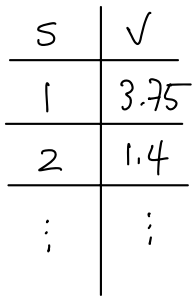
Part I: Tabular Methods



s	v
1	3.75
2	1.4
⋮	⋮

Ch. 2: Bandits → (+ states) Ch. 3: MDPs

Ch. 4: Dynamic Programming (Model-based)

Ch. 5: Monte Carlo

Ch 6: Temporal Differences

Ch. 7: N-step bootstrapping (Model-free)

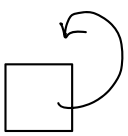
Ch. 9: Learning and Planning

2.0: Intro

- RL: differentiating factor is it uses training information that evaluates the actions taken, rather than instructs by giving correct actions. ← one state
- We study the nonassociative, evaluative feedback problem: k-armed bandit problem.

Definitions

- Evaluative feedback: indicates how good the action taken was.
- Instructive feedback: gives the correct answer.
- Associative: the optimal action depends on the current state or situation.
- Non-associative: only 1 state/context to consider. Just action and rewards. This is the bandit problem!



1-state

2.1 : A k-armed bandit problem

- Learning problem where the agent chooses among k different options / actions (like arms in a slot machine).
- Probability distribution of rewards depending on action.
- Objective : maximize expected total reward over some time period, say, 1000 time steps.

Formalization:

At time t:

Action $A_t \in \{ \text{Arm } 1, \ldots, k \}$ // pick an arm

Reward $R_t \in$ Rewards of Bandit_t // Reward distribution of the selected arm

value of arbitrary action $a \equiv q_*(a)$ = the ground truth expected reward.

$$q_*(a) \doteq E[R_t \mid A_t = a]$$

If we knew the reward of each action, then we would just choose the action with the highest value.
But we don't. We only have estimates.

Denote:

$Q_t(a)$ = estimated value of action a at time t.

would like $Q_t(a)$ to be close to $q_*(a)$.

2.2: Action-value methods

AVMs are methods for calculating $Q_t(a)$ estimates. One way to do it is to average the rewards actually received:

Sample average:

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ was taken at time } < t}{\#\text{ of times } a \text{ was taken at time } < t}$$

$$= \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i = a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i = a}}$$

// sum of rewards at times when $A_t = a$

// count of times when $A_t = a$.

By the law of large numbers $Q_t(a)$ converges to $q_*(a)$.

Now that we have an estimate for the rewards, how do we use it to select actions?

Greedy Action Selection (Definition):

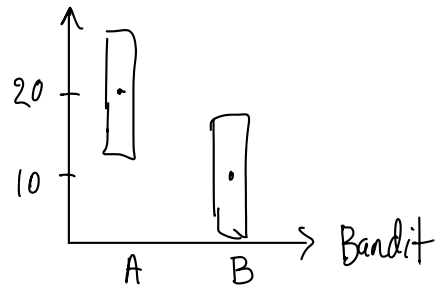
$$A_t \doteq \underset{a}{\text{argmax}}\ Q_t(a).$$

// pick the action a based on our current estimates of all the actions, such that action a is the one with the highest estimated reward.

ε-Greedy Action Selection: Behave greedily except for a small proportion ε, during which the action is selected at random with equal probability (including the greedy action).

$\varepsilon \in [0, 1]$.

Let's say:

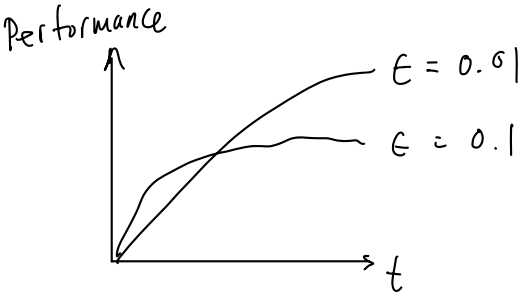


Init: $A = B = 0$ // both are equal.

In $\epsilon = 0$: Let's say we pull B first, and got:

$B = 15$ → we are stuck w/ B since $B > 0$ (initial value).

$\Rightarrow \epsilon = 0$ is Not a good strategy.



However, ϵ too big is not good bc too much exploration:

Mitigation strategies

- Set $\epsilon = 0$ after certain point.
- Decaying ϵ over time.

2.3 : The 10-armed testbed

TLDR; ε-greedy > greedy.

2.4: Incremental Implementation

How can the computation of the $Q_t(a)$ estimates be done in a computationally effective manner?

For a single action, the estimate of this action value after it has been selected $n-1$ times is:

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$ // average the rewards.

We're not going to store all the values of R to recompute the sum at every time step, so we use the following update rule:

$$Q_{n+1} \doteq Q_n + \frac{1}{n}\left[\underbrace{R_n}_{\text{Reward at } t=n} - \underbrace{Q_n}_{\text{Estimate of Q at } t=n}\right]$$ // derivation in the book,

The general form is:

New Estimate $\leftarrow$ Old Estimate + $\underbrace{\text{Step Size}}_{\text{learning rate}}$ $\left[\underbrace{\text{Target} - \text{Old Estimate}}_{\text{error in the estimate.}}\right]$

This applies to value function, action-value functions.

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Loop forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow \mathit{bandit}(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\big[R - Q(A)\big]$

2.5: Tracking a non-stationary problem

- Now the reward probabilities can change over time.
- we want to give more weight to the most recent rewards.

One easy way to do this is to use a constant step size so Q_{n+1} becomes a weighted average of past rewards and the initial estimate Q_n:

$$Q_{n+1} \doteq Q_n + \alpha \left[R_n - Q_n \right]$$

That is equivalent to:

$$Q_{n+1} = \underbrace{(1-\alpha)^n}_{<1} Q_1 + \sum_{i=1}^{n} \alpha (1-\alpha)^{n-i} R_i$$

Q_1: initial estimate

// exponential recency-weighted average

- sum of weights add up to 1
- weight given to R_i depends on how many rewards ago $(n-i)$ it was observed.
- since $(1-\alpha) < 1$, the weight given to R_i decreases as the number of intervening rewards increases.

criteria for α (learning rate schedule):

C1) $\sum_{n=1}^{\infty} \alpha_n = \infty$

C2) $\sum_{n=1}^{\infty} \alpha_n^2 < \infty$

Ex: $\alpha_n = \frac{1}{n}$: converges. // sample-average case.

C1) Harmonic series $\rightarrow$ diverges.

C2) converges to $\pi^2/6$.

Ex: α = constant.

C1) $\infty \cdot C = \infty$

C2) $\infty \cdot C^2 = \infty$

2.6: Optimistic initial values

- Setting optimisticly high initial values is a way to encourage exploration, which is effective in stationary problems.
- Select initial estimates to a high number (higher than the reward we can actually get in the environment), so that unexplored states have a higher value than explored ones, and are selected more often.
- How to set the values? What is optimistic?

2.7: Upper-Confidence-Bound (UCB) Action Selection

- Another way to tackle the exploration problem.
- ε-greedy methods choose randomly the actions.
- maybe there is a better way to select those actions according to their potential of being optimal:

$$A_t \doteq \underset{a}{\text{argmax}} \left[Q_t(a) + c \sqrt{\frac{\ln t}{N_t(a)}} \right]$$

- A_t: next action to choose
- argmax$_a$: choose action a that has highest value
- $Q_t(a)$: exploitation part. Favors actions that work well.
- $\sqrt{\frac{\ln t}{N_t(a)}}$: Exploration bonus. measure of uncertainty or variance in estimate of a's value.
- c: Constant that controls how much to explore.

can use var(Q(a)) instead of ln t.

- $Q_t(a)$: average observed reward for action a.
- ln t increases at each time step.
- $N_t(a)$ is the # of times action a has been selected prior to the time t, so it increases every time a is selected.
- $c > 0$, is our confidence level that controls the degree of exploration.
  - $c = 0$: pure exploitation
  - $c = \infty$: pure exploration.

UCB balances exploitation (choosing actions with high average rewards) and exploration (trying actions we are uncertain about).

The quantity being max'ed over is thus a sort of upper bound on the possible true value of action a, with c determining the confidence level.

- Each time a is selected, the uncertainty is reduced: $N_t(a)$ increments and uncertainty term decreases.
- Each time another action is selected, $\ln t$ increases but not $N_t(a)$, so the uncertainty increases.

2.8 : Gradient Bandit Algorithms

- Another way to select actions.
- So far we estimate action-values (q-values) and use these to select actions.
- Here we compute a numerical preference for each action a, which we denote $H_t(a)$:

$$H_t(a) \in \mathbb{R}.$$

- we select the action with a softmax. $\pi_t(a)$.

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a)$$

$\pi_t(a)$: probability of taking action a at time t.

The algorithm is based on stochastic gradient ascent: at each step t, after selecting action A_t and receiving reward R_t, the action preferences are updated by:

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha\left(R_t - \bar{R}_t\right)\left(1 - \pi_t(A_t)\right)$$

and for all $a \neq A$:

$$H_{t+1}(a) \doteq H_t(a) + \alpha\left(R_t - \bar{R}_t\right)\pi_t(a)$$

- α is the step size
- $\bar{R}_t$ is the average of all rewards up through and including time t.

The $\bar{R}_t$ term serves as a baseline with which the reward is compared. If the reward is higher than the baseline, then the probability of taking A_t in the future is increased, and if the reward is below the baseline, then the probability is decreased. The non-selected actions move in the opposite direction.

$$\frac{d\,E[R]}{d\,H(a)} = \sum_x q_*(x) \frac{d\,\pi(x)}{d\,H(a)} \quad // X = \{\text{set of possible actions}\}$$

- $\sum_x$: sum over all possible actions
- $q_*(x)$: how good is action x?
- $\frac{d\,\pi(x)}{d\,H(a)}$: how probability change?

SGD:

$E[R|x] = q_*(x)$ // Expected reward of action x = true reward of action x.

1) variance of the gradient matters. Need a baseline.
2) What's the baseline? Why are we allowed to subtract it?

$\sum_x \frac{d\,\pi(x)}{d\,H(a)} = 0$. since we know that the preferences are fixed sum, increasing one decreases another.

sum of gradients = 0.

$\sum_x \pi(x) = 1$ // Policy is a probability distribution.

$\sum [R - b] \frac{d\pi}{dH}$ // The influence of b on expectation is 0. Hence, there is no bias.

3) Why do subtracting the baseline actually reduce the variance itself? See Below!

Ex: 2-arm bandits: Bandit 1 and 2.

Ground truth means are the same:

$\mu(1) = \mu(2) = 9.$

Initialize: $H(1) = 1$ $\pi(1) = 0.5$ $H(2) = 1.$ $\pi(2) = 0.5$

$$p(a) = \frac{e^{H(a)}}{\sum_x e^{H(x)}}$$

Let $\alpha = 1$, $B = 9$.

No Baseline

Sample $a = 1$, $r = 8$.

$H'(1) = H(1) + \alpha r (1 - \pi(1))$
$= 1 + (8)(5) = 5$

$H'(2) = H(2) - \alpha r (\pi(2))$
$= 1 - 8(0.5) = 3$

$\pi'(1) = e^5 / [e^5 + e^{-3}]$
$= 0.9997$

wild swing.

$\pi'(2) = 1 - \pi'(1) = 0.0003$

Baseline

Sample $a = 1$, $r - b = -1$

$H'(1) = 1 + (-1)(0.5) = 0.5$

$H'(2) = 1 - (-1)(0.5) = 1.5$

$\pi'(1) = 0.269$

$\pi'(2) = 0.731$

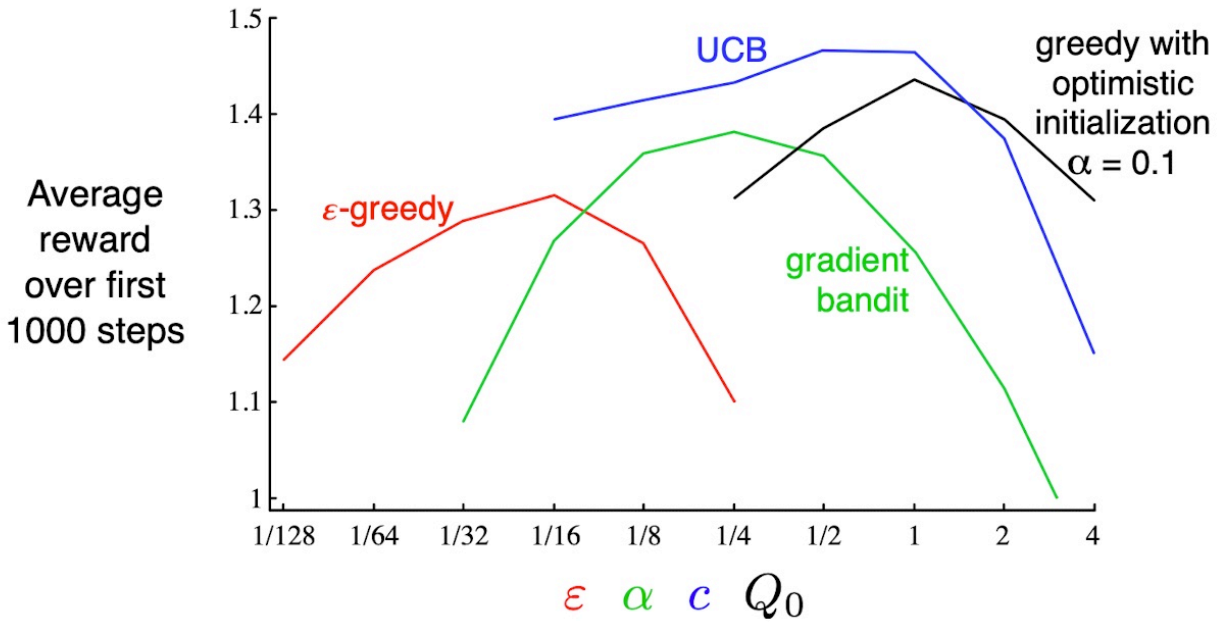
Swing isn't as bad.

2.9 : Associative Search (contextual bandits)

- Nonassociative tasks : no need to associate different actions with different situations (states).

2.10 : Summary

- One key topic is balancing exploitation vs exploration.
- Exploitation is straight forward : select action with the highest estimated value.
- Exploration is trickier, and we have seen several ways to deal with it :
  - ε-greedy choose randomly.
  - UCB favors the more uncertain actions.
  - gradient bandit estimate a preference instead of action values.



2.11 : Thomson Sampling (lecture)

All the explorations are costly exploration.

The exploration cost is known as regret.

Regret : expected loss after n actions due to the fact that you sometimes act suboptimally.

UCB1 : logarithmic regret,

Thompson sampling : "

Does not extend to the full reinforcement learning problem.

Free exploration : up to a certain time n,
explore the options.

- Stop pulling the bandit you know is the worse
- " the best.