
DTU

Mathematical formulation supporting the work done on the Multi-Armed Bandit problem

Objective of this presentation

- Here we only cover the mathematical formulation of the Multi-Armed Bandit work done for Informs
- On the GitHub you find two other presentations that cover the motivation, purpose for P2P markets, examples and results:
  - ELMA Presentation
  - Danske Bank Presentation
- Please read them before this one
- You also find more information in the *README.md* file on the GitHub repo

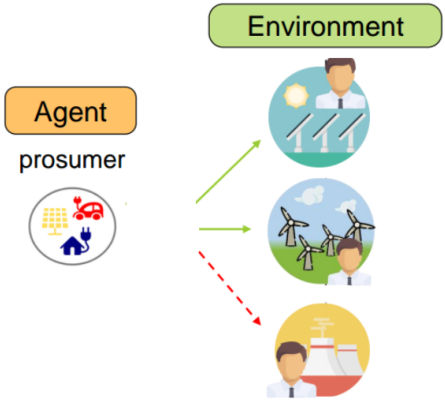
Problem statement

The Reinforcement Learning (RL) framework for our P2P market:

- Prosumer is the Agent
- Market with the participants is the Environment

Assumptions adopted:

- Each time step t of the P2P market is an episode e
- Each episode e is defined by number of steps $n = \{1, ..., n_{\max}\}$. We have considered $n_{\max} = 40$



Agent action

Keep in mind, we will describe the formulation for a single episode e. In more detail, we consider a $j-$partner selection, in which only one partner can be selected to trade energy at any step n:

Notation

- Set of partners $j \in \mathcal{J} = \{1, 2, \ldots, J\}$, and $\mathcal{J}$ is kept the same for all episodes e
- Our action vector is $a_n = j(n)$, where $j(n)$ denotes the partner j selected at step n

For $\mathcal{J} = \{1, 2, 3\}$ we could have action $a_n = \{1, 2, 1, 3, 1, 2\}$. Although only one partner j is selected, the same partner j can be selected at different steps n.

Reward function I

Upon selecting partner j as action a_n, the agent receives a reward that is a random variable modeled by a Bernoulli distribution.

Non-negative reward R_n

$$R_n(j) \sim \mathrm{B}(1, p_j) \tag{1}$$

where the expected value $\hat{R}_n = \mathbb{E}(R_n(j)) = p_j$

Note that our $R_n(j) = \{0, 1\}$ although R_n could assume other non-negative values if we consider other distributions.

Reward function II

$R_n(j)$ signals that our agent will successfully (or not) trade energy with partner j,

$$R_n(j) = \begin{cases} 0 & \text{partner } j \text{ don't accept for trading energy} \\ 1 & \text{partner } j \text{ accepts for trading} \end{cases} \tag{2}$$

Reward function III

Since our agent received a sequence of R_n we can as well calculate the total reward R_{Total} for the episode e, and is equal to

Total reward:

$$R_{Total} = \mathbb{E}(R_n) = \frac{1}{N}\sum R_n = \hat{\mu}_{n-1} + \frac{1}{N}(R_n - \hat{\mu}_{n-1}) \tag{3}$$

where $\hat{\mu}_{n-1} = \mathbb{E}(R_{Total})$ is the expected value from previous step.

Reward function IV

The goal for our agent is to maximize the R_{Total}

Maximize R_{Total}

$$\max \sum_{n \in N} \mathbb{E}(R_n(j)) = \sum_{j} \mu_j \tag{4}$$

where μ_j denotes the expectation $\mathbb{E}(R_n(j))$ of partner j.

The optimal policy is then $a_n^* = \text{argmax} \sum \mu_j$.
Keep in mind: The agent has no knowledge about the μ_j, which makes this maximization non-trivial. Therefore, the agent must learn μ_j by observing $R_n(j)$ in order to deduce the optimal policy a_n^*.

Budget constraint I

Besides receiving the $R_n(j)$, the agent trades an energy quantity, denoted $E_n(j)$ upon selecting partner j. The literature instead addresses this as a pulling cost c_j and the agent has a cost budget B, which cannot exceed during the same episode e.
Our energy budget considering partners $j \in \mathcal{J}$ is

Energy constraint

$$\sum_{j \in \mathcal{J}} N_j E_n(j) \leq E_{Target} \tag{5}$$

where N_j is the number of **successful** selections of partner j,

Budget constraint II

We can re-write (5) using step $n = \{1, \ldots, n_{\max}\}$ instead

$$s_n = \sum_{n \in N} E_n(j) R_n(j) \tag{6}$$

$E_n(j)$ is multiplied by $R_n(j)$ because of (2). Equation (6) represents the state s_n of the RL framework, which is used as a stopping condition:

State s_n as stopping condition

RL environment stops when:

- step $n = n_{\max}$ OR
- $s_n = E_{Target}$

Mathematical formulation

The formulation of our P2P market as multi-armed bandit problem is

Energy-target MAD problem

$$\max_{N_j} \quad \sum_{j \in \mathcal{J}} N_j \mu_j \tag{7a}$$

$$\text{s.t.} \sum_{j \in \mathcal{J}} N_j E_j \leq E_{target} \tag{7b}$$

where N_j is the number of times that partner j has to be selected.

So, the optimal policy $a_n^* = N_j^*$.

Reinforcement learning approach I

In order to solve this problem we would have to know the expected value μ_j in advance, which does not hold in our case.

RL problem version

Instead, we can adapt to a setup where our agent learns the optimal policy:

- agent
  - Select $a_n = j(n)$ for step n
  - Send a_n to the environment
- environment
  - Calculate $R_n(j)$ by (1) for step n
  - Update R_{Total} by (3), state s_n by (6)
  - Send back $R_n(j)$, R_{Total} and s_n to the agent
  - Stop episode (if stopping conditions are *true*)

Reinforcement learning approach II

RL uses the Q-action function $Q_n(j)$ as estimator of the *unknown* distribution behind μ_j, our case modelled as Bernoulli (1). Then, $Q_n(j)$ is calculated as:

Q-function

$$Q_n(j) = Q_{n-1}(j) + \frac{1}{N_j}\left(R_n(j) - Q_{n-1}(j)\right) \tag{8}$$

where N_j is the number of times partner j has been selected.

Important note

- The correct $Q_n(j)$ estimation is key for solving the MAD problem

For this reason, all strategies focus on the learning/estimating the $Q_n(j)$ over steps n.

Exploration *versus* exploitation I

Assuming we eventually get the correct $Q_n(j)$ estimation, then

Optimal policy

Our optimal policy is derived from

$$a_n^* = \text{argmax} \sum_{j \in \mathcal{J}} Q_n(j) \tag{9}$$

a_n^* corresponds to the set of partners j with maximum Q_n until the energy target constraint (5) is fulfilled.

I omit the mathematical proof of this equation. However, there is an **important question:**

- How do we correctly estimate $Q_n(j)$? Otherwise, (9) does **not** hold

Exploration *versus* exploitation II

Estimating $Q_n(j)$

This comes to the famous trade-off exploration *vs* exploitation

1. First steps n are used to explore $Q_n(j)$ of each partner j (**exploration part**)
2. Next steps n selects, or exploits, the maximum $Q_n(j)$ (**exploitation part**)

Any policy strategy deals with this trade-off and many can be found in the literature, which is out of the point here. The key point is to adopt a proper **exploration part**, retrieving an accurate $Q_n(j)$ estimation, so that we are sure the **exploitation part** is choosing the optimal policy (or close by).

Policy strategies

The classic MAD problem The literature present many strategies for the MAD problem, these are the main ones:

- Random or naive
- Greedy (and ϵ-decay variants)
- Thompson sampler
- Upper confidence bound

The strategies coloured in green were implemented and can be found on the GitHub repo: between lines 261-287 of file *MAD_Penv.py*.

Propagate between episodes I

The classic MAD problem is solved on a single episode, which results on a continuous exploration *vs* exploitation. However, our energy-target MAD problem is composed by

1. Several episodes, with small number of steps n, that are independent from each other
2. New episode $e + 1$ starts without knowing the final estimation of $Q_n(j)$ from previous episode e
3. The $Q_n(j)$ exploration *vs* exploitation part re-starts every new episode

Our work then uses what we call the *propagation mechanism* to overcome this issue.

Propagate between episodes II

We have inspired from the *experience replay* used from the cross-entropy RL method, which trains NN for the optimal policy of classic RL problems. Our propagation mechanism is then

Algorithm 1: Propagation mechanism

Run first episodes $e = [1 - 20]$ independent from each other ;
Create batch $\leftarrow \{ R^e_{Total}; Q^e_n(j) \}$;
while *length(batch)* ≥ 20 **do**
    Calculate bound $\overline{R_{batch}} \leftarrow$ percentile(R^e_{Total}, λ) ;
    Select episodes e from batch $\leftarrow R^e_{Total} \geq \overline{R_{batch}}$;
    Calculate next episode Q-function $Q^{e+1}_n(j) \leftarrow \mathbb{E}(Q^e_n(j))$;
end

Propagate between episodes III

Basically, the Q-function for next episodes $Q_n^{e+1}(j)$ is

- Moving mean $Q_n^e(j)$ from the episodes in the top-percentiles above λ
- I had simulations with $\lambda = 70, 80, 90$, meaning 70%, 80% and 90% percentiles

Important remarks from the code that relates to the propagation mechanism:

Remarks

- *batch_size* parameter changes the length of first episodes; I consider 20
- *rd_pct* parameter represents the λ in the percentile

This is a standard, but not the optimal way, of doing the propagation that can have further improvements.

Pitfalls implementation I

Here, I enumerate the pitfalls with this implementation, which can be motivators as future work:

1. Single partner j selection as action a_n every step n
2. Reward function in (1) is stationary
3. Non-existing market or consumer-centric features in the RL algorithm
4. Missing a comparison with a benchmark approach
5. Not enough *playing around with* policy strategies and simulations

Pitfalls implementation II

Single partner selection

- Convenient but it is not optimal for our problem
- This implementation made the policy strategies, after exploration, select always the single partner j with highest $Q_n(j)$
- The switch would occur few times like the case study with 15 partners:
  - One of partners 4 and 5 with $p_j = \{0.95; 0.96\}$ was always selected at different episodes

Therefore, I added the condition $N_j \leq 3$ (times partner j was selected with $R_n(j) = 1$), which bounds the $Q_n(j)$ for partners with high p_j leaving more exploration for other partners. You can find this condition in the code: *trading_agent.action* (line 213) on file *MAD_env.py*.

Pitfalls implementation III

Reward function in (1)

It stays the same for all episodes, which is very unlikely in real applications. However, the literature presents MAD problems with non-stationary distributions, where you would have to use a discount factor γ, which can be found in the Richard Sutton book (link here). Note that this discount factor is similar to the one used in the Bellman equation.

Non-existing market or consumer-centric features

- Total reward R_{Total} only calculates the success of trading energy
- State s_n only models the energy profile until reaches the E_{Target}

I did not model the offering price or consumer preferences for each partner j, which would make sense in future versions of this problem.

Pitfalls implementation IV

Missing a comparison with a benchmark approach

The lack of a benchmark approach is a clear pitfall of this work. We could solve this problem as well using chance constrained programming (or a similar stochastic approach), and compare the results with the RL algorithm.

Not enough *playing around with* policy strategies and simulations

Due to time constraints, I have just implemented standard policy strategies for the MAD problem, excluding the UCB strategy. Definitely, these strategies can be more adapted to the specific P2P market problem, which may result in better results. In terms of simulation, there is more to be done as well. The GitHub repo has results for simulations with 15 and 30 partners. We can always make with more partners and/or different p_j (1).