Module 5

Chapter 8: Instance Based Learning

Chapter 13: Reinforcement Learning

Module 5

Chapter 8: Instance Based Learning

1. **Introduction**
2. K-nearest neighbor Learning
3. Locally Weighted regression
4. Radial basis functions
5. Case based reasoning
6. Summary

Introduction

MACHINE LEARNING

- all learning methods presented so far construct a general explicit description of the target function when examples are provided
- **Instance-based learning:**
  - examples are simply stored
  - generalizing is postponed until a new instance must be classified
  - in order to assign a target function value, its relationship to the previously stored examples is examined
  - sometimes referred to as **lazy learning**

Introduction

MACHINE LEARNING

- **advantages:**
  - instead of estimating for the whole instance space, local approximations to the target function are possible
  - especially if target function is complex but still decomposable

- **disadvantages:**
  - classification costs are high
    - efficient techniques for indexing examples are important to reduce computational effort
  - typically all attributes are considered when attempting to retrieve similar training examples
    - if the concept depends only on a few attributes, the truly most similar instances may be far away

K-nearest neighbor learning

(For classification and regression)

MACHINE LEARNING

- most basic instance-based method
- **assumption:**
  - instances correspond to a point in a n-dimensional space $\Re^n$
  - thus, nearest neighbors are defined in terms of the standard **Euclidean Distance**

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

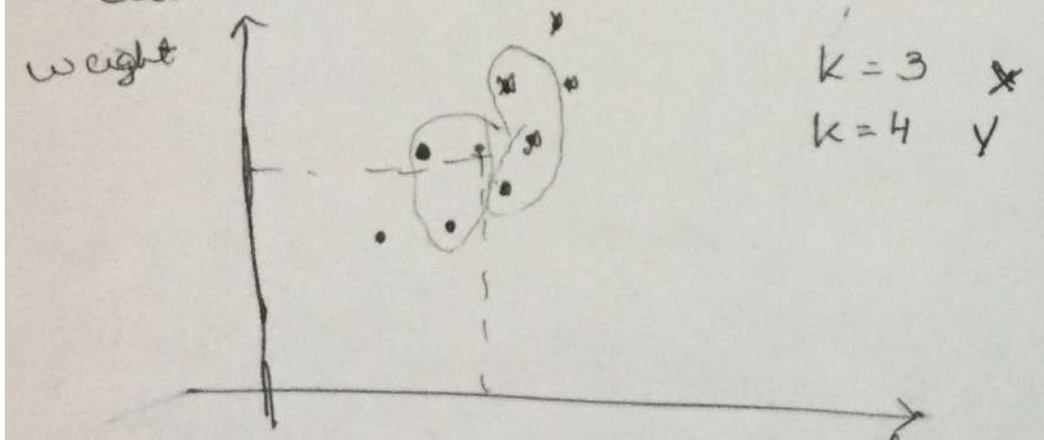
where an instance x is described by $< a_1(x), a_2(x), ..., a_n(x) >$

- target function may be either discrete-valued or real-valued

K-nearest neighbour

Supervised learning

Diameter	Fruit
36	lemon
58	apple
122	watermelon.



major drawback

1) perform KNN- classification Algo on following dataset & predict the class for x ($P_1 = 3$ and $P_2 = 7$) for $K = 3$.

	P_1	P_2	class
(i)	7	7	False
(ii)	7	4	False
(iii)	3	4	True
(iv)	1	4	True

Euclidian distance $= \sqrt{(x_H - H_1)^2 + (x_W - W)^2 + \dots}$

x_H → observed value, H_1 → actual value

$= \sqrt{\quad}$

$= \sqrt{3 \cdot}$

$D(x_s i) = \sqrt{(3-7)^2 + (7-7)^2}$

$= (4)_4 \rightarrow N_3 \Rightarrow$ False

$D(x_s ii) = \sqrt{(3-7)^2 + (7-4)^2}$

$= \sqrt{4^2 + 3^2}$

$= \sqrt{16+9}$

$= \sqrt{25} = (5)$.

$D(x_s iii) = (3) \rightarrow N_1 \Rightarrow$ True

$D(x_s iv) = \sqrt{13} = (3.6)$ N_2 → True

so, when $k = 3$ then it belongs to True class.

X → Instance arbitary

$((a_1(x), a_2(x) \dots a_n(x))$

Feature vector

x_q → Instance query.

K-nearest neighbor learning

- **discrete-valued target function:**
  - $f : \Re^n \rightarrow V$ where V is the finite set $\{v_1, v_2, ..., v_s\}$
  - the target function value is the most common value among the k nearest training examples

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{argmax} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = (a == b)$

- **continuous-valued target function:**
  - algorithm has to calculate the mean value instead of the most common value
  - $f : \Re^n \rightarrow \Re$

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

KNN Algorithm for classification

MACHINE LEARNING

Training algorithm:

- For each training example $(x, f(x))$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

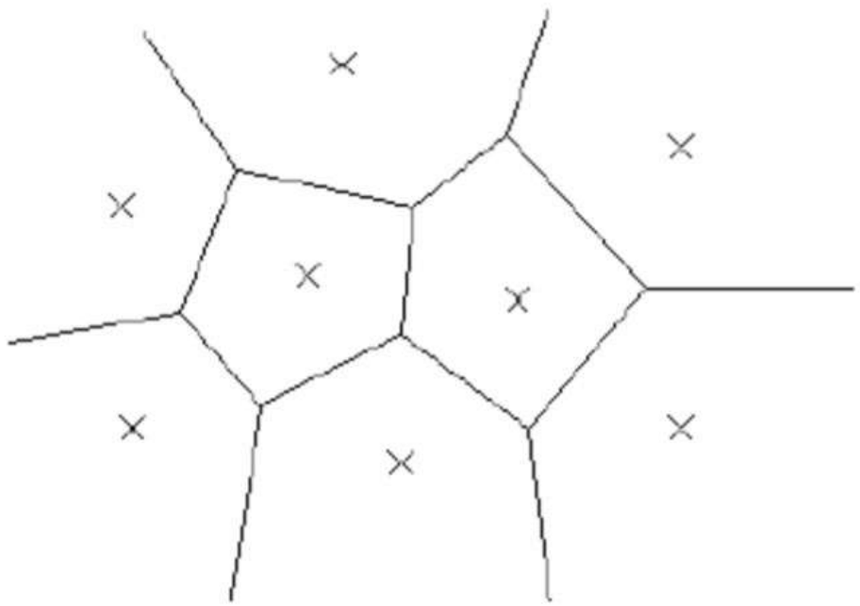
TABLE 8.1

The k-NEAREST NEIGHBOR algorithm for approximating a discrete-valued function $f : \Re^n \rightarrow V$.

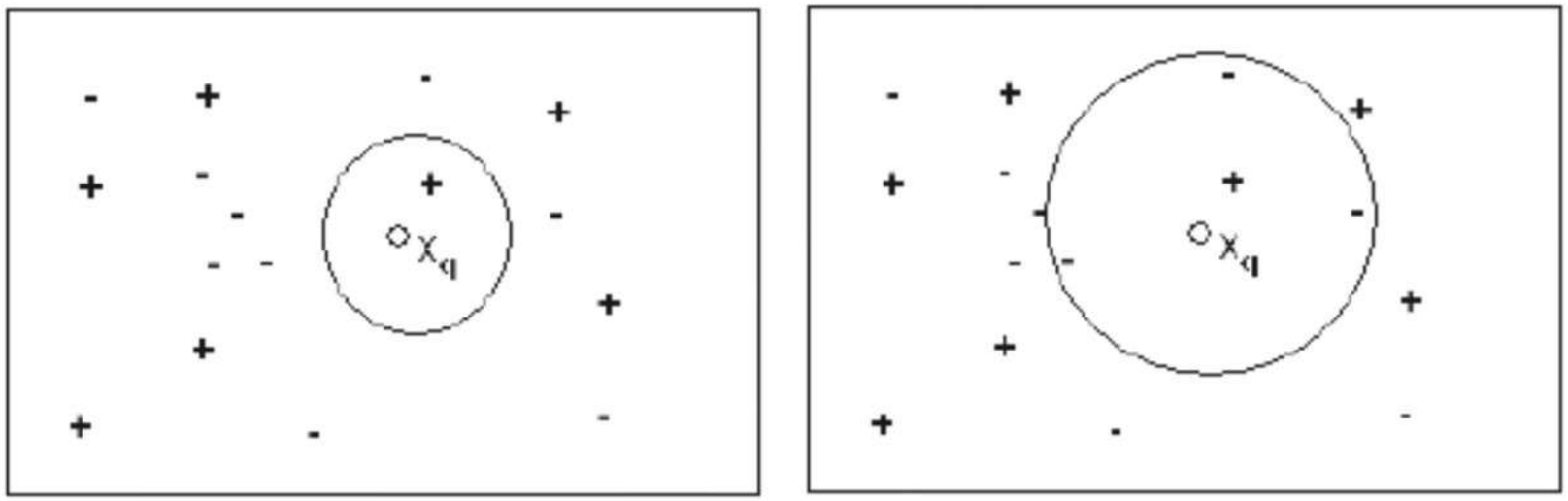
K-NN Hypothesis Space

MACHINE LEARNING

- **no explicit** hypothesis is formed
- decision surface is a combination of convex polyhedra surrounding each of the training examples
- for each training example, the polyhedron indicates the set of possible query points x_q whose classification is completely determined by this training example (**Voronoi diagram**)



K-nearest neighbor learning



- e.g. instances are points in a two-dimensional space where the target function is boolean-valued
  - 1-nearest neighbor: x_q is classified positive
  - 4-nearest neighbor: x_q is classified negative

Distance Weighted Nearest Neighbor

- contribution of each of the k nearest neighbors is weighted accorded to their distance to x_q
  - **discrete-valued target functions**

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{argmax} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$$

where $w_i \equiv \frac{1}{d(x_q, x_i)^2}$ and $\hat{f}(x_q) = f(x_i)$ if $x_q = x_i$

  - **continuous-valued target function:**

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

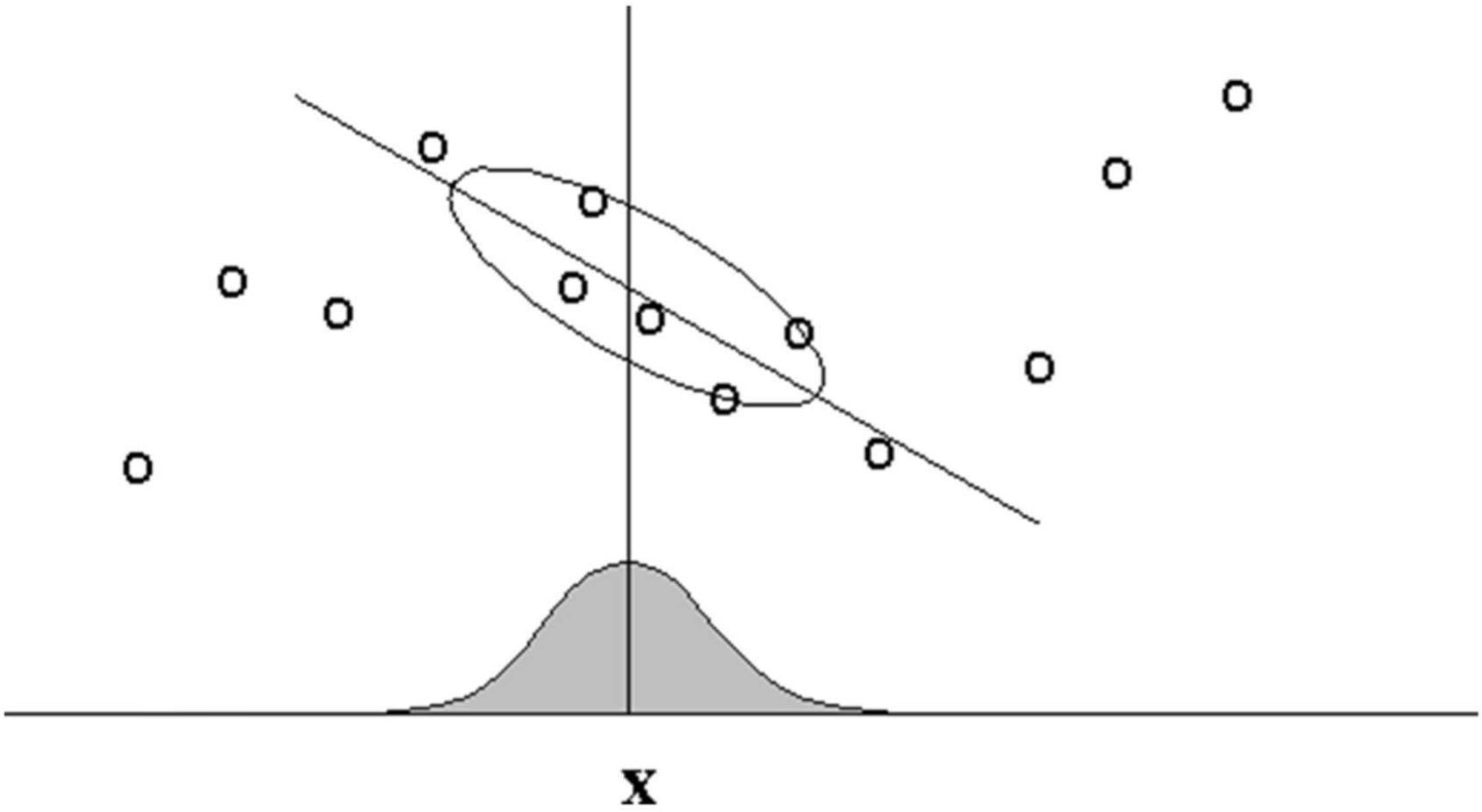
Remarks on K-NN

- highly effective inductive inference method for many practical problems provided a sufficiently large set of training examples
- robust to noisy data
- weighted average smoothes out the impact of isolated noisy training examples
- **inductive bias of k-nearest neighbors**
  - assumption that the classification of x_q will be similar to the classification of other instances that are nearby in the Euclidean Distance
- **curse of dimensionality**
  - distance is based on all attributes
  - in contrast to decision trees and inductive logic programming
  - solutions to this problem
    - attributes can be weighted differently
    - eliminate least relevant attributes from instance space

Locally Weighted Regression

- a note on terminology:
  - *Regression* means approximating a real-valued target function
  - *Residual* is the error $\hat{f}(x) - f(x)$ 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 $w_i = K(d(x_i, x_q))$

- nearest neighbor approaches can be thought of as approximating the target function at the single query point x_q
- locally weighted regression is a generalization to this approach, because it constructs an explicit approximation of f over a local region surrounding x_q


x

Locally weighted linear regression

MACHINE LEARNING

- target function is approximated using a **linear function**

$$\hat{f}(x) = w_0 + w_1 a_1(x) + ... + w_n a_n(x)$$

- methods like **gradient descent** can be used to calculate the coefficients $w_0, w_1, ..., w_n$ to minimize the error in fitting such linear functions

- ANNs require a global approximation to the target function
- here, just a local approximation is needed

$\Rightarrow$ the error function has to be redefined

Locally weighted linear regression

MACHINE LEARNING

- possibilities to redefine the error criterion E
  1. Minimize the squared error over just the k nearest neighbors

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest neigbors}} (f(x) - \hat{f}(x))^2$$

  2. Minimize the squared error over the entire set D, while weighting the error of each training example by some decreasing function K of its distance from x_q

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \cdot K(d(x_q, x))$$

  3. Combine 1 and 2

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest neighbors}} (f(x) - \hat{f}(x))^2 \cdot K(d(x_q, x))$$

Locally weighted linear regression

MACHINE LEARNING

- choice of the error criterion
  - E_2 is the most esthetically criterion, because it allows every training example to have impact on the classification of x_q
  - however, computational effort grows with the number of training examples
  - E_3 is a good approximation to E_2 with constant effort

$$\Delta w_j = \eta \sum_{x \in k \text{ nearest neighbors}} K(d(x_q, x))(f(x) - \hat{f}(x))a_j$$

- Remarks on locally weighted linear regression:
  - in most cases, constant, linear or quadratic functions are used
  - costs for fitting more complex functions are prohibitively high
  - simple approximations are good enough over a sufficiently small subregion of X

Algorithm

MACHINE LEARNING

1. Read the Given data Sample to **X** and the curve (linear or non linear) to **Y**
2. Set the value for Smoothening parameter or Free parameter say **τ**
3. Set the bias /Point of interest set **X0** which is a subset of **X**
4. Determine the weight matrix using :

$$w(x, x_o) = e^{-\frac{(x-x_o)^2}{2\tau^2}}$$

5. Determine the value of model term parameter β using :

$$\hat{\beta}(x_o) = (X^T W X)^{-1} X^T W y$$

6. Prediction = $x_0 * \beta$

Radial basis function

MACHINE LEARNING

One of the approach to function approximation

that is closely related to

distance-weighted regression and

artificial neural networks

is learning with radial basis functions

Radial basis function (for regression)

MACHINE LEARNING

- closely related to distance-weighted regression and to ANNs
- learned hypotheses have the form

$$\hat{f}(x) = w_0 + \sum_{u=1}^{k} w_u \cdot K_u(d(x_u, x))$$

where

- each x_u is an instance from X and
- $K_u(d(x_u, x))$ decreases as $d(x_u, x)$ increases and
- k is a user-provided constant

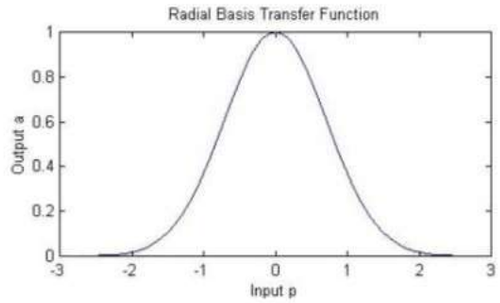
- though $\hat{f}(x)$ is a global approximation to $f(x)$, the contribution of each of the K_u terms is localized to a region nearby the point x_u

Radial basis function

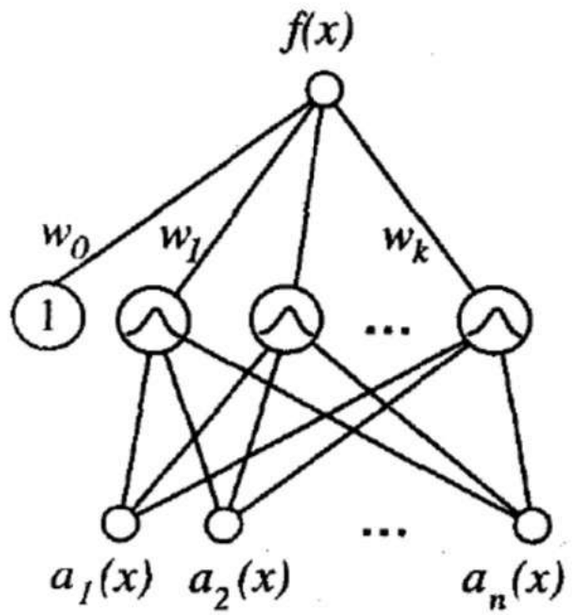
MACHINE LEARNING

- it is common to choose each function $K_u(d(x_u, x))$ to be a Gaussian function centered at x_u with some variance σ^2

$$K_u(d(x_u, x)) = e^{\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

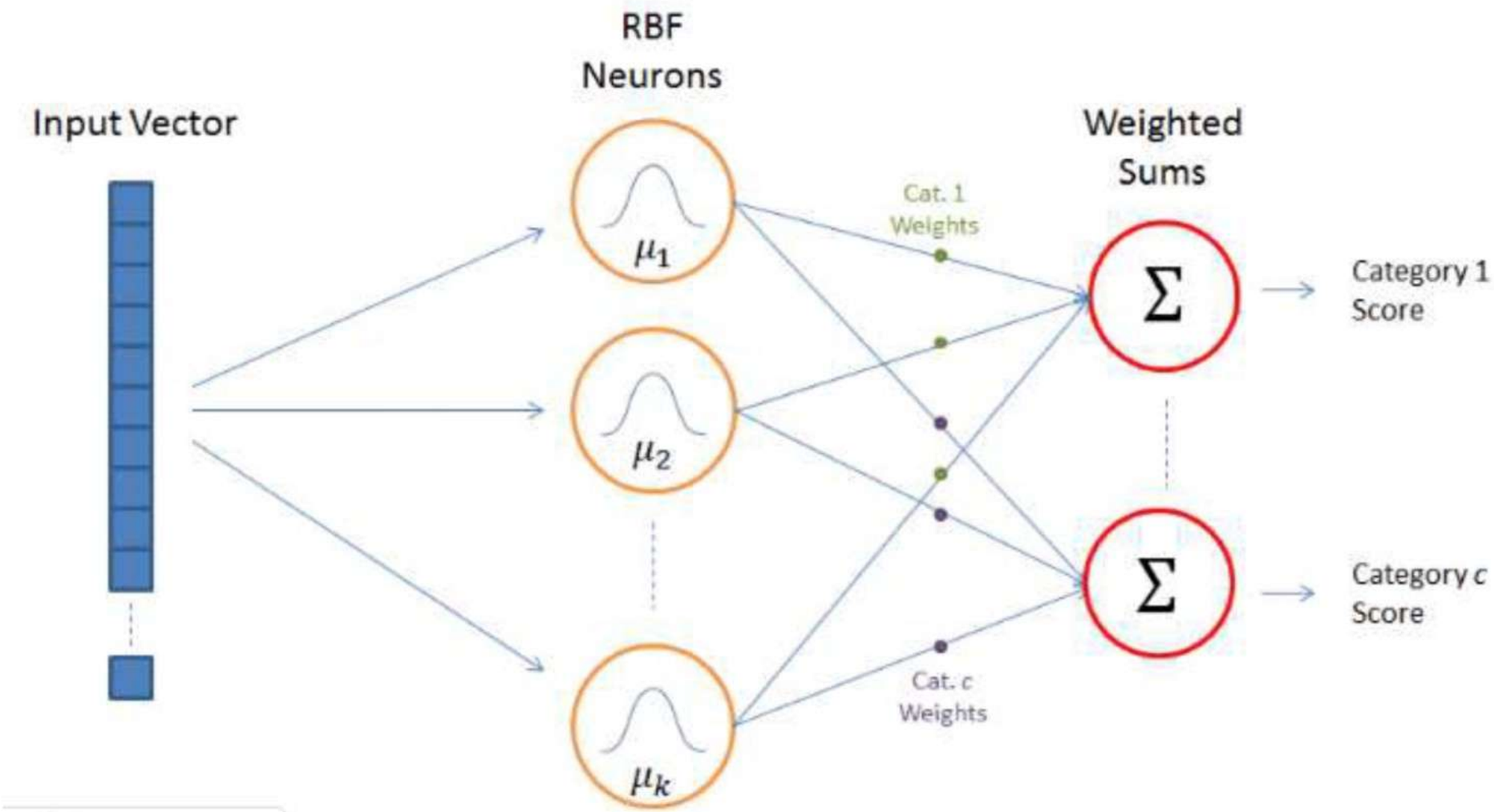


- the function of $\hat{f}(x)$ can be viewed as describing a two-layer network where the first layer of units computes the various $K_u(d(x_u, x))$ values and the second layer a linear combination of the results

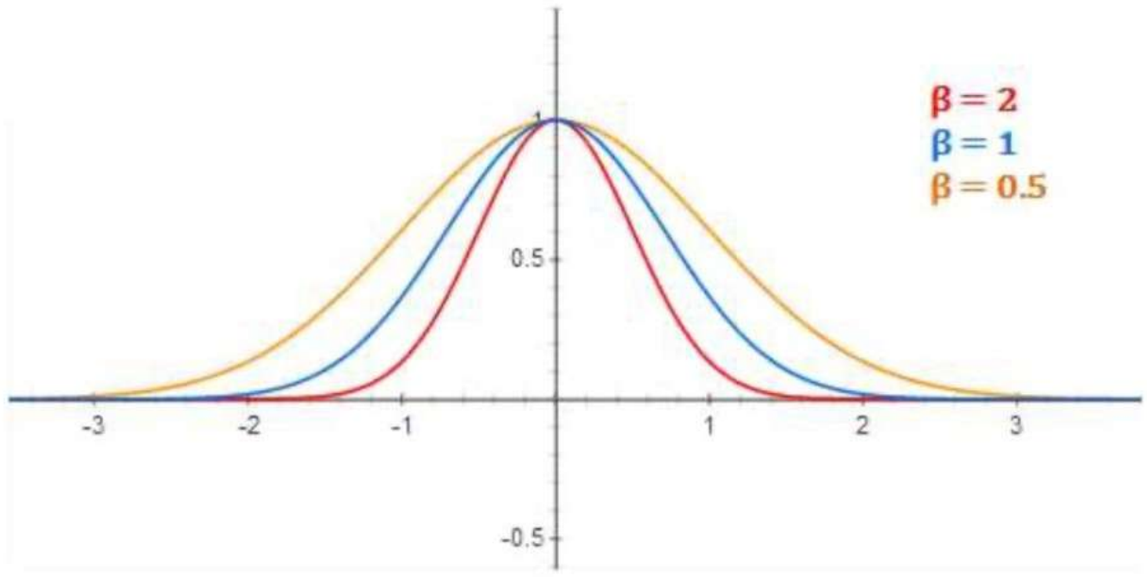


RBF NN

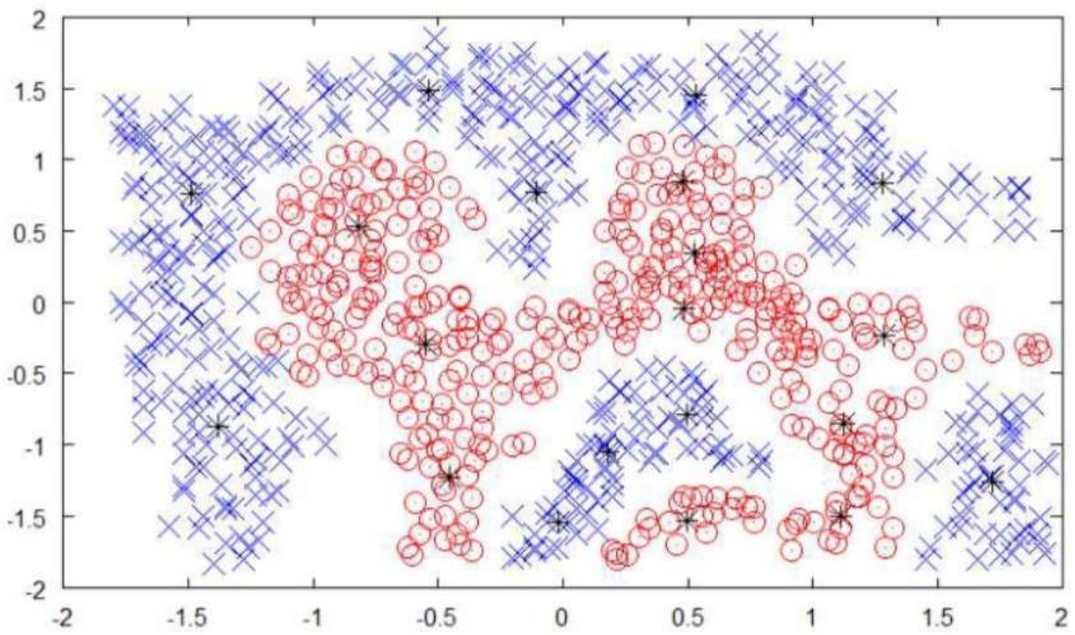
MACHINE LEARNING

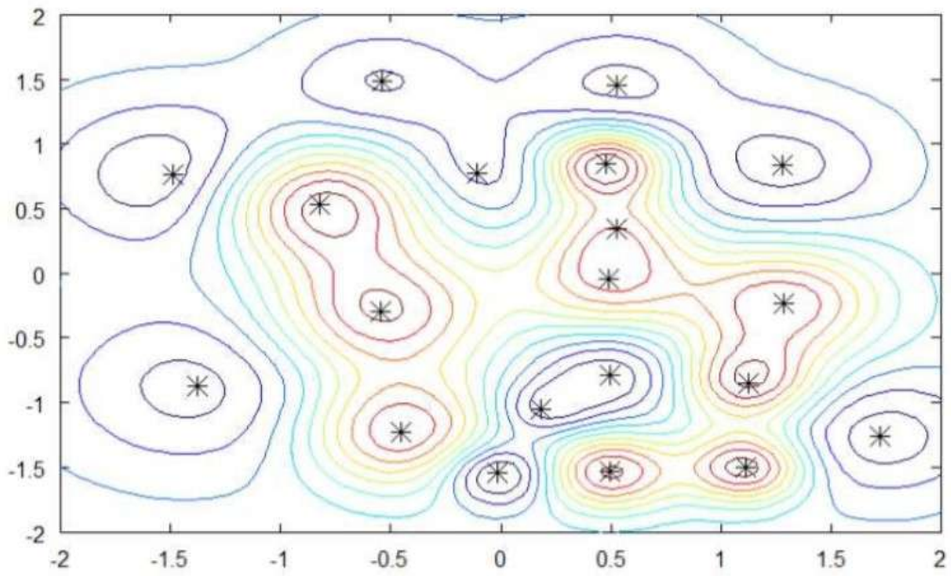


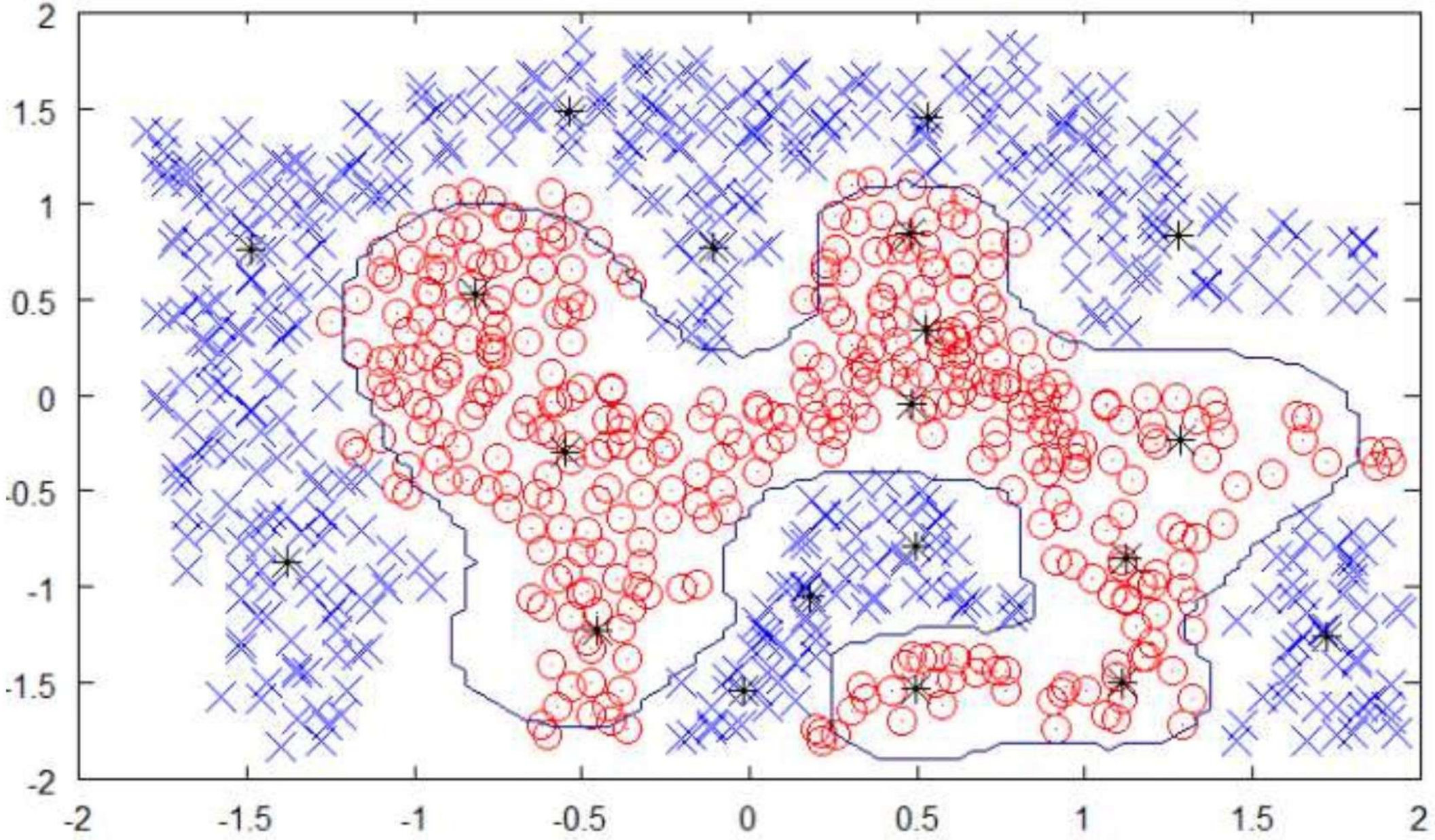
$$\varphi(x) = e^{-\beta\|x-\mu\|^2}$$



RBF Neuron activation for different values of beta






2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

Case Based Reasoning

MACHINE LEARNING

- Instance-based methods such as k-NN, locally weighted regression share three key properties.
  1. They are lazy learning methods

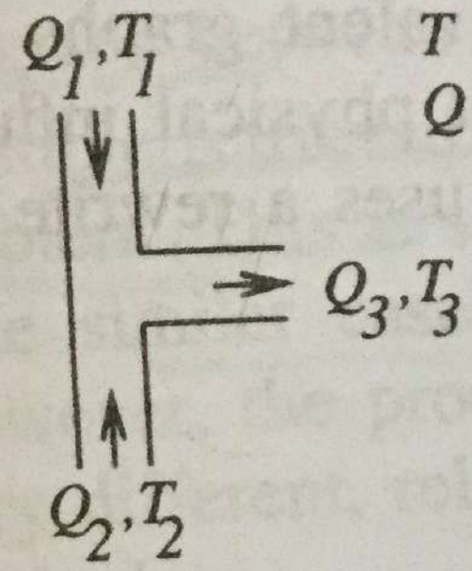
     They defer the decision of how to generalize beyond the training data until a new query instance is observed.
  2. They classify new query instances by analyzing similar instances while ignoring instances that are very different from the query.
  3. Third, they represent instances as real-valued points in an n-dimensional Euclidean space.
- Case-based reasoning (CBR) is a learning paradigm based on the first two of these principles, but not the third.

Case Based Reasoning

- In CBR, instances are typically represented using more rich symbolic descriptions, and the methods used to retrieve similar instances are correspondingly more elaborate.
  - CBR has been applied to problems such as conceptual design of mechanical devices based on a stored library of previous designs (Sycara et al. 1992),
  - reasoning about new legal cases based on previous rulings (Ashley 1990),
  - solving planning and scheduling problems by reusing and combining portions of previous solutions to similar problems (Veloso 1992).

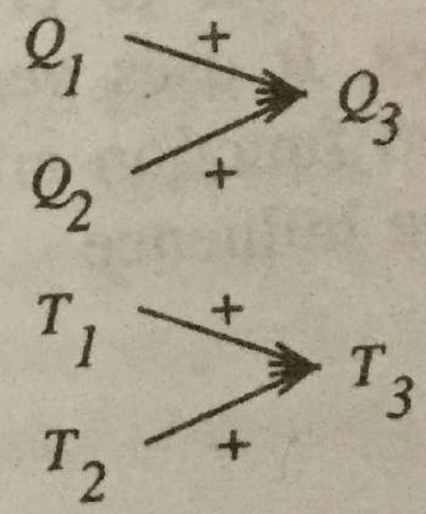
A stored case: T–junction pipe

Structure:



T = temperature
Q = waterflow

Function:

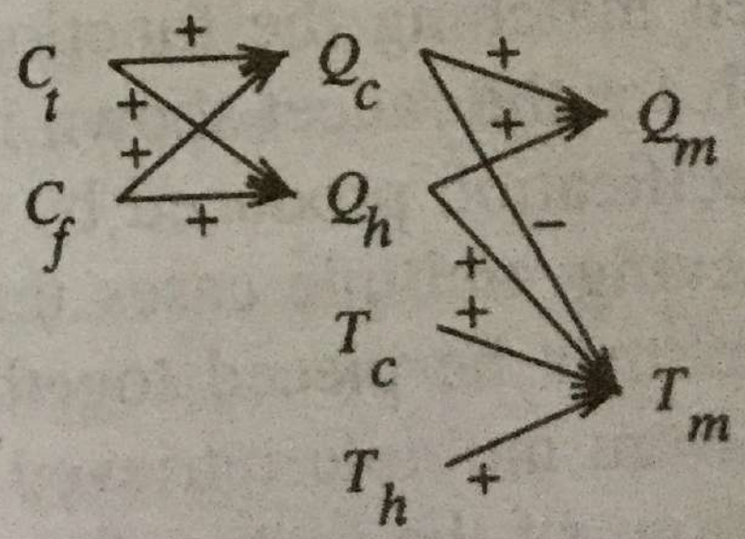


A problem specification: Water faucet

Structure:

?

Function:



$$A \xrightarrow{+} B$$

$$A \xrightarrow{+} x \xrightarrow{+} B$$

Case Study

MACHINE LEARNING

- The CADET system (Sycara et al. 1992)
  - 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.

- Complete Case study - Self study

Summary

MACHINE LEARNING

- instance-based learning simply stores examples and postpones generalization until a new instance is encountered
- able to learn discrete- and continuous-valued conepts
- noise in the data is allowed (smoothed out by weighting distances)
- **Inductive Bias of k-nearest neighbors**: classification of an instance is similar to the classification of other instances nearby in the Euclidean Distance
- Locally Weighted Regression forms a local approximation of the target function

Module 5

Chapter 13: Reinforcement Learning

1. **Introduction**
2. The Learning Task
3. Q Learning
4. Summary

Introduction

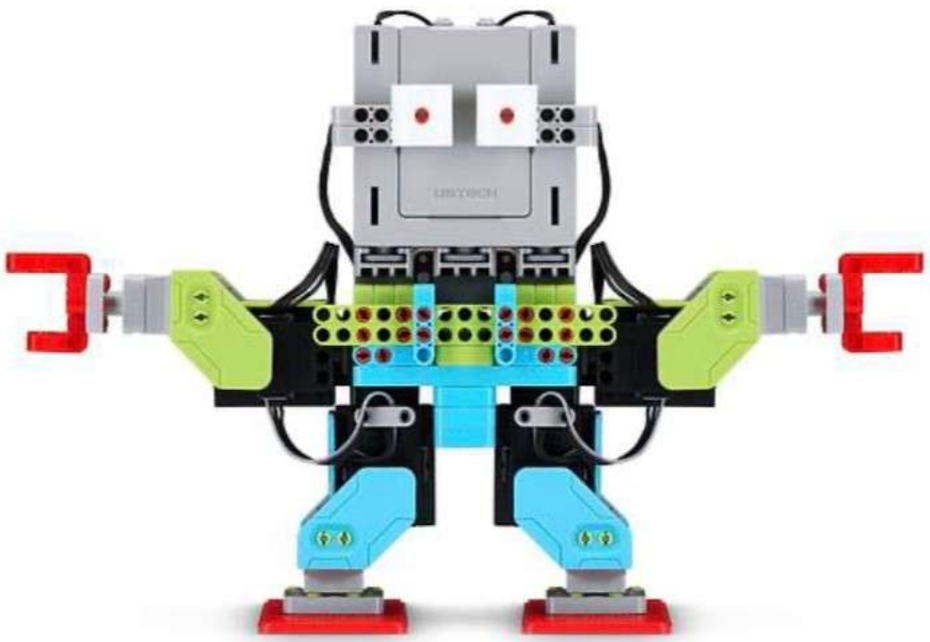
MACHINE LEARNING

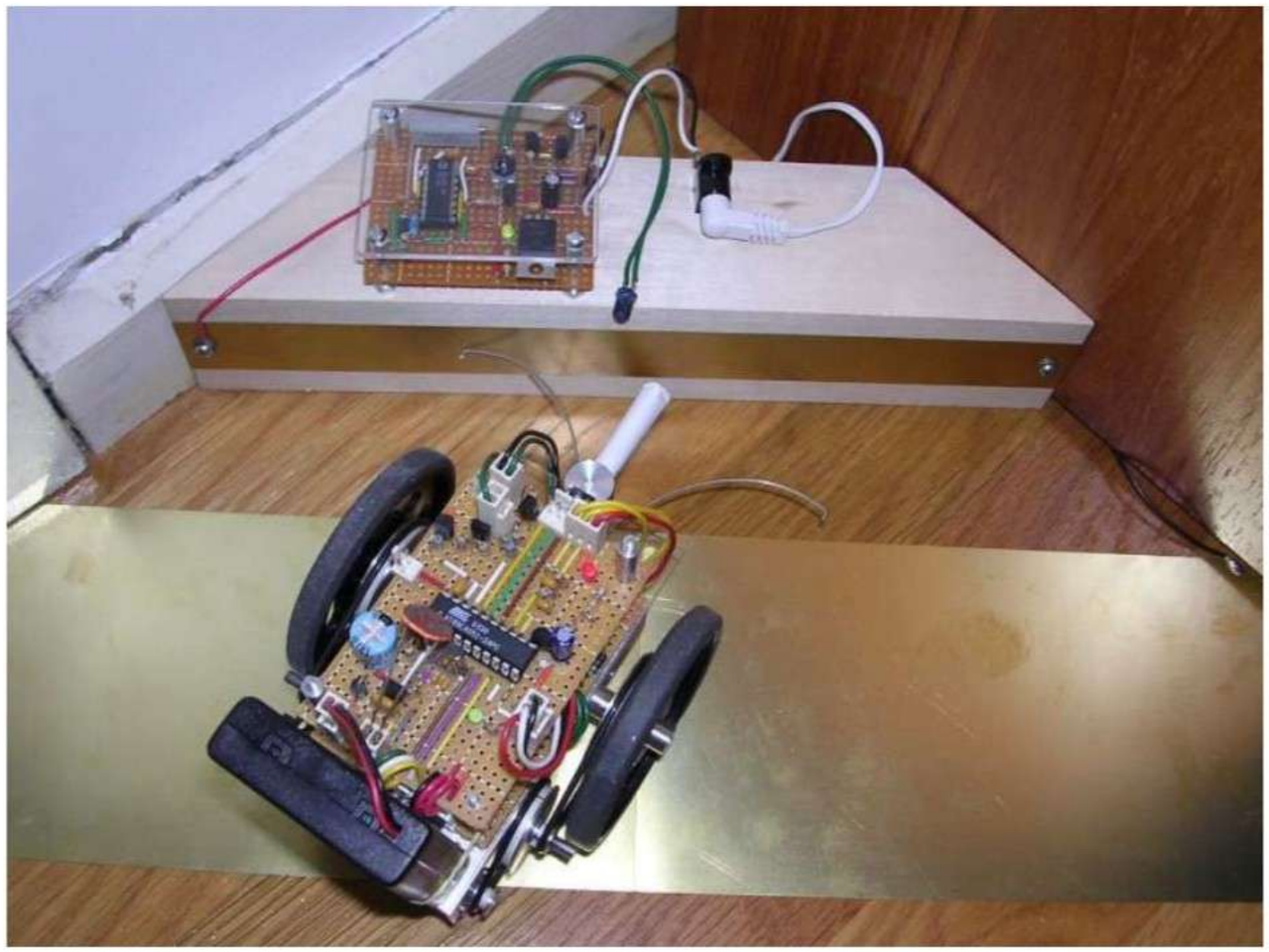
- Reinforcement learning addresses the question of
  - how an autonomous agent that senses and
  - acts in its environment
  - can learn to choose optimal actions to achieve its goals.

- Applications
  - learning to control a mobile robot
  - learning to optimize operations in factories
  - learning to play board games.

Introduction

MACHINE LEARNING





Introduction

MACHINE LEARNING

- Consider building a learning robot called as agent.
- It has
  - a set of sensors to observe the state of its environment

    Ex: Camera, Sonar
  - a set of actions it can perform to alter this state

    Ex: “Move forward”, “Turn Right”
- Its task is to learn a control strategy, or policy, for choosing actions that achieve its goals.
- For example, the robot may have a goal of docking onto its battery charger whenever its battery level is low.

Introduction

- The goals of the agent can be defined by a reward function
- Reward function assigns a numerical value - an immediate payoff -to each distinct action the agent may take from each distinct state.
- For example, the goal of docking to the battery charger can be captured by
  - assigning a positive reward (e.g., +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.

Introduction

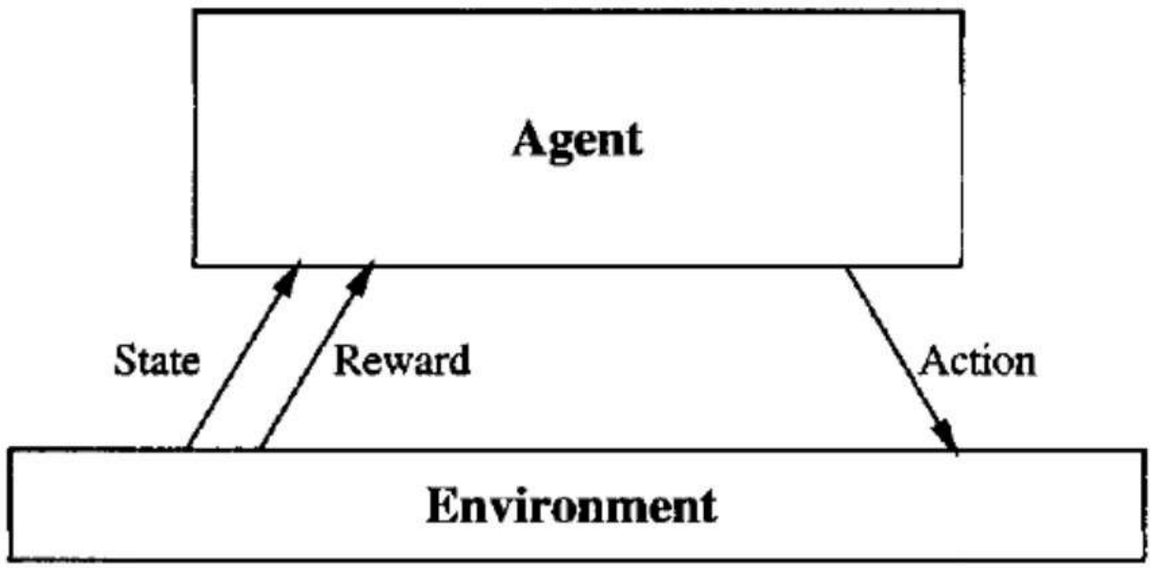
MACHINE LEARNING

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

Robot learning



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

Goal: Learn to choose actions that maximize

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

FIGURE 13.1
An agent interacting with its 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 a_t in some state s_t the agent receives a real-valued reward r_t that indicates the immediate value of this state-action transition. This produces a sequence of states s_i, actions a_i, and immediate rewards r_i as shown in the figure. The agent's task is to learn a control policy, $\pi : S \rightarrow A$, that maximizes the expected sum of these rewards, with future rewards discounted exponentially by their delay.

Introduction

MACHINE LEARNING

- **considered settings:**
  - deterministic or nondeterministic outcomes
  - prior backgound knowledge available or not

- **similarity to function approximation:**
  - approximating the function $\pi : S \rightarrow A$
    where S is the set of states and A the set of actions

- **differences to function approximation:**
  - Delayed reward: training information is not available in the form $< s, \pi(s) >$. Instead the trainer provides only a sequence of immediate reward values.
  - Temporal credit assignment: determining which actions in the sequence are to be credited with producing the eventual reward

Introduction

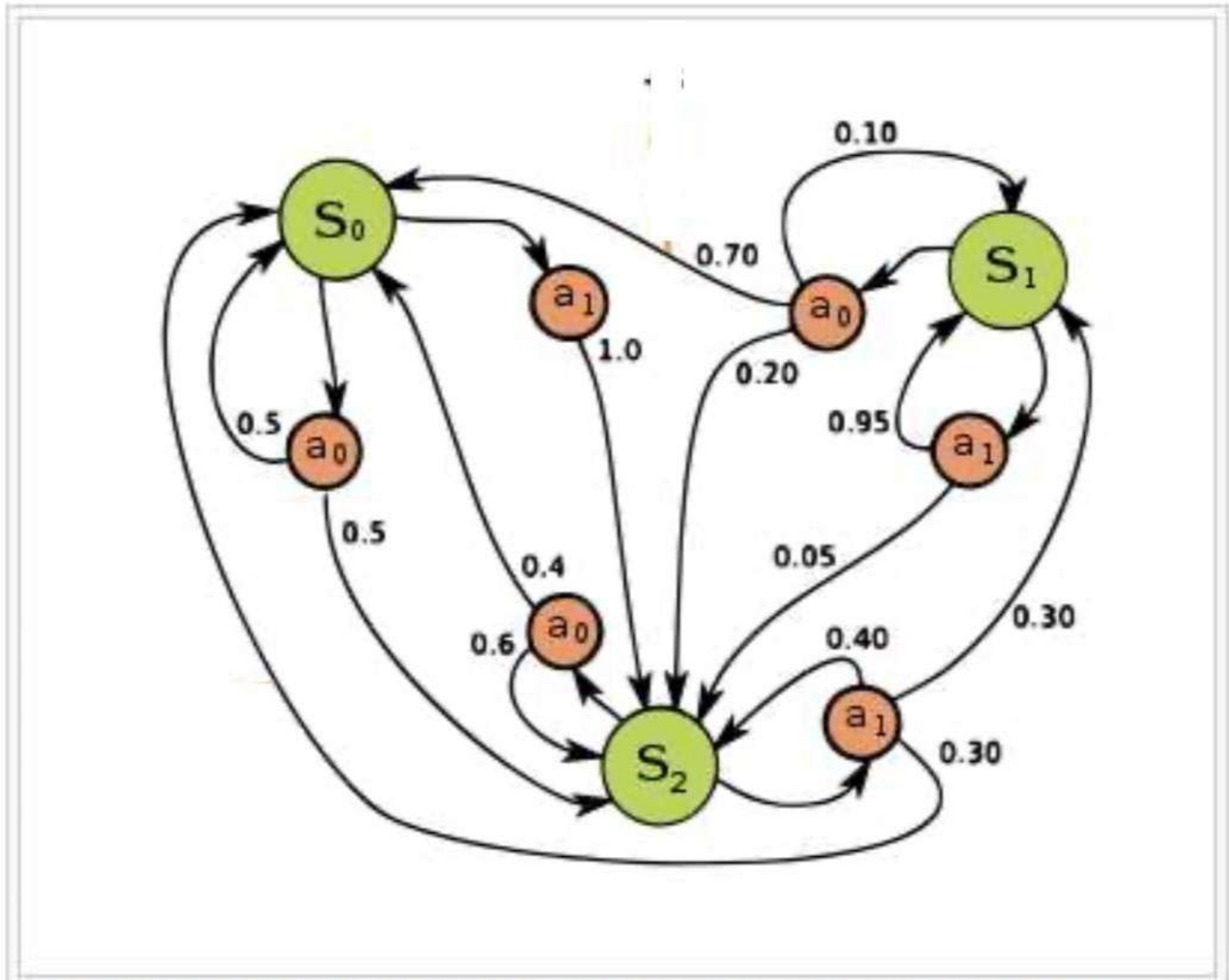
MACHINE LEARNING

- **differences to function approximation (cont.):**
  - exploration: distribution of training examples is influenced by the chosen action sequence
    - which is the most effective exploration strategy?
    - trade-off between exploration of unknown states and exploitation of already known states
  - partially observable states: sensors only provide partial information of the current state (e.g. forward-pointing camera, dirty lenses)
  - life-long learning: function approximation often is an isolated task, while robot learning requires to learn several related tasks within the same environment

The Learning Task

MACHINE LEARNING

- based on Markov Decision Processes (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** s_t, chooses a **current action** a_t and performs it
  - the environment responds by returning a **reward** $r_t = r(s_t, a_t)$ and by producing the **successor state** $s_{t+1} = \delta(s_t, a_t)$
  - the functions r and δ are part of the environment and not neccessarily known to the agent
  - in an MDP, the functions $r(s_t, a_t)$ and $\delta(s_t, a_t)$ depend only on the current state and action



Example of a simple MDP with three states (green circles) and two actions (orange circles), with two rewards (orange arrows).

Source: Wikipedia

The Learning Task

MACHINE LEARNING

- the task is to learn a policy $\pi : S \rightarrow A$
- one approach to specify which policy π the agent should learn is to require the policy that produces the greatest possible cumulative reward over time (**discounted cumulative reward**)

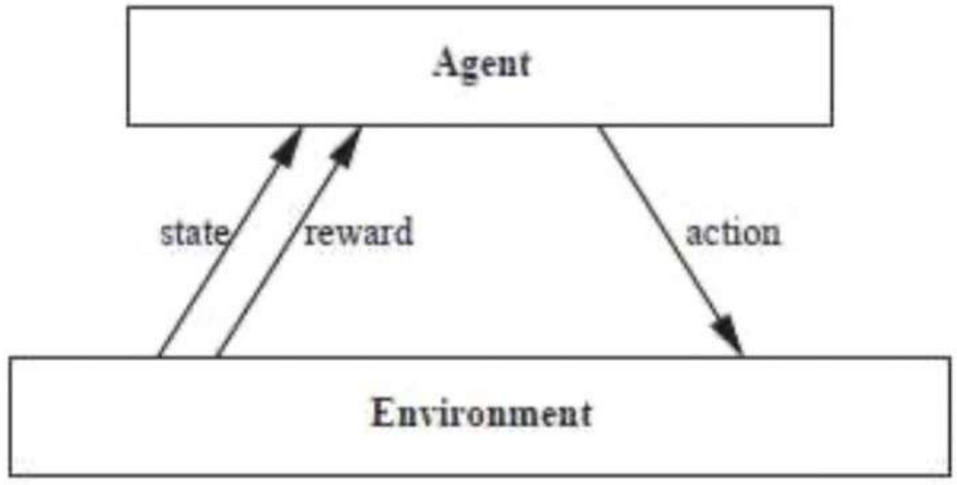
$$V^{\pi}(s_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+1}$$

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

where $V^{\pi}(s_t)$ is the cumulative value achieved by following an arbitrary policy π from an arbitrary initial state s_t

r_{t+i} is generated by repeatedly using the policy π and γ $(0 \leq \gamma < 1)$ is a constant that determines the relative value of delayed versus immediate rewards

The Learning Task



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

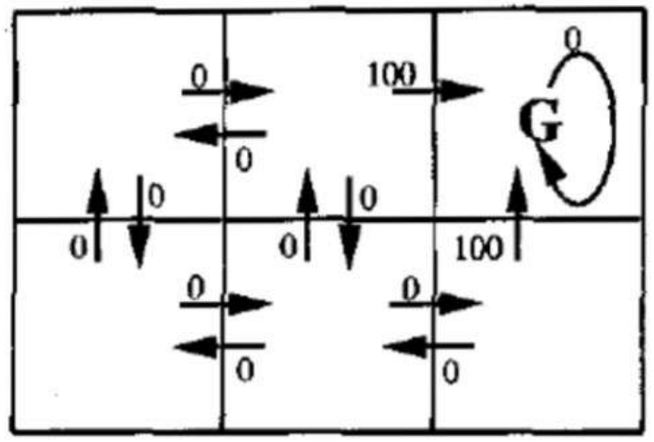
Goal: Learn to choose actions that maximize

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

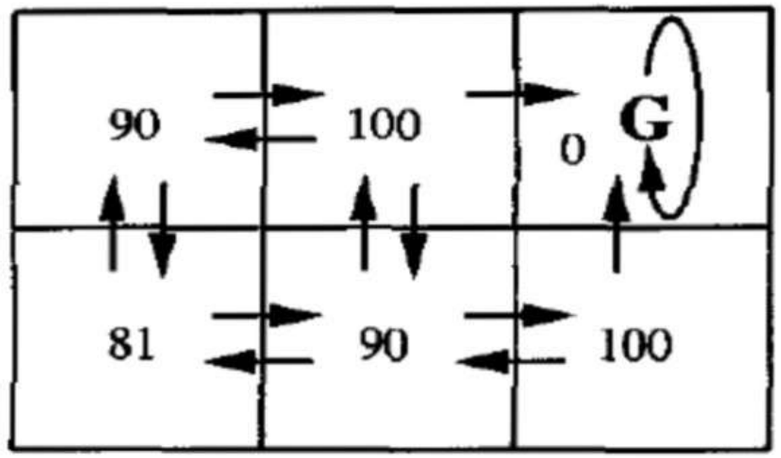
- hence, the agent's learning task can be formulated as

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

Illustrative Example



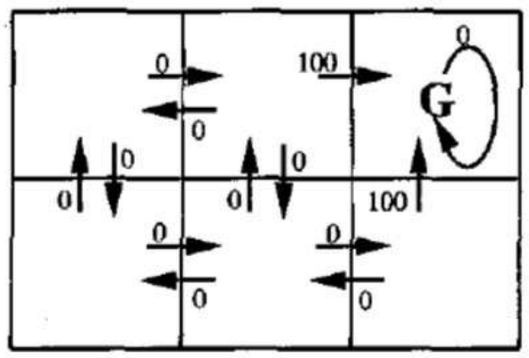
$r(s, a)$ (immediate reward) values



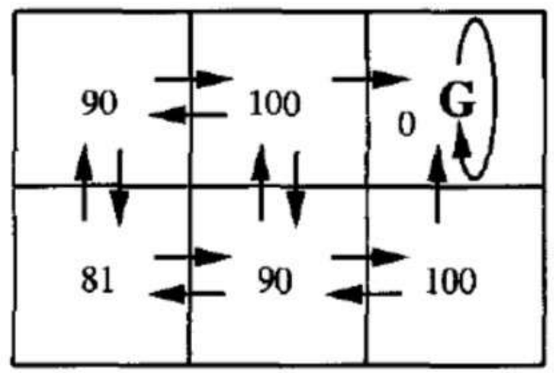
$V^*(s)$ values

- the left diagramm depicts a simple grid-world environment
  - squares $\approx$ states, locations
  - arrows $\approx$ possible transitions (with annotated $r(s, a)$)
  - $G \approx$ goal state (absorbing state)
- $\gamma = 0.9$
- once states, actions and rewards are defined and γ is chosen, the optimal policy π^* with its value function $V^*(s)$ can be determined

Illustrative Example



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



$V^*(s)$ values

- the right diagram shows the values of V^* for each state
- e.g. consider the bottom-right state
  - $V^* = 100$, because π^* selects the "move up" action that receives a reward of 100
  - thereafter, the agent will stay G and receive no further awards
  - $V^* = 100 + \gamma \cdot 0 + \gamma^2 \cdot 0 + ... = 100$
- e.g. consider the bottom-center state
  - $V^* = 90$, because π^* selects the "move right" and "move up" actions
  - $V^* = 0 + \gamma \cdot 100 + \gamma^2 \cdot 0 + ... = 90$
- recall that V^* is defined to be the sum of discounted future awards over **infinite** future

Module-5

9 a. Write short notes on the following:
   (i) Estimating Hypothesis accuracy.
   (ii) Binomial distribution. **(08 Ma**

b. Discuss the method of comparing two algorithms. Justify with paired to tests method. **(08 Ma**

OR

10 a. Discuss the K-nearest neighbor language. **(04 Ma**

b. Discuss locally weighted Regression. **(04 Ma**

c. Discuss the learning tasks and Q learning in the context of reinforcement learning. **(08 Ma**

* * * * *

Q Learning

MACHINE LEARNING

- it is easier to learn a numerical evaluation function than implement the optimal policy in terms of the evaluation function
- **question:** What evaluation function should the agent attempt to learn?
- one obvious choice is V^*
- the agent should prefer s_1 to s_2 whenever $V^*(s_1) > V^*(s_2)$
- **problem:** the agent has to chose among actions, not among states

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

the optimal action in state s is the action a that maximizes the sum of the immediate reward $r(s,a)$ plus the value of V^* of the immediate successor, discounted by γ

Q Learning

- thus, 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

- in many problems, it is impossible to predict in advance the exact outcome of applying an arbitrary action to an arbitrary state

- the Q function provides a solution to this problem
  - $Q(s, a)$ indicates the maximum discounted reward that can be achieved starting from s and applying action a first

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

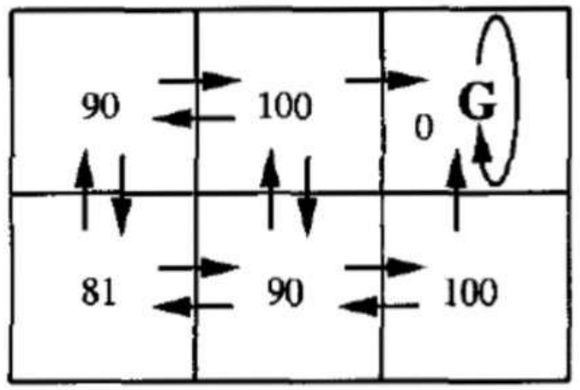
$$\Rightarrow \pi^*(s) = \underset{a}{argmax} Q(s, a)$$

Q Learning

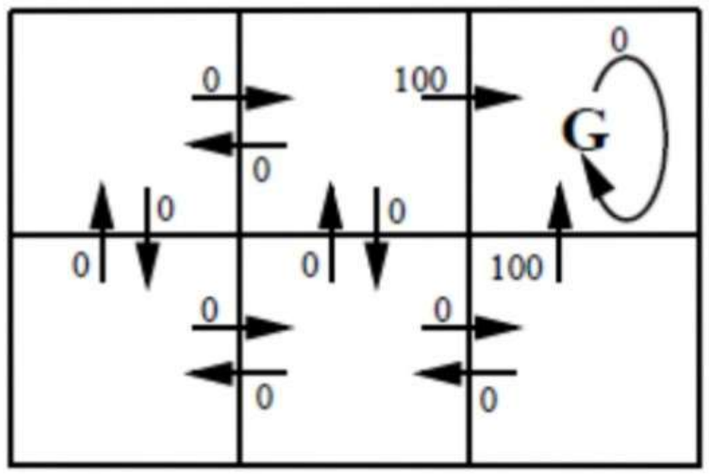
MACHINE LEARNING

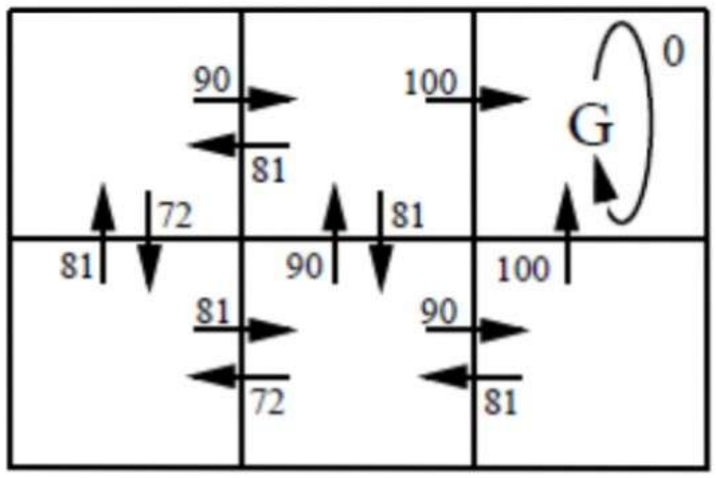
- hence, learning the Q function corresponds to learning the optimal policy π^*
- if the agent learns Q instead of V^*, it will be able to select optimal actions even when it has *no knowledge of* r and δ
- it only needs to consider each available action a in its current state s and chose the action that maximizes $Q(s, a)$
- the value of $Q(s, a)$ for the current state and action summarizes in one value all information needed to determine the discounted cumulative reward that will be gained in the future if a is selected in s

Q learning



$V^*(s)$ values





- the right diagramm shows the corresponding Q values
- the Q value for each state-action transition equals the r value for this transition plus the V^* value discounted by γ

Q Learning Algorithm

MACHINE LEARNING

- **key idea:** iterative approximation
- relationship between Q and V^*

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

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

- this recursive definition is the basis for algorithms that use iterative approximation
- the learner's estimate $\hat{Q}(s, a)$ is represented by a large table with a separate entry for each state-action pair

Q Learning Algorithm

For each s, a initialize the table entry $\hat{Q}(s, a)$ to zero
Oberserve the current state s

Do forever:

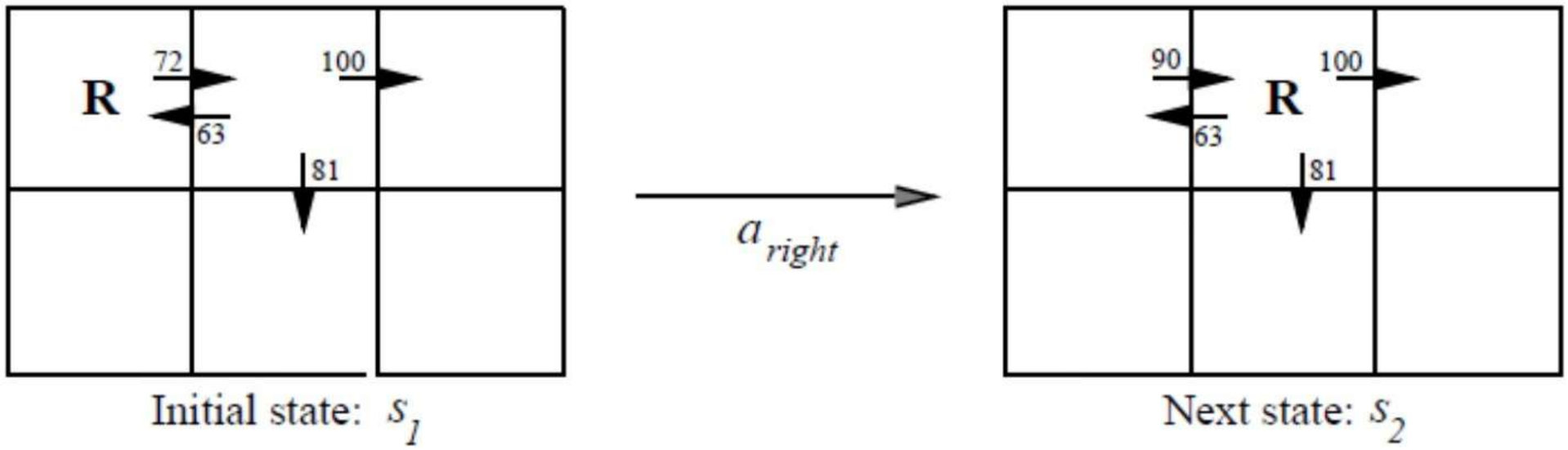
- Select an action a and execute it
- Receive immediate reward r
- Observe new state s'
- Update each table entry for $\hat{Q}(s, a)$ as follows

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

- $s \leftarrow s'$

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

Illustrative Example



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

- each time the agent moves, Q Learning propagates $\hat{Q}$ estimates *backwards* from the new state to the old

Experimentation Stages

- algorithm does not specify how actions are chosen by the agent
- **obvious strategy:** select action a that maximizes $\hat{Q}(s,a)$
  - risk of overcommiting to actions with high $\hat{Q}$ values during earlier trainings
  - exploration of yet unknown actions is neglected
- **alternative:** probabilistic selection

$$P(a_i|s) = \frac{k^{\hat{S}(s,a_i)}}{\sum_j k^{\hat{Q}(s,a_i)}}$$

k indicates how strongly the selection favors actions with high $\hat{Q}$ values

k large $\Rightarrow$ exploitation strategy

k small $\Rightarrow$ exploration strategy

Generalizing from Examples

- so far, the target function is represented as an explicit lookup table
- the algorithm performs a kind of rote learning and makes no attempt to estimate the Q value for yet unseen state-action pairs

$\Rightarrow$ unrealistic assumption in large or infinite spaces or when execution costs are very high

- incorporation of function approximation algorithms such as BACKPROPAGATION
  - table is replaced by a neural network using each $\hat{Q}(s, a)$ update as training example (s and a are inputs, $\hat{Q}$ the output)
  - a neural network for each action a

Relationship to Dynamic Programming

MACHINE LEARNING

- Q Learning is closely related to dynamic programming approaches that solve Markov Decision Processes

- **dynamic programming**
  - assumption that $\delta(s, a)$ and $r(s, a)$ are known
  - focus on how to compute the optimal policy
  - mental model can be explored (no direct interaction with environment)

  $\Rightarrow$ *offline system*

- **Q Learning**
  - assumption that $\delta(s, a)$ and $r(s, a)$ are not known
  - direct interaction inevitable

  $\Rightarrow$ *online system*

Relationship to Dynamic Programming

MACHINE LEARNING

- relationship is appent by considering the Bellman's equation, which forms the foundation for many dynamic programming approaches solving Markov Decision Processes

$$(\forall s \in S) V^*(s) = E[r(s, \pi(s)) + \gamma V^*(\delta(s, \pi(s)))]$$

Summary

- Reinforcement learning
  - Learning control strategies for autonomous agents.
  - It assumes that training information is available in the form of a real-valued reward signal given for each state-action transition.
  - The goal of the agent is to learn an action policy that maximizes the total reward it will receive from any starting state.

Summary

MACHINE LEARNING

- The reinforcement learning algorithms addressed in this chapter fit a problem setting known as a Markov decision process.
- In Markov decision processes, the outcome of applying any action to any state depends only on this action and state (and not on preceding actions or states).
- Markov decision processes cover a wide range of problems including many robot control, factory automation, and scheduling problems.

Summary

MACHINE LEARNING

- The reinforcement learning algorithms addressed in this chapter fit a problem setting known as a Markov decision process.
- In Markov decision processes, the outcome of applying any action to any state depends only on this action and state (and not on preceding actions or states).
- Markov decision processes cover a wide range of problems including many robot control, factory automation, and scheduling problems.

Summary

- Q learning is one form of reinforcement learning in which the agent learns an evaluation function over states and actions.

- Evaluation function Q(s, a) is defined as the
  - maximum expected, discounted, cumulative reward
  - the agent can achieve by applying action a to state s.

- Advantage - it can-be employed even when the learner has no prior knowledge of how its actions affect its environment.