REINFORCEMENT LEARNING

- Reinforcement learning addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goals.
- This very generic problem covers tasks such as learning to control a mobile robot, learning to optimize operations in factories, and learning to play board games.
- Each time the agent performs an action in its environment, a trainer may provide a reward or penalty to indicate the desirability of the resulting state
- For example, when training an agent to play a game the trainer might provide a positive reward when the game is won, negative reward when it is lost, and zero reward in all other states.
- The task of the agent is to learn from this indirect, delayed reward, to choose sequences of actions that produce the greatest cumulative reward.
- Reinforcement learning algorithms are related to dynamic programming algorithms frequently used to solve optimization problems.

- The robot, or ***agent, has a set of sensors to*** observe the ***state of its environment, and a set of actions it can perform to alter*** this state.
- A mobile robot may have sensors such as a camera and sonars, and actions such as "move forward" and "turn.“
- Its task is to learn a control strategy, or policy, for choosing actions that achieve its goals.
- The robot may have a goal of docking onto its battery charger whenever its battery level is low
- The goals of the agent can be defined by a reward function that assigns a numerical value-an immediate payoff-to each distinct action the agent may take from each distinct state.
- the goal of docking to the battery charger can be captured by assigning a positive reward (e.g., +loo) to state-action transitions that immediately result in a connection to the charger and a reward of zero to every other state-action transition. This reward function may be built into the robot, or known only to an external teacher who provides the reward value for each action performed by the robot. The task of the robot is to perform sequences of actions, observe their consequences, and learn a control policy.

RL

- manufacturing optimization problems in which a sequence of manufacturing actions must be chosen, and the reward to be maximized is the value of the goods produced minus the costs involved.
- sequential scheduling problems such as choosing which taxis to send for passengers in a large city, where the reward to be maximized is a function of the wait time of the passengers and the total fuel costs of the taxi fleet.
- any type of agent that must learn to choose actions that alter the state of its environment and where a cumulative reward function is used to define the quality of any given action sequence

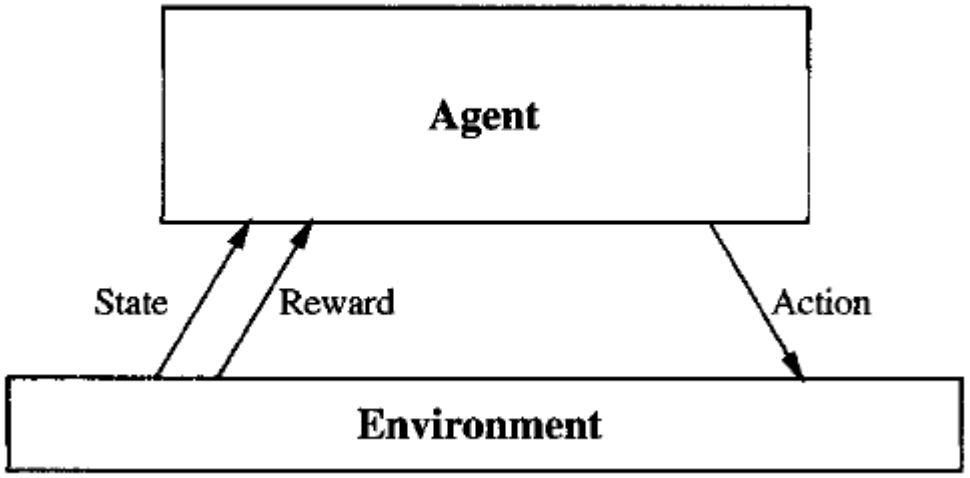
RL

- Include settings in which the actions have deterministic or nondeterministic outcomes, and settings in which the agent has or does not have prior knowledge about the effects of its actions on the environment.
- The agent exists in an environment described by some set of possible states *S. It can* perform any of a set of possible actions A. Each time it performs an action *ai, in* some state *st the agent receives a real-valued* reward *ri, that indicates the immediate value* of this state-action transition. This produces a sequence of states *si, actions ai, and*immediate rewards *ri*
- The agent's task is to learn a control policy, π *: S -> A, that maximizes the expected* sum of these rewards,
- Consider problems where the actions may have nondeterministic outcomes and where the learner lacks a domain theory that describes the outcomes of its actions.

- The control policy we desire is one that, from any initial state, chooses actions that maximize the reward accumulated over time by the agent

$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \ldots$$

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \text{, where } 0 \leq \gamma < 1$$



- *[Tesauro, 1995] Learn to play Backgammon*

Immediate reward

- *+100 if win*
- *-100 if lose*
- *0 for all other states*
- *Trained by playing 1.5 million games against*

itself Now approximately equal to best human

player

RL

- ***Delayed reward***
- ***Exploration***
- ***Partially observable states***
- ***Life-long learning***

- ***Delayed reward.*** *The task of the agent is to learn a target function n that* maps from the current state *s to the optimal action a = π(s).*
- In other learning some target function such as *π, each training example would be a pair of the form (s, n(s))* In reinforcement learning, however, training information is not available in this form. Instead, the trainer provides only a sequence of immediate reward values as the agent executes its sequence of actions. The agent, therefore, faces the problem of *temporal credit assignment: determining which of the actions* in its sequence are to be credited with producing the eventual rewards.

Exploration

In reinforcement learning, the agent influences the distribution of training examples by the action sequence it chooses. This raises the question of which experimentation strategy produces most effective learning.

The learner faces a tradeoff in choosing whether to favor *exploration of unknown* states and actions (to gather new information), or *exploitation of states and* actions that it has already learned will yield high reward (to maximize its cumulative reward).

Partially observable states.

- *Although it is convenient to assume that the* agent's sensors can perceive the entire state of the environment at each time step, in many practical situations sensors provide only partial information.
- For example, a robot with a forward-pointing camera cannot see what is behind it. In such cases, it may be necessary for the agent to consider its previous observations together with its current sensor data when choosing actions, and the best policy may be one that chooses actions specifically to improve the observability of the environment.

Life-long learning

- *Unlike isolated function approximation tasks, robot learning* often requires that the robot learn several related tasks within the same environment, using the same sensors. For example, a mobile robot may need to learn how to dock on its battery charger, how to navigate through narrow corridors, and how to pick up output from laser printers. This setting raises the possibility of using previously obtained experience or knowledge to reduce sample complexity when learning new tasks

THE LEARNING TASK

- formulate the problem of learning sequential control strategies
- Assume the agent's actions are deterministic or that they are nondeterministic.
- assume that the agent can predict the next state that will result from each action, or that it cannot
- assume that the agent is trained by an expert who shows it examples of optimal action sequences, or that it must train itself by performing actions of its own choice

Markov decision processes

- In a Markov decision process (MDP) the agent can perceive a set **S of distinct** states of its environment and has a set **A of actions that it can perform.**
- At each discrete time step t, the agent senses the current state **st,** chooses a current action **at** and performs it.
- The environment responds by giving the agent a reward **rt =r (st, at) and** by producing the succeeding state **st+l = δ(st, at)**
- the functions **δ and r** are part of the environment and are not necessarily known to the agent.
- In an MDP, the functions ***6(st, a,) and r(s,, a,) depend only on the current state*** and action, and not on earlier states or actions

MDP

- The task of the agent is to learn a ***policy, π : S -> A, for selecting its next*** action ***at, based on the current observed state st; that is, π(st) = at.***
- policy that produces the greatest possible cumulative reward for the robot over time.
- Define the cumulative value $V^{\pi}(st)$ achieved by following an arbitrary policy π from an
- arbitrary initial state ***st*** as follows

$$V^{\pi}(s_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$$

$$\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

where the sequence of rewards rt+i is generated by beginning at state s, and by repeatedly using the policy π to select actions.
$0<\gamma \leq 1$ *is a constant that determines the relative*
value of delayed versus immediate rewards

- In particular, rewards received i time steps into the future are discounted exponentially by a factor of γ^t . Note if we set $\gamma = 0$, only the immediate reward is considered. As we set γ closer to 1, future rewards are given greater emphasis relative to the immediate reward.
- The quantity $V^{\pi}(s)$ defined by Equation (13.1) is often called the discounted cumulative reward achieved by policy π from initial state s.
- It is reasonable to discount future rewards relative to immediate rewards because, in many cases, we prefer to obtain the reward sooner rather than later.

- finite horizon reward considers the undiscounted sum of rewards over a finite number h of steps.

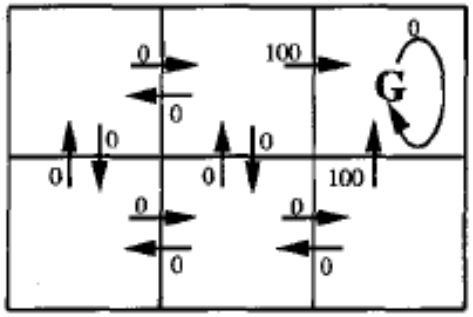
$$\sum_{i=0}^{h} r_{t+i}$$

- average reward which considersthe average reward per time step over the entire lifetime of the agent Lim $_{h->\infty}$ $\sum_{i=0}^{h} r_{t+i}$

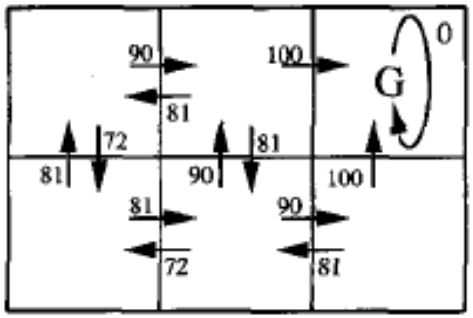
- Agent learn a policy π that maximizes $V^{\pi}(s)$ for all states s called optimal policy

$$\pi^* \equiv \underset{\pi}{\operatorname{argmax}}\, V^{\pi}(s),\ (\forall s)$$

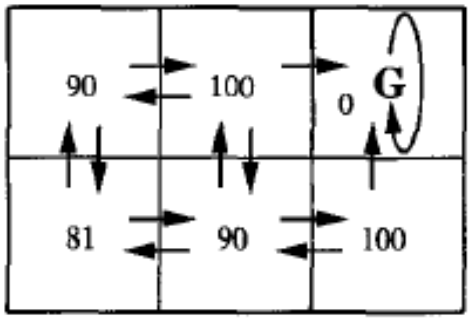
- refer to the value function $v^{\pi}*(s)$ of such an optimal policy as V*(s), which gives the maximum discounted cumulative reward that the agent can obtain starting from state s; that is, the discounted cumulative reward obtained by following the optimal policy beginning at state s.



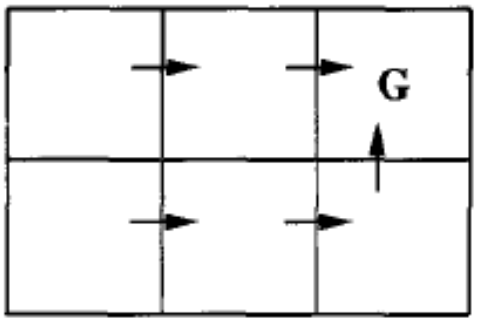
$r(s, a)$ (immediate reward) values



$Q(s, a)$ values



$V^*(s)$ values



One optimal policy

A simple deterministic world to illustrate the basic concepts of Q-learning. Each grid square represents a distinct state, each arrow a distinct action. The immediate reward function, ***r(s, a) gives reward 100*** for actions entering the goal state G, and zero otherwise. Values of ***V*(s) and Q(s, a) follow from r(s, a), and the discount factor y = 0.9. An optimal policy, corresponding to actions with maximal*** Q values, is also shown.

- The six grid squares in this diagram represent six possible states, or locations, for the agent. Each arrow in the diagram represents a possible action the agent can take to move from one state to another. The number associated with each arrow represents the immediate reward r(s, a) the agent receives if it executes the corresponding state-action transition. The immediate reward in this particular environment is defined to be zero for all state-action transitions except for those leading into the state labeled G. It is convenient to think of the state G as the goal state, because the only way the agent can receive reward, in this case, is by entering this state. in this particular environment, the only action available to the agent once it enters the state G is to remain in this state. For this reason, G is called an absorbing state.

- Once the states, actions, and immediate rewards are defined, and once we choose a value for the discount factor γ, we can determine the optimal policy π* and its value function V*(s).
- In this case, let us choose γ = *0.9. The diagram* at the bottom of the figure shows one optimal policy for this setting (there are others as well). Like any policy, this policy specifies exactly one action that the agent will select in any given state. Not surprisingly, the optimal policy directs the agent along the shortest path toward the state G.

- The diagram at the right of Figure 13.2 shows the values of V* for each state. For example, consider the bottom right state in this diagram. The value of V* for this state is 100 because the optimal policy in this state selects the "move up" action that receives immediate reward 100. Thereafter, the agent will remain in the absorbing state and receive no further rewards. Similarly, the value of V* for the bottom center state is 90. This is because the optimal policy will move the agent from this state to the right (generating an immediate reward of zero), then upward (generating an immediate reward of 100). Thus, the discounted future reward from the bottom center state is

$$0+y100+y^2 0+Y^3 0+...=90$$

- *V* is defined to be the sum of discounted future rewards over the* infinite future. In this particular environment, once the agent reaches the absorbing state ***G its infinite future will consist of remaining in this state and receiving*** rewards of zero.

Q LEARNING

- It is difficult to learn the function $\pi^* : S \rightarrow A$ ***directly, because the available training*** data does not provide training examples of the form *(s, a).*
- the only training information available to the learner is the sequence of immediate rewards *r(si, ai) for i = 0, 1,2, ...*
- given this kind of training information it is easier to learn a numerical evaluation function defined over states and actions, then implement the optimal policy in terms of this evaluation function.

Q LEARNING

- The agent should attempt to learn evaluation function *V*.*
- The agent should prefer state *sl over state s2 whenever V*(sl >nV*(s2), because the cumulative future reward will be greater from s1.*
- the agent's policy must choose among actions, not among states, However, it can use *V* in certain settings to choose among actions as well.*
- The optimal action in state *s is the action a that maximizes the sum of the immediate reward r(s, a)* plus the value *V* of the immediate successor state, discounted by y.*

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}[r(s, a) + \gamma V^*(\delta(s, a))] \qquad (13.3)$$

δ(s, a) denotes the state resulting from applying action a to state s

Q LEARNING

- the agent can acquire the optimal policy by learning *V*, provided it has* perfect knowledge of the immediate reward function ***r and the state transition*** function *δ. When the agent knows the functions r and δ used by the environment* to respond to its actions, it can then use Equation (13.3) to calculate the optimal action for any state *s*
- Unfortunately, learning *V* is a useful way to learn the optimal policy only* when the agent has perfect knowledge of *δ and r. This requires that it be able to* perfectly predict the immediate result (i.e., the immediate reward and immediate successor) for every possible state-action transition. This assumption is comparable to the assumption of a perfect domain theory in explanation-based learning,

In many practical problems, such as robot control, it is impossible for the agent or its human programmer to predict in advance the exact outcome of applying an arbitrary action to an arbitrary state. Imagine, for example, the difficulty in describing *δ for a robot arm shoveling dirt when the* resulting state includes the positions of the dirt particles. In cases where either *δ or r is unknown, learning V* is unfortunately of no use for selecting optimal* actions because the agent cannot evaluate Equation (13.3).

The Q Function

- Let us define the evaluation function *Q(s, a) so that its value is the maximum discounted* cumulative reward that can be achieved starting from state *s and applying* action *a as the first action*
- the value of *Q is the reward received* immediately upon executing action *a from state s, plus the value (discounted by* γ) of following the optimal policy thereafter.

$$Q(s, a) \equiv r(s, a) + \gamma V^*(\delta(s, a))$$

- *Q(s, a) is exactly the quantity that is maximized in Equation (13.3) in order to choose the optimal action a in state s.*
- Rewrite Equation (13.3) in terms of *Q(s, a) as*

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

- it shows that if the agent learns the *Q* function instead of the *V* function, it will be able to select optimal actions even when it has no knowledge of the functions r and δ.*
- it need only consider each available action *a in its current state s and choose the* action that maximizes *Q(s, a)*
- It may at first seem surprising that one can choose globally optimal action sequences by reacting repeatedly to the local values of *Q for the current state.*

- The agent can choose the optimal action without ever conducting a lookahead search to explicitly consider what state results from the action. Part of the beauty of *Q learning is that the evaluation function is defined to have precisely* this property-the value of *Q for the current state and action summarizes in a* single number all the information needed to determine the discounted cumulative reward that will be gained in the future if action *a is selected in state s.*

- *Q value for each state-action transition equals* the *r value for this transition plus the V* value for the resulting state discounted by* γ. Note also that the optimal policy shown in the figure corresponds to selecting actions with maximal *Q values.*

Algorithm for Learning Q

- Learning the *Q function corresponds to learning the optimal policy*
- The key problem is finding a reliable way to estimate training values for *Q, given only a sequence of immediate rewards r spread out over time. This can* be accomplished through iterative approximation.

$$V^*(s) = \max_{a'} Q(s, a')$$

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a')$$

- This recursive definition of Q provides the basis for algorithms that iteratively approximate Q (Watkins 1989).
- Use the symbol $\hat{Q}$ ɔ refer to the learner's estimate, or hypothesis, of the actual ***Q*** function
- In this algorithm the learner represents its hypothesis $\hat{Q}$

  by a large table with a separate entry for each state-action pair. The table entry for the pair (s, a) stores the value for $\hat{Q}$ (s,a)-the, learner's current hypothesis about the actual but unknown value Q(s, a).
- The table can be initially filled with random values (though it is easier to understand the algorithm if one assumes initial values of zero). The agent repeatedly observes its current state s, chooses some action a, executes this action, then observes the resulting reward r = r(s, a) and the new state s' = δ(s, a).

- It then updates the table entry for $\hat{Q}$ (s,***a) following each such*** transition, according to the rule:

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$$

- this training rule uses the agent's current $\hat{Q}$ ***values for the new state*** s' to refine its estimate of $\hat{Q}$ (s,a), for the previous state s.
- This training rule is motivated by Equation (13.6), although the training rule concerns the agent's approximation whereas Equation (13.6) applies to the actual Q function.
- although Equation (13.6) describes Q in terms of the $\hat{Q}$ functions 6(s, a) and r(s, a),the agent does not need to know these general functions to apply the training rule of Equation (13.7). Instead it executes the action in its environment and then observes the resulting new state s' and reward r. Thus, it can be viewed as sampling these functions at the current values of s and a.

Q learning algorithm, assuming deterministic rewards and actions. The discount factor γ may be any constant such that $0 \leq \gamma < 1$

Q learning algorithm

For each s, a initialize the table entry $\hat{Q}(s, a)$ to zero.
Observe the current state s
Do forever:

- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for $\hat{Q}(s, a)$ as follows:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

- $s \leftarrow s'$

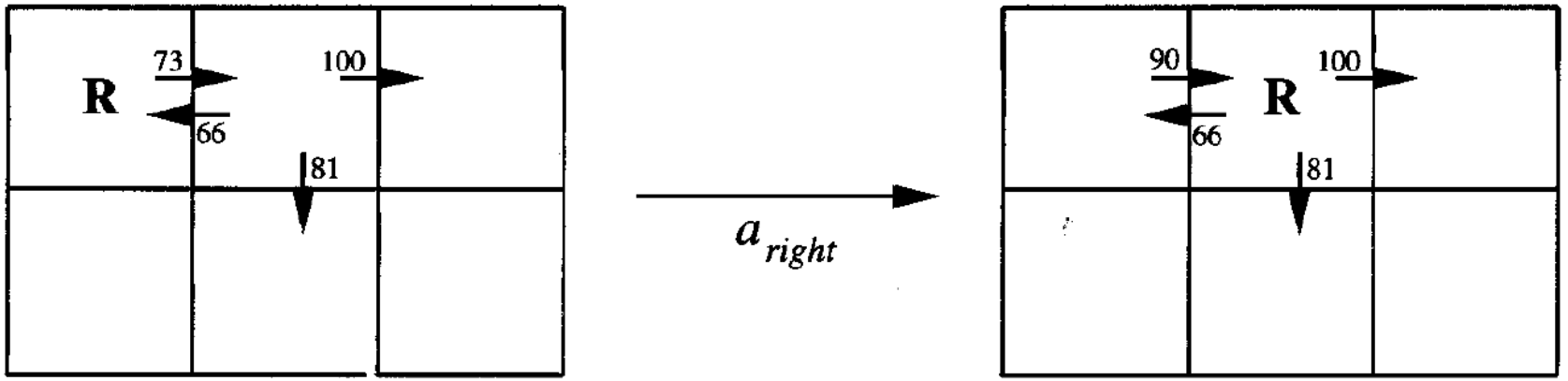
- Using this algorithm the agent's estimate $\hat{Q}$ converges in the limit to the actual Q function, provided the system can be modeled as a deterministic Markov decision process, the reward function r is bounded, and actions are chosen so that every state-action pair is visited infinitely often

An Illustrative Example

- To illustrate the operation of the ***Q learning algorithm, consider a single action*** taken by an agent, and the corresponding refinement to $\hat{Q}$
- In this example, the agent moves one cell to the right in its grid world and receives an immediate reward of zero for this transition.It then applies the training rule of Equation (13.7) to refine its estimate $\hat{Q}$ for the state-action transition it just executed. According to the training rule, the new $\hat{Q}$ estimate for this transition is the sum of the received reward (zero) and the highest $\hat{Q}$ value associated with the resulting state (loo), discounted by *y (.9).*

- Each time the agent moves forward from an old state to a new one, Q learning propagates $\hat{Q}$ estimates ***backward from the new state to the old. At the*** same time, the immediate reward received by the agent for the transition is used to augment these propagated values of $\hat{Q}$

- Consider applying this algorithm to the grid world and reward function shown in Figure 13.2, for which the reward is zero everywhere, except when entering the goal state. Since this world contains an absorbing goal state, we will assume that training consists of a series of ***episodes. During each episode, the*** agent begins at some randomly chosen state and is allowed to execute actions until it reaches the absorbing goal state. When it does, the episode ends **and** the agent is transported to a new, randomly chosen, initial state for the next episode



Initial state: s_1 Next state: s_2

$$\hat{Q}(s_1, a_{right}) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a')$$

$$\leftarrow 0 + 0.9 \max\{66, 81, 100\}$$

$$\leftarrow 90$$

With all the $\hat{Q}$ values initialized to zero, the agent will make no changes to any $\hat{Q}$ table entry until it happens to reach the goal state and receive a nonzero reward. This will result in refining the $\hat{Q}$ value for the single transition leading into the goal state. On the next episode, if the agent passes through this state adjacent to the goal state, its nonzero $\hat{Q}$ value will allow refining the value for some transition two steps from the goal, and so on. Given a sufficient number of training episodes, the information will propagate from the transitions with nonzero reward back through the entire state-action space available to the agent, resulting eventually in a $\hat{Q}$ table containing the $\hat{Q}$ values shown in Figure 13.2 under certain assumptions the Q learning algorithm of Table 13.1 will converge to the correct Q function.

- First consider two general properties of this Q learning algorithm that hold for any deterministic MDP in which the rewards are non-negative, assuming we initialize all values to zero. The first property is that under these conditions the $\hat{Q}$ values never decrease during training. More formally, let $\hat{Q}_n(s, a)$ denote the learned $\hat{Q}(s,a)$ value after the nth iteration of the training procedure (i.e., after the nth state-action transition taken by the agent). Then

$$(\forall s, a, n) \quad \hat{Q}_{n+1}(s, a) \geq \hat{Q}_n(s, a)$$

A second general property that holds under these same conditions is that throughout the training process every $\hat{Q}$ value will remain in the interval between zero and its true Q value.

$$(\forall s, a, n) \quad 0 \leq \hat{Q}_n(s, a) \leq Q(s, a)$$