Machine Learning in Finance
Section I3
Lecture 13

Michael G Sotiropoulos

NYU Tandon
Deutsche Bank

3-May-2019

Contents

Reinforcement Learning: Concepts (I)

Reinforcement learning (RL) involves an *agent* who interacts with an *environment* through a sequence of actions taken and rewards received.
The agent is a decision maker, who determines the action to take at each turn.
The environment provides a reward after an action, and moves to a new state.
The objective is to maximize the total reward after playing a sequence of turns (i.e. maximize the sum of per-turn rewards).

The agent learns about the environment from experience (trial and error).

- Experience is gained by the actions taken and their resulting rewards, i.e. the agent uses actions to **explore** the environment.
- At the same time, the agent's objective is to **exploit** his accumulated experience in order to maximize total reward.

This is called the exploration/exploitation tradeoff.

The fundamental object in RL is the **action-value function**, which gives the expected reward if a certain action is taken in a given environment state.

- ML algorithms are used to update the agent's action-value function.
- RL solves an optimal control problem, with an embedded statistical learning (fitting) task.

Reinforcement Learning: Concepts (II)

RL is best described as a Markov Decision Process (MDP).

- MDP is a mathematical model about learning from experience and acting in discrete time steps while optimizing a long term goal.

There is a sequence of discrete times (turns) $t = 0, 1, 2, \ldots T$, called the **episode.** The episode may be finite (ending in game-over state) or perpetual.

- At time t the agent knows the state of the environment $S_t \in \mathcal{S}$ and takes action $A_t \in \mathcal{A}$.
- At time $t+1$ the agent receives reward $R_{t+1} \in \mathcal{R}$ and the environment moves to state $S_{t+1} \in \mathcal{S}$.

This defines a "**SARSA**" trajectory: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, \ldots$
In a finite MDP, the sets of states, actions and rewards $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ are finite.

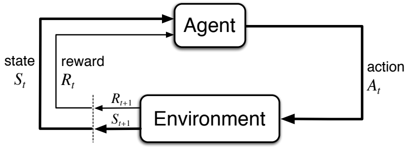


Figure: MDP and the agent-environment interaction. From [SB]

Reinforcement Learning: Concepts (III)

In a finite MDP, the dynamics are fully defined by the transition probabilities

$$p\left(s', r|s, a\right) := \Pr\left(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\right) \tag{1}$$

This is a consequence of the Markov property.

Since $p\left(s', r|s, a\right)$ is a probability, it satisfies the normalization property

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p\left(s', r|s, a\right) = 1 \tag{2}$$

The transition probability to state s' from state s, after action a is given by

$$p\left(s'|s, a\right) := \Pr\left(S_t = s'|S_{t-1} = s, A_{t-1} = a\right) = \sum_{r \in \mathcal{R}} p\left(s', r|s, a\right) \tag{3}$$

The expected reward from being in state s and taking action a is given by

$$r\left(s, a\right) := \mathbb{E}\left(R_t|S_{t-1} = s, A_{t-1} = a\right) = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p\left(s', r|s, a\right) \tag{4}$$

Reinforcement Learning: Concepts (IV)

If the agent is at time t, the total future payoff until the end of the episode is

$$G_t = R_{t+1} + R_{t+2} + \ldots + R_T = R_{t+1} + G_{t+1} \tag{5}$$

Future payoffs may have less value than present ones, so we apply discount $\gamma \leq 1$. This can define a finite discounted payoff even for perpetual episodes.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1} \tag{6}$$

Policy $\pi(a|s)$ is the probability of selecting action a in state s

- Policy defines agent behavior, and changes with experience.

State-value function of state s following policy π is the expected payoff of following π starting from s

$$v_\pi(s) := \mathbb{E}_\pi(G_t | S_t = s) \tag{7}$$

Action-value function is the expected payoff from taking action a in state s and following policy π thereafter

$$q_\pi(s, a) := \mathbb{E}_\pi(G_t | S_t = s, A_t = a) \tag{8}$$

Reinforcement Learning: Optimality and Bellman Equation (I)

Using the definition of the value function, eq. (7), and payoff, eq. (6), we get

$$v_\pi(s) = \mathbb{E}_\pi(G_t | S_t = s) = \mathbb{E}_\pi(R_{t+1} + \gamma G_{t+1} | S_t = s) \Rightarrow$$

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma \mathbb{E}_\pi(G_{t+1}|S_{t+1} = s')\right] \Rightarrow$$

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right] \tag{9}$$

The sums are taken over all actions $a \in \mathcal{A}$, states $s \in \mathcal{S}$, and rewards $r \in \mathcal{R}$. Eq. (9) is called the **Bellman equation**. It relates the value of a state s at time t with the values of all future states under policy π.

- *What are we trying to do?* Find the best policy, i.e. the policy with the highest expected payoff.
- *When is a policy π' better than a policy π?* When the following is satisfied

$$v_{\pi'}(s) \geq v_\pi(s), \quad \forall s \in \mathcal{S} \tag{10}$$

The above expression defines a partial ordering among all possible policies. In other words there must exist a best (optimal) policy.

- The objective of Reinforcement Learning is to compute the optimal policy.

Reinforcement Learning: Optimality and Bellman Equation (II)

Let's denote the optimal policy by $\pi_\star$. It has corresponding state-value function $v_\star(s)$ and action-value function $q_\star(s,a)$.

RL can be formulated as an optimization problem for the value function

$$v_\star(s) = \max_\pi v_\pi(s) \tag{11}$$

with the value function being subject to the Bellman equation.

The optimal action-value function is related to the state-value function as

$$q_\star(s,a) = \mathbb{E}\left(R_{t+1} + \gamma v_\star(S_{t+1}) \,|\, S_t = s, A_t = a\right) \tag{12}$$

So, RL can be alternatively formulated as an action-value optimization problem

$$q_\star(s,a) = \max_\pi q_\pi(s,a) \tag{13}$$

This is in general a difficult problem to solve. In what follows we explore methods for efficiently finding approximate solutions.

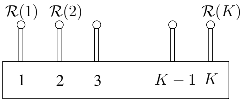
Multi-armed Bandits: Setup (I)

A multi-armed bandit is a machine with K arms (levers).

- At each turn $t = 1 \dots T$, the player chooses an action $A_t = 1 \dots K$ (pulls an arm) and receives reward R_t.
- The reward can be a deterministic or stochastic function of the action $R_t = \mathcal{R}(A_t = k)$. For example, lever k may always pay a fixed amount R_k, or R_k is a sample from a stationary distribution.
- The **reward function** $\mathcal{R}(A_t)$ is unknown to the player.
- The player's objective is to learn a policy π that maximizes the cumulative reward (payoff) over a run (episode), $G = \sum_{i=1}^{T} \gamma_t R_t$.
  Here we will ignore discounting, ($\gamma_t = 1$).

This game is our entry point to RL.

- The multi-armed bandit is an example of a sequential MDP with trivial state space (one single state). After an arm is pulled, the bandit has no memory, the next turn starts with a fresh bandit copy.



Multi-armed Bandits: Setup (II)

Since the state space is trivial, the RL problem simplifies.
We only need to consider the action-value function

$$q(k) = \mathbb{E}(R_t | A_t = k) \tag{14}$$

- If the player knew the action-value $q(k)$, finding the optimal policy $\pi_\star$ would be trivial. Simply choose the level $k_\star$ with the highest action value.
- Instead, at each turn t, the player updates a running **estimate** $Q_t(k)$ of the action-value for each arm $k = 1, 2 \ldots K$.
- The player needs to keep updating the estimate $Q_t(k)$ (explore) while using the current best estimate to maximize reward (exploit).

Policy is an algorithm for generating the action sequence $A = \{A_1, A_2, \ldots, A_T\}$.
Finding the optimal policy means solving the optimization problem

$$A^\star = \arg\max_A \sum_{t=1}^{T} q(A_t) \tag{15}$$

Equivalently, the optimal policy minimizes the regret

$$A^\star = \arg\min_A \sum_{t=1}^{T} \left[q(A_t^\star) - q(A_t)\right] \tag{16}$$

Multi-armed Bandits: Environment

To specify, simulate and tune the game we need three ingredients:

1. Bandit specification, i.e. the RL environment
2. Action-value update method, i.e. how to compute $Q(k)$
3. Exploration-exploitation policy

Bandit specification is equivalent to defining the number of arms K, and the reward function $\mathcal{R}(k)$ for $k = 1 \ldots K$. Common choices are

- **Fixed**: at each turn the reward is fixed at $\mathcal{R}(k) = c_k$.
- **Normal**: at each turn the reward is a draw from the normal distribution $\mathcal{R}(k) \sim \mathcal{N}(\mu_k, \sigma_k)$.
- **Bernoulli**: at each turn there are two possible rewards $\mathcal{R}(k) = 1, 0$ with corresponding probabilities q_k and $1 - q_k$.
- **Non-stationary**: any of the above, with the parameters being deterministic functions of time.

Multi-armed Bandits: Action-value Update

Action-value method specification is equivalent to defining the update rule

$$Q_{t+1}(k) \leftarrow f(Q_t, R_t) \tag{17}$$

Common choices are:
Sample Average. This is simply the average reward from action k.
Call $n_t(k)$ the number of times that arm k has been selected so far, i.e. in the interval $[t_1, t_k]$. Then update the estimate for the next turn as

$$Q_{t+1}(k) \leftarrow Q_t(k) + \frac{1}{n_t(k)} \left(\mathcal{R}(A_t = k) - Q_t(k)\right) \tag{18}$$

Exponential Moving Average with rate α.

$$Q_{t+1}(k) \leftarrow Q_t(k) + \alpha \left(\mathcal{R}(A_t = k) - Q_t(k)\right) \tag{19}$$

Bayesian Update (very efficient for Bernoulli bandit).
Assume that $Q_t(k)$ is random.
Start with a prior that is also Beta distributed, $Q_1(k) \sim B(\alpha_1, \beta_1)$.
Then, the posterior is also Beta distributed (conjugate), $Q_{t+1}(k) \sim B(\alpha_k, \beta_k)$.
The update rules for the posterior parameters are

$$\alpha_k \leftarrow \alpha_k + R_t, \quad \beta_k \leftarrow \beta_k + (1 - R_t) \quad \Rightarrow \mathbb{E}Q_{t+1}(k) = \alpha_k/(\alpha_k + \beta_k) \tag{20}$$

Multi-armed Bandits: Policies (I)

This is the crux of the algorithm. A policy balances exploration vs exploitation. Common policies are:

ϵ-Greedy
This policy takes the action considered best so far, but occasionally does random exploration.
At time t, the current best action is estimated to be

$$k_t^\star = \arg\max_k Q_t(k) \tag{21}$$

The algorithm uses the parameter $0 \leq \epsilon \leq 1$ to choose the action

$$A_t = \begin{cases} k_t^\star, & \text{with } Prob = (1-\epsilon) \\ \mathcal{U}(1, 2, ..., \underbrace{k_t^\star}_{\text{no}}, \dots K) & \text{with } Prob = \epsilon \\ \end{cases} \tag{22}$$

$\mathcal{U}$ is the uniform distribution over $K-1$ discrete actions, i.e. it excludes $k_t^\star$.
The parameter ϵ is to be tuned. Typically $\epsilon \leq 0.1$.

Multi-armed Bandits: Policies (II)

UCB (Upper Confidence Bound)
This policy adds a measure of uncertainty around the current estimate of each action and then chooses the best.
The idea is to encourage exploration of arms with higher uncertainty $\sigma(k)$, and avoid exploring those that are more certain.

$$A_t^\star = \arg\max_k \left[Q_t(k) + c * \sigma_t(k) \right] \tag{23}$$

The parameter c is the "risk weight", typically $\mathcal{O}(1)$.

The uncertainty $\sigma_t(k)$ is also a running estimate.
For a Bernoulli bandit with a Beta prior, the running estimate of the variance is also an efficient update

$$\sigma_t^2(k) \leftarrow \frac{\alpha_k \beta_k}{(\alpha_k + \beta_k)^2(\alpha_k + \beta_k + 1)} \tag{24}$$

Multi-armed Bandits: Policies (III)

Stochastic Gradient Ascent

Each action has a running numerical preference score $H_t(k)$.
The probability of choosing action k at turn t is determined by the softmax distribution

$$\mathbb{P}(A_t = k) = \frac{e^{H_t(k)}}{\sum_{i=1}^{K} e^{H_t(k)}} = \pi_t(k) \tag{25}$$

The overall level of $H_t(k)$ does not affect the probability, only relative differences do.

The question is how to update the preference score $H_{t+1}(k)$ for every k, after having chosen A_t at turn t.
It can be shown [SB] that a stochastic gradient ascent algorithm that maximizes expected total reward leads to the update rule

$$H_{t+1}(A_t) \leftarrow H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t)) \tag{26}$$

$$H_{t+1}(k) \leftarrow H_t(k) + \alpha(R_t - \bar{R}_t)\pi_t(k), \quad k \neq A_t \tag{27}$$

The baseline reward $\bar{R}_t$ is the average of all past rewards up to time t, and $\alpha > 0$ is the learning rate.

Multi-armed Bandits: Policies (IV)

Thomson Sampling
A greedy algorithm chooses the action with the highest current mean. Thomson sampling [RVR] draws a random deviate for the mean from the posterior distribution and then chooses the action with the highest mean.

For example, for a Bernoulli bandit at turn t we have the estimates $\mathbb{E}Q_t(k) = \alpha_k/(\alpha_k + \beta_k)$. Instead of choosing greedily

$$k_t^\star = \arg\max_k \{\alpha_k/(\alpha_k + \beta_k)\} \tag{28}$$

we first draw K random samples, one sample from each of the beta distributions $B(\alpha_k, \beta_k)$

$$\theta_k \sim B(\alpha_k, \beta_k), \quad k = 1, 2, \dots K \tag{29}$$

and then we choose the action

$$k_t^\star = \arg\max_k \{\theta_k\} \tag{30}$$

Implementation: Components

There are three components: `Bandit`, `Game`, and `Experiment`.

The abstract base class `Bandit` only needs to store the number of arms (K), and requires all derived bandits to implement a `get_reward(k)` method.

A game involves combining a bandit, an action-value estimation method and an exploration-exploitation policy. A *run* means playing the game for T time steps (turns). At the end of a run, the game has collected the necessary reward/regret time series and statistics.

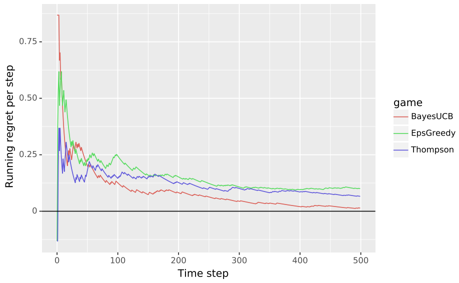
The abstract `Game` class contains a reference to the `Bandit`, visit counters for every action, and lists with the actions taken and rewards received at every step. It does all common accounting calculations. Its main method is `run(T)`. `Game` requires all derived game classes to implement the `run_one_step()` method, where the action-value updates and policy-specific decisions are made.

Finally, the `Experiment` class is a collection of games that are set up using different bandits, initial parameters, or policies. All games in an experiment are run for the same number of steps, over the same number of runs, calling the method `run(T, N)`.

Implementation: Experiments

A measure of efficiency is the regret per step.
In a good algorithm, this should quickly revert towards zero.
The figure below compares three games using the same Bernoulli bandit.

- The ϵ-Greedy policy has exploration probability $p = 10\%$.
- The UCB and Thompson policies are optimistically initialized ($\alpha = \beta = 1$)
- The risk weight for UCB is set to $c = 1$.



The Bandit/Game/Experiment implementation is in the package `bandits`.
For details on how to use it see the notebook **L13-Bandit-Examples.ipynb**.

Further Reading

1. Read chapters 2 and 3 in [SB]
2. For a review of Thomson sampling consult the paper [RVR]
3. For more code examples browse the Github repo in [LW]

References

[SB] Sutton R.S. and Barto A.G. "Reinforcement Learning An Introduction" 2nd edition. MIT, 2018

[RVR] Russo D., Van Roy B., Kazerouni A. and Osband I. "A Tutorial on Thomson Sampling". arxiv 1707.02038, 2017.

[LW] Weng Lilian. "The Multi-Armed Bandit Problem and Its Solutions". Github, 2018