Nov 11 Passive Learning

when losing

bet all-in if $ ≤ 5.

bet $4 if at $6

bet $2 if at $8

bet $1 if at $9.

when winning

bet $1 before at $9

bet 7-8 at $ 9.

Recall our notebook from last Friday, with a betting games:

1. You start with some integer amount of money from $1-$10.
2. You can wager any amount in increments of 1, with the maximum amount permitted = however much money you currently have.
3. You flip a ~~fair~~ unfair coin. If heads, you win your wager and your money increases by that amount. If tails, you lose your wager and your money decreases by that amount.
4. You will quit if you have at least 10 at any point, or if you lose all of your money. Your reward is the difference between your ending money and $\$5$ (typical starting value).

What were the optimal strategies if: Know

A) We're *heavily favored* to win $(p \approx .8)$

B) We're *heavily favored* to lose $(p \approx .2)$

Announcements and To-Dos

bounded Optimization: don't want d < 0

1) method = bounded to optimize.minimize

2) optimize.minimize_scalar input bounds

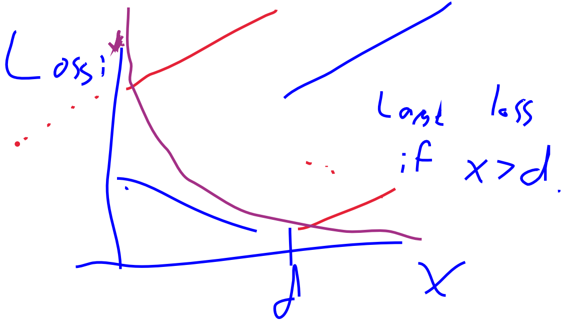
#1e

1. Homework posted! Exposition on last part of ~~#2~~ may be a little confusing: check Piazza clarification!

2. Last chance to turn in practicum!

Last time we learned:

1. Started Reinforcement Learning.



$x > d$

bus time

we go to the bus stop

we got there on time.

Reinforcement Learning

Definition: *Reinforcement Learning*, is the act of training agents to make a sequence of decisions to achieve some notion of utility or rewards. In *Passive* Learning, the agent learns the utilities of the stats es by taking a *fixed* and given policy.

1. Direct Estimation:
   Try many policies, see which is doing best.
2. Adaptive Dynamic Programming:
   Try to learn the transition *model* as we go, and then make better decisions.
3. Temporal-difference Learning:
   Try to make sure that the utilities of *neighboring* states follow the Bellman equations.

Recall the Bellman equations are the system that state that under optimal policies, the utility of each state is given by the utility of the *best action* a from that state:

$$U_{(}s) = R(s) + \sum_{s'} \underbrace{P(s'|\overset{\text{best action}}{\boxed{a}}, s)}_{\text{prob outcomes}} \underbrace{U(s')}_{\text{utility outcomes}}$$

Direct Estimation

Direct Estimation: Try many policies, see which is doing best.

1. Sample many action sequences beginning from state s. We call these *training episodes* or *trials*.
2. Move according to a policy $\pi(s)$, where S_t might be a random variable.
3. This is the same as sampling from a Bayes Net, where our transition probabilities $P(s'|a,s)$ behave the same as a conditional probability table.
4. We add up all our rewards along the way. In the end, the value of the policy is estimated by the utility achieved!

   multiple tries : np.mean(rewards) for each policy.

$$U^\pi(s) = \overset{\text{multiple tries}}{E}\left[\underbrace{\sum_{t=0}^{\infty}\gamma^t R(S_t)}_{\text{add up observed rewards}}\right]$$

don't use $P(s'|a,s)$

5. This can be done **model-free**. We can often *simulate* results of our actions without writing down or computing explicit probabilities $P(s'|a,s)$.

Direct Estimation

Example: Direct Estimation. Recall our notebook from last Friday. In it, there is a betting game that has these rules:

1. You start with some amount of money.
2. You can wager any amount in increments of 1, with the maximum amount permitted = however much money you currently have.
3. You flip a unfair coin. If heads, you win your wager and your money increases by that amount. If tails, you lose your wager and your money decreases by that amount.
4. You start playing this game with 5. You will quit if you have at least 10 at any point, or if you lose all of your money. Your reward is the difference between your ending money and $\$5$.

$R(s) = -.01$ if $s \neq 0$ or $s \geq 10.$

$R(0) = -5$ $R(s) = s - 5$

Direct Estimation

Example: Direct Estimation. In this game, suppose that we wish to evaluate a policy by *direct estimation.*

1. We choose a policy. Say we start with $\underline{\pi(s) = 1 \forall s}$, or we always wager \$1.
2. We start at some state. Maybe $s_0 = 2$. Then we play the game a few times, and observe a chain of results for one *training episode*:

$$\overset{s_0}{\boxed{2}} \to \overset{s_1}{\boxed{3}} \to \overset{s_2}{\boxed{2}} \to \overset{s_3}{\boxed{1}} \to \overset{s_4}{0}$$

We then tally our utilities. If we had $R(s) = -.01$ for non terminal results, we've now seen no less than *4* samples! For example, we in this trial were at state "1," got our $\underline{\text{-0.01}}$ reward, then lost and got our -5 terminal state.

From right to left:

$$U(1) = \boxed{-5.01,} \quad \underline{U(2) = \boxed{-5.02,}} \quad U(3) = \boxed{-5.03,} \quad \boxed{U(2) = -5.04}$$

Direct Estimation

After performing *a lot* of simulations from a variety of initial states, we can simply tabulate the rewards observed *given* the various policies undertaken. Upside:

1. Easy to write.
2. Intuitive.

Downside:

1. Really really expensive to explore the way that different policies at every state interact.
2. Doesn't formally take advantage of the fact that the transition probabilities $P(s'|a, s)$ (are fixed:) we require a huge sample because we're not exploiting this dependence!

might be!

Adaptive Dynamic Programming Estimation

Definition: *Adaptive Dynamic Programming* for reinforcement learning samples from the state space and updates estimates of $P(s'|s,\, a)$ as it runs.

This is *model-based*, because we're now possibly *assuming* things like the underlying Markov Property, that $P(s'|a,\, s)$ depends only on the current state s and not some notion of continuity or momentum based on earlier states.

One twist: we no longer need a fully observable state space: instead we can let our agent *explore* the space!

ADP RL

Ex: # of times we bet \$1 from s=5.

1. Initialize:
   a) $Na(s,a)$: the number of times we took action a from state s.

   b) $Nt(s',s,a)$: the number of times *when* we took action a from state s we ended up at state s'.

   bet \$1 from s=5: count s'=4 and s'=6.

   c) Set $P(s'|a,s) = \underline{Nt(s',s,a)}/Na(s,a)$ to estimate the transition model: the probability that action a in state s yields state s'.

   $$P(A|B) = \frac{P(\text{both})}{P(B)}$$

   B: action
   A: next state.

   d) $U(s)$: utility for each state, initialized somehow.

   0? 1?

ADP RL

2. Train. Run *many* training episodes, where for each:
   a) Start as some random state s_0.

   b) Sample a next state s_1. Store the *percept* $(s_1, R(s_1))$.

   (simulate s_1 given action).

   c) If s_1 is a new state, add its key to the list of states and the counting dictionaries. Store $R(s_1))$ in the rewards (vector/dictionary).

   d) Update the counters Na and Nt.

   e) Update the the estimate CPT $P(s'|s, a)$.

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

   f) Update $U(s)$ by *policy evaluation*. For example, use our current estimates of $P(s'|a,\ s)$ and $U(s')$ to choose a best policy for s and find its utility.

ADP RL Example

Suppose again we're playing our Roulette-like game, and observe a chain of results for one *training episode*:

$$2 \boxed{\boxed{\underset{W}{\rightarrow}}} 3 \underset{L}{\rightarrow} 2 \boxed{\underset{L}{\rightarrow}} 1 \underset{L}{\rightarrow} \boxed{0} \qquad \text{reward: } -5.$$

Now we tally up:

1. Counts of *actions*: $Na[(2, a = 1)] = 2$, $Na[(3, a = 1)] = 1$, $Na[(1, a = 1)] = 1$
2. Counts of *results*: $Nt[(3, 2, a = 1)] = 1$, $Na[(2, 3, a = 1)] = 1$, $Nt[(1, 2, a = 1)] = 1$, $Na[(0, 1, a = 1)] = 1$
3. Find some probabilities! Now we have:

   P(we win from state 2 given action: bet $1)

$$\boxed{P[(\underset{s'}{3}, \underset{s}{2}, \underset{bet}{a = 1})]} = \frac{Nt[(3, 2, a = 1)]}{Na[(2, a = 1)]} = \frac{\boxed{1}}{\boxed{2}}$$

4. We could now update $\underline{U(2)}$ as a function of $\underline{U(3)}$ and $\underline{U(1)}$, but right now our only policy is "1." We need more training episodes!

Using a Model

ADP is not model-free because it relies heavily on the Markov property. This tends to lead it to be much more accurate than direct estimation *when that assumption is correct*.

Our final method today is **Temporal-difference** learning, which is:

1. Model-free, because it doesn't estimate P.
2. Idea: instead of working on updating probabilities, work directly with the *utilities* and iterate with a modification on Bellman equations:

$$U_{i+1}(s) = R(s) + \gamma \sum_{s'} \boxed{\underbrace{P(s'|a,\, s)}_{don't\, know\, this}} U(s')$$

Temporal-Difference

only s' results from a:

$$U_{i+1}(s) = R(s) + \gamma \sum_{s'} \underbrace{P(s'|a,\, s)}_{don't\, know\, this}\; U(s')$$

update to i+1

Ignoring P for now, suppose that we always go where we want. P is 0 or 1, and

$$\underbrace{U(s)}_{\text{LHS}} = \underbrace{R(s) + \boxed{\gamma U(s')}}_{\text{RHS}}$$

Now define the *difference* between these two sides:

$$\Delta U = \underbrace{R(s) + \gamma U(s')}_{\text{RHS}} - \underbrace{U(s)}_{\text{LHS}}$$

$\Delta U > 0$ if RHS > LHS

Why this?

1. If we could always go where we want, should always be zero for the *actual* utility values.
2. What happens to this value of our *current* estimate of $U^{\pi}(s)$ is too high? $\Delta U < 0$
3. What happens to this value of our *current* estimate of $U^{\pi}(s)$ is too low? $\Delta U > 0$

Temporal-Difference

$$U_{i+1}(s) = R(s) + \gamma \sum_{s'} \underbrace{P(s'|a,\, s)}_{don't\, know\, this}\; U(s')$$

Ignoring P for now, suppose that we always go where we want. P is 0 or 1, and

$$U(s) = R(s) + \gamma U(s')$$

Now define the *difference* between these two sides:

$$\Delta U = R(s) + \gamma U(s') - U(s)$$

Why this?

1. If we could always go where we want, should always be zero for the *actual* utility values.
2. What happens to this value of our *current* estimate of $U^\pi(s)$ is too high? $\boxed{\Delta U < 0}$
3. What happens to this value of our *current* estimate of $U^\pi(s)$ is too low? $\boxed{\Delta U > 0}$

Temporal-Difference

$$\Delta U = R(s) + \gamma U(s') - U(s)$$

estimates how far off our value of $U(s)$ is from its true value.

Note that this estimate will be off by more if the probabilities of moving where we *don't* intend to is high!

Algorithm:
Update $U(s)$ by adding $\alpha \Delta U$:

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

where α: the *learning rate*.

prior utility

$+ \alpha \Delta U$

Temporal-Difference Learning Rates

α: the *learning rate*

1. The idea behind the learning rate is twofold: it helps include the missing information of P *not* being 0 or 1, and it adjusts to how much information we have.
2. Typically, we make α *decrease* with the number of trials. We might write $\alpha = \alpha(N(s))$ to explicitly state that alpha at state s depends on the number of visits to s.
3. This way, we don't make large changes to a state's utility if we've visited it many times!
4. Formal requirements for convergence of U_i to true utilities under policy π:

$\frac{1}{n^p}$

$$\underline{\sum_{n=0}^{\infty} \alpha(n) = \infty} \quad \textbf{AND} \quad \underline{\sum_{n=0}^{\infty} (\alpha(n))^2 < \infty}$$

$\sum \frac{1}{n} = \infty$

$\sum \frac{1}{n^{1.0001}} < \infty.$

Temporal-Difference Learning Rates

α: the *learning rate*

1. The idea behind the learning rate is twofold: it helps include the missing information of P *not* being 0 or 1, and it adjusts to how much information we have.
2. Typically, we make α *decrease* with the number of trials. We might write $\alpha = \alpha(N(s))$ to explicitly state that alpha at state s depends on the number of visits to s.
3. This way, we don't make large changes to a state's utility if we've visited it many times!
4. Formal requirements for convergence of U_i to true utilities under policy π:

$$\sum_{n=0}^{\infty} \alpha(n) = \infty \qquad \textbf{AND} \qquad \sum_{n=0}^{\infty} (\alpha(n))^2 < \infty$$

Examples: Things like $\alpha(n) = \frac{1}{n}$

Passive Learning Underview

That wraps up our content on passive learning. The main takeaways:

1. **Direct Estimation:** just involves simulating agent actions and tallying up the rewards. We actually did this at the end of the agents notebook long ago!
2. **Adaptive Dynamic Programing** involves simulating agent agents and estimating the transition probabilities. 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.
3. Temporal-difference learning calculates the utility of a policy by directly iterating on approximations of the Bellman equations without ever trying to write down $P(s'|a\,s)$, making it a model-free but 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 and/or $\underbrace{\text{Gaussian Processes}}_{\text{Zach's PhD lives here!}}$
   Sequel Class: STAT4010*, various grad classes.

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*

Moving Forward

- Coming up:
  1. Active learning!