Lecture 19: Exploration and Exploitation

Bolei Zhou

The Chinese University of Hong Kong

bzhou@ie.cuhk.edu.hk

April 1, 2020

Outline

1. Introduction on exploration and exploitation
2. Multi-armed bandits
    1. Greedy and ϵ-greedy algorithms
    2. Temperature in Softmax bandit algorithm
    3. Upper Confidence Bound (UCB) algorithm
    4. Thompson sampling
3. Other exploration strategies in RL
    1. Entropy
    2. Curiosity
    3. Learning from failure

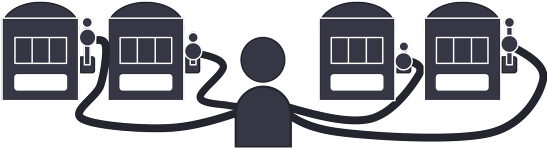
Trade-off between Exploration and Exploitation

1. Decision making invovles a fundamental choice:
   Exploitation: Choose the best known action
   Exploration: Explore some unknown action
2. Short-term reward v.s. long-term reward: To collect information about action which results to long-term reward may involve the sacrifice in short-term reward.
3. Collect enough information to make the best overall decisions

Examples

1. Going restaurant
    1. Exploitation: Go to your favorite restaurant
    2. Exploration: Try a new restaurant
2. Oil Drilling
    1. Exploitation: Drill at the best known location
    2. Exploration: Drill at a new location
3. Webpage design
    1. Exploitation: Copy some existing template
    2. Exploration: Design your own from scratch
4. Game playing
    1. Exploitation: Play the move you already knows to work
    2. Exploration: Do some experimental move
5. New Year Resolution
    1. Exploitation: Stay in your comfort zone
    2. Exploration: Try some new thing

k-Armed Bandit



1. A multi-armed bandit is a tuple $< \mathcal{A}, \mathcal{R} >$
2. k actions to take at each step t
3. $\mathcal{R}^a(r) = P(r|a)$ is unknown probability distribution over rewards
4. At each step t the agent selects an action $a_t \in \mathcal{A}$, then the environment generates a reward $r_t \sim \mathcal{R}^{a_t}$
5. The goal of agent is to maximize cumulative reward $\sum_{\tau=1}^{T} r_\tau$

Bernoulli Arm and Normal Arm

```
class BernoulliArm():
  def __init__(self, p):
    self.p = p
  def draw(self):
    if random.random() > self.p:
      return 0.0
    else:
      return 1.0

class NormalArm():
  def __init__(self, mu, sigma):
    self.mu = mu
    self.sigma = sigma
  def draw(self):
    return random.gauss(self.mu, self.sigma)
```

Definition of Value Function and Action-Value Function

1. The action-value is the mean reward for action a (unknown)

$$Q(a) = \mathbb{E}(r|a) \tag{1}$$

2. The optimal value

$$V^* = Q(a^*) = \max_{a \in \mathcal{A}} Q(a) \tag{2}$$

3. To estimate $Q(a)$, we can let $Q(a)$ at step t

$$Q_t(a) = \frac{\text{sum of rewards when a taken prior to t}}{\text{number of times a taken prior to t}} = \frac{\sum_{i=1}^{t-1} R_i \cdot 1_{A_i=a}}{\sum_{i=1}^{t-1} 1_{A_i=a}} \tag{3}$$

Greedy Action and ϵ-Greedy Action to Take

1. The estimation of $Q(a)$ at step t

$$Q_t(a) = \frac{\text{sum of rewards when a taken prior to t}}{\text{number of times a taken prior to t}} = \frac{\sum_{i=1}^{t-1} R_i \cdot 1_{A_i=a}}{\sum_{i=1}^{t-1} 1_{A_i=a}} \tag{4}$$

2. Greedy action selection algorithm: $A_t = \arg\max_a Q_t(a)$
3. Problem with the greedy algorithm?
4. ϵ-Greedy: greedy most of the time, but with small probability ϵ select random actions (ϵ is usually as 0.1)
    1. probability $1 - \epsilon : A_t = \arg\max_a Q_t(a)$
    2. probability $\epsilon : A_t = \mathit{uniform}(\mathcal{A})$

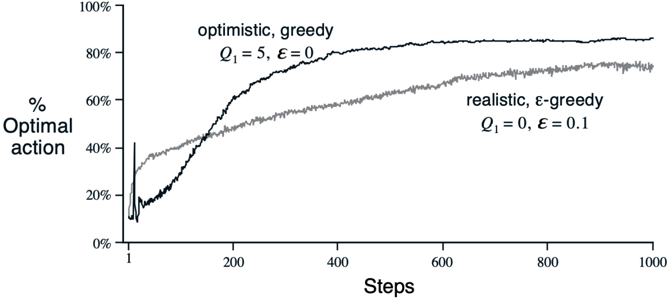
ϵ-Greedy Algorithm

Algorithm 1 Simple *epsilon*-Greedy bandit algorithm

for a = 1 to k **do**
  $Q(a) = 0, N(a) = 0$
end for
loop
  $$A = \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\epsilon \\ uniform(\mathcal{A}) & \text{with probability } \epsilon \end{cases}$$
  $R = bandit(A)$
  $N(A) = N(A) + 1$
  $Q(A) = Q(A) + \frac{1}{N(A)}[R - Q(A)]$
end loop

Optimistic Initial Values

1. Simple idea: initialize Q(a) to high value
2. Encourage the exploration over all possible actions early on



Softmax Bandit Algorithm

1. To learn a numerical preference for each action a (like learning a policy function, denoted as $H_t(a)$,

$$\pi_t(A_t) = P(A_t = a) = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \tag{5}$$

2. Learning based on the idea of stochastic gradient descend

$$\text{For } A_t, H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t)). \tag{6}$$

$$\text{For all } a \neq A_t, H_{t+1}(a) = H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a). \tag{7}$$

   here $\bar{R}_t$ is the average of all the rewards up to time t. Page 38 contains the full derivation.

3. Learning based on the incremental estimation

$$H_t(A_t) = H_{t-1}(A_t) + \frac{1}{N_t(A_t)}[R - H_{t-1}(A_t)] \tag{8}$$

Temperature

1. Scaling factor, temperature τ, to control the degree of exploration. High temperature, atoms will behave more random.

$$P(A_t = a) = \frac{e^{H_t(a)/\tau}}{\sum_{b=1}^{k} e^{H_t(b)/\tau}} \tag{9}$$

Annealing

1. Annealing is the process of modifying an algorithm's behavior so that it will explore less over time
2. Effect to different algorithms:
   1. To reduce ϵ in ϵ-Greedy algorithm
   2. To make the temperature go lower and lower in Softmax Bandit algorithm

UCB - The Upper Confidence Bound Algorithm

1. Both the ϵ-Greedy algorithm and the Softmax algorithm share the following broad properties:
    1. select the arm that currently has the highest estimated value
    2. explore and choose an arm that isn't the one that currently seems the best
    3. reduce the exploration by annealing (dynamically change some parameters ϵ and τ
2. UCB takes a very different approach.
    1. UCB does not use randomness at all
    2. UCB doesn't have any free parameters to configure before you can deploy it.

UCB - The Upper Confidence Bound Algorithm

1. $U_t(a)$ is the upper confidence bound of the reward value, so that the true value is below the bound with **high probability**,

$$Q(a) \leq Q_t(a) + U_t(a) \tag{10}$$

2. The upper bound $U_t(a)$ is a function of $N_t(a)$, where a larger number of trials $N_t(a)$ should give us a smaller bound $N_t(a)$ (less uncertain).
3. In UCB1 algorithm,

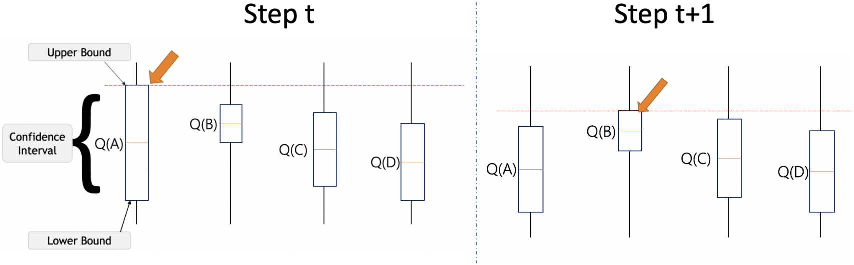
$$U_t(a) = \sqrt{\frac{2 \log t}{N_t(a)}} \tag{11}$$

4. Thus the action is selected as to maximize the UCB

$$a_t = \arg\max_a [Q_t(a) + \sqrt{\frac{2 \log t}{N_t(a)}}] \tag{12}$$

UCB - The Upper Confidence Bound Algorithm

1. If we are uncertain about an action, we should optimistically assume that it is the correct action.



UCB - The Upper Confidence Bound Algorithm

$$a_t = \arg\max_a [Q_t(a) + \sqrt{\frac{2 \log t}{N_t(a)}}] \tag{13}$$

1. Upper bound term brings significant values from the beginning.
2. UCB is an explicitly curiosity-driven algorithm that tries to seek out the unknown
3. Square root term is a measure of the uncertainty or variance in the estimate of a's value. Each time a is selected, the uncertainty term decreases.
4. Logarithm increases get smaller over time.

Deriving the Upper Confidence Bound

Hoeffding's Inequality

Let $X_1, ..., X_n$ be i.i.d random variables and they are all bounded by the interval $[0, 1]$. The sample mean is $\bar{X}_n = \frac{1}{n}\sum_{\tau=1}^{n} X_\tau$. Then for $u > 0$, we have:

$$P(\mathrm{E}[X] > \bar{X}_n + u) \leq e^{-2nu^2} \tag{14}$$

1. Following the Hoeffding's Inequality, then we have

$$P(Q(a) > Q_t(a) + U_t(a)) \leq e^{-2N_t(a)U_t(a)^2}$$

Deriving the Upper Confidence Bound

1. We want to pick up a bound so that with high chances the true mean is below the sample mean + the upper confidence bound,

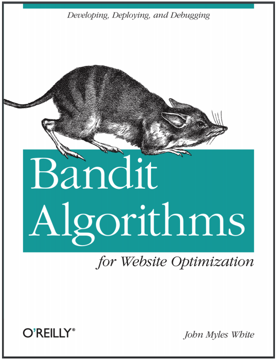
$$P(Q(a) > Q_t(a) + U_t(a)) \leq e^{-2N_t(a)U_t(a)^2} = p,$$

$$\text{thus, } U_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}}$$

2. One heuristic is to reduce the threshold p in time, as we want to make more confident bound estimation with more rewards observed. Set $p = t^{-4}$ we get UCB1 algorithm:

$$a_t^{UCB1} = \arg\max_a [Q_t(a) + \sqrt{\frac{2\log t}{N_t(a)}}] \qquad (15)$$

Example Code for Bandit Algorithms



- Bandit Algorithms for Website Optimization by John Myles White.
- `https://github.com/cuhkrlcourse/RLexample/tree/master/bandits`

Thompson Sampling: Bayesian decision making

1. An algorithm for online decision problems where actions are taken sequentially
2. Bayesian inference to compute the posterior with the known prior and the likelihood of getting the sampled data

Algorithm 1 BernGreedy(K, α, β)

for $t = 1, 2, \ldots$ **do**
  #estimate model:
  for $k = 1, \ldots, K$ **do**
    $\hat{\theta}_k \leftarrow \alpha_k / (\alpha_k + \beta_k)$
  end for

  #select and apply action:
  $x_t \leftarrow \mathrm{argmax}_k \, \hat{\theta}_k$
  Apply x_t and observe r_t

  #update distribution:
  $(\alpha_{x_t}, \beta_{x_t}) \leftarrow (\alpha_{x_t}, \beta_{x_t}) + (r_t, 1 - r_t)$
end for

Algorithm 2 BernThompson(K, α, β)

for $t = 1, 2, \ldots$ **do**
  #sample model:
  for $k = 1, \ldots, K$ **do**
    Sample $\hat{\theta}_k \sim \mathrm{beta}(\alpha_k, \beta_k)$
  end for

  #select and apply action:
  $x_t \leftarrow \mathrm{argmax}_k \, \hat{\theta}_k$
  Apply x_t and observe r_t

  #update distribution:
  $(\alpha_{x_t}, \beta_{x_t}) \leftarrow (\alpha_{x_t}, \beta_{x_t}) + (r_t, 1 - r_t)$
end for

3. More detailed tutorial: `https://arxiv.org/pdf/1707.02038.pdf`

Other Exploration Strategies in RL

1. Entropy-regularized policy optimization
2. Learning from internal rewards: Curiosity-driven exploration
3. Learning from failures: Hindsight Experience Replay

Entropy-regularized policy optimization

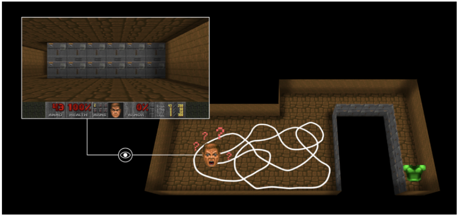
1. SAC incorporates **entropy regularization**
2. Entropy is a quantity which measures how random a random variable is, $H(P) = E_{x \sim P}[-\log P(x)]$
3. Entropy-regularized RL: the policy is trained to maximize a trade-off between expected return and entropy, a measure of randomness in the policy

$$\pi^* = \arg\max E_{\tau \sim \pi}[\sum_t \gamma^t \big(R(s_t, a_t, s_{t+1}) + \alpha H(\pi(.|s_t))\big)]$$

Curiosity-driven Learning

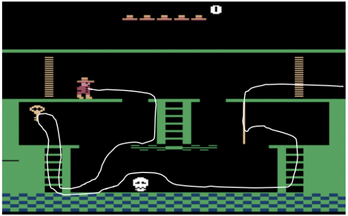
1. Environments with sparse rewards or non-existing rewards are very challenging

Vizdoom "DoomMyWayHome"



From Felix Steger

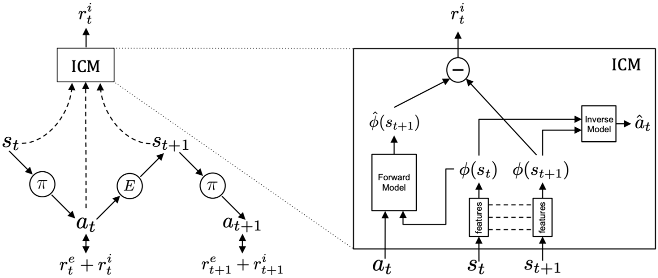
Montezuma's Revenge



2. A new reward function: curiosity
    1. Curiosity is an intrinsic reward which can be considered as the error of the agent to predict the consequence of its own actions given its current state
    2. Encourage the agent to perform actions that reduce the uncertainty in the agent's ability to predict the consequence of its own action
    3. How to measure the curiosity

Curiosity-driven Learning

1. Pathak ICML'17 Curiosity-driven Exploration by self-supervised prediction: Intrinsic Curiosity Module (ICM)



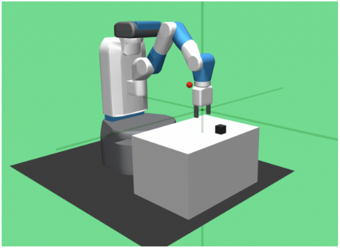
1. Forward model is to predict the transition dynamics (world model): $\hat{\phi}_{s_{t+1}} = f(\phi(s_t), a_t; \theta_f)$
2. Inverse Model is trained to predict the action a given s_t and s_{t+1}: $\hat{a} = g(\phi(s_t), \phi(s_{t+1}); \theta_g)$
3. Intrinsic reward is defined as residual $r_t^i = ||\hat{\phi}(s_{t+1}) - \phi(s_{t+1})||$

Curiosity-driven Learning

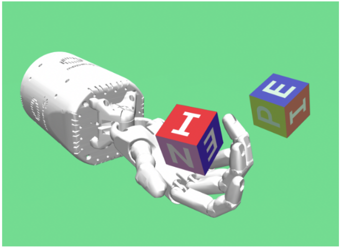
1. Pathak ICML'17 Curiosity-driven Exploration by self-supervised prediction: Intrinsic Curiosity Module (ICM)
2. Demo: `https://pathak22.github.io/noreward-rl/`
3. Further work: Large-Scale Study of Curiosity-Driven Learning. ICLR'19: `https://pathak22.github.io/large-scale-curiosity/`

Goal-oriented Environments

Pick and place at a desired goal



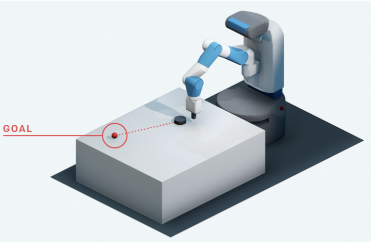
Manipulate block to a desired goal

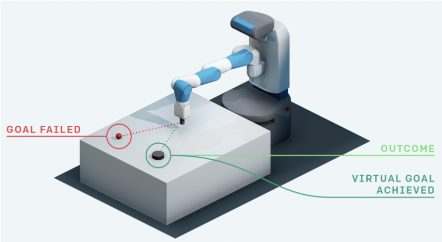


1. The desired goal might appear at any state
2. The reward is very sparse

Goal-oriented Environments

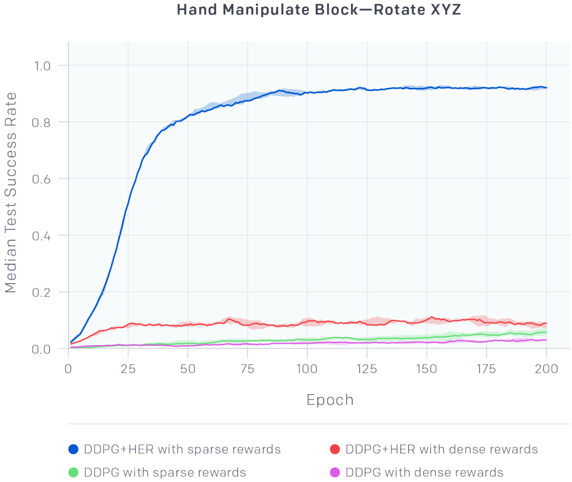
1. NIPS'17 Hindsight Experience Replay (HER): Learning from failure





   1. Intuitive idea: Even though we have not succeeded at a specific goal, we have at least achieved a different one.
   2. So we can just pretend that we wanted to achieve this goal to begin with, instead of the original one
   3. By doing this substitution, the reinforcement learning algorithm can obtain a learning signal since it has achieved some goal (**so we create some dense reward**)

HER for Goal-oriented Environments



Summary for Exploration in RL

1. It is beneficial to create extra **external rewards** and **internal rewards** for the agent to learn