Online and Reinforcement Learning (2025)
Home Assignment 1

Jan Ljubas, PMJ557

Contents

1 Find an online learning problem from real life

1.1 Example 1: Quantum Communication with an Eavesdropper

- **Setting:**
  - Alice (A) wants to send a classical or quantum message to Bob (B) over a quantum channel, but Eve (E, the adversary) can introduce noise into the channel, possibly in an adversarial manner.
  - A and B can apply different strategies to mitigate E's interference.
  - The goal is to minimize the error rate in the received message.
- **Set of Actions:** Selecting encoding-decoding and error correction strategies for sending the quantum bits through a communication channel.
- **Losses/Rewards:** Loss can be the error rate in the received messages. Alternatively, the reward can be based on quantum fidelity measure - how well the received quantum state matches the sent quantum state. Both in the range $[0, 1]$.
- **Stateless or Contextual:** Could be both stateless and contextual, but more likely stateless. The reason is that A and B don't have a reason to think that E's adversarial future behavior should be predicted based on previous behavior.
- **IID or Adversarial:** If E's behavior follows a distribution or if there exist a set of statistically optimal strategies depending on the type of message sent (number of quantum bits, types of quantum gates for generating the states, etc.) - then IID.
  If E is actively trying to come up with a better way of intercepting and decoding the messages with high fidelity (usual case), then it is adversarial.
- **Feedback:** Bandit - B receives the potentially corrupt messages, without knowledge if it's been intercepted and modified by E.

1.2 Example 2: Stock Price Prediction

- **Set of Actions:** Buy, sell or hold the stock.
- **Losses/Rewards:** Profit/loss based on price movement; so Δ of the price.
- **Stateless or Contextual:** Contextual; stock price depends on past prices and trends.
- **IID or Adversarial:** Probably hybrid. "Semi-adversarial" because of market fluctuations which add another layer of stochasticity for predicting the stock price.
- **Feedback:** Full (we can observe profit/regret of action for whatever stock there is).

2 Follow The Leader (FTL) algorithm for i.i.d. full information games

The following "proof" follows a standard approach of using concentration inequalities (Hoeffding's inequality, more precisely) to bound the probability that a suboptimal action is played: $\mathbb{P}[\hat{\mu}_{t-1}(a) \geq \hat{\mu}_{t-1}(a^*)]$. Then we just sum over all rounds to bound the total regret. So, we start with:

$$\hat{\mu}_{t-1}(a) \geq \hat{\mu}_{t-1}(a^*) \tag{1}$$

Using Hoeffding's concentration of measure inequality:

$$\mathbb{P}[\hat{\mu}_{t-1}(a) \geq \hat{\mu}_{t-1}(a^*)] \leq \exp(-t\Delta(a)^2/2) \tag{2}$$

Note that instead of working with the more complicated expression involving $\max_{a'} \hat{\mu}_{t-1}(a')$ we can simply bound it using just $a*$, which is easier to handle. This was hinted in the instructions.

Using the relation of this bound with number of times a is played $N_T(a)$, we can write:

$$N_T(a) \leq \frac{c}{1 - \exp(-\Delta(a)^2/2)} \tag{3}$$

Now we can bound the R_T using $N_T(a)$, since we know this, too: $\bar{R_T} = \sum_a \Delta(a)\, \mathbb{E}[N_t(a)]$.

So, by substituting for $N_t(a)$, we have this bound on regret after summing over all suboptimal actions:

$$\bar{R_T} \leq \sum_{a:\Delta(a)>0} \frac{c}{1 - \exp(-\Delta(a)^2/2)} \Delta(a) \tag{4}$$

Thus, the regret does not grow with T. This is a crucial result because it means that in the i.i.d. setting, FTL achieves regret as a function of number of actions, meaning that it does not grow with time.

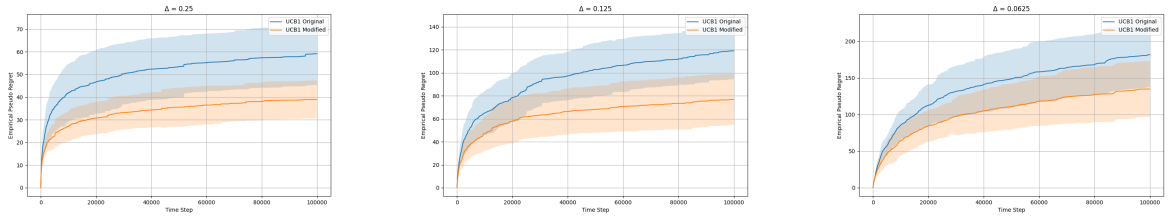
3 Improved Parametrization of UCB1

Improved algorithm:

1. Play the action with the highest empirical mean.
2. Randomly observe the reward of another arm.
3. Update empirical means using both observations.

The code is available in the zipped folder, under file `3.py`

3.1 Results



(a) $\Delta = 0.25$ (b) $\Delta = 0.125$ (c) $\Delta = 0.0625$

Figure 1: Comparison of the two algorithm versions for different Δ values.

3.2 Analysis

Q1: Which values of Δ lead to higher regret?
A1: In the experiments I've done, **smaller deltas lead to higher regret** after T steps. Maybe one way to intuitively explain it is that smaller deltas make it harder to determine which arm is optimal - hence the exploration time grows and regret piles up.

Q2: What can you say about the relative performance of the two parametrizations?
A2: It seems from the diagrams that the **improved UCB1 algorithm (orange) does constantly outperform the original UCB1 (blue)**.
I didn't know what to expect beforehand because on one hand, the improved algorithm does tighten the confidence bounds, meaning that it explores less - so I wouldn't be surprised if that lead to higher regret over time. But perhaps what happens is that tighter confidence bounds reduce over-exploration once the optimal arm is "identified".

4 Example of Policies in RiverSwim

4.1 Policy Classification

1. π_a - Swim to the right if the current state is not 1; otherwise swim to the left.
   (a) Class: Π^{SD} (Stationary Deterministic)
   (b) Explanation: The action depends deterministically on the current state (right if not state 1, left otherwise). No history or randomness.
2. π_b - If time slot t is an even integer, swim to the right; otherwise, flip a fair coin, then swim to right (resp. left) if the outcome is 'Head' (resp. 'Tail')
   (a) Class: Π^{HR} (History-Dependent Randomized).
   (b) Explanation: The policy uses the time step t (history) and includes a stochastic coin flip.
3. π_c - At t = 1, swim to the left. For t ¿ 1, swim to the right if the index of the previous state is odd; otherwise swim to the left. (For t = 1, swim to the left.)
   (a) Class: Π^{HD} (History-Dependent Deterministic).
   (b) Explanation: The action depends on the parity of the previous state's index, which requires history (previous state).
4. π_d - Flip a fair coin. If the outcome is 'Head' and the current state is either L-1 or L, then swim to the right; otherwise swim to the left.
   (a) Class: Π^{SR} (Stationary Randomized).
   (b) Explanation: The action depends on a coin flip and the current state, with fixed probabilities (no history dependence).
5. π_e - If it rains, swim to the right; otherwise, swim to the left.
   (a) Class: Π^{SR}(Stationary Randomized).
   (b) Explanation: The action depends on an external random event (rain), which is independent of the agent's state or history but results in a fixed probability distribution over actions.

4.2 History-Dependent Deterministic Policy

Policy: "Swim left until visiting state $L-1$, then always swim twice right $(L+1)$, and from there on continue to swim once left (L), once right$(L+1)$"
Class: Π^{HD}
Explanation: The action depends on whether state $L-1$ has been visited before (history), after which it enters non-stochastic loop of actions, making it non-stationary and fully deterministic.

5 Robbing Banks

5.1 State Space and Action Space

The MDP state is defined as:

$$s = (p_{\text{agent}}, p_{\text{police}}) \in \mathcal{S}$$

where:

- $p_{\text{agent}} = (x_a, y_a)$ represents the agent's grid coordinates
- $p_{\text{police}} = (x_p, y_p)$ represents the police's grid coordinates

The action space consists of:

$$\mathcal{A} = \{\text{up}, \text{down}, \text{left}, \text{right}, \text{stay}\}$$

5.2 Reward Function

The reward function is defined as:

$$R(s,a) = \begin{cases} 100,000 & \text{if } p_{\text{agent}} \in \mathcal{B} \wedge p_{\text{agent}} \neq p_{\text{police}} \\ -10,000 & \text{if } p_{\text{agent}} = p_{\text{police}} \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{B}$ denotes the set of bank positions.

5.3 Transition Probabilities

When $p_{\text{agent}} = B_1$ and $p_{\text{police}} = B_4$:

1. **Agent Transition:** Deterministic based on action (with wall clipping)
2. **Police Transition:** Since positions are neither aligned nor adjacent:
$$P(p'_{\text{police}}|p_{\text{police}}) = \frac{1}{4} \text{ for each direction (up, down, left, right)}$$
3. **Collision Check:** If $p'_{\text{agent}} = p'_{\text{police}}$, transition to initial state with $p_{\text{agent}} = B_1$, $p_{\text{police}} = PS$

The complete transition function is:

$$P(s'|s,a) = \begin{cases} 1 & \text{if collision occurs (reset)} \\ \frac{1}{4} & \text{for each possible police move} \\ 0 & \text{otherwise} \end{cases}$$

5.4 Bonus - Key Implementation Details

This refers to the `8.py` file.

- `_move()` handles boundary conditions with clip()
- Police movement rules implemented in `_police_movement()`
- Reset condition checked after each step
- State tracking maintains both agent and police positions