw_{jk} is calculated as

$$\Delta w_{jk} = w_{jk} + (H_j \Delta_k^{(2)}).$$

The change in weights between input layer and the hidden layer Δw_{ij} is calculated as

$$\Delta w_{ij} = w_{ij} + (x_i \Delta_j^{(1)}).$$

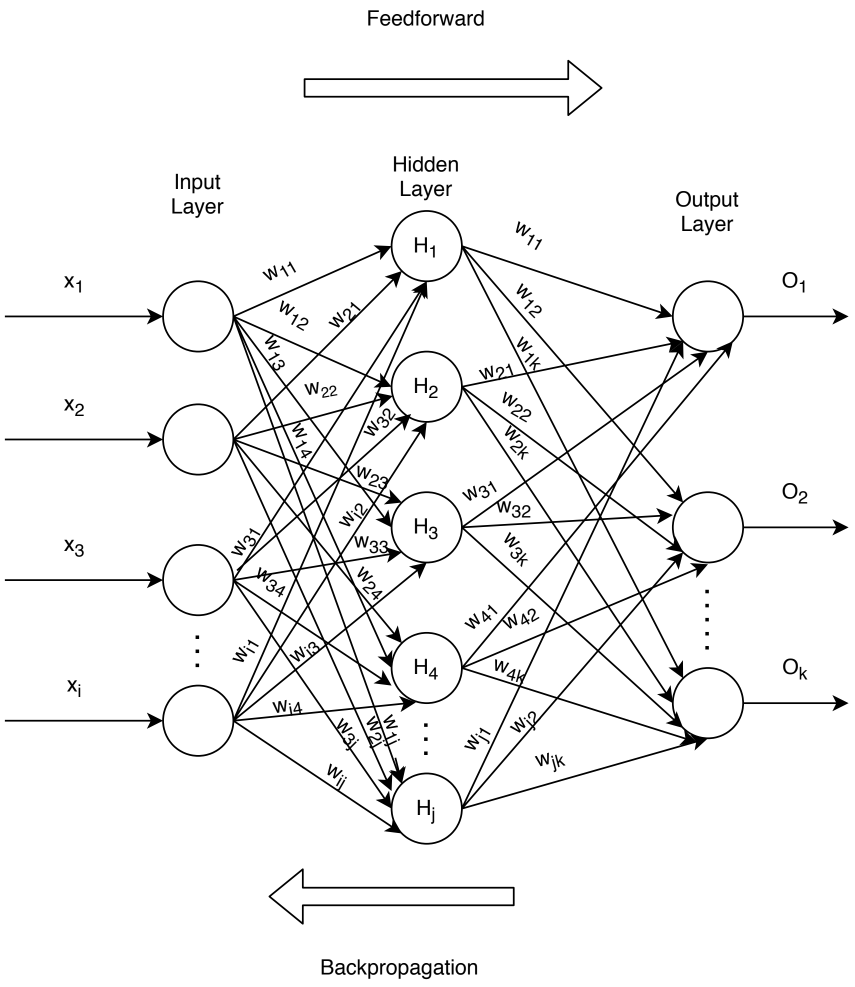


Figure 2.2: Artificial Neural Network

2.2 Reinforcement Learning

This section describes the basic concepts in reinforcement learning, the state and action value functions, Bellman's equation and Q–learning.

2.2.1 Basic Concepts in Reinforcement Learning

The structure on how reinforcement learning work is shown in Figure 2.3. *Reinforcement learning* is an area of machine learning; the goal of reinforcement learning is to train an agent to complete a task within a given environment. *States* are the representation of the world and *actions* are something the agent can do to change the states. The state and action spaces define the range of possible states the agent might perceive and the actions that are available to the agent. In this section the actions and spaces mostly discussed are discrete actions and states. This is because initially the concepts of reinforcement learning were applied for discrete actions and states and it is easy to explain in terms of discrete actions and states. Reinforcement learning for continuous states and actions is explained in Chapter 4. The *discrete environment* consists of finite states and actions. The agent receives observations and reward from the environment and sends actions to the environment. The *reward* is a measure of how successful the action is with respect to completing the task. The agent consists of two components namely the policy and reinforcement learning algorithm. The *policy* is a mapping that selects actions based on the observations from the environment. There are two types of policies namely on-policy and off–policy. An *off–policy learner* learns the value of optimal policy independently of the agent's actions, ensuring enough exploration. An *on–policy learner* learns the value of optimal policy being carried out by the agent.

The reinforcement learning algorithm continuously updates the policy based on the actions, observations, and rewards. The goal of the learning algorithm is to find an optimal policy that maximizes the cumulative reward received during the task. Based on the learning algorithm used, an agent maintains one or more parameterized function approximators for training the policy. The two types of function approximators are critics and actors. The critic finds the expected value of the long-term future reward for a task, given the observation and action. An actor finds the action that maximizes the long-term future reward, given the observation.

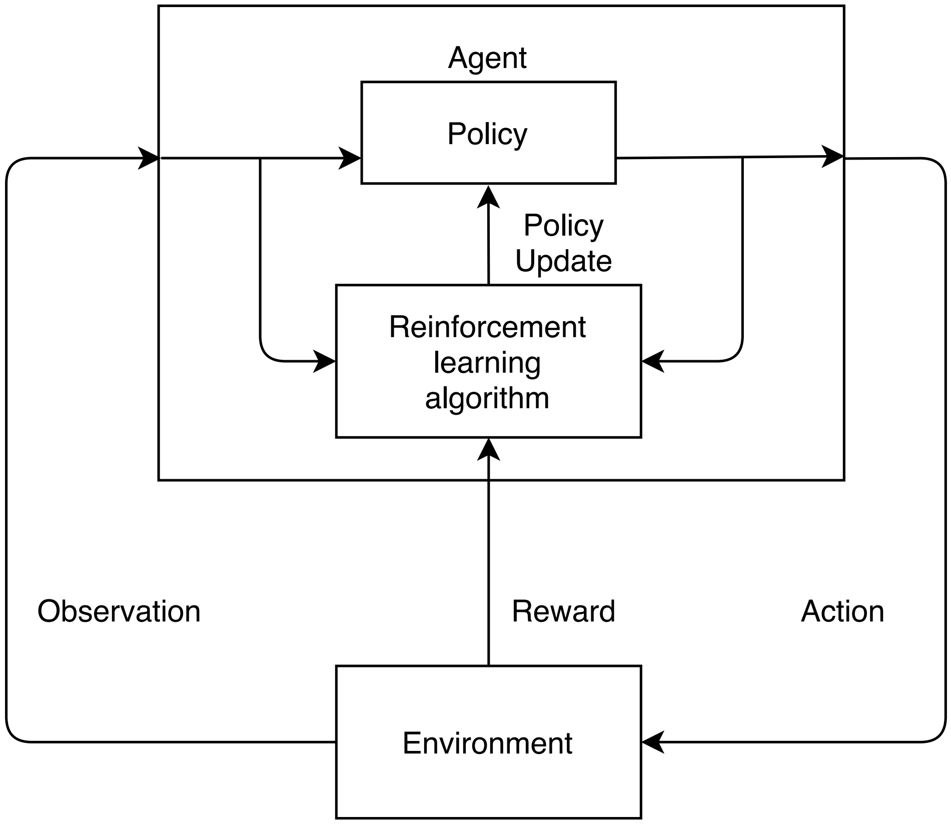


Figure 2.3: Reinforcement Learning

The input should be in the initial state from which the model would start.

There are many outputs as there are different types of solutions for a particular problem. The training is done based on the input and the model would return a state and reward based on the action. This way the model continues to learn and the solution with maximum reward is the best solution.

The difference in reinforcement learning compared to supervised learning is that in supervised learning the training data has the target output so that it can be used for further learning. On the other hand, reinforcement learning does not have a target output but the agent will decide what should be done in order to perform the task and it should learn from its experience. Reinforcement learning is used in environments where the only way to collect information about the environment is to interact with it.

2.2.2 State and Action Value Functions

Value functions are used in reinforcement learning to estimate how the agent can perform for a given action in a particular state. The performance is measured in terms of future rewards that can be expected. The rewards the agent can expect to receive in the future depend on the actions taken. The difference between the state and the action value function is that the *state value function* expresses the expected reward value of following the policy π forever when the agent starts following it from state s. Whereas the *action value function* expresses the expected reward value of first taking action a from state s and then following the policy π forever. The equations for state and action value functions are

$$v^{\pi}(s) = E_{\pi}(R_t | s_t = s)$$

where v^π is the state value function for policy π, E_π is the expected value for a given policy, R_t is the future discounted reward at any time step t.

$$q^\pi(s, a) = E_\pi(R_t|s_t = s, a_t = a)$$

where q^π is the action value function for policy π.

2.2.3 Bellman's Equation

The *Bellman's equation* is used to find the optimal policy. The Bellman's equation is given by the value of the given state $V(s)$ is equal to the action which maximizes the value $V(s)$ of the reward of optimal action a in a given state s, $R(s, a)$ and add a discount rate γ multiplied by the next state's value V(s').

$$V(s) = max_a(R(s, a) + \gamma V(s'))$$

This is Bellman's equation for *deterministic environments.* The environment is deterministic is when the agent knows certainly what the next state would be, given the current state and action taken. For a stochastic environment, the equation is different which is given below

$$V(s) = max_a(R(s, a) + \gamma \sum_{s'} P(s, a, s')V(s'))$$

In *stochastic environment* when an action is taken it is not certain that it will end up in a particular next state s'. So the probability $P(s, a, s')$ of ending up in a particular state s' from current s when action a is used in the equation. It is summed up to the total number of future states.

2.2.4 Q-Learning

Q-learning is an example of an off-policy reinforcement learning algorithm. The Q stands for quality in Q-learning, which represents how useful an action is in obtaining some reward. It seeks to find the best action to take given the current state. It is considered an off–policy because the Q-learning function learns from actions that are outside the policy.

The first step in performing the Q-learning algorithm is to create *Q-table* and the shape of the table is [number of states, number of actions] and all values in the table are initialized to zero. The update of the *Q-values* is done after each time the reward is received. This table acts as the reference table for the agent to select the action based on the Q–value.

The next step is to choose and perform an action. An agent interacts with the environment in one of the two ways. One is to use the Q-table as a reference and look at all actions for a given state. The agent selects the action corresponding to the maximum value in the table. This is called as *exploitation* since the agent uses the information available (Q-table) to decide (take an action). Another way is to take actions randomly. This is called *exploration.* Taking actions in random is important because it allows the agent to explore and discover new states which may not be selected during the process of exploitation. Sometimes the learning system with incomplete knowledge about the environment will be in a dilemma whether to select the action corresponding to the maximum value in the table (exploitation) or to select actions in random (exploration). This problem is called *exploration–exploitation trade-off.* The exploration and exploitation can be balanced by using *epsilon-greedy strategy.* In this strategy the *epsilon* (ϵ) refers to the probability of choosing to explore or exploit.

$$a(t) = \begin{cases} \text{max Q(a), with probability 1- } \epsilon \\ \text{any action(a), with probability } \epsilon \end{cases}$$

where a is the action and t is the time step.

The next step is to update the Q-values in the table. The equation to update the Q–values is given as

$$Q^{new}(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma \max Q'(s',a') - Q(s,a)]$$

Where $Q^{new}(s,a)$ is the new Q value for the corresponding state and action, $Q(s,a)$ is the current Q value, α is the learning rate, $R(s,a)$ is reward for taking action a at state s, γ is the discount rate, $\max Q'(s',a')$ is the maximum expected future reward given the new state s' and all possible actions in the new state. The discount rate affects how much weight should be given to the future rewards in the value function. If the discount rate is higher it will result in values representing the discounted future reward an agent expects to receive. The learning rate is used to control the speed at which the algorithm learns.

2.3 Artificial Neural Networks in Reinforcement Learning

All the concepts discussed in the section on reinforcement learning can work efficiently for discrete actions and discrete states. In continuous environment, the actions are infinite in number and states are also infinite. Previous studies (Shah and Hougen, 2017) note that traditional reinforcement learning approaches considers discrete actions. Because some approximation mechanism is required for reinforcement learning to work well in continuous action spaces, neural networks are used.

Typically neural networks are deterministic which means the output of the model is fully determined by the parameter values and for reinforcement learning to work efficiently in an environment it requires exploration. So, to provide exploration some random noise is added to the neural network policies. In the previous section it is discussed that for discrete action spaces exploration can be done using the method of epsilon-greedy. Adding random noise to the neural network is like epsilon-greedy method because both helps in exploration. But adding stochastic noise to neural network is complex compared to the epsilon-greedy method. The reasons are twofold. The first reason is the discrete action case where the number of possible actions is finite and on the other hand the continuous action where the number of possible actions is infinite. The second reason is that in discrete actions the similarities between the actions are assumed to be unknown, whereas it is not appropriate for continuous actions. Another issue that occurs in continuous spaces is that for attaining the reward multiple synapses may be responsible for it and such synapses should be recognized properly. This is called *structural credit assignment problem* (Gullapalli, 1992). The approach used in continuous action spaces is to sample random noise using the zero-mean normal distribution and add this noise to the continuous action value which is deterministically provided by the neural network.

Sometimes the environment will provide *delayed reward* instead of immediate reward, in which the agent must take series of actions before giving a reward. The problem with delayed reward is that it is hard to determine which action is responsible for the reward. This is called *the temporal credit assignment problem* (Sutton, 1984). The approach to consider those elements necessary for actions to be eligible for change by the process called *eligibility traces*. The eligibility traces keep track of which parameters have been mostly updated in addition to the states which are most recently updated.

2.3.1 Mini-Batch Learning and Replay Buffer

In *batch learning* the total set of state and action pairs are considered. The major task in batch learning is to find a policy that maximizes the sum of expected rewards in the agent-environment loop (Lange et al., 2012). In Hafner and Riedmiller (2011), where neural networks which are used as functional approximators, batch learning was used to bring stability but it became hard to control for large neural networks because of the number of values in the batch was too large to handle. In *mini–batch learning* a small set of states and actions are taken rather taking all the states and actions for learning. The major advantage of using mini-batch learning is for quick learning when the function approximators used in reinforcement learning are large neural networks.

One of the major challenges in using neural networks on reinforcement learning is that in most optimization algorithms it is generally assumed that the samples are independently and identically distributed. But when the samples are generated from exploring sequentially in an environment this assumption does not hold in general. Additionally, to make efficient use of hardware optimizations like increasing scalability in memory system the mini-batch learning is used.

To solve the above issue the *replay buffer* is used; the replay buffer is a finite-sized cache. Transitions are sampled from the environment according to the exploration policy and the tuple (s_t, a_t, r_t, s'_t) are stored in the replay buffer R. The s_t is the state at time step t, at is the corresponding action, r_t is the reward receive on taking the action, and s'_t is the next state. When the replay buffer is full the old tuples are discarded.

2.4 Deterministic Policy Gradient

The *deterministic policy gradient* is the expected gradient of the action value function. It was previously believed that deterministic policy gradient did not exist, but it was shown in Silver et al. (2014), that the deterministic policy gradient is possible and that it follows the gradient of action–value function.

The major difference between the stochastic policy gradient and the deterministic policy gradient is in case of *stochastic policy gradient*, the policy gradients integrated over both state and the action spaces whereas in the case of deterministic it only integrates over the state space. As a result, computing the stochastic policy gradient may require more samples, especially when the action space is continuous.

The deterministic policy gradient is to derive an off–policy actor–critic algorithm that calculates the action-value function using a functional approximator and then update the policy in the direction of approximate action–value gradient. The deterministic policy gradient is calculated as

$$\nabla_\theta J_\beta(\pi_\theta) = E_s = \rho^\beta[\nabla_\theta \pi_\theta(s) \nabla_a Q^\pi(s, a)|a = \pi_\theta(s)]$$

where $E_s = \rho^\beta$ is the expected value with respect to the discounted state distribution ρ^β, $\nabla_\theta J_\beta(\pi_\theta)$ indicates the change in performance gradient with respect to the parameter θ and $\theta \in R^n$, $\nabla_\theta \pi_\theta(s)$ indicates the change in target policy $\pi_\theta(s)$ with respect to the parameter θ, $\nabla_a Q^\pi(s, a)$ indicates the change in action value pair $Q^\pi(s, a)$ with respect to action a.

Chapter 3

Literature Survey

This chapter is divided into two sections, one focuses on literature related to SSRL and the other focuses on literature related to DDPG. This chapter is divided into two sections because each algorithm has different previous works and concepts that should be discussed.

3.1 Literature Survey for SSRL

Previous studies have introduced a novel algorithm (Williams, 1992) called the REINFORCE (Reward Increment Non-negative Factor Offset Reinforcement Characteristic Eligibility) algorithm which represents a prescription for devising statistical gradient algorithms for reinforcement learning neural networks. Also, because it is a gradient based approach, it integrates with other gradient computation techniques like backpropagation. The method introduces stochasticity at the level of the neural unit. This novel algorithm works well for observation spaces and action spaces but for continuous action and observation the reinforcement learning faces lots of challenges (Kober et al., 2013). So, to work well on high-dimensional action spaces novel algorithm called Stochastic Synapse Reinforcement Learning (SSRL) (Shah and Hougen, 2017). This paper was built from previous studies using stochastic approaches to reinforcement learning in artificial neural network for continuous values functions (Gullapalli, 1990; Williams, 1992). In this paper, an approximation mechanism is required for the reinforcement learning to work well in continuous action spaces, so neural networks are used. The foundational work in this general paradigm (Gullapalli, 1990; Williams, 1992) uses stochastic units whereas SSRL uses stochastic synapses and the research by Shah and Hougen (2020) shows that stochastic synapses tend to work better that stochastic units for various tasks.

Artificial neural networks are distributed processing systems that contains huge numbers of simple processing nodes connected in massively parallel structures (Jain et al., 1996). The network consists of neuron models interconnected using artificial synapses. The weights which acts as the strength of these synapses is used to store knowledge (Sandberg et al., 2001). Reinforcement learning allows the agent to learn a policy of appropriate actions for its environment through direct experience, provided there is some mechanism through which the agent receives feedback (Sutton and Barto, 1998).

While working with continuous action spaces the algorithm may encounter a certain problem. For attaining the reward multiple synapses may be responsible for it and such synapses should be recognized properly. This is called structural credit assignment problem (Gullapalli, 1992). One approach used in continuous action spaces is to sample random noise using a zero-mean normal distribution and add this noise to the continuous action value which is deterministically provided by the neural network.

Sometimes the environment will provide delayed reward instead of the immediate reward, in which the agent must take series of actions before giving a reward. The problem with delayed reward is that it is hard to determine which action is responsible for the reward. This is called the temporal credit assignment problem (Sutton, 1984). The approach to consider those elements necessary for actions to be eligible for change by the process called eligibility traces. Eligibility traces keep track of which parameters have been mostly updated in addition to the states that is most recently updated. SSRL considers both these problems and follows the above approaches to deal with the problem.

3.2 Literature Survey for DDPG

The goal in the field of artificial intelligence is to solve complex tasks from high dimensional sensory input. Significant progress has been made by combining advances in deep learning (like deep convolutional neural networks) for sensory processing (Krizhevsky et al., 2012) with reinforcement learning resulting in deep Q-network (Mnih et al., 2015, 2013) which was able to give human level performance on Atari video games using unprocessed raw pixels as input. But the problem with deep Q-network is that it solves problems of high-dimensional observation spaces and low-dimensional action spaces, but in case of high-dimensional action spaces the deep Q-network was not able to be directly applied because this algorithm relies on finding the action that maximizes the action value function, which for continuous value space requires optimization process at every time step.

So, for learning effective policies on continuous action spaces an off-policy, actor-critic algorithm using deep function approximators called the Deep Deterministic Policy Gradient (DDPG) is used (Lillicrap et al., 2015). This algorithm is based on the deterministic policy gradient (Silver et al., 2014). The deterministic policy gradient is the expected gradient of the action value function. It was previously believed that the deterministic policy gradient did not exist;however, it was shown in Silver et al. (2014), that the deterministic policy gradient is possible and that it follows the gradient of the action–value function. The deterministic policy gradient is used to derive an off–policy actor–critic algorithm that calculates the action- value function using a functional approximator and then updates the policy in the direction of the approximate action–value gradient.

To implement this algorithm an actor and critic approach is used. It maintains an actor for specifying the current policy by deterministically mapping state to action. The critic is learned using Bellman's equation. The actor is updated by using the chain rule to expected return to start distribution using actor parameter. The actor and the critic each use their own neural network which acts as an approximation mechanism. In previous studies (Hafner and Riedmiller, 2011), where neural networks are used as functional approximators batch learning was used to bring stability but it became hard to control for large neural networks because of the number of values in the batch was too large to handle. So, in this algorithm mini-batch learning is used so that when the policy is updated only a subset of the data is used at each update which is used to scale large neural networks. In mini–batch learning a small set of states and actions are taken rather than taking all the states and actions for learning. The major advantage of using mini-batch learning is for quick learning when the function approximators used in reinforcement learning are large neural networks. One of the major challenges in using neural networks on reinforcement learning is that for most optimization algorithms it is generally assumed that the samples are independently and identically distributed. But when the samples are generated from exploring sequentially in an environment this assumption does not hold in general. So, it is important for the mini-batch learning to be used. To solve the above issue, the replay buffer (Mnih et al., 2015, 2013) which was used in Deep Q-network is used. The replay buffer is a finite-sized cache. The transitions are sampled from the environment according to the exploration policy and the tuples are stored in the replay buffer. When the replay buffer is full the old tuples are discarded.

Sometimes when learning for low-dimensional feature vector observations, the observations may have different units. This will be difficult for the network to learn effectively and have a problem in generalizing the hyper-parameters. The approach to solve the problem is to scale the features so that are of similar ranges across environment and units. This approach is called batch normalization (Ioffe and Szegedy, 2015). This approach normalizes each dimension across the samples in a mini-batch to have unit mean and variance.

Chapter 4

Experimental Setup

In this chapter the Stochastic Synapse Reinforcement Learning (SSRL) algorithm and the Deep Reinforcement Policy Gradient (DDPG) algorithm, are explained step by step. Another section describes the environments used for the experiment and an overview of the experimental conditions used for the experiment and the results received.

4.1 Stochastic Synapse Reinforcement Learning

This section describes the neural network structure used in Stochastic Synapse Reinforcement Learning and discusses the step by step algorithm of the process.

4.1.1 Neural Network Structure

Here we consider the standard feedforward, fully connected networks with summation units and sigmoid activation functions which is shown in Figure 4.1 across the hidden layer and the output layer. To learn more complex relationships in the environment an intermediate layer or hidden layer is added. Only the output layer of the network is trained by using Stochastic Synapse Reinforcement Learning, whereas the previous layers of the network are trained by the method of backpropagation where the weight adjustments from the output layer through the hidden layer. The bias value +1 is added to the input and the hidden layer. The bias value is an additional parameter that is used to adjust the output along with the weighted sum of inputs to the artificial neuron. Therefore, the bias value is a constant that helps the neural network in a way that it can fit best for the given inputs. The reason for using neural networks in reinforcement learning is that previous studies (Shah and Hougen, 2017, 2020) note that traditional reinforcement learning approaches considers discrete actions. Because some approximation mechanism is required for the reinforcement learning to work well in continuous action spaces, neural networks are used.

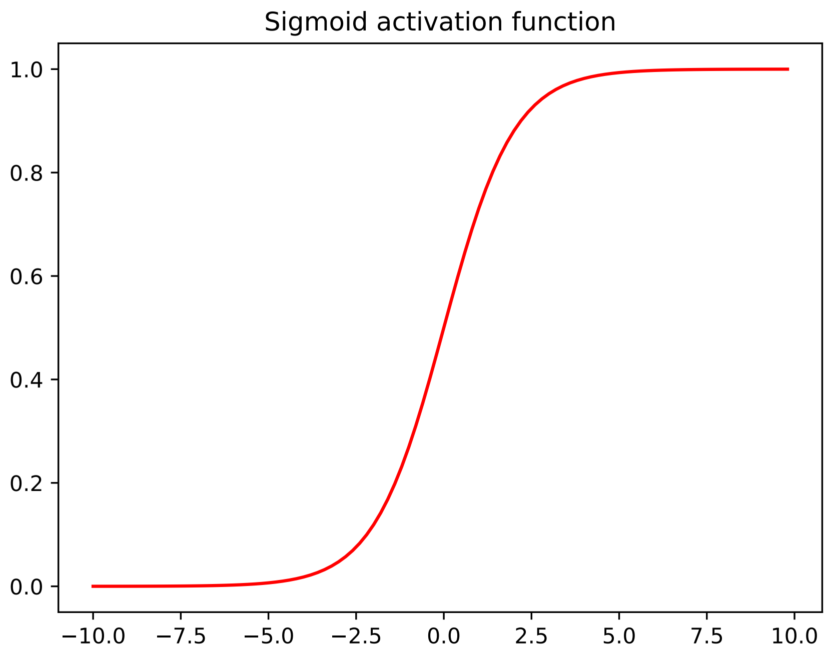


Figure 4.1: Sigmoid Function

4.1.2 Algorithm

The weights of the synapses between the input and the hidden layer and the synapses between hidden and the output layer are initialized between -2 to 2. The standard deviation is initialized uniformly randomly between 0 and 1.

For each time step, Stochastic Synapse Reinforcement Learning carries out the following steps.

The first step is to normalize the input, which is the observation vector. The input is normalized between 0 and 1 inclusive. The reason for normalizing the input is that the algorithm is built in such a way that it takes it values between 0 and 1. The reasons the algorithm is built this way are, having all inputs in the vector in the same range to ensure that none is inherently over-or undervalued due to having a greater range and another reason is that positive and negative values should be reserved for opposites so as not to encode false contrasts in the input; because there are no opposite meanings inherent in the inputs, they were all coded with same sign. So, in order for the algorithm to work in environments having continuous actions and continuous observations that have different ranges of values, they are normalized using

$$x_i = \frac{ob_i - ob_i^{(l)}}{ob_i^{(h)} - ob_i^{(l)}}$$

where ob_i is the observation input, $ob_i^{(l)}$ is the minimum observation value, $ob_i^{(h)}$ is the maximum observation value, x_i is the normalized inputs and i represents each unit in the input layer.

The next step is to find the net interior value for each unit in the hidden layer.

The interior value net_j of each unit j in the hidden layer is computed as

$$net_j = \sum_{i=1}^{I} x_i w_{ij}$$

where I is the total number of input units, i represents each unit in the input layer, x represents the inputs and w_{ij} represents the weight between the input layer and the hidden layer.

The next step is to find the value of the hidden unit. The value of each hidden unit H_j is calculated as

$$H_j = \Omega(net_j).$$

The activation function $\Omega(f)$ (sigmoid function) is defined as

$$\Omega(f) = \frac{1}{1+e^{-f}}.$$

The next step is to find the net interior value for each unit in the output layer. The interior value net_k of each unit k in the output layer is computed as

$$net_k = \sum_{j=1}^{J} \frac{H_j(w_{jk} + \epsilon_{jk})}{J}$$

where J is the total number of hidden units, j represents each unit in the hidden layer, w_{jk} represents the weight between the hidden layer and the output layer and ϵ_{jk} is the noise. The purpose for adding noise to the weights is to introduce stochasticity at the level of the synapse and because the final layer is trained to reinforcement learning the noise is added to the weights of each synapse between the hidden and output layers.

The noise is calculated by the sampling from the normal distribution. The noise is given by

$$\epsilon_{jk} = \psi(\mu_{jk}, \sigma_{jk})$$

where μ_{jk} is the mean for each synapse between the hidden and the output layer and it is always set to 0, σ_{jk} is the standard deviation for each synapse between the hidden and the output layers. The noise is sampled from a normal distribution because the normal distribution is able to generate a wide range of random values that can be used to bring stochasticity to the synapse level. The purpose of uniformly setting the mean to be 0, is so that the value of each synapse is determined by its weight plus the stochastic noise times the input value on that connection. Then the weight is updated directly, which is equivalent to leaving weight unchanged and updating mean.

The next step is to find the value of each output unit. The value for each output unit O_k is calculated as

$$O_k = \Omega(net_k).$$

The next step is to find the *eligibility weight traces* E_w which is calculated as

$$E_w^{new} = E_w^{old} + e_w(k) d_w^{(t-k)}$$

where $e_w(k)$ is the eligibility of the synaptic weight at time step k and t denotes the total time steps in a trial and d_w is the discount factor and it is set to 0.5. The idea behind calculating the weight eligibility traces is that it considers the past actions in the observation and action spaces while calculating the change in weights between the hidden and the output layer in order to determine the degree of responsibility of each synapse on the action chosen and, consequently, on the reward received. This helps to address both the structural and temporal credit assignment problems.

The $e_w(k)$ is calculated as

$$e_w(k) = H_j \epsilon_{jk}$$

where H_j is the output value of hidden node j and ϵ_{jk} is the noise. This equation is used to find those synapses whose noise and the hidden unit are more active which might be eligible for the adjustment of the synaptic weights between the hidden and the output layer.

The next step is to find the *eligibility standard deviation traces* E_σ which is calculated as

$$E_\sigma^{new} = E_\sigma^{old} + e_\sigma(k) d_\sigma^{(t-k)}$$

where $e_\sigma(k)$ is the eligibility of the synaptic weight standard deviation at time step k and t denotes the total time steps in a trial and d_σ is the discount factor and it is set to 0.5. The idea behind the adjustment in the standard deviation at synaptic level is that it acts as a control knob for the exploration and exploitation trade-off by considering the past actions on observation and action space. These past actions are considered by calculating the eligibility standard deviation traces E_σ.

The $e_\sigma(k)$ is calculated as

$$e_\sigma(k) = H_j(|\epsilon_{jk}| - \sigma_{jk})$$

where H_j is the output of hidden node j, ϵ_{jk} is the noise and σ_{jk} is the standard deviation. The purpose for using this equation is to identify the exploration eligibilities of a given synapse. The absolute value of noise is taken because σ determines the magnitude of the noise, so it tries to determine whether to increase or decrease the magnitude of the noise. So, the absolute value of ϵ is taken, which gives the magnitude of it. Then, it is subtracted from σ to consider whether the magnitude of the sampled noise was larger or smaller than the standard deviation of the noise. If the magnitude of ϵ was larger than σ, then the subtraction will give a positive value to add to the eligibility e_σ. If the magnitude of ϵ was smaller than σ, then the subtraction will give a negative value to add to the eligibility e_σ.

The above steps are to be executed for each time step because the above equations need to be updated at each time step. Now, at the end of each trial, Stochastic Synapse Reinforcement Learning carries out the following steps.

The first step is to update the weights between the hidden layer and the output layer. The change in weights between the hidden layer and the output layer Δw_{jk} is calculated as

$$\Delta w_{jk}(\tau) = \sum_{j=1}^{J} \eta_w (r(\tau) - r'(\tau)) E_w$$

where η_w is the learning rate and the value is kept as 0.5, $r(\tau)$ is the reward collected at a trial τ, $r'(\tau)$ is the expected reward at a trial τ, and E_w is the weight eligibility traces. The reward collected is calculated at the end of each trial and the equations for calculating the reward are given in Section 4.3 for different environments. This equation is used to update the weights between the hidden and the output layer thereby helping the algorithm to learn better in next coming trials. The intuition behind this updating is that the weight is adjusted in the direction of the noise. This is true only when the reward received is greater than the expected reward. If the reward received is less than the expected reward, then the weight is updated in the opposite direction of the noise. This is because if the reward is lower than expected, the noise was probably moving the action in the wrong direction.

The expected reward $r'(\tau+1)$ for a trial $\tau+1$ is calculated as

$$r'(\tau+1) = dr(\tau) + (1-d)r'(\tau)$$

where $r(\tau)$ is the reward collected at the current trial τ, $r'(\tau)$ is the expected reward at the current trial τ and d is the discount factor, and it is set to 0.5. The purpose of expected reward is to determine whether the actual reward received is higher than expected, therefore likely to be the result of actions that are more appropriate than those it had been trying.

The next step is to change the standard deviation. The change in standard deviation $\Delta\sigma_{jk}$ is calculated as

$$\Delta\sigma_{jk}(\tau) = \sum_{j=1}^{J} \eta_{\sigma}(r(\tau) - r'(\tau))E_{\sigma}$$

where η_{σ} is the learning rate and the value is kept as 0.5, $r(\tau)$ is the reward collected at a trial τ, $r'(\tau)$ is the expected reward at a trial τ and E_{σ} is the standard deviation eligibility traces. The intuition behind adjusting the standard deviation is that adjusting standard deviation opposite to the magnitude of noise. If the noise is small, then standard deviation is made larger which means more exploration and if the noise is large then standard deviation is small which means more exploitation. This true only when the reward received is less than expected reward, in which case the agent should probably try the opposite of what it tried previously. If the reward received is larger than expected reward, the agent should try the same thing it tried previously, so if the noise was small then the standard deviation is reduced, whereas if the noise is large then the standard deviation is increased.

The next step is to calculate the change in weights between the input and the hidden layer. To update the weights between the input and the hidden layer backpropagation is used. The change in weights between the input layer and the hidden layer Δw_{ij} is calculated as

$$\Delta w_{ij}(\tau) = w_{ij} + (x_i \Delta_j^{(1)})$$

where w_{ij} is a weight between the input and the hidden layer, $\Delta_j^{(1)}$ is the change in the values of hidden units in a hidden layer and τ are the trials. This equation is used to update the weights between the input and the hidden layer thereby helping the algorithm to learn better in next coming trials.

The change in the values of hidden units in a hidden layer $\Delta_j^{(1)}$ is calculated as

$$\Delta_j^{(1)}(\tau) = \delta_j^{(1)} d(H_j)$$

where $\delta_j^{(1)}$ is the error in units of the hidden layer, $d(H_j)$ is the derivative of the units in the hidden layer. The purpose of this equation is to update the values of the hidden units which will be in turn helpful for updating weights between the input and the hidden layer.

The derivative (sigmoid derivative) $d(x)$ is calculated as

$$d(x) = \Omega(x)(1 - \Omega(x))$$

The purpose for calculating the derivative is that the derivative gives a gradient that shows the neuron the direction and approximate magnitude to adjust given a particular input.

The error in units of the hidden layer $\delta_j^{(1)}$ is calculated as

$$\delta_j^{(1)}(\tau) = w_{jk}\Delta_k^{(2)}$$

where $\Delta_k^{(2)}$ denotes the change in the values of output units in output layer, w_{jk} is the weights between the hidden layer and the output layer. The purpose of calculating the error is to give an idea of how much the model needs to change the weights to minimize the error.

The change in the values of output units in output layer $\Delta_k^{(2)}$ is calculated as

$$\Delta_k^{(2)}(\tau) = (r(\tau) - r^{|}(\tau))E_w d(O_k)$$

where $d(O_k)$ is the derivative of the output units in the output layer. The purpose of this equation is to update the values of the output units which will be in turn helpful for identifying the error in the hidden units.

4.2 Deep Deterministic Policy Gradient

This section describes the structure of the actor and critic networks, the replay buffer, and the step by step algorithm of Deep Deterministic Policy Gradient (DDPG).

4.2.1 Actor and Critic Networks

It is not possible to directly apply Q-learning on continuous action spaces, because in continuous spaces finding a greedy policy requires optimization of the action at every time step which is practically too slow because regardless of the number of dimensions, there are infinite possible actions. So, this is the reason why actor and the critic networks are used.

The critic network is built as follows. The input layer is the observation vector and the number of units in the input layer is equal to the number of observations. The first hidden layer is set to 400 hidden units. The activation function used is the ReLU (Rectified Linear Units) activation function which is shown in Figure 4.2. The motivation behind using this activation function is that it is non-linear function and a non-linear function allows the model to create complex mappings between the network input and output, it is less computationally expensive than tanh, sigmoid and logistic activation functions, it works well on hidden layers because it can produce dead neurons which means when the value of input are negative it will output into true zero value which can accelerate learning and simplify the model. The equation for the ReLU activation function is defined as

$$F(x) = \max(0, x).$$

The second hidden layer is set to 300 hidden units. The activation function used is ReLU. The motivation behind using this activation function is that it is non-linear function and a non-linear function allows the model to create complex mappings between the network input and output, it is less computationally expensive than tanh, sigmoid and logistic activation functions, it works well on hidden layers because it can produce dead neurons which means when the value of input are negative it will output into true zero value which can accelerate learning and simplify the model.. The output layer consists of one unit. The activation function used is the linear activation function which is shown in Figure 4.3. The motivation for using linear activation is that since it is a regression model where the output are dependent on the input received by the critic network. So, the linear function is used in the output layer of the network. The equation for the linear activation function is defined as

$$F(x) = cx$$

where c is a constant.

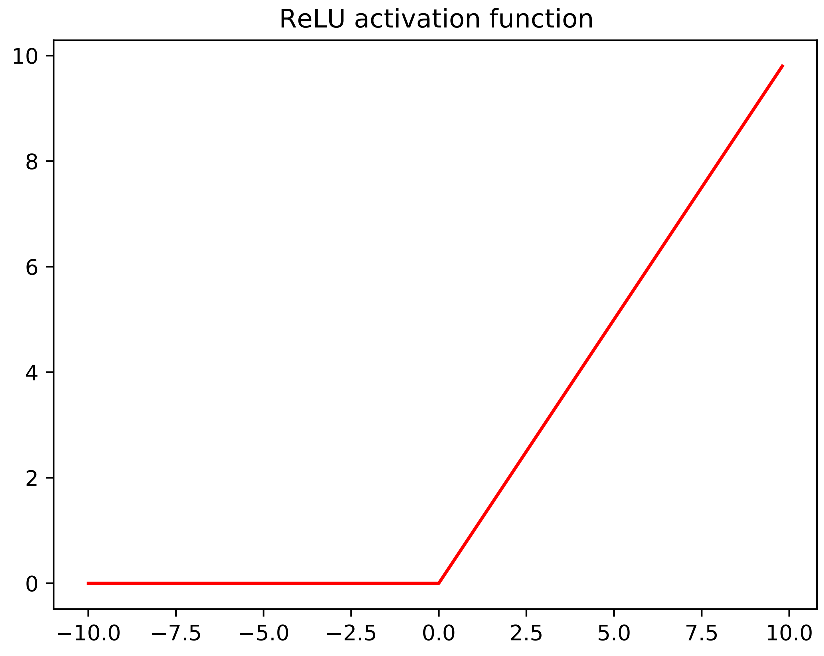


Figure 4.2: ReLU Function

The actor network is built as follows. The input layer is the observation vector

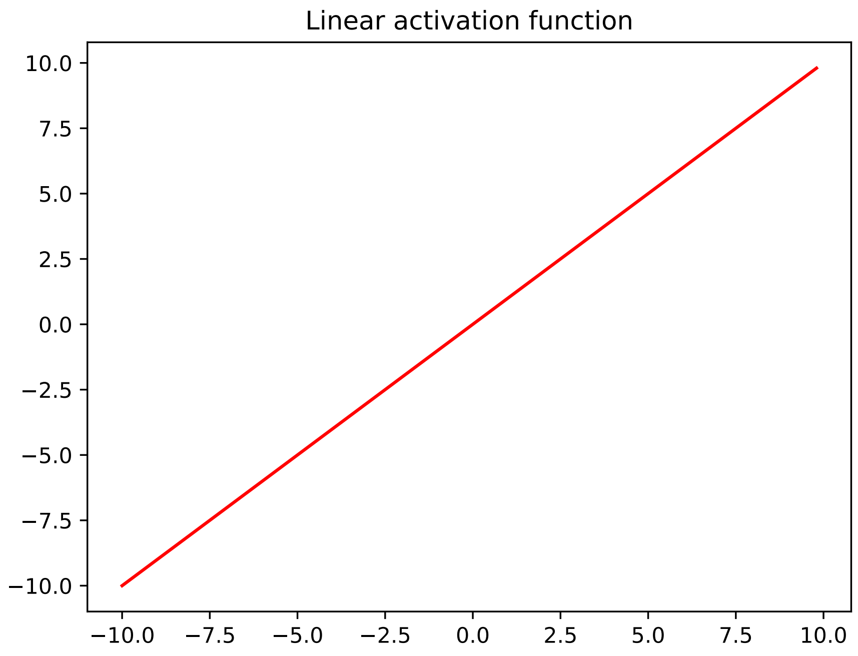


Figure 4.3: Linear Function

and the number of units in the input layer is equal to the number of observations. The first hidden layer is set to 400 hidden units. The activation function used is the ReLU activation function. The second hidden layer is set to 300 hidden units. The activation function is the ReLU activation function. The output layer consists of number of units equal to number of actions. The activation function used between the second hidden layer and the output layer is the hyperbolic tangent (tanh) activation function as it is shown in Figure 4.4. The motivation for using tanh function is that it is non-linear and a non-linear function allows the model to create complex mappings between the network input and output and this function is zero-centered. The equation for the tanh activation function is defined as

$$F(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

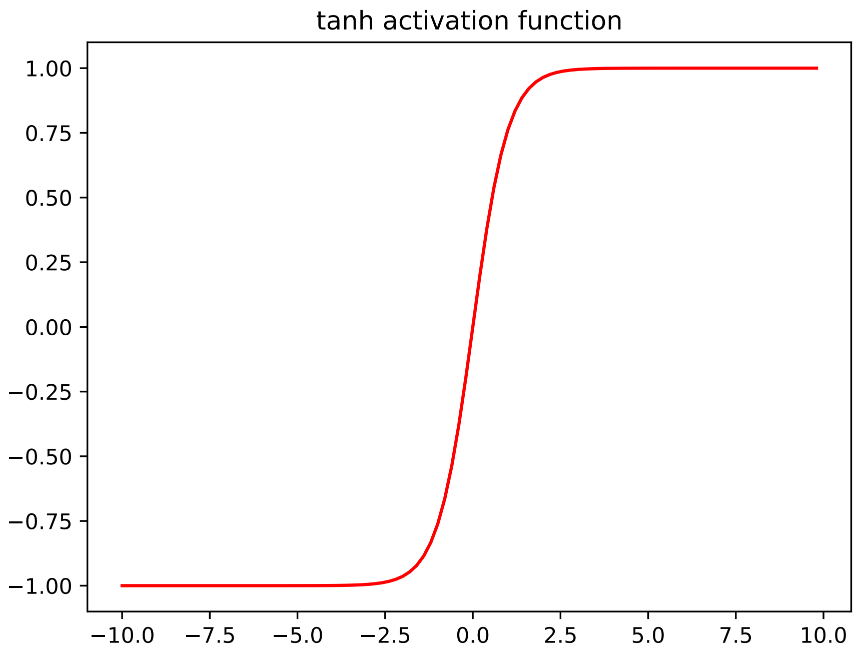


Figure 4.4: Tanh Function

The final layer weights for both actor and critic network are initialized from a uniform distribution [-0.003, 0.003]. The other layers were initialized from a uniform distribution for both the actor and critic networks in the range of $[\frac{-1}{\sqrt{f}}, \frac{1}{\sqrt{f}}]$ where f is the fan-in of the layer. The weights are initialized based on work in Lillicrap et al. (2015).

4.2.2 Replay Buffer

One of the major challenges in using neural networks for reinforcement learning is that generally it is assumed that the samples are independently and identically distributed. But when the samples are generated from exploring sequentially in an environment this assumption does not hold. And, to make efficient use of hardware optimizations mini-batch learning is used because in mini-batches the memory consumed is less which optimizes the hardware memory.

To solve the above issue the replay buffer is used. The transitions are sampled from the environment according to the exploration policy and the tuple (s_t, a_t, r_t, s'_t) are stored in the replay buffer R, s_t is the state at time step t, a_t is the corresponding action, r_t is the reward received on taking the action and s'_t is the next state. When the replay buffer is full the old tuples are discarded. The buffer size kept in this experiment is 10^6. At each time step t the actor and critic are updated by randomly sampling a mini-batch of transitions uniformly from the buffer. The batch size here is set as 64.

4.2.3 Algorithm

The first step is to initialize the critic network $Q(s, a|w^Q)$ where s is the state and w^Q denotes the weights for the critic network and the actor network $\mu(s|w^\mu)$ where w^μ denotes the weights for the actor. The target networks are also initialized which are Q' and μ' respectively. Initially, the weights of target networks $w^{Q'}$ and $w^{\mu'}$ are initialized to w^Q and w^μ respectively. The next step is to initialize the replay buffer R.

For each trial, the following steps are carried out. The noise process N is initialized. The noise process used is temporally correlated noise and it is executed through a process known as the *Ornstein-Uhlenbeck process* (Uhlenbeck and Ornstein, 1930). This process is used for exploring better in the environments. The noise process is calculated as

$$N_e = \sum_{i=1}^{e}(-\theta N_e dt) + \sigma\sqrt{dt}$$

where N_e is the noise, e is the number of trials, dt is the change in time and it is set to e^{-2} , θ is the mean and it is set to 0.15 and σ is the volatility and it is set to 0.3.

For each time step, the following steps are carried out. The action is selected according to the policy and the noise process. One of the major challenges in continuous action spaces is to explore effectively. One of the uses of an off-policy algorithm like the Deep Deterministic Policy Gradient is that the problem of exploration can be considered independently from the learning algorithm. So, an exploration policy a_t is constructed. It is calculated as

$$a_t = \mu(s_t|w^{\mu}) + N$$

where $\mu(s_t|w^{\mu})$ is the actor policy and N is the noise. Based on the action the reward r_t is observed and the next state s'_t is also observed. Now the transition, which is a tuple (s_t, a_t, r_t, s'_t), is stored in the replay buffer R. Now a random mini-batch of size 64 of the transitions (s_i, a_i, r_i, s'_i) are sampled from the replay buffer R. Now the critic is updated by minimizing the loss. The loss function is calculated as

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|w^Q))^2$$

where $Q(s_i, a_i|w^Q)$ is the critic network with state s_i and actor a_i, i represents the transitions in the mini batch and N represents the number of transitions in the mini-batch, the target y_i is set as

$$y_i = r_i + \gamma Q'(s'_i, \mu'(s'_i|w^{\mu'})|w^{Q'})$$

where $Q'(s'_i, \mu'(s'_i|w^{\mu'})|w^{Q'})$ represents the target critic network with next state s'_i in transition i and $w^{Q'}$ is the weight of the target critic network, $\mu'(s'_i|w^{\mu'})$ is the target actor with next state s'_i in transition i and $w^{Q'}$ represents weights of the target actor, γ represents the discount factor and parameter is set to 0.99 and r_i represents the reward in transition i. This target y_i is a stable target that is used to consistently train the critic without divergence. Now the actor policy is updated using the sampled gradient. It is calculated as

$$\nabla_{w^\mu}\mu|s_i = \frac{1}{N}\sum_i \nabla_a Q(s_i, \mu(s_i)|w^Q)\nabla_{w^\mu}\mu(s_i|w^\mu)$$

where $\nabla_a Q(s_i, \mu(s_i)|w^Q)$ represents the change in critic with respect to the actor and $\nabla_{w^\mu}\mu(s_i|w^\mu)$ represents the change in actor with respect to the weight of the actor. The final step is to update the target weights. They are updated using

$$w^{Q'} = \alpha w^Q + (1-\alpha)w^{Q'}$$

$$w^{\mu'} = \alpha w^\mu + (1-\alpha)w^{\mu'}$$

where α is set to 0.001. The α value is always set very less than 1 because this makes the target values to change slowly, improving the stability of learning.

4.3 Open AI Gym

Open AI gym is a Python library that provides a large number of test environments to work on reinforcement learning algorithms with interfaces for writing new reinforcement algorithms and testing their behaviour on the environment. The environments discussed in this section are *mountain car continuous* and *delayed-reinforcement pendulum.*

4.3.1 Mountain Car Continuous Environment

The *mountain car continuous environment* is shown in the Figure 4.5. An under-powered car must climb a one–dimensional hill to reach a target. In this environment the actions are continuous values. The target is on the top of a hill on the right-hand side of the car. If the car reaches the top or goes beyond the episode terminates.

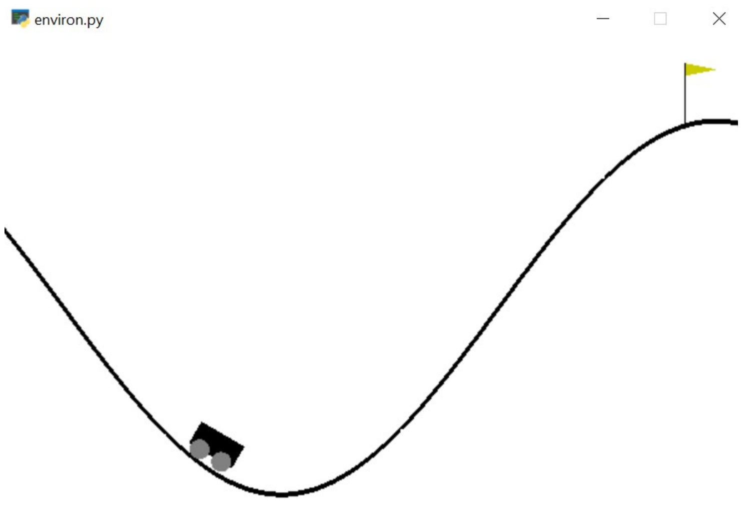


Figure 4.5: Mountain Car Continuous Environment

The observation consists of two values, one is the velocity of the car and the other is the position of the car. The minimum value and the maximum value of the car position are -1.2 and 0.6, respectively. The minimum value and the maximum value of the car velocity are -0.07 and 0.07, respectively. The velocity has been constrained to require exploration. The action has a positive value when the car moves to the right or the action has a negative value when car moves to the left. The minimum and the maximum value of the action is -1.0 and 1.0 respectively. The reward is calculated as 100 minus the squared sum of actions from start to goal. The reward function raises the exploration challenge, because if the agent does not reach the goal sooner than expected, it will find out that it is better not to move and it will stop finding the target any further.

4.3.2 Delayed-Reinforcement Pendulum Environment

The *delayed-reinforcement pendulum environment* is shown in the Figure 4.6. The goal of this problem is to swing a frictionless pendulum upright, so that it stays vertical pointed upwards. The version of pendulum problem used in this work is a non-standard version that is likely much harder than the standard pendulum problem. This is because the standard pendulum problem gives reward at each time step, but this version of the pendulum problem accumulates the reward till the end of the episode and only the total accumulated reward is given to the algorithm at the end of the episode. Immediate reward (as given in the standard pendulum problem) is typically much easier for reinforcement learning agents because the temporal credit assignment problem is minimized, whereas delayed reward (as used in this thesis) is much more difficult because of the extensive temporal credit assignment problem. The pendulum starts in a random position every time and because it is an unsolved environment, it does not have a specific reward threshold at which it is considered solved nor at which episode will terminate.

There are three observation inputs for the delayed-reinforcement pendulum environment. The first observation input is $\sin\theta$ and the second observation input is $\cos\theta$ and both inputs range from -1.0 to 1.0. The angle θ is used to give

Figure 4.6: Delayed-Reinforcement Pendulum Environment

the angle at which the pendulum is displaced. If it is displaced to the left it is a negative displacement and if it is displaced right then it is a positive displacement.The $\sin\theta$ and $\cos\theta$ are given as inputs rather than just θ because θ is necessarily discontinuous (e.g., if measured in degrees, there will be a discontinuity from 359° to 0°), whereas $\sin\theta$ and $\cos\theta$ are everywhere continuous. The third observation input represents the angular velocity of the pendulum and this input ranges from -8.0 to 8.0. The action represents the amount of left or right force on the pendulum and the value ranges between -2.0 and 2.0. The equation for reward is defined by

$$r = -\theta + 0.1 * \dot{\theta} + 0.001 * a$$

where r represents the reward, θ represents the angle of the pendulum and the units used to measure the angle is radians and the angle to represent upright vertical is 0, $\dot{\theta}$ represents the angular velocity of the pendulum and the units used to measure the angular velocity is radians per second, and a is the action taken in the delayed-reinforcement pendulum environment.

4.4 Experimental Condition

A *trial* refers to length of the simulation at the end of which a cumulative reward is calculated and ends in a terminal state. A *repetition* refers to number of times a trial is repeated. The number of trials considered in this experiment are 1000 and the number of time steps considered are 2000, 3000, 4000, respectively, for each algorithm with 10 repetitions each. Results are obtained for Stochastic Synapse Reinforcement Learning (SSRL), Deep Deterministic Policy Gradient (DDPG) and a random action controller. The results on a random action controller are used especially to provide a baseline on the results for both these algorithms. The performance of the algorithms is evaluated using the cumulative rewards that are collected for each repetition. The average and standard deviation of cumulative rewards on overall repetitions for different experimental conditions is collected. The average and standard deviation of cumulative rewards on overall repetitions across all the experimental conditions are collected.

Chapter 5

Results

This chapter presents the results obtained from Stochastic Synapse Reinforcement Learning (SSRL) and Deep Deterministic Policy Gradient (DDPG). The results are also obtained for a random action controller to provide a baseline for the results on both the algorithms. The actions are sampled randomly from a uniform distribution over the entire valid range of values. The number of trials considered in this experiment are 1000 and the number of time steps considered are 2000, 3000, 4000 for each algorithm. The environments used in this experiment are the mountain car continuous environment and the delayed-reinforcement pendulum environment.

The performance of the algorithms is evaluated using the cumulative rewards that are collected for each repetition. The average and standard deviation of cumulative rewards on overall repetitions for different experimental conditions is collected. The average and standard deviation of cumulative rewards on overall repetitions across all the experimental conditions is collected.

In addition to viewing the data graphically and determining mean and standard deviation in performance we include linear regression models to look at the overall trends, particularly when these trends are not otherwise clear. Finally, the average of average of cumulative rewards across 10 repetitions and the average of standard deviation of cumulative rewards across 10 repetitions is calculated for different time steps and the average of average of cumulative rewards across 30 repetitions and the average of standard deviation of cumulative rewards across 30 repetitions is calculated for different algorithms mentioned above across pendulum and mountain car environments and is tabulated.

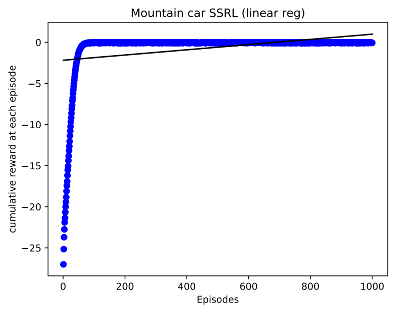
5.1 Mountain Car Continuous Environment

In this section the graphs for cumulative rewards for sample repetitions, the average of cumulative rewards across 10 repetitions and the standard deviation of cumulative rewards across 10 repetitions for various time steps 2000, 3000, 4000 and the average of cumulative rewards across 30 repetitions and the standard deviation of cumulative rewards across 30 repetitions for Stochastic Synapse Reinforcement Learning, Deep Deterministic Policy Gradient, and random action controller in the mountain car continuous environment are shown and the graphs are explained.

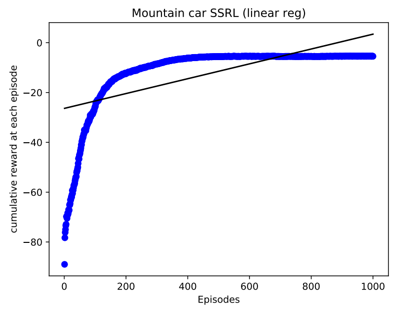
5.1.1 Stochastic Synapse Reinforcement Learning (SSRL)

2000 Time Steps

Figure 5.1 shows cumulative rewards for repetition 1 and 2 for 2000 time steps. The rest of the repetitions are shown in Figure A.1 from the appendix. As in Figure 5.1a it is seen that in repetition 1 the rewards start from around -28 and the rewards gradually rise upwards and around episode 100 the rewards reach the maximum of 0 and remain steady across the remaining episodes. As in Figure 5.1b it can be seen that in repetition 2 the rewards start from around -90 and the rewards gradually rise upwards and around episode 300 the rewards reach the maximum of -10 and remains steady across the remaining episodes. Thus, the algorithm tries to learn how to achieve better rewards and maintains the better rewards as the number of episodes progresses. The regression line in repetition 1 starts around -2.5 and goes up to 1 and the slope is 0.003. The regression line in repetition 2 starts around -25 and goes up to 1 and the slope is 0.029.



(a) Cumulative rewards for repetition 1.

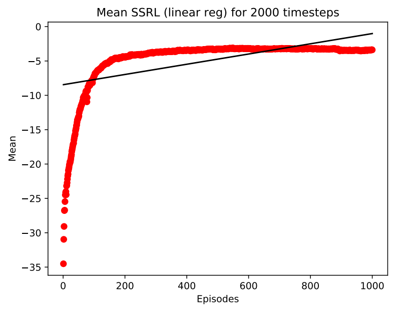


(b) Cumulative rewards for repetition 2.

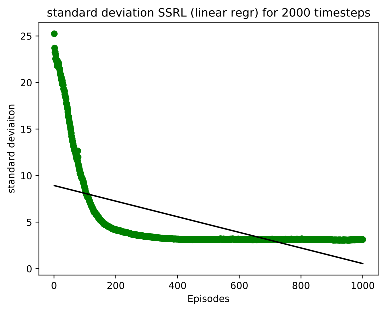
Figure 5.1: Cumulative rewards (blue circles) for repetitions 1 and 2 for 2000 time steps using SSRL in mountain car continuous environment. The black line indicates the linear regression line.

Figure 5.2 shows the average of cumulative rewards and the standard deviation of cumulative rewards across 10 repetitions for 2000 time steps. In the graph of average of cumulative rewards it starts from around -35 and the average rewards gradually rise upwards and around episode 100 the average rewards reach the maximum of -5 and remain steady across the remaining episodes and that in standard deviation of the cumulative rewards start from around 24 and the rewards gradually go downwards and around episode 200 the standard deviation of rewards goes down to 3 and remain steady across the remaining episodes. The regression line in average of cumulative rewards starts around -8 and goes up to

-2 and the slope is 0.007. The regression line in standard deviation of cumulative rewards starts around 8 and goes down to 1 and the slope is -0.008.



(a) Average of cumulative rewards.



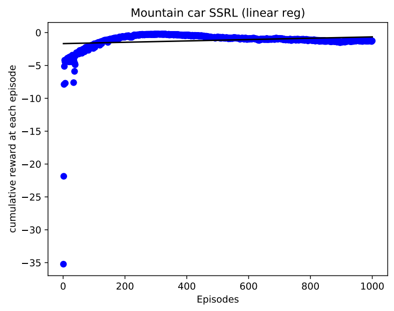
(b) Deviation of cumulative rewards.

Figure 5.2: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 10 repetitions for 2000 time steps using SSRL in mountain car continuous environment. The black line indicates the linear regression line.

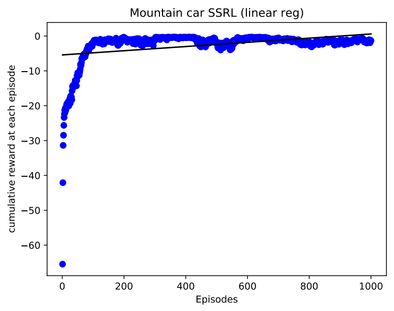
3000 Time Steps

Figure 5.3 shows cumulative rewards for repetition 1 and 2 for 3000 time steps. The rest of the repetitions are shown in Figure A.2 from the appendix. As in Figure 5.3a it is seen that in repetition 1 the rewards start from around -35 and the rewards gradually rise upwards and around repetition 100 the rewards reach the maximum of around 0 and remain steady across the remaining episodes even though there is little decline in rewards from around episode 400. As in Figure 5.3bit is seen that in repetition 2 the rewards start from around -65 and the rewards gradually rise upwards and around episode 100 the rewards reach the maximum of -2 and remain steady across the remaining episodes even though there is occasional slight decline of rewards from around episode 100. Thus, the algorithm tries to learn how to achieve better rewards and maintains the better rewards as the number of episodes progresses. The regression line in repetition

1 is around -2.5 and the slope is 0.001. The regression line in repetition 2 starts around -5 and goes up to -1 and the slope is 0.006.



(a) Cumulative rewards for repetition 1.



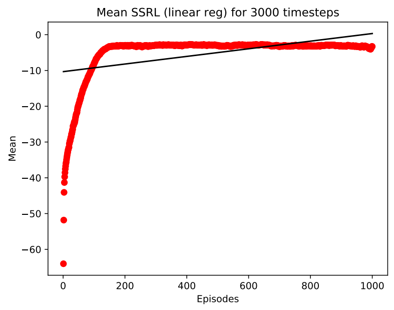
(b) Cumulative rewards for repetition 2.

Figure 5.3: Cumulative rewards (blue circles) for repetitions 1 and 2 for 3000 time steps using SSRL in mountain car continuous environment. The black line indicates the linear regression line.

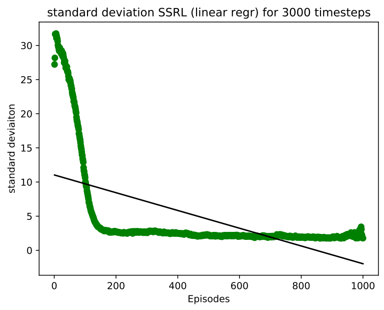
Figure 5.4 shows the average of cumulative rewards and the standard deviation of cumulative rewards across 10 repetitions for 3000 time steps. In the graph of average of cumulative rewards it starts from around -65 and the average rewards gradually rise upwards and around episode 150 the average rewards reach the maximum of -5 and remain steady across the remaining episodes and that in standard deviation of the cumulative rewards start from around 23 and the rewards gradually going downward and around episode 150 the standard deviation of rewards goes down to 2.5 and remains steady across the remaining episodes. The regression line in average of cumulative rewards it starts around -10 and goes up to -1 and the slope is 0.011. The regression line in standard deviation of cumulative rewards starts around 12 and goes down to -2 and the slope is -0.013.

4000 Time Steps

Figure 5.5 shows cumulative rewards for repetition 1 and 2 for 4000 time steps. The rest of the repetitions are shown in Figure A.3 from the appendix. As in the



(a) Average of cumulative rewards.

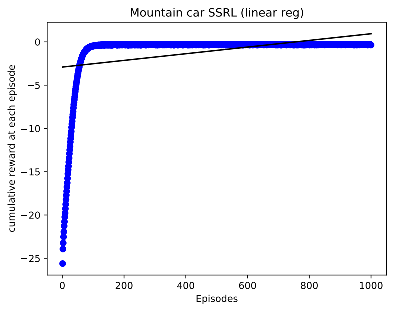


(b) Deviation of cumulative rewards.

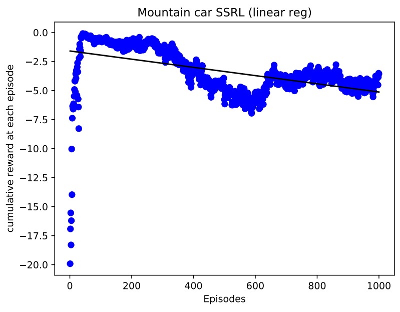
Figure 5.4: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 10 repetitions for 3000 time steps using SSRL in mountain car continuous environment. The black line indicates the linear regression line.

Figure 5.5a it is seen that in repetition 1 the rewards start from around -26 and the rewards gradually rise upwards and around episode 100 the rewards reach the maximum of -1 and remain steady across the remaining episodes. As in the Figure 5.5b it is seen that in repetition 2 the rewards start from around -20 and the rewards gradually rise upwards and around episode 50 the rewards reach the maximum of 0 and the reward falls gradually to around -6 around episode 550 and rises to about -3 around episode 650. Thus, the algorithm tries to learn how to achieve better rewards and maintains the better rewards as the number of episodes progresses and occasionally the rewards decline. The regression line in repetition 1 starts around -3 and goes up to 0 and the slope is 0.004. The regression line in repetition 2 starts around -2 and goes down to -5 and the slope is -0.003.

Figure 5.6 shows the average of cumulative rewards and the standard deviation of cumulative rewards across 10 repetition for 4000 time steps. In the graph of average of cumulative rewards it starts from around -17 and the average rewards gradually rise upwards and around episode 100 the average rewards reach



(a) Cumulative rewards for repetition 1.



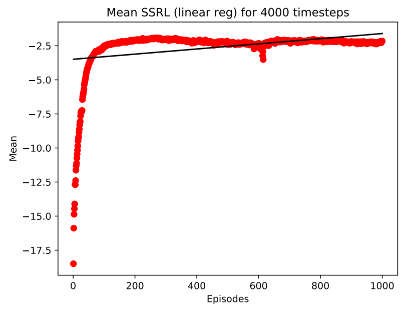
(b) Cumulative rewards for repetition 2.

Figure 5.5: Cumulative rewards (blue circles) for repetitions 1 and 2 for 4000 time steps using SSRL in mountain car continuous environment. The black line indicates the linear regression line.

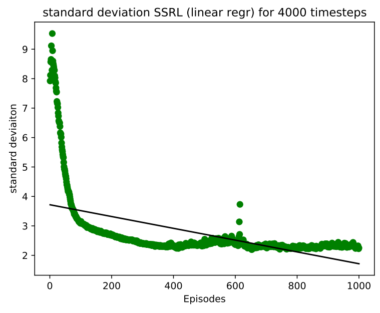
the maximum of -2.5 and remain steady across the remaining episodes and there is a slight fall in average reward around 650 episode and that in standard deviation of the cumulative rewards start from around 9.5 and the rewards gradually going downward and around episode 200 the standard deviation of rewards goes down to 2.5 and remain steady across the remaining episodes but there is rise it standard deviation around episode 600. The regression line in average of cumulative rewards it starts around -3 and goes up to -2 and the slope is 0.002. The regression line in standard deviation of cumulative rewards starts around 3.7 and goes down to 1.5 and the slope is -0.002.

Overall Results of Mean and Standard Deviation of Cumulative Rewards

Figure 5.7. shows the average of cumulative rewards and the standard deviation of cumulative rewards across 30 repetitions. In the graph of average of cumulative rewards it starts from around -41 and the average rewards gradually rise upwards and around episode 100 the average rewards reach the maximum of -4 and remain steady across the remaining episodes and that in standard deviation



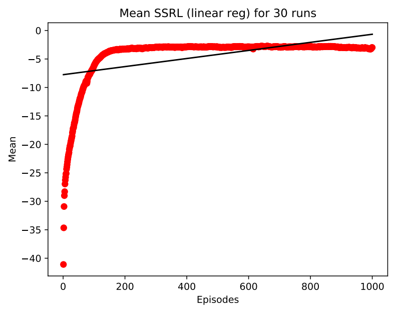
(a) Average of cumulative rewards.



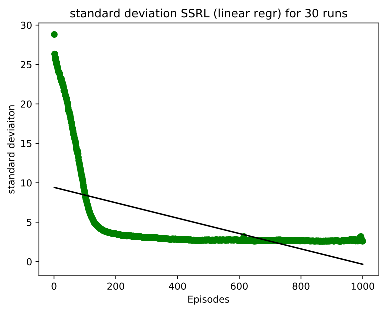
(b) Deviation of cumulative rewards.

Figure 5.6: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 10 repetitions for 4000 time steps using SSRL in mountain car continuous environment. The black line indicates the linear regression line.

of the cumulative rewards start from around 29 and the rewards gradually going downward and around episode 150 the standard deviation of rewards goes down to 3 and remains steady across the remaining episodes. The regression line in average of cumulative rewards it starts around -8 and goes up to -2 and the slope is 0.007. The regression line in standard deviation of cumulative rewards starts around 10 and goes down to -1 and the slope is -0.009.



(a) Average of cumulative rewards.



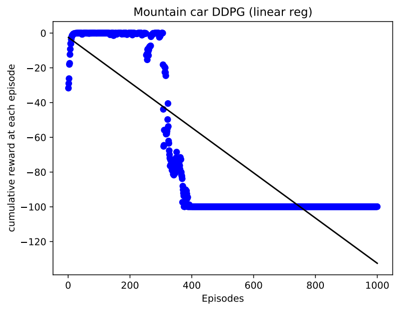
(b) Deviation of cumulative rewards.

Figure 5.7: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 30 repetitions using SSRL in mountain car continuous environment. The black line indicates the linear regression line.

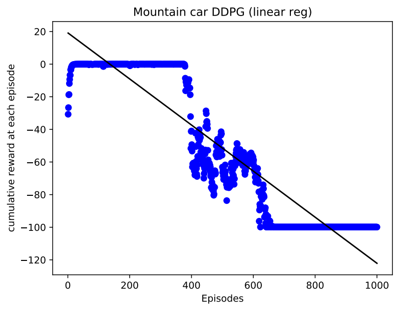
5.1.2 Deep Deterministic Policy Gradient (DDPG)

2000 Time Steps

Figure 5.8. shows cumulative rewards for repetition 1 and 2 for 2000 time steps. The rest of the repetitions are shown in Figure A.4 from the appendix. In the Figure 5.8a for repetition 1 the cumulative rewards start from -35 and the and rise up to 0 around episode 20 and maintains the reward till around episode 350 and then reward declines drastically to about -100 around episode 400 and maintains the reward till the end. In the Figure 5.8b for repetition 2 the cumulative rewards start from -30 and rise up to 0 around episode 20 and maintains the reward till around episode 350 and then reward decline drastically to about -100 around episode 600 and maintains the reward till the end. The regression line in repetition 1 starts around -1 and goes down to -130 and the slope is -0.130. The regression line in repetition 2 starts around 20 and goes down to -120 and the slope is -0.141.



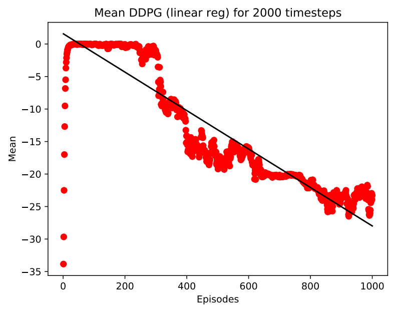
(a) Cumulative rewards for repetition 1.



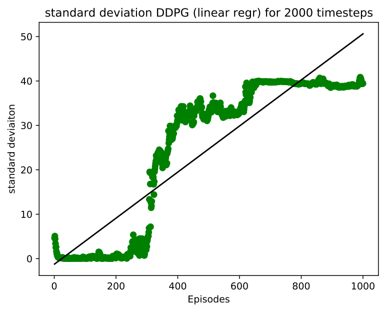
(b) Cumulative rewards for repetition 2.

Figure 5.8: Cumulative rewards (blue circles) for repetitions 1 and 2 for 2000 time steps using DDPG in mountain car continuous environment. The black line indicates the linear regression line.

Figure 5.9 shows the average of cumulative rewards and the standard deviation of cumulative rewards across 10 repetition for 2000 time steps. In the graph of average of cumulative rewards, it start from -35 and the and rise up to 0 around episode 20 and maintains the average reward till around episode 250 and then reward declines drastically to about -25 at the end. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 5 and then it goes down to 0 and maintains the deviation until episode 250 and then the deviation rise to about 40 around episode 650 and then maintains deviation throughout the end. The regression line in average cumulative rewards starts around 2 and goes down to -27 and the slope is -0.029. The regression line in standard deviation of cumulative rewards starts around -1 and goes up to 50 and the slope is 0.052.



(a) Average of cumulative rewards.

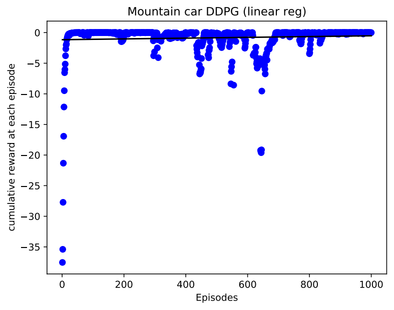


(b) Deviation of cumulative rewards.

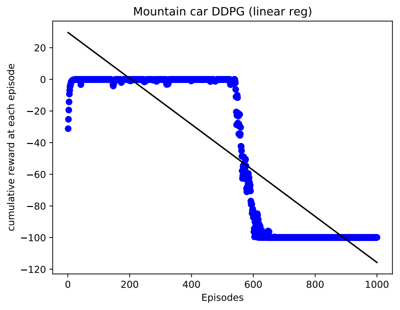
Figure 5.9: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 10 repetitions for 2000 time steps using DDPG in mountain car continuous environment. The black line indicates the linear regression line.

3000 Time Steps

Figure 5.10. shows cumulative rewards for repetition 1 and 2 for 3000 time steps. The rest of the repetitions are shown in Figure A.5 from the appendix . In the Figure 5.10a for repetition 1 the cumulative rewards start from -38 and then rise up to 0 around episode 50 and maintains the reward throughout the episodes but the reward decline occasionally and at most to about -20 around episode 650. In the Figure 5.10b for repetition 2 the cumulative rewards start from -35 and the and rise up to 0 around episode 20 and maintains the reward till around episode 550 and then reward decline drastically to about -100 around episode 600 and maintains the reward till the end. The regression line in repetition 1 is around -1 and the slope is 0.001. The regression line in repetition 2 starts around 20 and goes down to -120 and the slope is -0.145.



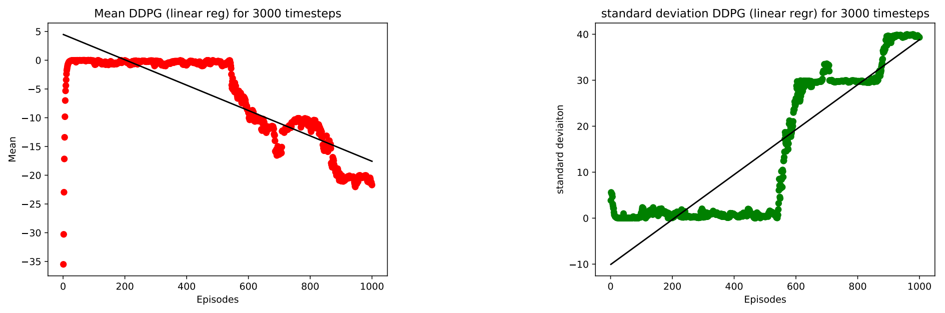
(a) Cumulative rewards for repetition 1.



(b) Cumulative rewards for repetition 2.

Figure 5.10: Cumulative rewards (blue circles) for repetitions 1 and 2 for 3000 time steps using DDPG in mountain car continuous environment. The black line indicates the linear regression line.

Figure 5.11 shows the average of cumulative rewards and the standard deviation of cumulative rewards across 10 repetitions for 3000 time steps. In the graph of average of cumulative rewards, it start from -35 and the and rise up to 0 around episode 20 and maintains the average reward till around episode 550 and then reward decline drastically to about -23 at the end. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 5 and then it goes down to 0 and maintains the deviation until episode 550 and then the deviation rise to about 30 around episode 650 and then maintains deviation until episode 850 and again rise to about 40 around episode 900 and maintains throughout the end. The regression line in average cumulative rewards starts around 4 and goes down to -18 and the slope is -0.022. The regression line in standard deviation of cumulative rewards starts around -1 and goes up to 40 and the slope is 0.048.

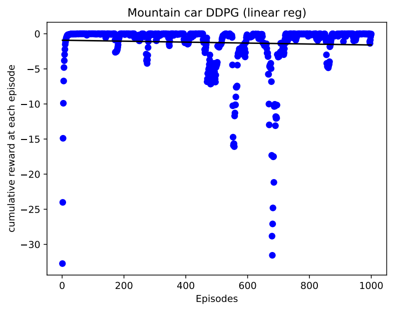


(a) Average of cumulative rewards. (b) Deviation of cumulative rewards.

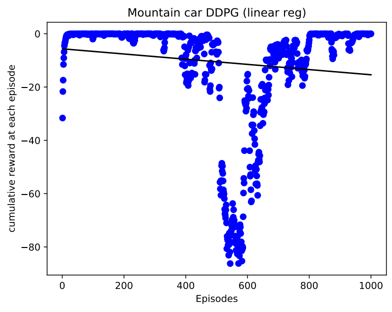
Figure 5.11: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 10 repetitions for 3000 time steps using DDPG in mountain car continuous environment. The black line indicates the linear regression line.

4000 Time Steps

Figure 5.12 cumulative rewards for repetition 1 and 2 for 4000 time steps. The rest of the repetitions are shown in figure A.6 from the appendix. In Figure 5.12a for repetition 1 the cumulative rewards start from -33 and then rise up to 0 around episode 50 and maintains the reward throughout the episodes but the reward decline occasionally and at most to about -32 around episode 650. In Figure 5.12b for repetition 2 the cumulative rewards start from -35 and the and rise up to 0 around 20 episodes and maintains the reward till around episode 400 and then reward decline drastically to about -90 around episode 550 and again rises to about 0. The regression line in repetition 1 is around -2 and the slope is -0.001. The regression line in repetition 2 starts around -5 and goes down to -15 and the slope is -0.009.



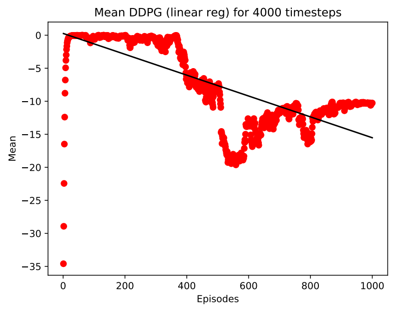
(a) Cumulative rewards for repetition 1.



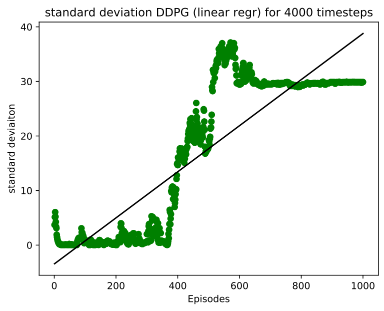
(b) Cumulative rewards for repetition 2.

Figure 5.12: Cumulative rewards (blue circles) for repetitions 1 and 2 for 4000 time steps using DDPG in mountain car continuous environment. The black line indicates the linear regression line.

Figure 5.13. shows the average of cumulative rewards and the standard deviation of cumulative rewards across 10 repetitions for 4000 time steps. In the graph of average of cumulative rewards, it start from -35 and the and rise up to 0 around episode 20 and maintains the average reward till around episode 350 and then reward decline drastically to about -18 and slowly rise to -12 at the end. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 5 and then it goes down to 0 and maintains the deviation until episode 350 and then the deviation rises to about 37 around episode 600 and decline a little to around 30 and maintains it throughout. The regression line in average cumulative rewards starts around 4 and goes down to -15 and the slope is -0.015. The regression line in standard deviation of cumulative rewards starts around -3 and goes up to 38 and the slope is 0.042.



(a) Average of cumulative rewards.

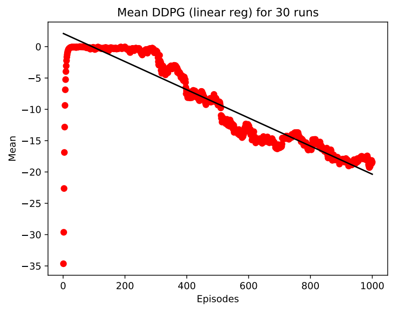


(b) Deviation of cumulative rewards.

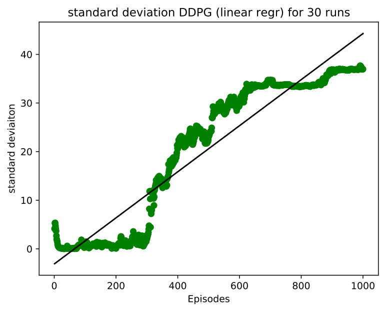
Figure 5.13: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 10 repetitions for 4000 time steps using DDPG in mountain car continuous environment. The black line indicates the linear regression line.

Overall Results of Mean and Standard Deviation of Cumulative Rewards

Figure 5.14 shows the average of cumulative rewards and the standard deviation of cumulative rewards across 30 repetitions. In the graph of average of cumulative rewards, it start from -35 and the and rise up to 0 around episode 20 and maintains the average reward till around episode 300 and then reward falls drastically to about -19 at the end. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 5 and then it goes down to 0 and maintains the deviation until episode 300 and then the deviation rise to about 32 around episode 650 and then maintains deviation until episode 850 and again rise to about 35 around episodes 870 and maintains throughout the end. The regression line in average cumulative rewards starts around 2 and goes down to -20 and the slope is -0.022. The regression line in standard deviation of cumulative rewards starts around -2 and goes up to 43 and the slope is 0.047.



(a) Average of cumulative rewards.



(b) Deviation of cumulative rewards.

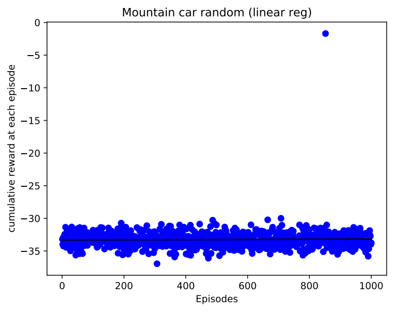
Figure 5.14: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 30 repetitions using DDPG in mountain car continuous environment. The black line indicates the linear regression line.

5.1.3 Random Action Controller

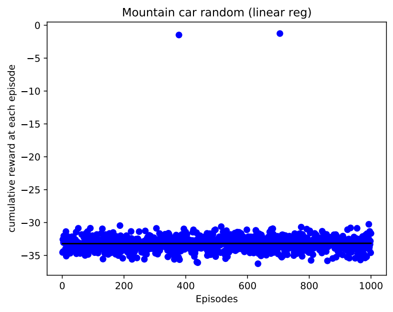
2000 Time Steps

Figure 5.15 shows cumulative rewards for repetitions 1 and 2 for 2000 time steps. The rest of the repetitions are shown in Figure A.7 from the appendix. As in Figure 5.15a and Figure 5.15b it is seen that for repetition 1 and repetition 2 throughout the episodes the cumulative rewards is between -30 and -36 and occasionally the rewards went up to about -1 around episode 820 for repetition 1 and about -1 around episodes 350, 700 for repetition 2. Thus, the controller does not learn much as it consistently gives poor rewards. The regression line in repetition 1 is around -33 and slope is 0.000. The regression line in repetition 2 is around -33 and slope is 0.000.

Figure 5.16 shows the average of cumulative rewards and the standard deviation of cumulative rewards across 10 repetitions for 2000 time steps. In the graph of average of cumulative rewards initially the average rewards starts from around -33.5 and most of the average ranges between -33 and -34 across most of



(a) Cumulative rewards for repetition 1.



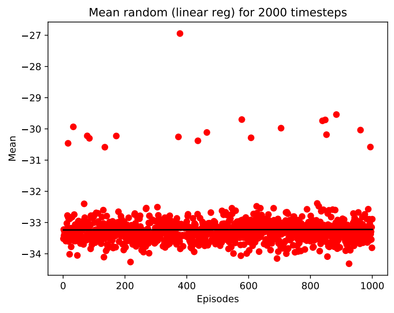
(b) Cumulative rewards for repetition 2.

Figure 5.15: Cumulative rewards (blue circles) for repetitions 1 and 2 for 2000 time steps using random action controller in mountain car continuous environment. The black line indicates the linear regression line.

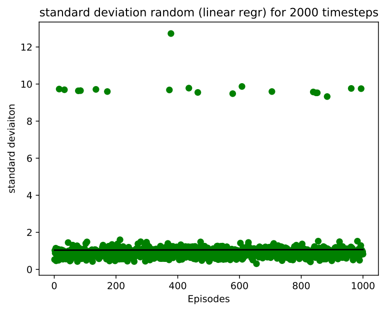
the episodes and some of the average cumulative reward goes up to about -30 as it can be seen around episodes 50, 550, 900 and some of the average rewards goes up to -27 as it can be seen around episode 350. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 1 and most of the standard deviation ranges between 0 and 1 across most of the episodes and some of the standard deviation of cumulative reward goes up to around 10 as it can be seen around episodes 600, 950, 980 and some of the standard deviation of rewards goes up to 13 as it can be seen around episode 350. The regression line in average of cumulative rewards is around -33.5 and slope is 0.000. The regression line in standard deviation of cumulative rewards is around 1 and slope is 0.000.

3000 Time Steps

Figure 5.17 shows cumulative rewards for repetitions 1 and 2 for 3000 time steps. The rest of the repetitions are shown in Figure A.8 from the appendix. As in Figure 5.17a it is seen that for repetition 1 throughout the episodes the cumulative rewards went down to around -37 and occasionally the rewards went up to -30.5



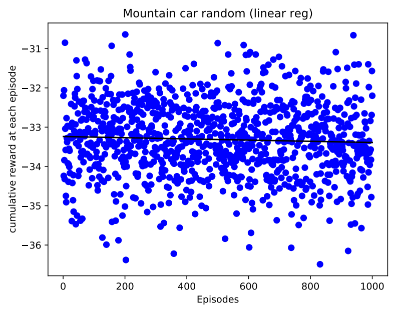
(a) Average of cumulative rewards.



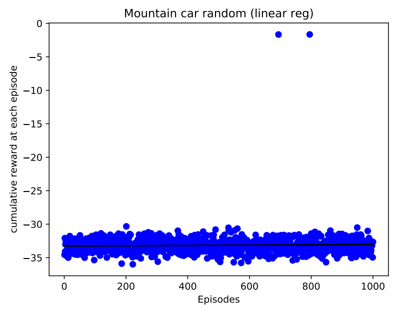
(b) Deviation of cumulative rewards.

Figure 5.16: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 10 repetitions for 2000 time steps using random action controller in mountain car continuous environment. The black line indicates the linear regression line.

around episodes 150 and 900.As in Figure 5.17b it is seen that for repetition 2 throughout the episodes the cumulative rewards is between -32 and -36 and occasionally the rewards went up to about -2 around episodes 650 and 750. Thus, the controller does not learn much as it consistently gives poor rewards. The regression line in repetition 1 is around starts around -33.25 and goes down to -33.5 and the slope -0.0001. The regression line in repetition 2 is around -33 and slope is 0.000.



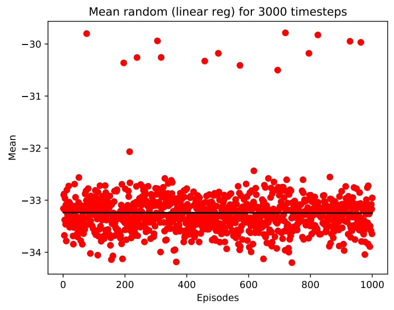
(a) Cumulative rewards for repetition 1.



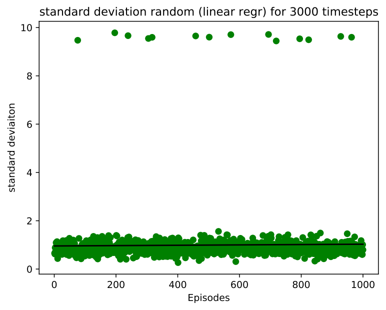
(b) Cumulative rewards for repetition 2.

Figure 5.17: Cumulative rewards (blue circles) for repetitions 1 and 2 for 3000 time steps using random action controller in mountain car continuous environment. The black line indicates the linear regression line.

Figure 5.18 shows average of cumulative rewards and the standard deviation of cumulative rewards across 10 repetitions for 3000 time steps. In the graph of average of cumulative rewards initially the average rewards starts from around -33.5 and most of the average ranges between -33 and -34 across most of the episodes and some of the average cumulative reward goes up to about -32 as it can be seen around episode 250 and some of the average rewards goes up to -29.5 as it can be seen around episodes 75 and 700. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 0.5 and most of the standard deviation ranges between 0 and 1 across most of the episodes and some of the standard deviation of cumulative reward goes up to around 9.5 as it can be seen around episodes 150, 700. The regression line in average of cumulative rewards is around -33.25 and slope is 0.000. The regression line in standard deviation of cumulative rewards is around 1 and slope is 0.000.



(a) Average of cumulative rewards.

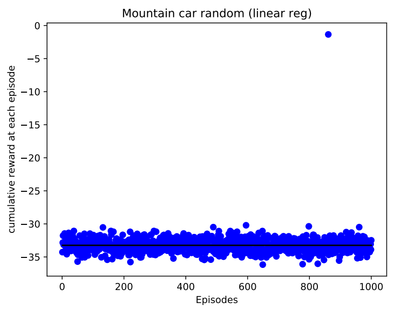


(b) Deviation of cumulative rewards.

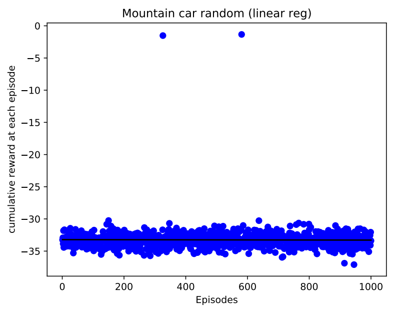
Figure 5.18: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 10 repetitions for 3000 time steps using random action controller in mountain car continuous environment. The black line indicates the linear regression line.

4000 Time Steps

Figure 5.19 shows cumulative rewards for repetitions 1 and 2 for 4000 time steps. The rest of the repetitions are shown in Figure A.9 from the appendix. As in the Figure 5.19a and Figure 5.19b it is seen that for repetitions 1 and repetitions 2 throughout the episodes the cumulative rewards is between -30 and -36 and occasionally the rewards went up to about -1 around episode 850 for repetition 1 and about -1 around episodes 300, 550 for repetition 2. Thus, the controller does not learn much as it consistently gives poor rewards. The regression line in repetition 1 is around -33 and slope is 0.000. The regression line in repetition 2 is around -33 and slope is 0.000.



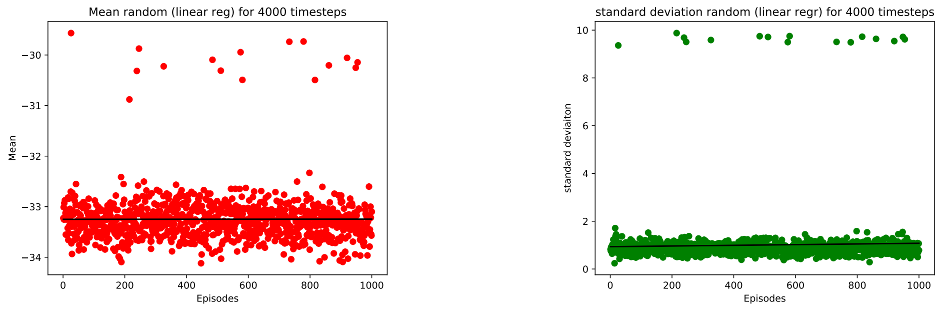
(a) Cumulative rewards for repetition 1.



(b) Cumulative rewards for repetition 2.

Figure 5.19: Cumulative rewards (blue circles) for repetitions 1 and 2 for 4000 time steps using random action controller in mountain car continuous environment. The black line indicates the linear regression line.

Figure 5.20 shows average of cumulative rewards and the standard deviation of cumulative rewards across 10 repetitions for 4000 time steps. In the graph of average of cumulative rewards initially the average rewards starts from around -33.5 and most of the average ranges between -33 and -34 across most of the episodes and some of the average cumulative reward goes up to about -29.5 as it can be seen around episode 50. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 1 and most of the standard deviation ranges between 0 and 1 across most of the episodes and some of the standard deviation of cumulative reward goes up to around 10 as it can be seen around episode 250. The regression line in average of cumulative rewards is around -33.5 and slope is 0.000. The regression line in standard deviation of cumulative rewards is around 1 and slope is 0.000.

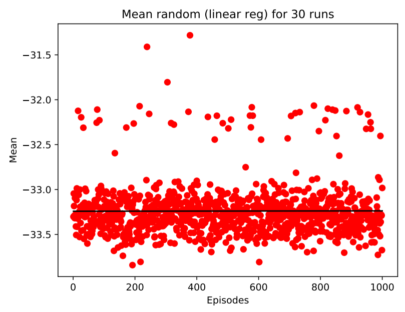


(a) Average of cumulative rewards. (b) Deviation of cumulative rewards.

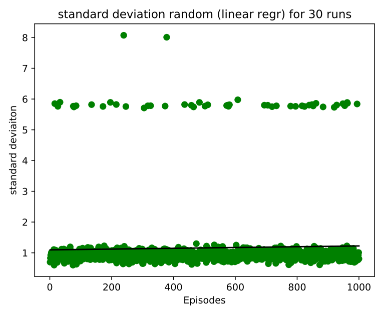
Figure 5.20: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 10 repetitions for 4000 time steps using random action controller in mountain car continuous environment. The black line indicates the linear regression line.

Overall Results of Mean and Standard Deviation of Cumulative Rewards

Figure 5.21 shows the average of cumulative rewards and the standard deviation of cumulative rewards across 30 repetitions. In the graph of average of cumulative rewards initially the average rewards starts from around -33.5 and most of the average ranges between -33 and -33.5 across most of the episodes and some of the average cumulative reward goes up to about -31.75 as it can be seen around episode 300 and some of the average rewards goes up to -31 as it can be seen around episode 350. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 0.5 and most of the standard deviation ranges between 0 and 1 across most of the episodes and some of the standard deviation of cumulative reward goes up to around 6 as it can be seen around episodes 50, 580 and some of the standard deviation of rewards goes up to 8 as it can be seen around episodes 250, 380. The regression line in average of cumulative rewards is around -33.25 and slope is 0.000. The regression line in standard deviation of cumulative rewards is around 1 and slope is 0.000.



(a) Average of cumulative rewards.



(b) Deviation of cumulative rewards.

Figure 5.21: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 30 repetitions using random action controller in mountain car continuous environment. The black line indicates the linear regression line.

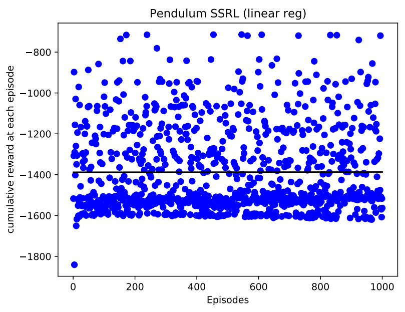
5.2 Delayed-Reinforcement Pendulum Environment

In this section the graphs for cumulative rewards for sample repetitions, the average of cumulative rewards across 10 repetitions and the standard deviation of cumulative rewards across 10 repetitions for various time steps 2000, 3000, 4000 and the average of cumulative rewards across 30 repetitions and the standard deviation of cumulative rewards across 30 repetitions for Stochastic Synapse Reinforcement Learning, Deep Deterministic Policy Gradient, and random action controller in the delayed-reinforcement pendulum environment are shown and the graphs are explained.

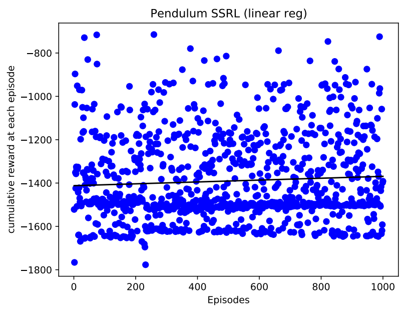
5.2.1 Stochastic Synapse Reinforcement Learning (SSRL)

2000 Time Steps

Figure 5.22 shows cumulative rewards for repetitions 1 and 2 for 2000 time steps. The rest of the repetitions are shown in Figure A.10 from the appendix. As in Figure 5.20a and Figure 5.20b it is seen that in the initial few episodes the cumulative rewards went down to -1800 and after few episodes the rewards maintain around -1600 throughout the repetition 1 and around -1700 throughout the repetition 2 and occasionally the rewards go up to -700 as it can be seen around episodes 200, 400, 600. Thus, the algorithm tries to avoid learning bad rewards as the number of episodes progresses. The regression line in repetition 1 is around -1400 and the slope is 0.001. The regression line in repetition 2 is also around -1400 and the slope is 0.042.



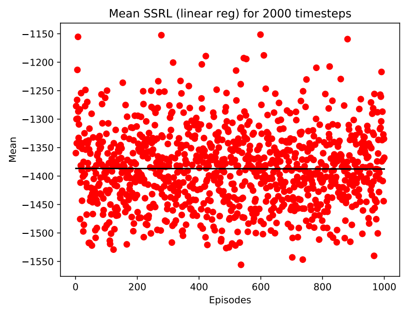
(a) Cumulative rewards for repetition 1.



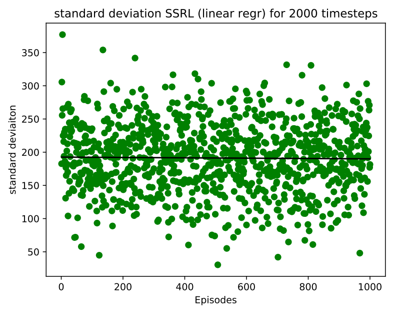
(b) Cumulative rewards for repetition 2.

Figure 5.22: Cumulative rewards (blue circles) for repetitions 1 and 2 for 2000 time steps using SSRL in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

Figure 5.23 shows average of cumulative rewards and the standard deviation of cumulative rewards across 10 repetitions for 2000 time steps. In the graph of average of cumulative rewards initially the average rewards starts from around -1350 and most of the average ranges between -1300 and -1400 across most of the episodes and some of the average cumulative reward goes down to -1550 as it can be seen around episodes 500, 700, 900 and some of the average rewards goes up to -1150 as it can be seen around episodes 300, 600, 850. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 200 and most of the standard deviation ranges between 150 and 250 across most of the episodes and some of the standard deviation of cumulative reward goes down to 50 as it can be seen around episodes 100, 500, 700 and some of the standard deviation of rewards goes up to 350 as it can be seen around episodes 50 and 200. The linear regression line of average of cumulative rewards is around -1370 and the slope is -0.001 and the linear regression line of standard deviation of cumulative rewards is around 180 and the slope is -0.002.



(a) Average of cumulative rewards.

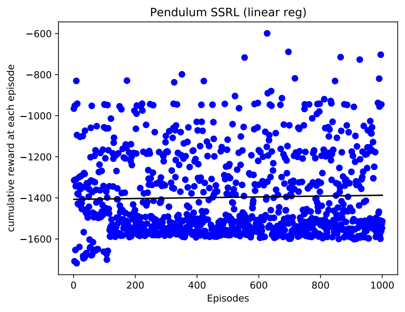


(b) Deviation of cumulative rewards.

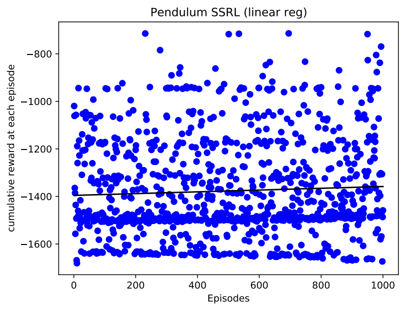
Figure 5.23: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 10 repetitions for 2000 time steps using SSRL in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

3000 Time Steps

Figure 5.24 shows cumulative rewards for repetitions 1 and 2 for 3000 time steps. The rest of the repetitions are shown in Figure A.11 from the appendix. As in Figure 5.24a and Figure 5.24b it is seen that in the initial few episodes the cumulative rewards went down to -1700 and after few episodes the rewards maintain around -1600 throughout the repetition 1 and around -1700 throughout the repetition 2 and occasionally the reward go up to -600 for repetition 1 as it can be seen around episode 600 and -700 for repetition 2 as it can be seen around episodes 200, 450, 500, 700, 900. Thus, the algorithm tries to avoid learning bad rewards as the number of episodes progresses. The regression line in repetition 1 is around -1400 and slope is 0.019. The regression line in repetition 2 is also around -1400 and the slope is 0.037.



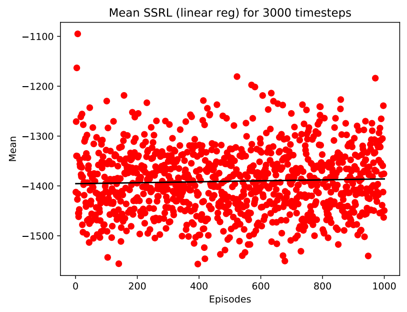
(a) Cumulative rewards for repetition 1.



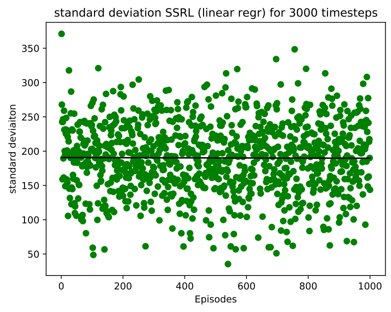
(b) Cumulative rewards for repetition 2.

Figure 5.24: Cumulative rewards (blue circles) for repetitions 1 and 2 for 3000 time steps using SSRL in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

Figure 5.25 shows the average of cumulative rewards and the standard deviation of cumulative rewards across 10 repetitions for 3000 time steps. In the graph of average of cumulative rewards initially the average rewards starts from around -1350 and most of the average ranges between -1300 and -1400 across most of the episodes and some of the average cumulative reward goes down to -1550 as it can be seen around episodes 100, 150, 400, 700 and some of the average rewards goes up to -1150 as it can be seen around episodes 300, 600, 850. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 160 and most of the standard deviation ranges between 150 and 250 across most of the episodes and some of the standard deviation of cumulative reward goes down to 40 as it can be seen around episode 550 and some of the standard deviation of rewards goes up to 370 as it can be seen around episode 30. The linear regression line of average of cumulative rewards is around -1390 and the slope is 0.009 and the linear regression line of standard deviation of cumulative rewards is around 190 and the slope is -0.001.



(a) Average of cumulative rewards.

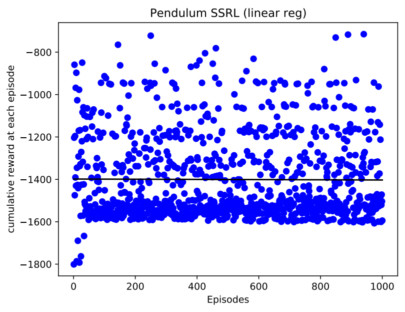


(b) Deviation of cumulative rewards.

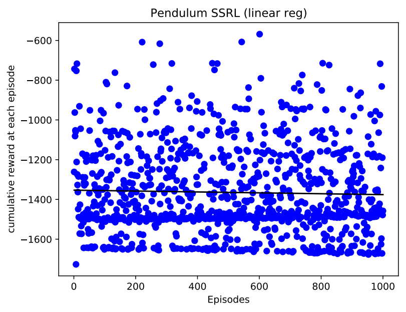
Figure 5.25: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 10 repetitions for 3000 time steps using SSRL in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

4000 Time Steps

Figure 5.26 shows cumulative rewards for repetitions 1 and 2 for 4000 time steps. The rest of the repetitions are shown in Figure A.12 from the appendix. As in Figure 5.26a and Figure 5.26b it is seen that in the initial few episodes the cumulative rewards went down to -1800 and after few episodes the rewards maintain around -1600 throughout the repetition 1 and around -1700 throughout the repetition 2 and occasionally the reward go up to -700 for repetition 1 as it can be seen around episodes 250, 800, 850, 900 and -590 for repetition 2 as it can be seen around episode 650. Thus, the algorithm tries to avoid learning bad rewards as the number of episodes progresses. The regression line in repetition 1 is around -1400 and slope is -0.004. The regression line in repetition 2 is around -1350 and the slope is -0.022.



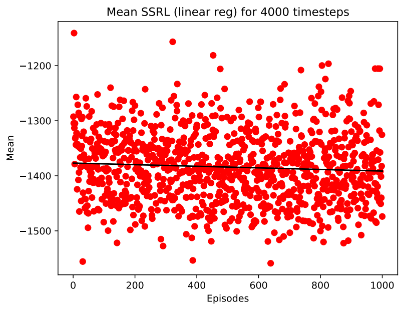
(a) Cumulative rewards for repetition 1.



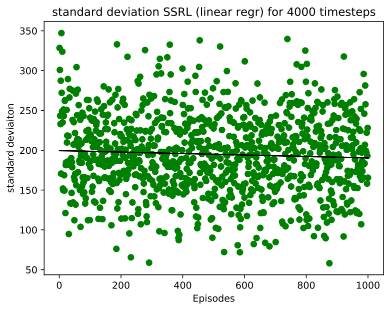
(b) Cumulative rewards for repetition 2.

Figure 5.26: Cumulative rewards (blue circles) for repetitions 1 and 2 for 4000 time steps using SSRL in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

Figure 5.27 average of cumulative rewards and the standard deviation of cumulative rewards across 10 repetitions for 4000 time steps. In the graph of average of cumulative rewards initially the average rewards starts from around -1380 and most of the average ranges between -1300 and -1450 across most of the episodes and some of the average cumulative reward goes down to -1550 as it can be seen around episodes 50, 400, 650 and some of the average rewards goes up to -1150 as it can be seen around episodes 25, 350. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 160 and most of the standard deviation ranges between 150 and 250 across most of the episodes and some of the standard deviation of cumulative reward goes down to 50 as it can be seen around episodes 250, 850 and some of the standard deviation of rewards goes up to 350 as it can be seen around episode 25. The linear regression line of average of cumulative rewards is around -1370 and the slope is -0.014 and the linear regression line of standard deviation of cumulative rewards is around 200 and the slope is -0.009.



(a) Average of cumulative rewards.

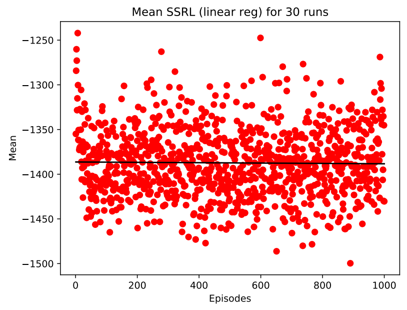


(b) Deviation of cumulative rewards.

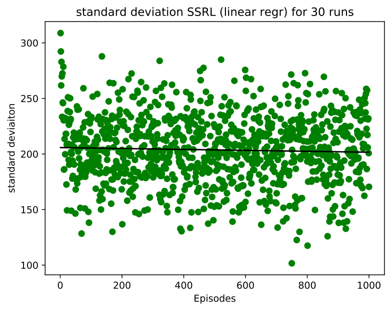
Figure 5.27: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 10 repetitions for 4000 time steps using SSRL in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

Overall Results of Mean and Standard Deviation of Cumulative Rewards

Figure 5.28 shows average of cumulative rewards and the standard deviation of cumulative rewards across 30 repetitions. In the graph of average of cumulative rewards initially the average rewards starts from around -1350 and most of the average ranges between -1350 and -1425 across most of the episodes and some of the average cumulative reward goes down to -1500 as it can be seen around episode 900 and some of the average rewards goes up to -1245 as it can be seen around episode 10. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 310 and most of the standard deviation ranges between 175 and 225 across most of the episodes and some of the standard deviation of cumulative reward goes down to 100 as it can be seen around episode 780 and some of the standard deviation of rewards goes up to 310 as it can be seen around initial episode. The linear regression line of average of cumulative rewards is around -1380 and the slope is -0.002 and the linear regression line of standard deviation of cumulative rewards is around 210 and the slope is -0.004.



(a) Average of cumulative rewards.



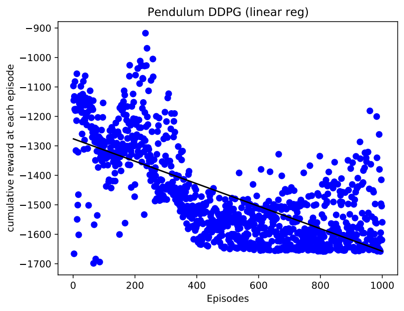
(b) Deviation of cumulative rewards.

Figure 5.28: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 30 repetitions using SSRL in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

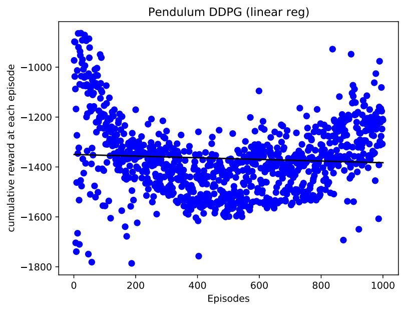
5.2.2 Deep Deterministic Policy Gradient (DDPG)

2000 Time Steps

Figure 5.29 shows cumulative rewards for repetitions 1 and 2 for 2000 time steps. The rest of the repetitions are shown in Figure A.13 from the appendix. As in Figure 5.29a it is seen that in the initial few episodes the cumulative rewards in repetition 1 went down to -1680 and after episode 100 the rewards goes up to -900 and after episode 200 the rewards goes down to around -1600 and the rewards flatten out after episode 400 with slight rise in rewards at the end. As in Figure 5.29b it is seen that in repetition 2 the rewards start from around -1800 in the initial episodes and around episode 30 the rewards raise up to around -850 and after episode 100 the rewards fall and raises again around episode 800. Thus, the algorithm tries to learn better rewards but sometimes the rewards decline ending up learning badly and then learns better later. The regression line in repetition 1 starts around -1290 and goes down to -1650 and the slope is -0.381. The regression line in repetition 2 starts around -1350 and goes down to -1400 and the slope is -0.033.



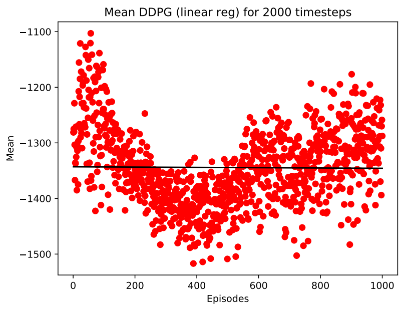
(a) Cumulative rewards for repetition 1.



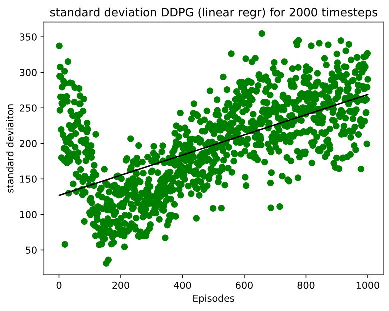
(b) Cumulative rewards for repetition 2.

Figure 5.29: Cumulative rewards (blue circles) for repetitions 1 and 2 for 2000 time steps using DDPG in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

Figure 5.30 shows the average of cumulative rewards and the standard deviation of cumulative rewards across 10 repetitions for 2000 time steps. In the graph of average of cumulative rewards initially the average rewards starts from around -1390 and the mean rises to around -1100 in episode 100 and then the mean decline to -1500 around episode 400 and later the mean rise up to -1200 around episode 850. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 170 and in the initial episodes the standard deviation rise up to 340 and decline quickly to around 40 in episode 170 and then it rise again up to 350 around episode 600. The regression line in average cumulative rewards starts around -1350 and maintain that value throughout the line and the slope is -0.002. The regression line in standard deviation of cumulative rewards starts around 125 and goes up to 250 and the slope is 0.142.



(a) Average of cumulative rewards.



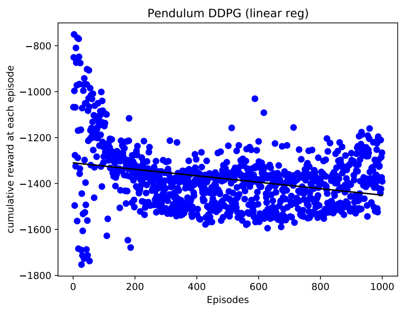
(b) Deviation of cumulative rewards.

Figure 5.30: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 10 repetitions for 2000 time steps using DDPG in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

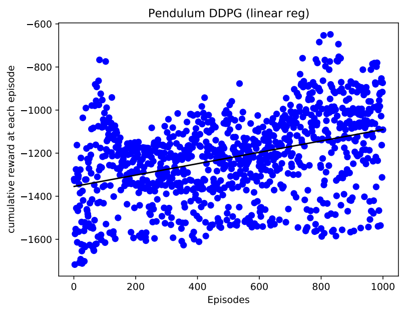
3000 Time Steps

Figure 5.31 shows cumulative rewards for repetitions 1 and 2 for 3000 time steps. The rest of the repetitions are shown in Figure A.14 from the appendix. As in Figure 5.31a it is seen that in the initial few episodes the cumulative rewards in repetition 1 went down to -1750 and after episode 50 the rewards goes up to -750 and immediately rewards goes down to around -1700 and the rewards flatten out after episode 200 with slight rise in rewards around episode 600. As in Figure 5.31b it is seen that in repetition 2 the rewards start from around -1700 in the initial episodes and around episode 100 the rewards rise up to around -750 and after episode 100 the rewards fall and rise again to -650 around episode 800.

Thus, the algorithm tries to learn better rewards but sometimes the rewards fall ending up learning badly and then learns better later. The regression line in repetition 1 starts around -1300 and goes down to -1400 and the slope is -0.141. The regression line in repetition 2 starts around -1350 and goes up to -1100 and the slope is 0.264.



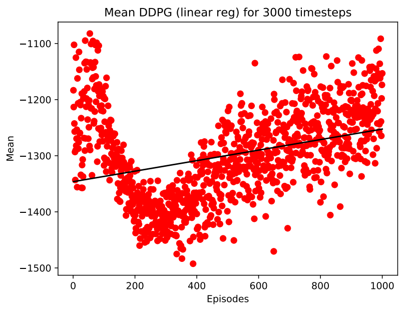
(a) Cumulative rewards for repetition 1.



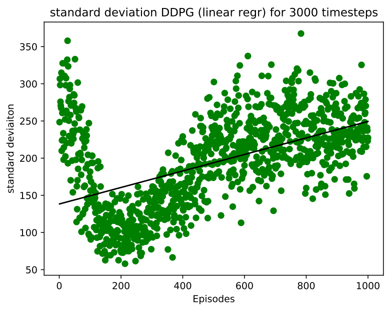
(b) Cumulative rewards for repetition 2.

Figure 5.31: Cumulative rewards (blue circles) for repetitions 1 and 2 for 3000 time steps using DDPG in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

Figure 5.32 average of cumulative rewards and the standard deviation of cumulative rewards across 10 repetitions for 3000 time steps. In the graph of average of cumulative rewards initially the average rewards starts from around -1350 and the mean rise to around -1100 in episode 100 and then the mean decline to -1470 around episode 200 and later the mean rise up to -1100 around episode 950. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 250 and in the initial episodes the standard deviation rise up to 360 and decline quickly to around 50 in episode 200 and then it rise again up to 370 around episode 750. The regression line in average cumulative rewards starts around -1350 and goes up to -1250 and the slope is 0.094. The regression line in standard deviation of cumulative rewards starts around 140 and goes up to 240 and the slope is 0.111.



(a) Average of cumulative rewards.

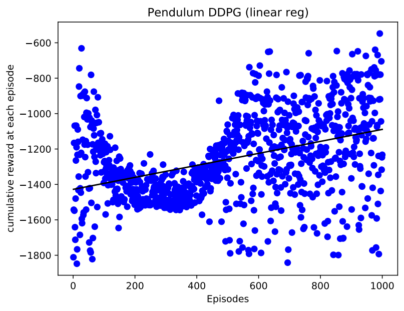


(b) Deviation of cumulative rewards.

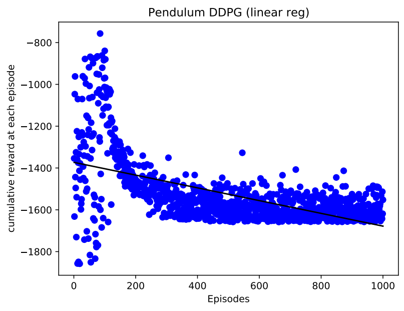
Figure 5.32: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 10 repetitions for 3000 time steps using DDPG in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

4000 Time Steps

Figure 5.33 shows cumulative rewards for repetitions 1 and 2 for 4000 time steps. The rest of the repetitions are shown in Figure A.15 from the appendix. As in Figure 5.33a it is seen that in the initial few episodes the cumulative rewards in repetition 1 went down to -1800 and after episode 20 the rewards goes up to -600 and after episode 100 the rewards goes down to around -1500 and the rewards flatten out up to episode 500 and rise again in rewards and goes up to -590 at episode 950. In case of repetition 2 the rewards start from around -1850 in the initial episodes and around episode 100 the rewards rise up to around -780 and after episode 150 the rewards decline to about -1600 and maintains the rewards throughout. Thus, the algorithm tries to learn better rewards but sometimes the rewards fall ending up learning badly and then learns better later. The regression line in repetition 1 starts around -1350 and goes up to -1100 and the slope is 0.339. The regression line in repetition 2 starts around -1370 and goes down to -1650 and the slope is -0.305.



(a) Cumulative rewards for repetition 1.



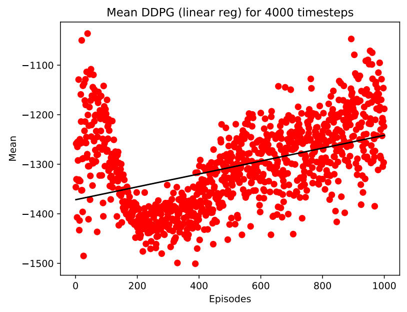
(b) Cumulative rewards for repetition 2.

Figure 5.33: Cumulative rewards (blue circles) for repetitions 1 and 2 for 4000 time steps using DDPG in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

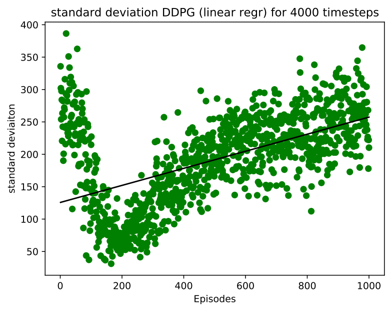
Figure 5.34 shows the average of cumulative rewards and the standard deviation of cumulative rewards across 10 repetitions for 4000 time steps. In the graph of average of cumulative rewards initially the average rewards starts from around -1390 and the mean rise to around -1050 in episode 50 and then the mean decline to -1500 around episode 400 and later the mean rise up to -1070 around episode 850. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 25 and in the initial episodes the standard deviation rise up to 380 and decline quickly to around 40 in episode 180 and then it rise again up to 360 around episode 950. The regression line in average cumulative rewards starts around -1370 and goes up to -1250 and the slope is 0.130. The regression line in standard deviation of cumulative rewards starts around 125 and goes up to 250 and the slope is 0.131.

Overall Results of Mean and Standard Deviation of Cumulative Rewards

Figure 5.35 shows the average of cumulative rewards and the standard deviation of cumulative rewards across 30 repetitions. In the graph of average of cumulative



(a) Average of cumulative rewards.



(b) Deviation of cumulative rewards.

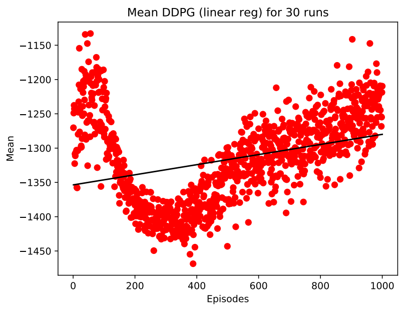
Figure 5.34: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 10 repetitions for 4000 time steps using DDPG in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

rewards initially the average rewards starts from around -1350 and the mean rise to around -1145 in episode 80 and then the mean decline to -1460 around episode 400 and later the mean rise up to -1150 around episode 850. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 220 and in the initial episodes the standard deviation rise up to 330 and decline quickly to around 60 in episode 200 and then it rise again up to 320 around episode 900. The regression line in average cumulative rewards starts around -1350 and goes up to -1290 and the slope is 0.074. The regression line in standard deviation of cumulative rewards starts around 140 and goes up to 250 and the slope is 0.128.

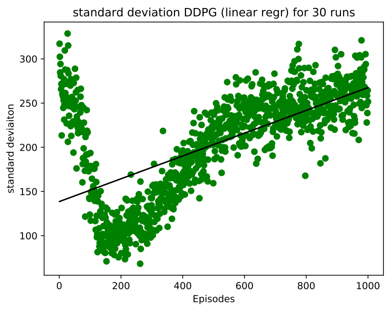
5.2.3 Random Action Controller

2000 Time Steps

Figure 5.36 shows cumulative rewards for repetitions 1 and 2 for 2000 time steps. The rest of the repetitions are shown in Figure A.16 from the appendix. As in



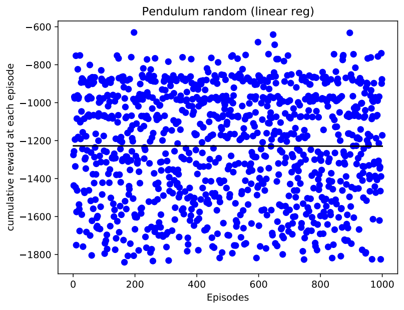
(a) Average of cumulative rewards.



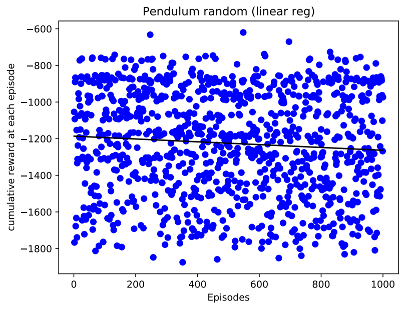
(b) Deviation of cumulative rewards.

Figure 5.35: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 30 repetitions using DDPG in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

Figure 5.36a and Figure 5.36b it is seen that for repetition 1 and repetition 2 throughout the episodes the cumulative rewards went down to around -1800 and occasionally the rewards went up to -600 around episodes 200, 850 for repetition 1 and -620 around episodes 250, 500 for repetition 2. Thus, the controller does not learn much as it consistently gives poor rewards. The regression line in repetition 1 is around -1200 and slope is -0.001. The regression line in repetition 2 starts around -1200 and the goes down to -1250 and the slope is -0.077.



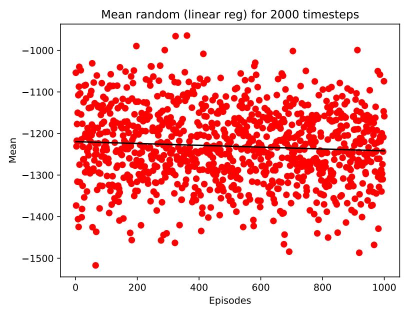
(a) Cumulative rewards for repetition 1.



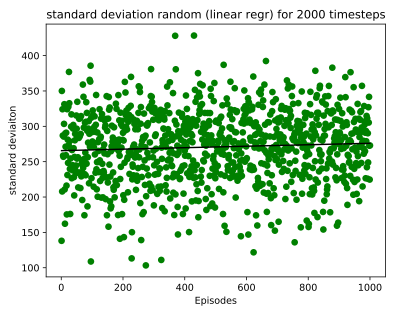
(b) Cumulative rewards for repetition 2.

Figure 5.36: Cumulative rewards (blue circles) for repetitions 1 and 2 for 2000 time steps using random action controller in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

Figure 5.37 shows average of cumulative rewards and the standard deviation of cumulative rewards across 10 repetitions for 2000 time steps. In the graph of average of cumulative rewards initially the average rewards starts from around -1400 and most of the average ranges between -1200 and -1300 across most of the episodes and some of the average cumulative reward goes down to -1550 as it can be seen around 50 episode and some of the average rewards goes up to -900 as it can be seen around episodes 300, 350. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 130 and most of the standard deviation ranges between 225 and 325 across most of the episodes and some of the standard deviation of cumulative reward goes down to 100 as it can be seen around episode 250 and some of the standard deviation of rewards goes up to 450 as it can be seen around episodes 370 and 420. The linear regression line of average of cumulative rewards starts around -1220 and goes down to -1250 and the slope is -0.022 and the linear regression line of standard deviation of cumulative rewards starts around 260 and goes up to 275 and the slope is 0.010.



(a) Average of cumulative rewards.

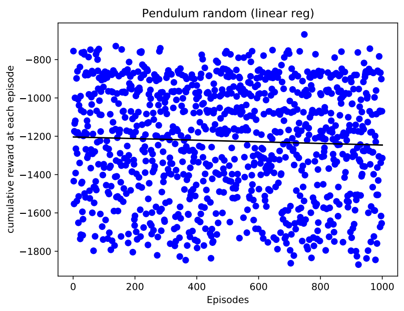


(b) Deviation of cumulative rewards.

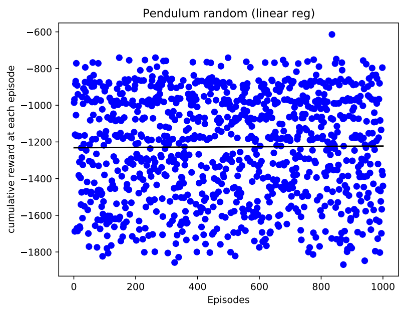
Figure 5.37: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 10 repetitions for 2000 time steps using random action controller in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

3000 Time Steps

Figure 5.38 cumulative rewards for repetitions 1 and 2 for 3000 time steps. The rest of the repetitions are shown in Figure A.17 from the appendix. As in Figure 5.38a and Figure 5.38b it is seen that for repetition 1 and repetition 2 throughout the episodes the cumulative rewards went down to around -1800 and occasionally the rewards went up to -700 around episode 700 for repetition 1 and -600 around episode 820 for repetition 2. Thus, the controller does not learn much as it consistently gives poor rewards. The regression line in repetition 1 is around starts around -1200 and goes down to -1250 and the slope is -0.042. The regression line in repetition 2 is around -1250 and slope is 0.008.



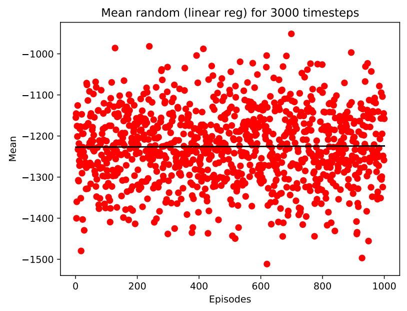
(a) Cumulative rewards for repetition 1.



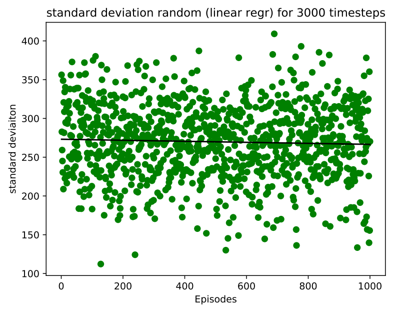
(b) Cumulative rewards for repetition 2.

Figure 5.38: Cumulative rewards (blue circles) for repetitions 1 and 2 for 3000 time steps using random action controller in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

Figure 5.39 shows the average of cumulative rewards and the standard deviation of cumulative rewards across 10 repetitions for 3000 time steps. In the graph of average of cumulative rewards initially the average rewards starts from around -1400 and most of the average ranges between -1180 and -1320 across most of the episodes and some of the average cumulative reward goes down to -1550 as it can be seen around episode 620 and some of the average rewards goes up to -950 as it can be seen around episode 650. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 275 and most of the standard deviation ranges between 250 and 325 across most of the episodes and some of the standard deviation of cumulative reward goes down to 110 as it can be seen around episode 170 and some of the standard deviation of rewards goes up to 410 as it can be seen around episode 650. The linear regression line of average of cumulative rewards is around -1225 and the slope is 0.003 and the linear regression line of standard deviation of cumulative rewards starts around 270 and goes down to 260 and the slope is -0.006.



(a) Average of cumulative rewards.



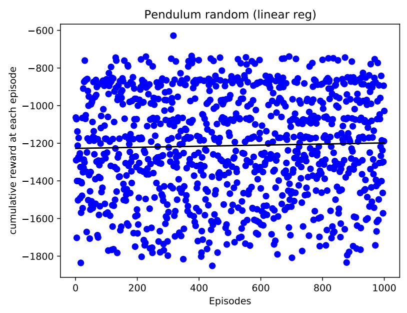
(b) Deviation of cumulative rewards.

Figure 5.39: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 10 repetitions for 3000 time steps using random action controller in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

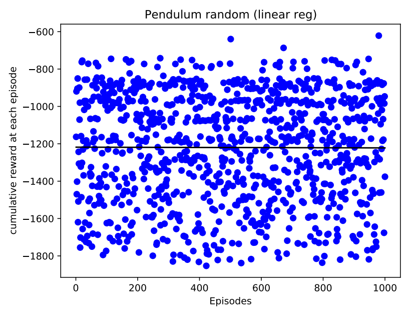
4000 Time Steps

Figure 5.40 shows cumulative rewards for repetitions 1 and 2 for 4000 time steps. The rest of the repetitions are shown in Figure A.18 from the appendix. As in Figure 5.40a and Figure 5.40b it is seen that for repetition 1 and repetition 2 throughout the episodes the cumulative rewards went down to around -1800 and occasionally the rewards went up to -620 around episode 300 for repetition 1 and

-600 around episode 950 for repetition 2. Thus, the controller does not learn much as it consistently gives poor rewards. The regression line in repetition 2 is around -1220 and slope is -0.004. The regression line in repetition 1 starts around -1220 and the goes up to -1200 and the slope is 0.029.



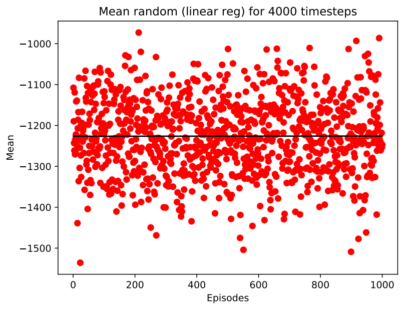
(a) Cumulative rewards for repetition 1.



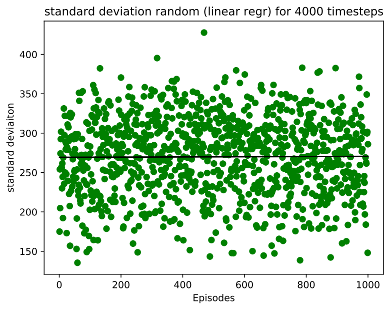
(b) Cumulative rewards for repetition 2.

Figure 5.40: Cumulative rewards (blue circles) for repetitions 1 and 2 for 4000 time steps using random action controller in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

Figure 5.41 shows the average of cumulative rewards and the standard deviation of cumulative rewards across 10 repetitions for 4000 time steps. In the graph of average of cumulative rewards initially the average rewards starts from around -1200 and most of the average ranges between -1200 and -1300 across most of the episodes and some of the average cumulative reward goes down to -1550 as it can be seen around 20 episode and some of the average rewards goes up to -970 as it can be seen around episode 200. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 175 and most of the standard deviation ranges between 225 and 325 across most of the episodes and some of the standard deviation of cumulative reward goes down to 125 as it can be seen around episode 50 and some of the standard deviation of rewards goes up to 440 as it can be seen around episode 450. The linear regression line of average of cumulative rewards is around -1220 and the slope is 0.001 and the linear regression line of standard deviation of cumulative rewards is around 270 and the slope is 0.001.



(a) Average of cumulative rewards.



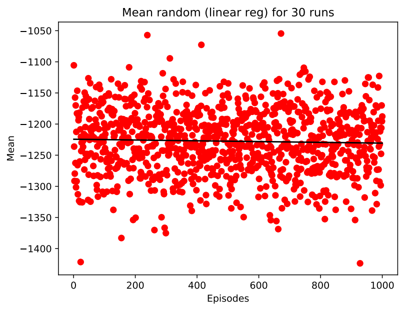
(b) Deviation of cumulative rewards.

Figure 5.41: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 10 repetitions for 4000 time steps using random action controller in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

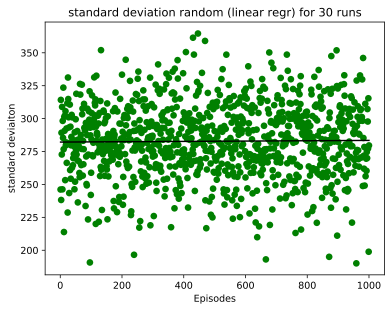
Overall Results of Mean and Standard Deviation of Cumulative Rewards

Figure 5.42 shows average of cumulative rewards and the standard deviation of cumulative rewards across 30 repetitions. In the graph of average of cumulative rewards initially the average rewards starts from around -1240 and most of the average ranges between -1150 and -1300 across most of the episodes and some of the average cumulative reward goes down to -1430 as it can be seen around episodes 20, 950 and some of the average rewards goes up to -1050 as it can be seen around episodes 150, 650. In the graph of standard deviation of cumulative rewards initially the standard deviation starts from around 270 and most of the standard deviation ranges between 275 and 300 across most of the episodes and some of the standard deviation of cumulative reward goes down to 180 as it can be seen around episode 100 and some of the standard deviation of rewards goes up to

380 as it can be seen around episode 390. The linear regression line of average of cumulative rewards starts around -1225 and goes down to -1230 and the slope is -0.006 and the linear regression line of standard deviation of cumulative rewards starts around 280 and goes up to 285 and the slope is 0.001.



(a) Average of cumulative rewards.



(b) Deviation of cumulative rewards.

Figure 5.42: Average of cumulative rewards (red circles) and standard deviation of cumulative rewards (green circles) across 30 repetitions using random action controller in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.

5.3 Overall Results

In this section a comparison of mean of mean of cumulative rewards across 10 repetitions and mean of standard deviation of cumulative rewards across 10 repetitions for various time steps 2000, 3000, 4000 and mean of mean of cumulative rewards across 30 repetitions and mean of standard deviation of cumulative rewards across 30 repetitions for Stochastic Synapse Reinforcement Learning, Deep Deterministic Policy Gradient and random action controller in the delayed-reinforcement pendulum environment and mountain car continuous environment is made.

Table 5.1 shows results for the mountain car continuous environment., we can observe that Stochastic Synapse Reinforcement Learning has better mean of mean of cumulative rewards across all the experimental conditions compared to the other two algorithms. On the other hand, the random action controller has lower mean of standard deviation of cumulative rewards across all the experimental conditions compared to the other two algorithms.

Table 5.2 shows results for the delayed-reinforcement pendulum environment., we can observe that the random action controller has better mean of mean of cumulative rewards across all the experimental conditions compared to the other two algorithms. On the other hand, Stochastic Synapse Reinforcement Learning has lower mean of standard deviation of cumulative rewards on experimental conditions 2000 and 3000 time steps, whereas Deep Deterministic Policy Gradient has lower mean of standard deviation of cumulative rewards on experimental conditions 4000 time steps and overall results.

Algorithm	Experimental condition	mean of mean of cumulative rewards	mean of standard deviation of cumulative rewards
SSRL	2000 timesteps	**-4.73**	4.73
	3000 timesteps	**-5.00**	4.53
	4000 timesteps	**-2.55**	2.72
	Overall results	**-4.21**	4.53
DDPG	2000 timesteps	-13.21	24.69
	3000 timesteps	-6.55	14.38
	4000 timesteps	-7.63	17.63
	Overall results	-9.13	20.61
Random	2000 timesteps	-33.23	**1.05**
	3000 timesteps	-33.24	**0.99**
	4000 timesteps	-33.26	**1.01**
	Overall results	-33.24	**1.16**

Table 5.1: Comparison of the mean of average of cumulative rewards and the mean of standard deviation of cumulative rewards for different time steps and overall 30 repetitions across different controllers in the mountain car continuous environment.

Algorithm	Experimental condition	mean of mean of cumulative rewards	mean of standard deviation of cumulative rewards
SSRL	2000 timesteps	-1387.09	**191.26**
	3000 timesteps	-1390.73	**190.15**
	4000 timesteps	-1384.15	194.95
	Overall results	-1387.32	203.70
DDPG	2000 timesteps	-1344.35	197.87
	3000 timesteps	-1299.34	193.92
	4000 timesteps	-1306.54	**191.60**
	Overall results	-1316.75	**203.11**
Random	2000 timesteps	**-1230.62**	270.82
	3000 timesteps	**-1225.67**	269.85
	4000 timesteps	**-1226.19**	269.91
	Overall results	**-1227.49**	282.82

Table 5.2: Comparison of the mean of average of cumulative rewards and the mean of standard deviation of cumulative rewards for different time steps and overall 30 repetitions across different controllers in the delayed-reinforcement pendulum environment.

Chapter 6

Discussions

This chapter discusses the similarities and differences in cumulative rewards for different repetitions across different time steps and also speculates as to the reasons for the similarities and differences. This chapter then discusses the trends, similarities, and differences across the different algorithms and speculates as to the reasons for them. The above discussion is done both for the mountain car continuous and delayed-reinforcement pendulum environments. Finally, the chapter discusses the similarities and differences across environments and speculates as to the reasons for them.

6.1 Mountain Car Continuous Environment

This section discusses the similarities and differences in cumulative rewards for different repetitions across different time steps and also speculates as to the reasons for the similarities and differences. Then it discusses the trends, similarities, and differences across the different algorithms and speculates as to the reasons for them on the mountain car continuous environment.

6.1.1 Stochastic Synapse Reinforcement Learning

2000 Time Steps

From Figure 5.1 and Figure A.1 it can be seen that for 7 out of 10 repetitions the cumulative rewards rises steadily and the algorithm consistently maintains the best reward it could achieve throughout the episodes. The trends of these repetitions are positive since it can be seen that the slope of the regression line has a positive slope showing the algorithm learns to get better rewards or at-least maintain the best rewards. There are 3 repetitions (Repetition 3, Repetition 5, Repetition 8) where the rewards slightly drop. Even-though SSRL manages not to get poor rewards its performance slightly drops from the best rewards received in previous episodes. The reason for the better trend may be because of the eligibility traces that consistently traces out the synapses that helped in gaining rewards and uses those synapses for better learning. Another reason might be because of the standard deviation which acts as a control for exploration and exploitation. When the rewards are rising initially the standard deviation may be larger which allows for more exploration and once it reaches the best rewards the standard deviation became small allowing for exploitation and the rewards are maintained consistent. The reason for the decline in reward may be because sometimes the algorithm may encounter certain bad observations from the environment which may disrupt the flow resulting in earning slightly poorer rewards. In such case, the value of standard deviation may become higher which means it has to do some exploration before it finds better rewards again.

3000 Time Steps

From Figure 5.3 and Figure A.2 it can be seen that for 7 out of 10 repetitions the cumulative rewards rises steadily and the algorithm consistently maintains the best reward it could achieve throughout the episodes. The trends of these repetitions are positive since it can be seen that the slope of the regression line has a positive slope showing the algorithm learns to get better rewards or at-least maintain the best rewards. There are 3 repetitions (Repetition 3, Repetition 6, Repetition 8) where the rewards slightly drop. Even-though SSRL manages not to get poor rewards its performance slightly drops from the best rewards received in previous episodes. The reason for the better trend may be because of the eligibility traces that consistently traces out the synapses that helped in gaining rewards and uses those synapses for better learning. Another reason might be because of the standard deviation which acts as a control for exploration and exploitation. When the rewards are rising initially the standard deviation may be larger which allows for more exploration and once it reaches the best rewards the standard deviation became small allowing for exploitation and the rewards are maintained consistent. The reason for the decline in reward may be because sometimes the algorithm may encounter certain bad observations from the environment which may disrupt the flow resulting in earning slightly poorer rewards. In such case, the value of standard deviation may become higher which means it has to do some exploration before it finds better rewards again.

4000 Time Steps

From Figure 5.5 and Figure A.3 it can be seen that for 6 out of 10 repetitions the cumulative rewards rises steadily and the algorithm consistently maintains the best reward it could achieve throughout the episodes. The trends of these repetitions are positive since it can be seen that the slope of the regression line has a positive slope showing the algorithm learns to get better rewards or at-least maintain the best rewards. There are 4 repetitions (Repetition 2, Repetition 3, Repetition 6, Repetition 9) where the rewards slightly drop. Even-though SSRL manages not to get poor rewards its performance slightly drops from the best rewards received in previous episodes. The reason for the better trend may be because of the eligibility traces that consistently traces out the synapses that helped in gaining rewards and uses those synapses for better learning. Another reason might be because of the standard deviation which acts as a control for exploration and exploitation. When the rewards are rising initially the standard deviation may be larger which allows for more exploration and once it reaches the best rewards the standard deviation became small allowing for exploitation and the rewards are maintained consistent. The reason for the decline in reward may be because sometimes the algorithm may encounter certain bad observations from the environment which may disrupt the flow resulting in learning slightly poorer rewards. In such case, the value of standard deviation may become higher which means it has to do some exploration before it finds better rewards again.

Comparison Across Time Steps

When comparing the trends of cumulative rewards across time steps it can be observed that in each group the number of repetitions showing a positive trend are greater than the number of repetitions showing negative trends. The reasons for such a positive trend are explained in the above paragraphs. Another observation is that the cumulative rewards are higher in case of 4000 time steps compared to other time steps as it can be seen on Table 5.1 where it has a mean of mean of cumulative rewards of -2.55 which is greater compared to mean of mean in other time steps. The reason may be because as the number of time steps increases the algorithm is exposed to more observations which makes it experienced in dealing with the environment. On the downside since it is exposed more to the environment there is a higher chance of getting exposed to bad or irregular observations more. This might be the reason for sometimes getting poor rewards compared to other time steps.

6.1.2 Deep Deterministic Policy Gradient

2000 Time Steps

From Figure 5.8 and Figure A.4 it can be seen that in most of the repetitions the cumulative reward rise to its best reward quickly but the algorithm gets poor rewards immediately and in most of the repetitions even though it receives very poor rewards it manages to learn well again receiving better rewards. But some repetitions like repetition 1, repetition 2, repetition 3 once it declines it is unable learn well like in previous episodes. The reason may be because of the replay buffer size used in DDPG. Once the memory is full the replay buffer adds new transitions but it pops out the old transitions. So there is chance that some transitions that are popped out may contain better experiences for receiving better rewards earlier in the episodes. Another reason for such inconsistency may be the batch size used in the algorithm that may not have enough information about the previous actions and states as the episodes progresses since the transitions in the batch are picked in random.

3000 Time Steps

From Figure 5.10 and Figure A.5 it can be seen that in most of the repetitions the cumulative reward rise to its best reward quickly but the algorithm gets poor rewards immediately and in most of the repetitions even though it receives very poor rewards it manages to learn well again receiving better rewards. But some repetitions like repetition 2, repetition 10 once it declines it is unable learn well like in previous episodes. The reason may be because of the replay buffer size used in DDPG. Once the memory is full the replay buffer adds new transitions but it pops out the old transitions. So there is chance that some transitions that are popped out may contain better experiences for receiving better rewards earlier in the episodes. Another reason for such inconsistency may be the batch size used in the algorithm that may not have enough information about the previous actions and states as the episodes progresses since the transitions in the batch are picked in random.

4000 Time Steps

From Figure 5.12 and Figure A.6 it can be seen that in most of the repetitions the cumulative reward rise to its best reward quickly but the algorithm gets poor rewards immediately and in most of the repetitions even though it receives very poor rewards it manages to learn well again receiving better rewards. But some repetitions like repetition 8 once it declines it is unable learn well like in previous episodes. The reason may be because of the replay buffer size used in DDPG. Once the memory is full the replay buffer adds new transitions but it pops out the old transitions. So there is chance that some transitions that are popped out may contain better experiences for receiving better rewards earlier in the episodes. Another reason for such inconsistency may be the batch size used in the algorithm that may not have enough information about the previous actions and states as the episodes progresses since the transitions in the batch are picked in random.

Comparison Across Time Steps

When comparing the trends of cumulative rewards across time steps it can be observed that in each group the number of repetitions showing a negative trend (negative slope of the regression line) are greater than the number of repetitions showing a positive trend (positive slope of the regression line) and the reasons are speculated in the previous paragraphs. Another observation is that the cumulative rewards are higher for 3000 time steps compared to the 4000 time steps as it can be seen on Table 5.1 where it has a mean of mean of cumulative rewards of -6.55 which is greater compared to mean of mean in other time steps. This may be because in case of 4000 times steps the buffer gets full quickly. So once it is full the buffer will pop out the transitions which may contain some experiences which gave better rewards which results in learning poorly.

6.1.3 Random Action Controller

When comparing the trends of cumulative rewards across time steps it can be observed that in each group there is no real change in trend as there is no real learning done in this controller as the actions are selected in random by uniform distribution and it can be seen in Table 5.1 that the mean of mean of cumulative rewards across different time steps are almost same which shows no real change across time steps or even within the repetitions of different set of time steps.

6.1.4 Comparison Across Algorithms

When compared across the algorithms it is observed that SSRL performs consistently compared to DDPG on the mountain car continuous environment may be because of the eligibility traces that consistently trace out the synapses that helped in gaining rewards and uses those synapses for better learning. Another reason might be because of the standard deviation which acts as a control knob for exploration and exploitation. When the rewards are rising initially the standard deviation may be larger which allows for more exploration and once it reaches the best rewards the standard deviation became small allowing for exploitation and the rewards are maintained consistently. In the case of DDPG, it gives rewards that are substantially higher than that of SSRL but due to its inconsistency the mean of mean of cumulative rewards are higher for SSRL compared to DDPG which is shown in Table 5.1. In the case of the random action controller, although it has minimum mean of standard deviation of cumulative rewards it is not considered beneficial because most of the rewards lie between the mean of mean of rewards and this mean is poor compared to SSRL and DDPG.

6.2 Delayed-Reinforcement Pendulum Environment

This section discusses the similarities and differences in cumulative rewards for different repetitions across different time steps and also speculates as to the reasons for the similarities and differences. Then it discusses the trends, similarities, and differences across the different algorithms and speculates as to the reasons for them on the delayed-reinforcement pendulum environment.

6.2.1 Stochastic Synapse Reinforcement Learning

2000 Time Steps

From Figure 5.22 and Figure A.10 it can be seen that in 9 out of 10 repetitions the trends are negative (slope of the linear regression lines are negative). The algorithm occasionally give better rewards but there is fluctuation in receiving the the rewards as the rewards go up and come down. But even though the algorithm shows fluctuations in learning it manages not to receive poor rewards. The fluctuations in rewards may be because the value of the standard deviation remains larger which results in more exploration and it could not find a suitable state where it gets satisfactory rewards. The reason for not receiving poor rewards is may be because the algorithm manages to trace somewhat suitable synapses where the actions from those synapses do not give poor rewards.

3000 Time Steps

From Figure 5.24 and Figure A.11 it can be seen that in 8 out of 10 repetitions the trends are negative (slope of the linear regression lines are negative). The algorithm occasionally give better rewards but there is fluctuation in receiving the the rewards as the rewards go up and come down. But even though the algorithm shows fluctuations in learning it manages not to receive poor rewards. The fluctuations in rewards may be because the value of the standard deviation remains larger which results in more exploration and it could not find a suitable state where it gets satisfactory rewards. The reason for not receiving poor rewards is may be because the algorithm manages to trace somewhat suitable synapses where the actions from those synapses do not give poor rewards.

4000 Time Steps

From Figure 5.26 and Figure A.12 it can be seen that in 8 out of 10 repetitions the trends are negative (slope of the linear regression lines are negative). The algorithm occasionally give better rewards but there is fluctuation in receiving the the rewards as the rewards go up and come down. But even though the algorithm shows fluctuations in learning it manages not to receive poor rewards. The fluctuations in rewards may be because the value of the standard deviation remains larger which results in more exploration and it could not find a suitable state where it gets satisfactory rewards. The reason for not receiving poor rewards is may be because the algorithm manages to trace somewhat suitable synapses where the actions from those synapses do not give poor rewards.

Comparison Across Time Steps

When comparing the trends of cumulative rewards across time steps it can be observed that the cumulative rewards are higher in case of 4000 time steps compared to other time steps as it can be seen on Table 5.2 where it has a mean of mean of cumulative rewards of -1384.15 which is greater compared to mean of mean in other time steps. The reason may be because as the number of time steps increases the algorithm is exposed to more observations which makes it experienced in dealing with the environment.

6.2.2 Deep Deterministic Policy Gradient

2000 Time Steps

From Figure 5.29 and Figure A.13 it can be seen that in 7 out of 10 repetitions the the trends are positive (slope of the linear regression lines are positive). The cumulative reward rises in the initial episodes and falls off to certain number of episodes but picks up again to give better rewards. But in some repetitions like Repetition 1, Repetition 6, Repetition 9 once it falls it is unable to get back and learn well like in previous episodes. The reason for picking up of rewards after falling off may be because in the batch the transitions are chosen in random and in that chosen samples it may have actions that enhances the reward. The initial fall off may be because during those episodes the memory might have become full resulting in popping the old transitions and the algorithm may be forced to choose the actions that might not give good rewards.

3000 Time Steps

From Figure 5.31 and Figure A.14 it can be seen that in 6 out of 10 repetitions the the trends are positive (slope of the linear regression lines are positive). The cumulative reward rises in the initial episodes and falls off to certain number of episodes but picks up again to give better rewards. But in some repetitions like Repetition 1, Repetition 7 once it falls it is unable to get back and learn well like in previous episodes. The reason for picking up of rewards after falling off may be because in the batch the transitions are chosen in random and in that chosen samples it may have actions that enhances the reward. The initial fall off may be because during those episodes the memory might have become full resulting in popping the old transitions and the algorithm may be forced to choose the actions that might not give good rewards.

4000 Time Steps

From Figure 5.33 and Figure A.15 it can be seen that in 8 out of 10 repetitions the the trends are positive (slope of the linear regression lines are positive). The cumulative reward rises in the initial episodes and falls off to certain number of episodes but picks up again to give better rewards. But in some repetitions like Repetition 2, Repetition 8 once it falls it is unable to get back and learn well like in previous episodes. The reason for picking up of rewards after falling off may be because in the batch the transitions are chosen in random and in that chosen samples it may have actions that enhances the reward. The initial fall off may be because during those episodes the memory might have become full resulting in popping the old transitions and the algorithm may be forced to choose the actions that might not give good rewards.

Comparison Across Time Steps

When comparing the trends of cumulative rewards across time steps it can be observed that in each group the number of repetitions showing a positive trend (positive slope of the regression line) are greater than the number of repetitions showing a negative trend (negative slope of the regression line) and the reasons are speculated in the previous paragraphs. Another observation is that the cumulative rewards are higher for 3000 time steps compared to the 4000 time steps as it can be seen in Table 5.2 where it has a mean of mean of cumulative rewards of -1299.34 which is greater compared to mean of mean in other time steps. This may be because in case of 4000 times steps the buffer gets full quickly. So once it is full the buffer will pop out the transitions which may contain some actions which gave better rewards which results in learning poorly.

6.2.3 Random Action Controller

When comparing the trends of cumulative rewards across time steps it can be observed that in each time steps there is no real change in trend as there is no real learning done in this controller as the actions are selected in random by uniform distribution. Even though the controller gives some better cumulative rewards it also consistently gives poor rewards across the episodes showing that there no real learning happening in the controller across time steps or even within the repetitions of different set of time steps.

6.2.4 Comparison Across Algorithms

When compared across the algorithms it is observed that SSRL consistently avoids getting to poor rewards compared to DDPG where in some repetitions the algorithm gets to poor rewards and in most of the repetitions in random action controller gets to poor rewards on delayed-reinforcement pendulum environment. This may be because of the eligibility traces that consistently traces out the synapses that helped in gaining rewards and uses those synapses for choosing actions that do not give poor rewards. In case of DDPG it gives rewards that are pretty high compared to that of SSRL which is shown in Table 5.2 where the average of average of rewards were better for DDPG compared to SSRL. In the case of the random action controller it gives better rewards across all the experimental conditions which is shown in Table 5.2 compared to both SSRL and DDPG but the average of deviation is very high which shows that it gives both good rewards and bad rewards consistently across the episodes showing the lack of ability to learn in the environment.

6.3 Comparison Across Environments

In spite of all the differences and similarities between the experimental conditions and the algorithms, when compared across the environments it can be seen that SSRL works consistently across the environments compared to DDPG and the DDPG gives better performances across the environments compared to SSRL. Even though these two algorithms have their own advantages and disadvantages when coming to delayed-reinforcement pendulum environment both algorithms do not perform well as can be seen in Table 5.2 where the mean of mean of rewards where higher for random action controller than SSRL and DDPG. The reason may be because the delayed-reinforcement pendulum environment keeps on swinging and it does not have definite terminal state, so the algorithms could not really find suitable actions for reaching a particular state. So, even though the algorithms earns high rewards temporarily, they tend to fall back.

Chapter 7

Conclusions and Future Work

This chapter is divided into two sections. The first section presents conclusions based on the research conducted for this thesis. The second section presents future work that arises from this thesis.

7.1 Conclusions

Many potential reinforcement learning environments are continuous, meaning that the states and potential actions are infinite. To deal with such environments, some type of approximation mechanism such as neural networks must be used along with the reinforcement learning. In this thesis, a comparison study has been conducted on two algorithms that use neural networks for reinforcement learning in continuous state and action spaces: Stochastic Synapse Reinforcement Learning (SSRL) and Deep Deterministic Policy Gradient (DDPG). The thesis presents the strengths and weaknesses of both algorithms on two continuous environments: the delayed-reinforcement pendulum environment and the mountain car continuous environment. The results collected are cumulative rewards for each repetition and average cumulative rewards across all the repetitions and the standard deviation of cumulative rewards across all the repetitions for different episode durations for each algorithm in each environment. Results are also collected for random action controller to establish a baseline on the trends. Based on the results obtained from the experiments and the discussion points made regarding the results several conclusions can be reached.

SSRL performs well when the number of time steps increases. The reason may be because as the number of time steps increases the algorithm is exposed to more observations which makes it experienced in dealing with the environment. SSRL works consistently across the environments compared to DDPG. This may be because of the eligibility traces that consistently traces out the synapses that helped in gaining rewards and uses those synapses for better learning. Another reason might be because of the standard deviation which acts as a control knob for exploration and exploitation. When the rewards are rising initially the standard deviation may be larger which allows for more exploration and once it reaches the best rewards the standard deviation becomes smaller allowing for exploitation and the rewards are maintained consistently. The reason for DDPG not being consistent may be because of the replay buffer size used in the algorithm. Once the memory is full the replay buffer retains new transitions but it pops out the old transitions. So there is chance that some of the transitions that are popped out may contain better actions that may have been the reason for getting better rewards earlier in the episodes. Another reason for such inconsistency is may be the batch size used in the algorithm may not have enough information about the previous actions and states as the episodes progress because the transitions in the batch are picked in random. Another conclusion that can be made is that DDPG gives more higher rewards compared to SSRL. The reason may be because when the buffer is not full the algorithm has high probability of considering older transitions too where some transitions may have the actions that give high rewards. Another conclusion can be made from this work is that in case of the delayed-reinforcement pendulum environment both algorithms did not perform well. The reason may be because the pendulum environment keeps on swinging and it does not have definite terminal state, so the algorithm could not really find the suitable action for reaching a particular state. So, even though the algorithm gives high rewards temporarily, it tends to decline in performance. All the claims made in this work are logical possibilities based on the observed behaviour of the algorithms but there is no definite analysis supporting the claims; that is left for future work. Also based on claims made it can be noticed that the hyper-parameters used in these algorithms (buffer size, batch size, standard deviation, expected rewards, eligibility traces, decay rates etc.) definitely affects the performance of the algorithms. determining how substantially these hyper-parameters affect the algorithms is also left as future work.

7.2 Future Work

Based on the research conducted in this thesis, the following future work arises:

1. Conduct an analysis on the conjectures made in the discussion regarding the performance of the learning algorithms and bring about new conjectures explaining the behaviours or bringing out new behaviours of SSRL and DDPG on the delayed-reinforcement pendulum and mountain car continuous environments.

2. In this thesis, hyper-parameters like discount factor, batch size, buffer size,

etc. are fixed throughout. Parameter sensitivity can be widely explored with different values on different hyper-parameters which might improve the working of the algorithms on continuous environments as well as in other environments.

3. One of the important calculations used in SSRL is for the determination of expected reward as it is used in updating the standard deviation which controls exploration and exploitation. A study could be conducted on how SSRL works on different environments when various equations of expected reward are used in the algorithm.

4. The algorithms in this thesis are studied on environments with both continuous actions and continuous states. But the algorithms can be made to learn on environments with discrete states and learn new the behaviours, the strengths and weaknesses of both algorithms.

5. Investigate the possibility of combining the strengths exhibited by both SSRL and DDPG and formulate an algorithm which might learn better when compared with both algorithms used in this work.

Bibliography

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, December 1989. ISSN 0932-4194. doi: 10.1007/BF02551274. URL `http://dx.doi.org/10.1007/BF02551274`.

Vijaykumar Gullapalli. A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, 3(6):671–692, 1990.

Vijaykumar Gullapalli. *Reinforcement learning and its application to control.* PhD thesis, University of Massachusetts at Amherst, 1992.

Roland Hafner and Martin Riedmiller. Reinforcement learning in feedback control: Challenges and benchmarks from technical process control. *Machine Learning*, 84:137–169, 07 2011. doi: 10.1007/s10994-011-5235-x.

Kur Hornik, Maxwell Stinchcombe, and Halber White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

Anil K. Jain, Jianchang Mao, and K. Moidin Mohiuddin. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, 1996.

Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, 32(11):1238–1274, 2013. doi: 10.1177/0278364913495721.

Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing systems (NIPS)*, pages 1008–1014, 2000. URL `http://papers.nips.cc/paper/1786-actor-critic-algorithms`.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances* *in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

Sascha Lange, Thomas Gabel, and Martin A. Riedmiller. Batch reinforcement learning. In *Reinforcement Learning*, 2012.

Su Young Lee, Choi Sungik, and Sae-Young Chung. Sample-efficient deep reinforcement learning via episodic backward update. In *Advances in Neural Information Processing Systems*, pages 2112–2121, 2019.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Warren Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 00280836. URL `http://dx.doi.org/10.1038/nature14236`.

Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 2004.

Irwin W Sandberg, James T Lo, Craig L Fancourt, Jose C Principe, Shigeru Katagiri, and Simon Haykin. *Nonlinear dynamical systems: feedforward neural network perspectives*, volume 21. John Wiley & Sons, 2001.

Syed Naveed Hussain Shah and Dean Frederick Hougen. Stochastic synapse reinforcement learning (SSRL). In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, 2017. doi: 10.1109/SSCI.2017.8285425.

Syed Naveed Hussain Shah and Dean Frederick Hougen. Improved stochastic synapse reinforcement learning for continuous actions in sharply changing environments. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020. doi: 10.1109/IJCNN48605.2020.9207622.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China, 22–24 Jun 2014. PMLR. URL `http://proceedings.mlr.press/v32/silver14.html`.

Richard S Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, 1984.

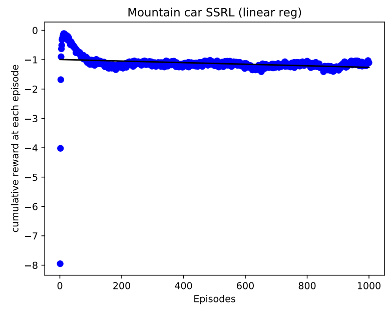
Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

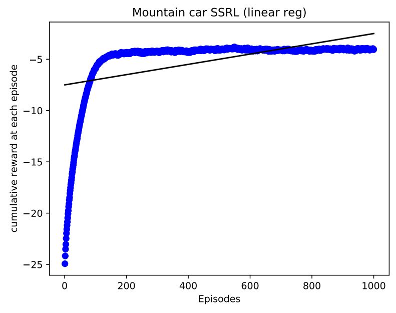
George E Uhlenbeck and Leonard S Ornstein. On the theory of the Brownian motion. *Physical review*, 36(5):823, 1930.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

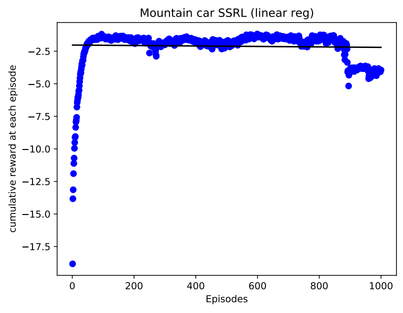
Appendix A

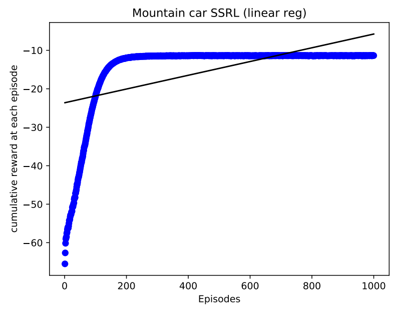
Other Results



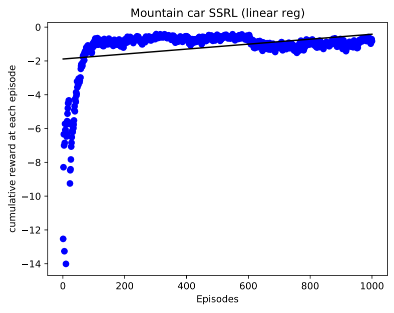


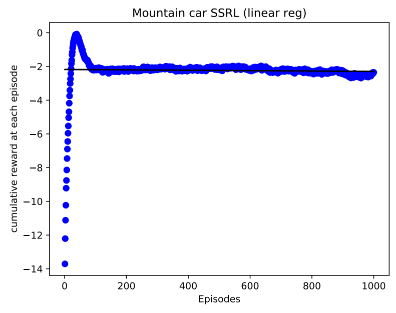
(a) Cumulative rewards for run 3. (b) Cumulative rewards for run 4.



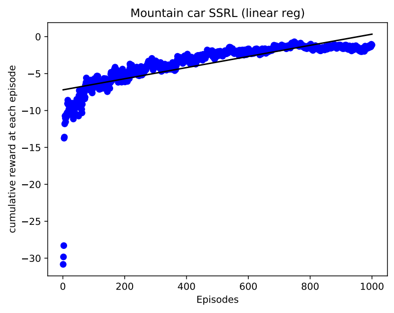


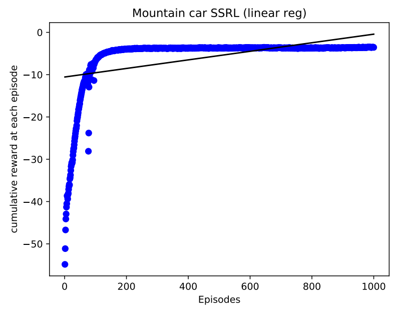
(c) Cumulative rewards for run 5. (d) Cumulative rewards for run 6.





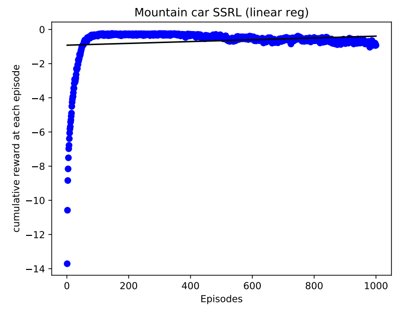
(e) Cumulative rewards run 7. (f) Cumulative rewards for run 8.



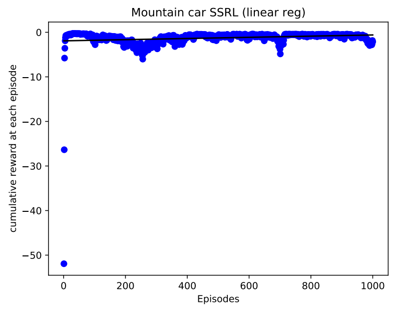


(g) Cumulative rewards for run 9. (h) Cumulative rewards for run 10.

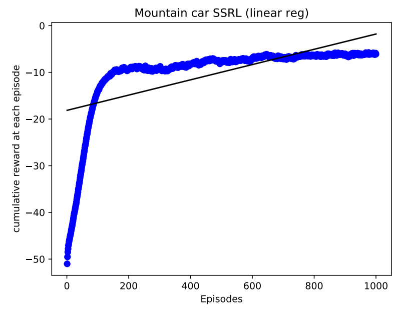
Figure A.1: Cumulative rewards (blue circles) for runs 3-10 for 2000 time steps using SSRL in mountain car continuous environment. The black line indicates the linear regression line.



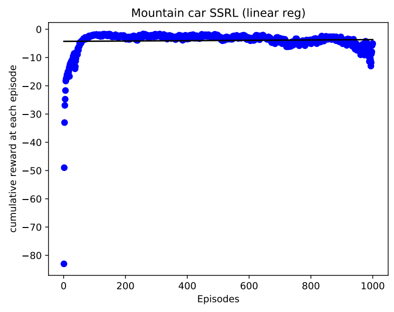
(a) Cumulative rewards for run 3.



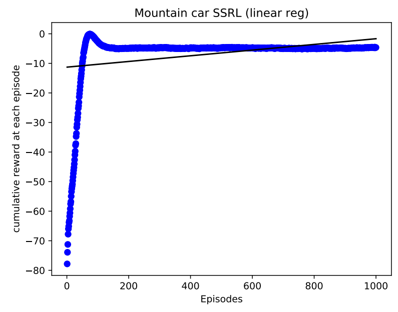
(b) Cumulative rewards for run 4.



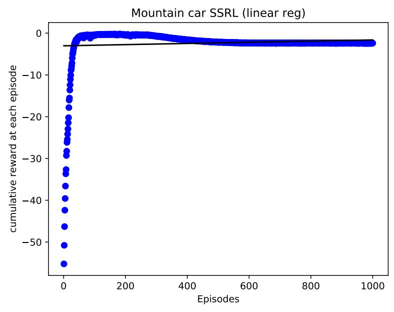
(c) Cumulative rewards for run 5.



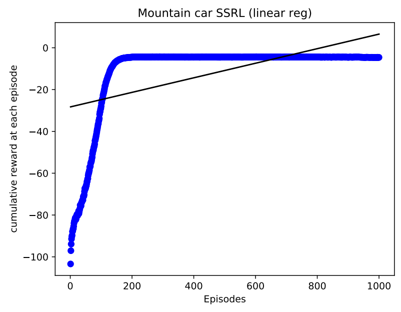
(d) Cumulative rewards for run 6.



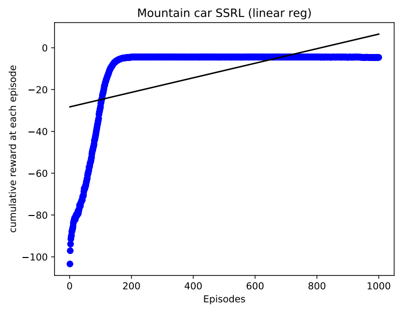
(e) Cumulative rewards for run 7.



(f) Cumulative rewards for run 8.

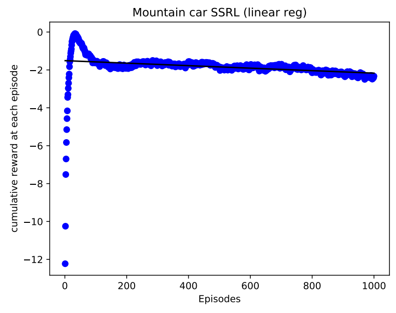


(g) Cumulative rewards for run 9.

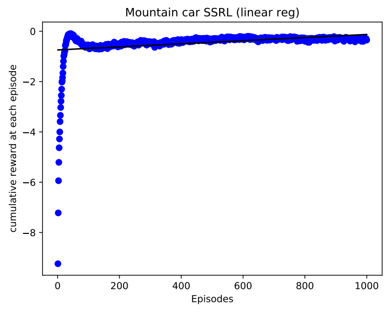


(h) Cumulative rewards for run 10.

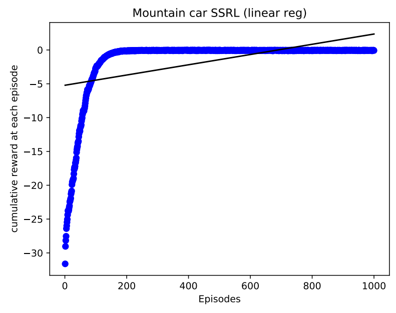
Figure A.2: Cumulative rewards (blue circles) for runs 3-10 for 3000 time steps using SSRL in mountain car continuous environment. The black line indicates the linear regression line.



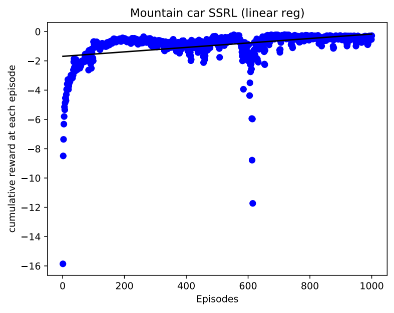
(a) Cumulative rewards for run 3.



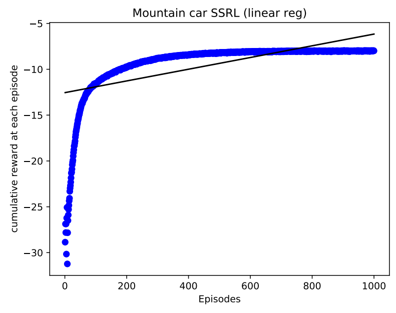
(b) Cumulative rewards for run 4.



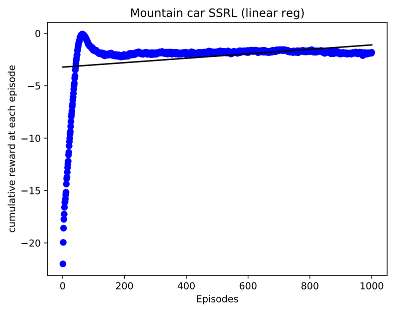
(c) Cumulative rewards for run 5.



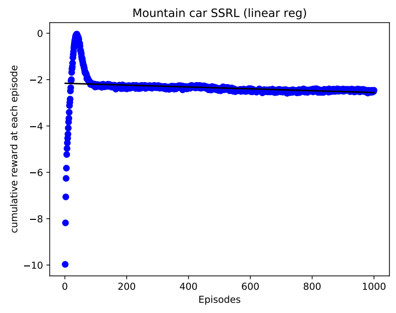
(d) Cumulative rewards for run 6.



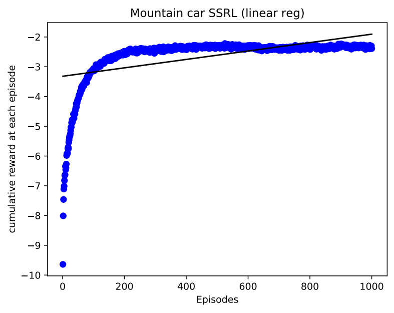
(e) Cumulative rewards for run 7.



(f) Cumulative rewards for run 8.

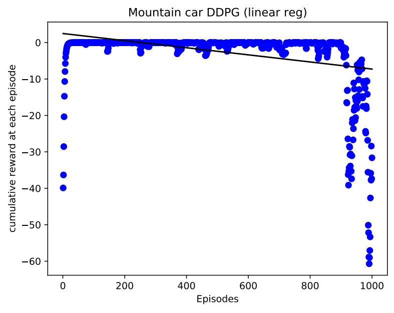


(g) Cumulative rewards for run 9.

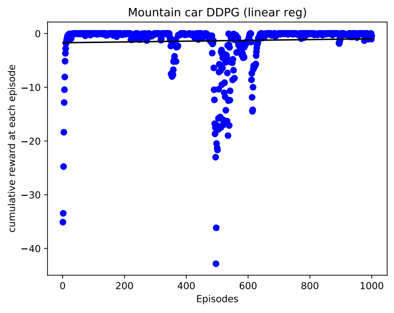


(h) Cumulative rewards for run 10.

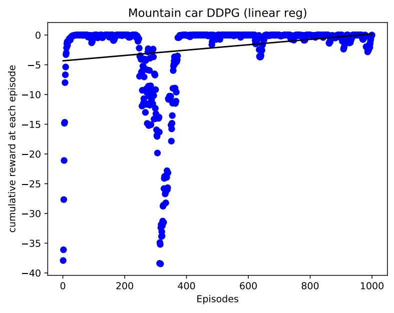
Figure A.3: Cumulative rewards (blue circles) for runs 3-10 for 4000 time steps using SSRL in mountain car continuous environment. The black line indicates the linear regression line.



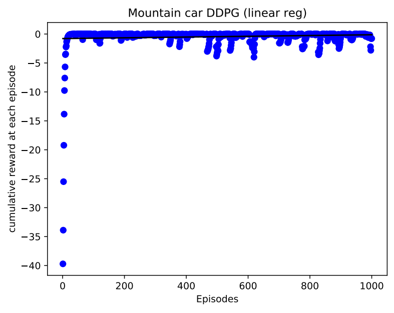
(a) Cumulative rewards for run 3.



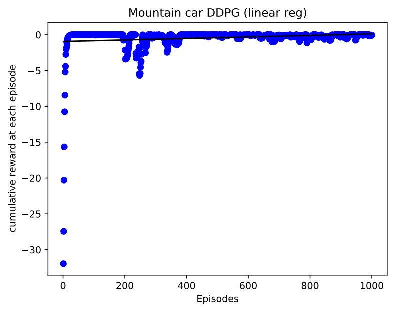
(b) Cumulative rewards for run 4.



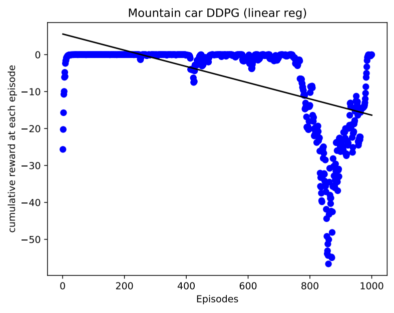
(c) Cumulative rewards for run 5.



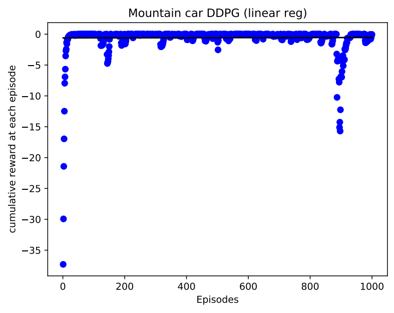
(d) Cumulative rewards for run 6.



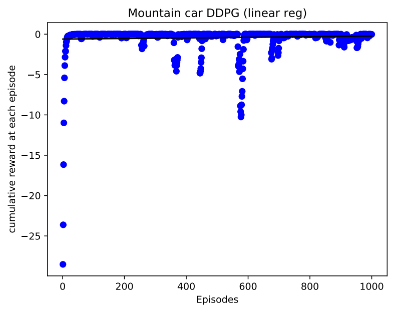
(e) Cumulative rewards for run 7.



(f) Cumulative rewards for run 8.

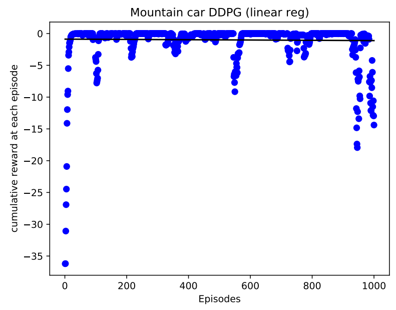


(g) Cumulative rewards for run 9.

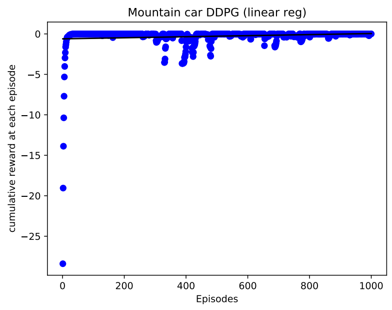


(h) Cumulative rewards for run 10.

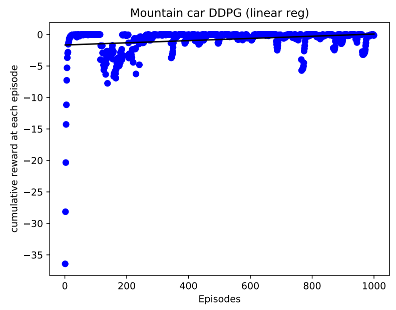
Figure A.4: Cumulative rewards (blue circles) for runs 3-10 for 2000 time steps using DDPG in mountain car continuous environment. The black line indicates the linear regression line.



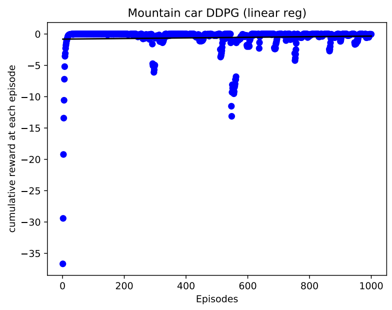
(a) Cumulative rewards for run 3.



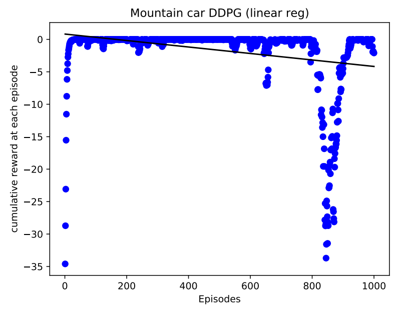
(b) Cumulative rewards for run 4.



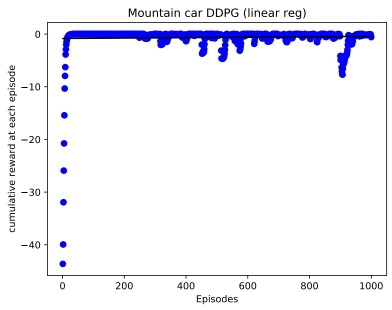
(c) Cumulative rewards for run 5.



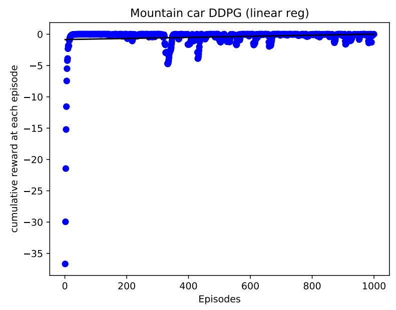
(d) Cumulative rewards for run 6.



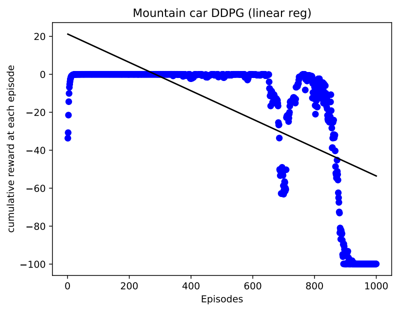
(e) Cumulative rewards for run 7.



(f) Cumulative rewards for run 8.

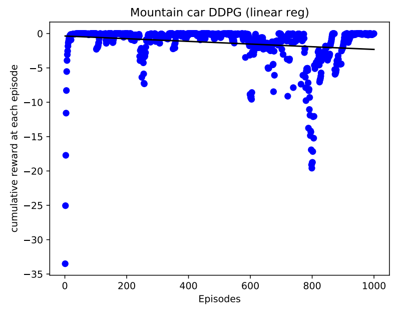


(g) Cumulative rewards for run 9.

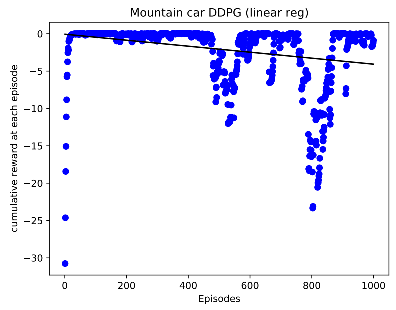


(h) Cumulative rewards for run 10.

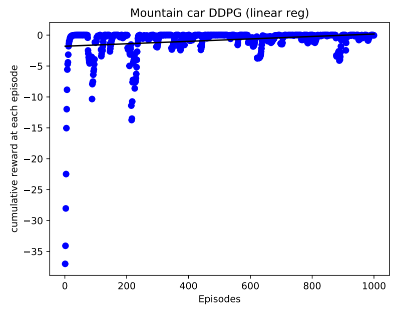
Figure A.5: Cumulative rewards (blue circles) for runs 3-10 for 3000 time steps using DDPG in mountain car continuous environment. The black line indicates the linear regression line.



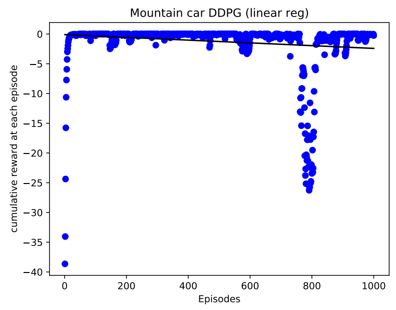
(a) Cumulative rewards for run 3.



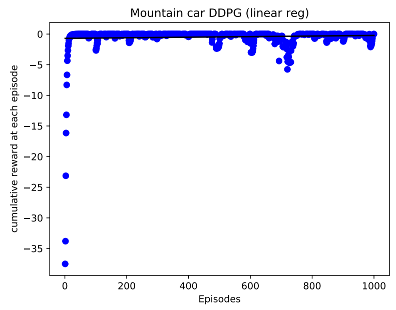
(b) Cumulative rewards for run 4.



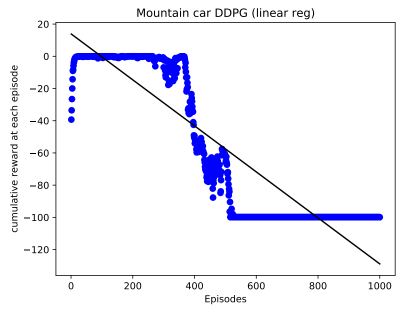
(c) Cumulative rewards for run 5.



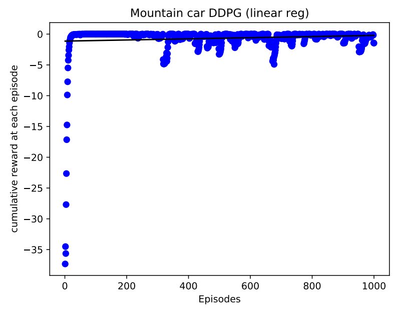
(d) Cumulative rewards for run 6.



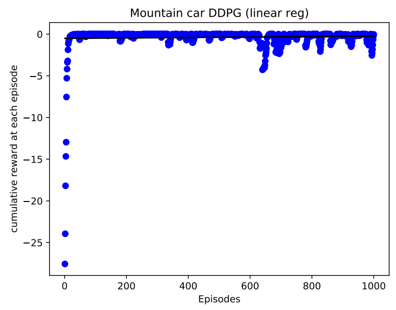
(e) Cumulative rewards for run 7.



(f) Cumulative rewards for run 8.

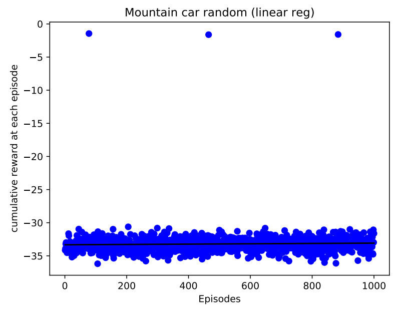


(g) Cumulative rewards for run 9.

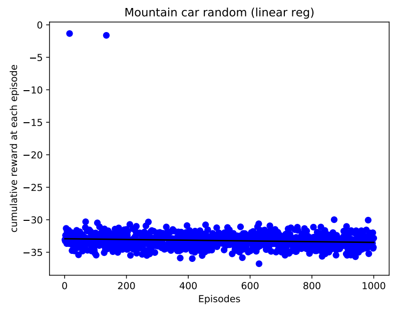


(h) Cumulative rewards for run 10.

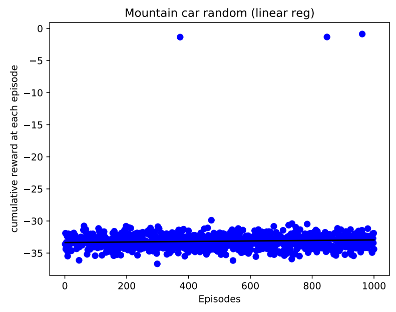
Figure A.6: Cumulative rewards (blue circles) for runs 3-10 for 4000 time steps using DDPG in mountain car continuous environment. The black line indicates the linear regression line.



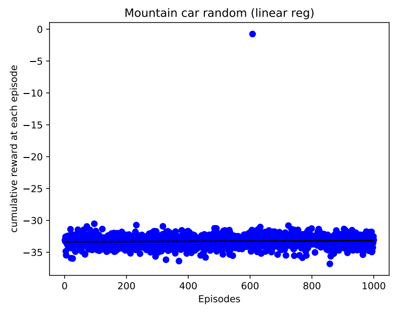
(a) Cumulative rewards for run 3.



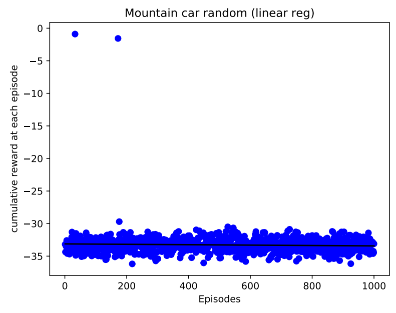
(b) Cumulative rewards for run 4.



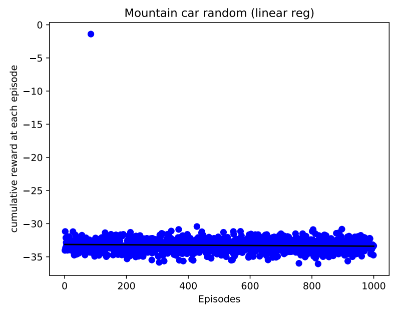
(c) Cumulative rewards for run 5.



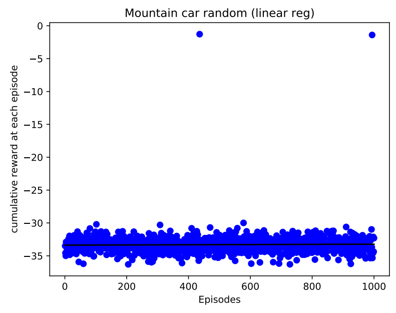
(d) Cumulative rewards for run 6.



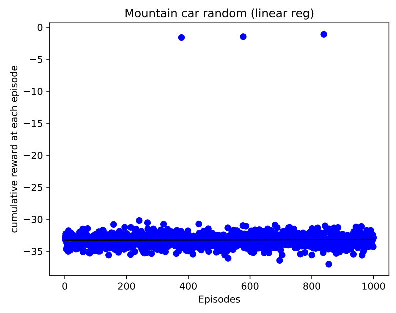
(e) Cumulative rewards for run 7.



(f) Cumulative rewards for run 8.

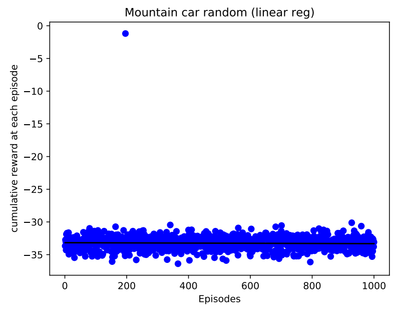


(g) Cumulative rewards for run 9.

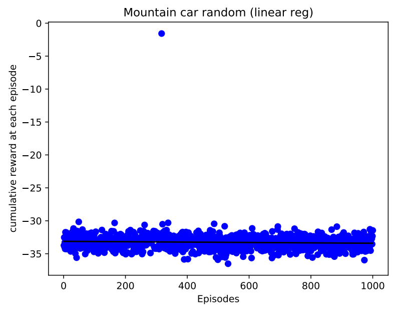


(h) Cumulative rewards for run 10.

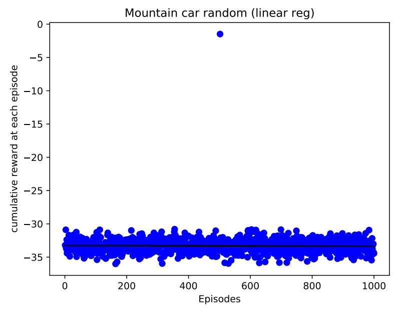
Figure A.7: Cumulative rewards (blue circles) for runs 3-10 for 2000 time steps using random action controller in mountain car continuous environment. The black line indicates the linear regression line.



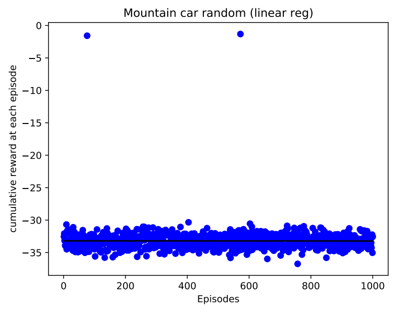
(a) Cumulative rewards for run 3.



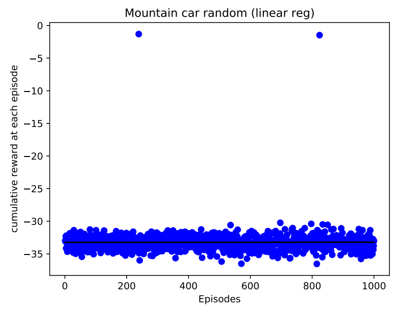
(b) Cumulative rewards for run 4.



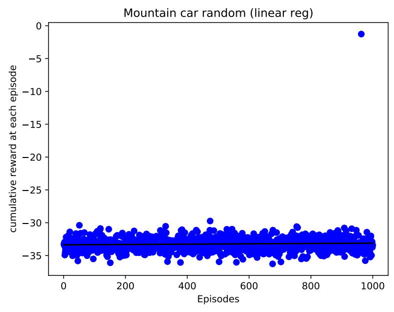
(c) Cumulative rewards for run 5.



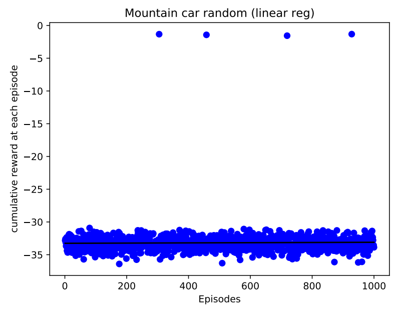
(d) Cumulative rewards for run 6.



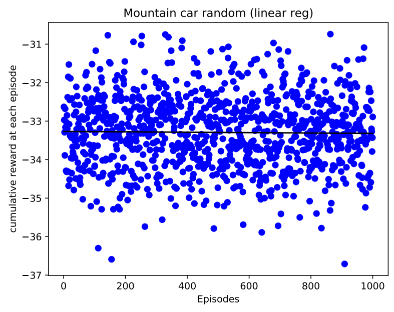
(e) Cumulative rewards for run 7.



(f) Cumulative rewards for run 8.

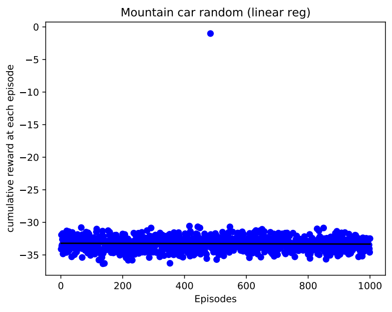


(g) Cumulative rewards for run 9.

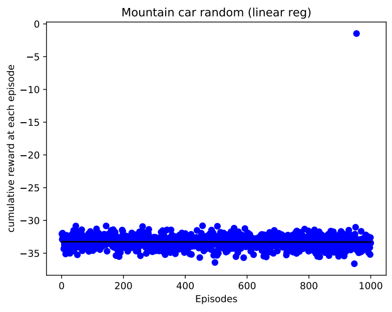


(h) Cumulative rewards for run 10.

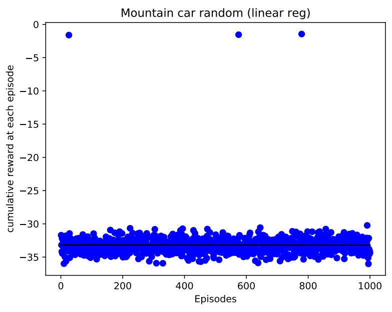
Figure A.8: Cumulative rewards (blue circles) for runs 3-10 for 3000 time steps using random action controller in mountain car continuous environment. The black line indicates the linear regression line.



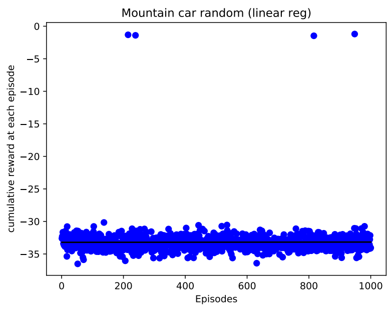
(a) Cumulative rewards for run 3.



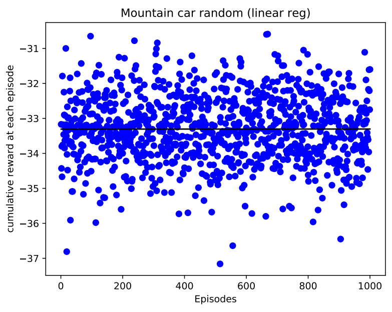
(b) Cumulative rewards for run 4.



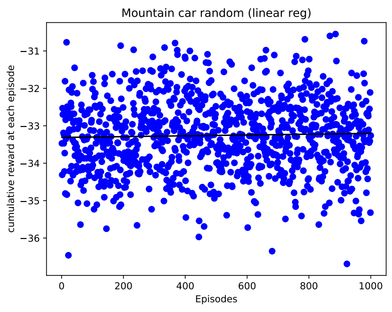
(c) Cumulative rewards for run 5.



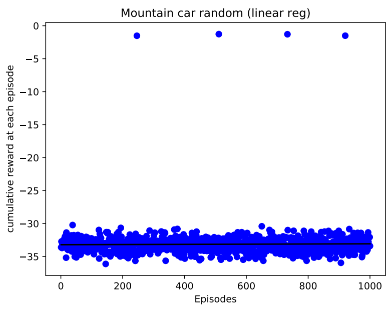
(d) Cumulative rewards for run 6.



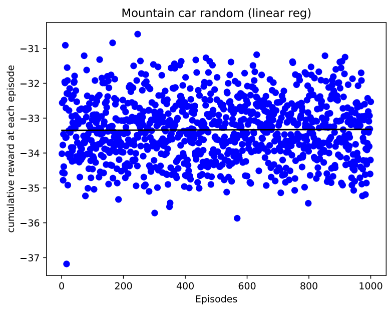
(e) Cumulative rewards for run 7.



(f) Cumulative rewards for run 8.

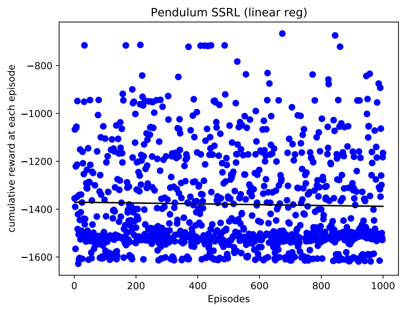


(g) Cumulative rewards for run 9.

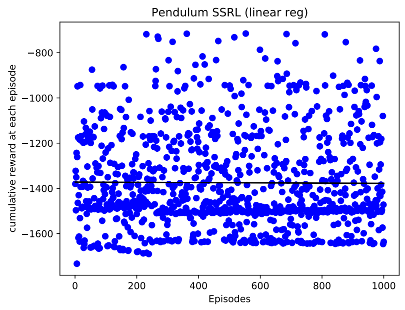


(h) Cumulative rewards for run 10.

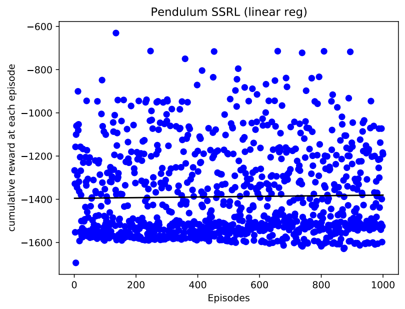
Figure A.9: Cumulative rewards (blue circles) for runs 3-10 for 4000 time steps using random action controller in mountain car continuous environment. The black line indicates the linear regression line.



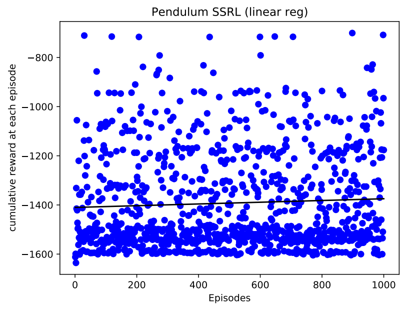
(a) Cumulative rewards for run 3.



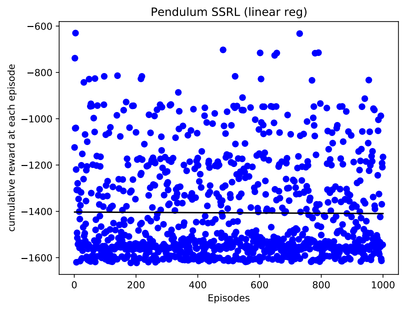
(b) Cumulative rewards for run 4.



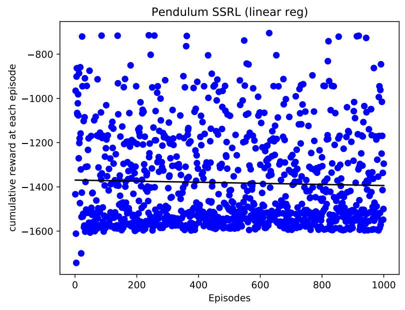
(c) Cumulative rewards for run 5.



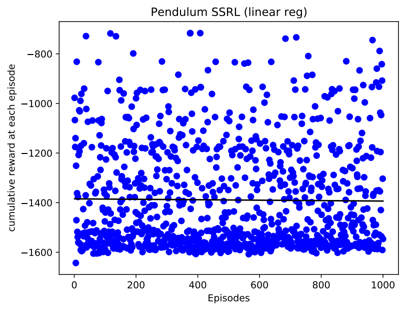
(d) Cumulative rewards for run 6.



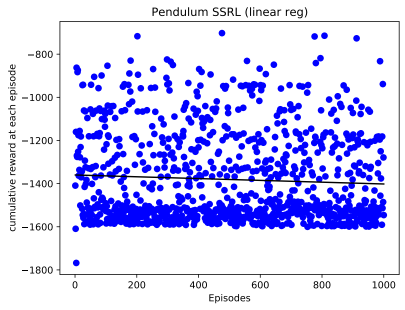
(e) Cumulative rewards for run 7.



(f) Cumulative rewards for run 8.

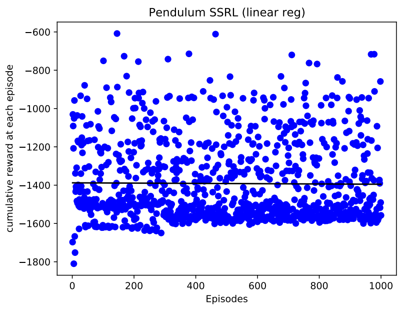


(g) Cumulative rewards for run 9.

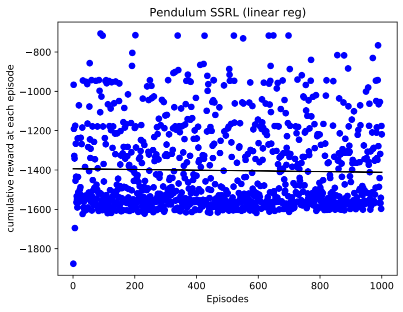


(h) Cumulative rewards for run 10.

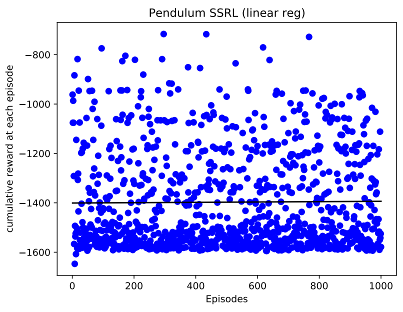
Figure A.10: Cumulative rewards (blue circles) for runs 3-10 for 2000 time steps using SSRL in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.



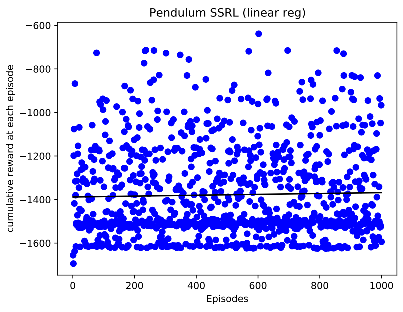
(a) Cumulative rewards for run 3.



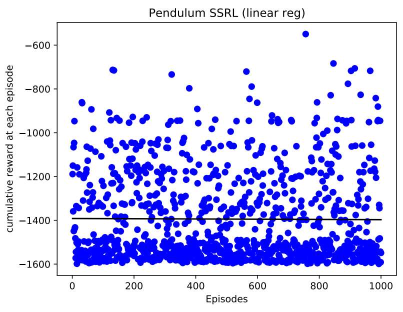
(b) Cumulative rewards for run 4.



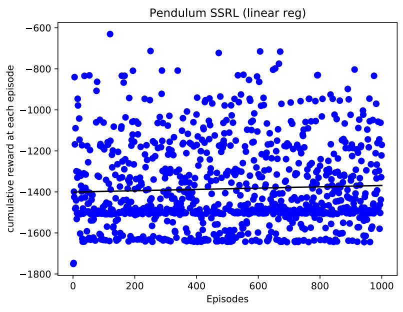
(c) Cumulative rewards for run 5.



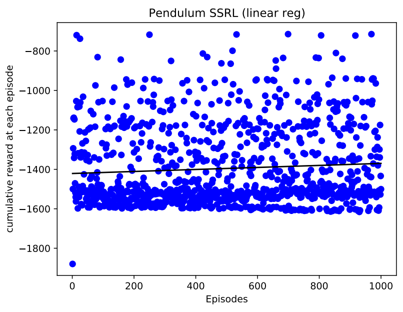
(d) Cumulative rewards for run 6.



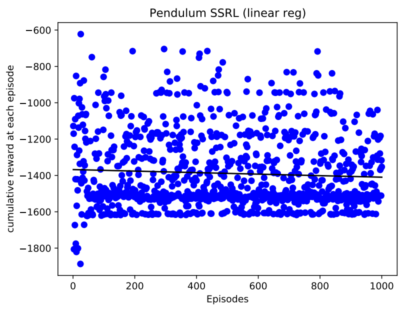
(e) Cumulative rewards for run 7.



(f) Cumulative rewards for run 8.

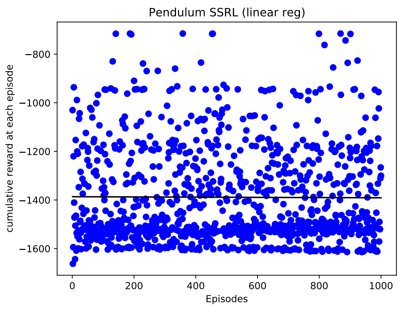


(g) Cumulative rewards for run 9.

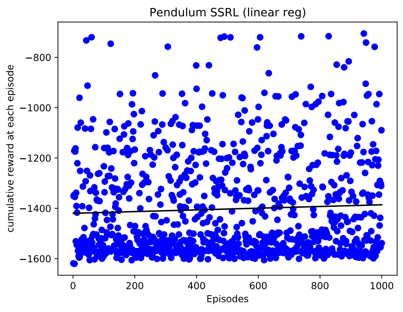


(h) Cumulative rewards for run 10.

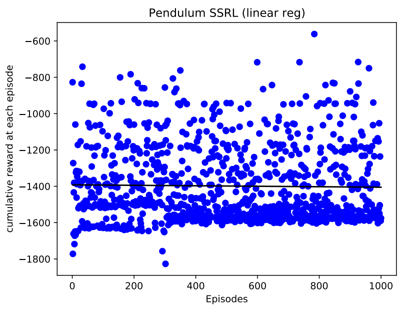
Figure A.11: Cumulative rewards (blue circles) for runs 3-10 for 3000 time steps using SSRL in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.



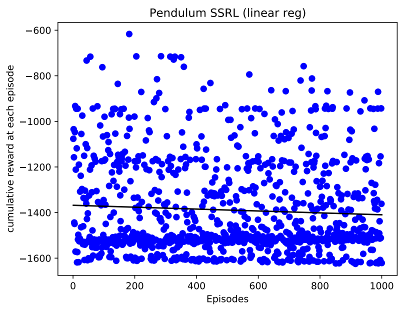
(a) Cumulative rewards for run 3.



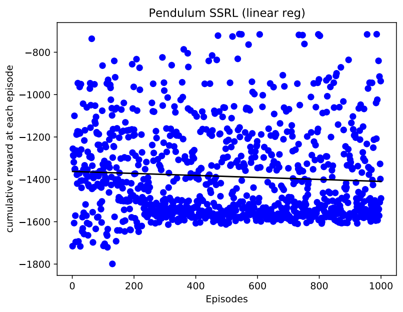
(b) Cumulative rewards for run 4.



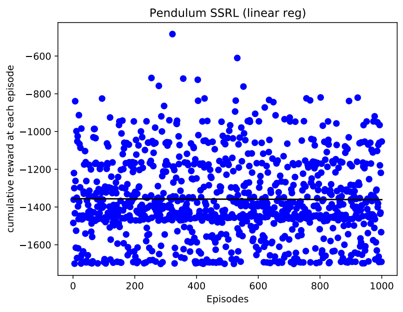
(c) Cumulative rewards for run 5.



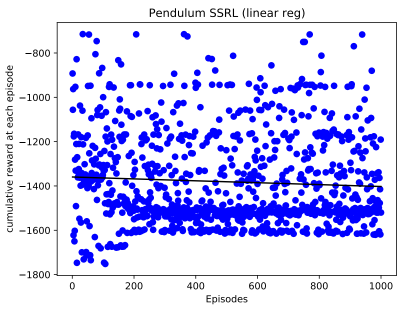
(d) Cumulative rewards for run 6.



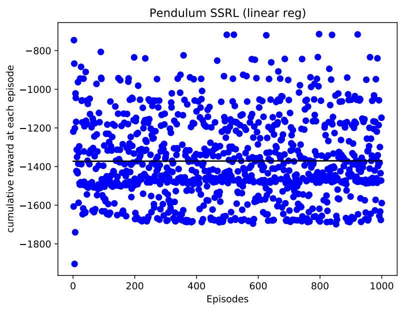
(e) Cumulative rewards for run 7.



(f) Cumulative rewards for run 8.

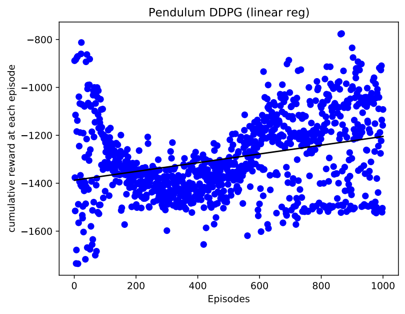


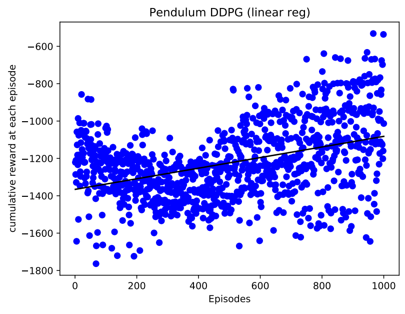
(g) Cumulative rewards for run 9.



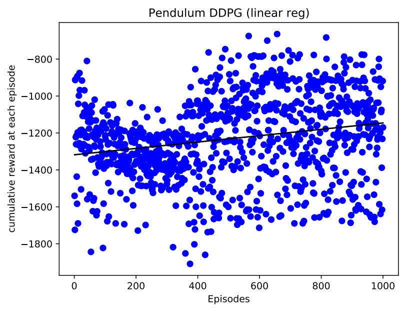
(h) Cumulative rewards for run 10.

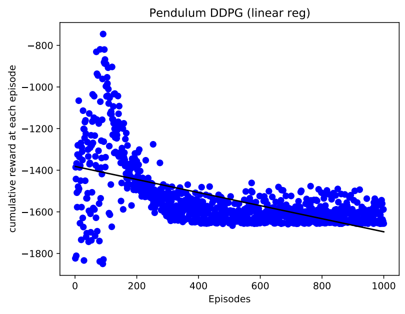
Figure A.12: Cumulative rewards (blue circles) for runs 3-10 for 4000 time steps using SSRL in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.



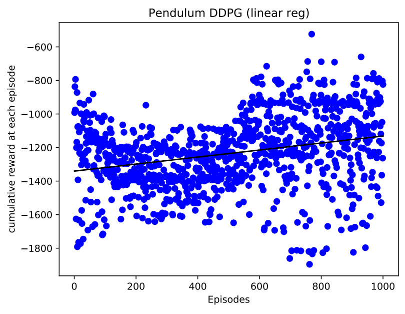


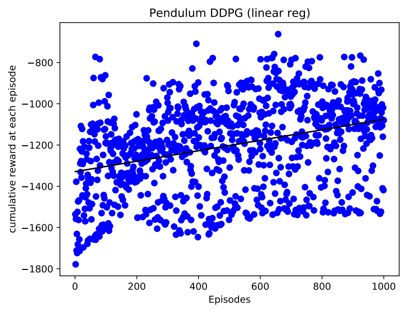
(a) Cumulative rewards for run 3. (b) Cumulative rewards for run 4.



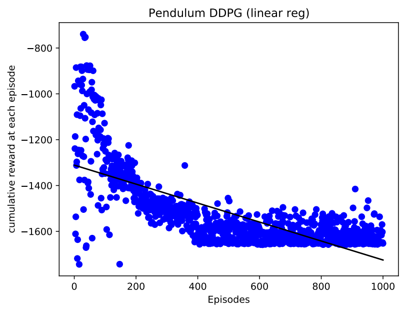


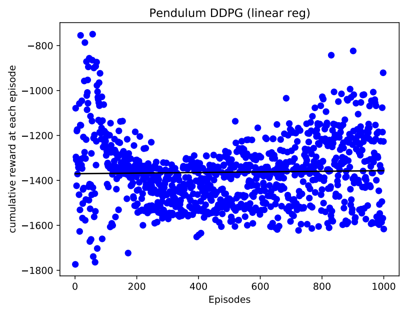
(c) Cumulative rewards for run 5. (d) Cumulative rewards for run 6.





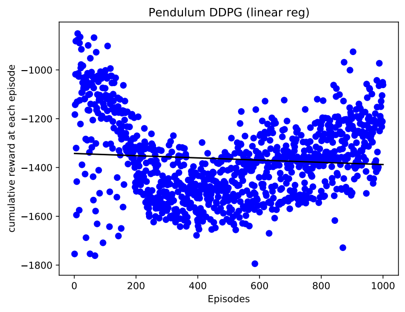
(e) Cumulative rewards for run 7. (f) Cumulative rewards for run 8.



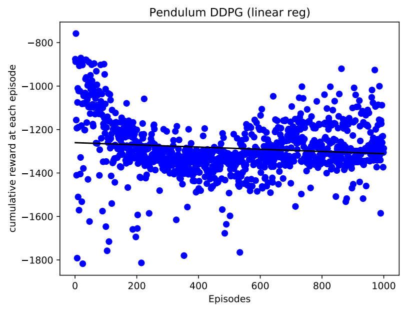


(g) Cumulative rewards for run 9. (h) Cumulative rewards for run 10.

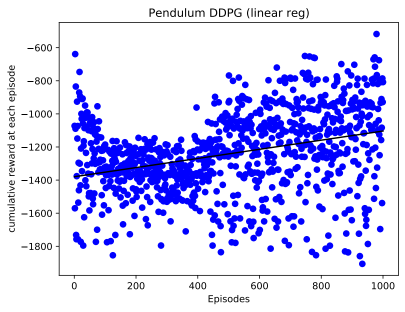
Figure A.13: Cumulative rewards (blue circles) for runs 3- 10 for 2000 time steps using DDPG in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.



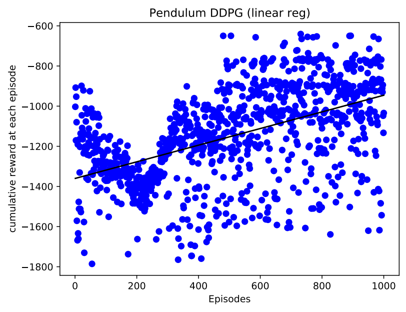
(a) Cumulative rewards for run 3.



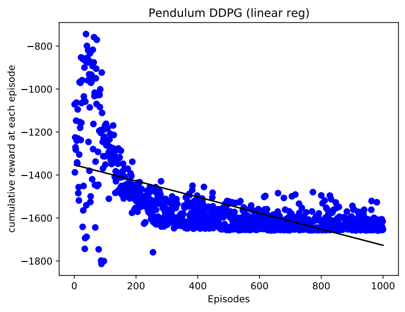
(b) Cumulative rewards for run 4.



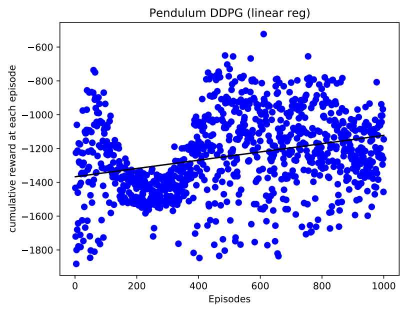
(c) Cumulative rewards for run 5.



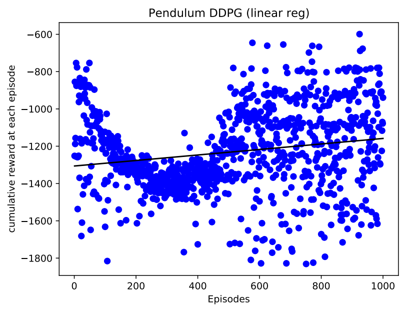
(d) Cumulative rewards for run 6.



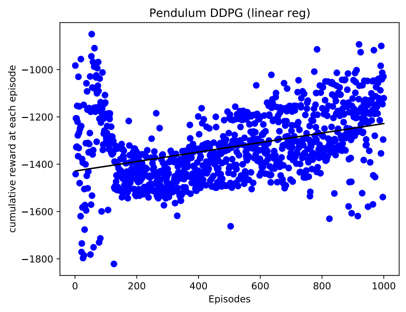
(e) Cumulative rewards for run 7.



(f) Cumulative rewards for run 8.

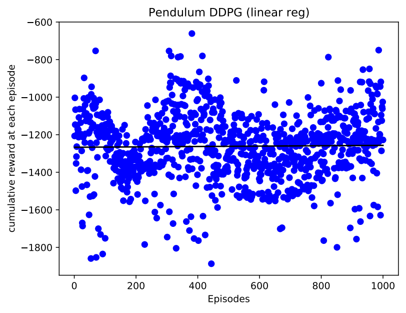


(g) Cumulative rewards for run 9.

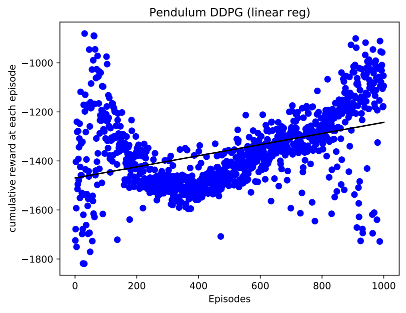


(h) Cumulative rewards for run 10.

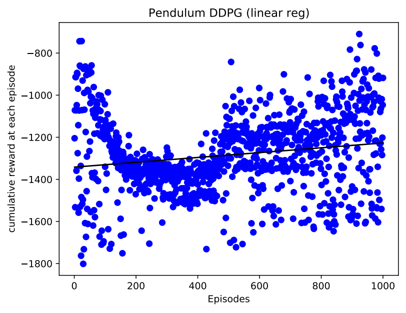
Figure A.14: Cumulative rewards (blue circles) for runs 3-10 for 3000 time steps using DDPG in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.



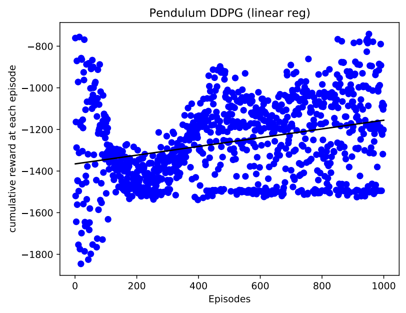
(a) Cumulative rewards for run 3.



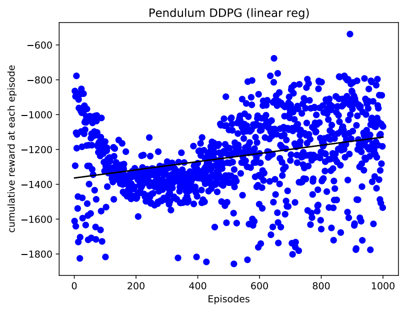
(b) Cumulative rewards for run 4.



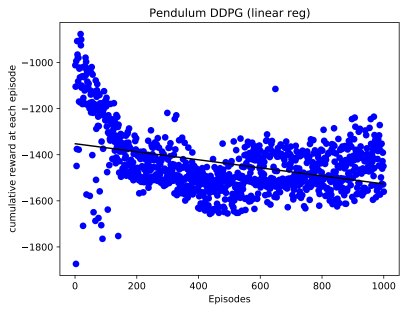
(c) Cumulative rewards for run 5.



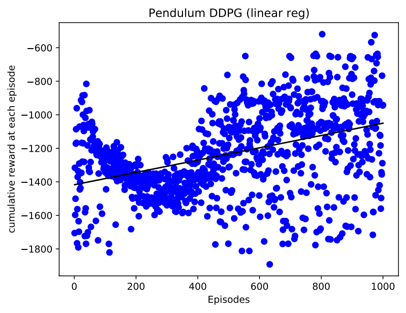
(d) Cumulative rewards for run 6.



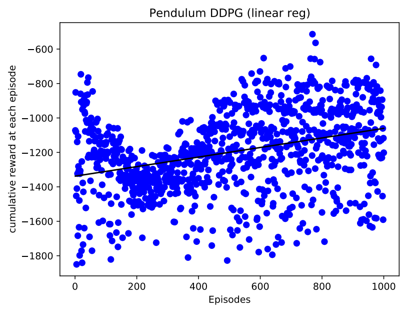
(e) Cumulative rewards for run 7.



(f) Cumulative rewards for run 8.

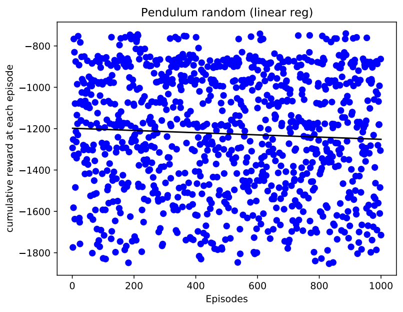


(g) Cumulative rewards for run 9.

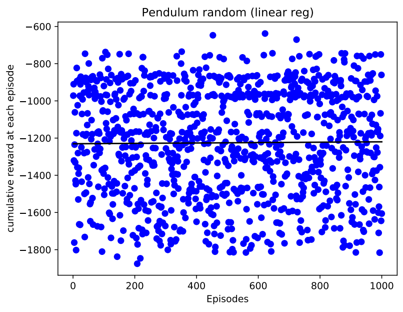


(h) Cumulative rewards for run 10.

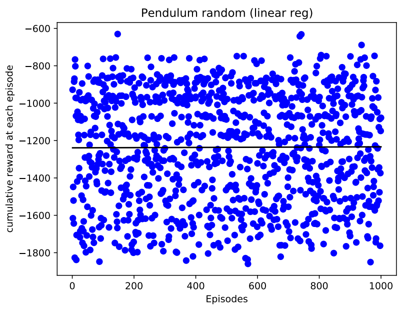
Figure A.15: Cumulative rewards (blue circles) for runs 3-10 for 4000 time steps using DDPG in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.



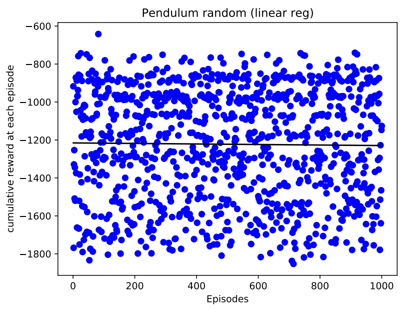
(a) Cumulative rewards for run 3.



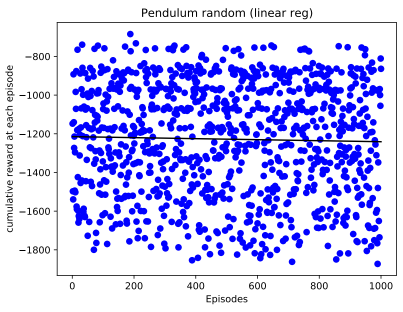
(b) Cumulative rewards for run 4.



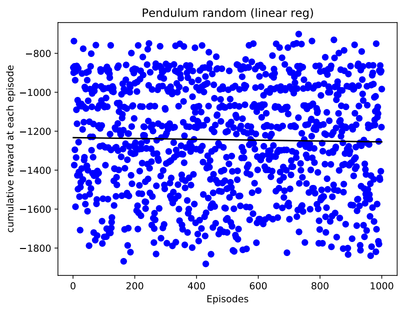
(c) Cumulative rewards for run 5.



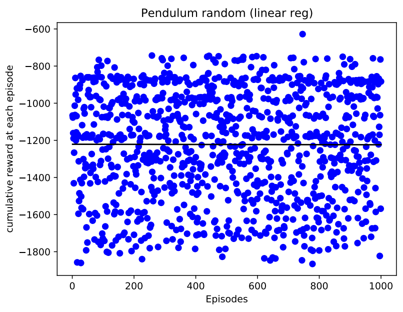
(d) Cumulative rewards for run 6.



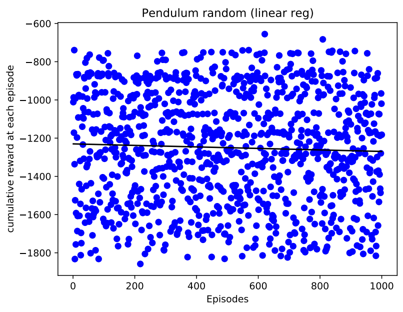
(e) Cumulative rewards for run 7.



(f) Cumulative rewards for run 8.

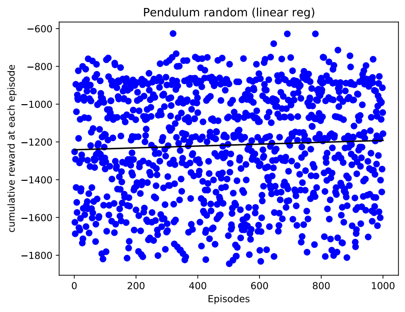


(g) Cumulative rewards for run 9.

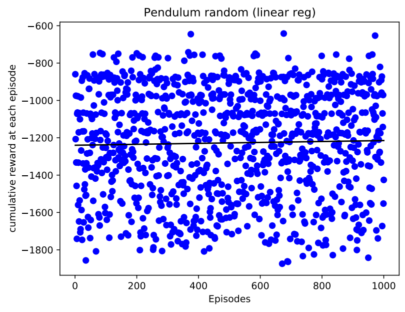


(h) Cumulative rewards for run 10.

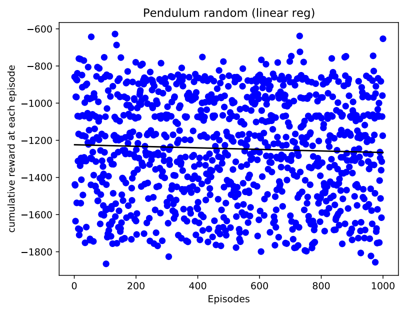
Figure A.16: Cumulative rewards (blue circles) for runs 3-10 for 2000 time steps using random action controller in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.



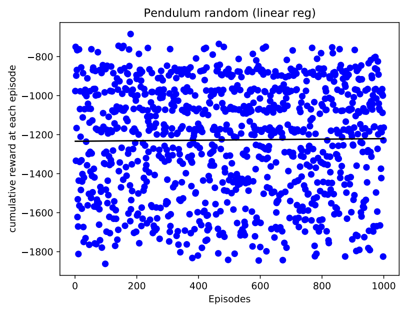
(a) Cumulative rewards for run 3.



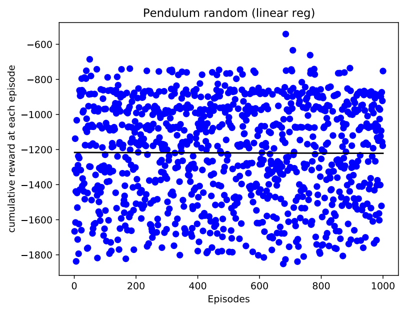
(b) Cumulative rewards for run 4.



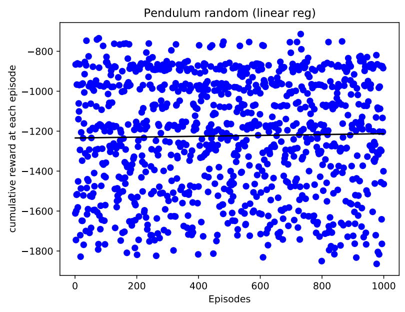
(c) Cumulative rewards for run 5.



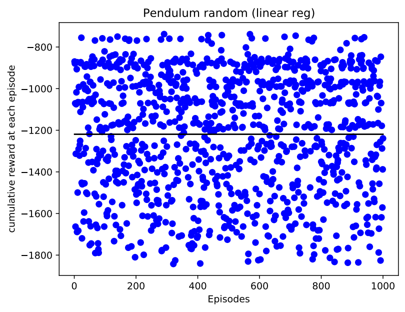
(d) Cumulative rewards for run 6.



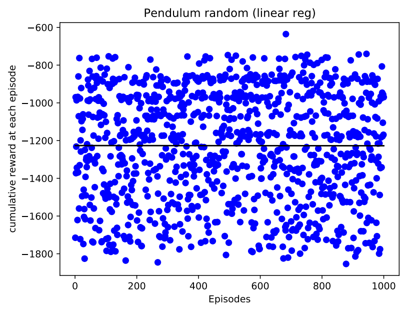
(e) Cumulative rewards for run 7.



(f) Cumulative rewards for run 8.

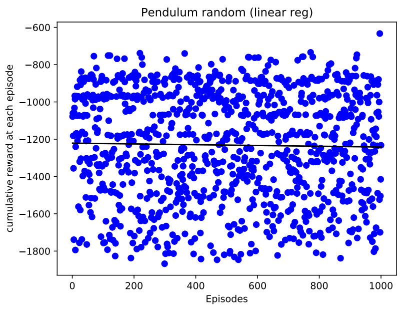


(g) Cumulative rewards for run 9.

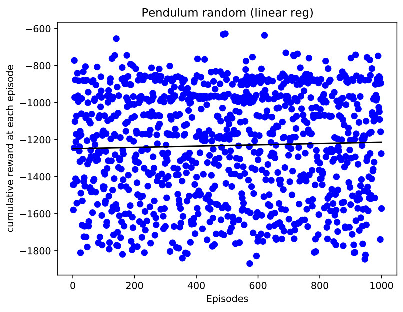


(h) Cumulative rewards for run 10.

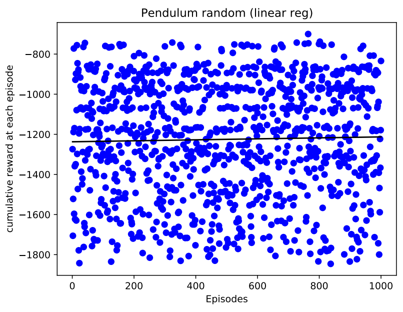
Figure A.17: Cumulative rewards (blue circles) for runs 3-10 for 3000 time steps using random action controller in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.



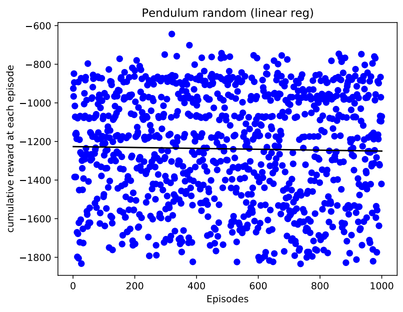
(a) Cumulative rewards for run 3.



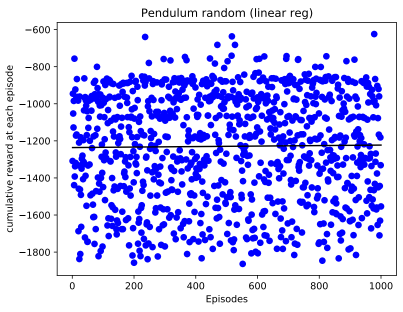
(b) Cumulative rewards for run 4.



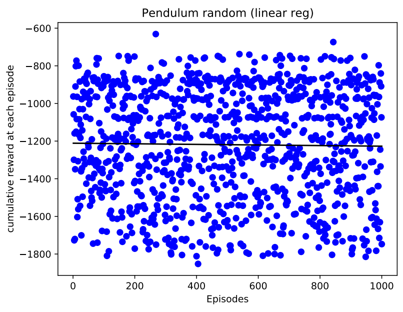
(c) Cumulative rewards for run 5.



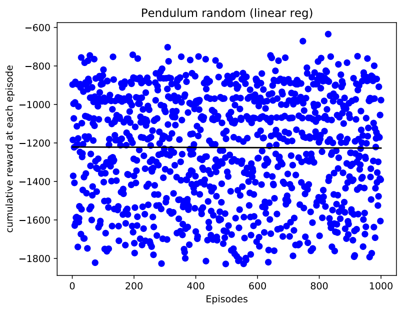
(d) Cumulative rewards for run 6.



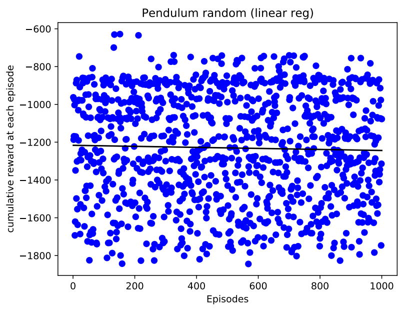
(e) Cumulative rewards for run 7.



(f) Cumulative rewards for run 8.



(g) Cumulative rewards for run 9.



(h) Cumulative rewards for run 10.

Figure A.18: Cumulative rewards (blue circles) for runs 3-10 for 4000 time steps using random action controller in delayed-reinforcement pendulum environment. The black line indicates the linear regression line.