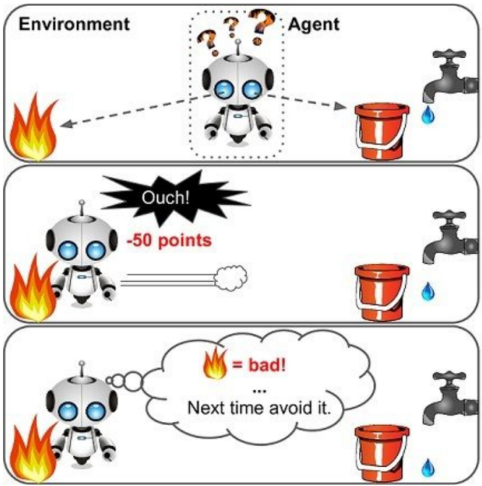
Nov 16 Active Learning



1. Observe
2. Select action using policy
3. Action!
4. Get reward or penalty
5. Update policy (learning step)
6. Iterate until an optimal policy is found

Announcements and To-Dos

1. Homework due today, next one (HMM/MDP) posted tomorrow!

Last time we learned:

1. Passive Reinforcement Learning.

Reinforcement Learning

In *Passive* Learning, the agent learns the utilities of the stats by taking a *fixed* and given policy.

1. Direct Estimation:
   Run *many* training episodes, where for each we the state started, the policy tested, the achieved rewards (and *maybe* the full list of states visited).

2. Adaptive Dynamic Programming:
   Update the estimate CPT $P(s'|s,a)$ as we go. This gives possibly exact $U^{\pi}(s)$ values from the Bellman equations. ADP is not model-free because it relies heavily on the Markov property.

$$U(s) = R(s) + \sum_{s'} P(s'|a,\, s)U(s')$$

s = location

s = (location, velocity)

   This is a great idea if the model is actually *conditionally independent* (or first-order Markov), but could lead to very biased actions if not.

Passive Learning Underview

3. Temporal-difference Learning:
   The value of a policy is based on the value of where you end up, taking it. **Update** $U^\pi(s)$ by adding $\alpha \Delta U$:

$$U^\pi_{i+1}(s) = U^\pi_i(s) + \alpha \left[R(s) + \gamma U^\pi_i(s') - U^\pi_i(s) \right]$$

   where α: the *learning rate*. Typically $\frac{1}{n}$ or $\frac{1}{n+1}$.

   Temporal-difference Model-free but otherwise similar to ADP.

As a data scientist, you should always ask what you think the underlying process might look like. If conditional independence is appropriate, go ahead and use ADP. If it's not, either use temporal-difference or direct estimation (but the latter only when the policy-space is sufficiently small)...

Schedule to Come

We have 9 more days of class. These will definitely include:

1. Nov 16: Active Learning
2. Nov 18: Intro to NLP (note: might happen *after* the two classification lectures)
3. Nov 20: Classification and Logistic Regression
4. Nov 23: Classification and intro to Perceptrons/NNs
5. Nov 25: Notebook day on Learning and/or Classifying
6. Dec 7: A final review/question session

This leaves us some room for flexibility on the week of Nov 30! What should we do?

Nov 16: Active Learning

Active Learning: The follow-up to today's lecture: how to make an agent explore new policies as it searches for utility-maximizing actions.

What we *will* discuss:

1. How to adjust passive learning AIs to force or coerce exploration: Q-learning and ε-greedy approaches.

What added content might include:

1. Some examples discussing when do MDPs versus direct versus active learning to explore.

Nov 18: Intro to NLP

Natural Language Processing: Studying how AI's take as input language or text and attempt to describe it: either as a classification or a similarity.

What we *will* discuss:

1. An overview of NLP techniques
2. Discussion on sentiment classification and naive Bayes.

What added content might include:

1. Bridging your discrete knowledge of predicate logic, quantifiers, etc. with AI classification

Nov 20: Classification and Logistic Regression

Prediction: Given a set of data, how can we predict response variables of missing or new information?

What we *will* discuss:

1. A (possible review from 3022) of regression theory, particularly with the goal of *classifying* data, leading to logistic regression.

What added content might include:

1. A brief intro to SVM's which try to solve classification problems by focusing on the *boundary* rather than the points themselves.
2. Overview/examples of prediction on *correlated* data (time series, spatial fields) and some tie-ins to Markov Models.

Nov 23: Classification and intro to Perceptrons/NNs

Prediction: Given a set of data, how can we predict response variables of missing or new information?

What we *will* discuss:

1. A brief intro to Perceptrons as a linear classifier
2. A hand-wavy intro to how Neural nets glue multiple perceptrons together

What added content might include:

1. Slightly more depth and discussion of how neural nets work.

Notebook day on Learning and/or Classifying

What we *will* do:

1. An example problem or two where we consider agents in non-perfectly observable environments.

What added content might include:

1. More examples of learning!

Vote now in Zoom, *then also in the minute form*

Topics for last days of class: (*=does *not* require 3202)

1. Add depth on NLP!
   Sequel Class: CSCI3832: Natural Language Processing*
2. Add depth on Neural Networks!
   Sequel Class: CSCI4622: Machine Learning
3. Statistical Learning methods: splines, regularization, and/or Gaussian Processes
   Zach's PhD lives here!
   Sequel Class: STAT4010*, various grad classes (CSCI: 5622, 5676)
4. Talk about Ethics, discussion day!
5. Just try to find and solve applied problems!
   Sequel Class: CSCI4302: Advanced Robotics
   Sequel Class: CSCI4022: Advanced Data Science*
   Sequel Class: CSCI4802: Data Science Team Companion Course

Definition: An **active learning** agent is allowed to choose between different policies as it tries to find an optimal policy.

In passive learning, an agent could figure out the utilities associated with a policy by taking *given* actions and recording things. Now we imagine an agent having some amount of choice regarding which policies to take!

1. Start with a policy π
2. Learn how good that policy is in terms of rewards.
3. Adapt and improve upon π

Suppose we're back to our game of "Kind Roulette," the wagering game with win probability p where we started with dollars in $[1, 9]$. The game ended if we reached \$0 or reached/exceeded \$10.

Suppose early on, the agent investigates the policy of π : "bet \$1 every time," and discovers that the expected rewards of starting at state \$5 is about a profit of \$2.

It then decides to continue to pursue this policy indefinitely, since it has positive rewards.

Question: Is this good behavior or not?

Suppose we're back to our game of "Kind Roulette," the wagering game with win probability p where we started with dollars in $[1, 9]$. The game ended if we reached \$0 or reached/exceeded \$10.

Suppose early on, the agent investigates the policy of π : "bet \$1 every time," and discovers that the expected rewards of starting at state \$5 is about a profit of \$2.

It then decides to continue to pursue this policy indefinitely, since it has positive rewards.

Question: Is this good behavior or not?

1. **Exploitation:** the agent should stick to good policies once they find them
2. **Exploration:** the agent should be encouraged to continue to look for better policies, since this might represent even better *long run* rewards.

Exploration and Exploitation

Example: The $N-$armed bandit

1

2

3

Consider a set of slot-machines. Each has equal payout, but the probabilities of winning are unknown. We set up an agent to pull 10 levels at random as **exploration**.

Each pull is a **training episode**. Maybe we try to estimate $Q[k]$, the estimated expected reward for pulling lever k.

N-armed Bandit

Suppose our first 10 training episodes are:

Lever #	Result
1	L
3	L
2	W
1	L
3	W
3	L
2	L
2	L
1	W
2	W

Count:
3x1
4x2:
3x3

estimate P/ Q(s)
1/3
2/4
1/3

What would we do if we're trying to exploit?

N-armed Bandit

Suppose our first 10 training episodes are:

Lever #	Result
1	L
3	L
2	W
1	L
3	W
3	L
2	L
2	L
1	W
2	W

We group our results by policy and find:
$P(W|1) \approx Q[1] = \frac{1}{3}$
$P(W|2) \approx Q[2] = \frac{2}{4}$
$P(W|3) \approx Q[3] = \frac{1}{3}$
So we would bet on lever 2 over the other levers.

What would we do if we're trying to exploit?

N-armed Bandit

Betting on 2 is a *greedy* strategy though. And just like greedy algorithms, the pursuit of short-term gains can prove costly if the *local* data isn't perfectly indicative of the overall structure.

Suppose our *next* 10 training episodes give:

Lever #	Result
3	W
2	L
2	W
3	W
1	L
2	L
3	W
1	L
1	W
2	L

We had:

$$P(W|1) \approx Q[1] = \frac{1}{3}$$
$$P(W|2) \approx Q[2] = \frac{2}{4}$$
$$P(W|3) \approx Q[3] = \frac{1}{3}$$

N-armed Bandit

Betting on 2 is a *greedy* strategy though. And just like greedy algorithms, the pursuit of short-term gains can prove costly if the *local* data isn't perfectly indicative of the overall structure.

Suppose our *next* 10 training episodes give:

Lever #	Result
3	W
2	L
2	W
3	W
1	L
2	L
3	W
1	L
1	W
2	L

We had:

$P(W|1) \approx Q[1] = \frac{1}{3}$

$P(W|2) \approx Q[2] = \frac{2}{4}$

$P(W|3) \approx Q[3] = \frac{1}{3}$

An update would give:

$P(W|1) \approx Q[1] = \frac{2}{6}$

$P(W|2) \approx Q[2] = \frac{3}{8}$

$P(W|3) \approx Q[3] = \frac{4}{6}$

So we would bet on lever 3, now!

Greedy or Not

The goal of any active learning agent is to be "greedy in the limit of infinite exploration." In other words: once we're explored *enough*, then it's time to get that bread (and be greedy).

Definition: An $\varepsilon-$greedy agent is a way to force exploration.

1. Keeps track of expected payouts $Q[k]$
2. Picks a *random* action with probability ε
3. Picks the current best estimated action with probability $1-\varepsilon$

Upsides? Downsides?
Solution: Upside is that it guarantees long-run infinite exploration!
Downside is that we may spend too much time in *obviously poor* areas of exploration.

5 options: true rewards: $\begin{cases} -20 \\ -100 \\ 0 \\ 8 \end{cases}$ it thinks #4 > #5.

Active functions

We can also incentivize exploration directly into the problem statement, if desired. Consider a function f that codifies the tradeoff between curiosity and greedy.

f will take in the current estimated utility u of a given action, but also the number of times n we've taken that action! We can then tell an agent to "always choose the action with the highest f, where we define:

$$f(u, n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & otherwise \end{cases}$$

1. R^+:= a reward plus: an optimistic *over*-estimate of the best reward that action could yield.
2. N_e:= some minimum number of times we want to ensure that we explore each action-state pair

Q-Learning

Instead of having to learn a model and utilities of individual states, we can also approach this problem the same way that *policy*-based methods did. We want to:
Estimate the value of doing action a in state s

Define $Q(s, a) :=$ The *expected value* of doing action a in state s.

Q naturally lets us calculate utilities, since for any state,

$$U(s) = \max_a Q(s, a)$$

Benefit: as with Temporal-difference *passive* learning, we do not always need to learn the actual transition model $P(s'|s, a)$ if we instead focus on *rewards*.

Temporal-Difference Q-Learning

As in temporal-difference passive learning, $Q-$learning requires us specify a learning rate α from which to update new Q values. Recall:

1. *Passive* temporal-difference learning: **Update** $U(s)$ by adding $\alpha \Delta U$:

$$U^{\pi}_{i+1}(s) = U^{\pi}_{i}(s) + \alpha \left[R(s) + \gamma U^{\pi}_{i}(s') - U^{\pi}_{i}(s) \right]$$

old one

2. *Active* $Q-$learning: **Update** $Q(s,a)$ by finding the difference between $Q(s,a)$ and its observed successor $Q(s',a)$.

s' is observed

$$Q_{i+1}(s,a) = Q_i(s,a) + \alpha \left[R(s) + \gamma \max_{a'} Q_i(s',a') - Q_i(s,a) \right]$$

Temporal-Difference Q-Learning

As in temporal-difference passive learning, $Q-$learning requires us specify a learning rate α from which to update new Q values. Recall:

1. *Passive* temporal-difference learning: **Update** $U(s)$ by adding $\alpha \Delta U$:

$$U_{i+1}^{\pi}(s) = U_i^{\pi}(s) + \alpha \left[R(s) + \gamma U_i^{\pi}(s') - U_i^{\pi}(s) \right]$$

2. *Active* $Q-$learning: **Update** $Q(s,a)$ by finding the difference between $Q(s,a)$ and its observed successor $Q(s',a)$.

$$\underbrace{Q_{i+1}(s,a)}_{new\,update} = \underbrace{Q_i(s,a)}_{old\,value} + \alpha \left[\underbrace{R(s)}_{immediate\,reward} + \underbrace{\gamma \max_{a'} Q_i(s',a')}_{value\,of\,where\,we\,went} - \underbrace{Q_i(s,a)}_{difference} \right]$$

Q-Learning

$Q-$learning:

$$Q_{i+1}(s,a) = Q_i(s,a) + \alpha \left[R(s) + \gamma \max_{a'} Q_i(s',a') - Q_i(s,a) \right]$$

Initialize some starter policy $\pi(s)$, initial action-utilities $Q(s,a)$ for the known states. Suppose we start a training episode.

1. Set initial state s_0.
2. Add our percept $s, R(s))$ to the known state space
3. Sample some subsequent state s'
   - 3.1 This first sample is generated from the *initialized* policies.
   - 3.2 This gives a new state s'. We have an initialized set of Q values for s', so we know $Q_i(s',a')$.
   - 3.3 We can now update $Q_{i+1}(s,a)$ as above.
   - 3.4 Crucially, we can *also* now update $\max_{a'} Q(s,a')$ for our *original* state s.

$Q-$learning:

$$Q_{i+1}(s,a) = Q_i(s,a) + \alpha \left[R(s) + \gamma \max_{a'} Q_i(s',a') - Q_i(s,a) \right]$$

Initialize some starter policy $\pi(s)$, initial action-utilities $Q(s,a)$ for the known states. Suppose we start a training episode.

1. Set initial state s_0.
2. Add our percept $s, R(s))$ to the known state space
3. Sample some subsequent state s'
   - 3.1 This first sample is generated from the *initialized* policies.
   - 3.2 This gives a new state s'. We have an initialized set of Q values for s', so we know $Q_i(s',a')$.
   - 3.3 We can now update $Q_{i+1}(s,a)$ as above.
   - 3.4 Crucially, we can *also* now update $\max_{a'} Q(s,a')$ for our *original* state s.

Then we repeat this process (our new state s' becomes our decision-location s) until the training episode ends.

Q-Learning

So this is a double loop:

FOR MANY TRAINING EPISODES:

FOR STEPS UNTIL THAT EPISODE IS COMPLETED:

1. Sample some subsequent state s' *from the exploration function* $f(u, n)$. The "best-case" exploration values $R+$ can be *very* optimistic... what kinds of models might we have for these sorts of things?
2. Since we know $Q_i(s', a')$, we can now update $Q_{i+1}(s, a)$ as prior slide.
3. *Also* update $\max_{a'} Q(s, a')$ for our *original* state s. One way: run a loop over the old state:
   - 3.1 $bestQ = -Inf$
   - 3.2 for all $ainA(s)$, $bestQ = \max(bestQ, Q[s', a'])$

Now that we're also using our exploration function, our agent will both *explore* and try to *exploit* once n is sufficiently large.

Active Learning Underview

We discussed 3 different ways to balance the exploration/exploitation tradeoff.

1. $\varepsilon-$ greedy:
   force exploration ε proportion of the time.
2. Functional rewards for exploration:
   encourage exploration until we've run each action-state pair N_e times.
3. $Q-$Learning.
   update action-pair utilities (combined with 2.) to *learn* while converging to an optimal action.

Moving Forward

- Coming up:
  1. Intro to Classifiers!