Chapter 4 - Balancing the Gathering and Use of Information

→ This chapter is about learning from evaluative feedback.

What is evaluative feedback? (Recap)

→ It is a summary for the learner of how well he or she has performed on a particular task. In terms of RL, the reward signal provides evaluative feedback to the agent.

→ In general, every decision made by an intelligent entity is a trade-off between exploration and exploitation.

→ It is important for the agent to "correctly" decide when to gather more knowledge from the environment and when to take an action based on previously learned knowledge.

$$\boxed{\text{Exploration builds the knowledge that allows for effective exploitation, and maximum exploitation is the goal of any decision maker.}}$$

Def^n:

Horizon: The number of time steps that the agent interacts with the environment in each episode.

Greedy Horizon: A special task with horizon 1, i.e each episode is exactly 1 time step.

Multi-Armed Bandits: (MAB's)

→ Special kind of RL problem in which the environment's MDP has a single state, multiple actions and each episode only lasts 1 time step (in other words MAB's have a greedy horizon).

→ Ex: Youtube trying to show you an ad, e-commerce websites recommending products.

→ MAB's highlight the exploration - exploitation tradeoff so their study can help agents in more complex environ-ments also.

Setup

→ The goal of our task is to maximise the expected total reward in some 'n' time steps, (In other words learn the P.d of reward signal as accurately as possible)

→ Each action has an expected reward. Call this the value of the action. (Keep in mind that the agent does not know the value of each action)

→ From past experience, the agent may have decent estimates of the value of each action. In a given episode, if the agent selects the action with highest reward (by its estimate), then this action is "greedy".

→ Greedy actions correspond to exploiting and on the other hand if the agent chooses an action with lower reward estimate, then in that time step, it is exploring.

→ Exploitation is the right thing to do ~~to~~ maximise reward in a single time step, but exploration might yield higher returns in the long run.

Recall the action-value function :

$$q_\pi(s,a) = \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right]$$

In the MAB setting, as there is only a single state,

$$q(a) = \mathbb{E}\left[R_t \mid A_t = a\right]$$

→ Here, V_* (optimal state-value function) will represent the highest expected reward we can get in a single time step :

$$\boxed{V_*^{\phi} = q(a_*)}, \text{ where } \boxed{a_* = \underset{a \in A}{\arg\max}\; q(a).}$$

Regret :

→ We want to tweak the statement of our goal which is "maximize the expected reward".

→ Why? Agent A may find the optimal action in the final episode after many episodes of random actions, while another Agent B might be much more efficient and find the optimal action much quicker than A. If our goal was "maximize the expected reward", both A and B would be thought to have achieved the goal, but obviously we want agent B.

So, a more robust way to capture a more complete goal is for the agent to maximize the per-episode expected reward, while still mimimizing the "total expected reward loss of rewards across all episodes".

→ We will call this quantity **total regret**, and to calculate it we sum the per-episode difference of the true expected reward of the optimal action and the true expected reward of the selected action.

→ True expected reward, requires access to the environment's MDP (the action value function q in MAB's) to be calculated.

(Total Regret will only be calculated to compare the behaviour of 2 agents)

$$\text{Total Regret } T = \sum_{e=1}^{E} \mathbb{E}\left[v_* - q(A_e)\right]$$

Approaches to Solving MAB's

3 major Kinds :

1) Random Exploration Strategies - Agent will mostly exploit but randomly explore.

2) Optimistic Exploration Strategies Quantifies uncertainty, and increases preference for states with highest uncertainty.

3) Information state-space exploration Strategies: Model the information state of the agent into the environment, i.e tries to encode the uncertainty of the agent into the ~~environment~~ state space. Risks large state spaces and high complexity.

→ Note that in any strategy, the estimation of the action-value function q, is done in the same way:

If action a is performed in k time steps/episodes and yields reward $Q_i(a)$, then our estimate of $q(a)$ will be

$$q(a) \approx \sum_{i=1}^{K} Q_i(a) / k$$

→ So strategies differ in how the estimate is used, not how the estimation is performed.

Two example Strategies:

1) Greedy (Always Exploit) :

→ Always select action with highest estimated Q value.

→ There is some chance, that the action we select first is optimal, but this chance decreases considerably as action space grows larger.

→ In an environment with Q-values (estimates) initialized to zero, and no negative rewards, our agent will get stuck with the first action it chooses.

→ An agent that is always exploiting, will often fail to find the optimal action, however, in general, if an agent only has a limited number of 'training' episodes left, greedy strategy is effective.

→ The agent is 'short-sighted' if it is always exploiting as it does not trade-in immediate satisfaction for information gain that could lead to higher future rewards.

2) Random (Always Explore):

→ Never exploits and sole goal of agent is to gain information.

→ This strategy is also very bad, and in case the agent does 'explore' the optimal action, it ~~will~~ might never come back to it as it is always exploring.

→ We want algorithms to balance both exploration and exploitation.

→ Note that random exploration is just one way to explore various actions. Other pure exploration strategies can also be designed.

Some viable Strategies:

1) ε-Greedy (Almost always greedy and sometimes random)

→ Select action agent thinks is best almost all the time, but randomly explore other actions as well.

→ Agent's estimate of action-value function has an opportunity to converge to q (true action-value function)

Sample ε - greedy Algo: (Q is lookup table with q-value estimates)

```
ε-greedy (MAB, epsilon = e, episodes = n) :   (0 < e < 1)

• Sample a random number in [0,1)   (we sample from uniform distribution on [0,1])
    if rand > e :
        # exploit
        action = argmax (Q)
    else:
        # explore
        sample a random integer in len Q (randint)
        action = randint
```

Remarks: → This algo has randomness at both stages:

1) Choosing between exploitation and exploration
2) Exploration.

→ Probability of exploration is exactly e and exploitation is $1-e$.

→ Note that in the exploration stage, greedy action may also be picked, so in reality, exploration probability ~~might be~~ is even less than 'e'.

2) Decaying ε Greedy: (First maximize exploration, then exploit

→ Explore more initially when agent hasn't experienced the environment much.

→ As value function estimates improve, exploit more.

→ Simple way to do this: Start with some $0 < \varepsilon < 1$ and decay its value at every step.

Sample algo: Linear decaying ε-greedy.

```
lindec_eps_greedy (MAB, $\varepsilon_i$, $\varepsilon_m$, $\gamma$, episodes = n):
($\varepsilon_i$ is initial $\varepsilon$, $\varepsilon_m$ is minimum $\varepsilon$, $\gamma$ is decay ratio)

    for i in range (n):
        decay_episodes = episodes * $\gamma$
        # Calculate $\varepsilon$ value for current episode
        $\varepsilon$ = 1 - i/decay-episodes
        $\varepsilon$ *= $\varepsilon_i - \varepsilon_m$
        $\varepsilon$ += $\varepsilon_m$

    \\ Now generate random number and proceed as in $\varepsilon$ greedy.
```

Exponential ε decay can also be executed insted of linear.

→ In general, many ways to execute ε decay, but all of them should favour exploration in the beginning.

3) Optimistic Initialization

→ Treats actions that were not sufficiently explored as the best actions.

→ Initialize agent's Q function estimate to a high value and then start acting greedily, using these estimates.

(usually good high value is not available, but assume that for now we do)

→ Counts of actions performed can act as a confidence measure in the agent's estimate of the action's q value. So, initializing counts to high values is also necessary, otherwise Q function will change too quickly.

→ Agent initially expects high rewards, but as experience is gained, Q value estimates drop and start to converge.

→ Algo is straight forward : Initialization + pure exploitation.

More advanced Strategies:

1) **Softmax** (selecting actions randomly in proportion to their estimates)

→ It is a random exploration strategy that takes into account estimated Q values to make a decision.

→ If an action has a low estimate, agent will try it with less likelihood.

Softmax Strategy does ~~[illegible]~~ exactly this:

→ Probability of selecting and action $\propto$ current Q value estimate of that action.

→ Hyperparameter, τ (called temperature) can also be added to control the algo's sensitivity to Q value estimates.

→ τ can also be decayed.

→ let π denote the policy, i.e returns probability of an action being taken. π depends on current Q ~~[illegible]~~ function estimate:

$$\pi(a) = \frac{e^{(Q(a)/\tau)}}{\sum_{i=1}^{K} e^{(Q(a_i)/\tau)}}$$

→ As $\tau \uparrow$, π resembles uniform distribution.

UCB:

First, some issues with optimistic initialization:

→ we don't know maximum reward agent can obtain, so setting initial values ~~[illegible]~~ much higher than this would lead to huge training times.

→ If initial values are much smaller than actual max reward, then agent will probably never discover optimal action.

→ Counts were also initialized, but we want to be able to use counts to quantify uncertainty.

→ The Upper Confidence Bound Strategy (UCB) is an improvement, in somesense, to optimistic initialization.

→ In UCB, uncertainty of value estimates is taken into consideration.

$$\underbrace{A_e}_{\text{action to be taken at episode e}} = \operatorname*{argmax}_a \left[\underbrace{Q_e(a)}_{\text{Q value estimate}} + \overbrace{c\sqrt{\frac{\ln e}{\underbrace{N_e(a)}_{\text{\# of times a was performed until episode 'e'.}}}}}^{\text{Uncertainty Bonus}} \right]$$

3) Thompson Sampling

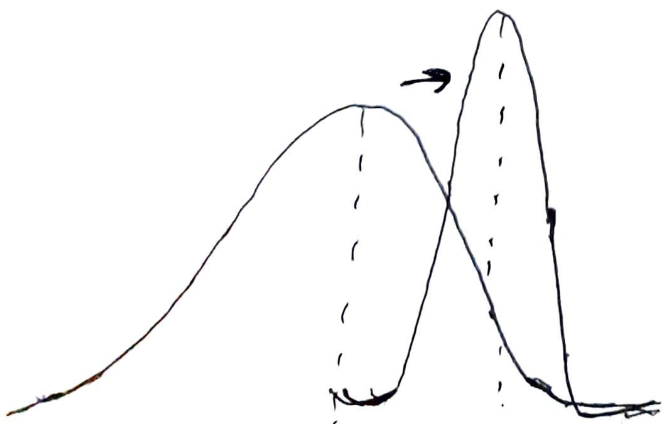
→ Keep track of each Q-value estimate as a Gaussian distribution (or any other suitable distribution).

→ In case of Gaussian distribution, there are 2 parameters μ and σ. μ will be the current Q value estimate, while s.d σ will measure the uncertainty of the ~~episode~~ estimate.

→ μ and σ are updated on each episode.

→ Whenever action needs to be taken, sample from these distributions and pick the action with highest sample.

→ As number of episodes ↑, means are supposed to get closer and closer to actual q value and σ's should drop.



→ Progression of the distribution of some action as episodes increase.