Reinforcement learning

Silèye Ba

Adjunct Lecturer, Neoma Business School, Paris
Machine Learning Scientist, L'Oreal, Paris

16.03.2022

Overview

Overview

Class evaluation

Group projects : deadline 31 of March 2023

- Choose a finance problem at `https://challengedata.ens.fr/`
- Constitute 2 or 3-persons groups : send group and problem
- Deliverables : data, working notebook and synthetic report

Personal projects

- Problem : credit scoring problem
- Deadline to submit 16th of March 2023 : today !

Projects expectations

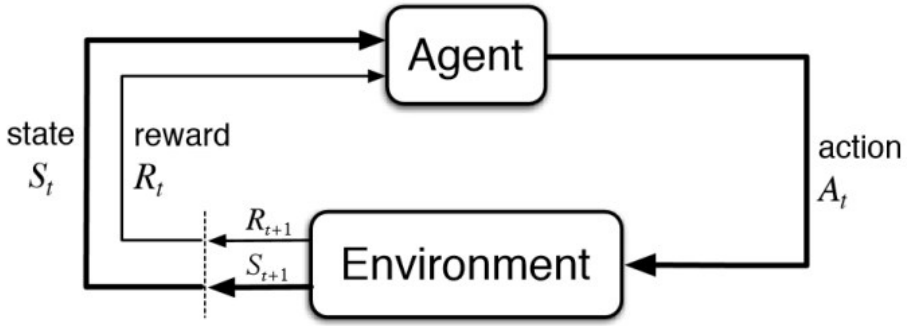
- Use what you learned during class : test and compare many models
- Evaluating your proposals properly : cross validation, use right metrics
- Being creative is always a plus : go beyong what you learned in class
- Code quality is part of the evaluation

Overview

Reinforcement learning : definition

- Reinforcement learning for an agent is learning to map observations to actions to maximize a reward.
- Agent must discover by trying actions generating high rewards.
- Rewards may be delayed from actions.

Reinforcement learning : formal problem



- S_t : **environment's state**
  - All information considered by agent to take actions
- A_t : agent's **action** taken according to state S_t
- R_t : **reward** resulting from action A_t
- Agent goal :
  - Maximize **cumulated rewards** $G_t = \sum_{k=1}^{K} \gamma^{k-1} R_{t+k}$
- Solution framework : **Markov Decision Processes**

Reinforcement learning for portfolio optimization

- **State variable** S_t
  - Postions, prices, return histories
  - Any other relevant historical information : news articles, tweets, ..
- **Actions** a_t :
  - Buy, hold, sell actions
  - Portfolio weights selection
- **Rewards** R_t :
  - Portfolio value
  - Stock return
  - Negative risk
  - Negative costs

Reinforcement learning specifities in machine learning

- **Machine learning** : unsupervised, supervised, reinforcement learning
- **Supervised learning** :
  - Teacher provides explicit feedbacks : labels
  - Agent(classifer) does not affect the environment
  - Problem is not interactive
- **Reinforcement learning** :
  - Teacher (environment) provides only partial feedbacks : rewards
  - Agent affects the environment
  - Problem is interactive by nature

Markov processes

Definition

A stochastic process (S_t) is a first order Markov process if given it's present value S_t it's future value S_{t+1} is independant of it's past values $S_{t-1:0} = S_{t-1}, S_{t-2}, ..., S_0$:

$$p(S_{t+1}|S_{t-1:0}) = p(S_{t+1}|S_t)$$

Markov reward process

Definition

A Markov reward process is a Markov process (S_t), associated to a reward process (R_t) markovian with respect to S_t defined by $p(S_{t+1}, R_{t+1}|S_t)$
Given state $S_t = s$, the average reward is defined as :

$$R(s) = \mathbb{E}(R_{t+1}|S_t = s)$$

Given discount factor $\gamma < 1$, **cumulated return** at time t is defined as :

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{k-1} r_{t+k} + ... = \sum_{k=1} \gamma^{k-1} r_{t+k}$$

The **state value function** is defined as :

$$V(s) = \mathbb{E}(G_t|S_t = s) = \mathbb{E}\left(\sum_{k=1} \gamma^{k-1} R_{t+k}|S_t = s\right)$$

Markov decision process (MDP)

Definition

- A Markov decision process is a triplet of action, state, and reward processes (S_t, A_t, R_t) such that (S_{t+1}, R_{t+1}) is Markovian wrt (S_t, A_t).
- MDP is characterized by transition probabilities $p(S_{t+1}, R_{t+1}|S_t, A_t)$
- Given state and action (s, a) average reward is given by :

$$R(s, a) = \mathbb{E}(R_{t+1}|S_t = s, A_t = a)$$

- **State-action value function** is defined as the average cumulated return when starting from s and acting as a :

$$Q(s, a) = \mathbb{E}(G_t|S_t = s, A_t = a) = \mathbb{E}(\sum_{k=1} \gamma^{k-1} R_{t+k}|S_t = s, A_t = a)$$

MDP and MRP

Policy

A policy $\pi(a|s)$ is a map between states and actions.

An MDP and a policy induce an MRP

An MDP characterized by transitions $p(S_{t+1}, R_{t+1}|S_t, A_t)$ together with a policy $\pi(A_t|S_t)$ induce an MRP defined by transitions :

$$p^{\pi}(S_{t+1}, R_{t+1}|S_t) = \sum_{a \in \mathcal{A}} \pi(A_t = a|S_t) p(S_{t+1}, R_{t+1}|S_t, A_t = a)$$

with average reward :

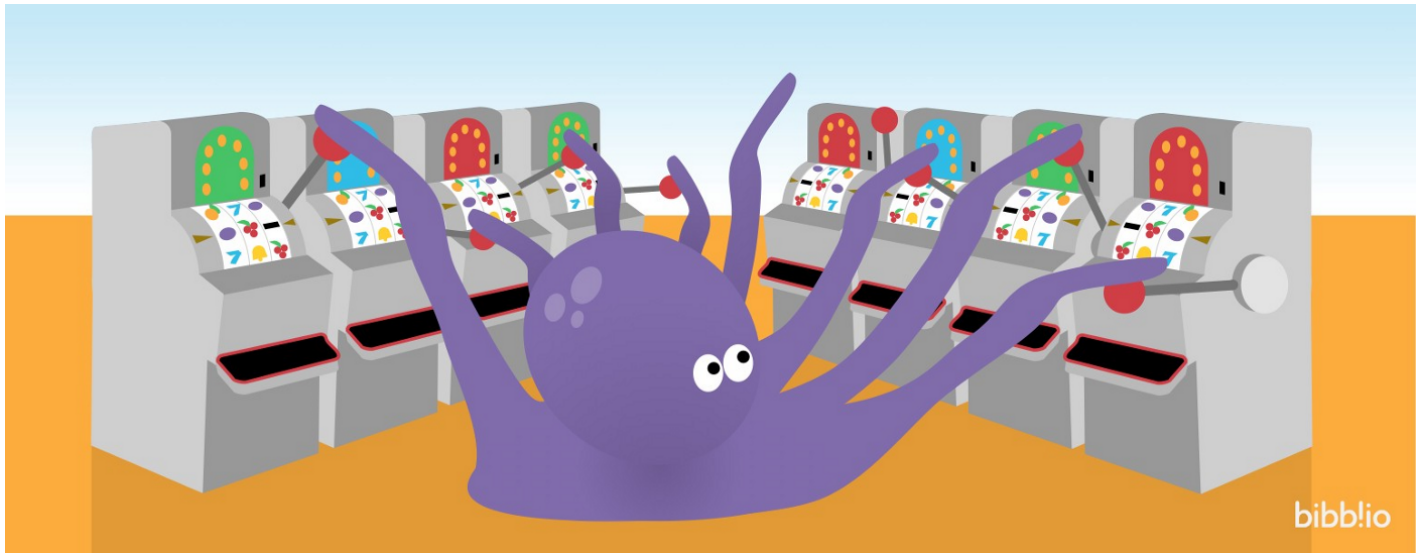
$$R^{\pi}(s) = \mathbb{E}^{\pi} R(s, a) = \sum_{a \in \mathcal{A}} R(s, a) \pi(a|s)$$

Reinforcement learning solutions

- Estimating optimal policy $\pi(a|s)$,
- estimating optimal value function $V(s)$,
- estimating optimal state-action value function $Q(s, a)$.

Overview

Multi-arms bandits



- Environment : multi-arms bandits
- Action : pull one of the arms

Multi-arms bandits : formal problem

- Agent pulls one arm out of K possible : action $A_t \in \{a_1, a_2, ..., a_K\}$
- Agent receives reward R_t for pulling arm A_t
- Agent's goal : maximize cumulated rewards after T actions
  - Episodic reinforcement learning : T is finite
- Simplification with respect to general R.L. : there is no state S_t

Action value function $Q_t(a_k)$

$$Q_t(a_k) = \frac{\sum_{s=1}^{t} \delta_{A_s}(a_k) R_t}{\sum_{s=1}^{t} \delta_{A_t}(a_k)}$$

- Action value function : average rewards for each action
- Measures action values
  - Rewarding actions have high action values

Multi-arms bandits : exploitation vs exploration

Policy : stragegy followed to pull arms

- **Exploitation** : pull arms that are known to provide high rewards
- **Exploration** : test arms with unknown rewards distribution
- Trade-off : how to balance exploration and exploitation ?

Greedy policy

- Agent always chooses the best arm $A_t = \arg\max_{a_k} Q_t(a_k)$
- Only exploitation : agent may be stucked on a non optimal choice

Epsilon-greedy policy

- Agent chooses an arm
  - either greedily with probability $1 - \epsilon$
  - or uniformly among the K arms with probability ϵ
- Balance exploration and exploitation to discover actions values

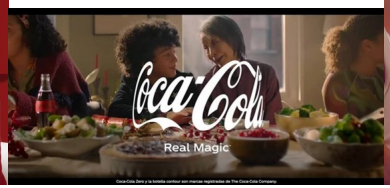
Online ad campain problem


Problem statement

- The business has four ad visuals for our online campain
- How should we choose which ad visual to show to customers ?

Ad campain : multi-arms bandit solution


a_1



a_2

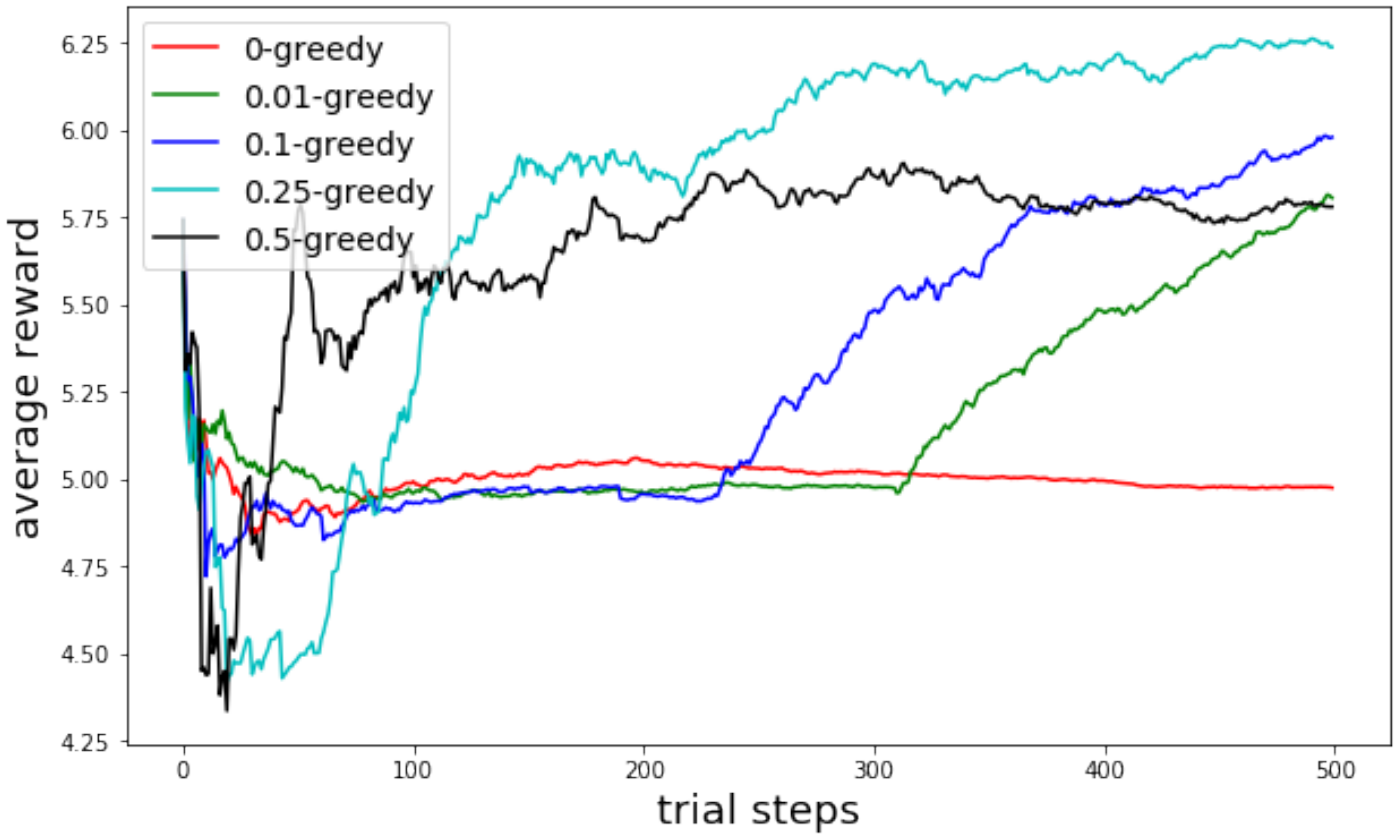
a_3


a_4

1. Consider each ad as an arm, initialize action value function Q_t
2. For each user t :
   - Choose an arm a_k according to epsilon greedy policy
   - Reward values :
     - 1 : when user clicks on ad
     - 0 : when user dose not click
3. Update action value function for selected arm $Q_t(a_k)$

Lab simulated case

- Agent has the choice between four arms during 500 trial steps
- Gaussian rewards distribution for each arms
- Action value function for various Epsilon-greedy policies



Questions about multi-arms bandit for to Ad campain

a_1

a_2

a_3

a_4

Questions

- Can you think about other forms of rewards ?
- Can you think about information about users to exploit ?

Think about multi-arms bandits for these problems

1. You work for a Netflix and have to design an algorithm to select the best snapshot for a video
2. You work for Netfix and have to design an algorithm to choose the best movie trailer
3. You work for Snapchat and have to design an algorithm to choose between the old an a new user interface
4. You work for HRT and have to design an algorithm to choose between investment strategies

Adaptive portfolio with multi-arms bandit

Hypotheses

- Fixed assets set
- Finite time horizon
- For simplicity, transaction costs are ignored

Principles

- Define classic investment strategies as bandit arms
- Build a portfolio using multi-arms bandits framework

Asset returns

- $t_l = l\Delta_t,\ l = 0, ..., L$: trading times
  - L : total number of time cycles
  - Δ_t : time interval duration (day, week, month)
- N : number of financial assets

Financial assets returns

- Price of asset n at time t_l : p_{nl}
- Return of asset n at time t_l : $r_{nl} = \frac{p_{nl}}{p_{nl-1}}$
- Return vector at time t_l :

$$r_l = \begin{pmatrix} r_{1l} \\ . \\ . \\ r_{nl} \\ . \\ . \\ r_{Nl} \end{pmatrix}$$

Portfolio weights

- Weight of asset n at time t_l : w_{nl}
- Weight vector at time t_l :

$$w_l = \begin{pmatrix} w_{1l} \\ . \\ . \\ w_{nl} \\ . \\ . \\ w_{Nl} \end{pmatrix}$$

Constraint

$$\sum_{n=1}^{N} w_{nl} = 1$$

Multi-arms bandit specifications

- What are arms ?
- What are rewards ?

Bandit arms : classic portfolio weighting

Classic weighting

1. Hold : $w_l = w_{l-1}$
2. Sell : $w_{1l} = 1,\ w_{nl} = 0,\ n \neq 1$
3. Equal weighting (EW) : $w_{nl} = \frac{1}{N}$ for all n
4. Value weighting (VW) : for all n

$$w_{nl} = \frac{w_{nl} r_{nl}}{\sum_{m=1}^{N} w_{ml} r_{ml}}$$

5. Mean-variance weighting (MVW) :

$$w_l = \arg\max_{v_l, \sum_{n=1}^{N} v_{nl}=1} v_l^\top \Sigma_l v_l - r_l^\top v_l$$

Bandit arms

$$A_l \in \{a_1 = Hold, a_2 = Sell, a_3 = EW, a_4 = VW, a_5 = MVW\}$$

Bandit rewards

Rewards

- Assumed a time t_l selected arm is a_k
- $SR(a_i)$: Sharpe ratio for action a_i
- $\mathcal{O} = \{i \neq k | SR(a_k) - SR(a_i) \geq 0\}$
- A fixed parameter $c \in \{1, 2, 3, 4\}$
- Reward for action a_k :

$$R_l = \begin{cases} 1 \text{ if} |\mathcal{O}| \geq c \\ 0 \text{ otherwise} \end{cases}$$

- With c a fixed parameter in $\{1, 2, 3, 4\}$

Policy based action value function

Action value function

$$Q_l(a_k) = \frac{\sum_{s=1}^{t} \delta_{A_s}(a_k) R_s}{\sum_{s=1}^{t} \delta_{A_s}(a_k)}$$

Epsilon greedy policy

- Policy to favor exploitation while allowing exploitation

$$\pi(A_l) = \begin{cases} \arg\max_{a_k} Q_t(a_k) \text{ with proba } 1-\epsilon \\ \text{uniformly in } \{a_1, a_2, a_3, a_4, a_5\} \text{ with proba } \epsilon \end{cases}$$

Other policies : Thompson sampling

Rewards distribution

Assume reward for action a_i is modelled bas a Beta distribution

$$P(r|a_i) = B(r; \alpha_i, \beta_i) = \mathbb{I}_{[0,1]}(r)\frac{r^{\alpha-1}(1-r)^{\beta-1}}{\Gamma(\alpha, \beta)}$$

Arm selection

- Sample $r_i \sim P(r|a_i) = B(r; \alpha_i, \beta_i)$
- Select arm according to $k = \arg\max_i r_i$
- Compute reward R_l for selected arm k

Reward model update

- If $R_l = 1 : \alpha_k = \alpha_k + 1$
- If $R_l = 0 : \beta_k = \beta_k + 1$

Multi arms bandit : summary

- Simplified reinforcement learning : only actions and rewards
- For more complex situations a state is necessary
- With a state, Markov decision process formalism is required

Overview

Reinforcement learning solutions

- Estimating optimal policy $\pi(a|s)$,
- estimating optimal value function $V(s)$,
- estimating optimal state-action value function $Q(s, a)$.

State value function estimation

Definition

- **Model based** : when underlying Markov reward process $P(S_{t+1}, R_{t+1}|S_t)$ is known
- **Episodic** :when interactions between agent and environment end in a finite number of steps T
- **Tabular** : when the number of environment's states is finite

state value function

State value function :

$$V(s) = \mathbb{E}(G_t|S_t = s) = \mathbb{E}\left(\sum_{k=1} \gamma^{k-1} R_{t+k}|S_t = s\right)$$

gives expected cumulated reward associated to each state.

Value function estimation using Bellman equation

Bellman equation

Value function for state s is the average reward of state s plus the average value functions of consecutive states s' weighted transition probabilities.

$$\begin{aligned} V(s) &= \mathbb{E}(G_t|S_t = s) \\ &= R(s) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s)V(s') \end{aligned}$$

Bellman equation in matrix form :

$$V = R + \gamma PV$$

Direct value function estimation

$$V = (I - \gamma P)^{-1} R$$

- For large state spaces matrix P can be of very large dimension

Iterative value function estimation using Bellman eq.

- Iterative approximation of the Bellman equation solution can be used to avoid inverting matrix P for large state spaces
- Iterative fixed point like approximation can be used

Iterative algorithm

1. For all $s \in \mathcal{S}$ initialize $V_0(s) = 0$
2. Iterate over $k = 1, 2, ...$ until convergence :
   1. for all $s \in \mathcal{S}$ update V_k from V_{k-1}

$$V_k(s) = R(s) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s) V_{k-1}(s')$$

Model based value function estimation for episodic cases

Monte Carlo based estimation for tabular case

1. For every state value s
   1. Use $p(S_{t+1}, R_{t+1}|S_t)$ to sample N episodes starting from $s_{n,0} = s$

$$s_{n,0}, (r_{n,1}, s_{n,1}), ..., (r_{n,T_n}, s_{n,T_n})$$

   2. Compute discounted return using sampled episodes :

$$G_{n,0} = r_{n,1} + \gamma r_{n,2} + ... + \gamma^{T_n - 1} r_{n,T_n}$$

2. Estimate value function as :

$$V(s) = \mathbb{E}(G_0|S_0 = s) \approx \frac{1}{N} \sum_{n=1}^{N} G_{n,0}$$

- State values can be enumerated only in finite (tabular) cases
- Large state spaces, long episodes : computationnally intensive

Bellman model based policy evaluation

Objective

- Given $p^{\pi}(s'|s)$ state transition induced by policy $\pi(a|s)$:

$$p^{\pi}(s'|s) = \sum_{a \in \mathcal{A}} \pi(a|s) p(s'|s,a)$$

- Compute MDP value function $V^{\pi}(s)$ associated to policy $\pi(a|s)$
  - Allow to compare among many policies which one is the best

Bellman policy evaluation

1. For all $s \in \mathcal{S}$ intialize $V_0^{\pi}(s) = 0$
2. Iterate $k = 1, 2, ...$ until convergence :
   1. For $s \in \mathcal{S}$ update V_k^{π} from V_{k-1}^{π}

$$V_k^{\pi}(s) = R^{\pi}(s) + \sum_{s' \in \mathcal{S}} p^{\pi}(s'|s) V_{k-1}^{\pi}(s')$$

State and state-action value functions

- State value $V(s)$ function allows to have information about cumulated reward associated to states
- Policies can be evaluated using state value function
- Limitations : state value function does not give information about how to best choose actions
- Solution : use state action value function $Q(s, a)$

Bellman model-based policy search

Objective

- Find the policy achieving the best state-action value function
- Search is based on policy.

Bellman policy search

1. For all $s \in \mathcal{S}$, initialize randomly $\pi_0(a|s)$, set $k = 0$
2. Compute state value function $V^{\pi_{k-1}}$ using policy evaluation
3. Compute state-action value function :

$$Q^{\pi_{k-1}}(s,a) = R(s,a) + \gamma \sum_{s'} p(s'|s,a) V^{\pi_{k-1}}(s')$$

4. Update policy as $\pi_k(a|s) = \arg\max_a Q^{\pi_{k-1}}(s,a)$
5. Update $k = k + 1$
6. If $||\pi_k - \pi_{k-1}|| > 0$, loop back to step 2.)

Model free reinforcement learning

- Compute
  - State value function :

$$V^{\pi}(s) = \mathbb{E}^{\pi}(G_t|S_t = s)$$

  - State-action value function :

$$Q^{\pi}(s, a) = \mathbb{E}^{\pi}(G_t|S_t = s, A_t = a)$$

- Without knowledge of the underlying model :

$$p(S_{t+1}, R_{t+1}|St)$$

Model free Monte Carlo policy evaluation

Principle

- Use policy $\pi(a|s)$ to draw episodes to estimate cumulated reward G_{it}
- Estimate value function as average cumulated reward

First visit Monte Carlo

1. For all $s \in \mathcal{S}$ initialize $N(s) = 0$ and $G(s) = 0$
2. Using policy $\pi(a|s)$:
   1. sample episodes $s_{i1}, a_{i1}, r_{i1}, s_{i2}, ..., s_{it}, a_{it}, r_{it}, s_{it}, s_{iT_i}, a_{iT_i}, r_{iT_i}, s_{iT_i}$
   2. Compute returns $G_{it} = \sum_{k=0}^{T_i - t} \gamma^k r_{it+k}$
3. for each s visited in episode i, for the first time state s is visited
   - Increment counts $N(s) = N(s) + 1$
   - Increment return $G(s) = G(s) + G_{it}$
4. For all state s compute value function $V^{\pi}(s) = \frac{G(s)}{N(s)}$

Model free temporal difference policy evaluation

Objective

- Estimate state value function using temporal difference without model
- Temporal difference can be applied to non-episodic problems
- Value function is updated at each sampling step

Temporal difference algorithm

1. For all states $s \in \mathcal{S}$ initialize $V^{\pi}(s) = 0$
2. Iterate until convergence :
   a. Given sttae s_t, use policy $\pi(a|s_t)$ to sample a_t and get r_{t+1}, s_{t+1}
   b. Update state value function

$$V^{\pi}(s_t) = V^{\pi}(s_t) + \alpha(r_{t+1} + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t))$$

   c. Update state $s_t = s_{t+1}$

Model free MC state-action value function estimation

MC on policy algorithm

1. For all s, a initialize $\pi_0(a|s)$, $N(s,a) = 0$, $G(s,a) = 0$, $Q^{\pi_0}(s,a) = 0$
2. Loop until convergence $k = 0, 1, 2, ...$
   a. Using policy π_k samples episode $s_{i1}, a_{i1}, r_{i1}, s_{i2}, ...$
   b. For all episode i, compute returns $G_{it} = r_{it} + \gamma r_{it+1} + ...$
   c. For all s, a visited in episode i, for the first time t s, a is visited :
      - Aggregate counts $N(s,a) = N(s,a) + 1$
      - Aggregate return $G(s,a) = G(s,a) + G_{it}$
   d. For all s, a update state-value function $Q^{\pi_k}(s,a) = \frac{G(s,a)}{N(s,a)}$
   e. Update policy $\pi_{k+1}(a|s) = (1-\epsilon) \arg\max_a Q^{\pi_k}(s,a) + \epsilon U(a)$

Model free TD state-action value function estimation

Q-learning algorithm

1. Initialize $s_t = s_0$
2. Sample a_t from $\pi(a|s_t)$, observe (r_t, s_t)
3. Loop until convergence :
   a. Sample a_{t+1} from $\pi_t(a|s_{t+1})$, observe (r_{t+1}, s_{t+1})
   b. State-action value function update :

$$Q^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q^{\pi}(s_{t+1}, a') - Q^{\pi}(s_t, a_t))$$

   c. Policy update $\pi(a|s) = (1 - \epsilon) \arg\max_a Q^{\pi}(s, a) + \epsilon U(a)$

Value function approximation

Problem

- State-action spaces can not always be enumerated :
  - Go game number of states : 10^{120}
  - Robot manipulation : continuous space
- Previously presented methods can not be applied

Solution

- Approximate $V^{\pi}(s)$ or $Q^{\pi}(s, a)$ by parametrized functions $V_{\mathbf{w}}(s)$ or $Q_{\mathbf{w}}(s, a)$
- Approximating functions :
  - Linear regression
  - Kernel methods
  - Neural networks

Policy evaluation with function approximation

Principle

- Given a policy $\pi(a|s)$
- Approximate true value function $V^{\pi}(s)$ with $V_{\mathbf{w}}(s)$
- Training loss :

$$J(\mathbf{w}) = \mathbb{E}_{\pi}[(V^{\pi}(s) - V_{\mathbf{w}}(s))^2]$$

- Loss gradient :

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = 2\mathbb{E}_{\pi}[(V^{\pi}(s) - V_{\mathbf{w}}(s))\nabla_{\mathbf{w}} V_{\mathbf{w}}(s)]$$

Monte Carlo policy evaluation with function approximation

- Monte Carlo method are for episodic problems
- For single episode $V^{\pi}(s_{ti})$ is approximated by cumulated reward G_{ti}

Algorithm

1. Initialize randomly parameters $\mathbf{w}$, set $i = 1$
2. Loop until convergence
   a. Use policy $\pi(a|s)$ to sample i'th episode
   $s_{1i}, a_{1i}, r_{1i}, s_{2i}, ..., s_{ti}, a_{ti}, r_{ti}, s_{ti}, ...$
   b. For $t = 1, ..., T_i$, compute returns $G_{ti} = \sum_{k=t}^{T_i} \gamma^{k-t} r_{T_i,i}$
   c. For $t = 1, ..., T_i$ for a first visit to state s_{ti}
      1. Compute gradient

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = 2(G_t - V_{\mathbf{w}}(s_{ti}))\nabla_{\mathbf{w}} V_{\mathbf{w}}(s_{ti})$$

      2. Update parameters with gradient descent step

$$\mathbf{w} = \mathbf{w} - \alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$

TD policy evaluation with function approximation

Algorithm

1. Initialize randomly parameters $\mathbf{w}$, set $t = 1$
2. Loop until convergence
   a. Use policy $\pi(a|s)$ to sample s_t, a_t, r_t, s_{t+1}
   b. Compute gradient
   $$\nabla_{\mathbf{w}} J(\mathbf{w}) = (r_t + \gamma V_{\mathbf{w}}(s_{t+1}) - V_{\mathbf{w}}(s_t))\nabla_{\mathbf{w}} V_{\mathbf{w}}(s_t)$$
   c. Update parameters with gradient descent step
   $$\mathbf{w} = \mathbf{w} - \alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$
   d. Set $t = t + 1$

Remarks

- TD learning can be applied to non-episodic problems
- For single step true $V^{\pi}(s_t)$ is approximated by $r_t + \gamma V_{\mathbf{w}}(s_{t+1})$

State-action value function approximation

Principle

- Approximate state-action value function $Q^{\pi}(s,a)$ by a parameterized function $Q_{\mathbf{w}}(s,a)$
- Approximation is done according to square loss :

$$J(\mathbf{w}) = \frac{1}{2}\mathbb{E}^{\pi}[(Q^{\pi}(s,a) - Q_{\mathbf{w}}(s,a))^2]$$

- Loss gradient :

$$J(\mathbf{w}) = \mathbb{E}^{\pi}[(Q^{\pi}(s,a) - Q_{\mathbf{w}}(s,a))\nabla_{\mathbf{w}} Q_{\mathbf{w}}(s,a)]$$

Update rule for state-action value function approximation

Update rule for Monte Carlo approximation :

From a sampled episode :

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = (G_t - Q_{\mathbf{w}}(s,a))\nabla_{\mathbf{w}} Q_{\mathbf{w}}(s,a)$$

Update rule for Q-learning :

From one step sampling :

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = (r + \gamma \max_{a'} Q_{\mathbf{w}}(s',a') - Q_{\mathbf{w}}(s,a))\nabla_{\mathbf{w}} Q_{\mathbf{w}}(s,a)$$

Overview

Labs : Q-learning based trading agent

- Lab session with Pr. Anindya ROY