Raunaq Jain
50320026

Reinforcement Learning
Assignment 3

A reinforcement learning system can be depicted in the following way –

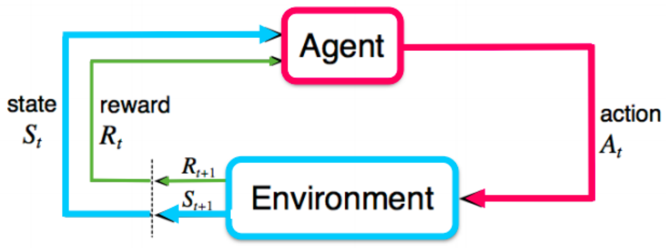


Figure 1

The system can be defined using a Markov Decision Process which is defined by tuple (S, A, P, R, γ), where

- S is the set of states, which characterizes the configuration of the environment.
- A denotes the actions the agent can take.
- R is the reward function.
- P is the state transition probability distribution.
- γ is the discount factor.

Q-learning algorithm was to go to method to solve reinforcement learning tasks but with high-dimensional and complex tasks, Q-learning algorithm breaks down and hence we use deep neural networks to solve these tasks.

Deep Q-Network was introduced in Playing Atari with Deep Reinforcement Learning on NIPS in 2013. It is a reinforcement learning algorithm that combines Q-Learning with deep neural networks to let RL work for complex, high-dimensional environments, like video games, or robotics.

DQN overcomes unstable learning by mainly 4 techniques –

1. **Experience Replay**
2. **Target Network**
3. **Clipping Rewards**
4. **Skipping Frames**

DQN is a value based approximation method, that is, we approximate the value or action-value function based on parameters θ;

$$V_\theta(s) \approx V^\pi(s)$$
$$Q_\theta(s,a) \approx Q^\pi(s,a)$$

And then based on some strategy, say ϵ-greedy strategy, we generate the policy from the value function.

Another method for solving reinforcement learning tasks is **policy based approximation method** or **policy gradient method**, wherein instead of approximating the value function, the policy is directly being model and optimized. It is usually modeled with a parametrized function with respect to θ, $\pi_\theta(a|s)$.

Policy gradient methods are generally believed to be applicable to a wider range of problems. For example, on occasions where the Q-learning function is too complex, DQN and in turn value based methods fails miserably. Whereas, Policy Gradients is still capable of learning a good policy since it directly operates in the policy space.

Policy Gradients model probabilities of actions, it is thus capable of learning stochastic policies, while DQN can't. Policy Gradient methods can also be easily applied to model continuous action space since the policy network is designed to model probability distribution. Lastly, Policy Gradient methods show faster convergence.

Even though there are so many advantages of Policy Gradient methods than value based approximation methods, we don't just use Policy Gradient methods all the time because it turns out that Policy Gradient methods suffer from a major drawback in estimating the gradient of the expected cumulative reward, $E[R_t]$ because of high variance. Moreover, Policy Gradient methods typically converge to the local minima instead of the global minima.

The Environment - CartPole

The Cartpole environment characteristics for the training are as follows:

- **Goal** – The goal of the environment is the maintain the cartpole in the vertical position for as long as possible.
- **State** – It's a 1×4 dimensional vector representing the current state.
- **Actions** – There are two possible actions to be taken – Left and Right
- **Reward** – The reward is +1 for every step that is not terminal

REINFORCE

The Vanilla Policy Gradient method is known as REINFORCE and its gradient update step is represented as

$$\nabla_\theta J(\theta) = E_\pi[Q_\pi(s,a)\nabla_\theta \ln \pi_\theta(a|s)]$$

which has no bias but high variance. The direct policy gradient method can be represented from the flowchart in the Figure.

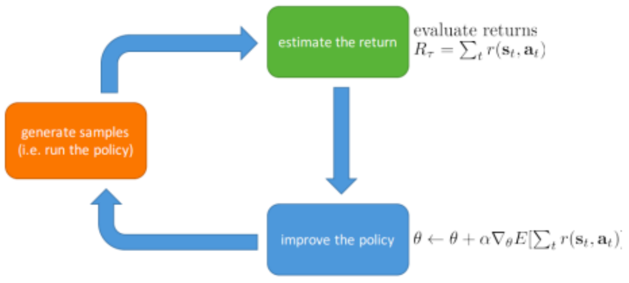
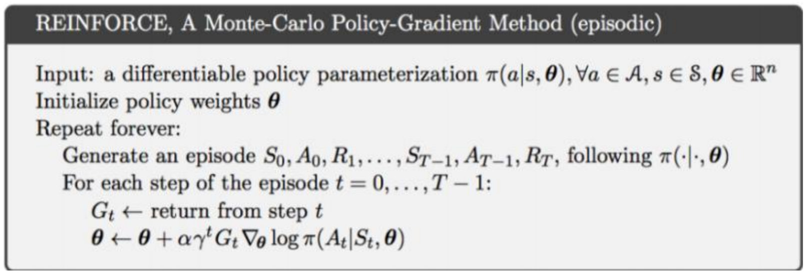

estimate the return
evaluate returns
$R_\tau = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$
generate samples (i.e. run the policy)
improve the policy
$\theta \leftarrow \theta + \alpha\nabla_\theta E[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)]$

Figure 2 Policy Gradient Methods Algorithmic Flowchart

The training algorithm is defined as in the following Figure.



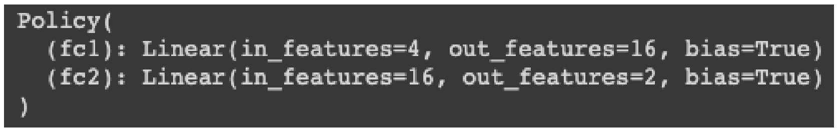
```
REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^n$
Initialize policy weights $\boldsymbol{\theta}$
Repeat forever:
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
    For each step of the episode $t = 0, \ldots, T-1$:
        $G_t \leftarrow$ return from step $t$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t, \boldsymbol{\theta})$
```

Figure 3 REINFORCE Algorithm

The training characteristics and parameters are:

- **Model Architecture**



```
Policy(
  (fc1): Linear(in_features=4, out_features=16, bias=True)
  (fc2): Linear(in_features=16, out_features=2, bias=True)
)
```

Figure 4 REINFORCE model architecture

- **Observations**

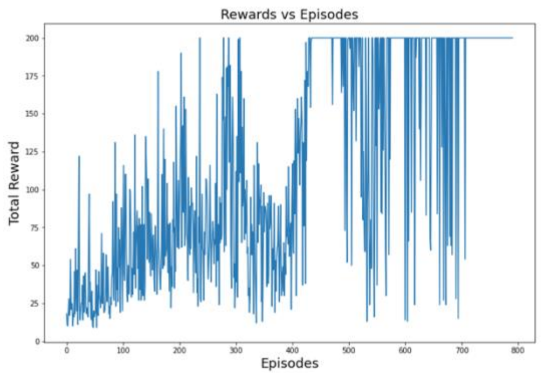


Figure 5 Plot of Rewards each episode [REINFORCE]

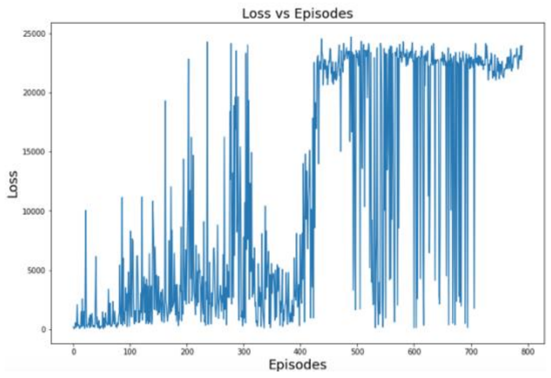
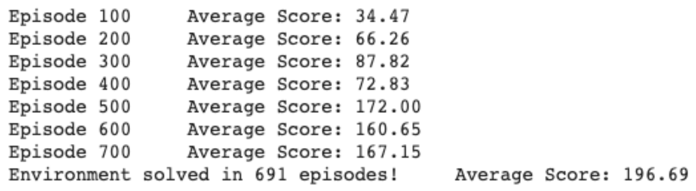


Figure 6 Plot of Loss each episode [REINFORCE]



```
Episode 100     Average Score: 34.47
Episode 200     Average Score: 66.26
Episode 300     Average Score: 87.82
Episode 400     Average Score: 72.83
Episode 500     Average Score: 172.00
Episode 600     Average Score: 160.65
Episode 700     Average Score: 167.15
Environment solved in 691 episodes!     Average Score: 196.69
```

Figure 7 Average Rewards per 100 Episode [REINFORCE]

Advantage Actor-Critic (A2C)

As mentioned above, Policy Gradients has a major drawback that its gradients have high variance and therefore to minimize it a number of approaches have been developed, for example, adding a baseline. One of the solutions is called the Actor-Critic method.

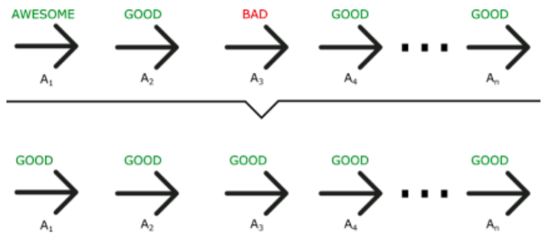


Figure 8 Monte-Carlo REINFORCE Policy Update Example

To understand, let’s visualize where the REINFORCE method suffers. In REINFORCE, even if an action was bad, if it leads to the terminal state then the update function considers it as good which leads to the problem of high variance; as it can be seen from Figure. Therefore, to solve this problem, we need to update the gradient update function.

The old policy function: $\Delta\theta = \nabla_\theta \ln \pi_\theta(a|s) * R(t)$

The new policy function: $\Delta\theta = \nabla_\theta \ln \pi_\theta(a|s) * Q(s,a)$

Actor-critic algorithm maintain two sets of parameters –

- Critic – Updates action-value function parameters w.
- Actor – Updates policy parameters θ, in direction suggested by critic.

Therefore, Actor-critic algorithms follow an approximate policy gradient –

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \ln \pi_\theta(s,a)\, Q_w(s,a)]$$
$$\Delta\theta = \alpha\, \nabla_\theta \ln \pi_\theta(s,a)\, Q_w(s,a)$$

- The **actor** is the policy $\pi_\theta(a|s)$ with parameters θ which conducts actions in an environment
- The **critic** computes value functions to help assist the actor in learning. These are usually the state value, state-action value, or advantage value, denoted as V(s), Q(s, a), and A(s, a), respectively.

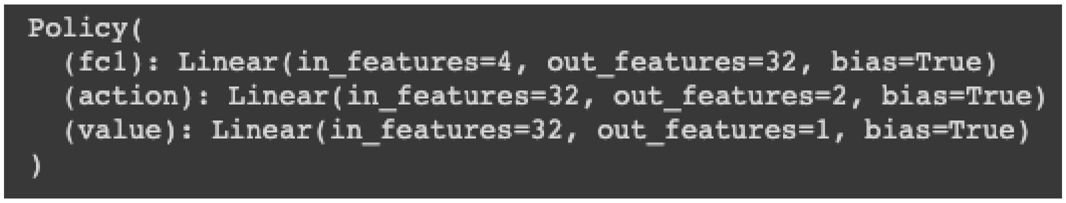
We need to calculate the error to train the actor-critic networks. The TD error,

$$\begin{aligned}\delta_{\pi_w} &= r + \gamma V_{\pi_w}(s') - V_{\pi_w}(s) \\ &= Q_{\pi_w}(s,a) - V_{\pi_w}(s) \\ &= A_{\pi_w}(s,a); \text{ known as the Advantage Function}\end{aligned}$$

Therefore, the policy gradient is $\nabla_w J(w) = E_{\pi_w}[\nabla_w \ln \pi_w(s,a)\delta_{\pi_w}]$

The training characteristics and parameters are:

- **Model Architecture**



```
Policy(
  (fc1): Linear(in_features=4, out_features=32, bias=True)
  (action): Linear(in_features=32, out_features=2, bias=True)
  (value): Linear(in_features=32, out_features=1, bias=True)
)
```

Figure 9 Advantage Actor-Critic model architecture

- **Observations**

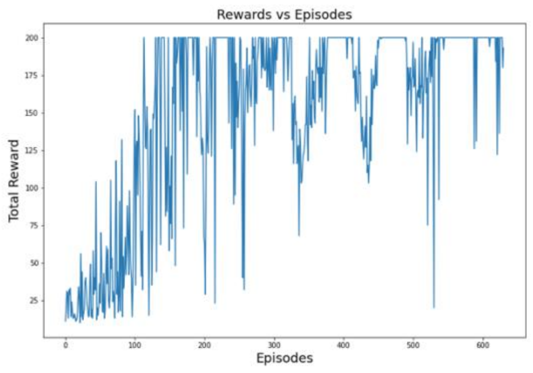


Figure 10 Plot of Rewards each episode [Advantage Actor-Critic]

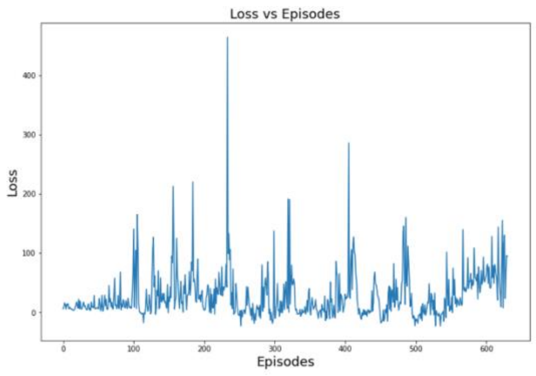
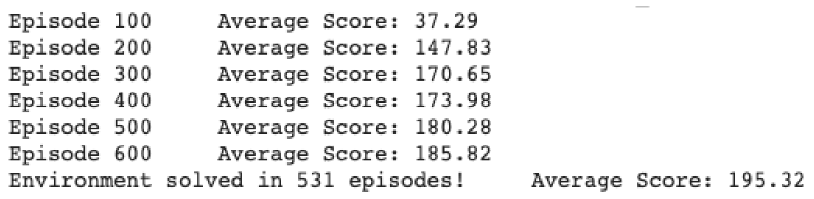


Figure 11 Plot of Rewards each episode [Advantage Actor-Critic]



```
Episode 100     Average Score: 37.29
Episode 200     Average Score: 147.83
Episode 300     Average Score: 170.65
Episode 400     Average Score: 173.98
Episode 500     Average Score: 180.28
Episode 600     Average Score: 185.82
Environment solved in 531 episodes!     Average Score: 195.32
```

Figure 12 Average Rewards per 100 Episode
[Advantage Actor-Critic]

Conclusion, Discussion and Remarks

First of all, it is easily visible from Figure 7 and Figure 12 that Advantage Actor-Critic algorithm converged faster taking only 531 episodes to score an average reward of 195.32 whereas REINFORCE took 691 episodes to score the same average reward. Moreover, from the previous assignment 2, it can be concluded that Policy Gradient methods converge faster than value-based function approximator methods.

From Figure 6 and Figure 11, we can observe that Advantage Actor-Critic algorithm is more stable than REINFORCE algorithm, hence proving that REINFORCE suffers from high variance and that having a critic stabilizes the learning.

In conclusion, there are various advantages to Policy Gradient methods and they are definitely one of the best methods to approach a reinforcement learning problem with. Having the understanding between the advantages and disadvantages between value-based approximation functions and policy-based approximation functions can help in better decision on which approach to take.