CSCI-567: Machine Learning

Prof. Victor Adamchik

U of Southern California

Aug. 3, 2020

Administration

Midterm Remote Exam

- Aug. 6, Thursday, from noon to 2:50 pm.
- Practice exam will be posted
- TA's Review: Aug. 5.

Second Exam

Instructions:

- This is a take-home open-book midterm exam.
- The exams is paper-and-pencil, which you scan it and submit electronically.
- Questions should be answered concisely.
- Write legibly, avoid cursive writings.

Second Exam

Topics:

- Support Vector Machines.
- Lagrangian and KKT conditions.
- Boosting.
- Principal Component Analysis.
- Gaussian Mixture Models.
- Hidden Markov Models.
- RL will NOT be included.

Outline

Three types of machine learning

Machine learning provides algorithms to detect patterns in data and then to use the latter to achieve some goals.

There are three types of machine learning algorithms:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

For solving these problems we always use the idea of function approximation that provides an upper bound on the error: SVM, decision tree, Gaussians, neural networks, etc.

Online decision making

Problems we have discussed so far:

- start with a training dataset
- learn a predictor (classifier or regression) or discover some patterns

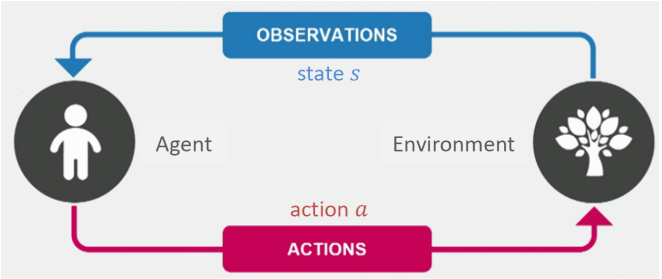
But many real-life problems are about **learning continuously**:

- make a prediction/decision
- receive some feedback
- repeat

Broadly, these are called **online** decision making problems, in which the learner receives only partial information at the end of each trial.

Human Learning

Human learning is a sequential decision-making process:



- A person with a goal observing an environment.
- Needs to decide an action to take.
- Gets a reward from the environment.
- Repeat...

Examples

Amazon/Netflix/MSN **recommendation systems**:

- a user visits the website
- the system recommends some produces/movies/news stories
- the system observes whether the user clicks on the recommendation

Playing games (Go/Atari/Dota 2/...) or **self-driving cars**:

- make a move
- receive some reward (e.g. score a point) or loss (e.g. fall down)
- make another move...

Reinforcement Learning

Reinforcement Learning is a general purpose framework mimicking human's learning process with a goal to maximize the total reward.

The RL agent does not require complete knowledge or control of the environment (compare it with DP); it only needs to be able to interact with the environment and collect information.

Reinforcement learning is the hottest topics in AI and ML today.

We discuss two the most popular mechanisms to represent RL problems:

- Multi-Armed Bandit (MAB)
- Markov Decision Processes (MDP)

Suggested reading: http://incompleteideas.net/book/RLbook2018.pdf.

Outline

MAB: Motivation

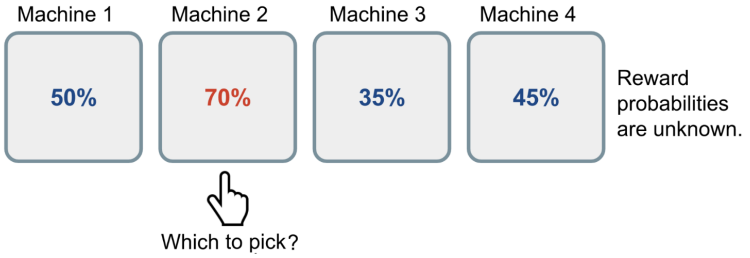
Imagine going to a casino to play a slot machine.

- that robs you, like a "bandit" with a single arm

Of course, there are many slot machines in the casino.

- like a bandit with multiple arms (hence the name)

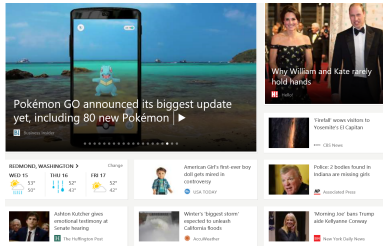
What is the best strategy to achieve the highest reward?

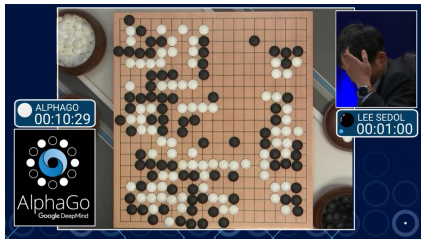


Applications

This simple model and its variants capture many real-life applications

- recommendation systems, each product/movie/news story is an arm (Microsoft MSN indeed employs a variant of bandit algorithm)
- game playing, each possible move is an arm (AlphaGo indeed has a bandit algorithm as one of the components)





Formal setup

There are K **arms** (actions/choices/...)

The problem proceeds in rounds between the environment and a learner: for each time $t = 1, \dots, T$

- the environment decides the reward for each arm $r_{t,1}, \dots, r_{t,K}$
- the learner picks an arm $a_t \in [K]$
- the learner observes the reward (or loss) r_{t,a_t} for arm a_t.

Importantly, *learner does not observe the reward for the arm not picked!*

This kind of limited feedback is now usually referred to as bandit feedback.

Environments

How does the environment decide on the rewards?

There are various settings in which MAB problems have been studied:

- Stochastic MABs: rewards generated via some fixed distribution P_a whose mean is μ_a.
- Adversarial MABs: rewards generated completely arbitrarily/adversarially.
- Markovian MABs: rewards follow a Markov process.

We focus on the stochastic MAB setting (Bernoulli bandits):

- rewards of arm a are i.i.d. samples of $P_a = \text{Ber}(\mu_a)$, that is, $r_{t,a}$ is 1 with prob. μ_a, and 0 with prob. $1 - \mu_a$, independent of anything else.
- each arm has a different mean $(\mu_1, \dots, \mu_K)$; the problem is essentially about finding the best arm $\text{argmax}_a \mu_a$ as quickly as possible

Objective

The goal is to maximize the total reward: $\sum_{t=1}^{T} r_{t,a_t}$.

Instead, we minimize the cumulative *regret*: how much I regret for not sticking with the best arm in hindsight?

$$\left(\max_{a \in [K]} \sum_{t=1}^{T} r_{t,a} \right) - \sum_{t=1}^{T} r_{t,a_t}$$

The first term is the largest reward one can achieve by always playing a fixed arm.

We need to find a slot machine that gives the highest reward, and keep playing it to maximize profit.

Greedy Selection

Let $Q(t, a)$ represent the empirical mean of the rewards received by pulling arm a up to time t. $Q(t, a)$ is an unbiased estimate of μ_a:

$$Q(t, a) = \frac{\text{Sum of rewards received from arm } a}{\text{Number of times we have picked arm } a}$$

Consider the following algorithm:

Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$, pick $a_t = \text{argmax}_a \; Q(t - 1, a)$

What's wrong with this greedy algorithm?

Greedy Selection

Consider the following example:

- $K = 2, \mu_1 = 0.6, \mu_2 = 0.5$ (so arm 1 is the best)
- suppose the alg. first pick arm 1 and see reward 0, then pick arm 2 and see reward 1 (this happens with decent probability 0.2)
- the algorithm will never pick arm 1 again!

The agent must keep in mind that it cannot play the same slot machine every single time, it occasionally needs to explore other slot machines.

Exploration vs. Exploitation

There is an inherent trade-off between **exploration** (trying a new arm that might yield better rewards) and **exploitation** (selecting an arm that has been observed to give good rewards so far).

So, each time we need to ask: *do I explore or exploit? and how?*

Any good MAB algorithm must somehow balance these two aspects.

We next discuss **three ways** to trade off exploration and exploitation for our simple multi-armed bandit setting.

A natural first attempt

Explore–then–Exploit

Input: a parameter $T_0 \in [T]$

Exploration phase: for the first T_0 rounds, pick each arm for T_0/K times

Exploitation phase: for the remaining $T - T_0$ rounds, stick with the empirically best arm $\text{argmax}_a \; Q(T_0, a)$

Parameter T_0 clearly controls the exploration/exploitation trade-off

Issues of Explore–then–Exploit

It's pretty reasonable, but the disadvantages are also clear:

- not clear how to tune the hyperparameter T_0
- in the exploration phase, even if an arm is clearly worse than others based on a few pulls, it's still pulled for T_0/K times
- clearly it won't work if the environment is changing

A slightly better algorithm

ϵ-Greedy

Pick each arm once for the first K rounds.

For $t = K+1, \ldots, T$, do one of these

- with probability ϵ, explore: pick an arm uniformly at random
- with probability $1-\epsilon$, exploit: pick $a_t = \operatorname{argmax}_a Q(t-1, a)$

Pros

- always exploring and exploiting
- applicable to many other problems
- first thing to try usually

Cons

- need to tune ϵ
- same uniform exploration

Is there a *more adaptive* way to explore?

More adaptive exploration

A modification of "Greedy" leads to the well-known:

Upper Confidence Bound (UCB) algorithm

For $t = 1, \ldots, T$, pick $a_t = \operatorname{argmax}_a \text{UCB}_{t,a}$ where

$$\text{UCB}_{t,a} = Q(t-1, a) + \sqrt{\frac{2 \ln t}{N(t-1, a)}}$$

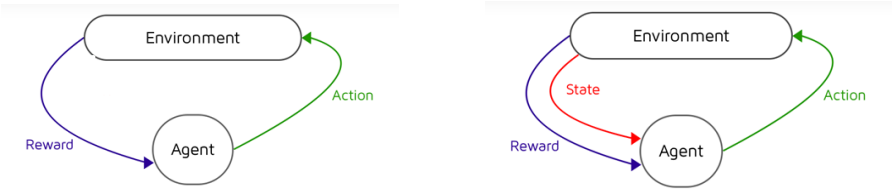
where $N(t, a)$ is the number of times arm a was pulled up to time t.

- the first term in $\text{UCB}_{t,a}$ represents exploitation, while the second (bonus) term represents exploration
- This incentive may cause the agent to pull non-greedy arms when it thinks it can get more rewards.
- UCB has an optimal regret, the proof for this is quite involved.

Outline

Motivation

Multi-armed bandit is among the simplest decision making problems with limited feedback.



It's often too simple to capture many real-life problems. One thing it fails to capture is the "state" of the learning agent, which has impacts on the reward of each action.

- e.g. for Atari games, after making one move, the agent moves to a different state, with possible different rewards for each action

Markov Decision Process

Markov Decision Process is one way to deal with this issue.

Markov Decision Process (MDP) is a combination of Markov model and multi-armed bandit

Huge recent success when combined with deep learning techniques

- Atari games, poker, self-driving cars, etc.

Markov decision process

An MDP (defined by Bellman in 1957) is parameterized by five elements

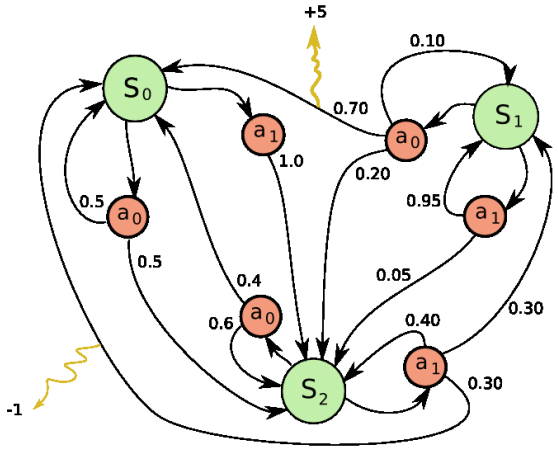
- $\mathcal{S}$: a set of possible states
- $\mathcal{A}$: a set of possible actions
- P: transition probability, $P_a(s, s')$ is the probability of transiting from state s to state s' after taking action a (Markov property)
- r: reward function, $r_a(s)$ is (expected) reward of action a at state s
- $\gamma \in (0, 1)$: discount factor, informally, reward of 1 from tomorrow is only counted as γ for today

Different from Markov models, the state transition is influenced by the taken action.

Different from Multi-armed bandit, the reward depends on the state.

Example

3 states, 2 actions



Policy

A **policy** $\pi : \mathcal{S} \to \mathcal{A}$ indicates which action to take at each state.

If we start from state $s_0 \in \mathcal{S}$ and act according to a policy π, the discounted rewards for time $0, 1, 2, \ldots$ are

$$r_{\pi(s_0)}(s_0), \quad \gamma r_{\pi(s_1)}(s_1), \quad \gamma^2 r_{\pi(s_2)}(s_2), \quad \cdots$$

respectively, where $s_1 \sim P_{\pi(s_0)}(s_0, \cdot), \quad s_2 \sim P_{\pi(s_1)}(s_1, \cdot), \quad \cdots$

If we follow the policy forever, the expected total reward is

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{\pi(s_t)}(s_t)\right]$$

where the randomness is from $s_{t+1} \sim P_{\pi(s_t)}(s_t, \cdot)$.

Note: the discount factor allows us to consider an infinite learning process

Optimal policy and Bellman equation

First goal: knowing all parameters, *how to find the optimal policy*

$$\operatorname*{argmax}_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{\pi(s_t)}(s_t)\right] \quad ?$$

We first answer a related question: *what is the maximum reward one can achieve starting from an arbitrary state s?*

$$V(s) = \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{\pi(s_t)}(s_t)\right] \qquad \text{(with } s_0 = s\text{)}$$
$$= \max_{a \in \mathcal{A}} \left(r_s(a) + \gamma \sum_{s' \in \mathcal{S}} P_a(s, s') V(s') \right)$$

This is called the **Bellman equation**, and $V(s)$ – the value function.

Bellman's optimality principle

Value Iteration

Initialize $V_0(s)$ randomly for all $s \in \mathcal{S}$

For $k = 1, 2, \ldots$ (until convergence)

$$V_k(s) = \max_{a \in \mathcal{A}} \left(r_s(a) + \gamma \sum_{s' \in \mathcal{S}} P_a(s, s') V_{k-1}(s') \right)$$

How do we solve it?

Now, knowing V, the optimal policy π^* is simply given by

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \left(r_s(a) + \gamma \sum_{s' \in \mathcal{S}} P_a(s, s') V(s') \right)$$

Convergence of Value Iteration

Does Value Iteration always find the true value function V?

Yes, we can show

$$\max_s |V_k(s) - V(s)| \leq \gamma \max_s |V_{k-1}(s) - V(s)|$$

i.e. V_k is getting closer and closer to the true V.

Learning MDPs

Now suppose we do not know the parameters of the MDP:

- transition probability P
- reward function r

How do you find the optimal policy?

We will discuss two families of learning algorithms:

- **model-based** approaches
- **model-free** approaches

But we do still assume we can observe the states (in contrast to HMM); otherwise, learning is much more difficult.

Model-based approaches

Key idea: learn the model P and r explicitly from samples.

Suppose we have a sequence of interactions:
$s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_T, a_T, r_T$, then the MLE for P and r is simply

$$P_a(s, s') \propto \#\text{transitions from } s \text{ to } s' \text{ after taking action } a$$
$$r_a(s) = \text{average observed reward at state } s \text{ after taking action } a$$

Having estimates of the parameters we can then apply value iteration to find the optimal policy.

Model-based approaches

How do we collect data: $s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_T, a_T, r_T$?

Simplest idea: follow a random policy for T steps.

This is similar to Explore–then–Exploit. Let's adopt the ϵ-Greedy idea instead.

A sketch for model-based approaches

Initialize V, P, r randomly

For $t = 1, 2, \ldots,$

- with probability ϵ, explore: pick an action uniformly at random
- with probability $1 - \epsilon$, exploit: pick the optimal action based on V
- update the model parameters P, r
- update the value function V

Model-free approaches

Key idea: do not learn the model explicitly. *What do we learn then?*

Define the $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ function as

$$Q(s, a) = r_a(s) + \gamma \sum_{s' \in \mathcal{S}} P_a(s, s') \max_{a' \in \mathcal{A}} Q(s', a')$$

$Q(s, a)$ is the expected reward one can achieve starting from state s with action a, then acting optimally.

Obviously, $V(s) = \max_a Q(s, a)$.

Knowing $Q(s, a)$, the optimal policy at state s is $\operatorname{argmax}_a Q(s, a)$.

Model-free approaches learn the Q function directly from samples.

Q-learning

How to learn the Q function?

$$Q(s,a) = r_a(s) + \gamma \sum_{s' \in \mathcal{S}} P_a(s,s') \max_{a' \in \mathcal{A}} Q(s',a')$$

On experience $\langle s_t, a_t, r_t, s_{t+1} \rangle$, with the current guess on Q, $r_t + \gamma \max_{a'} Q(s_{t+1}, a')$ is like a sample of the RHS of the equation.

So it's natural to do the following update:

$$Q(s_t,a_t) \leftarrow (1-\alpha) \underbrace{Q(s_t,a_t)}_{\text{previous guess}} + \alpha \underbrace{\left(r_t + \gamma \max_{a'} Q(s_{t+1},a') \right)}_{\text{new sample}}$$

α is a learning rate.

Q-learning

The simplest model-free algorithm:

Q-learning

Initialize Q randomly; denote the initial state by s_1.

For $t = 1, 2, \ldots$

- choose a_t from Q using ϵ-greedy.
  - with probability ϵ, explore: a_t is chosen uniformly at random.
  - with probability $1-\epsilon$, exploit: $a_t = \operatorname{argmax}_a Q(s_t, a)$.
- execute action a_t, observe reward r_t and state s_{t+1}.
- update

$$Q(s_t,a_t) \leftarrow (1-\alpha) Q(s_t,a_t) + \alpha \left(r_t + \gamma Q(s_{t+1}, a_{t+1}) \right)$$
$$s_t \leftarrow s_{t+1}$$

Comparisons

	Model-based	Model-free
What it learns	model parameters $P, r, \ldots$	Q function
Space	$O(\lvert\mathcal{S}\rvert^2 \lvert\mathcal{A}\rvert)$	$O(\lvert\mathcal{S}\rvert \lvert\mathcal{A}\rvert)$
Sample efficiency	usually better	usually worse

There are many different algorithms and RL is an active research area.

Summary

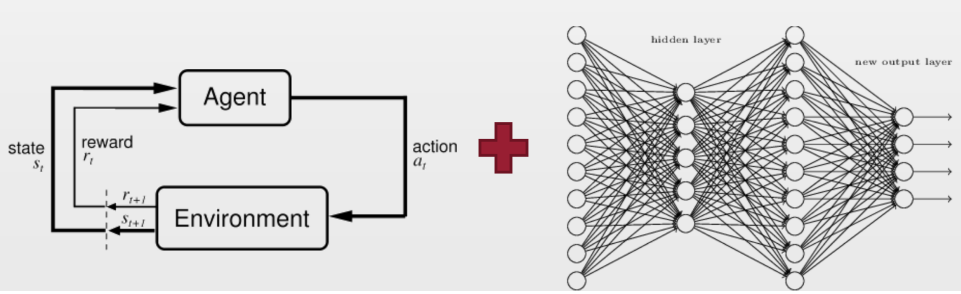
A brief introduction to some online decision making problems:

- **Multi-armed bandits**
  - most basic problem to understand **exploration vs. exploitation**
  - algorithms: explore–then–exploit, ϵ-greedy, **UCB**
- **Markov decision process**
  - a combination of Markov models and multi-armed bandits
  - learning the optimal policy with a known MDP: **value iteration**
  - learning the optimal policy with an unknown MDP: model-based approach and model-free approach (e.g. **Q-learning**)

Deep Reinforcement Learning

Can we apply deep learning to RL?

The biggest breakthrough in Q-Learning came from Google's DeepMind when they used deep neural networks (CNN)to approximate either a policy or a value function.



This technique is called Deep-Q-Networks.

Deep-Q-Networks

Deep-Q-Networks has become one of the best-known forms of Q-Learning:

- Represent value function by Deep Q-network with some weights.
- Define objective function by mean-squared error in Q-values.
- Optimise objective by SGD.

With this algorithm, RL agent was able to surpass the overall performance of a professional human player and all previous agents across a diverse range of 49 game scenarios.