Indian Institute of Technology Bombay

EE 691 RnD Project

Average Reward Deep Reinforcement Learning

January 13, 2022

Author	*Roll Number*
Tejas Pagare	190070067

under the guidance of

Prof. Vivek Borkar

Electrical Engineering, IIT Bombay

dedicated to Aai

Contents

1 Preliminaries

1.1 Finite-State Markov Chains

Definition 1.1. A Finite Markov chain is an integer-time process i.e. $\{X_n, n \geq 0\}$ where each X_n is a random variable from a countable finite set $\mathcal{S}$ and we have

$$\Pr\{X_n = j | X_{n-1} = i, X_{n-2} = k, \ldots, X_0 = m\} = \Pr\{X_n = j | X_{n-1} = i\} = P_{ij}$$

which says that state X_n is independent of past states given the last state X_{n-1} and further it doesn't depend on n. We denote the pair $\{\mathcal{S}, \mathcal{P}\}$ as a stationary finite-state markov chain, where $\mathcal{S}$ is finite set of states and $\mathcal{P}$ is a stochastic matrix or state transition probability matrix with P_{ij} as its elements.

Classification of states of a finite-state markov chain $\{\mathcal{S}, \mathcal{P}\}$

Denoting $\Pr\{X_n = j | X_0 = i\} = P_{ij}^n$ as Probability that after n^{th} transition $X_n = j$ starting from initial state $X_0 = i$.

A state j is accessible from state i ($i \to j$) iff $\Pr\{X_n = j | X_0 = i\} > 0$ for some $n \geq 1$.

Two state i and j communicate ($i \leftrightarrow j$) if i is accessible from j and j is accessible from i (i.e. $i \to j$ and $j \to i$) or if there exists two positive integers k_1 and k_2 such that $P_{ij}^{k_1} > 0$ and $P_{ji}^{k_1} > 0$.

Definition 1.2. For finite-state Markov chains, a recurrent state is a state i that is accessible from all states that are accessible from i (i is recurrent if $i \to j$ implies that $j \to i$). A transient state is a state that is not recurrent. A state i is transient if there is some j that is accessible from i but from which there is no possible return.

Definition 1.3. A class $\mathcal{C}$ of states is a non-empty set of states such that each $i \in \mathcal{C}$ communicates with every other state $j \in \mathcal{C}$ and communicates with no $j \notin \mathcal{C}$.

A recurrent or ergodic class therefore is a set of recurrent states that all communicate with each other and do not communicate with states outside this class.

Theorem 1.1. For finite-state Markov chains, either all states in a class are transient or all are recurrent.

If $\mathcal{S}$ itself is a recurrent class then we say that the Markov chain is **irreducible**.

We have $\lim_{k \to \infty} P_{ij}^k = 0$ iff i is transient.

Main takeaway - If the process starts in a recurrent class, it stays within that class. If it starts in a recurrent class, it eventually enters in a recurrent class (with probability one) after a finite number of transitions, and hence subsequently remains there.
Please refer to [9] and Appendix D of [4] for detailed description.

1.2 Markov Decision Processes

Definition 1.4. A MDP consists of

- A set of states $\mathcal{S}$, set of actions $\mathcal{A}$ for moving from a state to another. $\mathcal{S}$ and $\mathcal{A}$ can be both finite or infinite.
- Transition Probabilities P which is defined the probability distribution over next states given the current state and current action where $P_{ij}(a) = \Pr\{X_{n+1} = j | X_n = i, U_n = a\}$.
- A reward function which is defined as $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, where $\mathcal{R}_s^a$ or $r(s, a)$ is the expected reward of taking action a in state s.

Therefore a MDP is simply given as a pair $(\mathcal{S}, \mathcal{A}, \mathcal{R})$.

A policy $\pi : \mathcal{S} \to \mathcal{A}$ is a distribution over actions given states $\pi(a|s) = \Pr\{U_n = a | X_n = s\}$.
For stationary policy we have $U_n \sim \Pr(\cdot | X_n = s)\ \forall t > 0$.

Any policy induces a Markov chain whose transition probability or stochastic matrix is denoted as $P(\pi) \in [0,1]^{|\mathcal{S}|\times|\mathcal{S}|}$ where each entry $P_{ij}(\pi) = \sum_{a\in\mathcal{A}} \pi(a|s)p(j|i,a)$. This Markov chain is denoted as $(\mathcal{S}, \mathcal{P}(\pi))$.

Below definitions for MDPs are simply the extensions of the Markov chains under a stationary policy.
We say that a state i is accessible from state j if there exists a stationary policy π and an integer k such that $P(x_k = j|x_0 = i, \pi) > 0$

Definition 1.5. An MDP is ergodic or recurrent if the Markov chain induced by any stationary policy is ergodic or the transition matrix of the induced Markov chain has a single recurrent class.

Definition 1.6. An MDP is unichain if it consists of a single recurrent class and possibly some transient states.

An MDP is communicating if every pair of states communicate with each other under a stationary policy.

Definition 1.7. MDPs in which at least one policy results in two or more closed communicating classes (recurrent classes) and a transient class (possibly empty) are called Multichain MDPs.

A broader condition known as *Weak Accessibility* from Chapter 4. of [4] is important for most of the results to hold.

Definition 1.8. A Weak Accessibility (WA) condition holds if the set of states can be partitioned into two subsets S_t and S_c such that:
(i) All states in S_t are transient under every stationary policy.
(ii) For every two states i and j in S_c, j is accessible from i.

1.3 Average Reward Problems

Please refer to [5, 7, 12] for a broader view.
The objective here is to maximize over all policies $\pi = \{\mu_0, \mu_1, \dots\}$ where $\mu_n(i) \in U(i)$ denotes the *valid action in state i* for all i and n, the expected average reward per stage (also termed as gain) starting from a state i.

$$\rho^{\pi}(i) = \liminf_{N\to\infty} \frac{1}{N} E\Big\{ \sum_{n=0}^{N-1} r(x_n, \mu_n(x_n))|x_0 = i \Big\} \tag{1}$$

where r denotes the instantaneous reward incurred after taking action $\mu_n(x_n)$ in state x_n. Here, lim inf is used as opposed to lim to justify the fact that lim may not generally exist.

We note that, rewards incurred in the early stage do not matter since their contribution to the average reward per stage is reduced to zero as $N \to \infty$

$$\lim_{N\to\infty} \frac{1}{N} E\Big\{ \sum_{n=0}^{K} r(x_n, \mu_n(x_n))|x_0 = i \Big\} = 0 \tag{2}$$

for any fixed K.

$$\therefore \rho^{\pi}(i) = \rho^{\pi}(j)\ \forall i, j \text{ with } E\{N_{ij}(\pi)\} < \infty$$

where $N_{ij}(\pi)$ denotes the time of first visit to state j starting from state i and this condition is equivalent to saying that the system reaches j starting from i with probability 1.

Definition 1.9. A gain optimal policy $\pi*$ is defined as the policy which maximizes the average reward for all states i.e.

$$\pi^* \text{ is gain optimal if } \rho^{\pi^*}(s_0) \geq \rho^{\pi}(s_0)\ \forall\pi \text{ and } \forall s_0 \in \mathcal{S}$$
$$\text{and hence } \pi^* \in \arg\max_{\pi} \rho^{\pi}$$

Now, we refer to the general case of unichain MDPs which follows the broader *Weak Accessibility* criteria and hence shown the existence of Blackwell optimality policies refer Definition 1.11. We now state the important result that for unichain MDPs, the average reward for any stationary policy is independent of the initial state.

$$\rho^{\pi}(s) = \rho^{\pi}(s') = \rho^{\pi} \ \forall \pi \text{ and } \forall s, s' \in \mathcal{S}$$

This can be shown by a simple lookup of the unichain condition Definition 1.6.
Since, we know that the states in the recurrent class will be visited forever and hence the expected average reward cannot differ across the states in the recurrent class.
Now, for transient states we know that they will be never be reentered after we are in recurrent class, hence the transient states accumulate a finite total expected reward. This finite reward vanishes under the limit as stated in Eq. (2)

We define bias value $V_b^{\pi}(s_0)$ as the average adjusted sum of rewards from a policy

$$V_b^{\pi}(s_0) := \lim_{N\to\infty} E\Big\{ \sum_{n=0}^{N-1} \Big(r(x_n, \mu_n(x_n)) - \rho^{\pi} \Big) | x_0 = s_0 \Big\} \tag{3}$$

where ρ^{π} is the average reward associated with the policy π.
This bias value or relative value has the interpretation of relative difference in total reward gained by starting from some state s as opposed to starting from some other state s' which can be shown by following:

$$\because V_b^{\pi}(s) - V_b^{\pi}(s') = \lim_{N\to\infty} \left[E\Big\{ \sum_{n=0}^{N-1} r(x_n, \mu_n(x_n)) | x_0 = s \Big\} - E\Big\{ \sum_{n=0}^{N-1} r(x_n, \mu_n(x_n)) | x_0 = s' \Big\} \right]$$

Definition 1.10. A policy π^* is termed as *bias optimal* if it is gain optimal and it maximizes the bias values i.e. $V_b^{\pi^*}(s) \geq V_b^{\pi}(s) \forall s \in \mathcal{S}$ and stationary policies π.

Definition 1.11. A stationary policy π is said to be *Blackwell optimal* if it is simultaneously optimal for all the α-discounted problems with α in an interval $(\overline{\alpha}, 1)$ where $\overline{\alpha}$ is some scalar with $0 < \overline{\alpha} < 1$.

1.3.1 Bellman Equation

Theorem 1.2. For any unichain MDP, there exists a value function V^* and a scalar ρ^* satisfying the equation.

$$V^*(s) + \rho^* = \max_a \Big\{ r(s,a) + \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \Big\} \quad \forall s \in \mathcal{S} \tag{4}$$

The greedy policy π^* which selects action by maximizing the RHS of this equation achieves optimal average reward $\rho^* = \rho^{\pi^*}$ and is a gain-optimal policy.

2 Value Iteration Methods

We consider a controlled Markov chain $\{X_n\}$ on a finite state space $\mathcal{S} = \{1, 2, \ldots, d\}$ with a finite action space $\mathcal{A} = \{a_1, \ldots, a_r\}$ and transition probabilities $p(\cdot|s,a)$.
Also, from now on we assume Unichain condition (Definition 1.6) throughout unless specified.

Consider the mapping $T(V)(s)$ obtained by applying RHS of the Bellman equation (4)

$$T(V)(s) = \max_a \Big(r(s,a) + \sum_{s'} p(s'|s,a) V(s') \Big) \tag{5}$$

This mapping is shown to be a monotone mapping i.e. given two value functions $V(s)$ and $V'(s)$, where $V(s) \leq V'(s)$ $\forall s \in \mathcal{S}$, implies that $T(V)(s) \leq T(V')(s)$. This is important result for the further analysis to hold true.

The Relative Value Iteration algorithm is given by the iteration

$$V^{n+1}(s) = T(V^n)(s) - V^n(s_{\text{ref}}) \quad \forall s \in \mathcal{S} \tag{6}$$

where s_{ref} is an arbitrary but fixed reference state.

$$\textit{It is shown that as } k \to \infty,\ V^n(s_{\text{ref}}) \to \rho^{\pi^*}$$

However, it should be noted that $V^n(s_{\text{ref}})$ is not unique offset term as we describe later.
The iteration becomes

$$V^{n+1}(s) = \max_a \Big\{ r(s,a) + \sum_{s'} p(s'|s,a) V^n(s') \Big\} - V^n(s_{\text{ref}}) \tag{7}$$

This is the *Relative Value Iteration with synchronous updates.*

2.1 RVI Q Learning

We use RVI Q-learning [1] as a base of our algorithm.
Define the the relative state-action *"Q-value"* as following

$$Q^\pi(s,a) := \lim_{N\to\infty} E\Big\{ \sum_{k=0}^{N-1} \Big(r(x_k, \mu_k(x_k)) - \rho^\pi \Big) | x_0 = s, u_0 = a \Big\} \quad \forall (s,a) \in \mathcal{S} \times \mathcal{A} \tag{8}$$

We have the following Bellman Optimality equation for this

$$Q^*(s,a) + \rho^* = r(s,a) + \sum_{s' \in \mathcal{S}} p(s'|s,a) V_b^*(s') \quad \forall (s,a) \in \mathcal{S} \times \mathcal{A} \tag{9}$$

where $V_b^*(s') = \max_{a' \in \mathcal{A}} Q^*(s', a')$

The RHS of Eq. (9) has the same form as the quantity maximized over all actions in Eq. (4). Hence, we can obtain a gain optimal policy π^* by acting greedily according to the optimal state-action value.

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} Q^*(s,a) = \operatorname*{argmax}_{a \in \mathcal{A}} \Big[r(s,a) + \sum_{s' \in \mathcal{S}} p(s'|s,a) \max_{a' \in \mathcal{A}} Q^*(s', a') - \rho^* \Big]$$

We note that ρ^* is invariant to state s and action a and hence it has no effect in the above maximization step. This can be viewed as subtracting an "offset" to make the iterations *stable.*

By using this iteration we see that optimal policy π^* (also termed as a greedy policy) can be obtained by simply maximizing the *"Q-value"* without requiring the knowledge of reward or transition probabilities. Hence, it is suitable for data-driven algorithms of RL. And as we will see further this also facilitates stochastic approximation.

By applying the idea of RVI on state-action values Q^* we get the following DP equation:

$$\begin{aligned} Q^{n+1}(s,a) &= T(Q^n)(s,a) - V^n(s_{\text{ref}}) \\ &= r(s,a) + \sum_{s' \in \mathcal{S}} p(s'|s,a) \max_{a' \in \mathcal{A}} Q^n(s', a') - \max_{a'' \in \mathcal{A}} Q^n(s_{\text{ref}}, a'') \end{aligned}$$

where Q^n denotes the estimate of Q^* at iteration n.

For this we need to estimate $|\mathcal{S}| \times |\mathcal{A}|$ values instead of $|\mathcal{S}|$ in case of V iterations and sometimes $|\mathcal{S}| \times |\mathcal{A}| >> |\mathcal{S}|$. We also see that now the nonlinearity is inside the conditional expectation w.r.t. the transition probability function.

Now the principle idea of RVI Q-learning is to use stochastic approximation algorithm where we first replace this conditional expectation by actual evaluation at a real or simulated random variable ξ_{ia} with law $p(\cdot|s,u)$, and then make an incremental correction to the current guess based on it. This is based on the well-known averaging effect of stochastic approximation, which I refer reader to the textbook [6] (I myself unaware of this now!).

$$Q^{n+1}(s,a) = (1-a(n))Q^n(s,a) + a(n)\Big(r(s,a) + \max_{a' \in \mathcal{A}} Q^n(\xi_{ia}, a') - f(Q^n)\Big)$$

We obtain the following synchronous RVI Q-learning algorithm

$$\begin{aligned} Q^{n+1}(s,a) = Q^n(s,a) + a(n)\Big(r(s,a) + \max_{a' \in \mathcal{A}} Q^n(\xi_{ia}, a') \\ - f(Q^n) - Q^n(s,a)\Big) \end{aligned} \tag{10}$$

where $\{a(n)\} \in (0,1)$ is the diminishing step size satisfying the standard Robins-Monro conditions of stochastic approximation

$$\sum_n a(n) = \infty \quad \text{and} \quad \sum_n a(n)^2 < \infty$$

ξ_{ia} are independent $\mathcal{S}$ valued random variables with the law $p(\cdot|s,u)$.

If we consider a single run of the controlled Markov Chain $\{(X_n, U_n)\}$ then, when $\{(X_n =, U_n = u)\}$ X_{n+1} is given by the law same conditional of $p(\cdot|i,u)$.

Denote $v(n,i,u)$ as the number of updates of the $(i,u)^{\text{th}}$ component of Q till time n. We have,

$$v(n,i,u) = \sum_{k=0}^{n} I\{X_k = i, U_k = u\}$$

$I\{\dots\}$ is an indicator random variable which equal 1 if '$\dots$' holds else 0.

Then the asynchronous RVI Q-learning becomes

$$\begin{aligned} Q^{n+1}(i,u) = Q^n(i,u) + a(v(n,i,u))I\{X_n = i, U_n = u\}\Big(r(i,u) + \max_{v \in \mathcal{A}} Q^n(X_{n+1}, v) \\ - f(Q^n) - Q^n(i,u)\Big) \qquad \forall (i,u) \in \mathcal{S} \times \mathcal{A} \end{aligned} \tag{11}$$

since only the $(i,u)^{th}$ component is updated at time n.

Assumption 2.1. f is Lipschitz and for e equal to a constant vector of all 1's in $R^{d \times r}$, $f(e) = 1$ and $f(x+ce) = f(x)+c$ for some scalar $c \in R$.

Assumption 2.2. Comparable exploration $\forall (i,u) \in \mathcal{S} \times \mathcal{A}$, mathematically $\exists\ \Delta$ such that

$$\liminf_{n \to \infty} \frac{v(n,i,u)}{n+1} \geq \Delta \text{ a.s}$$

3 Towards Deep Q-Learning

We first of start by introducing Deep Q-Learning for discounted reward case and then formulate our average reward DQN. We denote discount by γ.
Tabular Q-learning scheme suffers from the 'curse of dimensionality' of MDPs and one obvious way to deal with this to replace Q by a parameterized family $(x,u,\theta) \to Q(x,u;\theta)$ where $\theta \in \Theta \subset R^d$ which gives rise to the world of Deep Q Learning or Deep RL in general (here d maybe enormous).

So, we start of with the 'true Bellman Error' minimising of which serves as the objective and is one of the measure of performance (NOTE: this maybe not be a best measure *sometimes*).

$$\bar{\mathcal{E}}(\theta) := E\Big[\Big(r(X_n, U_n) + \gamma \sum_{y \in \mathcal{S}} p(y|X_n, U_n) \max_v Q(y, v; \theta) - Q(X_n, U_n; \theta)\Big)^2\Big]. \tag{12}$$

The DQN algorithm is given by the following iterate

$$\theta_{n+1} = \theta_n + a(n)(Z_n - Q(X_n, U_n; \theta_n))\nabla_\theta Q(X_n, U_n; \theta_n), \quad n \geq 0, \tag{13}$$

where Z_n is the target value. For Vanilla DQN algorithm

$$Z_n := r(X_n, U_n) + \gamma \max_v Q(X_{n+1}, v; \theta_n) \tag{14}$$

Here, both target and Q value is parameterized by the same variable θ hence after every step, we change the target value which implies we have a *moving target* which destabilize the training procedure.

For "Classic" DQN [13] algorithm, Z_n parameterized by a *target network* whose parameters θ' are updated on a slower time scale.

$$Z_n := r(X_n, U_n) + \gamma Q(X_{n+1}, v; \theta'_n)\Big|_{v=\text{argmax}_{v'} Q(X_{n+1}, v'; \theta'_n)} \tag{15}$$

Here, we can see that the Q value as well as action v to be selected comes from the same *target network* which implies in some sense that their noise is correlated. It has been experimentally shown that this approach leads to overestimation of Q values.

The workaround for this is to use different networks to compute Q values and v. This led to the advancement of "Double Q-learning" [17] and in practice it takes the following form

$$Z_n := r(X_n, U_n) + \gamma Q(X_{n+1}, v; \theta'_n)\Big|_{v=\text{argmax}_{v'} Q(X_{n+1}, v'; \theta_n)} \tag{16}$$

where we use the same Q-network to compute the action v and a target network to compute the corresponding Q value.

The idea here to update the target network every K iterations by means of simple copying the Q-network parameters or taking Polyak Averaging of them so that the target Z_n remains stable throughout the $Q(\cdot, \cdot; \theta_n)$ iterations.

Replay Buffer

Here, we also maintain a replay buffer in which we tend to store the transitions (X_n, U_n, R_n, X_{n+1}). This leads to the term multiplying $a(n)$ in Eq. (13) by an empirical average over past transitions.

$$\theta_{n+1} = \theta_n + \frac{a(n)}{M} \times \sum_{m=1}^{M} \Big((Z_{n(m)} - Q(X_{n(m)}, U_{n(m)}))\nabla_\theta Q(X_{n(m)}, U_{n(m)}; \theta_{n(m)})\Big), \ n \geq 0, \tag{17}$$

where $(X_{n(m)}, U_{n(m)}),\ 1 \leq m \leq N$, are samples from past.

Important thing to note is that for typical Bellman error gradient methods, we have a conditional expectation (w.r.t. $p(\cdot|X_n, U_n)$) of a product instead of a product of conditional expectations, as indicated by the actual Bellman error formula.
The essence of the above algorithms is the use of experience replay which does one of the conditional expectations ahead of time therefore we the expression now is approximately a product of conditional expectations.
This is so because we average over past traces (X_m, U_m, X_{m+1}) as shown in Eq. (17) where X_m, Um are fixed at the present state-action (X_n, U_n), and hence that is essentially a Monte Carlo evaluation of the conditional expectation.

Now, here since we update target networks every K iterations which implies a delay in the Q iterates and hence it is shown to be produce an asymptotically negligible additional error with decreasing stepsizes with o.d.e approach of stochastic approximation. I refer reader to [2] for a formal proof.

3.1 Full Gradient DQN

Full Gradient DQN (FGDQN) [2] treats both occurrences of θ in the iterate on equal footing by treating it as a single variable. We take the full gradient with this variable θ and hence it resembles stochastic gradient descent!

$$\begin{aligned}\theta_{n+1} = \theta_n - a(n)\Big(r(X_n, U_n) + \gamma \max_v Q(X_{n+1}, v; \theta_n) - Q(X_n, U_n; \theta_n)\Big) \times \\ (\gamma \nabla_\theta Q(X_{n+1}, v_n; \theta_n) - \nabla_\theta Q(X_n, U_n; \theta_n))\end{aligned} \tag{18}$$

for $n \geq 0$, where $v_n \in \text{Argmax} Q(X_{n+1}, \cdot; \theta_n)$ chosen according to some tie-breaking rule when necessary. Important thing to note that when the term inside maximizer is not unique we lose its differentiability, but it can be seen in terms of Frechet sub-differential. Also, popular Deep Learning framework like PyTorch [14] allows this type of gradient iteration by essentially backpropagating the maximum value.

The FGDQN iterate is further expressed as following

$$\begin{aligned}\theta_{n+1} = \theta_n - a(n)\Bigg(\overline{(r(X_n, U_n) + \gamma \max_v Q(X_{n+1}, v; \theta_n) - Q(X_n, U_n; \theta_n))} \times \\ (\gamma \nabla_\theta Q(X_{n+1}, v_n; \theta_n) - \nabla_\theta Q(X_n, U_n; \theta_n)) + \xi_{n+1}\Bigg)\end{aligned} \tag{19}$$

for $n \geq 0$, where $\{\xi_n\}$ is extraneous i.i.d. noise componentwise distributed independently and uniformly on $[-1, 1]$. The overline dictates a modified version of experience replay which comprises of averaging at time n over past traces sampled from $(X_k, U_k, X_{k+1}), k \leq n$, for which $\{X_k = X_n, U_k = U_n\}$. So, we only do averaging over those samples for which we are currently going to update the Q value!

The DQN algorithms defined are off-policy in the sense that it may not always be the case that the action v which we use to update Q is the action the policy takes in the next step. E.g. ϵ-greedy policy we takes action $\text{argmax}(Q(X_n, \cdot; \theta_n))$ with probability $1 - \epsilon$, and chooses a action independently and with uniform probability from $\mathcal{A}$, with probability ϵ.

3.2 Average Reward FGDQN

Now, we minimise the new Bellman Error derived from RVI Q-learning iteration Eq. (11)

$$\bar{\mathcal{E}}(\theta) := E\Big[\Big(r(X_n, U_n) + \sum_{y \in \mathcal{S}} p(y|X_n, U_n) \max_v Q(y, v; \theta) - Q(X_n, U_n; \theta) - f(Q)\Big)^2\Big]. \tag{20}$$

where again $f(Q)$ is not a unique choice as discussed in Assumption 2.1.
Let's first define the DQN variant which is given by the iterate

$$\theta_{n+1} = \theta_n + a(n)(Z_n - Q(X_n, U_n; \theta_n))\nabla_\theta Q(X_n, U_n; \theta_n), \quad n \geq 0, \tag{21}$$

Hence, similarly we define the corresponding target for Average Reward based Double DQN as

$$Z_n := r(X_n, U_n) - f(Q; \theta'_n) + Q(X_{n+1}, v; \theta'_n)\Big|_{v = \text{argmax}_{v'} Q(X_{n+1}, v'; \theta_n)} \tag{22}$$

We now propose our Average Reward Deep RL algorithm based on FGDQN for which the iteration becomes as follows

$$\begin{aligned}\theta_{n+1} = \theta_n - a(n)\Big(r(X_n, U_n) + \max_v Q(X_{n+1}, v; \theta_n) - f(Q; \theta_n) - Q(X_n, U_n; \theta_n)\Big) \times \\ \Big(\nabla_\theta Q(X_{n+1}, v_n; \theta_n) - \nabla_\theta f(Q; \theta_n) - \nabla_\theta Q(X_n, U_n; \theta_n)\Big)\end{aligned} \tag{23}$$

In this work, I haven't done any convergence analysis and I hope to do so as an extension of this work.

4 Restless Bandits

First, we will formulate the problem of Restless Multi Armed Bandits (RMAB). We use superscript α to indicate α-th arm and n, m are used to denote discrete time-steps.

- Actions available are active ($u = 1$) and passive ($u = 0$).
- At each time n, we can activate exactly M out of N arms i.e. $\sum_{k=0}^{N} U_n^k = M \ \forall n$.
- Arms with action $u = 0$ can possible evolve and accrue reward in a distinct way when $u = 1$.

The objective is to maximize long run average reward

$$\liminf_{n\to\infty} \frac{1}{n} E\left[\sum_{m=0}^{n-1}\sum_{\alpha=1}^{N} r^{\alpha}(X_m^{\alpha}, U_m^{\alpha})\right] \tag{24}$$

Textbook example to make the formulation clear.
Example: Consider a bandit which is a measure of someone's vigour (or strength, lack of fatigue). Denote active and passive actions corresponding to the notion of work and rest. It is obvious that strength increases with rest and decreases with work.
Suppose, we consider strength s as state having values in $\{1, 2, \ldots, k\}$. If action is activate arm in state s, it accrues an immediate reward of $r(s)$ which increases with s and strength decreases to $\max\{x-1, k\}$. While the passive action gives no reward, it increase the strength to $\min\{x+1, k\}$.

We consider a family of N MDPs, the actions at time n is denoted as $U_n = (U_n^1, .., U_n^{\alpha}, .., U_n^N)$ subject to the RMAB constraint that $U_n \in \Omega$ where $\Omega := \{(U^1, .., U^{\alpha}, .., U^N) : U^{\alpha} \in \{0, 1\} \ \forall \ \alpha \text{ and } \sum_{k=0}^{N} U_n^k = M\}$.

Optimizing the Restless Multi Armed Bandit Problems Eq. (24) has been shown to be PSPACE-hard.

Definition 4.1. A problem is in PSPACE, if it can be solved using an amount of *space* that is polynomially bounded by the input size. A decision problem is in PSPACE-hard if any problem in PSPACE can be reduced to it in polynomial time.

Whittle [18] proposed a relaxed constraint where instead of activating M arms 'per time instant', we in expectation activate M arms (a 'time-averaged constraint')

$$\liminf_{n\to\infty} \frac{1}{n} E\left[\sum_{m=0}^{n-1}\sum_{\alpha=1}^{N} U_m^{\alpha}\right] = M \tag{25}$$

Whittle further simplified the problem by considering Lagrangian relaxation. Which in simple terms removes constraint by penalizing the violation of the constraint, so now the decoupled problem becomes

$$\liminf_{n\to\infty} \frac{1}{n} E\left[\sum_{m=0}^{n-1}\sum_{\alpha=1}^{N} \Big(r^{\alpha}(X_m^{\alpha}, U_m^{\alpha}) + \lambda(1 - U_m^{\alpha})\Big)\right] + \lambda M \tag{26}$$

We can drop the constant factor of M to match the exact Whittle setup, this doesn't affect the optimization problem since the policy is based just on the ordinal comparison which will be discussed later.

So finally, the objective becomes

$$\liminf_{n\to\infty} \frac{1}{n} E\left[\sum_{m=0}^{n-1} (r^{\alpha}(X_m^{\alpha}, U_m^{\alpha}) + \lambda(1 - U_m^{\alpha}))\right] \tag{27}$$

where we solve this separately for each α.

The Lagrange multiplier λ has some really nice intuition given by Whittle. One can simply view it as a subsidy for activating an arm.

Consider the state space divided into three sets P_0, P_1, P_{01} where, respectively, the optimal action is $u = 0, u = 1$, or some randomization between both $u = 0$ and $u = 1$. Due to our unichain assumption, the set P_{01} will never contain more than one state, a fact that is known for general Markov decision processes with constraints; see Ross (1989).

So, we just take into account two sets and lets denote P_0 and P_1 by $\mathscr{P}$ and $\mathscr{P}^C$ respectively.

Definition 4.2. An arm α is considered to be *indexable* if the set $\mathscr{P}$ increases monotonically from Φ to the entire state space as we increase λ^α (the subsidy for passive action). The RMAB problem is *indexable* if all N arms are *indexable*.

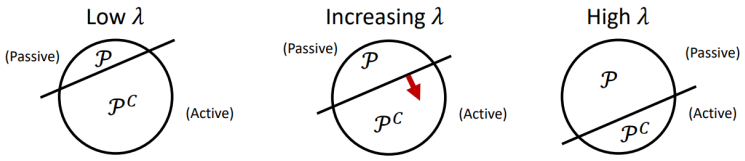


Figure 1

From the above Fig. (1) we can see that if an arm α alpha is rested (i.e. in set $\mathscr{P}$) for λ^α then it should also be rested for $\lambda_0^\alpha > \lambda^\alpha$ from the above indexability criteria.

Definition 4.3. For an arm α, Whittle index is defined as the $\lambda^\alpha = \inf\{\lambda^\alpha : \alpha \in \mathscr{P}\}$. It is the infimum subsidy λ^α one has to pay so that it is equally desirable to give an arm α action and passive action.

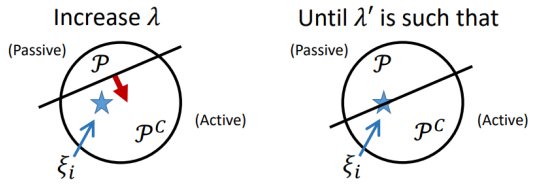


Figure 2: Whittle Index λ'

The above Fig. (2) makes the definition more clear, here ξ_i denotes the state and λ' is the whittle index.

Definition 4.4. A Whittle Index policy is the policy which activates or gives active actions to those M arms out of N which has highest whittle indices λ.

We note that, Whittle Index policy activates arms with highest whittle indices which are those M arms that are last to leave the state set $\mathscr{P}^C$ as the subsidy for the passive action λ increases.

4.1 Q-learning for Whittle Index

From [3] the Q-values satisfy the following DP equation

$$Q(i,u) = ur(i,1) + (1-u)(\lambda + r(i,0)) - \rho + \sum_j p(j|i,u) \max_v Q(j,v) \quad \forall i \in \mathcal{S}, u \in \mathcal{A} \text{ and all } M \text{ arms.} \tag{28}$$

Where we recall from above that ρ is uniquely specified optimal average reward and Q is unique upto an additive constant.

Whittle index $\lambda(\hat{k})$ is defined as

$$\lambda(\hat{k}) := r(\hat{k},1) + \sum_j p(j|\hat{k},1)V(j) - r(\hat{k},0) - \sum_j p(j|\hat{k},0)V(j). \tag{29}$$

which is equivalent to solving

$$Q(\hat{k},1) - Q(\hat{k},0) = 0, \quad \text{for } \lambda = \lambda(\hat{k}) \tag{30}$$

The proposed Q-learning for Whittle Index takes the following form which is essentially a two time scale optimization algorithm where we update the Q-values on a faster time-scale and update the whittle indices λ on a faster time scale so that the Q-value keep track of the slowly varying λ.

$$\begin{aligned} Q_{n+1}(i,u;\hat{k}) = Q_n(i,u;\hat{k}) + a(\nu(i,u,n)) \times I\{X_n = i, U_n = u\}\Big((1-u)(r(i,0) + \lambda_n(\hat{k})) + ur(i,1) \\ + \max_{v\in\mathcal{A}} Q_n(X_{n+1},v;\hat{k}) - f(Q_n(\hat{k})) - \ Q_n(i,u;\hat{k})\Big) \end{aligned} \tag{31}$$

where $\nu(i,u,n)$ is the local clock, f satisfies the Assumption 2.1 and $\lambda_n(\hat{k})$ is the estimated whittle index for state $\hat{k} \in \mathcal{S}$.

Whittle index $\lambda(\hat{k})$ for state $\hat{k}$ is updated by the following equation:

$$\lambda_{n+1}(\hat{k}) = \lambda_n(\hat{k}) + b(n)\left(Q_n(\hat{k},1;\hat{k}) - Q_n(\hat{k},0;\hat{k})\right). \tag{32}$$

where stepsize sequence $\{b(n)\}$ satisfies $\sum_n b(n) = \infty$, $\sum_n b(n)^2 < \infty$ and $b(n) = o(a(n))$.
Here we also assume comparable exploration 2.2.

The above method has shown to produce exact whittle indices.
We note here that as the state-space grows updating the indices λ is inefficient so we update λ's for a suitably chosen subset of $\mathcal{S}$ and interpolate.

4.2 FGDQN for Whittle-Index

Now, the obvious thing to do when we have a large state $\mathcal{S}$ and action $\mathcal{A}$ space is to use Deep Neural Networks as a function approximators. We get the following Q iteration for FGDQN by paremeterizing the Q and whittle network with parameters θ and θ' respectively.

$$\begin{aligned} \theta_{n+1} = \theta_n - a(n)\Bigg(\overline{(1-U_n)(r(X_n,0) + \lambda_n(\hat{k})) + U_n r(X_n,1) + \max_{v\in\{0,1\}} Q(X_{n+1},v;\theta_n,\hat{k}) - f(Q(\hat{k};\theta)) - Q(X_n,U_n;\theta_n,\hat{k})} \times \\ \Big(\nabla_\theta Q(X_{n+1},v_n;\theta_n,\hat{k}) - \nabla_\theta f(Q(\hat{k};\theta)) - \nabla_\theta Q(X_n,U_n;\theta_n,\hat{k})\Big) + \xi_{n+1}\Bigg) \end{aligned} \tag{33}$$

Now, we will prove our approach of using Whittle Indices along with FGDQN. Following we derive the iteration for θ' which is the Whittle Network parameters.
As we have seen, whittle index is equivalent to solving for $\lambda(\hat{k}_n)$ the following equation

$$Q(\hat{k}_n,1) = Q(\hat{k}_n,0)$$

Substituting $Q(\hat{k}_n,0)$ from the DP Eq. (28) we get

$$Q(\hat{k}_n,1) = r(\hat{k}_n,0) + \lambda(\hat{k}_n) - \rho + \sum p(\hat{k}_{n+1}|\hat{k}_n,0) \max_{v\in\{0,1\}} Q(\hat{k}_{n+1},v;\theta_n)$$

which is equivalent to

$$\lambda(\hat{k}_n) = Q(\hat{k}_n,1) - r(\hat{k}_n,0) + \rho - \sum p(\hat{k}_{n+1}|\hat{k}_n,0) \max_{v\in\{0,1\}} Q(\hat{k}_{n+1},v;\theta_n)$$

Now, using stochastic approximation we remove the conditional expectation by a real random variable ξ_{i0} with the law $p(\cdot|\hat{k}_n, 0)$ and make increment based on our current estimate. We consider a single run $\{X_n = \hat{k}_n, U_n = 0\}$ of the controlled Markov chain which gives $\hat{k}_{n+1}$ with the same conditional law.
$\therefore$ We now know have,

$$\lambda(X_n) = (1 - b(n))\lambda(X_n) + I\{X_n = \hat{k}_n, U_n = 0\}b(n)\Big(Q(X_n, 1) - r(X_n, 0) + \rho - \max_{v\in\{0,1\}} Q(X_{n+1}, v; \theta_n)\Big)$$

We replace ρ with it's current estimate i.e. $f(Q_n)$ to get

$$\lambda(X_n) = (1 - b(n))\lambda(X_n) + I\{X_n = \hat{k}_n, U_n = 0\}b(n)\Big(Q(X_n, 1) - r(X_n, 0) + f(Q_n) - \max_{v\in\{0,1\}} Q(X_{n+1}, v; \theta_n)\Big)$$

$$\begin{aligned}\lambda(X_n) = \lambda(X_n) + I\{X_n = \hat{k}_n, U_n = 0\}b(\mu(i, 0, n))\Big(Q(X_n, 1) - r(X_n, 0)\\ + f(Q_n) - \max_{v\in\{0,1\}} Q(X_{n+1}, v; \theta_n) - \lambda(X_n))\Big)\end{aligned} \tag{34}$$

here, $\mu(i, 0, n)$ is a local clock for Whittle Index and $b = o(n)$ also $a(n)$ and $b(n)$ satisfy the standard Robins-Monro Condition for stochastic approximation.

Following we get the iteration for Whittle Index parameters θ'

$$\begin{aligned}\theta'_{n+1} = \theta'_n + b(n)I\{U_n = 0\} \times \Big(Q(X_n, 1; \theta_n) - r(X_n, 0)\\ + f(Q_n) - \max_{v\in\{0,1\}} Q(X_{n+1}, v; \theta_n) - \lambda(X_n; \theta'_n)\Big)\nabla_{\theta'}\lambda(X_n; \theta'_n)\end{aligned} \tag{35}$$

So now we have developed Two-Time Scale Optimization Algorithm, wherein we update Q-values according to Eq. (33) on a faster time scale and update Whittle Indices for all states using Eq. (35) on a relatively slower time scale. Please refer to Algorithm (2) for the detailed explanation of the implementation.

5 Experiments

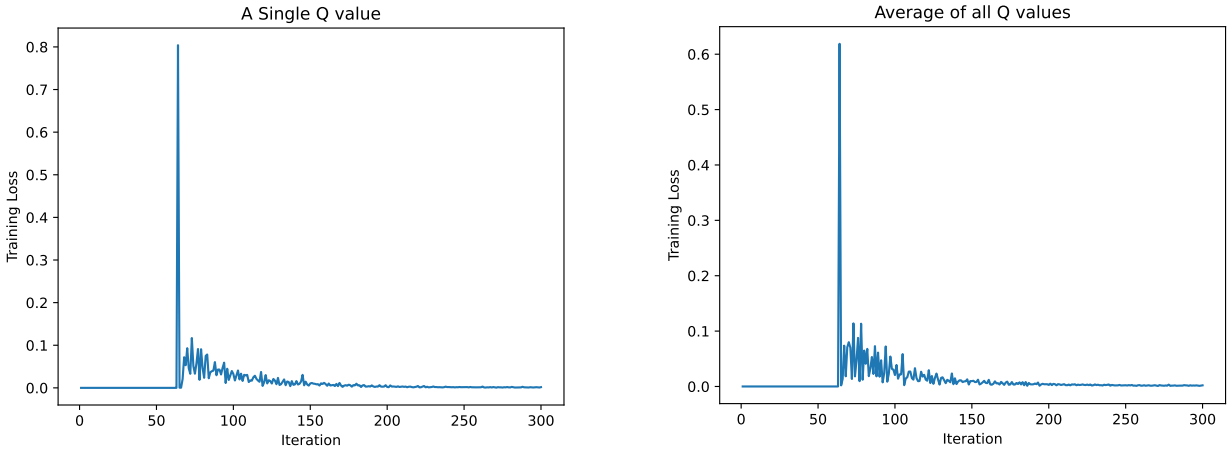
As a part of experiments I have integrated FGDQN with averaged reward described fully in Algorithm (1) in popular PyTorch Reinforcement Learning Libraries.
Following are some problems to test our average reward based FGDQN algorithm.

5.1 Forest Management

This is a simple MDP problem of Forest Management taken from [2]. The objective is to maintain an old forest for wildlife and make money by selling the cut wood.

The state of the forest at time n is $X_n \in \{0, 1, 2, 3, \cdots, M\}$ where each value of the state represents the age of the forest; 0 being the youngest and M being the oldest.
The forest is managed by two actions: 'Wait' and 'Cut'. If we apply the action 'Cut' at any state, the forest will return to its youngest age, i.e., state 0. On the other hand, when the action 'Wait' is applied, the forest will grow and move to the next state if no fire occurred. Otherwise, with probability p, the fire burns the forest after applying the 'Wait' action, leaving it at its youngest age (state 0).
Note that if the forest reaches its maximum age, it will remain there unless there is a fire or action 'Cut' is performed. Lastly, we only get a reward when the 'Cut' action is performed. In this case, the reward is equal to the age of the forest. There is no reward for the action 'Wait'.

For $p = 0.05$ and $M = 10$, the optimal policy obtained from policy iteration algorithm is [0, 1, 1, 1, 1, 1, 1, 1, 1, 1].
I have experimented for two variants of the reference value function f, one for fixed state-action pair of age 0 and action 'Wait' and other one is using average of all Q-values as shown in Fig. (3).
The Optimal policies obtained are:
Variant 1 : [0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Variant 2 : [0, 0, 1, 1, 1, 1, 1, 1, 1, 1] (1-bit error).



$f(Q) = Q(X_0, U_0)$ $\quad$ $f(Q) = \overline{Q(X_0, U_0)}$

Figure 3: Forest Management

5.2 Catcher

This is a game environment from [16] in which the aim of the agent is to catch the falling fruit with it's paddle.
Reward is +1 for successful fruit catch and -1 if the fruit is not caught.
The player is allowed to take only two actions of either moving right or left.
The states are non-visual representation of the game such as player's x position and velocity, fruits x and y position.
Following are the results, the agent learnt to catch the fruits pretty well but it hasn't mastered it very well. I hope to train it more longer in the hope of getting good results.

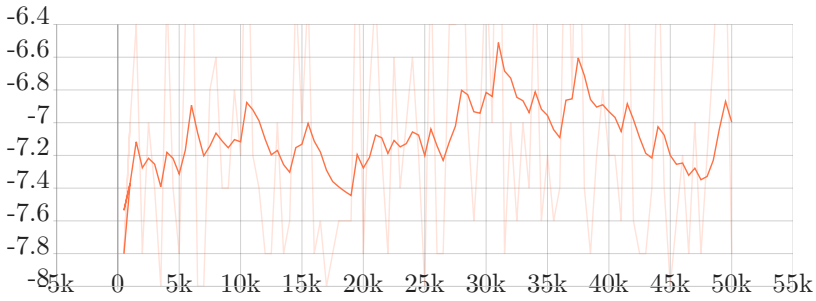


Figure 4: Reward

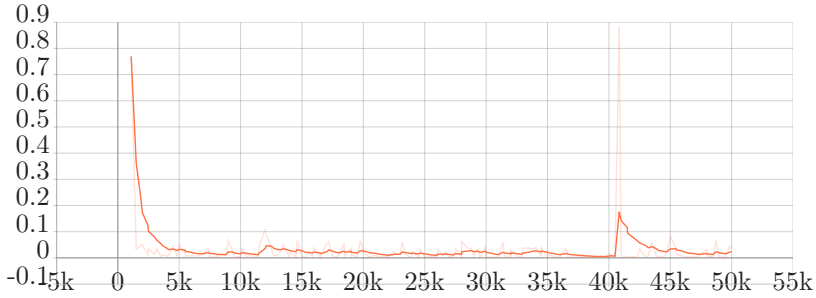


Figure 5: Training Loss

5.3 Whittle Indices

5.3.1 Circulant Dynamics

We consider a simple RMAB problem of Circulant Dynamics from [8], where we have I projects out of which we can do only K at a particular instant on priority basis. Each project has a underlying Markov chain for both Active ($u = 1$) and Passive action ($u = 0$). These two problems are taken from [3] as they serve as a basis for most of the other types of RMAB problems.

(a) Action Action (b) Passive Action

Figure 6: Underlying Markov chains of the Circulant Dynamics Problem

Consider the transition probability matrix $P_0 = \begin{bmatrix} 1/2 & 0 & 0 & 1/2 \\ 1/2 & 1/2 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 1/2 & 1/2 \end{bmatrix}$, and $P_1 = P_0^T$, for passive and active action respectively.

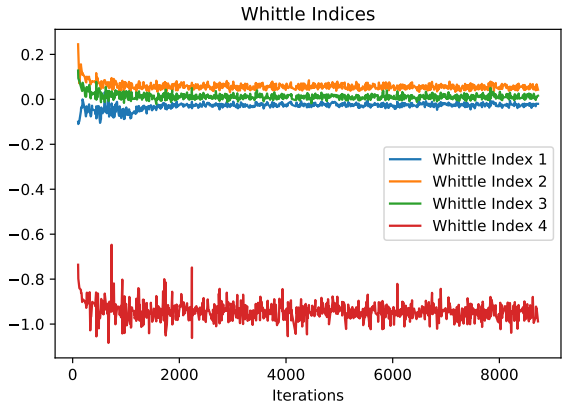
The rewards here do not depend on action and are given by $R(1,0) = R(1,1) = -1$, $R(2,0) = R(2,1) = 0$, $R(3,0) = R(3,1) = 0$, and $R(4,0) = R(4,1) = 1$. Where $R(s,u)$ denotes the reward in state s after taking action u. Intuitively, there is a preference to activate an arm when the arm is in state 3.

For experimentation, we consider a scenario with $N = 100$ arms, out of which $M = 20$ are active at each time. The exact whittle indices for this problem turns out to be $\lambda(1) = -1/2$, $\lambda(2) = 1/2$, $\lambda(3) = 1$, and $\lambda(4) = -1$, which give priority to state 3.

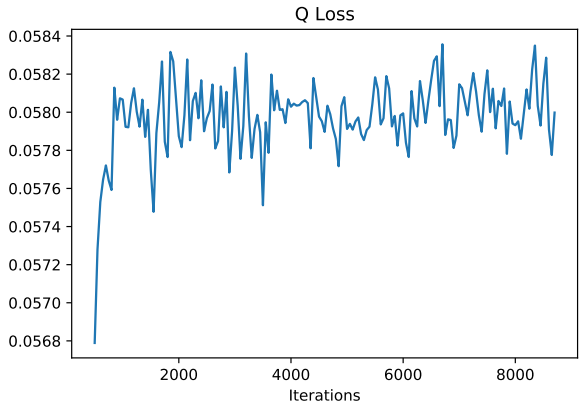
Here, we make two Neural Networks one for λ and the other for Q with parameters θ' and θ respectively. Please refer to 2 for more detailed explanation of the algorithm.

This problem is very stochastic in nature as at a particular state there is 1/2 probability for both actions to move to the other respective states. The order of the Whittle Indices settles pretty quickly as shown in Fig. 6c, but the order of the exact indices doesn't seem to match here. As it is clear the results show $\lambda(2) > \lambda(3)$ which should be the other

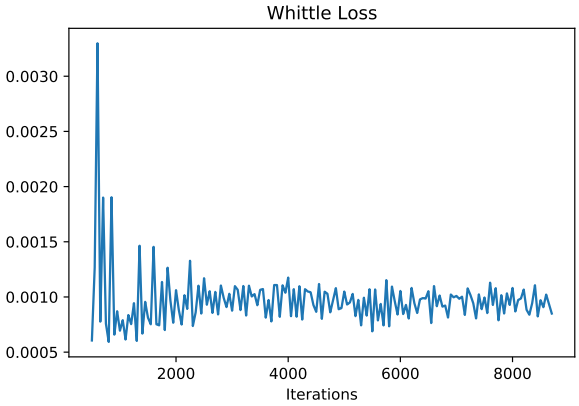
way around. One thing to note here, that the Q-loss was pretty high even after 10000 iteration, more hyperparameter tuning will surely give better results.



(c) Whittle Indices



(d) Q loss



(e) Whittle loss

Figure 6: Circulant Dynamics

5.3.2 Circulant Dynamics with Restart

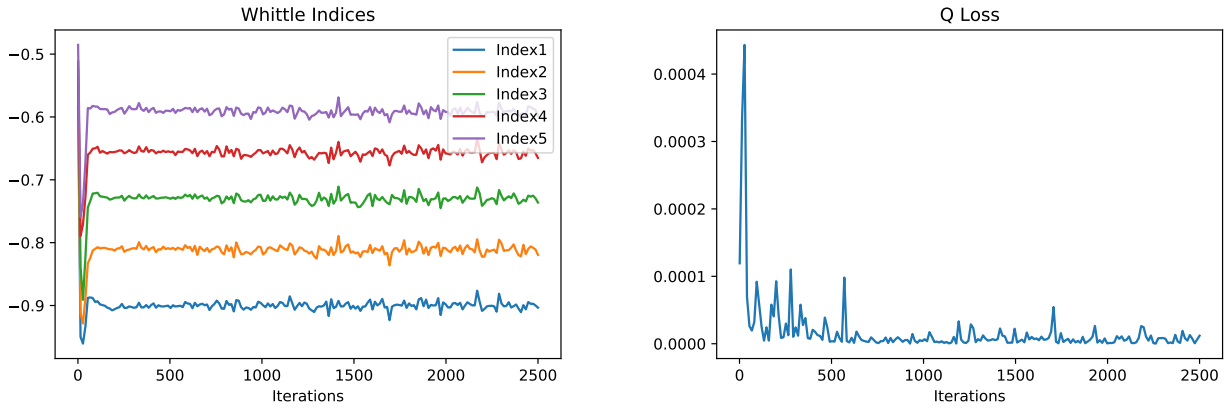
Now we consider an example where the active action forces an arm to restart from some state. We consider an example with 5 states, where in the passive mode ($u = 0$) an arm has tendency to go up the state space, i.e.,

$$P_0 = \begin{bmatrix} 1/10 & 9/10 & 0 & 0 & 0 \\ 1/10 & 0 & 9/10 & 0 & 0 \\ 1/10 & 0 & 0 & 9/10 & 0 \\ 1/10 & 0 & 0 & 0 & 9/10 \\ 1/10 & 0 & 0 & 0 & 9/10 \end{bmatrix},$$

whereas in the active mode ($u = 1$) the arm restarts from state 1 with probability 1, i.e.,

$$P_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The rewards in the passive mode are given by $R(k, 0) = \alpha^k$ (α is taken to be 0.9) and the rewards in the active mode are all zero.
We again take $N = 100$ and $M = 20$ for experimentation.
The results here are very good, as we can see in Fig. (7a) the order of the indices settles pretty quickly and both the Q and Whittle loss approaches 0 in about 500 iterations.



(a) Whittle Indices

(b) Q loss

Figure 7: Circulant Dynamics with Restart

5.4 More Experiments

Apart from this, I have experimented on some games like Catch, Tic-Tac-Toe, Deep Sea from OpenSpiel [10], these games are specifically made to test various aspect of an algorithm. I further tested on a Toy Autonomous Driving Environment [11], for problems like Intersection, Racetrack. I want to give a shoutout to the amazing library Stable-Baselines3 [15] which made my implementation easier and faster. I refer reader to go through the codebase and supplementary materials to find more about their implementation and to try the code yourselves.

6 Acknowledgments

I want to sincerely thank Prof. Vivek Borkar for his guidance throughout the course of this project work. He is one of the patient and dedicated Professors I have ever met. His dedication toward research motivates me alot. I also want to thank Dr. Kishor Patil for his valuable discussion and Dr. Konstantin Avrachenkov for his valuable insights.
Lastly, I'm very grateful to my parents, my sister and friends and I consider myself very lucky to have them.

7 Appendix

Algorithm 1: Full Gradient DQN (FG-DQN) for Average Reward

Input: replay memory $\mathcal{D}$ of size M, minibatch size B, number of episodes N, maximal length of an episode T, exploration probability ϵ.

Initialise the weights θ randomly for the Q-Network.
Receive initial observation s_1.
for *Episode = 1 to N* **do**
  for $n = 1$ *to* T **do**
    if *Uni[0,1]* $< \epsilon$ **then**
      Select action U_n at random.
    else
      $U_n = Argmax_u Q(X_n, u; \theta)$
    end
    Execute the action and take a step in the RL environment.
    Observe the reward R_n and obtain next state X_{n+1}.
    Store the tuple (X_n, U_n, R_n, X_{n+1}) in $\mathcal{D}$.
    Sample random minibatch of B tuples from $\mathcal{D}$. **for** $k = 1$ *to* B **do**
      Sample all tuples (X_j, U_j, R_j, X_{j+1}) with a fix state-action pair $(X_j = X_k, U_j = U_k)$ from $\mathcal{D}$

$$Set\, Z_j = \begin{cases} R_j, & \text{for terminal state,} \\ R_j + \max_u Q(X_{j+1}, u; \theta) - Q(x_0, y_0; \theta) & \text{otherwise.} \end{cases}$$

      Compute gradients and using Eq. (23) update parameters θ.
    end
  end
end

Algorithm 2: Whittle Indices with FGDQN

Input: replay memory $\mathcal{D}$ of size M, minibatch size B for Q iteration and C for λ iteration, number of episodes N, maximal length of an episode T, discount factor γ, exploration probability ϵ, whittle index λ.

Initialise the weights θ & θ' randomly for the Q-Network and Whittle-Network.

Consider RMABP with I projects such that at every time step K of them are active.

Denote state of the system at time n as $S(n) = (s_1(n), \cdots, s_i(n), \cdots, s_I(n))$ where $s_i(n)$ is a state of project $i \in \{1, 2, \cdots, I\}$

for $n = 1$ *to* T **do**

  for $s = 1$ *to* d **do**

    Select actions $U(n) = (u_1(n), \cdots, u_i(n), \cdots, u_I(n)))$ at random such that $\sum_{i=1}^{I} u_i(n) = K$.

    Execute actions and take a step.

    Observe the rewards $R(n) = (r_1(n), \cdots, r_I(n))$ and obtain next state of the system $S(n+1)$.

    Store all the tuples $(S(n), U(n), R(n), S(n+1))$ in $\mathcal{D}$.

    Sample random minibatch of B tuples from $\mathcal{D}$.

    for $a = \{0, 1\}$ **do**

      Sample all tuples (X_j, U_j, R_j, X_{j+1}) of size B with a fix state-action pair $(X_j = s, U_j = a)$ from $\mathcal{D}$

      for $\hat{k} = 1$ *to* d **do**

        $$Set\ Z_j = (1 - U_j)(R_j + \lambda(\hat{k})) + U_j R_j + \max_{v} Q(X_{j+1}, v; \theta, \hat{k}) - f(Q(\hat{k}; \theta))$$

        Compute gradients and using Eq. (33) update parameters θ.

      end

    end

  end

  On slower time-scale **do**

  Sample all tuples (X_k, U_k, R_k, X_{k+1}) of size B with fix action $U_k = 0$ from $\mathcal{D}$

  $$Set\ Z_k = Q(X_k, 1; \theta_n) - r(X_k, 0) + f(Q(X_n, \theta_n)) - \max_{v \in \{0,1\}} Q(X_{k+1}, v; \theta_n)$$

  Compute gradients and using Eq. (35) update parameters θ'.

end

7.1 Implementation Details

The below snippet shows how the *mse_loss* of PyTorch [14] works which allowed us to implement our algorithms in a very elegant way.
Following are the equations corresponding to *loss.backward*() in Pytorch.

$$
\begin{aligned}
mse\_loss(pred, target) \\
loss = (pred - target)^2 \\
\nabla_\theta loss = 2(pred - target)(\nabla_\theta pred - \nabla_\theta target) \\
\theta_n \leftarrow \theta_{n-1} - \gamma \nabla_\theta f(\theta_{n-1}) \\
\theta_n \leftarrow \theta_{n-1} - \gamma 2(pred - target)(\nabla_\theta pred - \nabla_\theta target) \\
\theta_n \leftarrow \theta_{n-1} - \gamma 2(target - pred)(\nabla_\theta target - \nabla_\theta pred) \\
\theta_n \leftarrow \theta_{n-1} + \gamma 2(target - pred)(\nabla_\theta pred - \nabla_\theta target)
\end{aligned}
$$

Details:

$$
\theta_n \leftarrow \theta_{n-1} - \gamma 2(target - pred)(\nabla_\theta target - \nabla_\theta pred) \tag{36}
$$

$$
\begin{aligned}
\theta_{n+1} = \theta_n - a(n) \Big(r(X_n, U_n) + \max_v Q(X_{n+1}, v; \theta_n) - f(Q; \theta_n) - Q(X_n, U_n; \theta_n) \Big) \times \\
\Big(\nabla_\theta Q(X_{n+1}, v_n; \theta_n) - \nabla_\theta f(Q; \theta_n) - \nabla_\theta Q(X_n, U_n; \theta_n) \Big)
\end{aligned} \tag{37}
$$

where $v_n \in \text{argmax} Q(X_{n+1}, \cdot, \theta_n)$

Above two equations Eq. (36) & Eq. (37) resembles, where
target is the average reward Q target and *pred* is predicted Q value from the Q network.

$$
\begin{aligned}
target = r(X_n, U_n) + \max_v Q(X_{n+1}, v; \theta_n) - f(Q; \theta_n) \\
pred = Q(X_n, U_n; \theta_n)
\end{aligned}
$$

7.1.1 New Implementation

For simplicity assume the fixed state action pair as (X, U).
Now, from the replay buffer sample all the transitions with this fixed state action pair as (X, U). Let's say we get $(X, U, R_1, X_1'), (X, U, R_2, X_2')$ and (X, U, R_3, X_3') and hence we have a batch size B of 3.

Let's denote

$$target_n = R_n + \max_v Q(X_n', v; \theta_n) - f(Q; \theta)$$
$$pred_n = Q(X, U; \theta)$$

Now, we compute $diff = \dfrac{(target_1 - pred_1) + (target_2 - pred_2) + (target_3 - pred_3)}{3}$
using $torch.mean(target\_batch - pred\_batch)$.
This is essentially our term under the bar in the FGDQN equation.

Now we compute it's gradient to get,
$\nabla_\theta diff = \dfrac{(\nabla_\theta target_1 - \nabla_\theta pred_1) + (\nabla_\theta target_2 - \nabla_\theta pred_2) + (\nabla_\theta target_3 - \nabla_\theta pred_3)}{3}$.

Now calculate $diff \times \nabla_\theta diff$, to get
$final\_grad = \dfrac{diff \times (\nabla_\theta target_1 - \nabla_\theta pred_1) + diff \times (\nabla_\theta target_2 - \nabla_\theta pred_2) + diff \times (\nabla_\theta target_3 - \nabla_\theta pred_3)}{3}$.

$final\_grad$ is essentially the term we wanted for the FGDQN iteration. Now we simply do,

$$\theta' \leftarrow \theta - \gamma(final\_grad)$$

7.1.2 Mistake in previous Implementation

In my previous implementation I simply calculated the gradient of

$loss = \dfrac{(target_1 - pred_1)^2 + (target_2 - pred_2)^2 + (target_3 - pred_3)^2}{3}$ which is
$\nabla_\theta loss = \dfrac{2(target_1 - pred_1)(\nabla_\theta target_1 - \nabla_\theta pred_1) + 2(target_2 - pred_2)(\nabla_\theta target_2 - \nabla_\theta pred_2) + \cdots}{3}$
and then did

$$\theta' \leftarrow \theta - \gamma(\nabla_\theta loss)$$

This doesn't resemble the FGDQN iteration, since here we are not multiplying the average (the term with the overline in iteration) with each gradient.

Consider parametrized families $\lambda(k,\sigma)$ and $Q(i,u;\theta,\sigma)$ where we render explicit the implicit dependence of Q on λ and therefore σ. Consider

$$\theta_{n+1} = \theta_n - a(n)\Big(\nabla_\theta Q(X_{n+1},v_n;\theta_n,\sigma_n) - \nabla_\theta f(Q(\theta,\sigma_n))\Big|_{\theta=\theta_n} \tag{38}$$

$$- \nabla_\theta Q(X_n,U_n;\theta_n,\sigma_n)\Big) \times \tag{39}$$

$$\overline{\Big((1-U_n)(r(X_n,0)+\lambda(X_n,\sigma_n)) + U_n r_n(X_n,1) + \max_{v\in\{0,1\}} Q(X_{n+1},v;\theta_n,\sigma_n)} \tag{40}$$

$$\overline{- f(Q(\theta_n,\sigma_n)) - Q(X_n,U_n;\theta_n,\sigma_n)\Big)} + a(n)\xi_{n+1}, \tag{41}$$

$$\sigma_{n+1} = \sigma_n - b(n)\overline{\Big(Q(X_n,1;\theta_n,\sigma_n) - r(X_n,0) + f(Q(X_n,0;\theta_n,\sigma_n))} \tag{42}$$

$$\overline{- \max_{v\in\{0,1\}} Q(X_{n+1},v;\theta_n,\sigma_n) - \lambda(X_n,\sigma_n)\Big)} \times \tag{43}$$

$$\Big(\nabla_\sigma Q(X_{n+1},v_n;\theta_n,\sigma_n) - \nabla_\sigma f(Q(\theta_n,\sigma))\Big|_{\sigma=\sigma_n} \tag{44}$$

$$- \nabla_\sigma Q(X_n,U_n;\theta_n,\sigma_n) - \nabla_\sigma \lambda(X_n,\sigma_n)\Big)$$

The θ_n iteration is the SGD for the mean square error

$$\mathcal{E}_1 := E\Big[\Big\|(1-U_n)(r(X_n,0)+\lambda(X_n,\sigma_n)) + U_n r_n(X_n,1) \tag{45}$$

$$+ \max_{v\in\{0,1\}} Q(X_{n+1},v;\theta_n,\sigma_n) - f(Q(\theta_n,\sigma_n)) - Q(X_n,U_n;\theta_n,\sigma_n)\Big\|^2\Big].$$

The σ_n iteration is the SGD to minimize the mean square error

$$\mathcal{E}_2 := E\Big[\Big\|Q(X_n,1;\theta_n,\sigma_n) - r(X_n,0) + f(Q(X_n,0;\theta_n,\sigma_n)) \tag{46}$$

$$- \max_{v\in\{0,1\}} Q(X_{n+1},v;\theta_n,\sigma_n) - \lambda(X_n,\sigma_n)\Big\|^2\Big].$$

References

[1] Jinane Abounadi, Dimitri P. Bertsekas, and Vivek S. Borkar. Learning algorithms for markov decision processes with average cost. *SIAM J. Control. Optim.*, 40(3):681–698, 2001.

[2] K. E. Avrachenkov, V. S. Borkar, H. P. Dolhare, and K. Patil. Full gradient dqn reinforcement learning: A provably convergent scheme, 2021.

[3] Konstantin E. Avrachenkov and Vivek S. Borkar. Whittle index based q-learning for restless bandits with average reward, 2021.

[4] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. 1.* Athena Scientific, 3rd edition, 2005.

[5] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. II.* Athena Scientific, 4th edition, 2012.

[6] Vivek S. Borkar. *Stochastic Approximation A Dynamical Systems Viewpoint.* Hindustan Book Agency, Gurgaon, 1st edition, 2008.

[7] Vektor Dewanto, George Dunn, Ali Eshragh, Marcus Gallagher, and Fred Roosta. Average-reward model-free reinforcement learning: a systematic review and literature mapping. *CoRR*, abs/2010.08920, 2020.

[8] Jing Fu, Yoni Nazarathy, Sarat Moka, and Peter G. Taylor. Towards q-learning the whittle index for restless bandits. In *2019 Australian New Zealand Control Conference (ANZCC)*, pages 249–254, 2019.

[9] Robert Gallager. Finite-State Markov Chains MIT OCW Chapter 3, 2011.

[10] Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinícius Flores Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas W. Anthony, Edward Hughes, Ivo Danihelka, and Jonah Ryan-Davis. Openspiel: A framework for reinforcement learning in games. *CoRR*, abs/1908.09453, 2019.

[11] Edouard Leurent. An environment for autonomous driving decision-making. `https://github.com/eleurent/highway-env`, 2018.

[12] Sridhar Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, 22:159–195, 2005.

[13] 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.

[14] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[15] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2019.

[16] Norman Tasfi. Pygame learning environment. `https://github.com/ntasfi/PyGame-Learning-Environment`, 2016.

[17] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[18] P. Whittle. Restless bandits: Activity allocation in a changing world. *Journal of Applied Probability*, 25:287–298, 1988.