# MODULE 5 INSTANCE BASED LEARNING

## INTRODUCTION

* Instance-based learning methods such as nearest neighbor and locally weighted regression are conceptually straightforward approaches to approximating real-valued or discrete-valued target functions.
* Learning in these algorithms consists of simply storing the presented training data. When a new query instance is encountered, a set of similar related instances is retrieved from memory and used to classify the new query instance
* Instance-based approaches can construct a different approximation to the target function for each distinct query instance that must be classified

### Advantages of Instance-based learning

1. Training is very fast
2. Learn complex target function
3. Don’t lose information

### Disadvantages of Instance-based learning

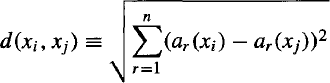
* The cost of classifying new instances can be high. This is due to the fact that nearly all computation takes place at classification time rather than when the training examples are first encountered.
* In many instance-based approaches, especially nearest-neighbor approaches, is that they typically consider all attributes of the instances when attempting to retrieve similar training examples from memory. If the target concept depends on only a few of the many available attributes, then the instances that are truly most "similar" may well be a large distance apart.

## NEAREST NEIGHBOR LEARNING

* + The most basic instance-based method is the K- Nearest Neighbor Learning. This algorithm assumes all instances correspond to points in the n-dimensional space Rn.
  + The nearest neighbors of an instance are defined in terms of the standard Euclidean distance.
  + Let an arbitrary instance x be described by the feature vector

((a1(x), a2(x), ………, an(x))

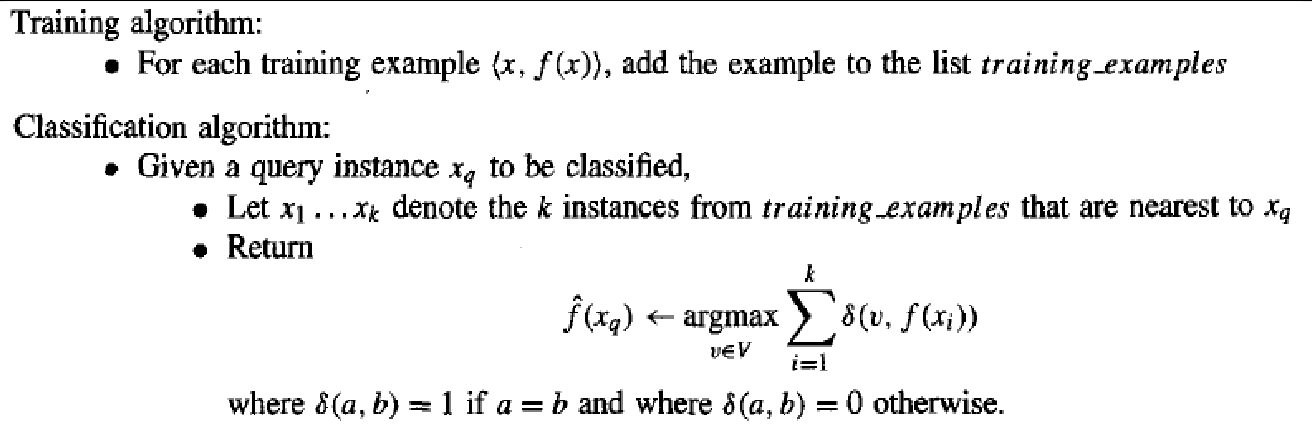
Where, ar(x) denotes the value of the rth attribute of instance x.

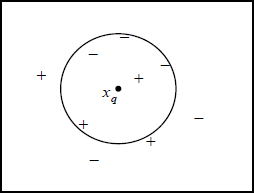
* + Then the distance between two instances xi and xj is defined to be d(xi , xj ) Where,
  + In nearest-neighbor learning the target function may be either discrete-valued or real- valued.

Let us first consider learning ***discrete-valued target functions*** of the form Where, V is the finite set {v1, . . . vs }

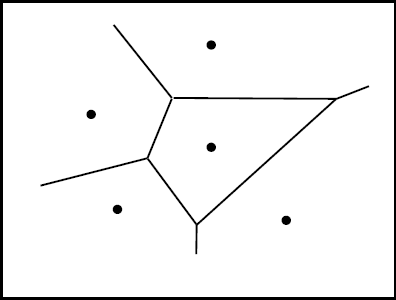
The k- Nearest Neighbor algorithm for approximation a **discrete-valued target function** is

given below:

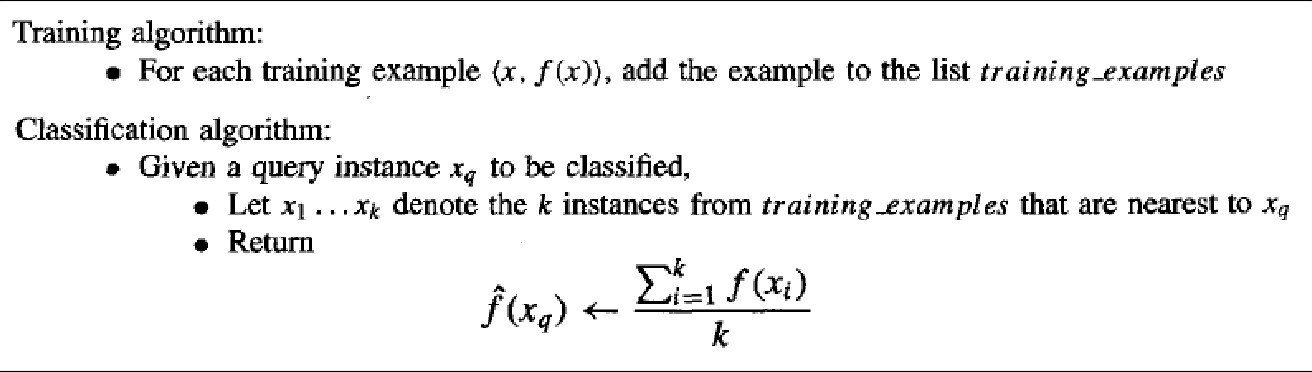


* + The value 𝑓̂(xq) returned by this algorithm as its estimate of f(xq) is just the most common value of f among the k training examples nearest to xq.
  + If k = 1, then the 1- Nearest Neighbor algorithm assigns to 𝑓̂(xq) the value f(xi). Where xi is the training instance nearest to xq.
  + For larger values of k, the algorithm assigns the most common value among the k nearest training examples.
* Below figure illustrates the operation of the k-Nearest Neighbor algorithm for the case where the instances are points in a two-dimensional space and where the target function is Boolean valued.
  + The positive and negative training examples are shown by “+” and “-” respectively. A

query point xq is shown as well.

* + The 1-Nearest Neighbor algorithm classifies xq as a positive example in this figure, whereas the 5-Nearest Neighbor algorithm classifies it as a negative example.
* Below figure shows the shape of this **decision surface** induced by 1- Nearest Neighbor over the entire instance space. The decision surface is a combination of convex polyhedra surrounding each of the training examples.
  + For every training example, the polyhedron indicates the set of query points whose classification will be completely determined by that training example. Query points outside the polyhedron are closer to some other training example. This kind of diagram is often called the ***Voronoi diagram*** of the set of training example

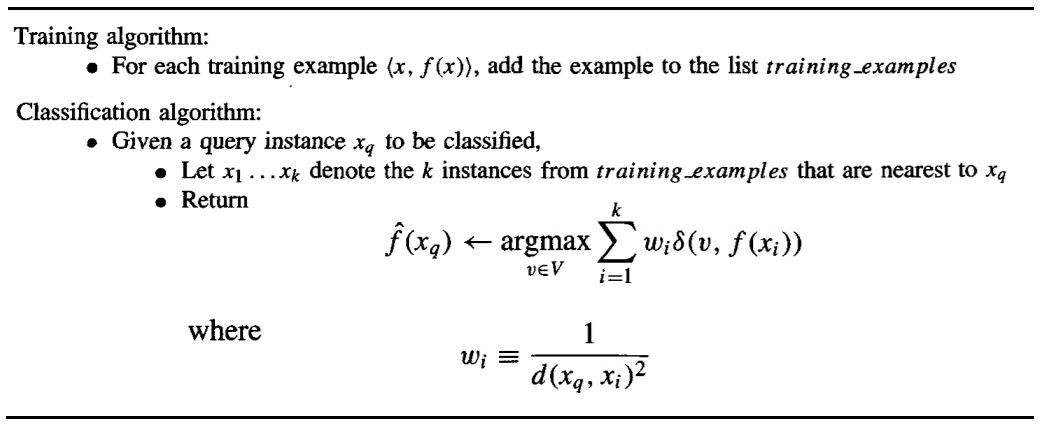
The K- Nearest Neighbor algorithm for approximation a **real-valued target function** is given below



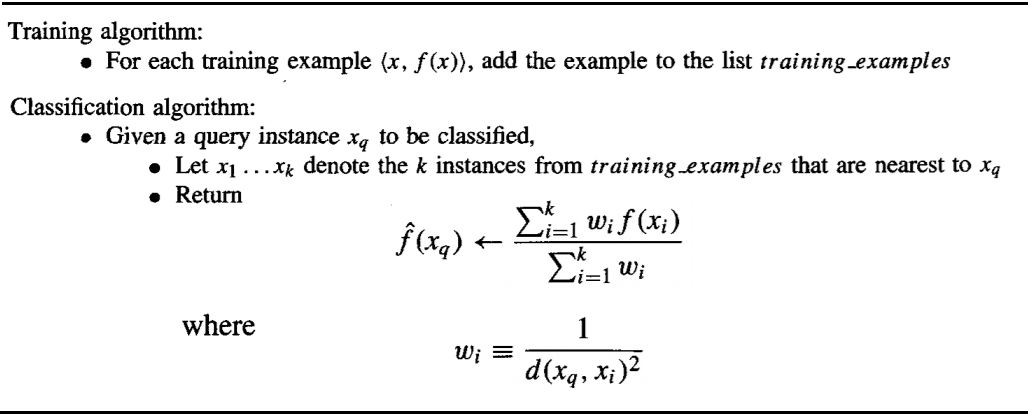
### Distance-Weighted Nearest Neighbor Algorithm

* + The refinement to the k-NEAREST NEIGHBOR Algorithm is to weight the contribution of each of the k neighbors according to their distance to the query point xq, giving greater weight to closer neighbors.
  + For example, in the k-Nearest Neighbor algorithm, which approximates discrete-valued target functions, we might weight the vote of each neighbor according to the inverse square of its distance from xq

Distance-Weighted Nearest Neighbor Algorithm for approximation a discrete-valued target functions



Distance-Weighted Nearest Neighbor Algorithm for approximation a Real-valued target functions



### Terminology

* + **Regression** means approximating a real-valued target function.
  + **Residual** is the error 𝑓̂(x) - f (x) in approximating the target function.
  + **Kernel function** is the function of distance that is used to determine the weight of each training example. In other words, the kernel function is the function K such that

wi = K(d(xi, xq))

## LOCALLY WEIGHTED REGRESSION

* + The phrase "**locally weighted regression**" is called **local** because the function is approximated based only on data near the query point, **weighted** because the contribution of each training example is weighted by its distance from the query point, and **regression** because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.
  + Given a new query instance xq, the general approach in locally weighted regression is to construct an approximation 𝑓̂ that fits the training examples in the neighborhood surrounding xq. This approximation is then used to calculate the value 𝑓̂(xq), which is output as the estimated target value for the query instance.

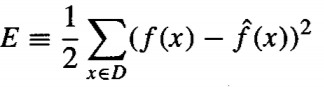
## Locally Weighted Linear Regression

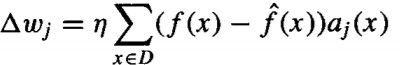
* + Consider locally weighted regression in which the target function ***f*** is approximated near xq using a linear function of the form



Where, ai(x) denotes the value of the ith attribute of the instance x

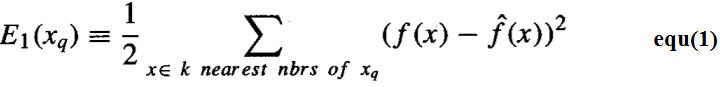
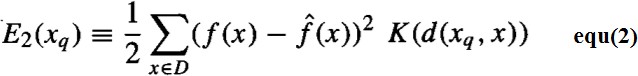
* + Derived methods are used to choose weights that minimize the squared error summed over the set D of training examples using gradient descent

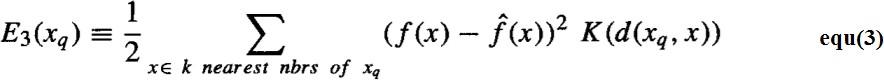


Which led us to the gradient descent training rule

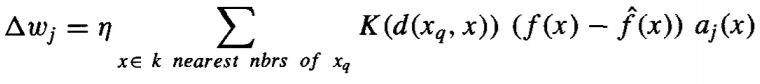
Where, η is a constant learning rate

* + Need to modify this procedure to derive a local approximation rather than a global one. The simple way is to redefine the error criterion E to emphasize fitting the local training examples. Three possible criteria are given below.

1. Minimize the squared error over just the k nearest neighbors:
2. Minimize the squared error over the entire set D of training examples, while weighting the error of each training example by some decreasing function K of its distance from xq :
3. Combine 1 and 2:

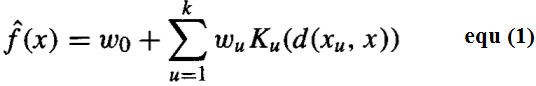


If we choose criterion three and re-derive the gradient descent rule, we obtain the following training rule



The differences between this new rule and the rule given by Equation (3) are that the contribution of instance x to the weight update is now multiplied by the distance penalty **K(d(xq, x)),** and that the error is summed over only the k nearest training examples.

## RADIAL BASIS FUNCTIONS

* + One approach to function approximation that is closely related to distance-weighted regression and also to artificial neural networks is learning with radial basis functions
  + In this approach, the learned hypothesis is a function of the form
  + Where, each xu is an instance from X and where the kernel function Ku(d(xu, x)) is defined so that it decreases as the distance d(xu, x) increases.
  + Here k is a user provided constant that specifies the number of kernel functions to be included.
  + 𝑓̂ is a global approximation to f (x), the contribution from each of the Ku(d(xu, x)) terms is localized to a region nearby the point xu.

Choose each function Ku(d(xu, x)) to be a Gaussian function centred at the point xu with some variance 𝜎 2

u



* + The functional form of equ(1) can approximate any function with arbitrarily small error, provided a sufficiently large number k of such Gaussian kernels and provided the width

𝜎2 of each kernel can be separately specified

* + The function given by equ(1) can be viewed as describing a two layer network where the first layer of units computes the values of the various Ku(d(xu, x)) and where the second layer computes a linear combination of these first-layer unit values

### Example: Radial basis function (RBF) network

Given a set of training examples of the target function, RBF networks are typically trained in a two-stage process.

1. First, the number k of hidden units is determined and each hidden unit u is defined by choosing the values of xu and 𝜎 2 that define its kernel function Ku(d(xu, x))

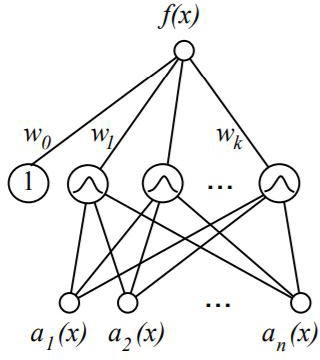
u

1. Second, the weights w, are trained to maximize the fit of the network to the training

data, using the global error criterion given by



Because the kernel functions are held fixed during this second stage, the linear weight values w, can be trained very efficiently



Several alternative methods have been proposed for choosing an appropriate number of hidden units or, equivalently, kernel functions.

* + One approach is to allocate a Gaussian kernel function for each training example (xi,f (xi)), centring this Gaussian at the point xi.

Each of these kernels may be assigned the same width 𝜎2. Given this approach, the RBF network learns a global approximation to the target function in which each training example (xi, f (xi)) can influence the value of ***f*** only in the neighbourhood of xi.

* + A second approach is to choose a set of kernel functions that is smaller than the number

of training examples. This approach can be much more efficient than the first approach, especially when the number of training examples is large.

### Summary

* + Radial basis function networks provide a global approximation to the target function, represented by a linear combination of many local kernel functions.
  + The value for any given kernel function is non-negligible only when the input x falls into the region defined by its particular centre and width. Thus, the network can be viewed as a smooth linear combination of many local approximations to the target function.
  + One key advantage to RBF networks is that they can be trained much more efficiently than feedforward networks trained with BACKPROPAGATION.

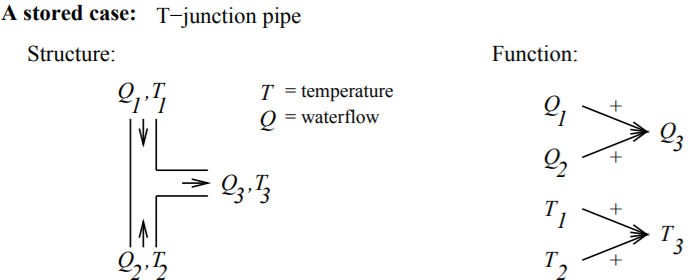
### CASE-BASED REASONING

* + Case-based reasoning (CBR) is a learning paradigm based on lazy learning methods and they classify new query instances by analysing similar instances while ignoring instances that are very different from the query.
  + In CBR represent instances are not represented as real-valued points, but instead, they use a ***rich symbolic*** representation.
  + CBR has been applied to problems such as conceptual design of mechanical devices based on a stored library of previous designs, reasoning about new legal cases based on previous rulings, and solving planning and scheduling problems by reusing and combining portions of previous solutions to similar problems

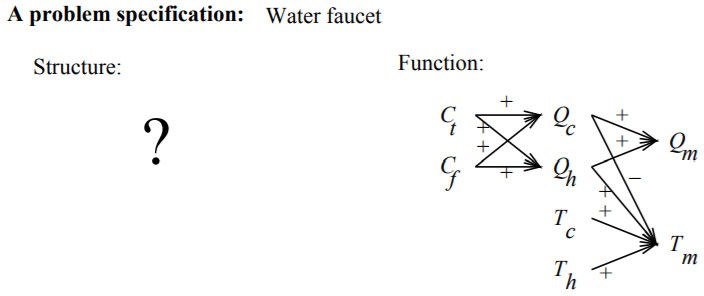
**A prototypical example of a case-based reasoning**

* + The CADET system employs case-based reasoning to assist in the conceptual design of simple mechanical devices such as water faucets.
  + It uses a library containing approximately 75 previous designs and design fragments to suggest conceptual designs to meet the specifications of new design problems.
  + Each instance stored in memory (e.g., a water pipe) is represented by describing both its structure and its qualitative function.
  + New design problems are then presented by specifying the desired function and requesting the corresponding structure.

The problem setting is illustrated in below figure



* + The function is represented in terms of the qualitative relationships among the water- flow levels and temperatures at its inputs and outputs.
  + In the functional description, an arrow with a "+" label indicates that the variable at the arrowhead increases with the variable at its tail. A "-" label indicates that the variable at the head decreases with the variable at the tail.
  + Here Qc refers to the flow of cold water into the faucet, Qh to the input flow of hot water, and Qm to the single mixed flow out of the faucet.
  + Tc, Th, and Tm refer to the temperatures of the cold water, hot water, and mixed water respectively.
  + The variable Ct denotes the control signal for temperature that is input to the faucet, and Cf denotes the control signal for waterflow.
  + The controls Ct and Cf are to influence the water flows Qc and Qh, thereby indirectly influencing the faucet output flow Qm and temperature Tm.



* + CADET searches its library for stored cases whose functional descriptions match the design problem. If an exact match is found, indicating that some stored case implements exactly the desired function, then this case can be returned as a suggested solution to the design problem. If no exact match occurs, CADET may find cases that match various subgraphs of the desired functional specification.

# 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.

## INTRODUCTION

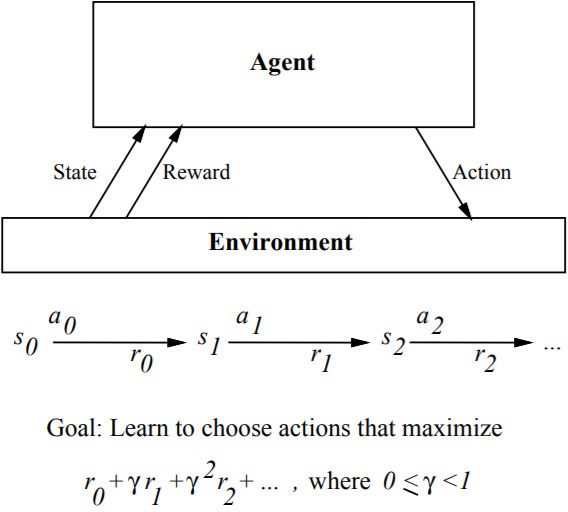
* + Consider building a **learning robot**. The robot, or ***agent***, has a set of sensors to observe the state of its environment, and a set of actions it can perform to alter this state.
  + Its task is to learn a control strategy, or ***policy***, for choosing actions that achieve its goals.
  + The goals of the agent can be defined by a ***reward function*** that assigns a numerical value to each distinct action the agent may take from each distinct state.
  + 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.
  + The control policy is one that, from any initial state, chooses actions that maximize the reward accumulated over time by the agent.

### Example:

* + A mobile robot may have sensors such as a camera and sonars, and actions such as "move forward" and "turn."
  + The robot may have a goal of docking onto its battery charger whenever its battery level is low.
  + The goal of docking to the battery charger can be captured by assigning a positive reward (Eg., +100) to state-action transitions that immediately result in a connection to the charger and a reward of zero to every other state-action transition.

### Reinforcement Learning Problem

* + An agent interacting with its environment. The agent exists in an environment described by some set of possible states S.
  + Agent perform any of a set of possible actions A. Each time it performs an action a, in some state st the agent receives a real-valued reward r, that indicates the immediate value of this state-action transition. This produces a sequence of states si, actions ai, and immediate rewards ri as shown in the figure.
  + The agent's task is to learn a control policy, 𝝅**: S → A**, that maximizes the expected sum of these rewards, with future rewards discounted exponentially by their delay.



### Reinforcement learning problem characteristics

1. **Delayed reward**: The task of the agent is to learn a target function 𝜋 that maps from the current state s to the optimal action a = 𝜋 (s). In reinforcement learning, training information is not available in (s, 𝜋 (s)). 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.
2. **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 trade-off in choosing whether to favor exploration of unknown states and actions, or exploitation of states and actions that it has already learned will yield high reward.
3. **Partially observable states:** 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. In such cases, the agent needs 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.
4. **Life-long learning:** Robot requires to 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

* + Consider Markov decision process (MDP) where 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). Here the functions δ(st, at) and r(st, at) depend only on the current state and action, and not on earlier states or actions.

The task of the agent is to learn a policy, 𝝅**: S → A**, for selecting its next action a, based on the current observed state st; that is, 𝝅**(st) = at**.

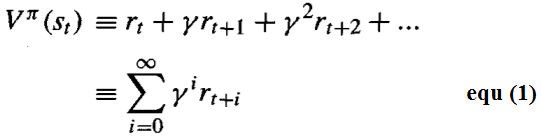
#### How shall we specify precisely which policy π we would like the agent to learn?

1. One approach is to require the policy that produces the greatest possible ***cumulative reward***

for the robot over time.

* + To state this requirement more precisely, define the cumulative value Vπ (st) achieved

by following an arbitrary policy π from an arbitrary initial state st as follows:



* + Where, the sequence of rewards rt+i is generated by beginning at state st and by

repeatedly using the policy π to select actions.

* + Here 0 ≤ γ ≤ 1 is a constant that determines the relative value of delayed versus immediate rewards. if we set γ = 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 Vπ (st) is 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.

1. Other definitions of total reward is ***finite horizon reward,***

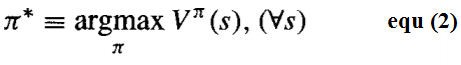
Considers the undiscounted sum of rewards over a finite number ***h*** of steps

1. Another approach is ***average reward***



Considers the average reward per time step over the entire lifetime of the agent.

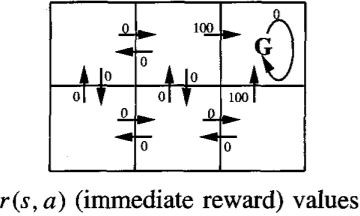
We require that the agent learn a policy π that maximizes Vπ (st) for all states s. such a policy is called an ***optimal policy*** and denote it by π\*



Refer the value function Vπ\*(s) an optimal policy as V\*(s). V\*(s) gives the maximum discounted cumulative reward that the agent can obtain starting from state ***s***.

**Example:**

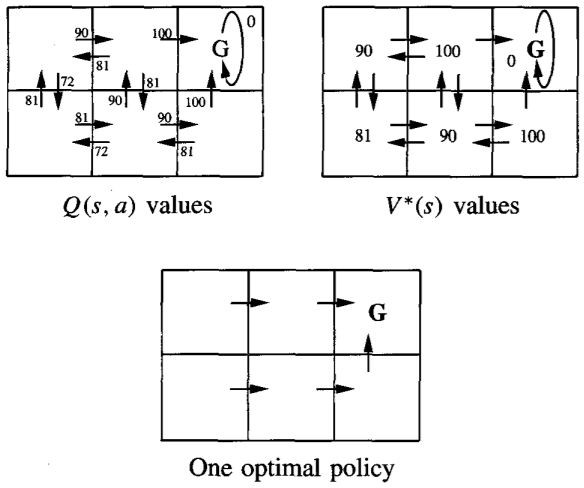
A simple grid-world environment is depicted in the diagram



* + 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 environment is defined to be zero for all state-action transitions except for those leading into the state labelled G. The state G as the goal state, and the agent can receive reward by entering this state.

Once the states, actions, and immediate rewards are defined, choose a value for the discount factor γ, determine the optimal policy π \* and its value function V\*(s).

Let’s choose γ = 0.9. The diagram at the bottom of the figure shows one optimal policy for this setting.



Values of V\*(s) and Q(s, a) follow from r(s, a), and the discount factor γ = 0.9. An optimal

policy, corresponding to actions with maximal Q values, is also shown.

The discounted future reward from the bottom centre state is

0+ γ 100+ γ2 0+ γ3 0+... = 90

### *Q* LEARNING

#### How can an agent learn an optimal policy π \* for an arbitrary environment?

The 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.

#### What evaluation function should the agent attempt to learn?

One obvious choice is V\*. The agent should prefer state sl over state s2 whenever V\*(sl) > V\*(s2), because the cumulative future reward will be greater from sl

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 γ.



### The *Q* Function

The value of Evaluation function ***Q(s, a)*** is the reward received immediately upon executing action a from state ***s***, plus the value (discounted by ***γ*** ) of following the optimal policy thereafter

Rewrite Equation (3) in terms of ***Q(s, a)*** as



Equation (5) makes clear, it need only consider each available action ***a*** in its current state ***s*** and choose the action that maximizes ***Q(s, a)***.

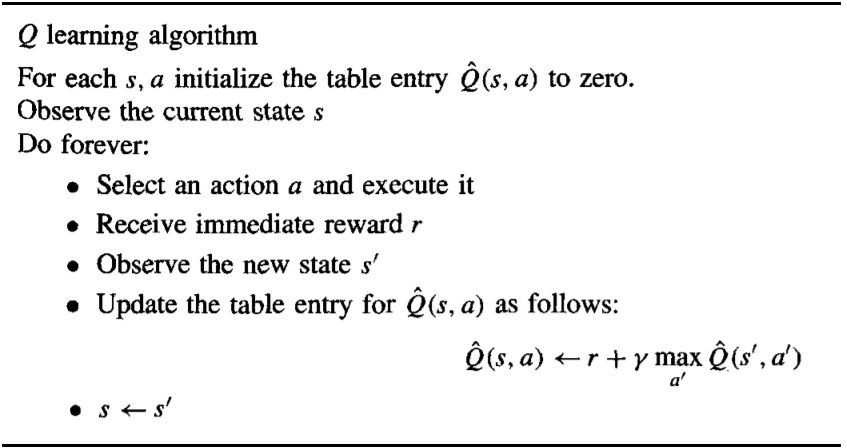
### An Algorithm for Learning *Q*

* + 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***

Rewriting Equation



### Q learning algorithm:



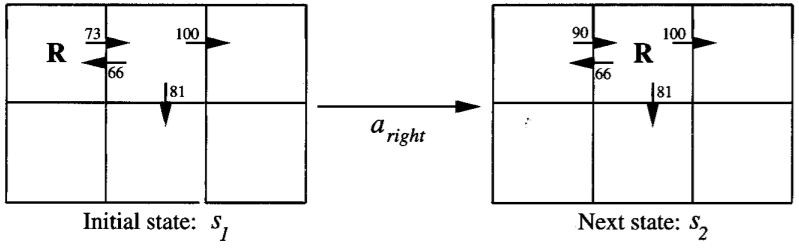
* + Q learning algorithm assuming deterministic rewards and actions. The discount factor

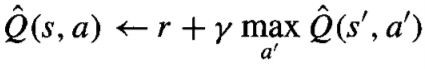
γ may be any constant such that 0 ≤ γ < 1

* + 𝑄̂ to refer to the learner's estimate, or hypothesis, of the actual Q function

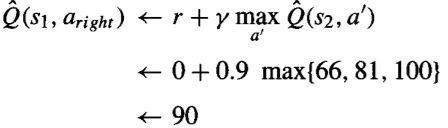
### An Illustrative Example

* + To illustrate the operation of the Q learning algorithm, consider a single action taken by an agent, and the corresponding refinement to 𝑄̂ shown in below figure



* + The agent moves one cell to the right in its grid world and receives an immediate reward of zero for this transition.
  + Apply the training rule of Equation

to refine its estimate Q for the state-action transition it just executed.

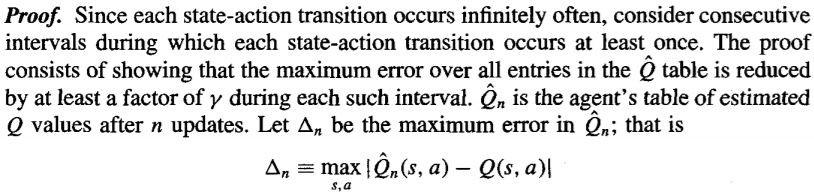
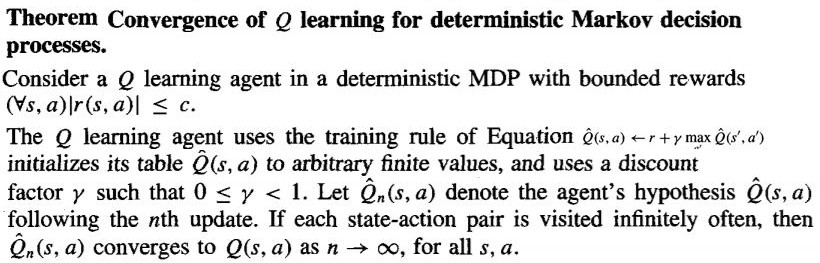
* + According to the training rule, the new 𝑄̂ estimate for this transition is the sum of the received reward (zero) and the highest 𝑄̂ value associated with the resulting state (100), discounted by γ (.9).

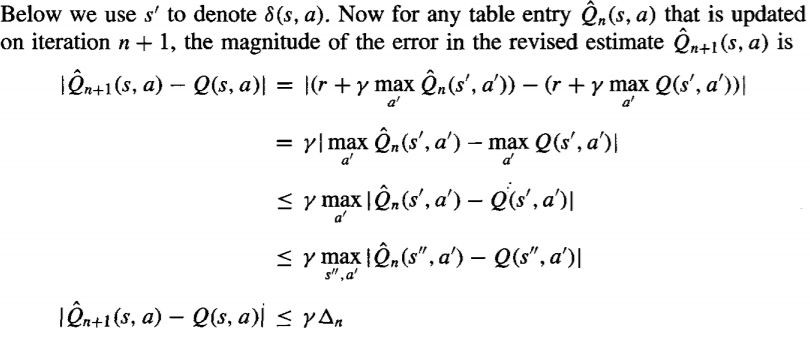
### Convergence

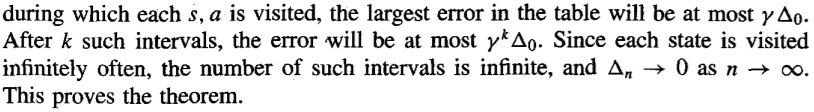
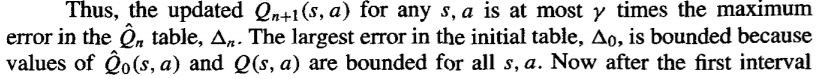
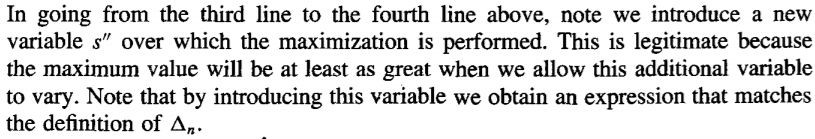
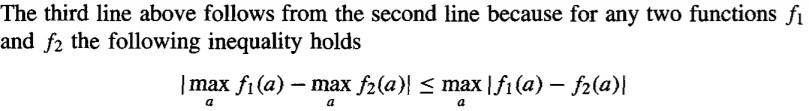
#### Will the Q Learning Algorithm converge toward a Q equal to the true Q function?

Yes, under certain conditions.

1. Assume the system is a deterministic MDP.
2. Assume the immediate reward values are bounded; that is, there exists some positive constant c such that for all states s and actions a, **| r(s, a)| < c**
3. Assume the agent selects actions in such a fashion that it visits every possible state- action pair infinitely often

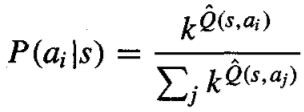






### Experimentation Strategies

#### The Q learning algorithm does not specify how actions are chosen by the agent.

* + One obvious strategy would be for the agent in state ***s*** to select the action ***a*** that maximizes 𝑄̂(s, a), thereby exploiting its current approximation 𝑄̂.
  + However, with this strategy the agent runs the risk that it will overcommit to actions that are found during early training to have high Q values, while failing to explore other actions that have even higher values.
  + For this reason, Q learning uses a probabilistic approach to selecting actions. Actions with higher 𝑄̂ values are assigned higher probabilities, but every action is assigned a nonzero probability.
  + One way to assign such probabilities is

Where, **P(ai |s)** is the probability of selecting action **ai**, given that the agent is in state **s**, and **k > 0** is a constant that determines how strongly the selection favors actions with high 𝑄̂ values