COMP 652: Machine Learning

Lecture 17

Today

- Control learning / the reinforcement learning problem
- Multi-armed bandit problems
- Markov decision processes
- Alternative definitions of return (long-term reward)

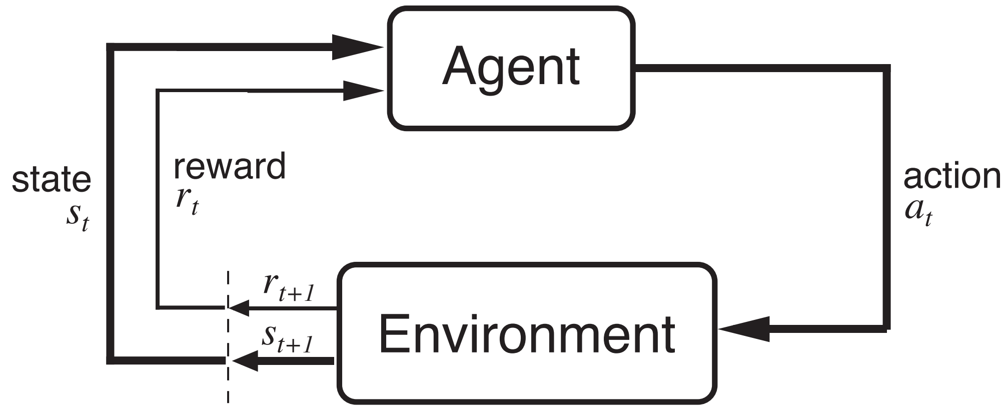
The general problem: Control Learning

Consider *learning to choose actions*, e.g.,

- Robot learning to dock on battery charger
- Choose actions to optimize factory output
- Playing Backgammon, Go, Poker, ...
- Choosing medical tests and treatments
- Conversation
- Portofolio management
- Flying a helicopter
- Queue / router control

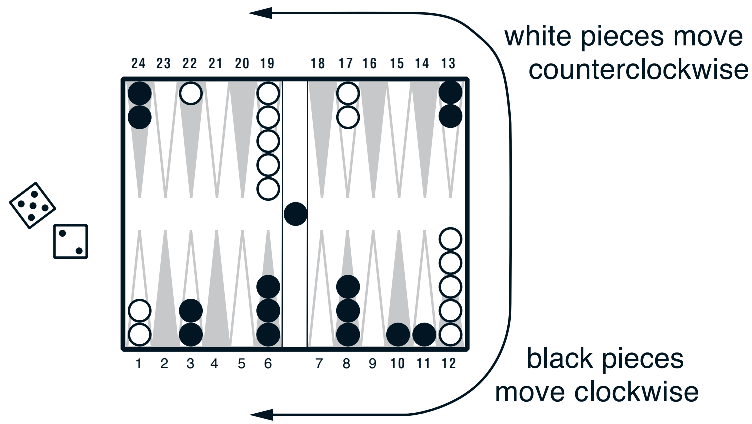
$\Rightarrow$ *All of these are sequential decision making problems*

Reinforcement Learning Problem



- At each discrete time t, the agent (learning system) observes state $s_t \in S$ and chooses action $a_t \in A$
- Then it receives an immediate reward r_{t+1} and the state changes to s_{t+1}
- Goal is to maximize the total reward over time

Example: Backgammon (Tesauro, 1992-1995)



- The states are board positions in which the agent can move
- The actions are the possible moves
- Reward is 0 until the end of the game, when it is ± 1 depending on whether the agent wins or loses
- Maximizing total reward thus equates to maximizing the chance of winning (regardless of how long it takes)

A simpler case: The multi-armed bandit

- There are k arms.
- Each "pull" of an arm i, gives a random reward with distribution $P_i(r)$ and expectation R_i, both unknown
- Each reward in an independent r.v.; the machine/arms have no internal state
- At each turn you choose one arm to pull.
- The game never ends.

A simpler case: The multi-armed bandit

Various problem can be considered:

- Identify the reward distributions of every arm
- Identify the expected reward of every arm
- Identify the arm with greatest expected reward
- Earn as much reward as possible

Some real-life motivations

- Actual gambling, of course
- Adaptive routing (on the internet)
- Experimental drug evaluation (sort of)
- Choosing a restaurant

Identifying reward distributions of every arm?

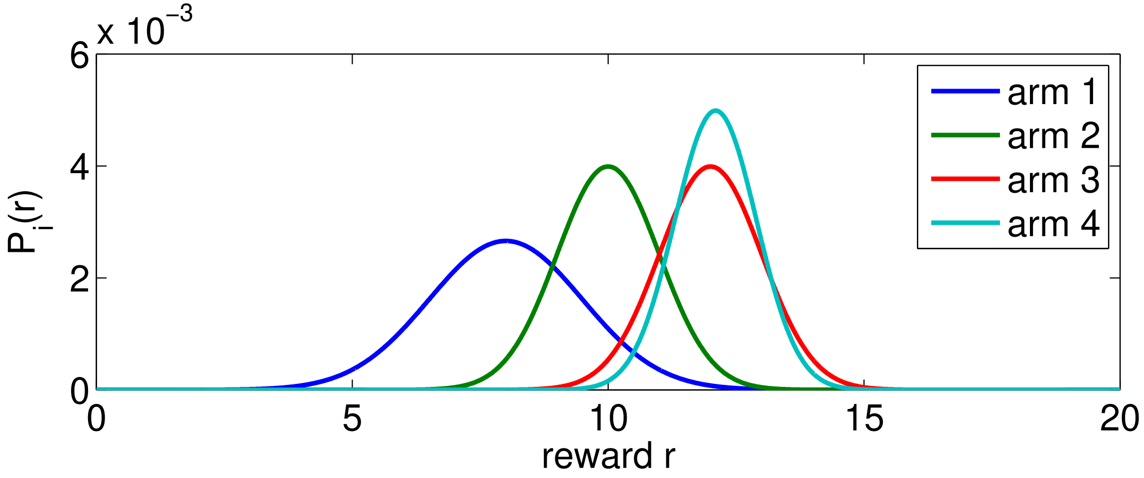
How can we do it?

Identifying reward distributions of every arm?

- Keep looping through all arms, pulling each once
  ...or, just keep choosing arms randomly
- Keep track of every reward obtained
- Use some kind of density estimator

Example 1

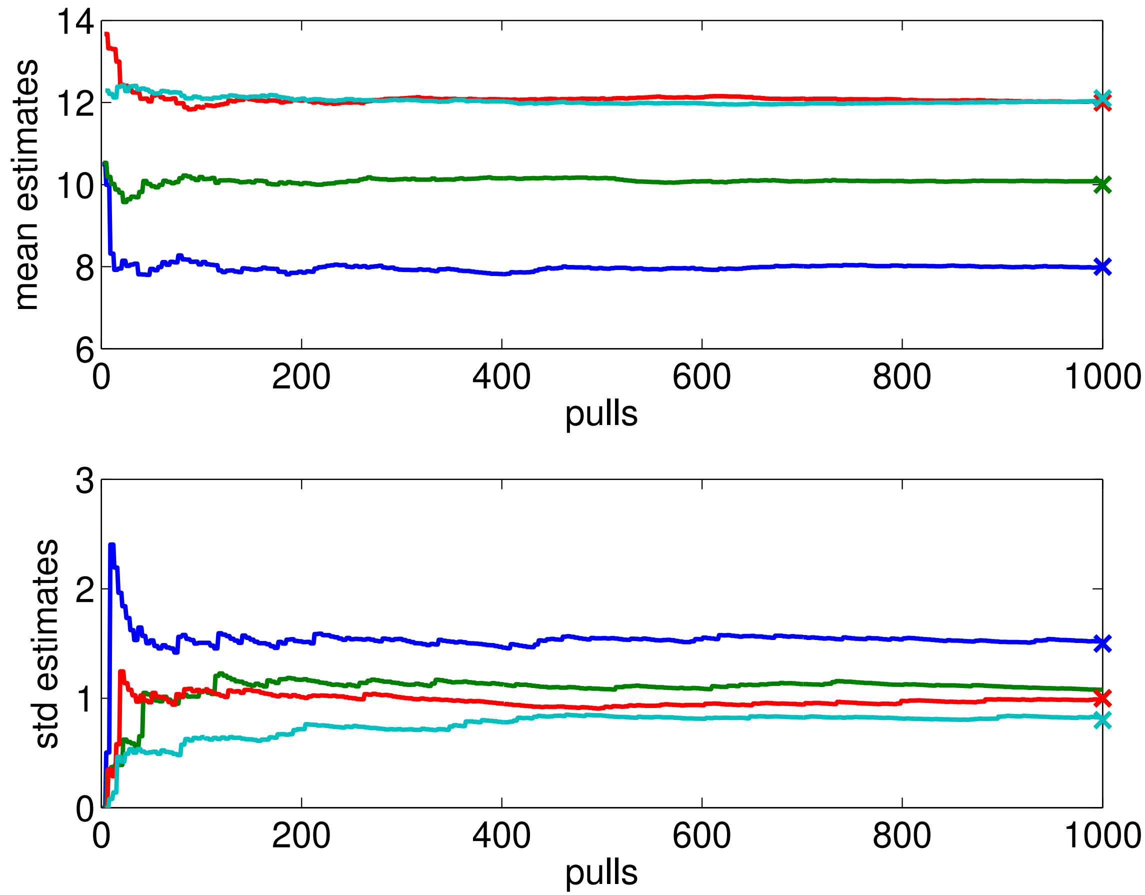
- Consider a 4-armed bandit with reward distributions depicted below:



- Suppose we:
  - Keep playing each arm in turn, for 1000 pulls
  - Assume each arm's reward distribution, P_i, is Gaussian with mean μ_i and standard deviation σ_i
  - Keep track of sample mean and sample standard deviation of rewards for each arm

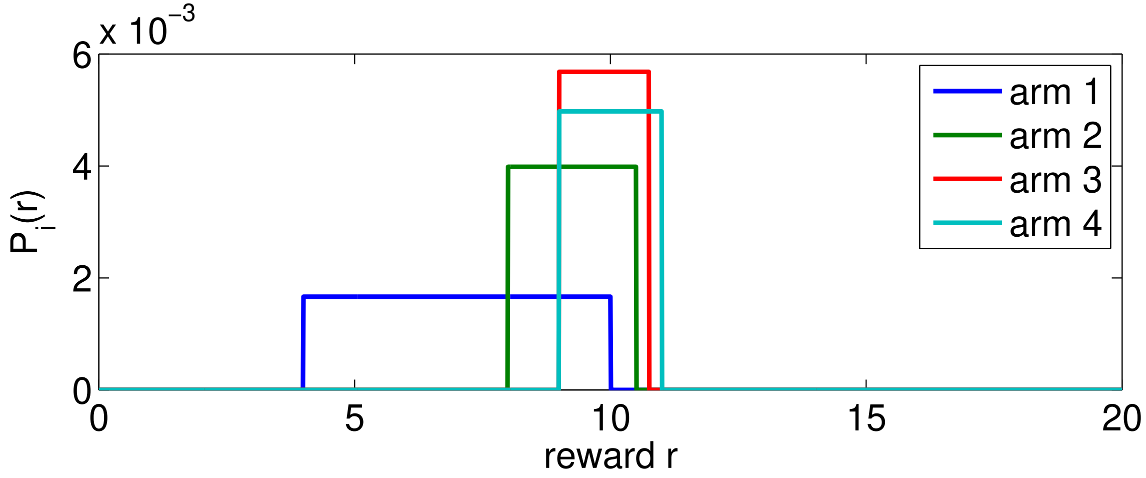
Example 1: Results

- Parameter estimates converge toward correct values (indicated by X's):



Example 2

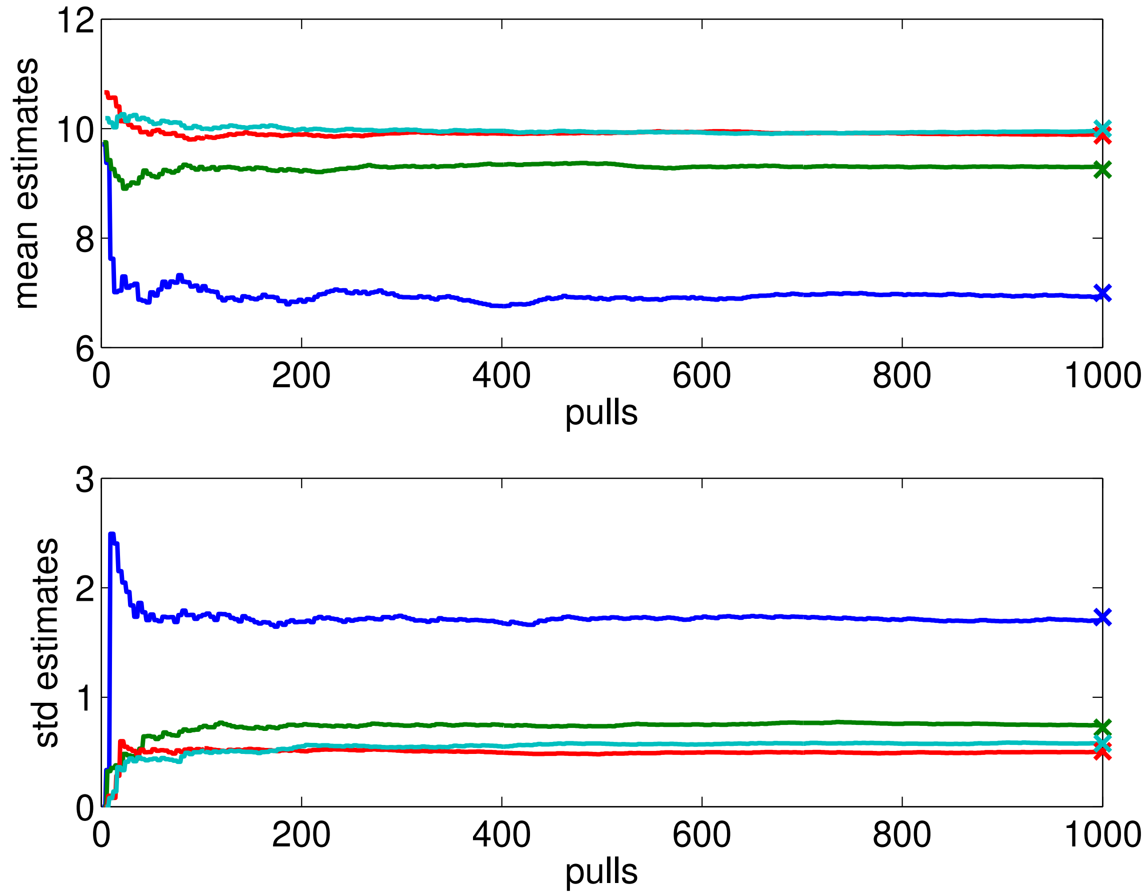
- Consider a 4-armed bandit with reward distributions depicted below:



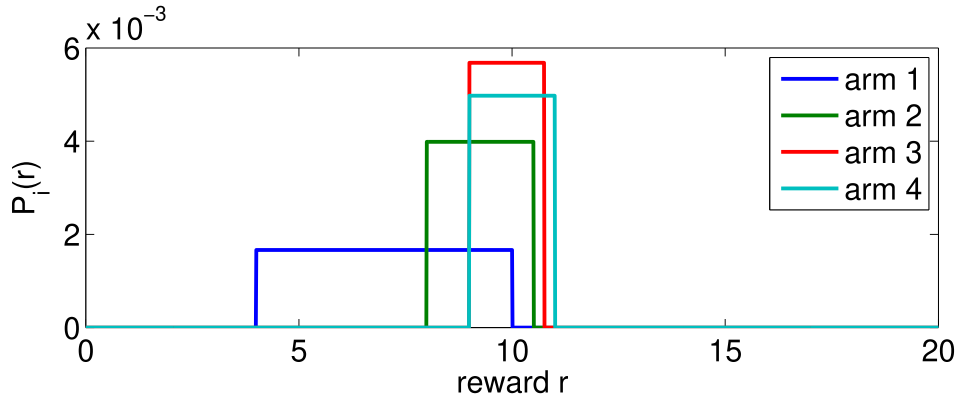
- Suppose we:
  - Keep playing each arm in turn, for 1000 pulls
  - Assume each arm's reward distribution, P_i, is Gaussian with mean μ_i and standard deviation σ_i
  - Keep track of sample mean and sample standard deviation of rewards for each arm

Example 2: Results

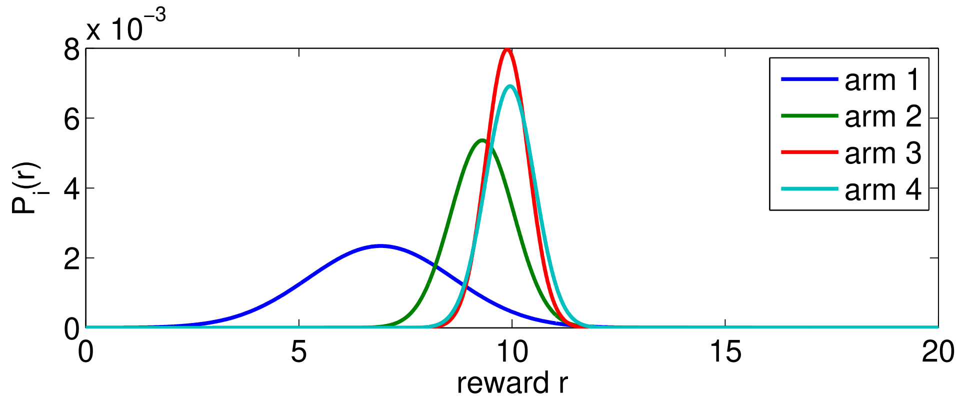
- Parameter estimates converge actual correct values mean and standard deviations of reward distributions: (indicated by X's):



- Of course, the estimate reward distributions do not converge to the correct thing:



$\neq$



Questions

- What can we do if we don't know the correct form of the distribution?
- How much reward is obtained during this process?

Questions and Answers

- What can we do if we don't know the correct form of the distribution?
  - Non-parametric density estimators may be a good choice. With proper parameterization, can prove convergence to correct distribution.
  - However, often we only care about the expected reward of each arm $\Rightarrow$ only need to track sample means
- How much reward is obtained during this process?
  - The expected reward is $\sum_{i=1}^{k} \frac{1}{k} R_i$ per step
  - This can be far less than $\max_i R_i$ per step

Maximizing reward / minimizing regret

- Rather than explicitly requiring the distributions or expected rewards of each arm to be estimated, we could simply ask for high reward.
- Suppose the game runs for $T \in \{1, 2, 3, \ldots, +\infty\}$ turns.
- Let r_t be the reward obtained ons tep t.
- We can ask for a strategy that:
  - Maximizes the mean reward: $\frac{1}{T}\sum_{t=1}^{T} r_t$
  - Or minimizes the regret: $\frac{1}{T}\sum_{t=1}^{T}(\max_i R_i - r_t)$
- How can we do that? Or how can we do that approximately / nearly?

Some possible strategies

- Choosing arms randomly doesn't work.
- Choose arm with highest estimated expected reward
- Choose arm with highest estimated expected reward most of the time, and occasionally choose something else
- Choose arms with probabilities related to their estimated expected reward
- Choose arm with highest 95% confidence interval
- Choose arm with lowest 95% confidence interval

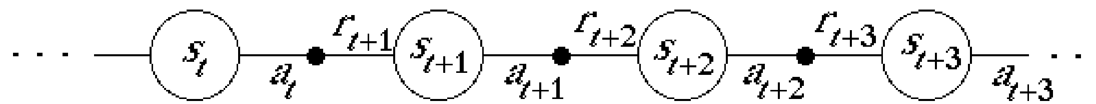
$\Rightarrow$ Try it out in MATLAB!

Remark: The exploration-exploitation trade-off/dilemma

- In reinforcement learning in general, "exploration" refers to trying new things—or things that we haven't yet learned much about
- "Exploitation" refers to acting as seems best based on our current knowledge
- In many situations, there is a tension between the two – exploration means we end up spending time doing things that may have low reward, but exploitation means we may end up never discovering a better way.

Markov Decision Processes (MDPs)

- More general, we assume the agent's environment has some *state* which changes over time



- The environment is *Markovian*:
  - The rewards obtained depends (stochastically) on the most recent state and action
  - The next state depends (stochastically) on the most rec

Markov Decision Processes (MDPs) (II)

More formally, an MDP is defined by:

- Set of *states* S
- Set of *actions* $A(s)$ available in each state s
- *Rewards*:

$$r^a_{ss'} = E\left\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\right\}$$

- *Transition probabilities*

$$p^a_{ss'} = P\left(s_{t+1} = s' | s_t = s, a_t = a\right)$$

Agent's Learning Task

Execute actions in environment, observe results, and learn *policy* (strategy, way of behaving) $\pi : S \times A \to [0,1]$,

$$\pi(s,a) = P\left(a_t = a | s_t = s\right)$$

If the policy is deterministic, we will write it more simply as $\pi : S \to A$, with $\pi(s) = a$ giving the action chosen in state s.

- Note that the target function is $\pi : S \to A$ but we have *no training examples* of form $\langle s, a\rangle$
  Training examples are of form $\langle\langle s, a\rangle, r, s', \ldots\rangle$
- Reinforcement learning methods specify how the agent should change the policy as a function of experience

The objective: Maximize long-term return

Suppose the sequence of rewards received after time step t is $r_{t+1}, r_{t+2} \ldots$.
We want to maximize the **expected return** $E\{R_t\}$ for every time step t

- *Episodic tasks*: the interaction with the environment takes place in episodes (e.g. games, trips through a maze etc)

$$R_t = r_{t+1} + r_{t+2} + \cdots + r_T$$

  where T is the time when a terminal state is reached

The objective: Maximize long-term return

Suppose the sequence of rewards received after time step t is $r_{t+1}, r_{t+2} \ldots$.
We want to maximize the **expected return** $E\{R_t\}$ for every time step t

□ *Discounted continuing tasks* :

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=1}^{\infty} \gamma^{t+k-1} r_{t+k}$$

where $\gamma =$ discount factor for later rewards (between 0 and 1, usually close to 1)
Sometimes viewed as an "inflation rate" or "probability of dying"

The objective: Maximize long-term return

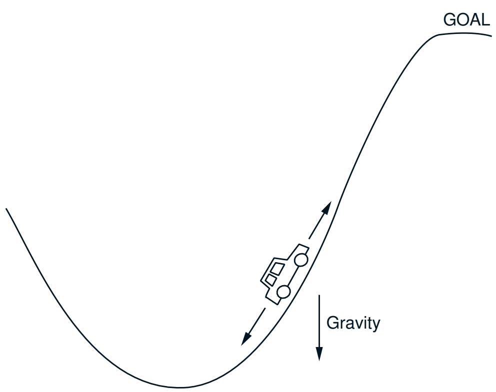
Suppose the sequence of rewards received after time step t is $r_{t+1}, r_{t+2} \ldots$.
We want to maximize the **expected return** $E\{R_t\}$ for every time step t

- *Average-reward tasks*:

$$R_t = \lim_{T\to\infty} \frac{1}{T}\left(r_{t+1} + r_{t+2} + \cdots + r_T\right)$$

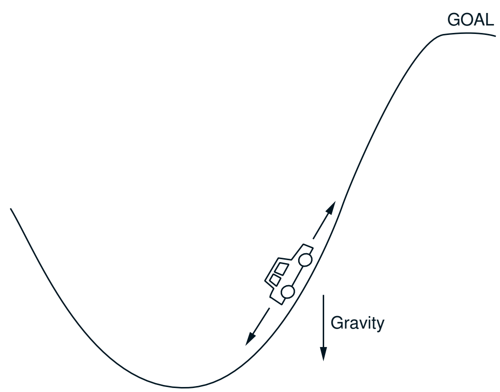
This represents the reward per time step.

Example: Mountain-Car



- States: position and velocity
- Actions: accelerate forward, accelerate backward, coast
  (It is assumed the engine is weak!)
- We want the car to get to the top of the hill as quickly as possible
- What are the rewards and the return?

Example: Mountain-Car



- States: position and velocity
- Actions: accelerate forward, accelerate backward, coast
  (It is assumed the engine is weak!)
- Two reward formulations:
  - reward $= -1$ for every time step, until car reaches the top
  - reward $= 1$ at the top, 0 otherwise $\gamma < 1$
- In both cases, the return is maximized by minimizing the number of steps to the top of the hill