**Submitted by:**
Akshay Singh Rana - 260963467
Harmanpreet Singh - 260962547

# 1. Theory

1. **Question 1. Off-policy Reinforcement Learning**
**Answer (i)** Reference paper: [1]. Given an off-policy setting, where *behaviour policy* $\mu$ is to gather data and *target policy* $\pi$ is to evaluate. Consider importance sampling ratios to re-weight the states to mitigate the off-policy learning divergence caused due to the different state distribution of target and behaviour policy. Updates to the weights $\theta$ of the linear function approximator at time $t$ as below,

$$\Delta_t = \alpha(R_t^\lambda - \theta^\top \phi_t)\phi_t \rho_1 \rho_2 \dots \rho_t$$

where,

$$\rho_i = \frac{\pi(s_i, a_i)}{\mu(s_i, a_i)}$$

$$R_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} \bar{R}_t^{(n)}$$

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n \theta^\top \phi_{t+n}$$

$$\bar{R}_t^{(n)} = r_{t+1} + \gamma r_{t+2}\rho_{t+1} + \ldots + \gamma^{n-1} r_{t+n}\rho_{t+1}\dots\rho_{t+n-1} + \gamma^n \rho_{t+1}\dots\rho_{t+n}\theta^\top \phi_{t+n}$$

and, $\alpha$ is the learning rate

To show,

$$\mathbb{E}_\mu[\Delta\tilde{\theta}|s_0, a_0] = \mathbb{E}_\pi[\Delta\theta|s_0, a_0], \forall s_0 \in S, a_0 \in A$$

where $\Delta\theta$ and $\Delta\tilde{\theta}$ are the sum of the parameter increments over an episode under on-policy TD($\lambda$) and importance sampled TD($\lambda$) respectively, assuming that the starting weight vector is $\theta$ in both cases.

$$\begin{aligned}
\mathbb{E}_\mu[\Delta\tilde{\theta}|s_0, a_0] &= \mathbb{E}_\mu\left[\sum_{t=0}^{\infty} \alpha(R_t^\lambda - \theta^\top\phi_t)\phi_t\rho_1\rho_2\dots\rho_t\right] \\
&= \mathbb{E}_\mu\left[\sum_{t=0}^{\infty} \alpha((1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\bar{R}_t^{(n)} - \theta^\top\phi_t)\phi_t\rho_1\rho_2\dots\rho_t\right] \\
&= \mathbb{E}_\mu\left[\sum_{t=0}^{\infty} \alpha((1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\bar{R}_t^{(n)} - (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\theta^\top\phi_t)\phi_t\rho_1\rho_2\dots\rho_t\right] \\
&= \mathbb{E}_\mu\left[\sum_{t=0}^{\infty} \alpha(1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}(\bar{R}_t^{(n)} - \theta^\top\phi_t)\phi_t\rho_1\rho_2\dots\rho_t\right] \\
&= \mathbb{E}_\mu\left[\sum_{t=0}^{\infty}\sum_{n=1}^{\infty} \alpha(1-\lambda)\lambda^{n-1}(\bar{R}_t^{(n)} - \theta^\top\phi_t)\phi_t\rho_1\rho_2\dots\rho_t\right]
\end{aligned}$$

Now, we can say that if we can prove above for any n-step TD($\lambda$), we are good, i.e.,

$$\mathbb{E}_\mu\left[\sum_{t=0}^{\infty}(\bar{R}_t^{(n)} - \theta^\top\phi_t)\phi_t\rho_1\rho_2\dots\rho_t\right] = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}(R_t^{(n)} - \theta^\top\phi_t)\phi_t\right]$$

Let $\Omega_t$ denote the set of all possible paths of state-action pairs starting with $s_0, a_0$ and going through time $t$. Let $\omega$ denote one such trajectory and $p_\mu(\omega)$ its probability of occurring under policy $\mu$. Now, using the markov property, we expand the $\mathbb{E}_\mu[\cdot]$ as follow,

$$\begin{aligned}\mathbb{E}_\mu\left[\sum_{t=0}^{\infty}(\bar{R}_t^{(n)} - \theta^\top\phi_t)\phi_t\rho_1\rho_2\dots\rho_t\right] &= \sum_{t=0}^{\infty}\sum_{\omega\in\Omega_t} p_\mu(\omega)\mathbb{E}_\mu\left[\bar{R}_t^{(n)} - \theta^\top\phi_t|s_t,a_t\right]\phi_t\rho_1\rho_2\dots\rho_t \\ &= \sum_{t=0}^{\infty}\sum_{\omega\in\Omega_t} p_\mu(\omega)\phi_t\prod_{k=1}^{t}\rho_k\mathbb{E}_\mu\left[\bar{R}_t^{(n)} - \theta^\top\phi_t|s_t,a_t\right]\end{aligned}$$

We can write $p_\mu(\omega)$ in-terms of transition probabilities for behavior policy $\mu$. Therefore substituting the following $p_\mu(\omega)$ equation in $\mathbb{E}_\mu[\cdot]$

$$p_\mu(\omega) = \prod_{j=1}^{t} p(s_j|s_{j-1},a_{j-1})\mu(s_j,a_j)$$

$$\begin{aligned}\mathbb{E}_\mu[\cdot] &= \sum_{t=0}^{\infty}\sum_{\omega\in\Omega_t} p_\mu(\omega)\phi_t\prod_{k=1}^{t}\rho_k\mathbb{E}_\mu\left[\bar{R}_t^{(n)} - \theta^\top\phi_t|s_t,a_t\right] \\ &= \sum_{t=0}^{\infty}\sum_{\omega\in\Omega_t}\prod_{j=1}^{t} p(s_j|s_{j-1},a_{j-1})\mu(s_j,a_j)\phi_t\prod_{k=1}^{t}\rho_k\mathbb{E}_\mu\left[\bar{R}_t^{(n)} - \theta^\top\phi_t|s_t,a_t\right] \\ &= \sum_{t=0}^{\infty}\sum_{\omega\in\Omega_t}\prod_{j=1}^{t} p(s_j|s_{j-1},a_{j-1})\mu(s_j,a_j)\phi_t\prod_{k=1}^{t}\frac{\pi(s_k,a_k)}{\mu(s_k,a_k)}\mathbb{E}_\mu\left[\bar{R}_t^{(n)} - \theta^\top\phi_t|s_t,a_t\right] \\ &= \sum_{t=0}^{\infty}\sum_{\omega\in\Omega_t}\prod_{j=1}^{t} p(s_j|s_{j-1},a_{j-1})\pi(s_j,a_j)\phi_t\mathbb{E}_\mu\left[\bar{R}_t^{(n)} - \theta^\top\phi_t|s_t,a_t\right] \\ &= \sum_{t=0}^{\infty}\sum_{\omega\in\Omega_t}\prod_{j=1}^{t} p(s_j|s_{j-1},a_{j-1})\pi(s_j,a_j)\phi_t\cdot\left(\mathbb{E}_\mu\left[\bar{R}_t^{(n)}|s_t,a_t\right] - \theta^\top\phi_t\right)\end{aligned}$$

Now, as per the per-decision off-policy $n$-step importance sampling algorithm introduced by Precup, Sutton and Singh (2000) [2], $\mathbb{E}_\mu[\bar{R}_t^\lambda|s_t,a_t] = \mathbb{E}_\pi[R_t^\lambda|s_t,a_t]$. We use

this result to further solve $\mathbb{E}_\mu[\cdot]$

$$\begin{aligned}\mathbb{E}_\mu[\cdot] &= \sum_{t=0}^{\infty}\sum_{\omega\in\Omega_t}\prod_{j=1}^{t} p(s_j|s_{j-1},a_{j-1})\pi(s_j,a_j)\phi_t\cdot\left(\mathbb{E}_\mu\left[\bar{R}_t^{(n)}|s_t,a_t\right]-\theta^\top\phi_t\right)\\ &= \sum_{t=0}^{\infty}\sum_{\omega\in\Omega_t} p_\mu(\omega)\cdot\phi_t\cdot\left(\mathbb{E}_\pi\left[R_t^{(n)}|s_t,a_t\right]-\theta^\top\phi_t\right)\\ &= \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}(R_t^{(n)}-\theta^\top\phi_t)\phi_t\right]\end{aligned}$$

$$\mathbb{E}_\mu[\Delta\tilde{\theta}|s_0,a_0] = \mathbb{E}_\pi[\Delta\theta|s_0,a_0], \forall s_0\in S, a_0\in A$$

Hence proved!

**Answer (ii)** Consider eligibility trace vector $\boldsymbol{z_t}$ which has same number of components as weight vector $\boldsymbol{\phi_t}$. We also introduce a non-negative random variable $g_t$, which is allowed to depend only on events up to (and including) time $t$. The value $g_t$ represents the extent to which an episode is considered to start at time $t$. The function $g$: $\Omega_t \to R^+$ gives the expected value of $g_t$ for any trajectory up through $t$.

**Algorithm 1** Online algorithm update rule using traces for importance sampled TD($\lambda$)

**for** *each episode* **do**
  Initialize $c_0 = g_0$, $z_0 = c_0\phi_0$
  **for** $0 \le t < T$, *on each transition* $(s_t,a_t)\to(r_{t+1},s_{t+1},a_{t+1})$ **do**
    $\rho_{t+1} = \frac{\pi(s_{t+1},a_{t+1})}{\mu(s_{t+1},a_{t+1})}$
    $\delta_t = r_{t+1} + \gamma\rho_{t+1}\theta^\top\phi_{t+1} - \theta^\top\phi_t$
    $\Delta\theta_t = \alpha\delta_t z_t$
    $c_{t+1} = \rho_{t+1}c_t + g_{t+1}$
    $z_{t+1} = \gamma\lambda\rho_{t+1}z_t + c_{t+1}\phi_{t+1}$
  **end**
  $\theta \leftarrow \theta + \sum_t \Delta\theta_t$
**end**

2. **TRACK A:**
Experience replay is a commonly used technique with deep neural networks to stabilize the training. The experience replay stores the agent's experience at each time step in a replay memory (or a buffer) that is accessed to perform the weight updates. This is precisely what is done in DQN [3].

   (a) Highlight and explain in detail what are the advantages of using Q-learning with experience replay.

   (b) Explain the idea of prioritized experience replay (Schaul et al., 2015).

   (c) Highlight and compare the pros and cons of using prioritized experience replay over original experience replay.

(a) **Experience Replay** is a technique used with deep neural networks to stabilize the training by storing the agent's experience at each time step in a replay memory. It lets online reinforcement learning agents remember and reuse experiences from the past. It is important to note that experience replay itself is not a complete learning algorithm, it has to be combined with other algorithms like Q-learning to form a complete learning system.

Instead of running Q-learning on state/action pairs as they occur during simulation or the actual experience, the system stores the data discovered for [state, action, reward, next-state] – in a large table and then we sample a small batch of tuple to feed our neural network. Using Experience Replay with Q-learning helps us solve two problems i.e. to reduce correlations between experiences and to avoid forgetting the previous rare experiences. As a consequence of using an experience replay, it can be more efficient to make use of previous experience, by learning with it multiple times instead of throwing away the experience.

Since we know our next action pair is dependent on the previous state, our sequences of training samples are highly correlated in nature and we risk our agent being influenced by this effect of this correlation. By sampling from the replay buffer at random, we can break this correlation. This prevents action values from oscillating or diverging catastrophically. Experience replay not only provides uncorrelated data to train a neural network, but also significantly improves the data efficiency (Lin 1992;Wanget al. 2016), which is a desired property for many RL algorithms as they are often pretty hungry for data.

(b) **Prioritized Experience Replay** was introduced in 2015 (Schaul et al., 2015) [4]. Experiences in the replay were uniformly sampled from a replay memory at the same frequency that they were originally experienced, regardless of their significance. This method however prioritizes some experiences that may be more important than others for our training, but might occur less frequently, and therefore learn more efficiently. Experience replay frees online learning agents from processing transitions in the exact order they are experienced by randomly sampling from the replay buffer. Prioritized replay further liberates agents from considering transitions with the same frequency that they are experienced. Prioritized experience replay tries to change

the sampling distribution by using a criterion to define the priority of each tuple of experience. The priority is given to the experience where there is a big difference between our prediction and the TD target, since it means that we have a lot to learn about it. Greedily sampling using TD-error prioritization have several issues and the author proposes two ways of getting these priorities.

(a) Rank Based Method: $p_i = 1/rank(i)$ which sorts the items according to the TD-error($\delta$) to get the rank.

(b) Proportional Method:$p_i = |\delta| + \epsilon$, where $\epsilon$ is a small constant ensuring that the sample has some non-zero probability of being drawn.

During exploration, we may not know the $p_i$ for new states because we wouldnt have the TD error for them. So thoses states are initialized with the maximum priority so far, thus favouring those terms during sampling. From this, we can easily get the probability distribution.

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

where $\alpha$ determines the level of prioritization. $\alpha \to 0$ means no prioritization and $\alpha \to 1$ means full prioritization which would likely cause over-fitting.

This prioritization introduces biases because it changes this distribution in an uncontrolled fashion, and therefore changes the solution that the estimates will converge to. This can be corrected by using importance sampling weights. Since we are not using the 'true' uniform distribution and instead over-sampling those with high priorities, importance sampling correction should reduce the impact of the sampled term and it does by scaling the gradient term so that it has less impact on the parameters.

$$w_i = \Big(\frac{1}{N} \cdot \frac{1}{P(i)}\Big)^\beta$$

where N is the experience replay size and $P(i)$ represents the probability of sampling data point i according to priorities. The $\beta$ controls the amount of prioritization to apply. We control the amount of importance sampling to apply by defining a schedule over $\beta$ which gradually reaches 1 towards the end. Note that the choice of this hyper-parameter interacts with choice of prioritization exponent $\alpha$; increasing both simultaneously prioritizes sampling more aggressively at the same time as correcting for it more strongly.

(c) **Pros of Prioritized Experience Replay**
The benefits of Prioritized Experience Replay are abundantly clear as the agent can learn more effectively from some transitions than from others and hence prioritizing these with some weights is beneficial and helps obtain faster learning and state-of-the-art performance.

Considerations that help determine which transitions to replay are likely to also be relevant for determining which memories to store and when to erase them (e.g. when it becomes likely that they will never be replayed anymore). An explicit control over which memories to keep or erase can help reduce the required total memory size, because it reduces redundancy while automatically adjusting for what has been learned already (dropping many of the 'easy' transitions) and biasing the contents of the memory to where the errors remain high.

It can also be viewed in the context of prioritized supervised learning where the unbalanced samples are boosted to put the focus of the model on those sample by devoting additional resources to the (hard) boundary cases.

Intermediate variants by choosing $\alpha, \beta$, possibly with annealing or ranking are very useful in practice as they can control the bias-variance trade off.

**Cons of Prioritized Experience Replay**
The greedy prioritization had some cons like addition of bias due to the change in sampling distribution from uniform to uncontrolled prioritized sampling. Also the states with a low TD error will never be seen again during the sampling. The TD errors can be a poor estimate in the some circumstances where the rewards are noisy. Moreover, with function approximation, the initially high error transitions get replayed frequently and the system may be get prone to over-fitting due to lack of diversity. However all these problems are overcome by using a stochastic sampling method that interpolates between pure greedy prioritization and uniform random sampling.

In Prioritized Experience replay, the priorities for new transitions were initialized to the maximum priority seen so far, and only updated once they were sampled. This would not scale well in practice because the samples would need to wait for a long time to get updated.

Another practical problem arises during sampling as the complexity for sampling from such a distribution cannot depend on N(replay size). To overcome this, DeepMind implemented a "Sum-Tree" data structure which save the transition priorities and the sum over all underlying priority of its children. Leaving the parent node with the sum over all priorities. This gives an efficient way of calculating cumulative sums of priorities, allowing O(logn) updates and sampling. Though tree search simplifies our problem, it still faces the curse of dimensionality.

## 2. Coding

1. **Baird's counterexample**
   https://colab.research.google.com/drive/1gpsPFM4VeGf5enfy4K4Exv1LPFbSlvRe
2. **Cartpole REINFORCE/Actor-Critic — Reference:** [5, 6]
   https://colab.research.google.com/drive/1LCR6xAifg3fXRj665zT9uNeiV1IF07wf

## References

[1] Doina Precup, Richard S Sutton, and Sanjoy Dasgupta. Off-policy temporal-difference learning with function approximation. In *ICML*, pages 417–424, 2001.

[2] D Precup, RS Sutton, and S Singh. Eligibility traces for off-policy policy evaluation. int. conf. on machine learning, 2000.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[4] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[5] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[6] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.