CS6515 Project Report

Analysis of Stochastic Optimization Algorithms on RL

Chella Thiyagrajan (ME17B179), Vihaan Akshaay (ME17B171), Naveen Prashanna (ME17B156)

May 5, 2022

Abstract

Deep Q-Network(DQN) is one of the early deep reinforcement learning methods proposed by DeepMind which combines reinforcement learning with a class of artificial neural networks to approximate a state-value function in a Q-learning framework. It used a Convoluted Neural Network architecture as a function approximator. Progressive and commonly used optimization algorithms are benchmark to analyse the challenges and advantages to learn control agents to maximize returns. OpenAI gym environments are used to create the control agents and the optimization algorithms were implemented for Back-Propagation in DQN model. This project is to study the performance of different optimization algorithms on these control agents.

Contents

1 Introduction

Human beings have the ability to learn several tasks through out their lives without having to be given explicit instructions. Thanks to the innate reward system in the brain, our body gets a feedback for every action that we take and helps us learn about our environment consciously or unconsciously. The body is designed in such a way as to cause discomfort and hunger when food is needed and makes one feel better after eating. This is a natural system inherently present in the body which helps one learn to eat.

1.1 Reinforcement Learning

Formalizing the setting that Reinforcement Learning Algorithms work with, The decision maker is called the **Agent**. The agent continuously interacts with its surroundings called **Environment**. These interactions consist of an **Action** that the agent takes that alters the environment's **State**. The agent also receives a scalar **reward** for every step that it takes. The goal of the agent is to maximize the total reward it gets. To model the real world better, it would make sense for immediate rewards to matter more for the agent than future reward. The **Discount factor** ($0 \leq \gamma \leq 1$) is a constant that is multiplied to future rewards to make it less impactful. We define this new discounted cumulative reward as the **Return** and the goal of the agent is to maximize the Return from every state. A **Policy** can be thought of as a rule book that helps an agent to decide the action to take from a given state. We can consider that an Environment is solved if we find the optimal policy that helps the agent choose actions such that it receives the maximum return corresponding to each possible state in the environment.

This straightforward framing of the problem of learning from interaction to achieve a goal can be represented by a **Markov Decision Process** (MDPs). The interactions can be formalized as follows. The agent and environment interact at discrete time steps, $t = 0, 1, 2, 3, ...$ .At each time step t, the agent receives some representation of the environment's state, $S_t \in \mathcal{S}$ and with the help of a policy Π_t selects an action $A_t \in A(s)$. The Reward that the agent obtains at a particular step is denoted by R_t and the total discounted cumulative reward (Return) that the agent obtains from a particular state is given by G. The goal of the agent is to obtain the optimal policy π^* which helps

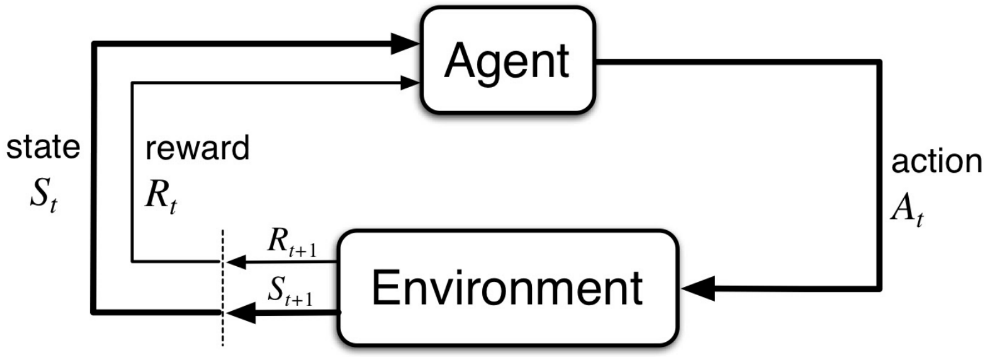


Figure 1: The Agent-Environment Interaction

the agent choose actions in each state that returns the maximum return(G) possible.

1.2 Q-Learning

We start this section by introducing Q-Values. These are scalars we assign to each state-action pair. These are supposed to be the equivalent of the expected Return for the agent taking a certain action from a particular state. The advantage of this formulation is that, once the Q-Values converge to the values that we desire (optimal returns), just following the action that has the maximum Q-Value in each state, gives us the optimal policy to solve the environment.

Q-Learning is an off-policy reinforcement learning algorithm that is used to find the best action for each state. For performing the Q-learning algorithm, we start with a **Q-Table** which is essentially a matrix of shape, states x actions, which has the corresponding Q-value in each of its entries. This becomes the reference table for the agent to choose actions.

The next step is for the agent to interact with the environment and update these Q-Values. The agent is designed as to first choose to **explore** more (choose random actions in each state) and update the Q-values. As the agent keeps gaining more experience, it is slowly made to **exploit** its knowledge and choose the action it thinks best.

With each of these interactions, we utilize the transition data (current state - S_t, current action - A_t, next state - S_{T+1}, and the reward R_t) to iteratively update the Q-Value (corresponding to state S_t and action A_t). The stochastic iterative update rule solving the Bellman equation is given below:

$$Q^{new}(S_t, A_t) = Q(S_t, A_t) + \alpha(R_t + \gamma. \max_a Q(S_{t+1}, a) - Q(S_t, A_t)).$$

Where α is the control parameter/ learning rate, and γ is the discount factor.

Once we have done enough exploration of the environment, the Q-Table is expected to converge at optimal Q-Values. Once it has converged, the optimal policy to solve the environment would be to choose the action that has the maximum return. This can be achieved by choosing the action that has the maximum Q-value in a given state.

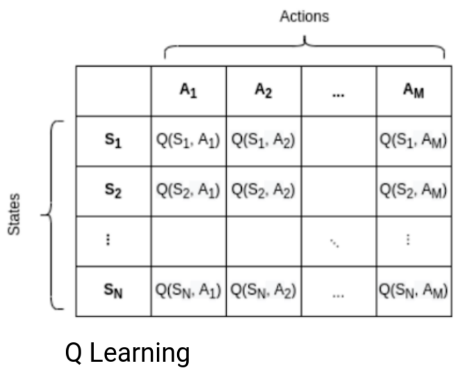
2 DQN

Q-learning is a very intuitive algorithm with a simple update rule and it is easy to obtain the policy corresponding to a Q-Table. Although Q-Learning works best with grid worlds and similar environments with limited states and actions, most real-life tasks do not follow the same trend. Even a simple task like balancing a pole on a cart (We'll specifically explore this environment later in the experiments) has infinite states. This causes the size of the Q-Table to be massive, which would lead to issues with memory as well as time taken to estimate each of the entries in the table with sufficient transitions. This 'curse of dimensionality' is one of Q-Learning's disadvantages.

Q-Learning also has issues with generality. Since it only saves values for certain states, if the agent encounters a new state, it as no clue which action to take.

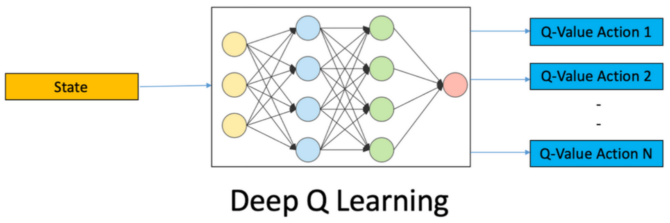
This lead to the use of function approximators. In DQNs, instead of using a Q-Table, a deep neural network is used as function approximator. The input to the neural network is the current state, and the output is the corresponding Q-Values for each function.

In DQNs, let the parameters of the neural network be denoted by θ, (ie $Q(S_t, A_t) = Q(S_t, A_t; \theta_t)$). We want the predictions of the neural network to match the Bellman Equation. Therefore, the loss function can be defined as the mean squared error between the targets ($R_t + \max_A Q(S_{t+1}, A)$) and



	A_1	A_2	...	A_M
S_1	$Q(S_1, A_1)$	$Q(S_1, A_2)$		$Q(S_1, A_M)$
S_2	$Q(S_2, A_1)$	$Q(S_2, A_2)$		$Q(S_2, A_M)$
⋮			⋱	⋮
S_N	$Q(S_N, A_1)$	$Q(S_N, A_2)$	...	$Q(S_N, A_M)$

Actions
States
Q Learning



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

$$Loss = (r + \gamma max_{a'} Q(s', a'; \theta') - Q(s, a; \theta))^2$$

Figure 2: Q-Table and Q-Network

the prediction ($Q(S_{t+1}, A_t)$) using stochastic gradient descent. The update rule is as follows:

$$\theta_{t+1} \leftarrow \theta_t + \alpha[(R_t + \max_A Q(S_{t+1}, A; \theta_t) - Q(S_t, A_t; \theta_t))\nabla_{\theta_t} Q(S_t, A_t; \theta_t)].$$

3 Problem Statement

3.1 Analysis of Stochastic Optimization algorithms in Reinforcement Learning

Reinforcement Learning (RL) has achieved state-of-the-art results in domains such as robotics and games. We build on this previous work by applying stochastic optimization algorithms to a selection of RL problems with a range of practical applications.

We have chosen three Open-AI Gym environments to analyse the performance of the following stochastic optimization algorithms:

- Fixed Step Stochastic Gradient Algorithm for Empirical Risk Minimization
- Mini Batch Stochastic Gradient Algorithm
- Full Batch Stochastic Gradient Algorithm
- Stochastic Variance Reduced Gradient Algorithm for Empirical Risk Minimization
- Broyden–Fletcher–Goldfarb–Shanno Algorithm

Gym is an open source Python library for developing and comparing reinforcement learning algorithms by providing a standard API to communicate between learning algorithms and environments. Open-AI Gym environments are used to collect the state observations and rewards for the actions. The DQN uses the state observations as input and predicts the Q-values for the given action. The true Q-values are calculated using the Bell-Man equation which assures the convergence of the Q-values. We assume the loss function to be Mean-Squared error and our objective is to minimize this loss function. The above seven optimization algorithms are benchmark against each other to minimize this loss function.

The systematic retrospect and summary of the optimization methods from the perspective of reinforcement learning are of great significance, which can offer guidance for both developments of optimization and reinforcement learning research. We introduce the principles and progresses of commonly used optimization methods. Next, we will summarize the analysis and developments of optimization methods in some popular reinforcement learning fields. Finally, we will also explore and give some challenges and open problems for the optimization in reinforcement learning.

4 Enivironments

4.1 Environment - Acrobot

In the Acrobot environment, the agent tries to swing up a double pendulum by applying either clockwise or anti-clockwise torque at the second joint and keep the tip above a certain line. The episode runs for a fixed amount of 500 steps and the agent receives -1 for each step where the tip is below the target line.

- The state space is a 6-dimensional vector - $\cos(\theta_1)$ $\sin(\theta_1)$ $\cos(\theta_2)$ $\sin(\theta_2)$ $\dot{\theta_1}$ $\dot{\theta_2}$.
- Three possible actions - clockwise/anticlockwise/no torque at the joint connecting the two links.
- The agent receives -1 reward for every step with the tip below the target and receives 0 for

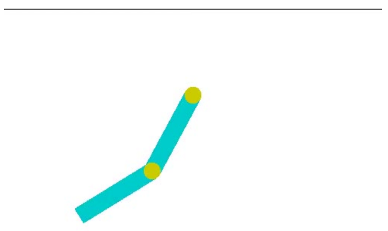


Figure 3: Environment - Acrobot

every step the tip is above the target.

4.2 MountainCar - Environment

In the Mountain Car environment, the agent tries to drive up a car located in between two hills, over the hill on the right. The catch is that the car does not have enough power to go up the hill in a single shot. The car has to go up and down the hills to gain momentum and then pass the hill on the right.

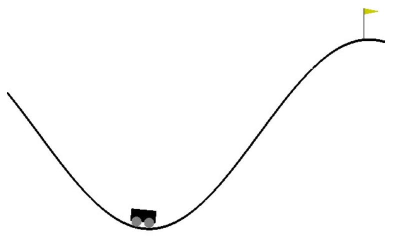


Figure 4: Environment - MountainCar

- The state space is a 2-dimensional vector - Position and Velocity of the car.
- Three possible actions - push left/no push/ push right.
- The agent receives -1 reward for every step until it reaches the top of the right hill.

4.3 CartPole - Environment

In the Cart Pole environment, the agent tries to balance a pole on a cart by applying unit force either on right or left direction. The episode terminates when the pole falls for more than 12 degrees from the vertical or if the cart goes out of the frame.

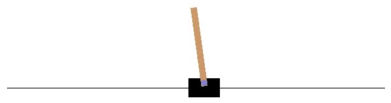


Figure 5: Environment - Cartpole

- The state space is a 4-dimensional vector -Cart position, Cart velocity, Pole angle, Pole angular velocity.
- There are two possible actions - push left or push right
- The agent receives +1 reward for every step until the episode terminates

5 Algorithms

5.1 Fixed Step Stochastic Gradient Descent

In stochastic gradient descent, the true gradient of $F(\omega)$ is approximated by a gradient at a single sample. Since we are considering just one example at a time the cost will fluctuate over the training examples and it will not necessarily decrease. But in the long run, you will see the cost decreasing with fluctuations.

$$\omega := \omega - \eta * \nabla F_i$$

5.2 Mini Batch Stochastic Gradient Descent

A compromise between computing the true gradient and the gradient at a single sample is to compute the gradient against ”m” samples (more than one training sample), called a "mini-batch", at each step. Just like SGD, the average cost over the epochs in mini-batch gradient descent fluctuates because we are averaging a small number of examples at a time.

$$\omega := \omega - \eta * \sum_{i=1}^{m} \nabla F_i$$

5.3 Full Batch Stochastic Gradient Descent

In Batch Gradient Descent, all the training data is taken into consideration to take a single step. We take the average of the gradients of all the training examples and then use that mean gradient to update our parameters. So that's just one step of gradient descent in one epoch.

$$\omega := \omega - \eta * \sum_{i=1}^{n} \nabla F_i$$

5.4 Stochastic Variance Reduced Gradient Algorithm

A Stochastic Variance Reduced Gradient (SVRG) Algorithm is a SGD-based Algorithm designed to reduce the SGD's Variance. Two loss terms are introduced to the gradient so that the variance of SGD is reduced.

$$\omega := \omega - \eta * [\nabla F(\omega, x) - \nabla F(\omega_s, x) + 1/n * \sum_{i=1}^{n} \nabla F(\omega_s)]$$

5.5 BFGS

BFGS is a second-order optimization algorithm. The BFGS algorithm is one specific way among all quasi-newton methods for updating the calculation of the inverse Hessian, instead of recalculating it every iteration.

$$\omega := \omega - \eta * H * \nabla F$$

6 Analysis

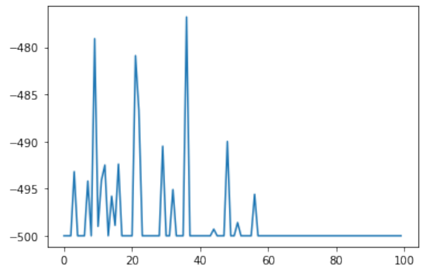
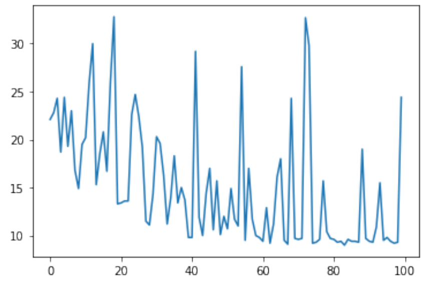
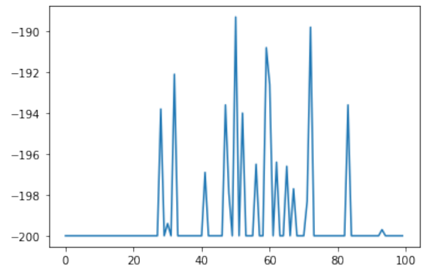
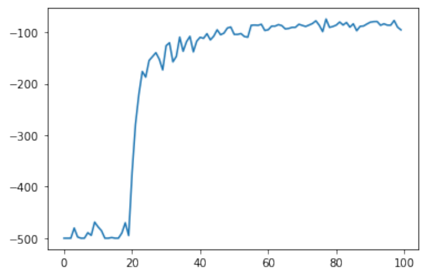
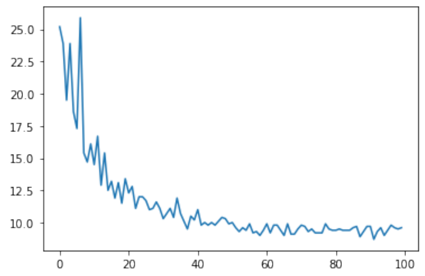
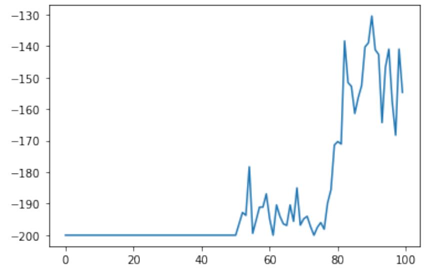
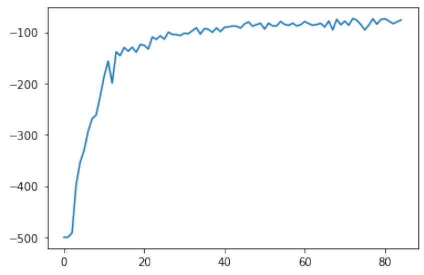
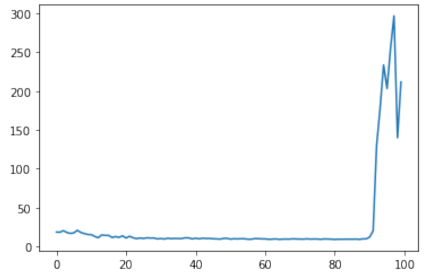
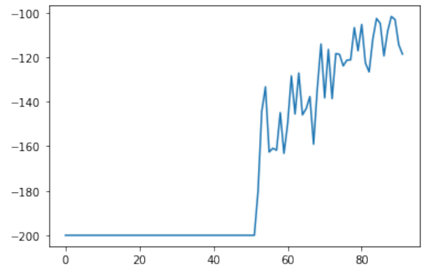
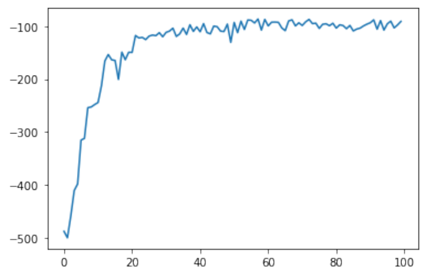
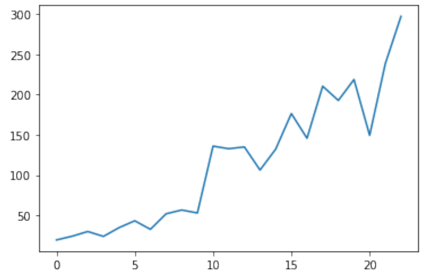
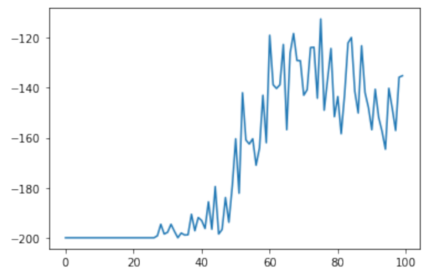
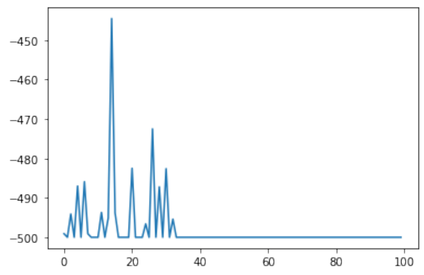
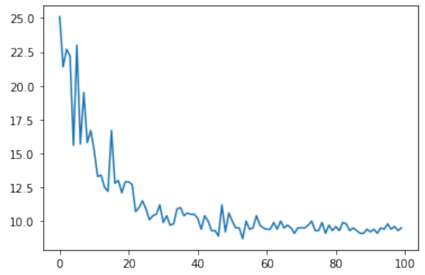
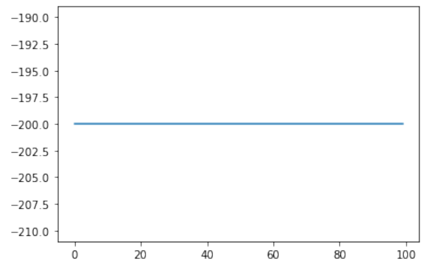
Algorithms	Acrobat	Cartpole	Mountain Cars
Fixed Step Stochastic Gradient			
Mini Batch Stochastic Gradient			
Full Batch Stochastic Gradient			
SVRG			
BFGS			

Table 1: Reward plot for all algorithms across each environment (scale: y-axis = reward per episode x-axis*10 = number of episode)

Episodes to solve (Acrobot): 1000+ 1000+ 754 1000+ 1000+

Episodes to solve (CartPole): 1000+ 1000+ 1000+ 135 1000+

Episodes to solve (MountainCar): 1000+ 1000+ 825 1000+ 1000+

7 Observation

7.1 Acrobot

7.1.1 Fixed Step Stochastic Gradient

It appears to not be learning how to solve the environment in 1000 episodes. The learning curve is absent in the reward plot. It seems to be spiking frequently which can be a consequence of updating the policy network at each time step. This affects the policy network of the consequent time step to temporarily achieve a better reward but fails very soon.

7.1.2 Mini Batch Stochastic Gradient

It took more than 1000 episodes to reach the goal episodic reward.The plot seems to be performing way better than the Fixed Step SG variant for this environment. There is a clear learning curve and saturation curve present in the reward plot.Increasing the experiment to a few more episodes would most probably result in the agent reaching the goal reward. There is a delay before which the agent starts to learn.

7.1.3 Full Batch Stochastic Gradient

It took 754 episodes for the agent to reach the goal reward.We can observe a gradual learning curve over 200 episodes and saturation curve over the next 600 episodes. Once it has figured out the right strategy and the score has increased significantly, it only seems to be constantly getting better. This is the only variant that has learnt to solve the environment within the given time frame.

7.1.4 SVRG

It took more than 1000 episodes to reach the goal episodic reward.This shows to perform really good. It starts to learn as soon as it collects samples. Although this is the case, the improvement seems to be a bit gradual in comparison with Full Batch Stochastic Gradient. That's why it takes more than 1000 episodes to reach the goal, while Full Batch reached it sooner. Increasing the experiment to a few more episodes would most probably result in the agent reaching the goal reward.

7.1.5 BFGS

It appears to not be learning how to solve the environment in 1000 episodes. Due to exploration dominating in the initial runs of the experiment,it imparts high variance to the policy network and there seems to be spikes in rewards. But sooner than Fixed Step Stochastic Gradient, it seems to perform terribly and unlearn the good strategies/ learn bad strategies. There seems to no improvement after a point.

7.2 Cartpole

7.2.1 Fixed Step Stochastic Gradient

It appears to not be learning how to solve the environment in 1000 episodes. It appears slightly similar to the performance of Fixed Step Stochastic Gradient for the Acrobot environment. Initially there is a good rise in the reward, which is probably due to the random actions taken. Over time, maybe because of bad use of samples generated, the agent seems to be learning very slowly, while forgetting to explore more for the right strategy. This lack of effectiveness and high frequency in updating parameters (due to the Fixed Step Stochastic Gradient), it is not able to catch up with the reduction of exploration and so seems to be going down in performance with time.

7.2.2 Mini Batch Stochastic Gradient

It appears to not be learning how to solve the environment in 1000 episodes. We observe a really weird reward plot. For unknown reasons, the agent seems to be getting worse with time. One possible reason could be that, unfortuantely, this agent kept receiving bad samples and the mini-batch size only resulted repetition of this behavior for this particular environment.

7.2.3 Full Batch Stochastic Gradient

It took more than 1000 episodes to reach the goal episodic reward. It takes a really long time and large number of explored samples in order to figure out the right strategy to maximise the reward. This happened after almost 800 episodes but the agent seems to be doing well after that. This could be because of the improvement in the sample quality after the rise. After this, there is a sharp rise and a constant improvement in the policy. Increasing the experiment to a few more episodes would most probably result in the agent reaching the goal reward.

7.2.4 SVRG

It only takes about 200 episodes to solve the environment. Despite the RL setup and the environment being potent in providing biased and random samples, the unbiased and averaged out updates of SVRG clearly helps the agent learn steadily. This could explain why SVRG was able to learn the Cartpole environment (notoriously known to be biased) easily in comparison to the other networks. We can also see that SVRG is learning steadily by breaking down to right actions instead of one-shot attempts at guessing the right strategy.

7.2.5 BFGS

It appears to not be learning how to solve the environment in 1000 episodes. It seems to perform similar to the mini-batch SG in Cartpole. We again observe a really weird reward plot. We can see that the agent with time doesn't seem to be learning any policy. The decrease in the exploration and randomness in action selection is seen in the reduction of rewards over episodes.

7.3 MountainCar

7.3.1 Fixed Step Stochastic Gradient

It appears to not be learning how to solve the environment in 1000 episodes. It performs very similar to the Fixed Step stochastic gradient case in Acrobot. Because of the difference in the environment sample generation and physics, the placement of the spikes comes out a little later.

7.3.2 Mini Batch Stochastic Gradient

It took more than 1000 episodes to reach the goal episodic reward. We make an interesting observation where the agent takes a while to start learning (which is common as sometimes agents require a certain amount of samples to stabilize its policy initially) but learns only a certain sub-optimal strategy. It takes a bit to realise it isn't helping, and goes on to find the best strategy and the rewards seem to shoot up soon. Increasing the experiment to a few more episodes would most probably result in the agent reaching the goal reward.

7.3.3 Full Batch Stochastic Gradient

It took about 850 episodes to reach the goal episodic reward. This is very similar to the Mini Batch SG variant from the MountainCar environment. Luckily, using more samples and a bigger mini batch size, it is able to reach the optimal strategy directly instead of sticking to sub-optimal strategies.

7.3.4 SVRG

It took more than 1000 episodes to reach the goal episodic reward. Despite learning steadily in the beginning, it seems to slightly dip over time. This could be because it still the total average gradient in addition to the current gradient. We observe that it goes to a sub-optimal policy close to 900-1000 episodes but considering its behaviour and gradient updation method, we hope with more episodes it will learn the right strategy and reach the goal reward.

7.3.5 BFGS

The agent doesn't seem to be learning anything. This pattern is observed through out the different environments.The flat line observed for this variant for MountainCar environment is similar to the initial flat lines observed for other algorithms.Since this environment has only terminal rewards, this variant seem to have never reached the terminal state even once and flat lines represent the delay of the algorithms to learn any strategy that would lead the agent to the terminal state even once before the episode ends.

8 Conclusion

8.1 Fixed Step Stochastic Gradient

The updation of the policy network at each time step contributes to the high variance of the network. As a result we can observe random spikes throughout all the environments when we use Fixed Step SG for updating parameters.

8.2 Mini Batch Stochastic Gradient

There is a delay before which the agent starts to learn. This maybe because the agent requires a sufficient number of batches to be trained on in order to figure out the right strategy that helps the agent maximise the reward. This is clearly observed in Acrobot and MountainCar because they have goal states and the agent can only reach them following a few niche strategies. The sudden rise in the graphs show that the agent has somehow figured out the right strategy and only continues to improve it over time (that way the graph does not fall back down).

8.3 Full Batch Stochastic Gradient

This uses samples efficiently by improving performance in Acrobot environment, meanwhile takes a bit to figure the right strategy to improve in Cartpole and MountainCar. This maybe due to the lack of exploration, leading to not finding the right policy. The gradual increase in the learning curve implies that the bias and variance are balanced and we get a monotonously increasing plot. Despite taking time to figure the right policy, the agent only seems to be improving over time, thanks to the stability provided by Full Batch Stochastic Gradient update to the policy parameters.

8.4 SVRG

SVRG assures a very good learning curve across all the environments. Despite learning steadily over most environments, it seems to slightly dip in MountainCar. This could be because it still the total average gradient in addition to the current gradient. Since we hold a huge experience replay buffer, it contains experiences from good as well as bad policies. This lets it affect our current policy network by updating the weights with information from comparatively bad/old policy samples from earlier episodes as well. Despite the RL setup and the environment being potent in providing biased and random samples, the unbiased and averaged out updates of SVRG clearly helps the agent learn steadily. This could explain why SVRG was able to learn the Cartpole environment (notoriously known to be biased) easily in comparison to the other networks.

8.5 BFGS

The plots show high variance in the policy network when there is lot more exploration but flattens out with time as the sample generation strategy chooses exploitation of the network over exploration by choosing random actions. It was also observed that the time taken by BFGS algorithm was significantly and consistently higher than all other algorithms across all the environments. This could be because of the matrix inversion and the complex computation steps involved in each parameter update and gradient calculation. Despite being of great theoretical value, for cases like these where the estimate of the global function that we wish to maximize, being dynamic, makes the agent very unstable and overall learning process very shaky. This can be clearly seen from the reward plots. This variation doesn't seem to be performing well in any of the environments.