Introduction to Reinforcement Learning

Lecture Notes

Tetiana Bogodorova

Ukrainian Catholic University
Computer Science Programme

2rd term
Spring 2018

APPLIED SCIENCES FACULTY

Lecture 1[1]. Introduction to Reinforcement Learning. Problem Formulation

[1]Based on CS8803-003 Reinforcement Learning by Georgia Tech and Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. Vol. 1. No. 1. Cambridge: MIT press, 1998, 2017

Outline

What is Reinforcement Learning Problem?

Supervised Learning is about knowing pairs $x_1 \rightarrow y_1$, $x_2 \rightarrow y_2$, ..., $x_n \rightarrow y_n$ (a knowledgeable external supervisor). What is the function $f(\cdot)$ that defines dependency between pairs and what is y_{n+1} value given x_{n+1}? It is a function approximation problem.
Unsupervised learning is about knowing $x_1, x_2, ..., x_n$ to learn $f(\cdot)$ that describe these data. It is called clustering description.

Reinforcement Learning is about knowing state s (analogue of x) and reward r, learn $f(\cdot)$ and action a (analogue of y) (learning from agent's own experience).

History

Three directions in science

- Idea of trial-and-error learning from the psychology: selectional, associative, combining search and memory
- Optimal control from control theory - the problem of designing a controller to minimize a measure of a dynamical system's behavior over time.
- Thread concerning temporal-difference methods - originates from notion of secondary reinforcers in psychology

Reference: "Steps Toward Artificial Intelligence" (Minsky, 1961)

Examples

- Playing chess
- An adaptive controller adjusts parameters of the industrial process optimising the yield/cost/quality trade-off
- A robot-trash collector decides whether to open a new door or be back to charge using the previous knowledge on how fast it returned before.
- A baby learns to walk repeating actions to stand up

Common features

Agent-environment interaction, a goal, uncertainty, affecting the future, ability to judge progress to the goal

What is Reinforcement Learning (RL) Problem?

Learner is a decision-making agent

Goal-directed learning from agent-environment interaction

The agent takes action that affects a state of an environment

The environment returns state and reward

The action has to be learned via reward maximization

Two features of RL: trial-and-error search and delayed reward

Reinforcement Learning

Characterized by a goal-directed learning problem

The Agent-Environment Interface

- Exploration-exploitation dilemma (in control: estimation-control)
- Whole problem consideration: to maximize the total amount of reward that received over the long run

Agent-Environment Boundary

The general rule: anything that cannot be changed arbitrarily by the agent is considered to be outside of it and thus part of its environment.

Elements of Reinforcement Learning

Subelements

A policy, a reward function, a value function, a model of environment (optional)

- A state - situation that characterise environment the agent interacts with
- A policy - a mapping from perceived states to actions when in those states (instruction to do an action when being in some state)
- A reward function - defines the goal by mapping each perceived state or state-action pair with some number - reward. Unalterable by the agent.
- A value function - defines the total amount of reward that can be expected to accumulate in the future starting from the particular state.
- A model - mimics behaviour of environment, used for planning

Evolutionary methods vs. A value function evaluation

Evolutionary methods: search methods such as genetic algorithms, simulated annealing, etc.

Never evaluate value function, only policies

Works when: space of policy is small, the agent cannot accurately sense the state.
Doesn't track policy dependency on state-action pair $\rightarrow$ search is not efficient.

Example

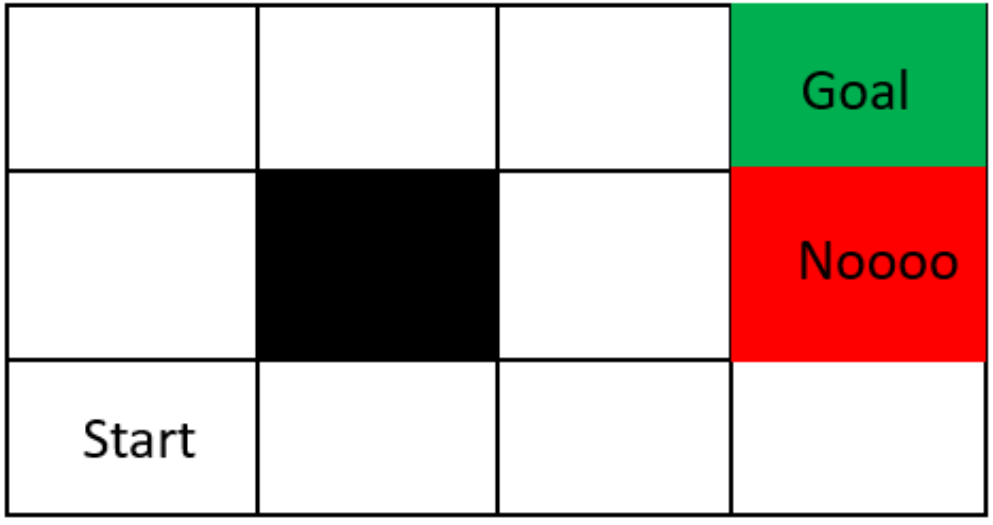
If the player wins, then all of its behaviour in the game is given credit, independently of how specific moves might have been critical to the win. Credit is even given to moves that never occurred!

Value function methods

in contrast, allow individual states to be evaluated.

Example: Moving in the world

You may use U, D, L and R for Up, Down, Left, and Right (respectively) instead of full words. Give only one sequence.
What is the shortest sequence getting from Start to Goal?



What if probability that action executes is 0.8, when moving at right angle is 0.1. What is reliability (probability of success) of sequence UP, UP, RIGHT, RIGHT, RIGHT? (Hint: there are two ways to get to the goal)

K-Armed Bandit Problem

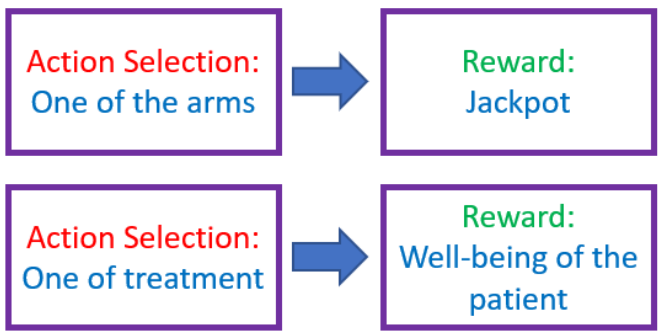
K-Armed Bandit

Stands from a hypothetical slot machine with K levels.

10-armed testbed:

- 2000 rand. generated k-armed bandit problems with k=10.
- For each bandit problem, the action values, $Q^*(a)$, $a = 1, ..., 10$, were selected according to a normal distribution with mean 0 and variance 1.
- When a learning method applied to that problem selected action A_t at time step t, the actual reward, R_t, was selected from a normal dist. N($Q^*(A_t)$,1).

Figure: Credit to https://vwo.com/blog/ multi-armed-bandit-algorithm/



Action-Value Methods: Value estimation

Action-Value Methods

Methods for estimating the values of actions, and therefore, making action selection decisions.

Let a - action, r - reward, $Q^*(a)$ - true value of action, $Q_t(a)$ - estimated value at t^{th} play, then

Sample-average method

$Q_t(a) = \frac{r_1 + r_2 + \ldots + r_{k_a}}{k_a}$, k_a - number of times a has been chosen prior to t^{th} play.

By the law of large numbers:

As $k_a \to \infty$, $Q_t(a) \to Q^*(a)$.

Action-Value Methods: Action selection

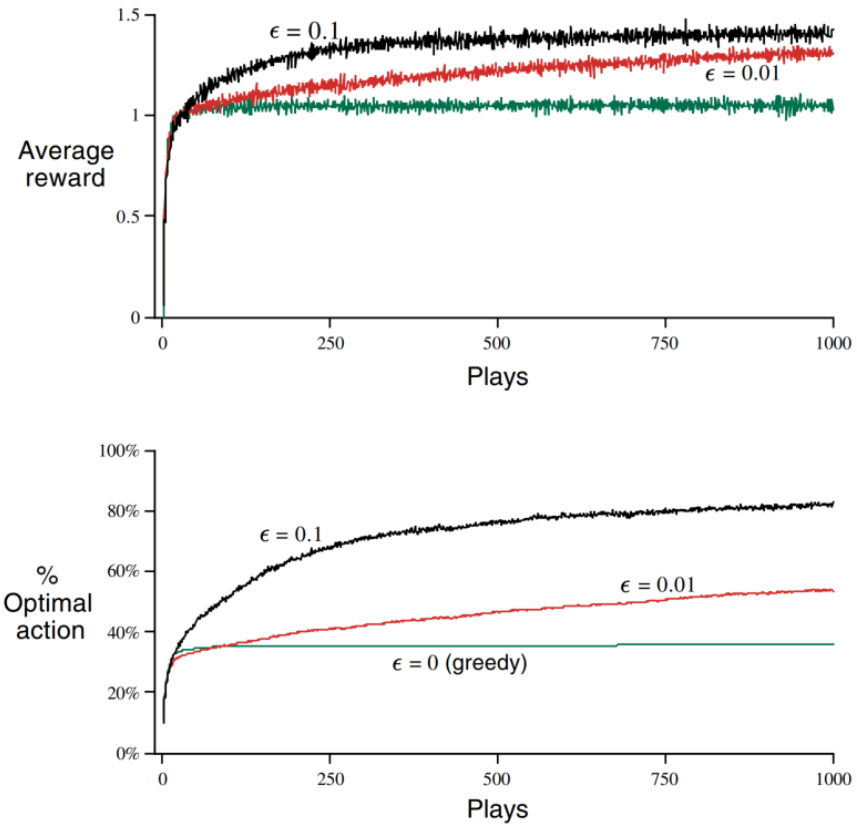
Exploitation is done by choosing the best (greedy) action given available information about the actions.

Action selection rule:

$Q_t(a^*) = max_a Q_t(a)$, where a^* - greedy action

Sometimes (with small probability ϵ) choosing exploration: an action at random, uniformly, independently of the action-value estimates, called ϵ-greedy method.
The probability of selecting the optimal action converges to $1 - \epsilon$ as $k_a \to \infty$.

ϵ-greedy method assessment



In Figure 2.1 (Barto, 1998: 29) greedy method stuck in performing suboptimal actions.

- How do you think methods will perform if rewards are noisier? deterministic?
- If the true values change in time (nonstationary process)?

Figure: Figure 2.1 (Barto, 1998: 29)

Softmax Action Selection

ϵ-greedy method chooses equally among all actions.

Solution:

softmax action selection that vary the action probabilities $p(a)$ as a graded function of estimated value.

It often uses Gibbs, or Boltzmann, distribution:

$p(a) = \frac{exp(\frac{Q_t(a)}{\tau})}{\Sigma_{b=1}^{n} exp(\frac{Q_t(a)}{\tau})}$, where τ - "temperature" parameter.

As $\tau \to 0$, *softmax* action selection becomes the same as *greedy* action selection.

Additional reading: Tokic, M., & Palm, G. (2011). Value-difference based exploration: adaptive control between epsilon-greedy and softmax. In Annual Conference on Artificial Intelligence (pp. 335-346). Springer, Berlin, Heidelberg.

Evaluation Versus Instruction

In *evaluative* feedback when choosing among actions, correctness is a relative property, while in *instructive* feedback it is absolute, independent of action selected. Function optimization analogy to evaluative feedback:

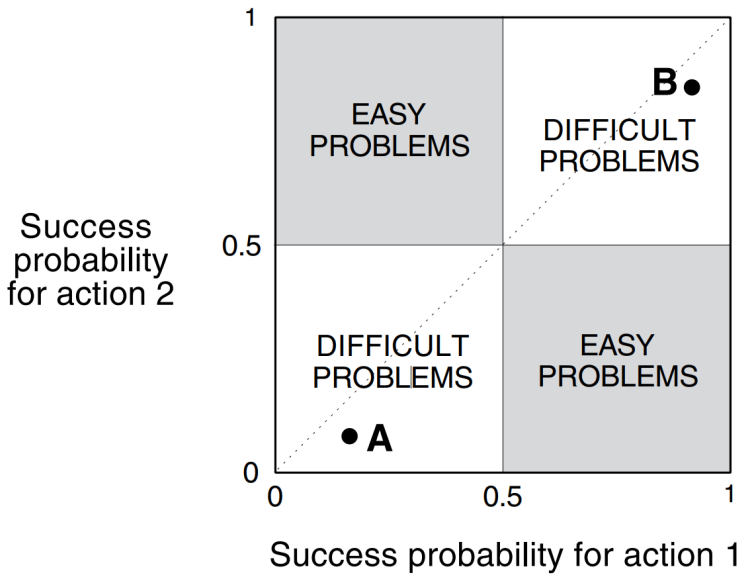


Figure: Fig. 2.2.Regions in space of all binary bandit tasks (Barto, 1998: 34)

Optimization algorithm that uses only function values, corresponding to evaluative information, and has to actively probe the function at additional points in the search space in order to decide where to go next. Classical examples of these types of algorithms are, respectively, the Robbins-Monro and the Kiefer-Wolfowitz stochastic approximation algorithms (see, e.g., Kashyap, Blaydon, and Fu, 1970).

Incremental Implementation (SB2017:24)

Computational requirements to store all the rewards in the form $Q_t(a) = \frac{r_1+r_2+...+r_{k_a}}{k_a}$ grow over time.

The update rule:

$$Q_{k+1} = \frac{1}{k+1}\Sigma_{i=1}^{k+1} r_i = \frac{1}{k+1}(r_{k+1} + \Sigma_{i=1}^{k} r_i = \frac{1}{k+1}(r_{k+1} + kQ_k + Q_k - Q_k) = \frac{1}{k+1}(r_{k+1} + (k+1)Q_k - Q_k) = Q_k + \frac{1}{k+1}[r_{k+1} - Q_k]$$

NewEstimate $\rightarrow$ *OldEstimate*+*StepSize* [*Target* - *OldEstimate*]

A simple bandit algorithm

Initialize, for $a = 1$ to k:
$Q(a) \leftarrow 0$
$N(a) \leftarrow 0$

Loop forever:
$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$
$R \leftarrow bandit(A)$
$N(A) \leftarrow N(A) + 1$
$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}[R - Q(A)]$

Tracking a Nonstationary Problem

For nonstationary: makes sense to weight recent rewards more heavily than long-past ones
Solution: to use a constant step-size α ($0 < \alpha \leq 1$) in:
$Q_{k+1} = Q_k + \alpha[r_{k+1} - Q_k]$

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

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

The weight decays exponentially according to the exponent on $(1-\alpha)$
$\rightarrow$ *exponential, recency-weighted average*.

Tracking a Nonstationary Problem: Derivation

Equality: $Q_k = Q_{k-1} + \alpha[r_k - Q_{k-1}] = (1-\alpha)^k Q_0 + \Sigma_{i=1}^{k} \alpha(1-\alpha)^{k-i} r_i$

Derivation

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

Tracking a Nonstationary Problem: Why weighted average?

Because the sum of the weights:

$(1-\alpha)^k + \Sigma_{i=1}^{k} \alpha(1-\alpha)^{k-i} = 1$

Derivation:

Considering the geometric series summation of the first k terms:
$\Sigma_{k=0}^{k-1} ar^k = a(\frac{1-r^n}{1-r})$;
the sum of the weights can be presented as:

$$(1-\alpha)^k + \Sigma_{i=1}^{k} \alpha(1-\alpha)^{k-i} =$$

$$(1-\alpha)^k + \Sigma_{i=1}^{k} \alpha(1-\alpha)^{k-i} + \alpha(1-\alpha)^k - \alpha(1-\alpha)^k$$

$$= (1-\alpha)^k + \Sigma_{i=0}^{k} \alpha(1-\alpha)^{k-i} - \alpha(1-\alpha)^k =$$

$$(1-\alpha)^k + \alpha(1-\alpha)^k \Sigma_{i=0}^{k} (\frac{1}{1-\alpha})^i - \alpha(1-\alpha)^k =$$

$$\frac{1-(\frac{1}{1-\alpha})^{k+1}}{1-\frac{1}{1-\alpha}} + (1-\alpha)^k - \alpha(1-\alpha)^k =$$

Convergence

The choice of step-size $\alpha_k(a)$ directly influences convergence. From the stochastic approximation theory (hint:e.g. Robbins-Monro algorithm), the conditions to assure the convergence with probability 1: $\Sigma_{k=1}^{\infty} \alpha_k(a) = \infty$ (to guaranty that the steps are large enough to overcome any initial conditions or random fluctuations) and $\Sigma_{k=1}^{\infty} \alpha_k^2(a) < \infty$ (guarantees that eventually the steps become small enough to assure convergence)

$\alpha_k(a)$	$\Sigma_{k=1}^{\infty} \alpha_k(a) = \infty$	$\Sigma_{k=1}^{\infty} \alpha_k^2(a) < \infty$	Converge
$\frac{1}{k}$	Yes	Yes	Yes
α	Yes	No	No

In case of constant step-size, the estimates never completely converge but continue to vary in response to the most recently received rewards.

Optimistic Initial Values

Both methods are biased by initial estimate $Q_0(a)$.

- - Has to be picked by the user
- + Allows to introduce prior knowledge

Initial action values can also be used as a simple way of encouraging exploration. This method called *optimistic initial values*.

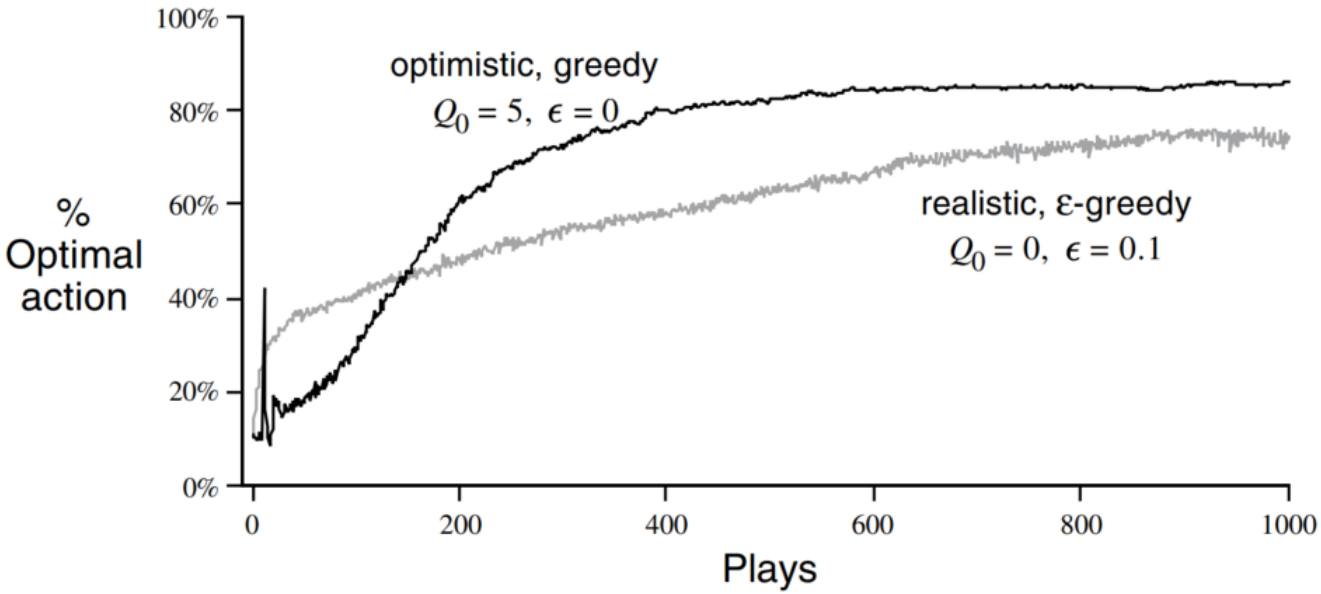


Figure: Figure 2.4 (Barto, 1998: 41)

Reinforcement Comparison learning method

To make judgement on how big a reward, a natural choice for the *reference reward* is an average of previously received rewards.

Then $\pi_t(a)$ - the probability of selecting action a on the tth play:

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

Action preferences ($p_t(a)$ for action a on play t) update:

$$p_{t+1}(a_t) = p_t(a_t) + \beta[r_t - \bar{r}_t], \tag{2}$$

where $\bar{r}_t$- the reference reward, β - a positive step size parameter

The reference reward $\bar{r}_t$ update:

$$\bar{r}_{t+1} = \bar{r}_t + \alpha[r_t - \bar{r}_t] \tag{3}$$

Reinforcement Comparison (cont.)

The initial value of reward $\bar{r}_0$ set:

- Optimistically
- According to prior knowledge

The learning problem is nonstationary (the dist. of rewards changing over time)

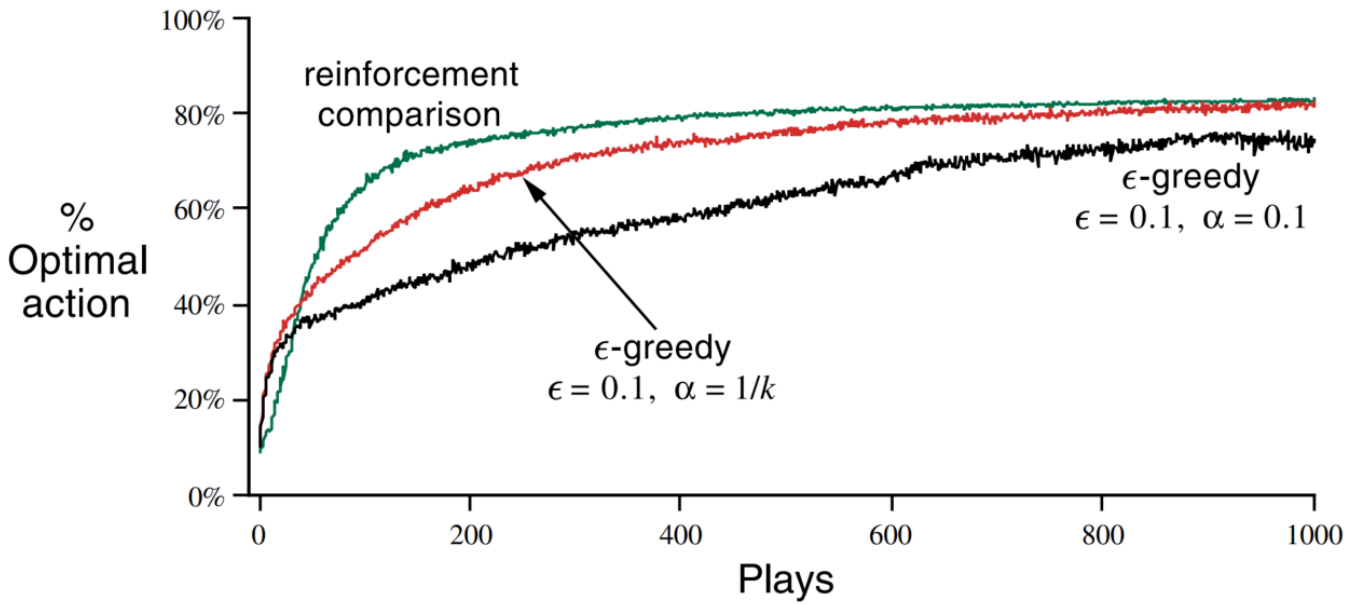


Figure: Figure 2.5 of (Barto, 1998: 43) Comparison of 3 methods on the 10-armed testbed

Pursuit Methods

Pursuit methods maintain both action-value estimates *and* action preferences, with the preferences continually "pursuing" the action that is greedy according to the current action-value estimates.

Denote a^* - greedy action:

$$a^*_{t+1} = argmax_a Q_{t+1}(a) \tag{4}$$

Then the probability of selecting $a_{t+1} = a^*_{t+1}$ is incremented a fraction, β, of the way toward 1:

$$\pi_{t+1}(a^*_{t+1}) = \pi_t(a^*_{t+1}) + \beta(1 - \pi_t(a^*_{t+1})) \tag{5}$$

while the probabilities of selecting other actions:

$$\pi_{t+1}(a_{t+1}) = \pi_t(a_{t+1}) + \beta(0 - \pi_t(a_{t+1})) \text{ for all } a \neq a^*_{t+1} \tag{6}$$

Pursuit Methods: Performance

The action values $Q_{t+1}(a)$ in one of the ways mentioned before, e.g. the sample averages of the observed rewards. It suits only for stationary env..

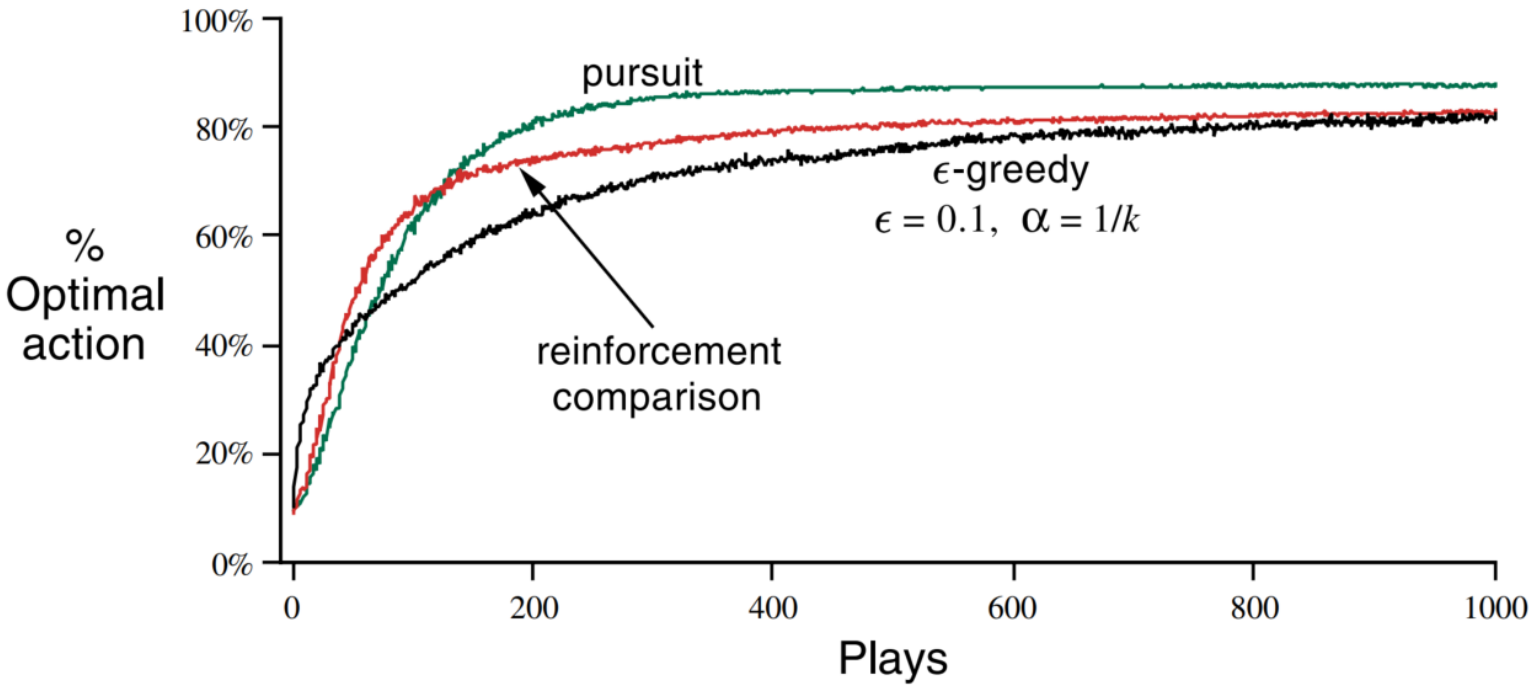


Figure: Figure 2.6 of (Barto, 1998: 44) Comparison of 3 methods on the 10-armed testbed. Pursuit method's settings: the initial action probabilities were $\pi_0(a) = \frac{1}{n}$, for all a, and the parameter $\beta = 0.01$

Upper-Confidence-Bound Action Selection

ϵ-greedy action selection: gives no preference to any of non-greedy actions
UCB: Evaluates both how close their estimates are to being maximal and the uncertainties in those estimates
Select actions according to:

$$A_t = argmax_a \left(Q_t(a) + c\sqrt{\frac{lnt}{N_t(a)}} \right) \tag{7}$$

where $c > 0$ - degree of exploration; $N_t(a)$ - the number of times action a has been selected; the square-root term is a measure of the uncertainty or variance in the estimate of a's value. A_t- upper bound on possible true value of action a

Upper-Confidence-Bound Action Selection: Performance

UCB: Evaluates both how close their estimates are to being maximal and the uncertainties in those estimates

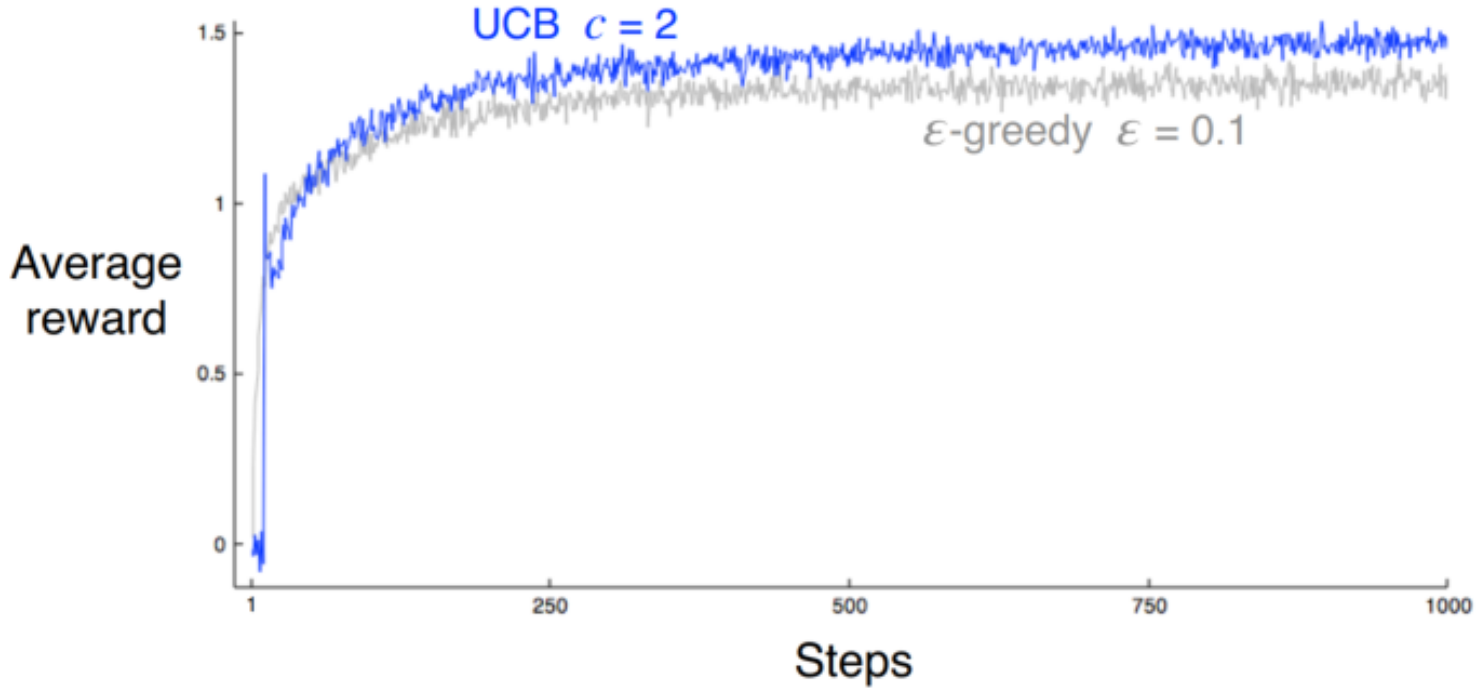


Figure: Figure 2.4 of (Barto, 2017: 28) Comparison of the methods on the 10-armed testbed

Gradient Bandit Algorithms

Let $H_t(a)$ - preferences, initially $H_1(a) = 0$ for all a

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

where $\pi_t(a)$ - the probability of action a at time t. Algorithm based on stochastic gradient ascent idea:

$$H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t)),$$
$$H_{t+1}(a) = H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a), \text{ for all } a \neq A_t$$

where $\alpha > 0$ - a step-size parameter, $\bar{R}_t$- average of all rewards (a baseline)

Gradient Bandit Algorithms

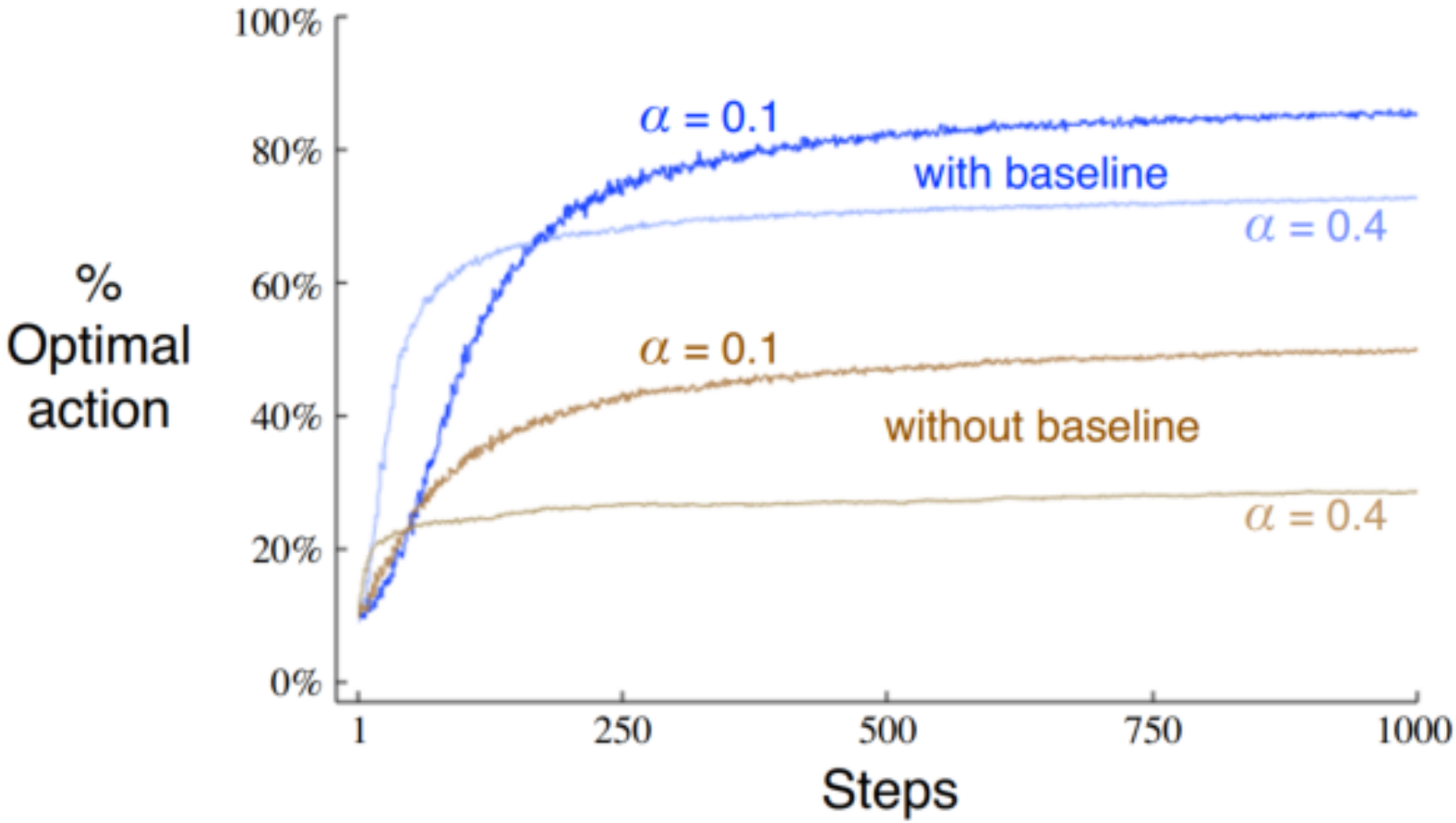


Figure: Figure 2.5 of (Barto, 2017: 29) Comparison of the methods on the 10-armed testbed. Rewards are of norm. dist. with a mean of +4 instead of zero (and with unit variance as before). Without baseline means $\bar{R}_t = 0$

Associative Search

So far non-associative tasks - no need to associate different actions with different situations - are reviewed.
A policy: a mapping from situations to the actions that are best in those situations.
An associative search task involves both:

- *trial-and-error learning* in the form of search for the best actions
- and *association* of these actions with the situations in which they are best.

Associative search tasks are intermediate between *the n-armed bandit problem* and *the full reinforcement learning problem*.
If actions are allowed to *affect* the next situation as well as the reward, then we have *the full reinforcement learning problem*.

Thompson sampling

Bayesian methods assume a known initial distribution over the action values and then update the distribution exactly after each step (assuming that the true action values are stationary).For conjugate priors computation is easy. One possibility is to then select actions at each step according to their posterior probability of being the best action - posterior sampling or Thompson sampling.

- Good empirical performance [Chapelle-Li 2011], [Kaufmann et al. 2012]
- Robust to delayed feedback [Chapelle-Li 2011]
- Easy to implement
- Naturally extendable to more general settings, e.g. contextual bandits [Graepel et al. 2010], [Chapelle-Li 2011]
- Used in industrial applications [Graepel et al. 2010]
- Naturally incorporates partial knowledge (about the arms)

Thompson sampling

Beta-Bernoulli Bandit

- When played, an action k produces a reward of one with probability θ_k and a reward of zero with probability $1 - \theta_k$.
- Each θ_k can be interpreted as an action's success probability or mean reward.
- The mean rewards $\theta = (\theta_1, ..., \theta_k)$ are unknown, but fixed over time.
- In the first period, an action x_1 is applied, and a reward r_1 *in* $\{0, 1\}$ is generated with success probability $P(r_1 = 1 | x_1, \theta) = \theta_x 1$.
- After observing r_1, the agent applies another action x_2, observes a reward r_2, and this process continues.
- The objective is to maximize the cumulative number of successes over T periods, where $T >> K$.

Thompson sampling

Algorithm 2 BernThompson(K, α, β)

for $t = 1, 2, \ldots$ **do**
    #sample model:
    for $k = 1, \ldots, K$ **do**
        Sample $\hat{\theta}_k \sim \text{beta}(\alpha_k, \beta_k)$
    end for

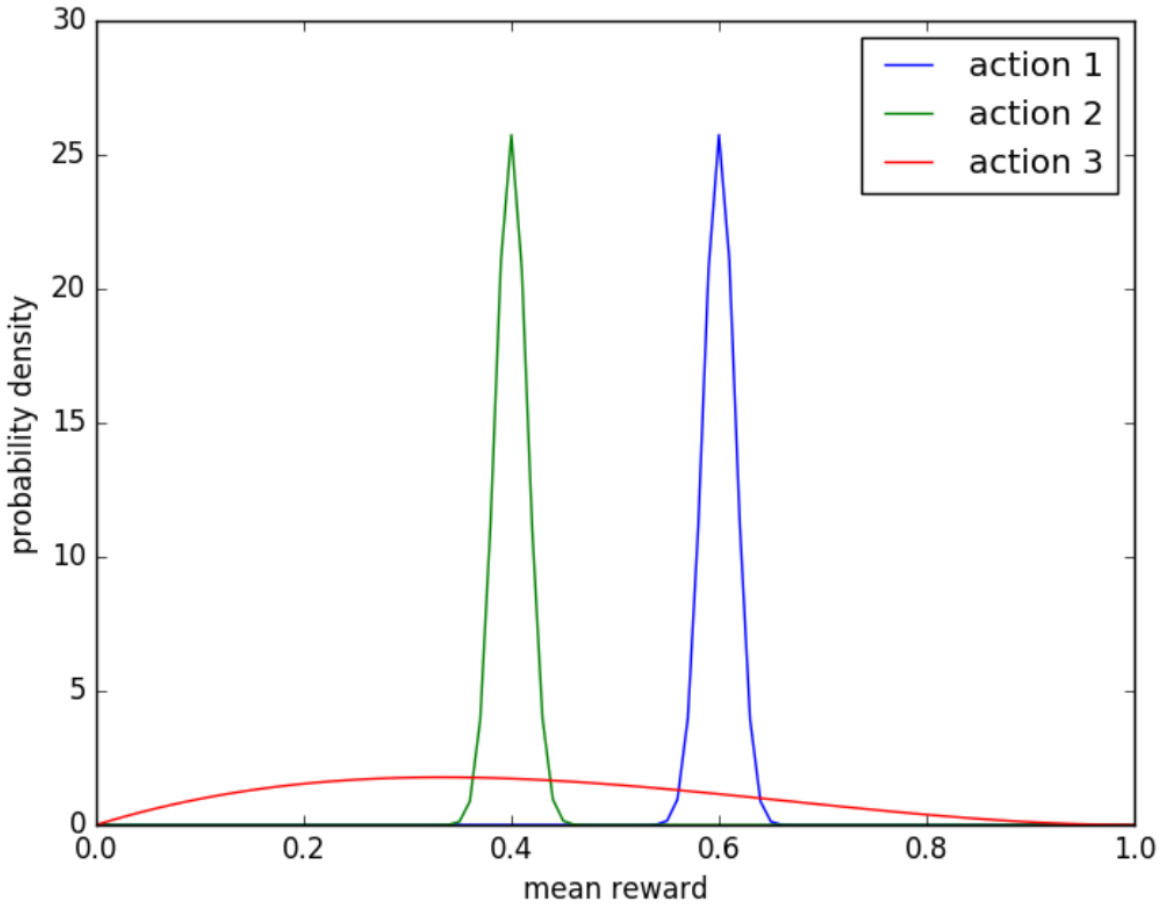
    #select and apply action:
    $x_t \leftarrow \text{argmax}_k \hat{\theta}_k$
    Apply x_t and observe r_t

    #update distribution:
    $(\alpha_{x_t}, \beta_{x_t}) \leftarrow (\alpha_{x_t}, \beta_{x_t}) + (r_t, 1 - r_t)$
end for

Figure: Figure from Russo, Daniel, et al. "A Tutorial on Thompson Sampling." arXiv:1707.02038 (2017)

Thompson sampling

Figure (below) plots probability density functions of beta distributions with parameters $(\alpha_1, \beta_1) = (600, 400)$, $(\alpha_2, \beta_2) = (400, 600)$, and $(\alpha_3, \beta_3) = (4, 6)$.



Thompson sampling

- Maintain belief about effectiveness (mean reward) of each arm
- Observe feedback, update belief of pulled arm *i* in Bayesian manner *Bayesrule* $Pr(\theta_i|r) \propto Pr(r|\theta_i)Pr(\theta_i)$
- Pull arm with posterior probability of being best arm

Thompson Sampling gives "optimal" benefit of doubt [Agrawal and Goyal, COLT 2012, AISTATS 2013]

Applications of Multi-armed bandit

- Clinical trials (Thompson (1933))
- Online advertising: Placement of ads in search results
- Online recommendations: News article recommendation

Conclusions

- Some simple ways of balancing exploration and exploitation were presented.
- *The ϵ-greedy* methods choose randomly a small fraction of the time
- *The softmax methods* grade their action probabilities according to the current action-value estimates
- *The pursuit methods* keep taking steps toward the current greedy action

Can we do better in exploration? Yes, using *interval estimation* methods. These methods estimate for each action a confidence interval of the action's value.
The classical solution to balancing exploration and exploitation in n-armed bandit problems is to compute special functions called *Gittins indices*.