MODULE - 5 - IA 3

3a)

1a) Explain K-Nearest Neighbour Algorithm.

Ans: * K - Nearest Neighbour is an INSTANCE - BASED METHOD based on LAZY LEARNING where KNN function is approximated locally and all computations are deferred until classification.

* It assumes all instances correspond to points in the n-dimensional space R^n. The nearest neighbours of an instance are defined in terms of the standard Euclidean distance.

* Let an arbitary instance x be described by a FEATURE VECTOR $(a_1(x), a_2(x), \ldots, a_n(x))$ where $a_r(x)$ denotes the value of the r^{th} attribute of instance x.

* Then the distance between two instances x_i & x_j is given by:

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

* In KNN, the target function may be either:
  - DISCRETE - VALUED
  - REAL - VALUED.

5b) Explain KNN algorithm for approximating a discrete-valued function $[f: R^n \to V]$ with pseudo code.

Camlin Page
Date / /

① KNN for APPROXIMATION OF A DISCRETE VALUED FN.

$$f: R^n \to V$$

→ TRAINING ALGORITHM:

* for each TRAINING EXAMPLE $\langle x, f(x) \rangle$:
  ⇒ 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 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 $a=b$]
$\delta(a,b) = 0$ [otherwise]

for 5b)

* The value $\hat{f}(x_q)$ returned by this algorithm as its estimate of $f(x_q)$ = the most common value of f among the k training examples nearest to x_q

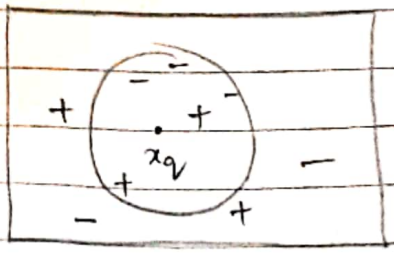
* If k=1, the 1-NN assigns to $\hat{f}(x_q)$ the value of $f(x_i)$

$$\hat{f}(x_q) = f(x_i)$$

where x_i is the training instance nearest to x_q

* For k = large value, the algorithm assigns the most common value among k-nearest training examples

Eg:



\+ POSITIVE TRAINING EXAMPLE
− NEGATIVE TRAINING EXAMPLE
1-NN (x_q) → Positive example,
5-NN (x_q) → Negative example.

② KNN for approximation of a REAL-VALUED TARGET FUNCTION $f : R^n \to R$

→ TRAINING ALGORITHM:

* For each TRAINING EXAMPLE $\langle x, f(x) \rangle$:
  ⇒ 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 k instances from training_examples that are nearest to x_q

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

1b) Describe LOCALLY WEIGHTED REGRESSION (LWR)

Ans: * The term LOCALLY WEIGHTED REGRESSION can be viewed as:

→ LOCAL: Function is approximated based on data near the query point

→ WEIGHTED: Contribution of each training example is weighted by its distance from query point.

→ REGRESSION: Approximation of a REAL-VALUED target function

* Given a new query instance x_q, the general approach is LWR is to construct an approx. $\hat{f}$ that fits the training examples in the neighbourhood, surrounding x_q.

* This approximation is used to calculate the value $\hat{f}(x_q)$ ⇒ estimated target value for query instance.

LOCALLY WEIGHTED REGRESSION ALGORITHM.

* Consider LWR in which TARGET FN f is approximated near x_q using:

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

where $a_i(x)$ denotes the value of i^{th} attribute of instance x.

* Derived methods are used to choose weights that minimize the squared error (using GRADIENT DESCENT)

$$E \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2$$

$$\Delta w_j = \eta \sum_{x \in D} (f(x) - \hat{f}(x)) a_j(x)$$

↳ constant learning rate

* Criteria for LOCAL APPROXIMATION:

① Minimize the squared error over just the k-NN:

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in \text{kNN of } x_q} (f(x) - \hat{f}(x))^2 \quad \cdots\cdots ①$$

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 x_q:

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

① + ②,

③ $$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in \text{kNN of } x_q} (f(x) - \hat{f}(x))^2 K(d(x_q, x)) \quad \cdots\cdots ③$$

Now,

⇒ GRADIENT DESCENT RULE will be:

$\underbrace{K(d(x_q, x))}_{\text{DISTANCE PENALTY}}$

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

error summed over only the k-nearest training examples

3a) What is Instance based learning? Explain K-NN algorithm

Ans: INSTANCE - BASED LEARNING (Memory-based learning)

* It is a family of learning algorithms that, compares new problem instances with instances seen in training which have been stored in memory, instead of performing explicit generalization.

* Instance-based approaches can construct a different approximation for each distinct query instance that must be classified.

* ..... in these algorithms consists of simply storing the presented data. When a new query instance is encountered, a set of similar related instances is retrieved from memory and used to classify a new query instance.

ADVANTAGES	DISADVANTAGES
* Training is very fast	* Cost of classification is high
* Learns complex Target fn.	* Requires large memory
* No information loss.	* Each query initiates a local model formation from scratch.

K-NN → Refer 1a)

6a) Explain CADET SYSTEM using Case Based Reasoning.

3b) Explain Case Based Reasoning with example.

Ans: CASE-BASED REASONING (CBR)

* CBR is a learning paradigm based on LAZY LEARNING methods and they classify new query instances by analyzing similar instances while ignoring instances that are very different from the query.

* In CBR, ~~represent~~ representation of 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 + SCHEDULING by reusing & combining portions of previous solutions to similar problems.

⇒ EXAMPLE : CADET SYSTEM

* The CADET system employs CBR to assist in the conceptual design of simple mechanical devices such as WATER FAUCETS.

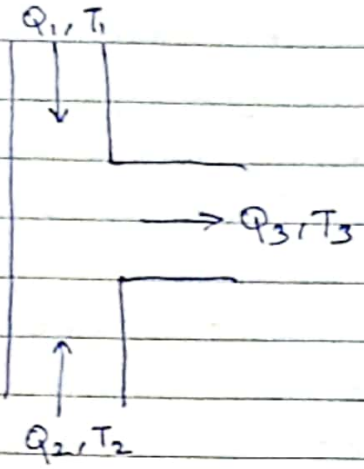
* It uses a library containing approx. 75 previous designs & design fragments to suggest conceptual designs to meet the specifications of new design problems.

* Each instance stored in memory is represented by describing both its structure + qualitative fn.

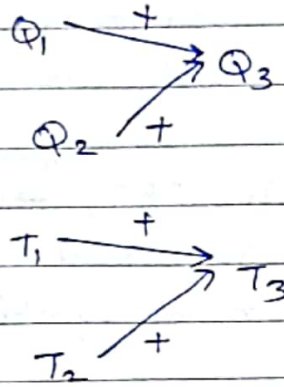
* New design problems are then presented by specifying the desired fn. & requesting the corresponding structure

I CASE 1: T-JUNCTION PIPE

STORED CASE (T → Temp., Q → water flow)



Function



Notations: $S_1 \xrightarrow{+} S_2$ | $S_3 \xrightarrow{-} S_4$

⇒ S_2 increases with S_1 | S_4 decreases

$T \leftrightarrow Q$ (Qualitative relationship)

Q/H values:

- Q_c → cold water flow into faucet
- Q_h → hot water flow into faucet
- Q_m → mixed flow out of faucet
- T_c → temp. of cold water
- T_h → temp. of hot water
- T_m → temp. of mixed water

CONTROL SIGNALS:

- C_t → control signal for TEMP. that is input to faucet
- C_f → control signal for water flow

CONTROLS / WATER FLOWS

* C_t & C_f influence Q_c & Q_h

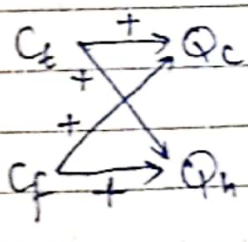
⇒ indirectly influence

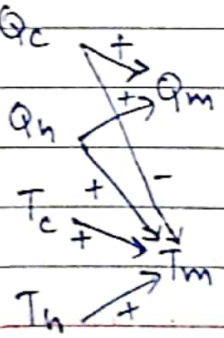
output flow → Q_m

temperature → T_m

II CASE 2: WATER FAUCET (Structure = ?)

Function:





* CADET searches its library for stored cases.

① Exact Match found

⇒ Indicates some STORED case implements DESIRED FN.

⇒ Return case as suggested solution to design problem

② Exact Match not found

CADET may find cases that match various SUBGRAPHS of the DESIRED FN. specification.

4a) Write short notes on RADIAL BASIS FUNCTION (RBF)

(8)

Example ⊛ **RADIAL BASIS FUNCTION (RBF) NETWORK**

* The function given eqn① can be viewed as describing a two layer network where the first layer of units computes the values of the various $K_u(d(x_u, x))$ and where the second layer computes a linear combination of these first-layer unit values

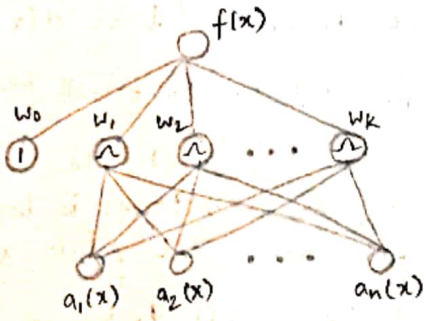
* Given a set of training examples of the target fn. RBF networks are typically trained in a two-stage process:

⇒ PROCESS ①: The number k of HIDDEN units, is determined and each HIDDEN UNIT u is defined by choosing the values of x_u & σ_u^2 that define its kernel function $K_u(d(x_u, x))$

⇒ PROCESS ②: The weights w, are trained to maximize the fit of the network to the training data, using the global error criterion given by:

$$E \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2$$

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



* Each hidden unit produces an activation determined by a GAUSSIAN FUNCTION centered at some instance x_u.
* ∴ Its activation will be close to zero unless the input x is near x_u
* The output unit produces a linear combination of hidden unit activations.
* Multiple output units can also be included ($f_1(x), f_2(x), f_3(x)..$)
* supports BACKPROPAGATION

Date / /

5a) What do you mean by REINFORCEMENT LEARNING? How reinforcement problem differs from other function approximation tasks?

Ans: → REINFORCEMENT LEARNING

* Reinforcement learning addresses the question of how an AUTONOMOUS AGENT that senses + acts in its ENVIRONMENT can learn to choose OPTIMAL ACTIONS to achieve its GOALS.

Eg: * A learning ROBOT / AGENT observes state of its environment with set of sensors & has set of actions to alter its state.

* It task is to perform sequences of actions, observe consequences and learn a CONTROL POLICY.

* CONTROL POLICY chooses actions that maximize the REWARD accumulated over time by the agent.

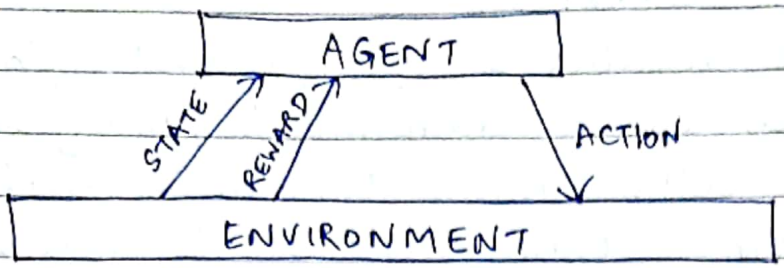
→ REINFORCEMENT LEARNING PROBLEM

① An AGENT interacts with its environment. It exists in an environment described by some set of possible states (S).

② AGENT performs a random 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 (immediate value of STATE-ACTION transition)

③ This produces a sequence of states s_i, actions a_i and immediate rewards r_i as shown in figure.

④ The agent task is to learn a CONTROL POLICY $\Pi : S \rightarrow A$, that maximizes the expected sum of rewards, discounted exponentially by DELAYS.



$$S_0 \xrightarrow[r_0]{a_0} S_1 \xrightarrow[r_1]{a_1} S_2 \xrightarrow[r_2]{a_2} \cdots$$

* CONTROL STRATEGY: MAX $(r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots)$ $(0 \leq \gamma < 1)$
  ↳ discount factor

→ REINFORCEMENT PROBLEM VS FN. APPROXIMATION TASKS

(1) DELAYED REWARD:

* TASK: Agent should learn target fn. Π that maps
  $S \rightarrow a = \Pi(S)$
  current state → optimal action
  [but information not available in $(s, \Pi(s))$ ✗]
* provided by only a sequence of immediate reward values
* Faces TEMPORAL CREDIT ASSIGNMENT problem
  (Determining which is the optimal sequence for eventual / overall rewards)

(2) EXPLORATION:

* In RL, the AGENT influences the distribution of training examples by the chosen action sequence.
* Does EXPERIMENTATION STRATEGY prove to be fruitful?
* Faces TRADE-OFF in choosing whether to favour EXPLORATION / EXPLOITATION of state-action transitions.

(3) PARTIALLY OBSERVABLE STATES:

* Practically, Agent SENSORS provide only partial information.
* It needs to ~~re~~consider previous observations + current sensor data to choose actions.

(4) LIFE-LONG LEARNING:

* Agent requires to learn several related tasks within same environment, using same sensors.
* This setting raises the possibility of using PREVIOUSLY OBTAINED EXP./KNOWLEDGE to reduce sample complexity when learning new tasks.

5. **Q-LEARNING**

* **Q-learning** is a basic form of Reinforcement Learning which uses Q-values (action-values) to iteratively improve the behavio. of the learning agent.

* **Q-VALUES / ACTION VALUES :** Q-values are defined for states and actions. $Q(S,A)$ is an estimation of how good is it to take the action A at the state S.

* **REWARDS + EPISODES :** * An agent over the course of its lifetime start state, makes a # transitions from its current state to a next state based on its choice of action and also the environment the agent is interacting in.

* At every step of transition, the agent from a state takes an action, observes a reward from the environment and then transits to another state.

* If at any point of time, the agent ends up in one of the terminating states that means there are no further transition possible ⇨ COMPLETION OF AN EPISODE.

* **TEMPORAL DIFFERENCE / TD-UPDATE :** The estimation of $Q(S,A)$ will be iteratively computed using TD-UPDATE RULE. The optimal action in state 's' is the action 'A' that maximizes the sum of the immediate REWARD $R(S,A)$ + the value V^* of the immediate successor state, discounted by γ.

$$\pi^*(s) = \underset{A}{\operatorname{argmax}} \left[R(s,A) + \gamma V^*(\delta(s,A)) \right] \quad \text{---} (1)$$

* **Q-FUNCTION ANALYSIS :** The value of Evaluation function $Q(S,A)$ is the reward received immediately upon executing action A from state S with TD-rule :

(2) $$Q(S,A) \equiv R(S,A) + \gamma V^*(\delta(S,A)) \Rightarrow$$ (3) from (1) $$\pi^*(s) \equiv \underset{A}{\operatorname{argmax}}\, Q(S,A)$$

⊛ Q - LEARNING ALGORITHM

→ The objective of Q-learning is to learn the optimal policy which can inform the agent that what actions should be taken for maximizing the reward under what circumstances.

$$\pi : S \rightarrow A$$

→ 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 achieved by:

⇒ ITERATIVE APPROXIMATION ; $\boxed{V^*(S) = \max_{A'} Q(S,A')}$

⇒ $\boxed{Q(S,A) = R(S,A) + \gamma \max_{A'} Q(\delta(S,A), A')}$ From ②

↳ discounting factor $(0 \leq \gamma < 1)$

* Q - LEARNING ALGORITHM (Pseudo code)

① For each S, A initialize the table entry $\hat{Q}(S,A) \leftarrow 0$

② Observe the current state S

③ Do forever:

① Select an action A and execute it

② Receive immediate reward R

③ Observe new state s'

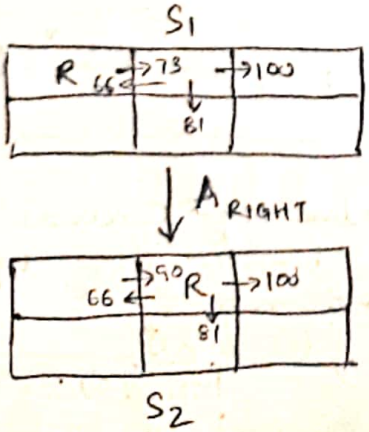
④ Update the table entry for $\hat{Q}(S,A)$ as:

$$\boxed{\hat{Q}(S,A) \leftarrow R + \gamma \max_{A'} \hat{Q}(s',A')}$$

⑤ $S \leftarrow s'$

Learner's estimate / hypothesis of actual Q function

Eg:



* The agent moves one cell to RIGHT in its grid world $(R=0)$ (A_{RIGHT})

* $\hat{Q}(S,A) \leftarrow R + \gamma \max_{A'} \hat{Q}(s',A')$ [Training rule]

$\Rightarrow \hat{Q}(S_1, A_{RIGHT}) \leftarrow R + \gamma \max_{A'} \hat{Q}(S_2, A')$

$\leftarrow 0 + (0.9) \max_{A'} \{66, 81, 100\}$

$\leftarrow 90.$