Evaluative Feedback
Week #2

Purely Evaluative Feedback of RL

- says *how favorable* the action taken by agent is.
- does not say what the best or worst action is

$$Q(s,a_{worst})?.....Q(s,a_{t_1}).....Q(s,a_{t_2}).....Q(s,a_{t_3})....Q(s,a_{t_4}).....Q(s,a_{best})?;\quad a_{t_i} \in A_s, i \in N$$

  where A_s is the set of actions valid at state s.
- action selection, thus, is based on the set of actions selected and evaluated so far (i.e., actions a_{t1},..., a_{t4})
  - action with the highest value is taken or given highest chance of selection (*exploitation*)

$$a^* = \arg\max_{a_{t_i}} \{Q(s,a_{t_i})\}$$

  - a chance is given to those actions with small values not to turn off the possibility to miss actions with potentially promising future values (*exploration*)

Purely Instructive Feedback of SL

- indicates the *correct action*
- is *independent of the action taken* as opposed to the purely evaluative feedback which is totally based on the action taken.
- *basis of supervised learning*

Nonassociative Setting

- *Single state* (nonassociative) to keep RL problem simple and focus on the evaluative feedback aspect of RL.
- We will observe here
  - the *differences among evaluative and instructive feedback* as well as
  - *how they can be combined*.
- Simple version of ***n-armed bandit*** problem studied to introduce a number of basic learning methods.
- At the end, we discuss an associative task to get a feel of the full RL problem.

N-Armed Bandit Problem

- Assume *n* bags filled with colored balls. Each bag, contains a different distribution of red, blue and yellow balls.
- We select repetitively among *n* bags to draw a red ball, for instance, and for each choice receive a numerical response from a stationary probability distribution depending on the action (i.e., what color appears on the drawn ball). For red we receive a favorable response and for others an unfavorable one.
- *Objective*: to ***maximize the expected total reward*** over some time.

N-Armed Bandit Problem

- *Action Value Estimation: (sample-average method)*
  - Actual action value: $Q^*(a)$; *mean reward for action a*
  - Estimate at time t (i.e., at t^{th} play): $Q_t(a)$

$$Q_t(a) = \frac{\sum_{i=1}^{k_a} r_i}{k_a}$$

where k_a is the number of times the action a is selected by the t^{th} play.

$$Q_t(a) \xrightarrow[k_a \to \infty]{} Q^*(a)$$

by law of large numbers.

N-Armed Bandit Problem

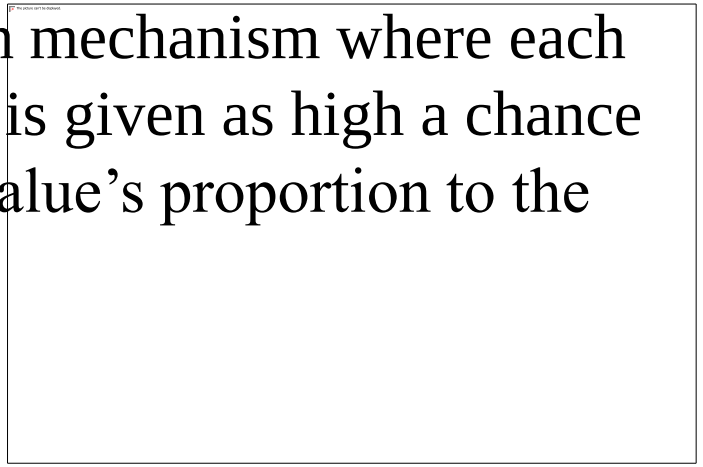
- *A Simple Action Selection Rule: (Exploitation vs Exploration)*
  - *ε-greedy* selection: select *most of the time (e.g., with probability 1-* ε, with $0 < \varepsilon << 1$ *)* the action a^* with the highest value (current knowledge *exploited*)

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

  - at other times (e.g., with probability ε) select an *ordinary* (i.e., not significant or dominant) action! (action space *explored*)
  - *Question*: How to select among *ordinary* actions?

N-Armed Bandit Problem

- How do we select among ordinary actions?
  - *randomly* where, regardless of its value, each *non-greedy* action is *equally likely* to be selected.
  - Using *Softmax* action selection mechanism where each action (greedy or non-greedy) is given as high a chance for selection as its estimated value's proportion to the entire distribution.



An Example to Softmax Selection

– ***Gibbs or Boltzmann*** distribution, an example to *Softmax* methods: action a is selected on the t^{th} play with the following formula:

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

– with τ a positive parameter called the *temperature*.

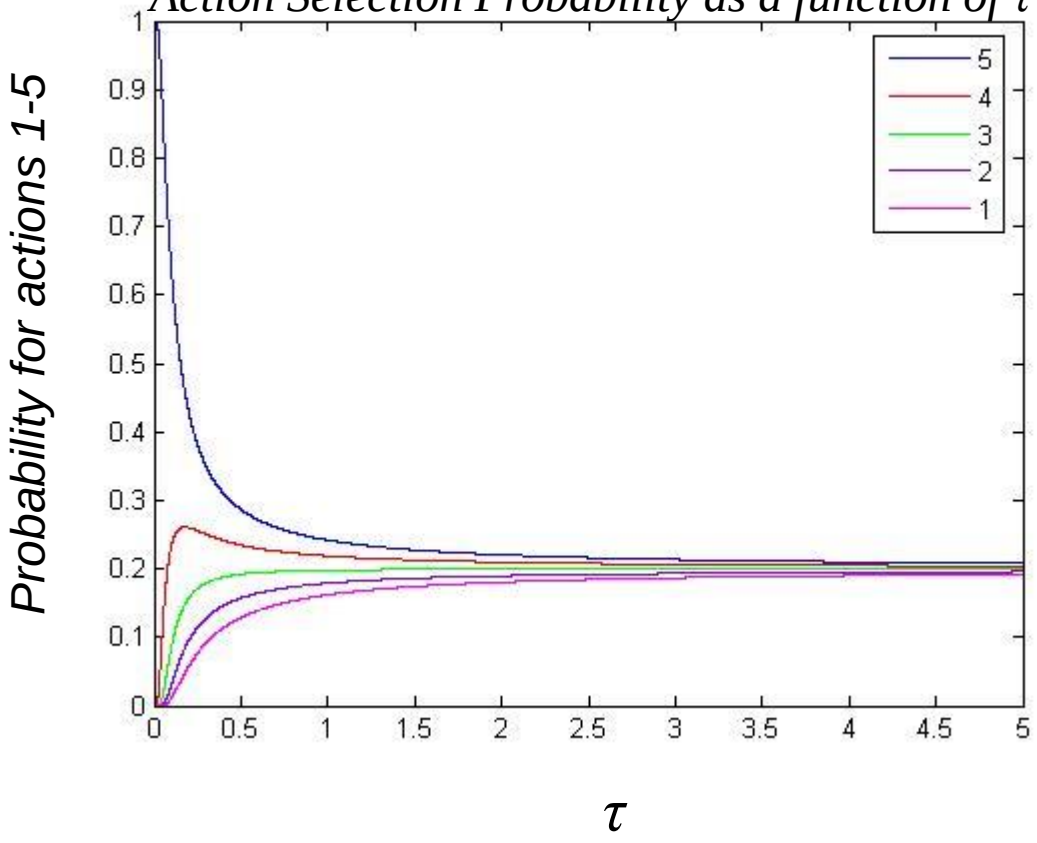
Example

Given are five actions with the following current values:

Action	Value
1	0
2	0.1
3	0.2
4	0.3
5	0.4

Temperature:
$0^+ < \tau \leq 5$

***Gibbs or Bolzmann Distribution** of Action Selection Probability as a function of τ*



greedy selection as $\tau \to 0$ ← *Softmax Selection by Gibbs* → *random selection as $\tau \to \infty$*

Evaluation vs Instruction

- Evaluation
  - Learning by selection (i.e., from experience)
  - Action-dependent
  - Performs search in action space
  - Constructs a mapping from situations (states) to action probabilities
  - Analyzes and controls environment
- Instruction
  - Learning by instruction
  - Action-independent
  - Performs search in weight space
  - Constructs a mapping from situations to correct actions to correctly generalize to new situations
  - Follows instructive information

Evaluation vs Instruction

- Comparison in terms of a simple, single-state example
- Assume you select action a_i among n possible actions $\{a_1,...,a_n\}$
  - Evaluative feedback will provide a measure of how favorable a_i is.
  - Instructive feedback will provide the correct action a_j.

Evaluation vs Instruction

- Assume a two-action $\{a_1, a_2\}$, two-reward $\{success, failure\}$ task at a single-state.

- For *deterministic rewards* (for noise-free instructive feedbacks), actions taken for which agent receives *success*, will mean the correct action. Hence, a supervised algorithm (i.e., one with instructive feedback) will solve the problem.

Evaluation vs Instruction

- For *stochastic rewards* (for noisy instructive feedbacks), we have four cases:

Cases	*1*	*2*	*3*	*4*
	Prob. of success	*Prob. of success*	*Prob. of success*	*Prob. of success*
a_1	< 0.5	< 0.5	> 0.5	> 0.5
a_2	< 0.5	> 0.5	< 0.5	> 0.5
problem	*hard*	*easy*	*easy*	*hard*

Evaluation vs Instruction

- Cases 2 and 3 are not hard to solve since the probability of success for the favorable action is higher than half the time, whereas the unfavorable action has a probability of success less than half.
- Cases 1 and 4, on the other hand, are not easy since the probability of success for both actions are either greater than or less than 0.5.
- In case 1, since both actions take mostly failures, the ***supervised algorithm*** (i.e., one receiving instructive feedback) will oscillate between the two actions, and not converge to the better action.
- In case 4, since both actions take mostly successes, the ***supervised algorithm*** will be stuck on any action, regardless of whether the action is better.
- The ***action-value*** (i.e., evaluative feedback) method solves the problem.

Incremental Implementation

- The following formula

$$Q_t(a) = \frac{\sum_{i=1}^{k_a} r_i}{k_a}$$

  requires k_a locations to keep rewards by time t.
- We may compute the action value $Q_t(a)$ not requiring this amount of memory by incrementally updating $Q_t(a)$.

Incremental Implementation

- To compute $Q_{k+1}(a)$ so as to include $k+1^{st}$ reward using $Q_k(a)$, we may follow the derivation below:

$$Q_{k+1} = \frac{1}{k+1}\sum_{i=1}^{k+1} r_i = \frac{1}{k+1}\left(r_{k+1} + \sum_{i=1}^{k} r_i\right)$$

$$Q_{k+1} = \frac{1}{k+1}\left(r_{k+1} + kQ_k + Q_k - Q_k\right) = \frac{1}{k+1}\left(r_{k+1} + (k+1)Q_k - Q_k\right)$$

$$Q_{k+1} = Q_k + \frac{1}{k+1}\left(r_{k+1} - Q_k\right)$$

Incremental Implementation

$$Q_{k+1} = Q_k + \frac{1}{k+1}\left(r_{k+1} - Q_k\right)$$

- where
  - Q_{k+1} is the new estimate
  - Q_k is the old estimate
  - $k+1$ is the step size
  - r_{k+1} is the target (current immediate reward)

Tracking a Nonstationary Problem

- In the above formula, the value update rule is based upon the assumption that the environment is *stationary*, that is, its behavior (i.e., its response to the actions taken by the agent) does not change with time.
- In a stationary environment, where the response of the environment mimics the same tendency throughout learning, an individual subsequent response r_{k+m} given to a action a_i at time $k+m$ should affect the accumulated response (i.e., value of the action $Q_{k+m}(a_i)$ less than a preceding action r_k since the action's value $Q_{k+m}(a_i)$ at time $k+m$ represents a longer sequence of responses than the value $Q_k(a_i)$ at time k does and, hence, is closer to convergence.
- To ensure this, the *step size* parameter is formulated to be inversely proportional to time on the previous page.

Tracking a Nonstationary Problem

- The behavior of a *non-stationary* environment may show changes.
- For the following two cases:
  1) Continuous changes in environment behavior may occur sufficiently slowly or
  2) the environment may behave stationary for a sufficiently long period,
- the agent may
  - track the environment behavior in 1;
  - establish experience characteristic to a specific period in 2 (considering each behavioral period of a non-stationary environment as a different stationary environment).

Tracking a Nonstationary Problem

- For the agent to track the nonstationary environment's continuously but sufficiently slowly changing behavior, the *latest response* of the environment should be given *as much weight as the preceding ones*, since its latest response reflects the newest viewpoint of the environment. Hence, the stepsize parameter should be kept constant as follows:

$$Q_{k+1} = Q_k + \alpha\left(r_{k+1} - Q_k\right); \quad 0 < \alpha \leq 1$$

Tracking a Nonstationary Problem

$$
\begin{aligned}
Q_k &= Q_{k-1} + \alpha\left(r_k - Q_{k-1}\right); \quad 0 < \alpha \le 1 \\
&= \alpha\, r_k + (1-\alpha) Q_{k-1} \\
&= \alpha\, r_k + (1-\alpha)(\alpha r_{k-1} + (1-\alpha) Q_{k-2}) \\
&= \alpha r_k + (1-\alpha)\alpha r_{k-1} + (1-\alpha)^2 Q_{k-2} \\
&= \alpha r_k + (1-\alpha)\alpha r_{k-1} + (1-\alpha)^2 \alpha r_{k-2} + (1-\alpha)^3 Q_{k-3} \\
&= \alpha r_k + (1-\alpha)\alpha r_{k-1} + (1-\alpha)^2 \alpha r_{k-2} + \cdots + (1-\alpha)^{k-1} \alpha r_1 + (1-\alpha)^k Q_0
\end{aligned}
$$

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

Tracking a Nonstationary Problem

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

- Given $0 < \alpha \leq 1$, the above formula implies that the older the responses r_i the higher the "*discounting*" effect of the coefficients.
- As a natural consequence of the above, the initial action value Q_0 gets progressively *discounted* in time.
- This action value is also called the *exponential, recency-weighted average*.

Conditions of Convergence

- Two important conditions in the stochastic approximation to assure *convergence*:

$$\sum_{i=1}^{\infty} \alpha_i(a) = \infty$$

$$\sum_{i=1}^{\infty} \alpha_i^2(a) < \infty$$

where

  - the first is to ensure *long enough steps* to rise above any initial conditions and random fluctuations, and
  - the second is to guarantee *eventually sufficiently small steps* for convergence assurance.

Optimistic Initial Values

- It is natural to initially assign *0* to all action values if there is no *a priori* knowledge on the environment and use *ε–greedy* RL method.
- Another option would be to *start with large (optimistic) initial action values and encourage the agent to explore*. It will pursue frequent exploration before actions taken attain a value higher than the optimistic initial action values.

Reinforcement Comparison

- Here the underlying idea is to provide the agent with the so-called *reference reward* so the agent can understand *through comparison* with the reference reward what it really means to receive a reward of 3 or 0.5. Reinforcement comparison methods do not keep action value estimates.
- *Preference*: probability $p_k(a)$ of action a at play k:

$$p_{k+1}(a) = p_k(a) + \beta\left(r_k - \overline{r_k}\right)$$

Reinforcement Comparison

- *Selection* done through *Softmax* relationship:

$$\pi_k(a) = \frac{e^{p_k(a)}}{\sum_{b=1}^{n} e^{p_k(b)}}$$

where the *reference reward is updated* at each new play as follows:

$$\overline{r_{k+1}} = \overline{r_k} + \alpha\left(r_k - \overline{r_k}\right)$$

Pursuit Methods

- *Pursuit methods* maintain both action value estimates and action preferences. They use action preferences (i.e., probability $\pi_k(a)$ of selecting action a at play k) to select the action and the value estimates $Q_k(a)$ to determine the greedy action a^*.
- After a^* is determined, $\pi_k(a^*)$ and the preference of remaining actions are updated as on the next page.
- $Q_k(a^*)$ is then updated using one of the methods (e.g., action value or Softmax) we discussed above.

Pursuit Methods: Algorithm

1. *Initialize action preference vector and action value estimates:*

$$\overline{\pi_0} = \begin{bmatrix} \pi_0(a_1) & \pi_0(a_2) & \pi_0(a_3) & \cdots & \pi_0(a_n) \end{bmatrix}^T$$

$$\overline{Q_0} = \begin{bmatrix} Q_0(a_1) & Q_0(a_2) & Q_0(a_3) & \cdots & Q_0(a_n) \end{bmatrix}^T$$

2. *Loop until a specific number of plays or* $|\pi_{k+1}(a^*) - \pi_k(a^*)| \leq \varepsilon$

   a. *Select action using the action preferences (i.e., generate RNs observing the preference vector's probability distribution)*

   b. *Receive environment's response*

   c. *Update action value estimate, say, by incremental implementation*

$$Q_{k+1}(a^*_{k+1}) = Q_k(a^*_{k1}) + \frac{1}{k+1}\left(r_{k+1} - Q_k(a^*_{k+1})\right)$$

   d. *Determine greedy action*

$$a^*_{k+1} = \arg\max_a \{Q_{k+1}(a)\}$$

   e. *Update action preferences*

$$\pi_{k+1}(a^*_{k+1}) = \pi_k(a^*_{k+1}) + \beta\left(1 - \pi_k(a^*_{k+1})\right)$$

$$\pi_{k+1}(a) = \pi_k(a) + \beta\left(0 - \pi_k(a)\right); \quad \forall a \neq a^*_{k+1}$$

Associative Search

- In the *non-associative* setting, we studied *n* actions at a single state in the n-armed bandit problem.
- *Associative search* incorporates the relation of actions to various situations (states) of the environment with the purpose to find the best action at any situation inhibiting any influence of the selection of action on situations.
- *Full RL framework*, studying the most general problem, involves possible state changes as a result of the selection of an action.

RL Loop: Non-associative Setting

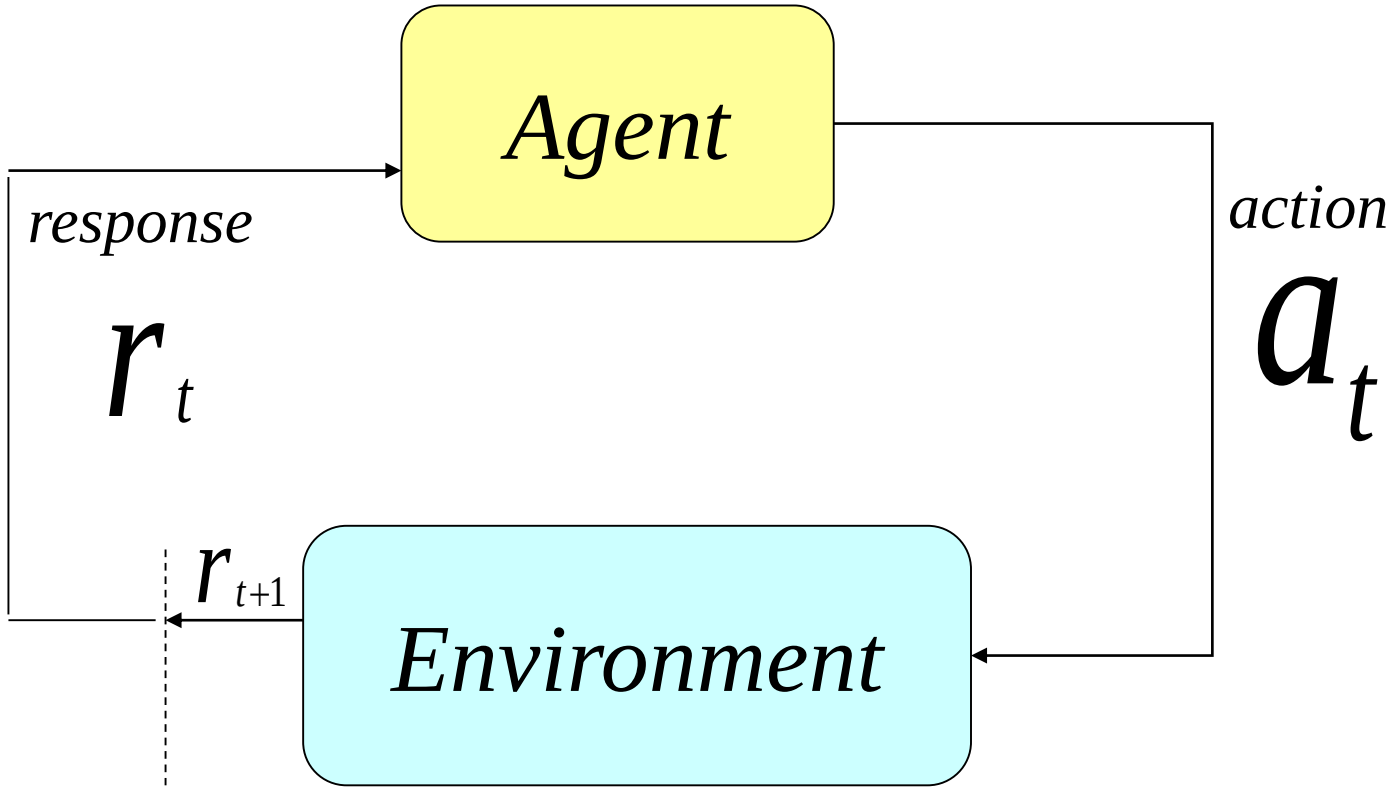


Figure reproduced from the figure on page 52 in reference [1]

RL Loop: Associative Search

State information s_{t+1} produced by environment is independent of action a_t selected.

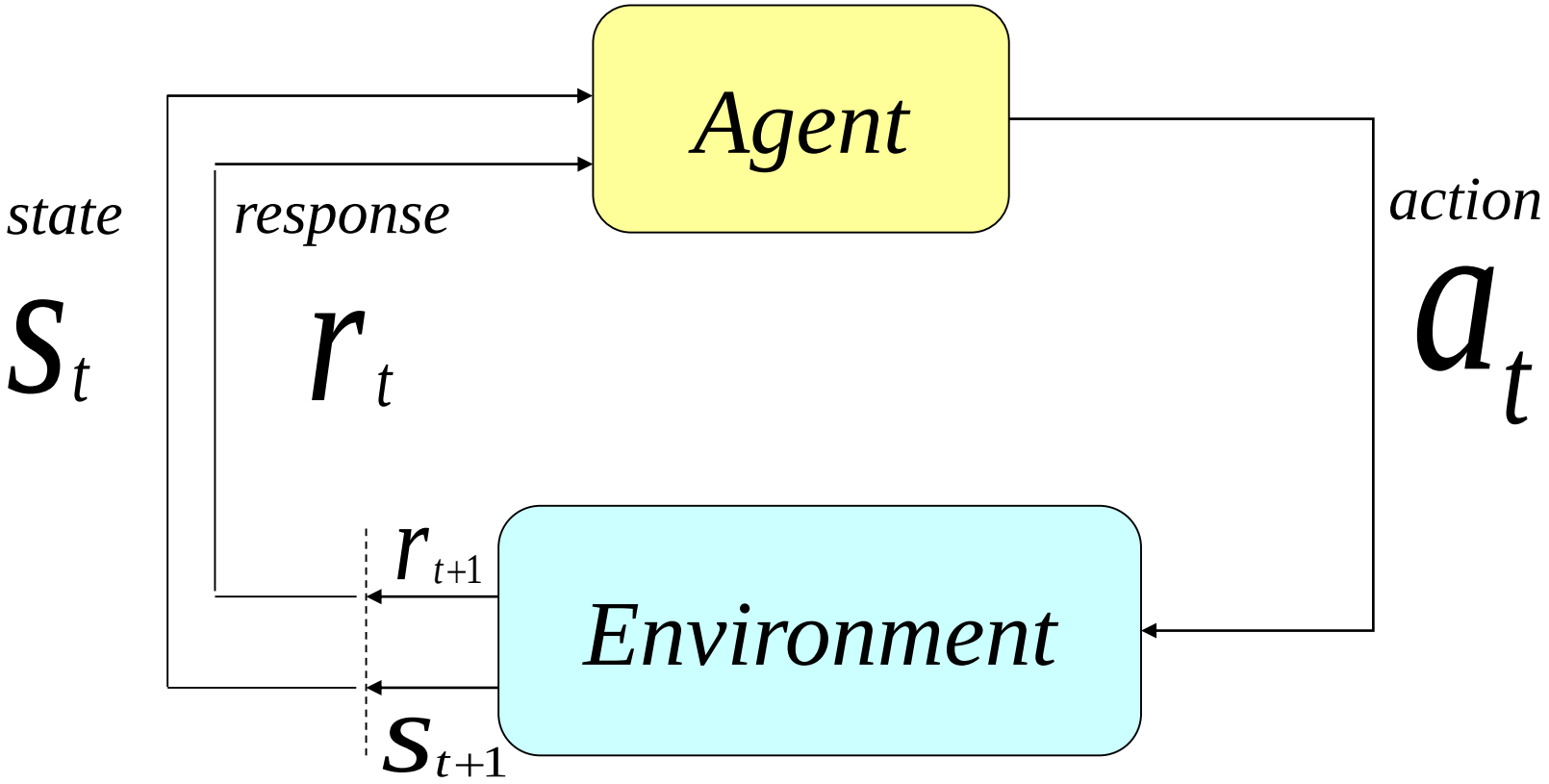


Figure reproduced from the figure on page 52 in reference [1]

RL Loop: Full RL Framework

State information s_{t+1} is affected by action a_t selected.

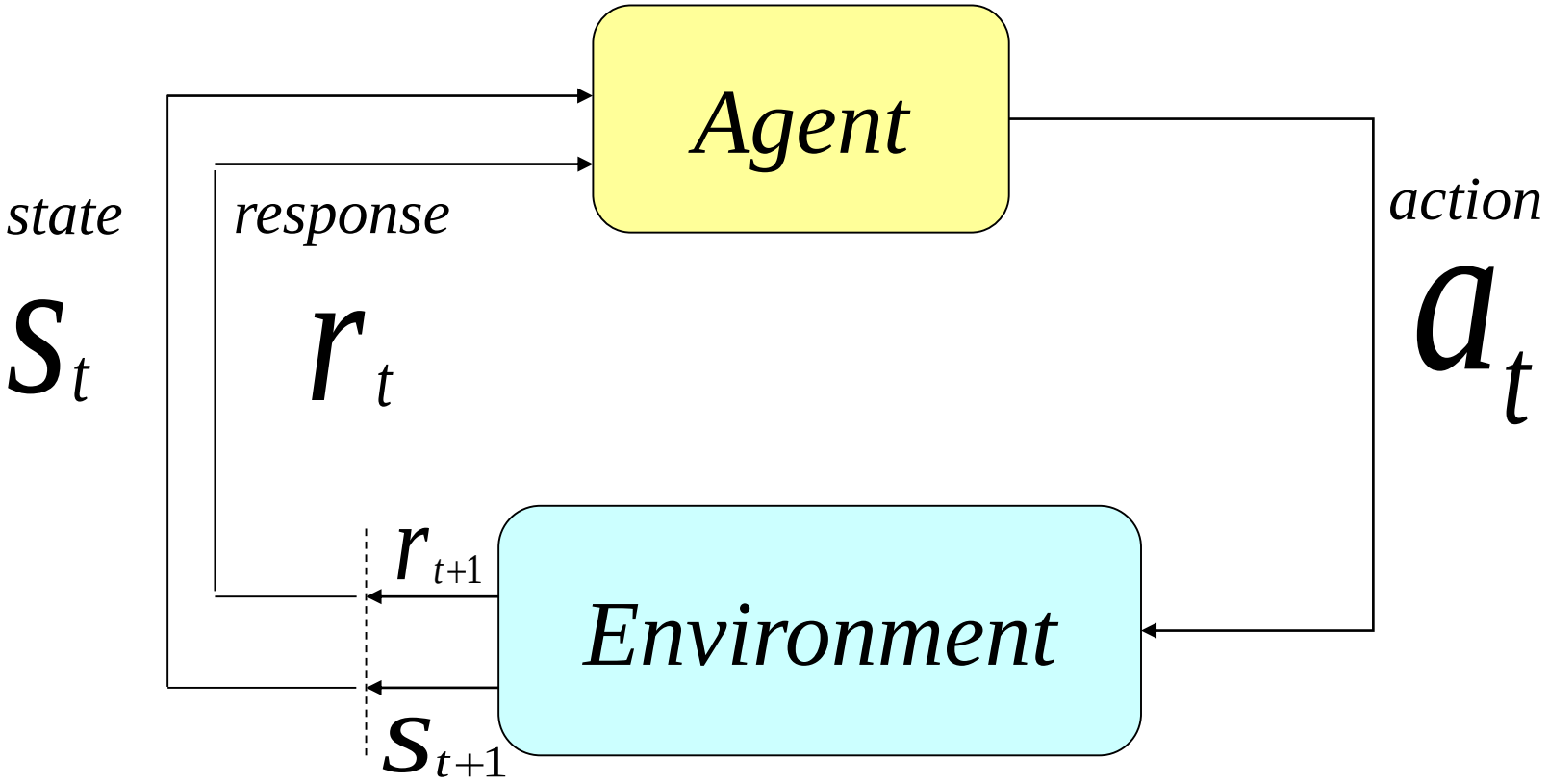


Figure reproduced from the figure on page 52 in reference [1]

References

- [1] Sutton, R. S. and Barto A. G., "*Reinforcement Learning: An introduction*," MIT Press, 1998