Evaluative Feedback

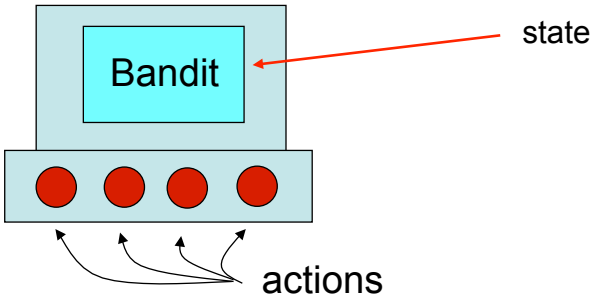
Suggested reading:

Chapter 2 in R. S. Sutton, A. G. Barto: Reinforcement Learning: An Introduction
MIT Press, 1998.

RL learning

- **Evaluates** actions rather **instructs** by giving correct actions
- Pure evaluative feedback depends totally on the action taken. Pure instructive feedback depends not at all on the action taken.
- Evaluative feedback indicates how good the action is, but not if it is the best or worst action possible.
- Supervised learning is instructive; optimization is evaluative
- **Associative** vs. **Non-associative**:
  - Associative: inputs mapped to outputs; learn the best output **for each** input
  - Non-associative: “learn” (find) one best output
- *n*-armed bandit (at least how we treat it) is:
  - Non-associative
  - Evaluative feedback
  - Learning and action selection in a single situation

The *n*-Armed Bandit Problem



A simple machine learning problem based on an analogy with a traditional slot machine (one-armed bandit) but with more than one lever.
The objective of the gambler is to maximize the collected reward sum through iterative pulls.

- Repeatedly choose among *n* different actions
- Each lever provides a reward drawn from a distribution associated with that specific lever.
- Each action has an expected or mean reward (given the action), called the *value* of the action.
- Only estimates of the values are available.
- A *greedy* action is the action whose estimated value is the greatest.
- For many plays it might be better to explore non-greedy actions to discover which of them are better than the greedy action.
- This results in a conflict between exploration and exploitation.

The *n*-Armed Bandit Problem

$Q^*(a)$: true value of the action a

$Q_t(a)$: estimated value of the action a at the t^{th} play

- Choose repeatedly from one of *n* actions; each choice is called a **play**
- After each play a_t, you get a reward r_t, where

$$E\left\{r_t \mid a_t\right\} = Q^*(a_t)$$

These are unknown **action values**
Distribution of r_t depends only on a_t

- Objective is to maximize the reward in the long term, e.g., over 1000 plays

To solve the *n*-armed bandit problem, you must **explore** a variety of actions and the **exploit** the best of them

The Exploration/Exploitation Dilemma

- Suppose you form estimates

$$Q_t(a) \approx Q^*(a) \qquad \textbf{action value estimates}$$

- The **greedy** action at *t* is

$$a_t^* = \arg\max_a Q_t(a)$$

$$a_t = a_t^* \Rightarrow \text{exploitation}$$

$$a_t \neq a_t^* \Rightarrow \text{exploration}$$

- You can't exploit all the time; you can't explore all the time
- You can never stop exploring; but you should always reduce exploring

Action-Value Methods

Methods that adapt action-value estimates and nothing else

e.g.: suppose by the *t*-th play, action a had been chosen k_a times, producing rewards $r_1, r_2, \ldots, r_{k_a}$, then the estimated value of action a is

$$Q_t(a) = \frac{r_1 + r_2 + \ldots + r_{k_a}}{k_a}$$

"sample average"

$$k_a = 0: \qquad Q_t(a) = 0$$

For many plays the estimated action-value converges to the true action-value:

$$\lim_{k_a \to \infty} Q_t(a) = Q^*(a)$$

Greedy and ε-Greedy action selection

Greedy action selection on play *t*:

$$a_t = a_t^* = \arg\max_a Q_t(a)$$

- ε-Greedy:

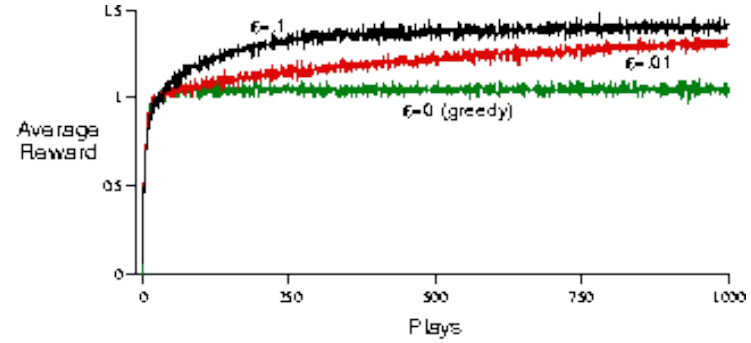
$$a_t = \begin{cases} a_t^* & \text{with probability } 1-\varepsilon \\ \text{random action with probability } \varepsilon \end{cases}$$

. . . the simplest way to try to balance exploration and exploitation and to ensure that all actions have been selected.

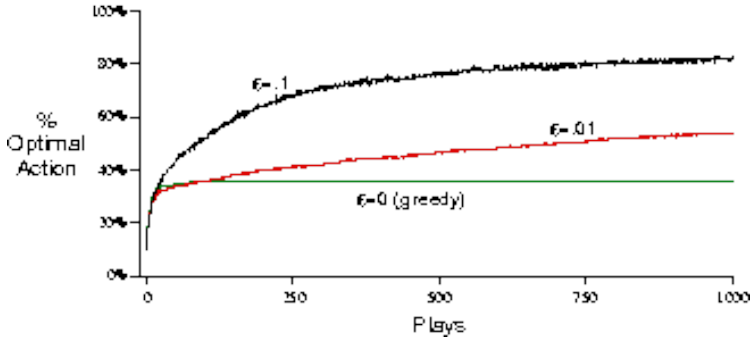
10-Armed Testbed

- *n* = 10 possible actions
- Each $Q^*(a)$ is chosen randomly from a normal distribution: $N(0,1)$
- each r_t is also normal: $N(Q^*(a_t),1)$
- 1000 plays
- repeat the whole thing 2000 times and average the results

ε-Greedy Methods on the 10-Armed Testbed



The greedy method improved slightly faster at the very beginning but then leveled off at a lower level, because it often gets stuck performing suboptimal actions.



The low probability to take random actions is initially slow but typically performs best in the long run.

ε-Greedy Methods on the 10-Armed Testbed

The advantage of ε-greedy over greedy methods depends on the task.

For example, suppose the reward variance had been larger, say 10 instead of 1. With noisier rewards it takes more exploration to find the optimal action, and -greedy methods should fare even better relative to the greedy method.

On the other hand, if the reward variances were zero, then the greedy method would know the true value of each action after trying it once. In this case the greedy method might actually perform best because it would soon find the optimal action and then never explore.

Greedy and ε-Greedy action selection

Disadvantage:

When it explores it chooses equally among all actions. This means that it is as likely to choose the worst-appearing action as it is to choose the next-to-best action.

Softmax Action Selection

Softmax action selection methods grade action probs. by estimated values.

The most common softmax uses a Gibbs, or Boltzmann, distribution:

$$\text{Choose action } a \text{ on play } t \text{ with probability}$$

$$\text{P(a)} = \frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}},$$

$$\text{where } \tau \text{ is the “computational temperature”}$$

For $\tau \rightarrow 0$ softmax action selection becomes the same as greedy action selection

Binary Bandit Tasks

Suppose you have just **two** actions: $a_t = 1$ or $a_t = 2$

and just **two** rewards: $r_t = success$ or $r_t = failure$

Then you might infer a **target** or **desired action**:

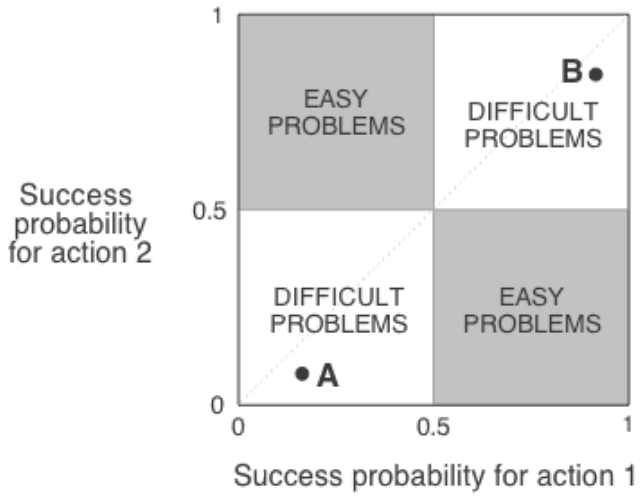
$$d_t = \begin{cases} a_t & \text{if } success \\ \text{the other action} & \text{if } failure \end{cases}$$

and then always play the action that was most often the target

Call this the **supervised algorithm**
It works fine on deterministic tasks…

Contingency Space

The space of all possible binary bandit tasks:



For the easy problems, the probability of success for the better action is greater than 0.5 and the probability of success for the poorer action is less than 0.5. For these tasks, the action inferred to be correct will actually be the correct action more than half the time.

However, consider a task with success probabilities 0.1 and 0.2, corresponding to point A.

In A both actions produce failure at least 80% of the time, any method that takes failure as an indication that the other action was correct will oscillate between the two actions, never settling on the better one.

Linear Learning Automata

Let $\pi_t(a) = \mathrm{P}\{a_t = a\}$ the probability of selecting an action a in play t.

$L_{R\text{-}P}$ (Linear, reward - penalty)

On *success*: $\pi_{t+1}(a_t) = \pi_t(a_t) + \alpha(1 - \pi_t(a_t)) \qquad 0 < \alpha < 1$

(the other action probs. are adjusted to still sum to 1)

On *failure*: $\pi_{t+1}(a_t) = \pi_t(a_t) + \alpha(0 - \pi_t(a_t)) \qquad 0 < \alpha < 1$

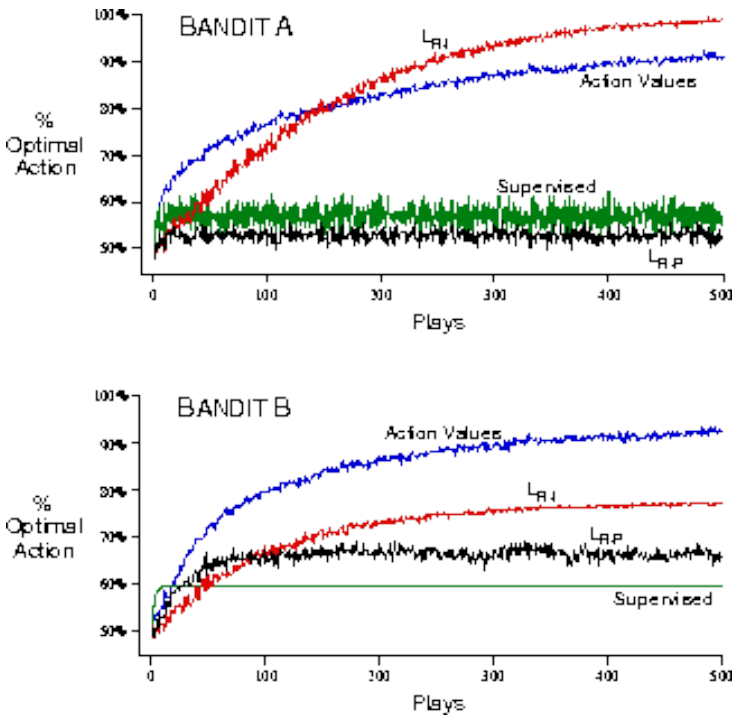
$L_{R\text{-}I}$ (Linear, reward - inaction)

On *success*: $\pi_{t+1}(a_t) = \pi_t(a_t) + \alpha(1 - \pi_t(a_t)) \qquad 0 < \alpha < 1$

On *failure*: no change

For two actions, a stochastic, incremental version of the supervised algorithm

Performance on Binary Bandit Tasks A and B



Incremental Implementation

Recall the sample average estimation method:

$$Q_k = \frac{r_1 + r_2 + \ldots + r_k}{k}$$

The average of the first *k* rewards is (dropping the dependence on *a*):

Can we do this incrementally (without storing all the rewards)?

We could keep a running sum and count, or, equivalently:

$$Q_{k+1} = Q_k + \frac{1}{k+1}\left[r_{k+1} - Q_k\right]$$

This is a common form for update rules:

NewEstimate = *OldEstimate* + *StepSize*[*Target* – *OldEstimate*]

Incremental Implementation

$$\begin{aligned}
Q_{k+1} &= \frac{1}{k+1}\sum_{i=1}^{k+1} r_i \\
&= \frac{1}{k+1}\left(r_{k+1} + \sum_{i=1}^{k} r_i\right) \\
&= \frac{1}{k+1}\left(r_{k+1} + kQ_k + Q_k - Q_k\right) \\
&= \frac{1}{k+1}\left(r_{k+1} + (k+1)Q_k - Q_k\right) \\
&= Q_k + \frac{1}{k+1}\left[r_{k+1} - Q_k\right]
\end{aligned}$$

Tracking a Nonstationary Problem

Choosing Q_k to be a sample average is appropriate in a stationary problem,

i.e., when none of the $Q^*(a)$ change over time,

But not in a nonstationary problem.

Better in the nonstationary case is:

$$Q_{k+1} = Q_k + \alpha\left[r_{k+1} - Q_k\right]$$

for *constant* α, $0 < \alpha \leq 1$

$$= (1-\alpha)^k Q_0 + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} r_i$$

initial estimate

exponential, recency-weighted average

Nonstationary Problem

$$\begin{aligned}
Q_k &= Q_{k-1} + \alpha\left[r_k - Q_{k-1}\right] \\
&= \alpha r_k + (1-\alpha) Q_{k-1} \\
&= \alpha r_k + (1-\alpha)\alpha r_{k-1} + (1-\alpha)^2 Q_{k-2} \\
&= \alpha r_k + (1-\alpha)\alpha r_{k-1} + (1-\alpha)^2 \alpha r_{k-2} + \ldots \\
&\quad + (1-\alpha)^{k-1} \alpha r_1 + (1-\alpha)^k Q_0 \\
&= (1-\alpha)^k Q_0 + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} r_i
\end{aligned}$$

This results in Q_k being a weighted average of past rewards and the initial estimate Q_0.

Nonstationary Problem

Note that $(1-\alpha)^k + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} = 1$

The weight $\alpha(1-\alpha)^{k-i}$ given to the reward r_i

depends on how many rewards ago $k-i$ it was observed.

Optimistic Initial Values

Stationary case:

$$Q_{k+1} = Q_k + \frac{1}{k+1}\left[r_{k+1} - Q_k\right]$$

Nonstationary case:

$$Q_{k+1} = (1-\alpha)^k Q_0 + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} r_i \qquad \alpha = \text{constant}$$

The nonstationary case depends on $Q_0(a)$, i.e., it is **biased.**

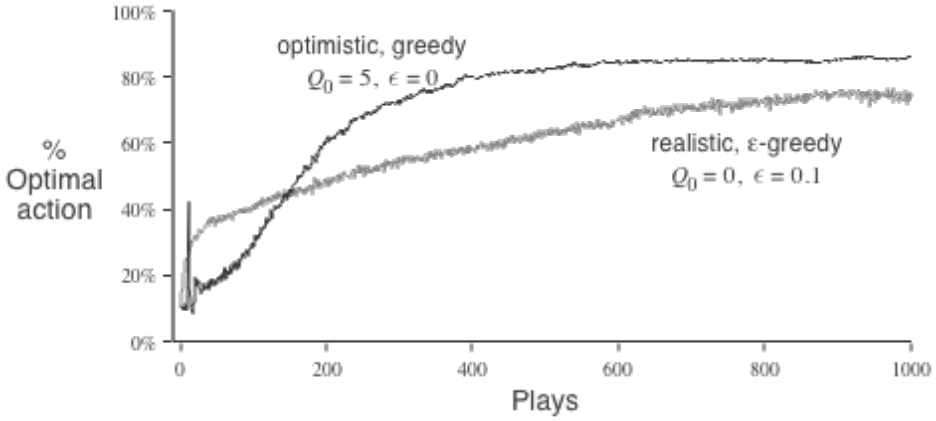
- usually not a problem in practical applications
- additional parameter to be set by the user
- allows to incorporate prior knowledge

Optimistic Initial Values

Initial action values can also be used as a simple way of encouraging exploration.

- Suppose we initialize the action values **optimistically**, i.e., on the 10-armed testbed, use

$$Q_0(a) = 5 \quad \text{for all } a$$



Exploration: Whichever actions are initially selected, the reward is less than the starting estimates. The result is that all actions are tried several times before the value estimates converge. What about nonstationary problems?

Reinforcement Comparison

A central intuition underlying reinforcement learning is that actions followed by large rewards should be made more likely to recur.

What is a small or large reward?

Solution: Compare rewards to a *reference reward*, e.g., an average of previously received rewards.

Learning methods based on this idea are called *reinforcement comparison* methods.

Reinforcement comparison methods typically do not maintain estimates of action values, but only of an overall reward level. In order to pick among the actions, they maintain a separate measure of their preference for each action.

Reinforcement Comparison

- Compare rewards to a reference reward $\bar{r}_t$
- Strengthen or weaken the action taken depending on $r_t - \bar{r}_t$
- Let $p_t(a)$ denote the **preference** for action a
- Preferences determine action probabilities

- The probability of taking an action a on the t*th* play is:

$$\pi_t(a) = \Pr\{a_t = a\} = \frac{e^{p_t(a)}}{\sum_{b=1}^{n} e^{p_t(b)}}$$

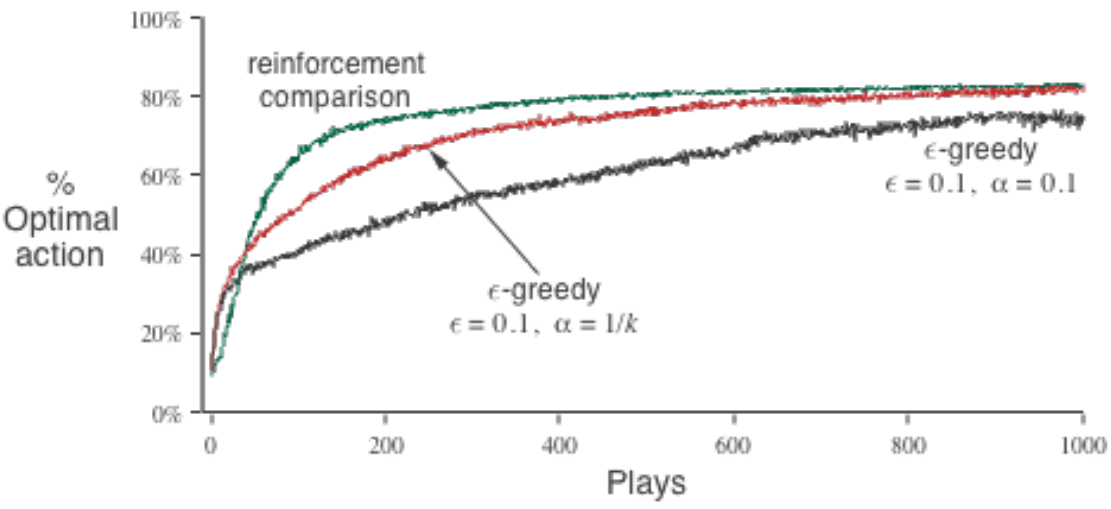
Gibbs distribution

- After each play, the preference for the action selected on that play, is incremented by the difference between the reward, and the reference reward:

$$p_{t+1}(a_t) = p_t(a) + \beta\left[r_t - \bar{r}_t\right] \quad \text{and} \quad \bar{r}_{t+1} = \bar{r}_t + \alpha\left[r_t - \bar{r}_t\right]$$

Performance of a Reinforcement Comparison Method

Reinforcement comparison methods can be very effective, sometimes performing even better than action-value methods.



Pursuit Methods

- Maintain both action-value estimates and action preferences
- Always "pursue" the greedy action, i.e., make the greedy action more likely to be selected
- After the *t*-th play, update the action values to get Q_{t+1}
- The new greedy action is $a^*_{t+1} = \arg\max_a Q_{t+1}(a)$

- Increase the probability of selecting $a_{t+1} = a^*_{t+1}$:

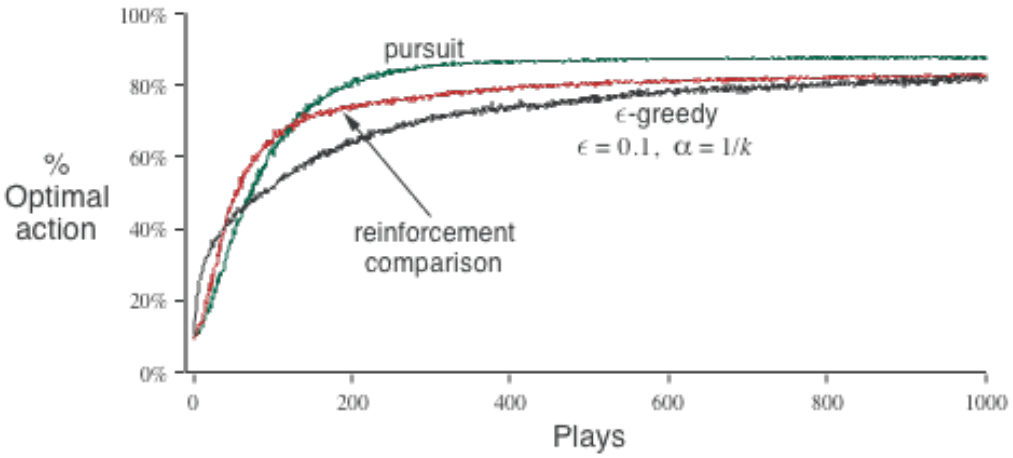
$$\pi_{t+1}(a^*_{t+1}) = \pi_t(a^*_{t+1}) + \beta\left[1 - \pi_t(a^*_{t+1})\right]$$

Here, the action values are set equal to the probabilities $\pi_t(a)$ with which action a is selected

and the probs. of the other actions decremented to maintain the sum of 1:

$$\pi_{t+1}(a) \quad = \quad \pi_t(a) + \beta\left[0 - \pi_t(a)\right] \quad \forall a \neq a^*_{t+1}$$

Performance of a Pursuit Method



Performance of the pursuit algorithm when the action values are estimated using sample averages. Although the pursuit algorithm performs the best of these three on this task at these parameter settings, the ordering could well be different in other cases. All three of these methods appear to have their uses and advantages.

Conclusions

- These are all very simple methods
  - but they are complicated enough—we will build on them
- Ideas for improvements:
  - estimating uncertainties . . . interval estimation
  - approximating Bayes optimal solutions
  - Gittens indices

    classical solution to balancing exploration and exploitation in n-armed bandit problems
- The full RL problem offers some ideas for solution . . .