Wrap-up of Bandits
Sequential decision making
Value functions
Monte Carlo

Recall: Classes of bandit algorithms

- Epsilon-greedy (simple randomization)
- Optimism in the face of uncertainty: optimistic initialization, UCB
- Gradient-based policy optimization
- Softmax / Boltzmann exploration (similar in shape to gradient-based but relies on value function estimation)
- *One more class: probability matching*

Recall: Probability matching

- *Select action a according to the probability of it being optimal:* $\pi(A_t = a \mid H_t) = \mathbb{P}[q^*(a) \geq q^*(a') \forall a' \neq a \mid H_t]$ *where* $H_t = \langle A_1 R_1 \ldots A_{t-1}, R_{t-1} \rangle$

- Note that probability matching is optimistic in the face of uncertainty - because uncertain action typically have a higher probability of being considered optimal

- How can we implement this idea?

Recall: Intuition

- If we knew the problem (reward distribution for each arm) we could easily compute the optimal action
- Initially, we have *uncertainty about the problem*
- Let's model the uncertainty directly, using a probability distribution over the problem parameters!
- This is an instance of *Bayesian reasoning*

Detour Example: Coin Toss

- Suppose you flip a coin and observe numbers of heads and tails N_H, N_T
- Maximum likelihood estimation says the probability of heads is:
$$\frac{N_H}{N_H + N_T}$$
- In the limit, this is guaranteed to converge to the right answer
- But what if you knew the coin is probably biased? Could you incorporate this information somehow?

Imagining some prior data

- Suppose in your head you imagine some initial tosses, eg $K_H = 9, K_T = 1$ if you think the coin should be biased towards heads
- You can mix these with data: $\dfrac{K_H + N_H}{K_H + N_H + K_T + N_T}$
- In the limit, this still converges to the correct estimate
- But in the short term, you use the bias
- How many tosses you imagine controls how quickly the bias washes out
- An instance of Bayesian reasoning!

Bayesian Coin Toss

- Coin toss: x ~ Bernoulli(θ)
- Let's assume that
  - θ ~ Beta(α_H, α_T)   Beta distribution
  - P(θ) $\propto \theta^{\alpha_H - 1} (1-\theta)^{\alpha_T - 1}$

- $$P(\theta|X) = \frac{P(X|\theta)P(\theta)}{\sum_\theta P(X|\theta)}$$

Prior

Posterior

The prior is conjugate!

More generally: Bayesian Reasoning

- Assume the parameters you're interested in have some prior distribution $p_0(\theta)$
- After some dataset *D* comes in, compute a *posterior:* $P(\theta | D) \propto P(D | \theta) p_0(\theta)$
- Now you can sample from the posterior!
- Advantages:
  - provides a good uncertainty estimate for θ
  - can incorporate existing knowledge through the prior
  - Converges in the limit to the same answer as max likelihood estimation but can give better estimates when you have small samples
- Disadvantage: Expensive
- Usually practiced with conjugate priors (eg Beta for Bernoulli distributions, Normal for Normal distributions…)

Back to bandits: Thompson sampling

- Instantiation of probability matching / Bayesian reasoning for bandits (developed in the 1930s)
- Idea: we are interested in the parameters of the reward distribution for each arm $\mathscr{R}_a, \forall a$
- So maintain a probability distribution over them!
- Eg if the distributions are Bernoulli, maintain a Beta distribution, with some prior (maybe equal probability) and update as data comes in
- Eg if the distributions are normal, maintain a normal over the mean, or mean and standard deviation

Algorithm

- Start with a prior over the reward distributions $p_0(\mathcal{R}_a), \forall a$
- Repeat
    1. Sample a bandit problem, aka rewards from the distributions: $\mathcal{R}_t \sim p(\mathcal{R}_a, \forall a \,|\, H_t)$
    2. Compute the best action for problem $A_t = a^*(\mathcal{R}_t)$
    3. Note this can be done easily for many problems of interest!
    4. Pull arm A_t and observe reward R_t
    5. Update the history: $H_{t+1} = \langle H_t, A_t, R_t \rangle$ and posterior $p(\mathcal{R}_a, \forall a \,|\, H_{t+1})$

Efficient implementation

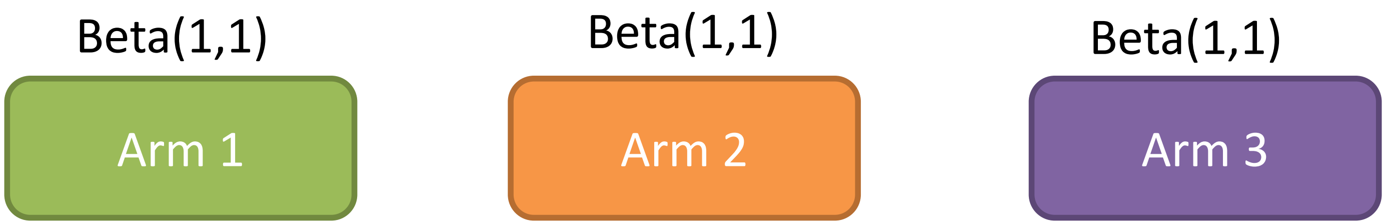
- Instead of maintaining the whole history, if we have conjugate priors, we can incrementally update the posterior
- This can in fact be done using an equivalent sample size trick: imagine you have some data sampled from the prior, which is added to your dataset
- For example:

Algorithm 1: Thompson Sampling for Bernoulli bandits

$S_i = 0, F_i = 0.$
foreach $t = 1, 2, \ldots,$ **do**
  For each arm $i = 1, \ldots, N$, sample $\theta_i(t)$ from the Beta$(S_i + 1, F_i + 1)$ distribution.
  Play arm $i(t) := \arg\max_i \theta_i(t)$ and observe reward r_t.
  If $r = 1$, then $S_i = S_i + 1$, else $F_i = F_i + 1$.
end

Example

- Start with a prior



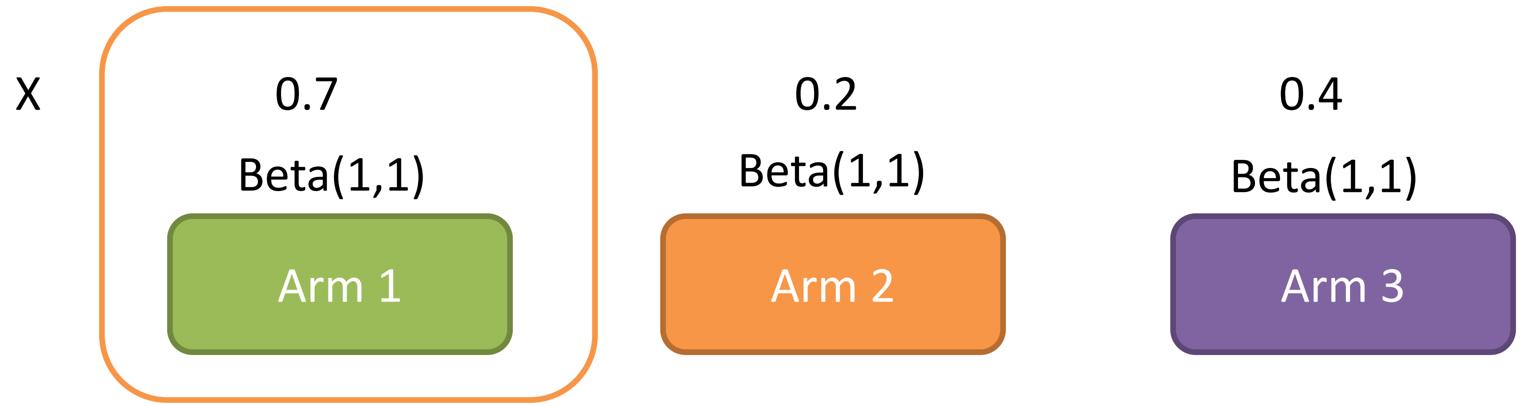
Example

- Sample a problem (bandit) from the prior

X	0.7	0.2	0.4
	Beta(1,1)	Beta(1,1)	Beta(1,1)
	Arm 1	Arm 2	Arm 3

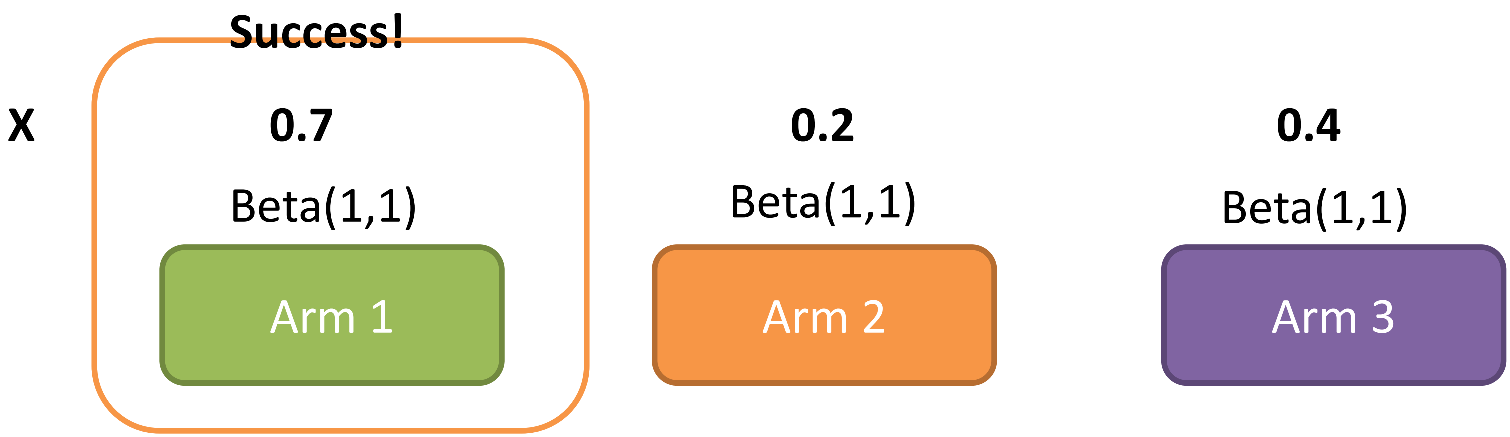
Example

- Compute the solution to the problem (best arm)



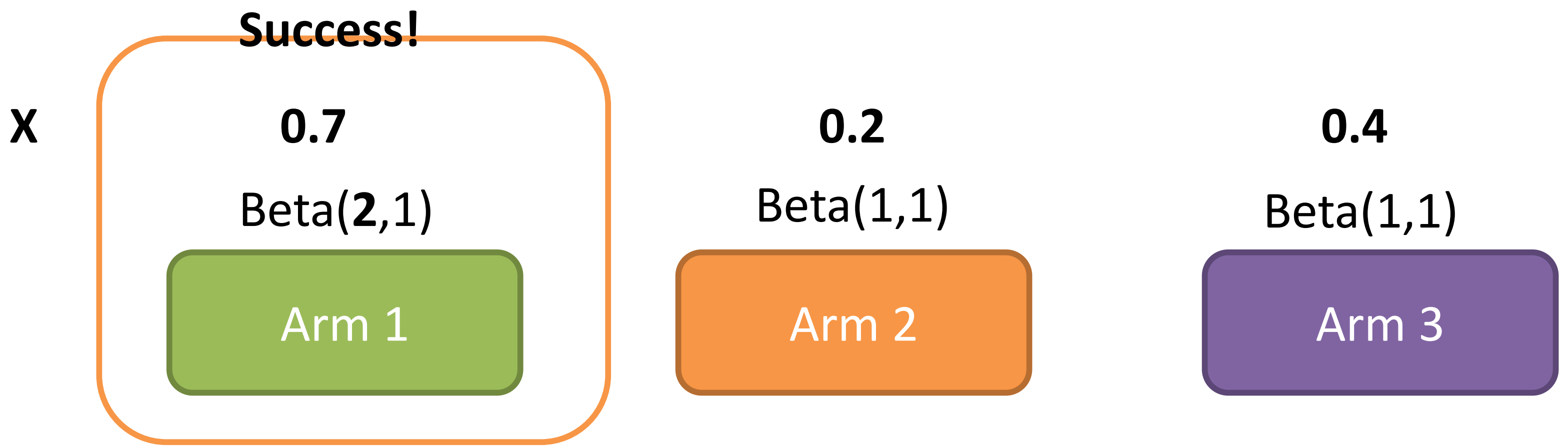
Example

- Execute the action in the real environment and observe its outcome (the reward)



Example

- Update the posterior to incorporate the observed data



Properties

- Like UCB, Thompson sampling is asymptotically optimal, ie. achieves $O(\log t)$ regret

- Took almost 80 years to prove that!! (https://arxiv.org/abs/1111.1797)

- Empirically, Thompson sampling works well for small sample sizes, especially if you know something about the problem

Problem space

	Single State	Associative
Instructive feedback		
Evaluative feedback		

Problem space

	Single State	Associative
Instructive feedback		
Evaluative feedback	Bandits (Function optimization)	

Problem space

	Single State	Associative
Instructive feedback		Supervised learning
Evaluative feedback	Bandits (Function optimization)	

Problem space

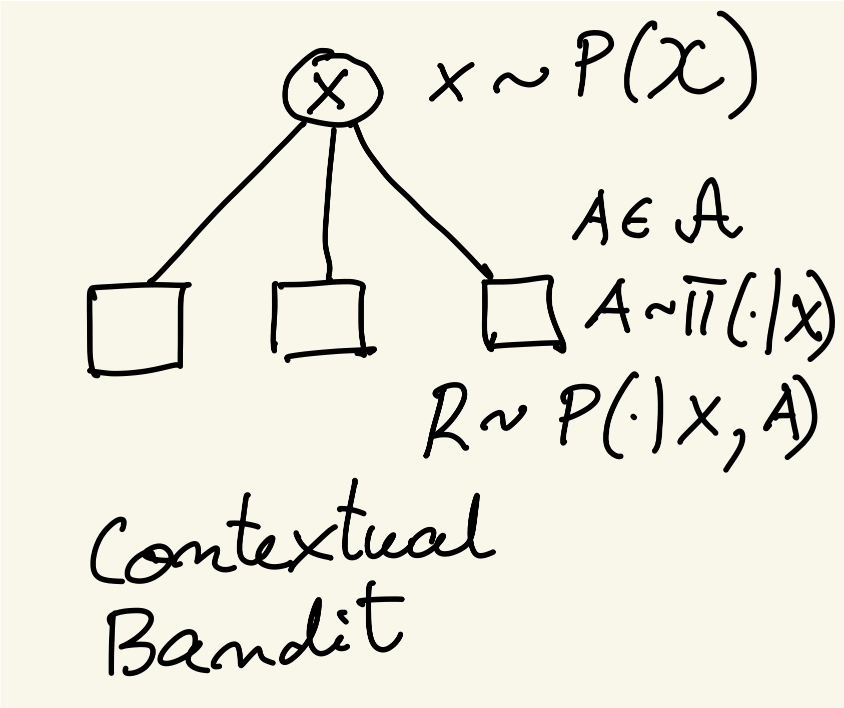
	Single State	Associative
Instructive feedback	Averaging (Imitiation)	Supervised learning
Evaluative feedback	Bandits (Function optimization)	

Problem space

	Single State	Associative
Instructive feedback	Averaging (Imitiation)	Supervised learning
Evaluative feedback	Bandits (Function optimization)	Contextual bandits

Contextual bandits

- We have some context, aka observation or state (discrete or continuous, often high-dimensional)
- The reward distribution depends on the context

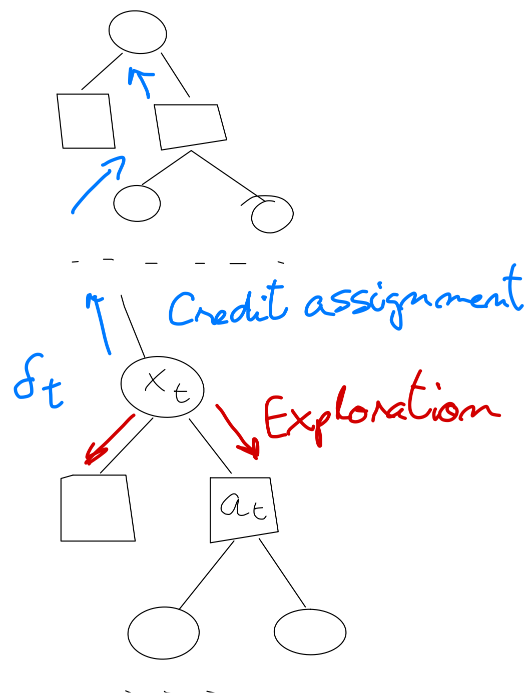


Not just exploration!

- We have to *assign credit to different features of the context*!
- Usually we will use *function approximation* to estimate action-values $Q_w(x, a)$ (or to estimate the preference function, or policy)
- Algorithms we talked about all have equivalents in this problem!
- Eg epsilon-greedy, softmax
- Eg UCB -> LinUCB (assuming $Q_w(x, a) = w_a^T x$)
- Eg Thompson sampling assuming linear rewards

Back to sequential decision making

- Recall the infinite tree of possible interactions of the agent and environment - what other assumptions can we make?
  - At time t, agent receives an observation from set $\mathcal{X}$ and can choose an action from set $\mathcal{A}$ (think finite for now)
  - Goal of the agent is to maximize long-term return



Finite-horizon assumption

- All paths end after at most T time steps
- These are called finite horizon problems
- Eg multi-stage medical treatment design

Rewards and returns

- The objective in RL is to maximize long-term future reward
- That is, to choose A_t so as to maximize $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$
- But what exactly should be maximized?
- The *return* at time t:

$$G_t = R_{t+2} + R_{t+2} + \ldots + R_T = \sum_{k=1}^{T-t} R_{t+k}$$

4 value functions

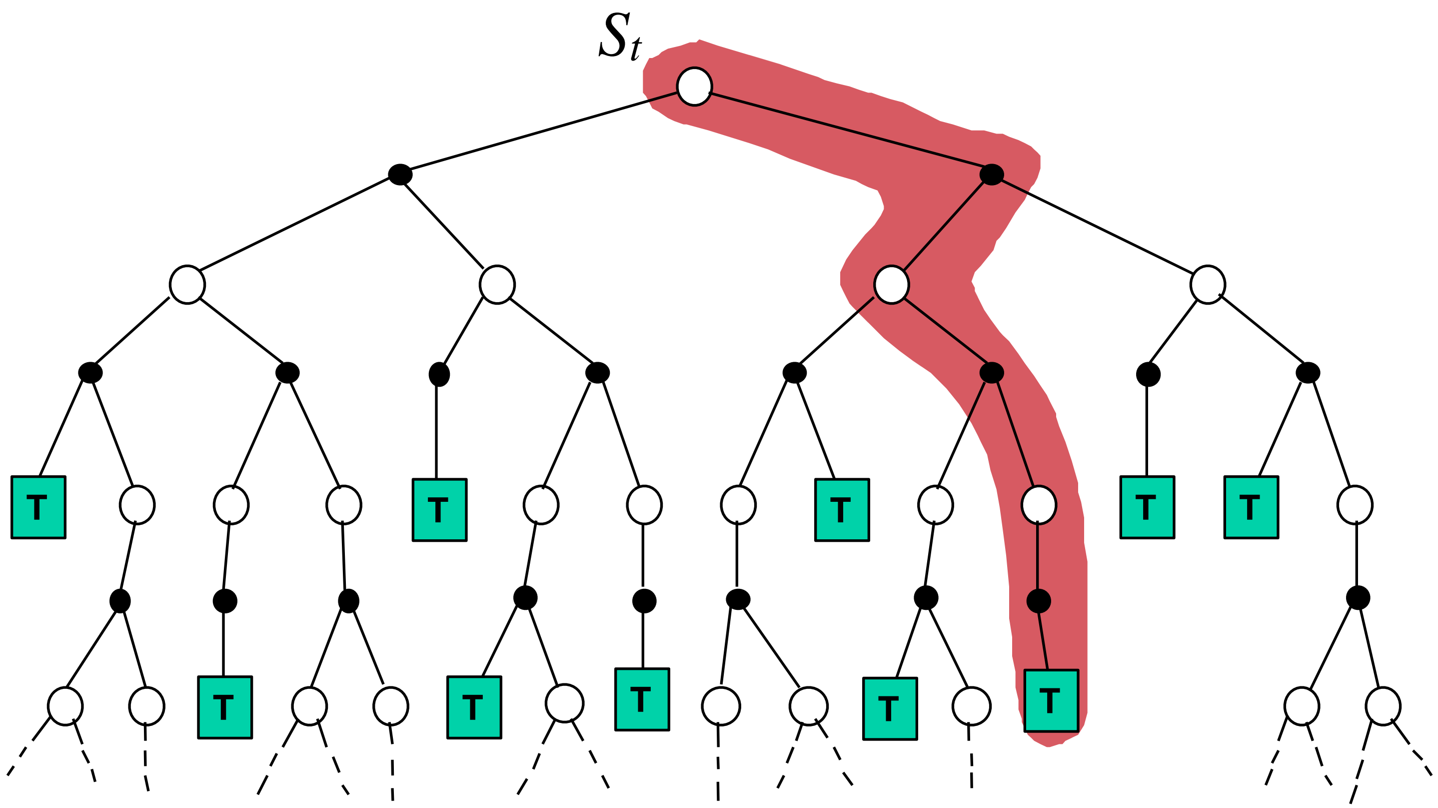
	state values	action values
prediction	v_π	q_π
control	v_*	q_*

- All theoretical objects, mathematical ideals (expected values)
- Distinct from their estimates:

$$V_t(s) \qquad Q_t(s,a)$$

Simple Monte Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t) \right]$$

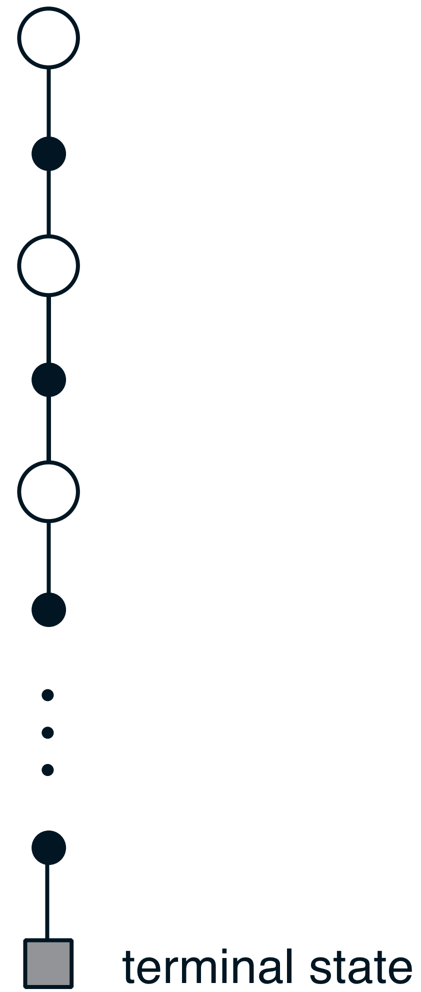


Monte Carlo Methods

- Monte Carlo methods are learning methods
  Experience → values, policy
- Monte Carlo methods can be used in two ways:
  - *model-free:* No model necessary and still attains optimality
  - *simulated:* Needs only a simulation, not a *full* model
- Monte Carlo methods learn from *complete* sample returns
  - Defined for episodic tasks (in the book)
- Like an associative version of a bandit method

Backup diagram for Monte Carlo

- Entire rest of episode included
- Only one choice considered at each state (unlike DP)
  - thus, there will be an explore/exploit dilemma
- Does not bootstrap from successor states's values (unlike DP)
- Time required to estimate one state does not depend on the total number of states



First-visit Monte Carlo policy evaluation

Initialize:

$\pi \leftarrow$ policy to be evaluated

$V \leftarrow$ an arbitrary state-value function

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:

Generate an episode using π

For each state s appearing in the episode:

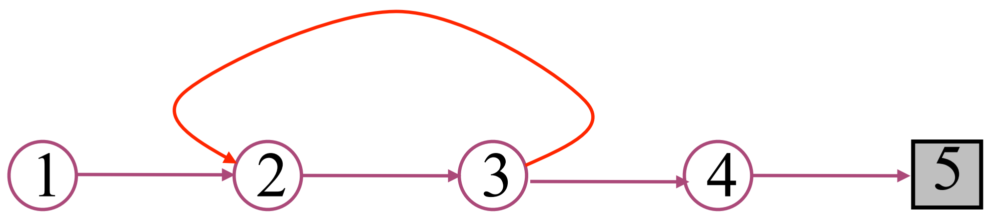
$G \leftarrow$ return following the first occurrence of s

Append G to $Returns(s)$

$V(s) \leftarrow$ average$(Returns(s))$

Monte Carlo Policy Evaluation

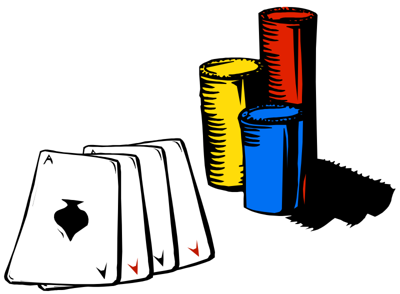
- *Goal:* learn $v_\pi(s)$
- *Given:* some number of episodes under π which contain *s*
- *Idea:* Average returns observed after visits to s



- *Every-Visit MC:* average returns for *every* time *s* is visited in an episode
- *First-visit MC:* average returns only for *first* time *s* is visited in an episode
- Both converge asymptotically

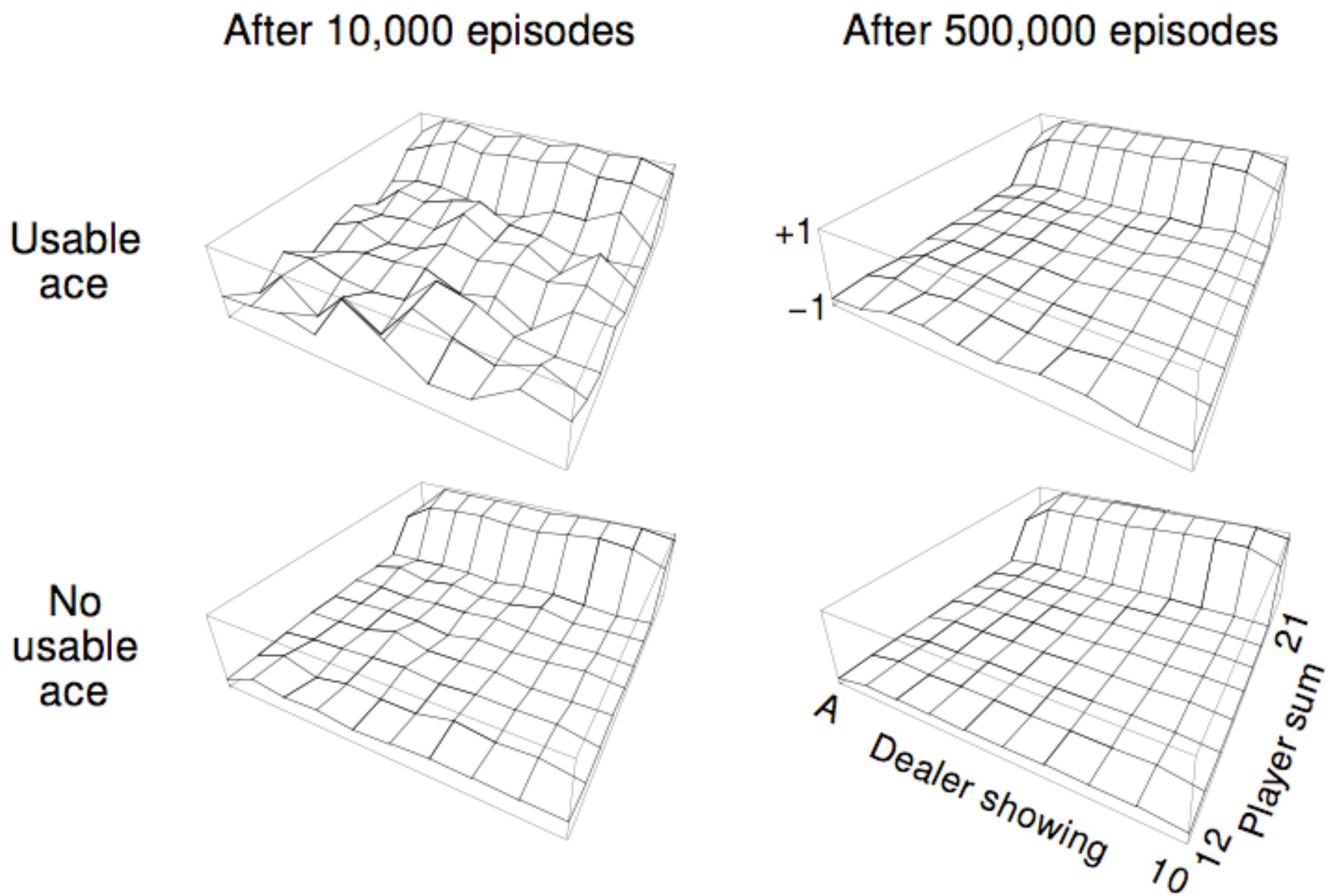
Blackjack example

- *Object:* Have your card sum be greater than the dealer's without exceeding 21.
- *States* (200 of them):
  - current sum (12-21)
  - dealer's showing card (ace-10)
  - do I have a useable ace?

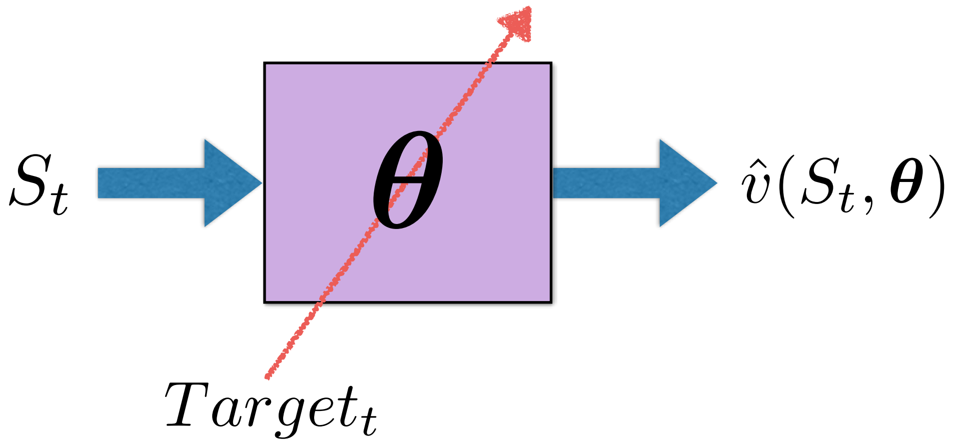


- *Reward:* +1 for winning, 0 for a draw, -1 for losing
- *Actions:* stick (stop receiving cards), hit (receive another card)
- *Policy:* Stick if my sum is 20 or 21, else hit

Learned blackjack state-value functions



Value function approximation (VFA)



Target depends on the agent's behavior!

A natural objective in VFA is to minimize the Mean Square Value Error

$$\text{MSVE}(\boldsymbol{\theta}) \doteq \sum_{s \in \mathcal{S}} d(s) \Big[v_\pi(s) - \hat{v}(s, \boldsymbol{\theta}) \Big]^2$$

where $d(s)$ is the fraction of time steps spent in *s*

Monte Carlo will provide *samples of the expectation*

- Use *sample return* instead of v_π
- Use *actual visited states* instead of $d(s)$

Gradient Monte Carlo Algorithm for Approximating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^n \to \mathbb{R}$

Initialize value-function weights $\boldsymbol{\theta}$ as appropriate (e.g., $\boldsymbol{\theta} = \mathbf{0}$)
Repeat forever:
    Generate an episode $S_0, A_0, R_1, S_1, A_1, \dots, R_T, S_T$ using π
    For $t = 0, 1, \dots, T-1$:
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big[G_t - \hat{v}(S_t,\boldsymbol{\theta})\big]\nabla\hat{v}(S_t,\boldsymbol{\theta})$

MC vs supervised regression

- Target returns can be viewed as a supervised label (true value we want to fit)
- State is the input
- We can use any function approximator to fit a function from states to returns! Neural nets, linear, nonparametric…

- *Unlike supervised learning: there is strong correlation between inputs and between outputs!*
- Due to the lack of iid assumptions, theoretical results from supervised learning cannot be directly applied

Monte Carlo Estimation of Action Values (Q)

- Monte Carlo is most useful when a model is not available
  - We want to learn q_*
- $q_\pi(s,a)$ - average return starting from state s and action a then following π
- Converges asymptotically *if* every state-action pair is visited
- *Exploring starts:* Every state-action pair has a non-zero probability of being the starting pair

Monte Carlo Exploring Starts

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \leftarrow$ arbitrary
    $\pi(s) \leftarrow$ arbitrary
    $Returns(s,a) \leftarrow$ empty list

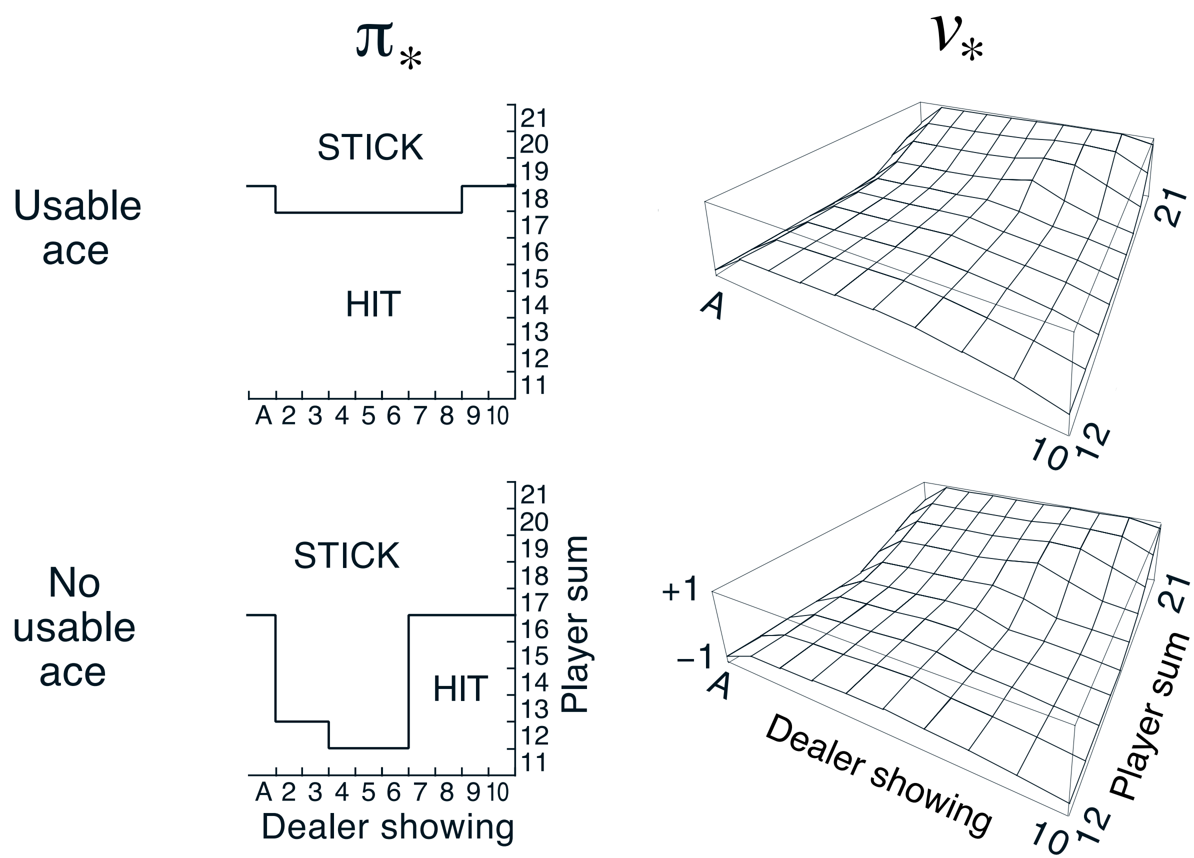
Fixed point is optimal policy π^*

Now proven (almost)

Repeat forever:
    Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ s.t. all pairs have probability > 0
    Generate an episode starting from S_0, A_0, following π
    For each pair s, a appearing in the episode:
        $G \leftarrow$ return following the first occurrence of s, a
        Append G to $Returns(s,a)$
        $Q(s,a) \leftarrow \text{average}(Returns(s,a))$
    For each s in the episode:
        $\pi(s) \leftarrow \arg\max_a Q(s,a)$

Blackjack example continued

- Exploring starts
- Initial policy as described before



On-policy Monte Carlo Control

- *On-policy:* learn about policy currently executing
- How do we get rid of exploring starts?
  - The policy must be eternally *soft*:
    - $\pi(a|s) > 0$ for all s and a
  - e.g. ε-soft policy:
    - probability of an action = $\frac{\epsilon}{|\mathcal{A}(s)|}$ (non-max) or $1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|}$ (max (greedy))

- An instance of *policy iteration*: move policy *towards* greedy policy (e.g., ε-greedy)

On-policy MC Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow$ arbitrary

$Returns(s, a) \leftarrow$ empty list

$\pi(a|s) \leftarrow$ an arbitrary ε-soft policy

Repeat forever:

(a) Generate an episode using π

(b) For each pair s, a appearing in the episode:

$G \leftarrow$ return following the first occurrence of s, a

Append G to $Returns(s, a)$

$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

(c) For each s in the episode:

$A^* \leftarrow \arg\max_a Q(s, a)$

For all $a \in \mathcal{A}(s)$:

$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$