e-PGPathshala

Subject: Computer Science

Paper: Machine Learning

Module: Q Learning

Module No: CS/ML/39

Quadrant 1- e-text

Welcome to the e-PG Pathshala Lecture Series on Machine Learning. In this module we will discuss in detail about Q Learning.

Learning Objectives

The learning objectives of the module are as follows:

- To discuss the Model based and Model free reward learning.
- To discuss the Deterministic and Non-deterministic rewards and actions
- To explain Q Learning a model free reinforcement learning method.

39.1 Model based and Model free Learning

There are four aspects of reinforcement learning: 1) Policy 2) Reward function 3) Value function and 4) Model of the environment. A policyπ characterizes how the agent behaves and is a mapping from states to actions, with the probability of selecting the action. The agent learns this policy $\pi : S \to \mathcal{A}$ and usesthat to maximize the cumulative reward $V^{\pi}(s_t)$in the long run. As the reward has to be earned sooner than later, $V^{\pi}(s_t)$is called discounted cumulative reward.There are two types of cumulative reward:Finite horizon, where reward is calculated for some finite number of steps and Infinite horizon cumulative reward where the steps are infinite.

Finite Horizon: $V^{\pi}(s_t) = E[r_{t+1} + r_{t+2} + \cdots + r_{t+T}] = E\left[\sum_{i=1}^{T} r_{t+i}\right]$

Infinite Horizon: $V^{\pi}(s_t) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots] = E\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}\right]$

$0 \le \gamma < 1$ is the discount rate

39.1.1 Value Function and Optimal Value Functions

The state value functionV^{π}(s) is the expected return when starting in *s* and following the policy π. Q^{π}(s,a) is the state-action value function which is the expected return when starting in *s*, performing *a,* and following π. This state-action value is useful for finding the optimal policy and can be estimated from experience. We need to pick

the best action using $Q^{\pi}(s,a)$. The general Bellman equation can now be restated as:

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[r^{a}_{ss'} + \gamma V^{\pi}(s') \right] = \sum_{a} \pi(s,a) Q^{\pi}(s,a)$$

Now there is a set of *optimal* policies and V^{π} defines partial ordering on policies which share the same optimal value function

$$V^{*}(s) = \max_{\pi} V^{\pi}(s)$$

The general Bellman optimality equation

$$V^{*}(s) = \max_{a} \sum_{s'} P^{a}_{ss'} \left[r^{a}_{ss'} + \gamma V^{*}(s') \right]$$

Is a system of n non-linear equations that can be solved for V*(s) and hence the optimal policy can be extracted. Having the state-action function Q*(s,a) makes it possible to choose the policy:

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

The agent has to learn the optimal policy π^{*} that will give optimal cumulative reward

$$V^{*}(s_t) = \max_{a_t} Q^{*}(s_t, a_t) \quad \text{Value of } a_t \text{ in } s_t$$

$$Q^{*}(s_t, a_t) = E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) \max_{a_{t+1}} Q^{*}(s_{t+1}, a_{t+1})$$

In the model based learning the agent has the knowledge of the environment model parameters, P ($s_{t+1} \mid s_t, a_t$), p ($r_{t+1} \mid s_t, a_t$), uses it to calculate the rewards. As the parameters are known there is no need for exploration and the rewards and policy are calculated using dynamic programming. Theoptimum policy $\pi^{*}(s_t) = \arg\max_{a_t} \left(E[r_{t+1} \mid s_t, a_t] + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) V^{*}(s_{t+1}) \right)$ is estimated by solving the equation $V^{*}(s_t) = \max_{a_t} \left(E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) V^{*}(s_{t+1}) \right)$.

The value iteration method given below computes the optimal value V(s) starting from an arbitrary V(s) value by computing the immediate reward and following the

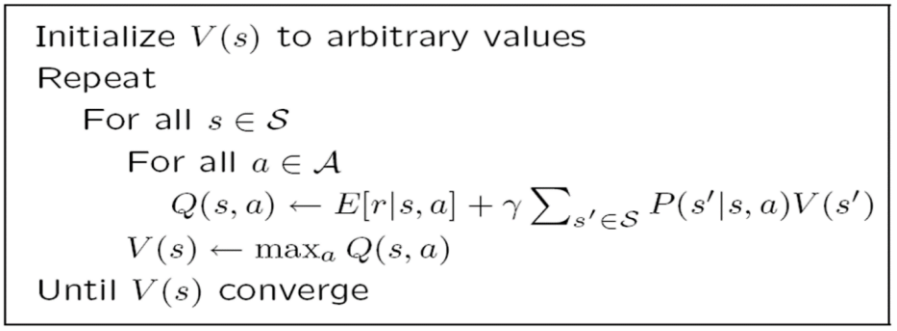


Figure 39.1 Value Iteration Method

Optimum policy thereafter (Figure 39.1).

The policy iteration method given below computes the optimal policy π' starting with an arbitrary policy π through policy evaluation i.e. finding the cumulative value $V^{\pi}(S)$and policy improvementi.e. modifying the policy, until convergence.

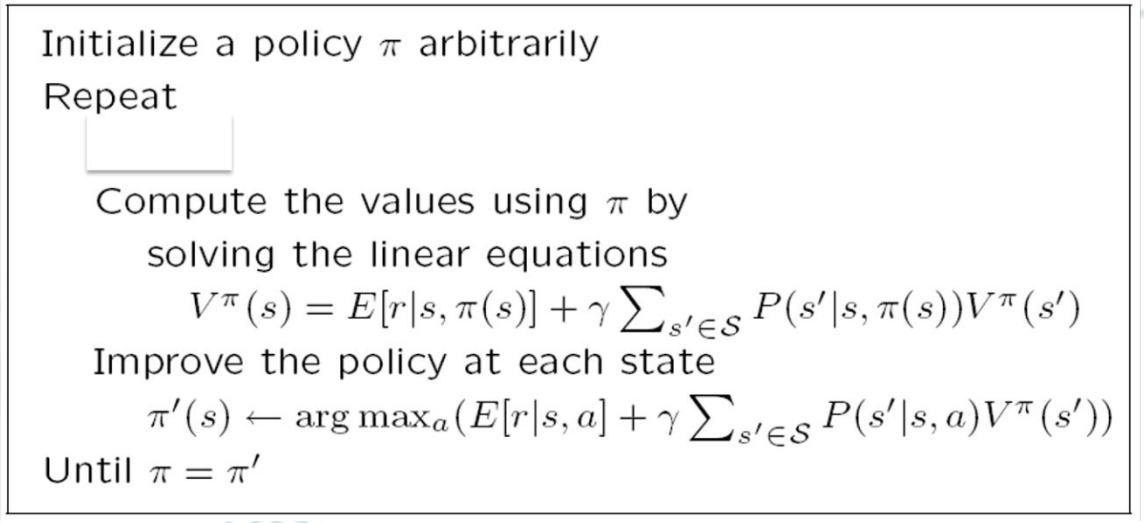


Figure 39.2 Policy Iteration Method

39.1.2 Deterministic Rewards and Actions:

In this case there is a single possible reward and next state for a state-action pair. The cumulative reward is given by $Q(s_t, a_t) = r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$and is used to update the value of $Q(s_t, a_t)$. Using any of the strategy explained above we move from state S_t to state S_{t+1} taking action a_t which yields the reward r_{t+1} and this is used to update the previous action as $\hat{Q}(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1})$. The q value increases as we try to find paths that gives higher cumulative reward and never decrease until optimal value is reached. For example in Figure 39.3 shown below consider the value of action marked by '*':

If path A is seen first,
Q(*)=0.9*max(0,81)=73 and then path B is seen,
Q(*)=0.9*max(100,81)=90Or,

If path B is seen first, Q(*)=0.9*max(100,0)=90
and then path A is seen, Q(*)=0.9*max(100,81)=90
We can infer that Q values increase but never decrease in both the cases shown above.

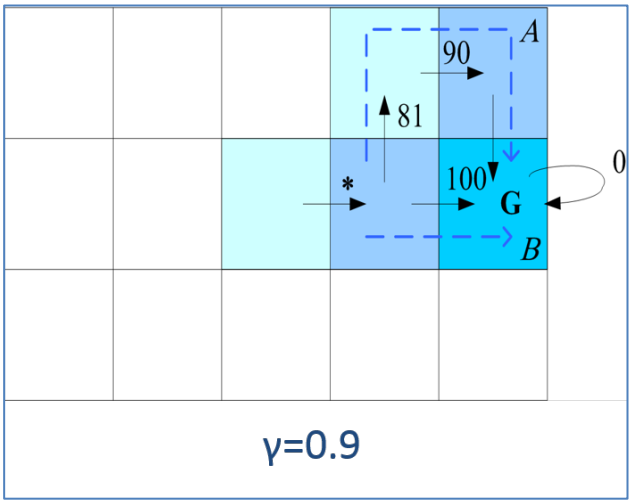


Figure 39.3 Calculation of Q-value

39.1.3 Nondeterministic Rewards and Actions:

When next states and rewards are nondeterministic (there is an opponent or randomness in the environment), then there is a probability distribution for next state $P(s_{t+1} \mid s_t, a_t)$ and reward $p(r_{t+1} \mid s_t, a_t)$. As there are different rewards for same action from a state we keep averages (expected values) instead of assignments. This is known as Q learning and predicted cumulative reward is given as $\hat{Q}(s_t,a_t) \leftarrow \hat{Q}(s_t,a_t) + \eta\left(r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1},a_{t+1}) - \hat{Q}(s_t,a_t)\right)$. The value of $r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1},a_{t+1})$ is the sample of the instances for each state - action pair. In temporal difference learning method Off-policy method is used i.e. policy is not used and the value of the next best action is used to estimate the temporal difference. In Q-learning based SARSA an On-policy method uses policy to determine temporal difference rather than the next best action. The state value $V(s_t)$ can be learned using the temporal difference learning and can be updated using the rule $V(s_t) \leftarrow V(s_t) + \eta(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$. The $r_{t+1} + \gamma V(s_{t+1})$ is the predicted value and $V(s_t)$ is the current estimated value and this temporal difference should be reduced to get the optimal value.

39.2 Basics of Temporal Learning

Temporal difference (TD) learning is a prediction-based machine learning method. It has primarily been used for the reinforcement learning problem, and is said to be a combination of Monte Carlo ideas because it learns by sampling the environment according to some *policy*, and dynamic programming (DP) ideasas it approximates its current estimate based on previously learned estimates. Temporal Difference (TD) learning methods can be used to estimate value functions. If the value functions were to be calculated without estimation, the agent would need to wait until the final reward was received before any state-action pair values can be updated. Once the final reward was received, the path taken to reach the final state would need to be

traced back and each value updated accordingly.Thiscan be expressed formally as:

$$V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)]$$

where s_t is the state visited at time t, R_t is the reward after time t and α is a constant parameter. On the other hand, with TD methods, an estimate of the final reward is calculated at each state and the state-action value updated for every step of the way:

$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

where r_{t+1} is the observed reward at time **t+1**

The TD method is called a "bootstrapping" method, because the value is updated partly using an existing estimate and not a final reward. The Environment, $P(s_{t+1} \mid s_t, a_t)$, $p(r_{t+1} \mid s_t, a_t)$, is not known; model-free learning. There is need for exploration to sample from

$P(s_{t+1} \mid s_t, a_t)$ and $p(r_{t+1} \mid s_t, a_t)$

We use the reward received in the next time step to update the value of current state (action). The temporal difference between the value of the current action and the value discounted from the next state.

39.3 Exploration Strategies

At each state (except a terminal state) the agent must select an action. There are several ways in which to decide which action to take. The simplest of these is greedy selection: the agent always selects the action that the highest state-action value. This method is pure exploitation. There are many more sophisticated strategies used which combine exploitation and exploration. Some of these methods are:

ε-greedy: In this method most of the time the action with the highest estimated reward is chosen, called the greediest action. Every once in a while, say with a small probability **ε**, an action is selected at random. The action is selected uniformly, independent of the action-value estimates.

Boltzman method: Here we move smoothly from exploration to exploitation.

$$P(a \mid s) = \frac{\exp[Q(s,a)/T]}{\sum_{b=1}^{\mathcal{A}} \exp[Q(s,b)/T]}$$

Here we need to choose action a in states with probability P Here we have the concept of Simulated annealing where T is a “temperature” .Simulated annealing is a probabilistic technique for approximating the global optimum of a given function. Simulated annealing is interpreted as slow cooling where there is a slow decrease in the probability of accepting worse solutions as it explores the solution space. Large T means that each action has approximately the same probability. Small T leads to more greedy behavior. Typically we start with a large T and decrease its value with time.

39.4 Q Learning

Q-Learning is an Off-Policy algorithm for Temporal Difference learning. Off-policy algorithms can update the estimated value functions using hypothetical actions, those which have not actually been tried.Q-learning is an approach of reinforcement learning that attempts to learn the value of taking each action in each state. This approach is called a model-free approach. Q-learning can be used to find an optimal action-selection policy for any given Markov Decision Process. As we discussed earlier a policy can be defined as a rule that the agent follows in selecting actions, given the state it is in. When such an action-value function is learned, the optimal policy can be constructed by simply selecting the action with the highest value in each state. Q-learning is able to compare the expected utility of the available actions without requiring a model of the environment. Now while solving MDPs we defined the function Q(i,a) which provides the expected value of taking action a in state i. The Q value incorporates both the immediate reward and expected value of the next state. Q-learning converges to an optimal policy in both deterministic and nondeterministic MDPs.

The advantages of Q-learning are it is easy to work with in practice and is exploration insensitive i.e. optimal value will be reached in the end. As the exploration strategy used has no effect on Q-learning it is the most popular and the most effective model-free algorithm for learning from delayed reinforcement. The disadvantage of Q-learning are, it does not scale well i.e. for generalizing over huge number of states and actions convergence is quite slow.Only practical in a small number of problems because Q-learning can require many thousands of training iterations to converge in even modest-sized problems and hence in many situations the memory resources required by this method become too large.

39.4.1 Q Learning Steps

Q-learning augments value iteration by maintaining an *estimated utility value Q(s,a)* for *every action* at every state. The utility of a state Q(s), is simply the maximum Q value over all the possible actions at that state. This method learns utilities of actions not the states and is in essence *model-free learning*

Initialize *Q(s, a)* arbitrarily
Repeat (for each episode)
Initialize *s*
Repeat (for each step of the episode)
Choose *a* from *s* using an exploratory policy
Take action *a*, observe *r, s'*
$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max Q(s', a') - Q(s, a)]$
a'
$s \leftarrow s'$

The working of Q-Learning is as follows: For every state it chooses the action a, which maximizes the function Q(s, a). Q is the estimated utility functionU and tells how good an action is given a certain state. Then update Q(s, At each step s, it chooses a) asthe addition of immediate reward for making the action and best utility (Q) for the resulting state. This is done recursively until convergence.

The Q-Learning rule is given below:

$Q(s, a) = Q(s, a) + \alpha [r + \gamma max\, Q(s', a') - Q(s, a)]$

wheres – current state, s' – next state, a – action, a' – action of the next state, r – immediate reward, γ – discount factor, Q(s,a) – expected discounted reinforcement of taking action a in state s.Here <s, a, r, s'> is an experience tuple.

39.4.2 Action-Value Function

In Q-Learning as the action should yield best next state we need to choose the best action that leads to that state. The action value pair is as given in Figure 39.4.

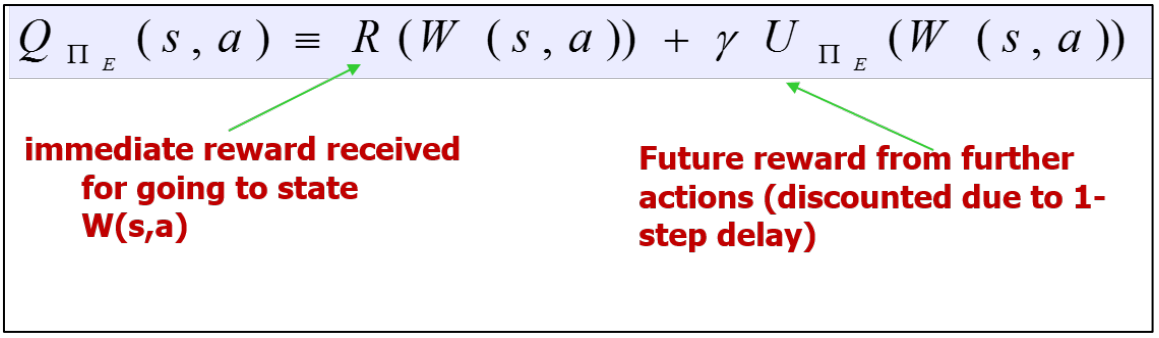


Figure 39.4 Action Value Pair

If the action value function is learnt accurately then the action a is chosen using $a = \arg\max_{a' \in actions} Q(s, a')$.

Figure 39.5shows the graphical representation of Q and U values. The utility values are stored in the nodes or states and q values are stored in the edges or the actions.

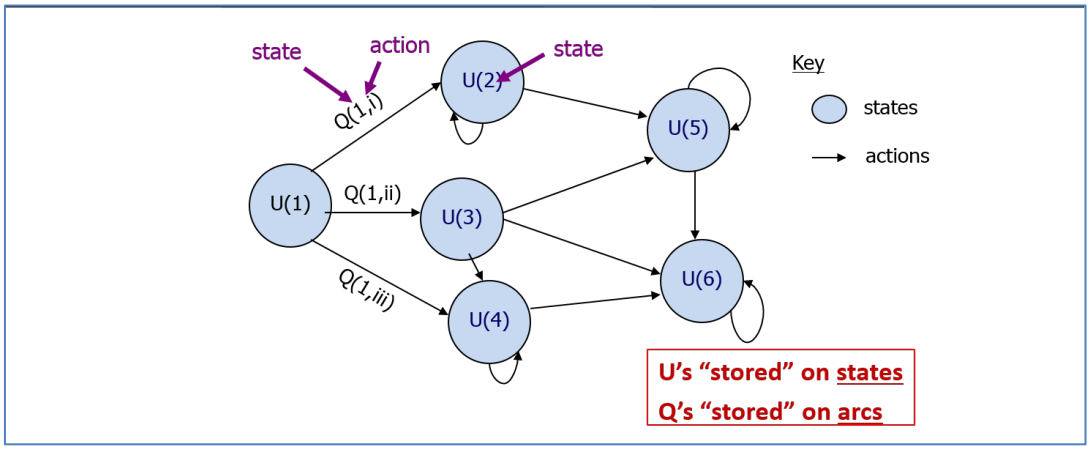


Figure 39.5 Graphical Representation of Q-Learning

Let $Q_t$be our current estimate of the correct Qthen the current policy and utility function estimate are as follows:

$\Pi_t(s) = a$ such that $Q_t(s,a) = \max_{\substack{b \in known \\ actions}} [Q_t(s,b)]$

Our current utility-function estimate is

$$U_t(s) = Q_t(s, \Pi_t(s))$$

As the U table is embedded in the Q tab there is no need to store both. In order to predict the estimate precisely, assume the agent is in state S_t, calculate the reward for some finite time step n.Following the current policy determine the actual reward and compare it with the predicted reward and reduce the error.

39.5 Q-Learning: Model-Free RL

InQ-Learning instead of learning the optimal value function V of the state we directly learn the optimal Q function. Recall that Q(s,a) is expected value of taking action a in state s and then following the optimal policy thereafter. The optimal Q-function satisfies the condition which gives: $V(s) = \max_{a'} Q(s,a')$.

To learn the Q function directly we select an action greedily according to Q(s,a) without using a model

$$Q(s,a) = R(s) + \beta \sum_{s'} T(s,a,s')V(s');$$

$Q(s,a) = R(s) + \beta \sum_{s'} T(s,a,s') \max_{a'} Q(s,a')$ is the bellman constraint on the optimal Q function.

The Q value is updatedas in temporal difference algorithm after taking action a in state s and reaching state s' as shown below and note that the reward R(s) is directly observed $Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \beta \max_{a'} Q(s',a') - Q(s,a))$ where $\alpha(R(s) + \beta \max_{a'} Q(s',a'))$ is the (noisy) sample of Q-value based on next state.

The most important RL algorithm in use today Q-Learning,uses one-step sample backup to learn action-value function Q(s,a). It uses one-step error given below for learning: $error(s,a) = \left\{ r + \gamma \max_b Q(s',b) \right\} - Q(s,a)$ and the incremental learning algorithm is defined as: $\Delta Q(s,a) = \alpha \left(\left\{ r + \gamma \max_b Q(s',b) \right\} - Q(s,a) \right)$ where a(t) follows same schedule as in TD algorithm.

The Q-Learning follows the below steps to learn the function:

1. Start with initial Q-function (e.g. all zeros)
2. Take action according to an explore/exploit policy (should converge to greedy policy, i.e. GLIE)
3. Perform TD update $Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \beta \max_{a'} Q(s',a') - Q(s,a))$
   Q(s,a) is current estimate of optimal Q-function.
4. Go to step 2

The Q-Learning algorithm is given below:

As this is an off-policy method no policy is used and the value of the best next action is used. Q-Learning is used when the environment cannot be controlled example: opponent in chess, the dice in backgammon etc.

Initialize all $Q(s,a)$ arbitrarily
For all episodes
  Initalize s
  Repeat
    Choose a using policy derived from Q, e.g., ϵ-greedy
    Take action a, observe r and s'
    Update $Q(s,a)$:
      $Q(s,a) \leftarrow Q(s,a) + \eta(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$
    $s \leftarrow s'$
  Until s is terminal state

39.6 Sarsa Algorithm

The Sarsa algorithm is an On-Policy algorithm for TD-Learning. The major difference between this algorithm and Q-Learning, is that the maximum reward for the next state is not necessarily used for updating the Q-values. Instead, a new action, and therefore reward, is selected using the same policy that determined the original action. The name Sarsa actually comes from the fact that the updates are done using the quintuple **Q(s, a, r, s', a')** where: **s, a** are the original state and action, **r** is the reward observed in the following state and **s', a'** are the new state-action pair. The On-policy SARSAinstead of looking for all possible next actions *a'* and choosing the best,uses the policy derived from *Q* values to choose *a'*.It uses the *Q* value of a' to calculate the temporal difference.On employing the GLIE (greedy in the limit with infinite exploration) policy where (1) all state-actionpairs are visited an infinite number of times, and (2) the policyconverges in the limit to the greedy policy, the algorithm converges with probability 1 to the optimal policy and state-actionvalues.

Initialize all $Q(s,a)$ arbitrarily
For all episodes
  Initalize s
  Choose a using policy derived from Q, e.g., ϵ-greedy
  Repeat
    Take action a, observe r and s'
    Choose a' using policy derived from Q, e.g., ϵ-greedy
    Update $Q(s,a)$:
      $Q(s,a) \leftarrow Q(s,a) + \eta(r + \gamma Q(s',a') - Q(s,a))$
    $s \leftarrow s',\ a \leftarrow a'$
  Until s is terminal state

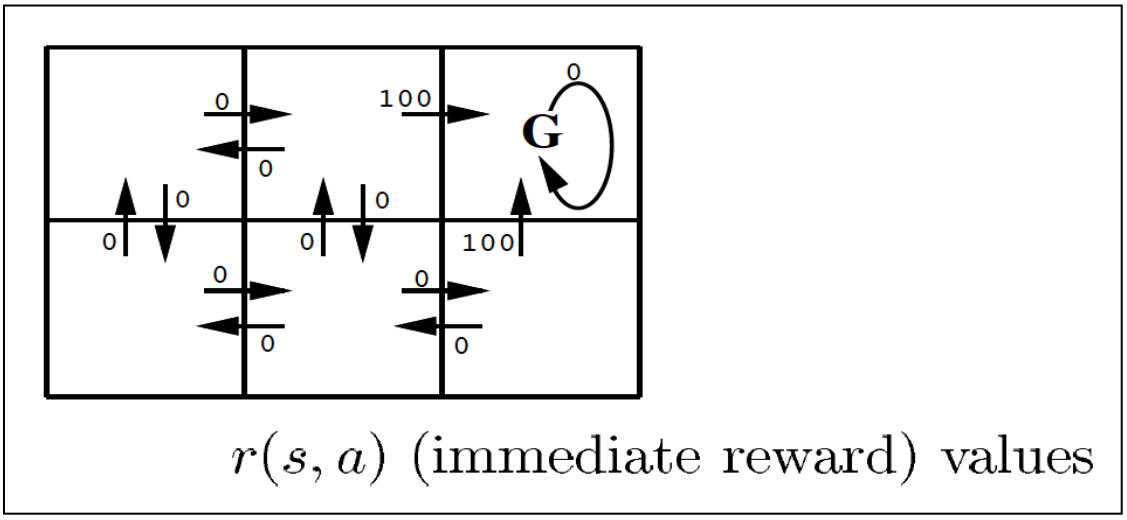
39.7 Value-Based TD vs. Q-learning

Both V-function and the Q-functioncan be learnt by the Temporal Difference approach but are used in different circumstances. If state space is small then this may not be such an important issue but if the state-spaces is large we have to choose based on the application. The Value-Based approach learns a model and utility function but it is difficult to learn good models for large complex environments(e.g. learning a DBN representation). If we can learn a model then learning utility function is simpler than learning Q(s,a) and it can be reused for related problems. In Q-learning approach the Q-functionis what is learnt which is simpler to implement since we do not need to worry about representing and learning a model. But Q-functions can be substantially more complex than utility functions a small price to be paid for not having the model.

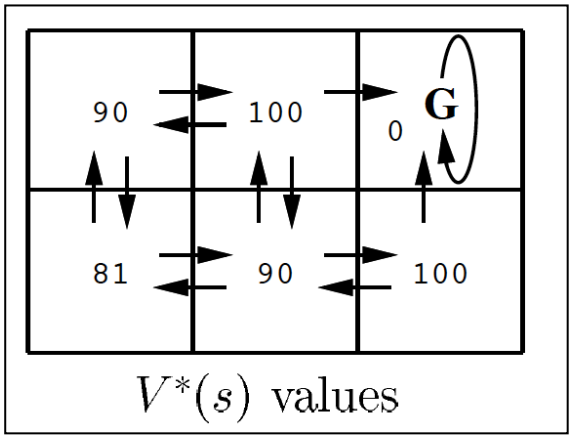
39.8 Q-Learning – Example

An example to demonstrate the Q-Learning process with a simple grid-world environment is shown in the Figure 39.6 (a). The six grid squares represents six possible states, or locations, each arrow represents a possible action the agent can take to move from one state to another and the number associated with each arrow represents the immediate reward r(s,a)the agent receives on performing the corresponding state-action transition. The state G is the goal state, the agentreceives the reward by entering this state and the immediate rewardforall state-action transitions except for those leading into the state labelled G is defined to be zero. Theonly action the agent can perform once it enters the state G isto remain in this state as G an absorbing state.All the states, actions, and immediate rewards are defined, the value for the discount factor $\lambda$is chosen as λ= 0.9, so that the optimal policy n*, its value function V*(s) can be determined (Figure 39.4 (b)) and the Q(s,a) (Figure 39.4 (c)). Figure 39.4(d) shows one of the optimal policies for this setting thatdirectsthe agent along the shortest path toward the state G.
Figure 39.5 shows the values of V*(S)for eachstate and Q(s,a) for the state. The value ofV* for this state is 100 as it receives immediate reward 100 since the optimal policy in this state selects the "moveup" action. Since the agent will remainin this absorbing state and hence no further rewards are received. Likewise, for the bottom center state value of V*is 90 since the optimal policy will movethe agent from this state to the right andthen upward which generates an immediate reward of 0 and

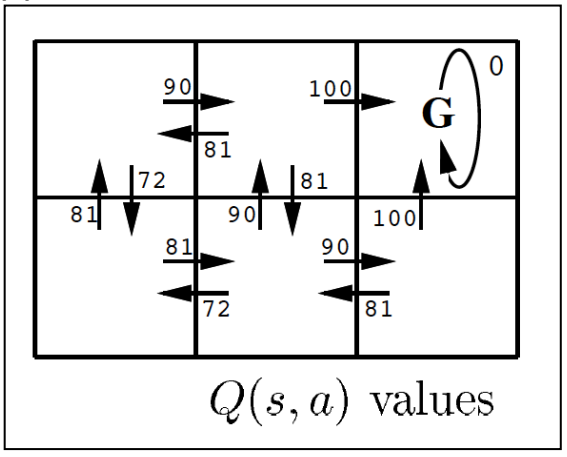
100respectively for the actions. Hence we get the discounted futurereward from the bottom center state as 0 +y*100+y^{2*}0+y^{3*}0+.....=90.As *V** is the sum of discounted future rewards over the infinite future. Once the absorbing state *G is reached by the agent* it remains in that state infinitely and receives rewards of zero.



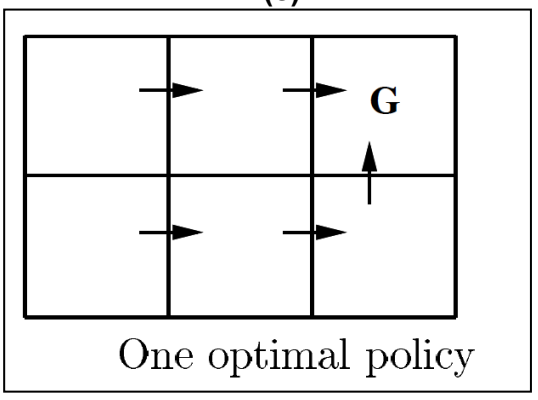
(a)



(b)



(c)



(d)

Figure 39.6 Example for Q-Learning

In many real problems calculating the state value is unrealistic. For example what will be the reward for the robot exploring mars which decides to take a right turn?Thisproblem can be evadedby exploring and experiencing how the environment reacts to the actions. It is difficult for the agent to learn the optimal policy with the sequence of immediate rewards. Hence a numerical evaluation function is defined over states and actions and the optimal policy is learnt in terms of the evaluation

function. If theagent hasperfect knowledge of the immediate reward function ***r*** and the state transitionfunction delta then the optimal policy can be acquired by learning *V**.We want a function Q(s,a) that directly learns good state-action pairs, i.e. what action should I take in this state. Given Q(s,a) it is now simple to execute the optimal policy, *without knowing r(s,a) and delta(s,a) since* $\pi^*(s) = \arg\max_a Q(s,a)$, $V^*(s) = \max_a Q(s,a)$ and Q(s,a) = r(s,a) + γV*(δ(s,a)) = r(s,a) + $\gamma \max_{a'} Q(\delta(s,a),a')$, and Q(s,a) depends on r and delta.Consider a robot is exploring its environment and taking new actions as it goes, it receives some reward "r" at every step and observes the environment changes i.e. the state is now changed to new s' for the action a. we can use these observations, (s,a,s',r) to learn a model $\hat{Q}$(s,a) = r + $\gamma \max_{a'} \hat{Q}$(s',a') where s' = s_{t+1}. This equation continually estimates Q at state s consistent with an estimate of Q at state s', one step in the future: temporal difference (TD) learning.Note that s' is closer to goal, and hence is more reliable, but is still an estimate itself. This type of updating estimates based on other estimates is called bootstrapping and is done after each state-action pairi.e. the agent learns online. The useful things we learn about explored state-action pairsare most useful as they are likely to be encountered again. Under suitable conditions, these updates can actually be proved to converge to the real answer.

Figure 39.7 shows the initial state *s1* of the robot (R) and relevant $\hat{Q}$ values in its initial hypothesis. For example, the value $\hat{Q}$ *(s1, aright)* = 72, where *aright* refers to the action that moves R to its right. On executing the action the robot receives an immediate reward r = 0 and moves to state *s2*. Then updates its estimate $\hat{Q}$ *(sl, aright)* based on its $\hat{Q}$ estimates for the new state *s2 as Q-learning propagates Q-estimates 1-step backwards.* The value of γ *is chosen as* 0.9 and the value is updated to $\hat{Q}$*(s1, aright)* = 90 after executing the single action as shown below.

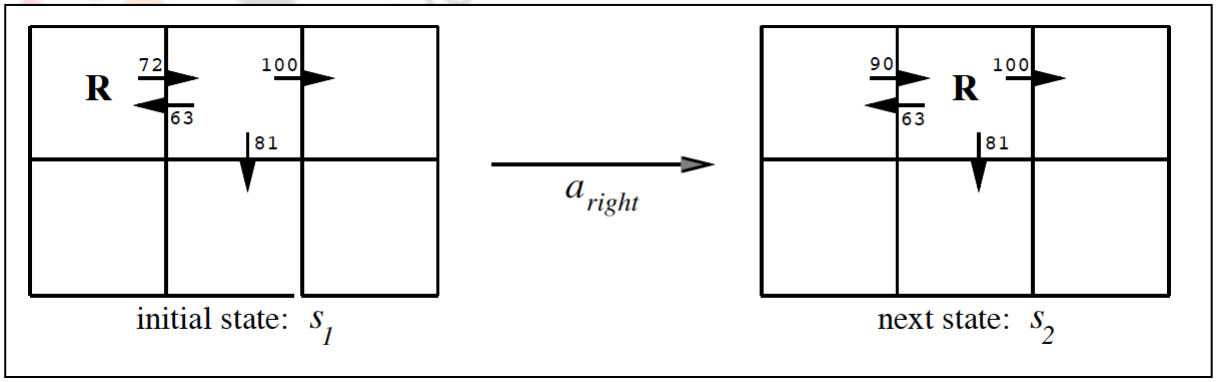


Figure 39.7Action for the example in Figure 39.6

$$\hat{Q}(s_1, a_{right}) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a')$$
$$\leftarrow 0 + 0.9 \max\{66, 81, 100\}$$
$$\leftarrow 90$$

Figure 39.4

When learning Q (off-policy learning)the agent should not simply follow the current policy.Because the agent may get stuck in a suboptimal solution as there may be other solutions out there that the agent has never seen. So it is good to try new things now and then. Generally if T value is large do lot of exploring, if T value small follow the current policy and the value can be decreased over timeas a tradeoff between exploration and exploitation.

To improve Q-Learning as a trade-off between memory and computation one can cache the (s,s',r) for observed transitions and replay the update $\hat{Q}$(s,a) = r + $\gamma \max_{a'} \hat{Q}$(s',a') after a while, as Q(s',a') has changed.Other methods to improve Q-Learning are

a). Anactive search for state-action pairs for which Q(s,a) is expected to change a lot (prioritized sweeping) can be done or

b). Updatescan be done for more than just one step along the sampled path (TD(λ) learning).

c) To deal with stochastic environments, the*expected* future discounted reward: $Q(s,a) = E[r(s,a)] + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q(s',a')$ needs to be maximized.Since the state space is too large to deal with all statesweneed to learn a function: Q(s,a) ≈ f_θ(s,a). Neural network with back-propagation have been quite successful for these types of environments.For example in the back-gammon program, TD-Gammon which plays at expert level the state-space is very large. Hence Neural Networks is used to approximate the value function and TD(lambda) is used for learning by TD-Gammon which trains by playing against itself.

Summary

- Outlined the value iterations, policy iterations, TD Learning and exploration strategies
- Explained deterministic vs nondeterministic rewards and actions
- Reviewed the Model based vs Model free RL
- Discussed Q-learning steps, example, and improvements on Q-learning