Multi-Arm Bandits

Easwar Subramanian

TCS Innovation Labs, Hyderabad

Email : cs5500.2020@iith.ac.in

November 09, 2024

Overview

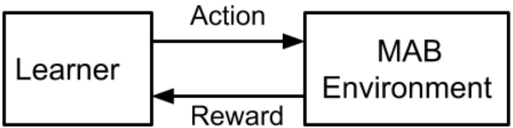
1. Introduction to Bandit Problem
2. Naive Approaches
3. Optimism in the Face of Uncertainty
4. Thompson Sampling

Introduction to Bandit Problem

Multi Arm Bandit

- **Learning Problem :** Which arm is the best ?
- **Decision Problem :** Which arm to pull next ?

Multi Arm Bandit : Setting

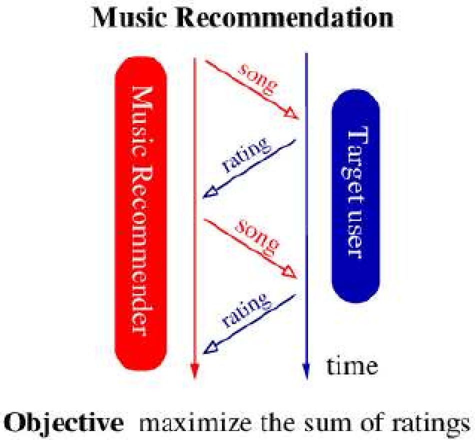


- There are K arms to pull and there are N rounds
- The agent can pull any of the K arms in each round $t \in \{0, 1, \cdots, N\}$
- On pulling arm a, the agent gets a random reward r_a sampled from a distribution (independent of previous choices and rewards)
- Goal
  - ★ **Regret minimization :** : Maximize the sum of rewarwds (in expectation)
  - ★ **Best Arm Identification :** Discover the best arm (given some budget)

Bandit Problems – Motivations

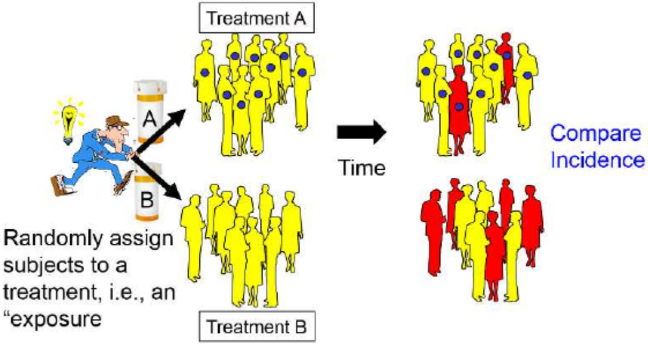
- Managing exploration-exploitation trade-off
- Baby reinforcement learning
- Lots of appliations in online learning

Example : Music Recommendation



- Decision Problem : Which song to play next ?

Example : Clinical Trial



- Decision Problem : Which treatment to be given for next patient ? Or Which line of treatment is best ?

Example : Dynamic Pricing

- Seller has a commodity to sell in market
- N possible discrete prices
- Observation : Sale or no sale for offered price
- Explore different prices or pick the best performing prices so far

Other Applications

- Ad Placement
- A/B Testing
- Network Routing
- Game Tree Search

Multi Arm Bandit : Formulation

- A multi-arm bandit is defined as a tuple $< \mathcal{A}, \mathcal{R} >$
- $< \mathcal{A} >$ is the set of arms available
- $\mathcal{R}^a(r) = \mathbb{P}(r|a)$ is the unknown of distribution of rewards of arm a
- At each step t the agent selects an action $a_t \in \mathcal{A}$ and gets a reward $r_t \sim \mathcal{R}^{a_t}$
- The goal is to maximise cumulative reward $\sum_{t=1}^{N} r_t$

Regret Minimization

- The goal is to maximize cumulative reward $\sum_{t=1}^{N} r_t$
- Define the action value function $Q(a)$ to be the mean reward for action a i.e. $Q(a) = \mathbb{E}(r|a)$
- The optimal value V^* is

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

- The regret is the lost opportunity at one step

$$l_t = \mathbb{E}[V^* - r_t] = V^* - \mathbb{E}[r_t]$$

- Total regret is the total opportunity loss

$$L_N = \mathbb{E}\left[\sum_{t=1}^{N} (V^* - r_t)\right] = NV^* - \mathbb{E}\left[\sum_{t=1}^{N} r_t\right]$$

- Maximize cumulative reward $\equiv$ Minimize total regret

Regret Minimization : Alternative Formulation

- Let the **count** $N_t(a)$ be the number of times arm a is pulled upto time t $(N_t(a) \equiv \sum_{\tau=1}^{t} \mathbb{1}_{A_\tau = a})$
- Let Δ_a be the **gap** between optimal reward (from optimal action a^*) and reward of arm a

$$\Delta_a = V^* - Q(a)$$

- Regret is a function of gaps and counts given as

$$\begin{aligned} L_N &= \mathbb{E}\left[\sum_{t=1}^{N} (V^* - r_t)\right] = \mathbb{E}\left[\sum_{t=1}^{N} \Delta_{A_t}\right] \\ &= \mathbb{E}\left[\sum_{t=1}^{N} \sum_{a \in \mathcal{A}} \mathbb{1}_{A_t = a}\, \Delta_a\right] = \sum_{a \in \mathcal{A}} \Delta_a\, \mathbb{E}\left(N_N(a)\right) \end{aligned}$$

- A good algorithm ensures small counts for large gaps

On Estimating Mean Rewards

- We consider algorithms that estimate $\hat{Q}_t(a) \approx Q(a)$
- The sample estimate $\hat{Q}(a)$ is estimated via Monte-Carlo simulations

$$\hat{Q}_t(a) = \frac{1}{N_t(a)} \sum_{\tau=1}^{t} r_\tau \mathbb{1}(A_\tau = a)$$

Naive Approaches

Greedy Algorithm

- At any time t, a greedy algorithm selects the action with highest $\hat{Q}_t(a)$, i.e.

$$a_t^* = \arg\max_{a \in \mathcal{A}} \hat{Q}_t(a)$$

- Greedy algorithm can lock into a sub-optimal arm forever
- $\implies$ Greedy has linear total regret

Explore Then Commit

Algorithm Explore then Commit

1: Let K be the number of arms; N be the total rounds; Initialize M
2: **for** $m = 1, 2, \cdots M$ **do**
3:     **for** $a = 1, 2, \cdots, K$ **do**
4:         Pull arm a; Observe reward r_a; Compute mean reward $\hat{Q}(a)$ for arm a;
5:     **end for**
6: **end for**
7: **for** $i = MK + 1, \cdots, N$ **do**
8:     Pull the arm with the best mean reward [i.e. $a^* = \arg\max_a \hat{Q}(a)$]
9: **end for**

Question : Which parts of the algorithm explores and which part exploits ?

Explore Then Commit

Algorithm Explore then Commit

Let K be the number of arms; N be the total rounds; Initialize M

Exploration Phase

for $m = 1, 2, \cdots M$ **do**
    for $a = 1, 2, \cdots, K$ **do**
        Pull arm a; Observe reward r_a; Compute mean reward $\hat{Q}(a)$ for arm a;
    end for
end for

Exploitation Phase

for $t = MK + 1, \cdots, N$ **do**
    Pull the arm with the best mean reward [i.e. $a^* = \arg\max_a \hat{Q}(a)$]
end for

Question : Why do we expect this algorithm to work ?

Law of Large Numbers

- Suppose $X_1, X_2, \cdots$ are independent samples of a random variable X having mean μ
- Denote empirical mean of m samples by $\hat{\mu}_m$ defined as

$$\hat{\mu}_m = \frac{1}{m}\sum_{i=1}^{m} X_i$$

- Weak law of large numbers states that $\hat{\mu}_m \to \mu$ in probability as $m \to \infty$
- Strong law of large numbers states that $\hat{\mu}_m \to \mu$ almost surely as $m \to \infty$

Explore and Commit : Analysis

- At round m, upon pulling arm a, the agent gets a random reward $r_m^a \sim \mathcal{R}^a$
- After M rounds, we have $\hat{Q}(a)$ as the empirical mean reward for pulling arm a

$$\hat{Q}(a) = \frac{1}{m}\sum_{i=1}^{m} r_m^a$$

$$\hat{Q}(a) \rightarrow Q(a)$$

as the number of rounds gets large

Question : Is there a shortcoming to ETC ?

ETC does not use the experience generated after the initial explore phase

Greedy Approach

Algorithm Greedy Algorithm

Let K be the number of arms; N be the total rounds; Initialize M

for $m = 1, 2, \cdots M$ **do**

  for $a = 1, 2, \cdots, K$ **do**

    Pull arm a; Observe reward r_a; Compute mean reward $\hat{Q}(a)$ for arm a;

  end for

end for

for $t = MK + 1, \cdots, N$ **do**

  Pull the arm with the **current** best mean reward [i.e. $a^* = \arg\max_a \hat{Q}(a)$]

  Update the mean observed rewards with the latest observation

end for

Question : Will this work well ? Can we improve exploration ?

The greedy algorithm is unlikely to explore during the exploitation phase

The ϵ - Greedy Approach

Algorithm ϵ - Greedy Algorithm

```
Let K be the number of arms; N be the total rounds; Initialize M and choose ε ∈ (0,1) small
for m = 1, 2, ··· M  do
    for a = 1, 2, ··· , K do
        Pull arm k; Observe reward r_a; Compute mean reward Q̂(a) for arm a;
    end for
end for
for t = MK + 1, ··· , N do
    With probability 1 − ε, pull the arm with the current best mean reward [i.e. a* =
    arg max_a Q̂(a)], else play another arm uniformly at random
    Update the mean observed rewards with the latest observation
end for
```

Quesiton : Do you see possible drawback ?

The ϵ-greedy algorithm explores forever. Also, has total linear regret.

Optimistic Initialization

- **Idea** : Initialise $Q(a)$ for all actions to high value
- Update action value by incremental Monte-Carlo evaluation; Let $a \in \mathcal{A}$ be the arm pulled at round t, Then,

$$\hat{Q}_t(a) = \hat{Q}_{t-1}(a) + \frac{1}{N_t(a)}\left(r_t - \hat{Q}_{t-1}\right)$$

  where $r_t \sim \mathcal{R}^a$ is the reward obtained at round t
- Encourages systematic exploration early on
- Locking onto sub-optimal arm is a possibility
- Greedy + optimistic initialization has linear total regret
- ϵ - Greedy + optimistic initialization has linear total regret

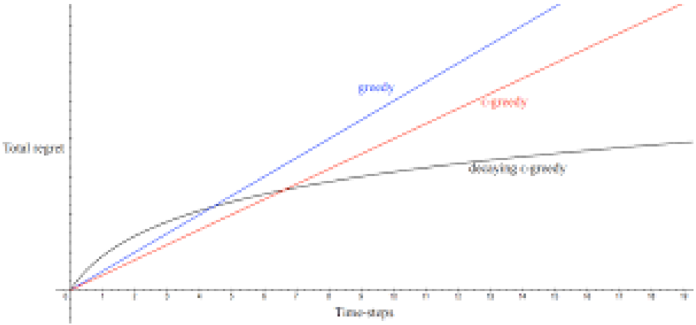
The ϵ - Greedy with Decay Approach

Algorithm ϵ - Greedy with Decay Algorithm

```
Let K be the number of arms; N be the total rounds; Initialize M and choose ε ∈ (0,1) small
  and choose a small decay rate r ∈ (0,1)
for m = 1, 2, ··· M do
    for a = 1, 2, ··· , K do
        Pull arm a; Observe reward r_a; Compute mean reward Q̂(a) for arm a;
    end for
end for
for t = MK + 1, ··· , N do
    With probability 1 − ε, pull the arm with the current best mean reward [i.e. a* =
      arg max_a Q̂(a)], else play another arm uniformly at random
    Update the mean observed rewards with the latest observation
    Reduce ε by fraction r
end for
```

Certain choices of decay schedule can achieve lograthmic asymptotic total regret

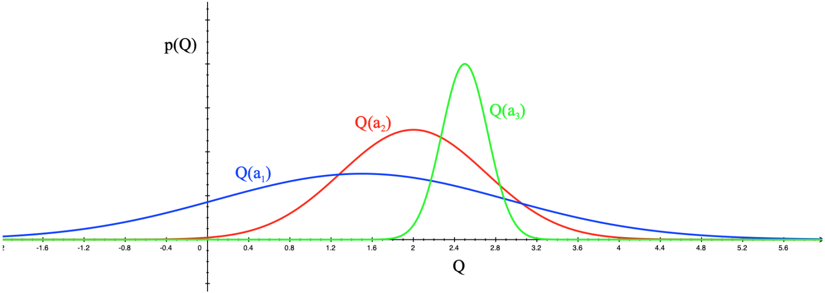
Linear or Sub-Linear Regret



- Algorithms that explore forever have total linear regret
- Algorithms that never explore have total linear regret
- **Question** : Is it possible for develop algorithms have sub-linear regret ?

Optimism in the Face of Uncertainty

Optimism in the Face of Uncertainty



- Which arm (among the three) should we choose at next round ?
- **Optimism in the Face of Uncertainty** $\implies$ pick the arm that we are most uncertain about
- The more uncertain we are about the action-value of an arm, the more we should explore that action; as it could turn out to be the best action

Upper Confidence Bound

- Estimate an upper confidence $\hat{U}_t(a)$ for action a at time t such that

$$Q(a) \leq \hat{Q}_t(a) + \hat{U}_t(a)$$

- The upper confidence bound depends on the number of times an arm a has been pulled so far
  - ★ Small $N_t(a) \implies$ Large $\hat{U}_t(a)$
  - ★ Large $N_t(a) \implies$ Small $\hat{U}_t(a)$
- Select action a, at time t, that maximizes

$$a_t = \arg\max_a \left[\hat{Q}_{t-1}(a) + \hat{U}_{t-1}(a)\right]$$

- Hoeffding's inequality provides a way to arrive at the formulation for $\hat{U}_t(a)$

Hoeffding's Inequality

Theorem

Let $X_1, \ldots, X_t$ be i.i.d. (independent and identically distributed) random variables and they are all bounded by the interval $[0, 1]$. The sample mean is $\overline{X}_t = \frac{1}{t}\sum_{\tau=1}^{t} X_\tau$. Then for $u > 0$, we have,

$$\mathbb{P}[\mathbb{E}[X] > \overline{X}_t + u] \leq e^{-2tu^2}$$

- We will apply Hoeffding's inequality to the rewards of the bandit

$$\mathbb{P}[Q(a) > \hat{Q}_t(a) + \hat{U}_t(a)] \leq e^{-2N_t(a)\hat{U}_t(a)^2}$$

Calculating Upper Confidence Bound

- Pick a probability p that true value exceeds UCB
- Now solve for $\hat{U}_t(a)$ by setting

$$p = e^{-2N_t(a)\hat{U}_t(a)^2}$$

then,

$$\hat{U}_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}}$$

- Reduce p as t^{-4} as we observe more rewards
- Ensures optimal action selection asymptotically (as $t \to \infty$)

$$\hat{U}_t(a) = \sqrt{\frac{2\log t}{N_t(a)}}$$

UCB1 Algorithm

Algorithm UCB1 Algorithm

1: Let K be the number of arms;
2: **for** $a = 1, 2, \cdots, K$ **do**
3: Pull arm a; Observe reward r_a; Compute mean reward $\hat{Q}(a)$ for arm a;
4: **end for**
5: **for** $t = K + 1, \cdots, N$ **do**
6: Pull arm a such that

$$a_t = \arg\max_a \left[\underbrace{\hat{Q}_t(a)}_{\text{Exploitation}} + \underbrace{\sqrt{\frac{2\log t}{N_t(a)}}}_{\text{Exploration}} \right]$$

7: Update the mean observed rewards and UCB coefficient of the arm chosen
8: **end for**

Assumptions Matter

- So far we have made no assumptions about the reward distribution $\mathcal{R}$ (except bound on rewards)
- Neccessary to make assumptions; Strong assumptions, when made the right way, lead to better algorithms
- Examples :
  - ★ Bernoulli
  - ★ Gaussian with unknown mean and unit variance
  - ★ Many more ...

Bayesian Bandits

- So far we have made no assumptions about the reward distribution $\mathcal{R}$ (except bound on rewards)
- Bayesian bandits exploit prior knowledge of rewards, $p[\mathcal{R}]$
- They compute posterior distribution of rewards $p[\mathcal{R}|h_t]$ where $h_t = \{a_1, r_1, \cdots, a_{t-1}, r_{t-1}\}$
- Use posterior to guide exploration (Bayesian UCB, probability matching)
- Better performance if prior knowledge is accurate

Bayesian UCB

- Assume reward distribution is Gaussian
  - ★ Reward of every arm is given by $\mathcal{N}(\mu_a, \sigma_a)$
- Upon pulling arm a, observe reward r_a ; Compute posterior using Baye's law
- Pick arm a that maximizes standard deviation of $\hat{Q}_t(a)$

$$a_t = \arg\max_a \left[\underbrace{\mu_{t,a}}_{\text{Exploitation}} + \underbrace{\sqrt{\frac{c\sigma_{t,a}}{N_t(a)}}}_{\text{Exploration}} \right]$$

Thompson Sampling

Bernoulli Bandits

- Consider a Bernoulli bandit
  - ★ Each one of the K machines has a probability θ_k of providing a reward to the player

Let us consider a single Bernoulli bandit with probability θ of obtaining a reward

- Suppose R be the random variable that denotes the outcome of pulling the arm of a bandit
  - ★ $\mathbb{P}(R = 1) = \theta$ and $\mathbb{P}(R = 0) = 1 - \theta$
  - ★ The probability mass function can be written as

$$\mathbb{P}(R = r) = \theta^r (1 - \theta)^{1-r}$$

  - ★ The expected reward after one round is given by $\mathbb{E}(R) = \theta$

Frequentist vs Bayesian Approach

Let $R_1, R_2, \cdots, R_n$ be outcomes of n rounds of pulling the bandit arm

- **Frequentist approach :** Estimate the fixed but unknown parameter θ using the average of $R_1, \cdots, R_n$ for large n
- **Bayesian approach :** Treat θ as an uncertain parameter, and estimate its distribution from the data $D_n = \{R_1, \cdots, R_n\}$ by computing the posterior distribution using Baye's formula

$$\mathbb{P}(\theta|D_n) = \frac{\mathbb{P}(D_n|\theta) \cdot \eta(\theta)}{\mathbb{P}(D)}$$

where $\eta(\theta)$ is a suitable prior distribution on θ

A suitable prior distribution for a Bernoulli bandit is uniform prior

Choice of Initial Prior

- Suppose we take a uniform prior, then,

$$\mathbb{P}(\theta|D_n) = \underbrace{c\theta^{S_n}(1-\theta)^{n-S_n}}_{\text{Beta Distribution}}$$

  with $S_n = R_1 + R_2 + \cdots + R_n$

- The posterior $c\theta^{S_n}(1-\theta)^{n-S_n}$ is of the form that resembles Beta distribution with parameters α and γ given by

$$\beta_{\alpha,\gamma}(\theta) = \frac{\Gamma(\alpha+\gamma)}{\Gamma(\alpha)\Gamma(\gamma)}\theta^{\alpha-1}\cdot(1-\theta)^{\gamma-1}$$

- Note that $\beta_{1,1}$ is a uniform distribution
- Initialize the Beta parameters α and β such that prior is uniform
  - ★ $\alpha = 1$ and $\gamma = 1$; we expect the reward probability to be 50% (uniform prior)
  - ★ $\alpha = 9000$ and $\gamma = 1000$; we strongly believe that the reward probability is 90% (not a recommended choice for prior)

Posterior Updates of Beta Distribution

- Assuming uniform prior, after n rounds, we have, $\theta | D_n \sim \beta_{\alpha_{n+1}, \gamma_{n+1}}$
- Recursive posterior updates :
  - ★ If $\theta | D_n \sim \beta_{\alpha_n, \gamma_n}$ then $\theta | D_{n+1} \sim \beta_{\alpha_{n+1}, \gamma_{n+1}}$ with

$$\begin{aligned} \alpha_{n+1} &= \alpha_n + R_{n+1} \\ \gamma_{n+1} &= \gamma_n + (1 - R_{n+1}) \end{aligned}$$

Thompson Sampling : Algorithm

Algorithm Thompson Sampling Algorithm

1: Let K be the number of arms;
2: **for** $t = 1, \cdots, N$ **do**
3:     **for** $a = 1, 2, \cdots K$ **do**
4:         Sample θ_t^a from its posterior; $\theta_t^a \sim \beta_{\alpha_t^a, \gamma_t^a}$
5:     **end for**
6:     Play the arm $a^* = \arg\max_a \theta_t^a$ and observe the reward R_t
7:     Update the posterior of the chosen arm by updating the parameters of the corresponding Beta distribution

$$\begin{aligned} \alpha_{t+1}^{a^*} &= \alpha_t^{a^*} + R_t \\ \gamma_{t+1}^{a^*} &= \gamma_t^{a^*} + (1 - R_t) \end{aligned}$$

8: **end for**

Closing Remarks

- The exploration techniques mentioned here can easily be extended to full reinforcement learning setting
- There are other variants of bandit problems that include **Best arm identification**, **PAC** and **Contextual Bandits**
  - ★ PAC : find an arm within ϵ of the best arm with probability at least $1 - \delta$
- Information state space approach involves modelling the arm selection problem as an MDP with state comprising of history (h_t) of past decisions and rewards. Subsequently, use model free RL or Bayesian RL to solve the MDP

Monte Carlo Tree Search

Easwar Subramanian

TCS Innovation Labs, Hyderabad

Email : cs5500.2020@iith.ac.in

Novemer 16, 2024

Overview

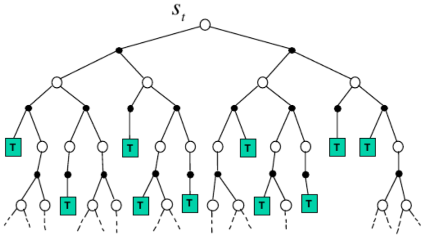
1. Introduction
2. On Truncated Tree Search
3. Naive Approach
4. Monte Carlo Tree Search
5. Derivative Free Methods

Introduction

Introduction

- We consider board games; Specifically, two player zero sum perfect information board games
  - ★ **Zero Sum :** Each participant's gain or loss is exactly balanced by the losses or gains of the other participant
  - ★ **Perfet Information :** No hidden information. During game-play every player can observe the whole game state.
- **Forward tree search** methods are popular to arrive at optimal moves in such board games
- Forward search algorithms select the best action by **lookahead**
- Lookahead is done using the model of the game MDP
- Apart from two player perfect games, tree search methods (such as MCTS) are used in situations where online planning using search is possible

Tree Search Methods : Framework



1. In most games, when described as MDP, there is no randomness in the environment; Moves are 'fullfilled'
2. Build a search tree with the current game position as the root
3. Compute value functions using simulated episodes
4. Select the next move to execute based on simulated epsiodes

Above framework is an example of online planning with search !!

Slide Credit: Katerina Fragkiadaki : CMU 10703

On the need for Online Learning

Question : Why can't value functions be learnt offline ?

- Environment has many states (Go : 10^{170}; Chess : 10^{48})
- Hard to compute a good value function for each one of them

Solution :

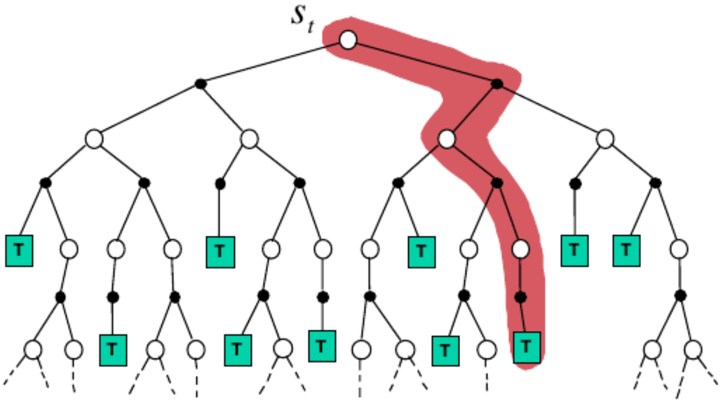
- Search tree is built with current game position and try to estimate the value function
- Solve the sub MDP ($\mathcal{M}^v$) starting from current game position
  - ★ Simulate episodes from current game position and apply model-free RL to simulated episodes

On Truncated Tree Search

Intelligent Vs Exhaustive Search

- The sub-MDP rooted at the current game position may still be very large
  - ★ More actions $\rightarrow$ Large Branching Factor
  - ★ More steps $\rightarrow$ Large Tree depth
- Reduce the breadth of the search by sampling actions from a policy $\pi(a|s)$ instead of trying every action
- Reduce depth of the search tree by position evaluation
  - ★ Truncate the search tree at state s and replacing the subtree below s by an approximate value function $V(s) = V^*(s)$ that predicts the outcome from state s

Intelligent Vs Exhaustive Search



Contrast with Minimax and Alpha-Beta pruning !!

 Figure Source: David Silver : UCL

Position Evaluation

- Engineer them using human experts (Example : DeepBlue !!)
  - ★ Replication across domain not possible
- Learn from self play

Naive Approach

Position/Action Evaluation using Monte Carlo

- Simulate K episodes of experience from the current board position with the model

$$\{s_t^k, a_t^k, r_{t+1}^k, s_{t+1}^k, a_{t+1}^k, r_{t+2}^k, \cdots, s_T^k\}_{k=1}^K \sim \mathcal{M}^v$$

- Apply model-free RL to the simulated episodes

State Value Function Evaluation : Monte Carlo

Algorithm Evaluate Given Board Position using MC

Let K be the number of simulations
Let s be the current state ; Initialize $w = 0$ and $l = 0$
for $k = 1, \cdots, K$ **do**
    $s' \leftarrow s$
    while s' is non-terminal **do**
        Choose an action a (using possibly a random policy) that is admissible from state s';
        Take action a from state s' and store next state in s'
    end while
    if game won **then**
        w++
    else
        l++
    end if
end for
Return $(w - l)/(w + l)$

Action Value Function Evaluation : Monte Carlo

- Given a model $\mathcal{M}^v$, current board position s_t and **simulation policy** π
- For each action $a \in \mathcal{A}$
  - ★ Simulate K episodes of experience from the current board position with the model

$$\{s_t^k, a_t^k, r_{t+1}^k, s_{t+1}^k, a_{t+1}^k, r_{t+2}^k, \cdots, s_T^k\}_{k=1}^K \sim \mathcal{M}^v, \pi$$

  - ★ Calculate accumulate total reward and use it to compute action value estimate

$$Q(s_t, a_t) = \frac{1}{K} \sum_{k=1}^{K} G_t$$

$$\frac{1}{K} \sum_{k=1}^{K} G_t \xrightarrow{P} Q^\pi(s_t, a_t)$$

- Select action with maximum Q value

$$a_t = \arg\max_a Q(s_t, a)$$

Monte Carlo Tree Search

Improvements to Simulation Policy

Question :
With more simulations, how can we improve the simulation policy ?

Answer :

- We can keep track of action values (Q) not only for the root but also for nodes internal to a tree we are expanding!
- How should we select the actions inside the tree ?
  - Use exploration algorithm(s) that we learnt in Bandit lectures
  - Specifically, we could use the variant of the UCB1 formula given by,

$$a_t = \arg\max_a \left[\underbrace{Q(s_t, a)}_{\text{Exploitation}} + \underbrace{c \cdot \sqrt{\frac{\log N}{n_a}}}_{\text{Exploration}} \right]$$

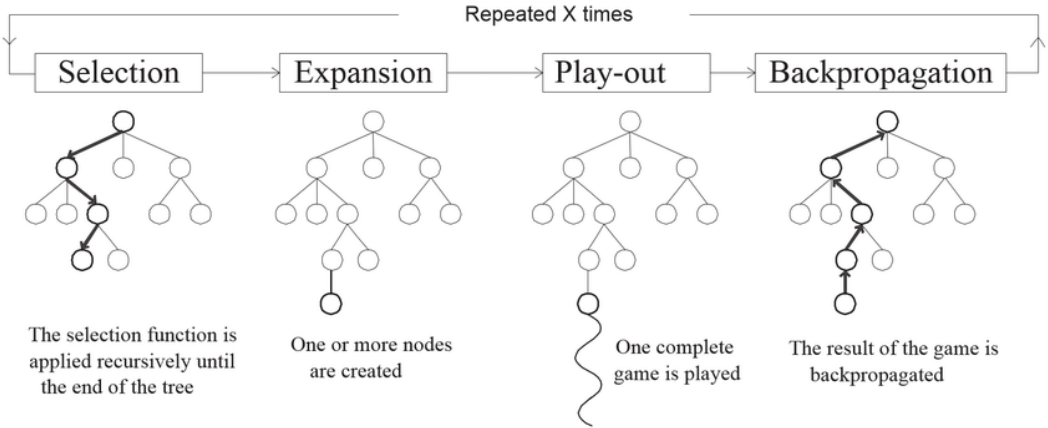
where N is the number of times the parent node is visited and n_a the number of times action a has been picked

Monte Carlo Tree Search

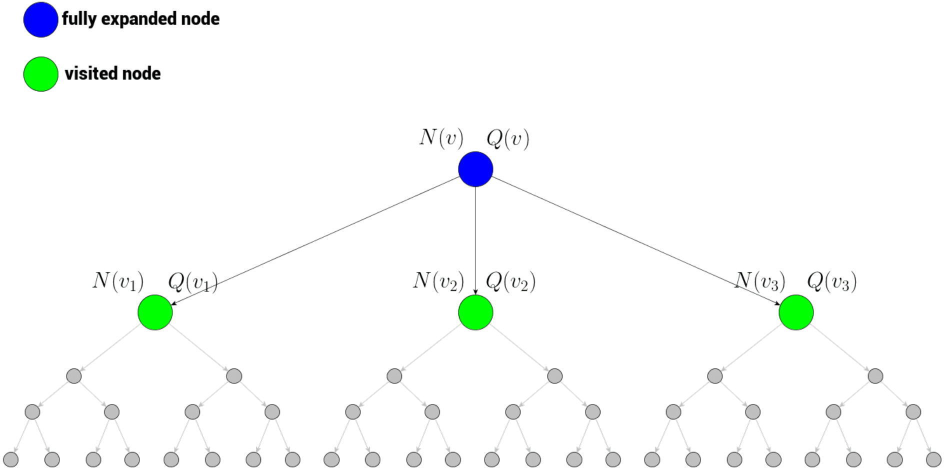
- Selection
  - Used for nodes we have seen before
  - Pick according to UCB
- Expansion
  - Used when we reach the frontier
  - Add one node per playout
- Simulation
  - Used beyond the search frontier
  - Don't bother with UCB, just play randomly
- Backpropagation
  - After reaching a terminal node
  - Update value and visits for states expanded in selection and expansion

Slide Credit: Katerina Fragkiadaki : CMU 10703

Monte Carlo Tree Search

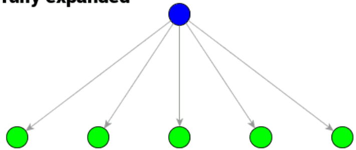


MCTS : Selection

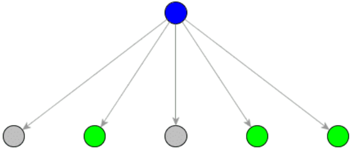


MCTS : Expansion

all children are marked visited - node is fully expanded

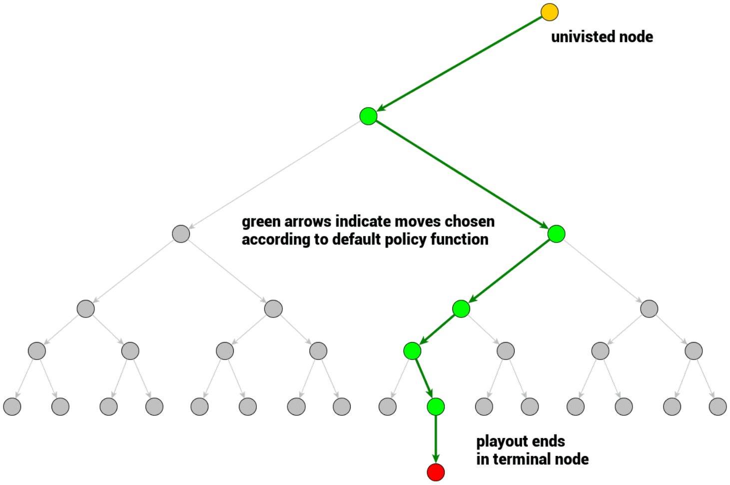


simulation/game state evaluation has been computed in all green nodes, they are marked visited

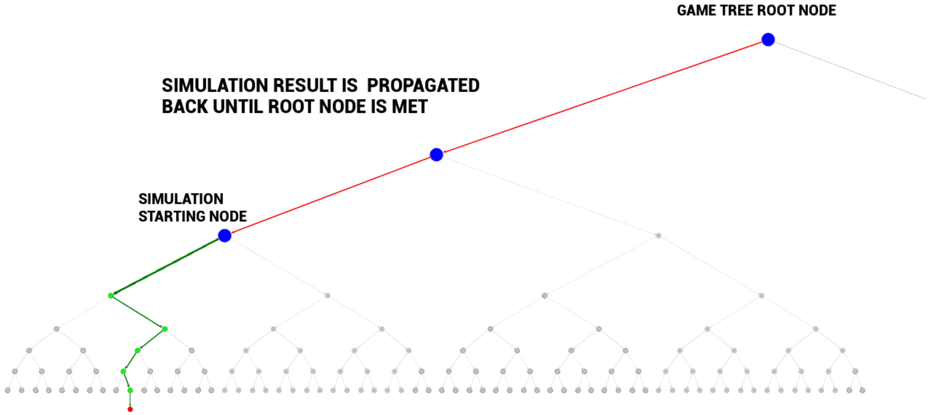


there are two nodes from where no single simulation has started - these nodes are unvisited, parent is not fully expanded

MCTS : Simulation



MCTS : BackPropagation



MCTS : Algorithm Sketch

Algorithm MCTS : Input 'node'

1: **for** $k = 1, \cdots, K$ **do**
2: leaf = TRAVERSE(node)
3: simresult = ROLLOUT(leaf)
4: BACKPROPAGATE(leaf, simresult)
5: **end for**
6: Return 'best' child of 'node'

Algorithm TRAVERSE : Input 'node'

1: **while** node is fully expanded **do**
2: node = SELECTION(node)
3: **end while**
4: **if** some children of node is not expanded **then**
5: node = RANDOMUNEXPANDEDCHILD(node)
6: **end if**
7: Return node

MCTS : Algorithm Sketch

Algorithm SELECTION : Input 'node'

for all children of node **do**

  $\text{UCB[child]} = \text{child.value} + C \cdot \sqrt{\frac{\log(\text{node.VISITS})}{\text{CHILD.VISITS}}}$

end for

Return child with maximum UCB[child]

Algorithm ROLLOUT : Input 'node'

if node is TERMINAL **then**

  Return result

else

  child = PICKRANDOM(node.children)

  Return RANDOMPLAYOUT(child)

end if

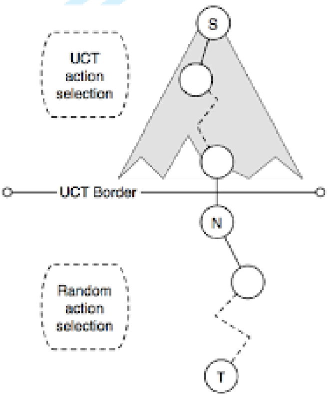
MCTS : Algorithm Sketch

Algorithm BACKPROPAGATE : Input 'node' and 'result'

```
1: if node is root then
2:     Return
3: else
4:     node.stats = result
5:     BACKPROPAGATE(node.parent)
6: end if
```

- The above pseudo-code is only a sketch. Please work out the details.
- For example, updating **'stats'** could involve incrementing number of visits to the node (needed for UCB computation) and augmenting the game results (win vs loss) from that node (needed to compute 'best' child)

Monte Carlo Tree Search



UCT (Upper confidence bound for Trees) based sampling of actions make the MCTS looks at more interesting moves more often

On Choice of Best Action

- How many simulations to run ?
  - ★ Time based : Run as long as you can
  - ★ Number based : Run K number of simulations
- When out of time, which move to play?
  - ★ Highest mean reward (highest probability to win)
  - ★ Highest UCB
  - ★ Most simulated move

AlphaGo : Successful Application of MCTS

- Value neural net to evaluate board positions
- Policy network to suggest actions
- Combine those networks with MCTS

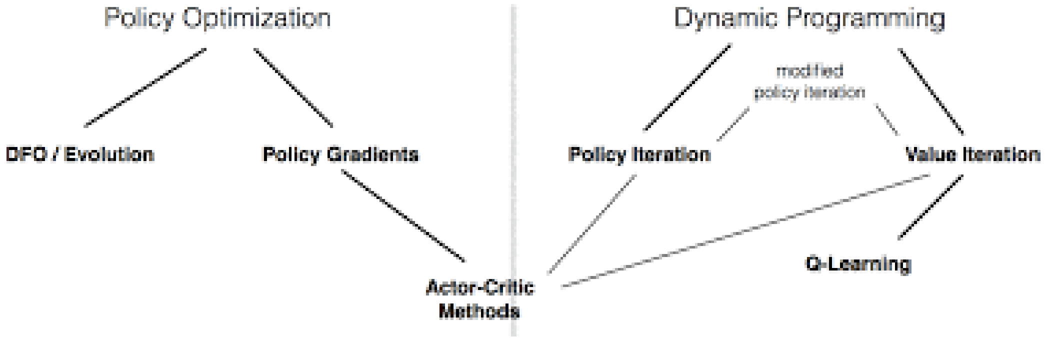
Slide Credit: Katerina Fragkiadaki : CMU 10703

MCTS : Strength and Impact

- One of the advantages of MCTS is its applicability to a variety of games, as it is domain independent
- Basis for extremely successful programs for games and many other applications
- Very general algorithm for decision making
- Anytime algorithm $\rightarrow$ can be stopped anytime, although with time results improve

Derivative Free Methods

RL Landscape



Evolutionary Methods

Goal of RL is to find a policy π_θ^* such that

$$\pi_\theta^* = \arg\max_{\theta} J(\theta) = \arg\max_{\pi_\theta} \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s\right]$$

General Algorithm

- Start with an initial parameter θ and construct a policy and evaluate $J(\theta)$
- Make some random changes to the parameter and evaluate $J(\theta)$
- If the result improves, keep the change
- Else **repeat**

Cross Entropy Method

Algorithm Cross Entropy Method

1: Initialize policy network π with parameters θ_1
2: **for** $i = 1$ to N **do**
3: Sample K parameters $\theta_{(i)}$ from a distribution $P_{\mu_i}(\theta)$
4: Execute roll-outs for each of the K parameters
5: Store $(\theta_i, J(\theta_i))$
6: Select the top $p\%$ of the parameters θ in terms of the utility $J(\theta)$
7: Fit a new distribution $P_{\mu_{i+1}}(\theta)$ from the top $p\%$
8: **end for**

- **Evolutionary** : The top $p\%$ of the parameter samples survive and the rest die. The top $p\%$ are then used arrive at the next generation of parameter samples
- **CMA-ES** : A popular variation that shrinks and expands the search area in the parameter space while fishing for parameters based on whether we are close to a good optima

Reinforcement Learning : Closing Notes

Easwar Subramanian

TCS Innovation Labs, Hyderabad

Email : cs5500.2020@iith.ac.in

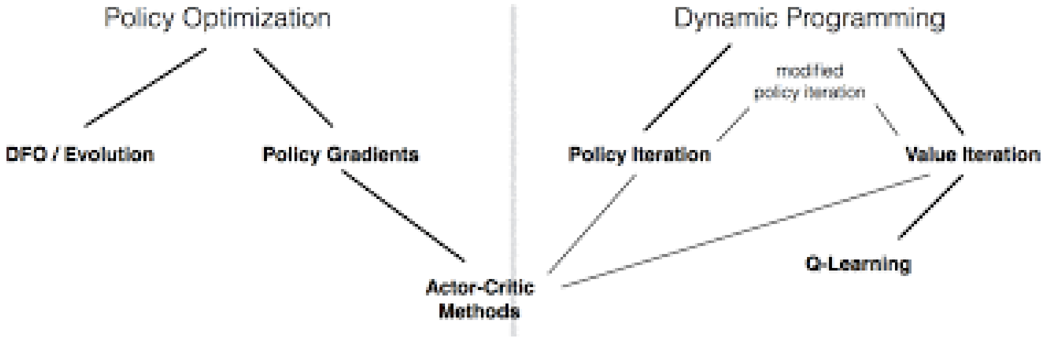
Novemer 16, 2024

Overview of this Lecture

1. Landscape, Summary and References
2. Other Topics
3. Practical Tips – Based on John Schulman's talk on Nuts and Bolts of Deep RL

Landscape, Summary and References

RL Landscape



Foundations of RL

Markov Property, transition probabilities, Markov reward process, Markov decision process

Three Key entities of RL

- Value Function - V
- Action Value Function - Q
- Policy - π

Optimal policies, notion of greedy policy, Bellman equations (evaluation and optimality)

Lecture Numbers : 2 and 3

Reference : David Silver's Lecture on RL

Value and Policy Iteration : Model Based Methods

Key Algorithms

- Value Iteration
- Policy Iteration

Drawbacks

- Requires full prior knowledge of the dynamics of the environment
- Can be implemented only on small, discrete state spaces

Lecture Number : 4

Reference : David Silver's Lecture on RL

Proofs on convergence available at : https://runzhe-yang.science/2017-10-04-contraction/

Model Free RL

Notion of bootstrap, lookahead and backup

Evaluation Algorithms

- Monte-Carlo methods (First Visit and Every Visit MC)
- Temporal difference methods
- TD-λ methods

Control Algorithms

- SARSA
- Watkin's Q-learning algorithm

Drawback : Not extendible to high dimensional state and action spaces

Lecture Numbers : 5 to 6

References : David Silver's Lecture on RL and Relevant Chapters on Sutton and Barto Book

Function Approximation and Q-Learning

Use of neural nets as function approximators, convergence of NN based algorithms

Algorithms

- Monte Carlo based value function estimation
- Fitted V iteration and Q iteration
- Deep Q networks

Lecture Number : 7 and 8

References : Deep RL course in Berkeley (2017,2018), Deep RL Bootcamp, Minh (2015), Riedmiller(2006)

Policy Gradient Techniques

Notion of Policy gradients, derivation of policy gradient expression, temporal structure, baseline and discounting for variance reduction, advantage function, deterministic policy gradient

Key Algorithms

- Actor-critic algorithms A2C and A3C
- DDPG

Lecture Numbers : 9 and 10

References : Deep RL course in Berkeley (2017,2018), Deep RL Bootcamp, Minh (2016), Lillicrap(2016)

Advanced Policy Gradient Techniques

Different approach to policy gradients by looking at distance in policy space; Surrogate loss function; Constrained policy optimization

Key Algorithms

- Natural Policy Gradient
- TRPO
- PPO

Lecture Number : 11

References : Deep RL course in Berkeley (2017,2018), Deep RL course in Berkeley, Joshua Aicham lecture by the same topic, NPG(Kakade, 2001), TRPO (Schulman2015) PPO (Schulman, 2017)

Stochastic Multi Arm Bandits and MCTS

Bandit Concepts :
Naive Exploration, Optimistic Initialization, Optimism in the face of Uncertainty

Key Algorithms

- UCB and Thompson Sampling
- Monte Carlo Tree Search (Tree search methods)

Lecture Number : 12 and 13

References : David Silver's Lecture on RL and Relevant Chapters on Sutton and Barto Book

References

- Reinforcement Learning : Sutton and Barto
- Dynamic Programming and Optimal Control (I and II) by Bertsekas
- Reinforcement Learning and Optimal Control, Bertsekas and Tsitsiklis
- David Silver's course on Reinforcement Learning
- Stanford course on Deep RL
- Deep RL BootCamp (Pieter Abeel)
- John Schulman's lectures in Policy Gradient Methods
- ... and many others

Other Topics

Model Based Reinforcement Learning

- **Central Question :** How can we make decisions better if we know system dynamics ? (possibly when state and action space is high dimensional or continuous)
  - ★ Games, navigating car etc, simulated environments
- If system dynamics in not known, can we identify them ?
  - ★ System identification - fit unknown parameters to a known model
  - ★ Learning - fit a general purpose model to observed transitions

Inverse Reinforcement Learning

Forward Reinforcement Learning

Given states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, reward function $\mathcal{R}(s, a)$ and possibly transition probabilities $P(s'|s, a)$ and

- Learn policy $\pi^*(a|s)$

Inverse Reinforcement Learning

Given states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, a policy $\pi(a|s)$ and possibly transition probabilities $P(s'|s, a)$ and

- Learn reward function $\mathcal{R}(s, a)$

and then use it learn $\pi^*(a|s)$

Transfer Learning in RL

- Forward transfer : train on one task, transfer to a new task
- Multi-task transfer : train on many task, transfer to a new task
- Multi-task meta learning : learn to learn from many tasks

Distributed RL

Question

How can we better utilize our computational resources to accelerate RL progress ?

Examples

- DQN and its variants (Large scale RL) (2013)
- GORILLA (2015)
- A3C (2016)
- IMPALA (2018)

Other Topics

- Topics like Hierarchical RL, Feudal RL etc
- Imitation Learning
- Partially Observable MDPs
- Multi-agent RL

OpenAI Gym

- Repository of multitude of environments
- Baseline implementation of several popular algorithms

Practical Tips – Based on John Schulman's talk on Nuts and Bolts of Deep RL

New Algorithms

- Test on small use cases and then run on medium-sized problems
- Interpret and visualize learning process: state visitation, value function, etc
- Construct toy problems where your idea will be strongest and weakest, where you have a sense of what it should do

New Problems

- Progressively increase the state and action space formulation
- Reward shaping is crucial to test the working of algorithm

Development and Tuning

- Explore sensitivity to each parameter
- Health indicators
  - Quality of value function
  - Entropy of the policy
  - KL diagnostics
- Run a battery of benchmarks

Development and Tuning

- Compare against baselines
  - Cross entropy method
  - Well tuned policy gradient method
  - Well tuned Q-learning or SARSA based method
- Use multiple random seeds
- Don't be deterred by published works
  - TRPO on Atari: 100K timesteps per batch for KL= 0:01
  - DQN on Atari: update freq=10K, replay buffer size=1M

Q-Learning

- DQN converges slowly - for ATARI it is often necessary to wait for 10-40 million frames (couple of hours to a day of training on GPU) to see results significantly better than random policy. **Be Patient**
- Optimize memory usage carefully: you'll need it for replay buffer
- Learning rate and exploration schedules are vital
- Do use Double DQN with prioritized experience replay – significant improvement

Policy Gradient

- Policy Initialization : More important than in supervised learning: determines initial state visitation
- KL spike $\rightarrow$ drastic loss of performance
- Not recommended to use DDPG when you have discrete action set

Miscellaneous Tips

- Automate your experiments;
- Techniques from supervised learning don't necessarily work in RL: batch norm, dropout, big networks
- Read older textbooks and theses, not just conference papers

Applications

- Games
- Robotics
- Wealth Management
- Supply Chain Management
- Control Systems Applications

Frontier Areas

- Risk Sensitive RL
- Optimizing Expected Reward with Constraints
- Multi Agent Systems
- Transfer and Meta Learning in RL
- Improving Data efficiency in RL

Towards Intelligent Systems

- Things that we can all do (Walking) (Evolution, may be)
- Things that we learn (driving a bicycle, car etc)
- We learn a huge variety of things (music, sport, arts etc)
- We can learn 'difficult' tasks as well

We are still far from building a 'reasonable' intelligent system

- We are taking baby steps towards the goal of building intelligent systems
- **Reinforcement Learning (RL) is one of the important paradigm towards that goal**

Slide Credit: Sergey Levine course on Deep RL at UCB

Thank You and Good Luck