Module-1

1 a. Define a Well-Posed Learning Problem. Quote some successful applications of machine learning. **(10 Marks)**

b. Elaborate the design choices of choosing the training experience and choosing the Target Function while designing a learning system. **(10 Marks)**

OR

2 a. Depict the program modules of learning systems. **(04 Marks)**

b. Write the notation for most general hypothesis and most specific hypothesis. **(04 Marks)**

c. Consider the following set of training examples:

Example	Sky	Air Temp	Humidity	Wind	Water	Forecast	Enjoy Sport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Write the Candidate-Elimination Algorithm and illustrate the steps to arrive at a final version space using the above training examples. **(12 Marks)**

Module-2

3 a. List the advantages of Decision Tree representation. Which problems are appropriate for Decision Tree learning? **(10 Marks)**

b. Present the ID3 algorithm for decision tree learning. **(10 Marks)**

OR

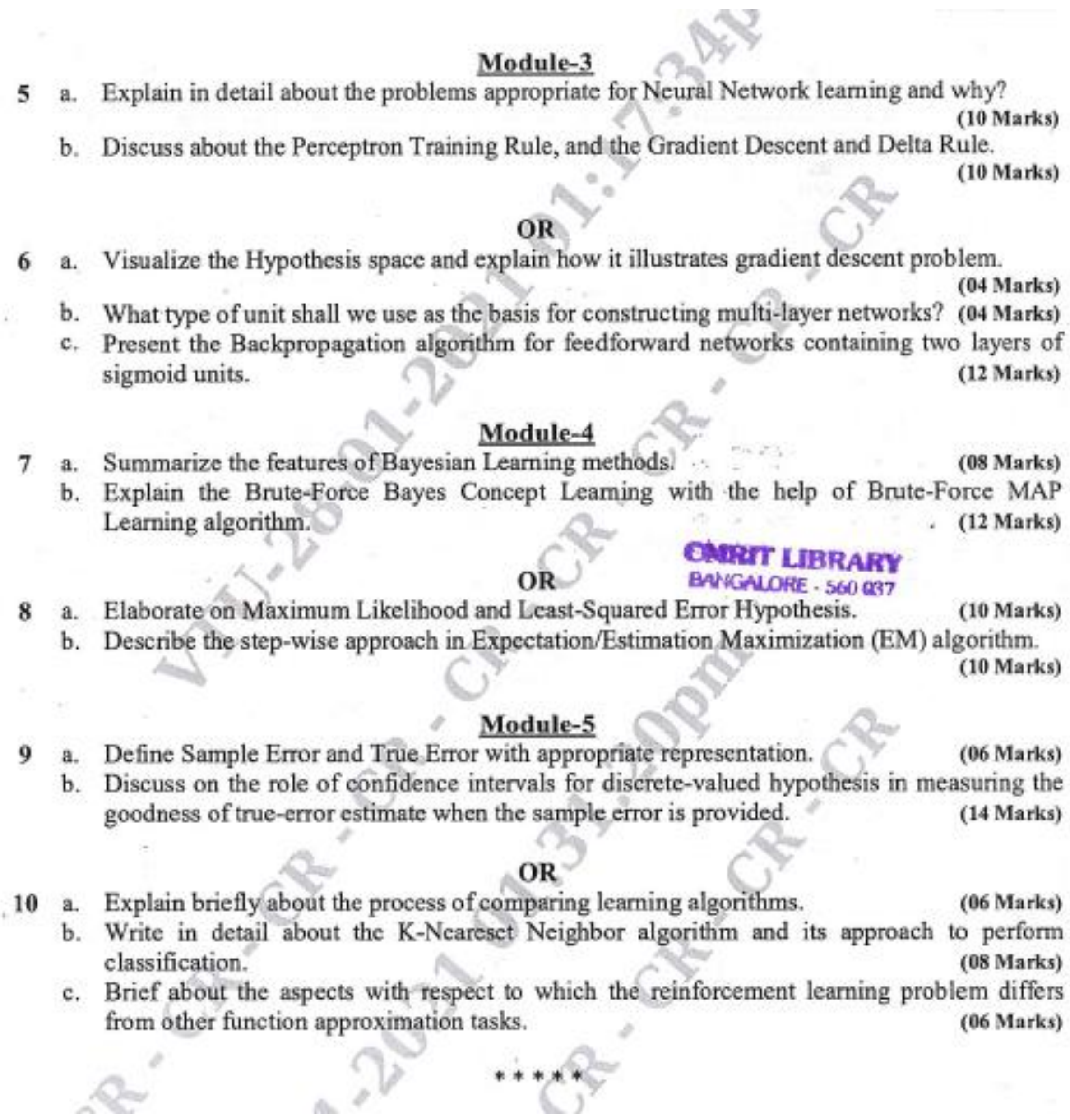
4 a. Consider the following set of training examples:

Instance	Classification	a_1	a_2
1	+	T	T
2	+	T	T
3	–	T	F
4	+	F	F
5	–	F	T
6	–	F	T

(i) What is the entropy of this collection of training examples with respect to the target function-classification?

(ii) What is the information gain of a_2 relative to these training examples? **(08 Marks)**

b. Present your understanding about over fitting the data with decision tree learning and how it can be avoided. **(12 Marks)**



Module-3

5 a. Explain in detail about the problems appropriate for Neural Network learning and why? **(10 Marks)**

b. Discuss about the Perceptron Training Rule, and the Gradient Descent and Delta Rule. **(10 Marks)**

OR

6 a. Visualize the Hypothesis space and explain how it illustrates gradient descent problem. **(04 Marks)**

b. What type of unit shall we use as the basis for constructing multi-layer networks? **(04 Marks)**

c. Present the Backpropagation algorithm for feedforward networks containing two layers of sigmoid units. **(12 Marks)**

Module-4

7 a. Summarize the features of Bayesian Learning methods. **(08 Marks)**

b. Explain the Brute-Force Bayes Concept Learning with the help of Brute-Force MAP Learning algorithm. **(12 Marks)**

OR

8 a. Elaborate on Maximum Likelihood and Least-Squared Error Hypothesis. **(10 Marks)**

b. Describe the step-wise approach in Expectation/Estimation Maximization (EM) algorithm. **(10 Marks)**

Module-5

9 a. Define Sample Error and True Error with appropriate representation. **(06 Marks)**

b. Discuss on the role of confidence intervals for discrete-valued hypothesis in measuring the goodness of true-error estimate when the sample error is provided. **(14 Marks)**

OR

10 a. Explain briefly about the process of comparing learning algorithms. **(06 Marks)**

b. Write in detail about the K-Nearesct Neighbor algorithm and its approach to perform classification. **(08 Marks)**

c. Brief about the aspects with respect to which the reinforcement learning problem differs from other function approximation tasks. **(06 Marks)**

* * * * *

1.a Well-Posed Learning Definition: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Examples:

Checkers Game: A computer program that learns to play checkers might improve its performance as measured by its ability to win at the class of tasks involving playing checkers game, through experience obtained by playing games against itself:

***checkers learning problem*:**

- Task T: playing checkers
- Performance measure P: percent of games won against opponents
- Training experience E: playing practice games against itself

A handwriting recognition learning problem:

- Task T: recognizing and classifying handwritten words within images
- Performance measure P: percent of words correctly classified
- Training experience E: a database of handwritten words with given classifications

***A robot driving learning problem*:**

- Task T: driving on public four-lane highways using vision sensors
- Performance measure P: average distance travelled before an error (as judged by human overseer)
- Training experience E: a sequence of images and steering commands recorded while observing a human driver.

1.b. The basic design issues and approaches to machine learning are illustrated by designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament

1. Choosing the Training Experience
2. Choosing the Target Function
3. Choosing a Function Approximation Algorithm
   1. Estimating training values
   2. Adjusting the weights

1. Choosing the Training Experience

- The first design choice is to choose the type of training experience from which the system will learn.
- The type of training experience available can have a significant impact on success or failure of the learner.

<u>There are three attributes which impact on success or failure of the learner</u>

1. Whether the training experience provides ***direct or indirect feedback*** regarding the choices made by the performance system.

   <u>For example, in checkers game</u>:

   In learning to play checkers, the system might learn from ***direct training examples*** consisting of ***individual checkers board states*** and ***the correct move for each***.

   Indirect training examples consisting of the ***move sequences*** and ***final outcomes*** of various games played. The information about the correctness of specific moves early in the game must be inferred indirectly from the fact that the game was eventually won or lost.

   Here the learner faces an additional problem of ***credit assignment***, or determining the degree to which each move in the sequence deserves credit or blame for the final outcome.
2. The degree to which the ***learner controls the sequence of training examples***

   <u>For example, in checkers game</u>:

   The learner might depends on the ***teacher*** to select informative board states and to provide the correct move for each.

Alternatively, the learner might itself propose board states that it finds particularly confusing and ask the teacher for the correct move.

The learner may have complete control over both the board states and (indirect) training classifications, as it does when it learns by playing against itself with ***no teacher present***.

3. How well it represents the ***distribution of examples*** over which the final system performance P must be measured

   <u>For example, in checkers game:</u>
   In checkers learning scenario, the performance metric P is the percent of games the system wins in the world tournament.

   If its training experience E consists only of games played against itself, there is a danger that this training experience might not be fully representative of the distribution of situations over which it will later be tested.
   It is necessary to learn from a distribution of examples that is different from those on which the final system will be evaluated.

2. Choosing the Target Function

The next design choice is to determine exactly what type of knowledge will be learned and how this will be used by the performance program.

Let's consider a checkers-playing program that can generate the legal moves from any board state.
The program needs only to learn how to choose the best move from among these legal moves. We must learn to choose among the legal moves, the most obvious choice for the type of information to be learned is a program, or function, that chooses the best move for any given board state.

1. Let ***ChooseMove*** be the target function and the notation is

   $$ChooseMove : B \rightarrow M$$

   which indicate that this function accepts as input any board from the set of legal board states B and produces as output some move from the set of legal moves M.
   ChooseMove is a choice for the target function in checkers example, but this function will turn out to be very difficult to learn given the kind of indirect training experience available to our system

2. An alternative target function is an ***evaluation function*** that assigns a ***numerical score*** to any given board state
   Let the target function V and the notation

   $$V: B \quad \rightarrow R$$

   which denote that V maps any legal board state from the set B to some real value. Intend for this target function V to assign higher scores to better board states. If the system can successfully learn such a target function V, then it can easily use it to select the best move from any current board position.

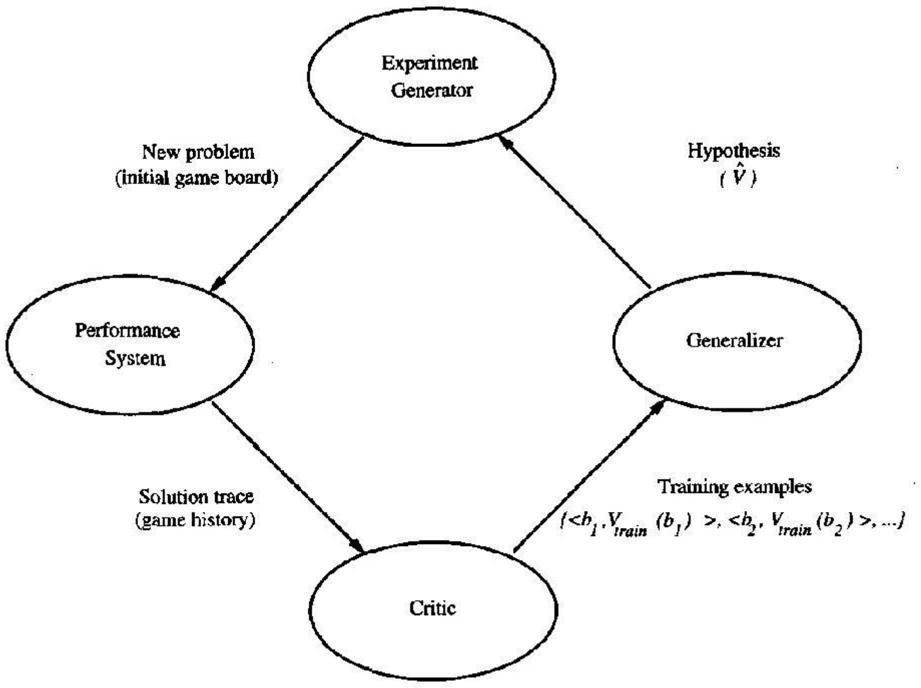
Let us define the target value V(b) for an arbitrary board state b in B, as follows:

- If b is a final board state that is won, then V(b) = 100
- If b is a final board state that is lost, then V(b) = -100
- If b is a final board state that is drawn, then V(b) = 0
- If b is a not a final state in the game, then V(b) = V(b'),

Where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game

Q2.a

learning system can be described by four distinct program modules that represent the central components in many learning systems


Experiment Generator
New problem (initial game board)
Hypothesis ($\hat{V}$)
Performance System
Generalizer
Solution trace (game history)
Training examples $\{<b_1, V_{train}(b_1)>, <b_2, V_{train}(b_2)>, \ldots\}$
Critic

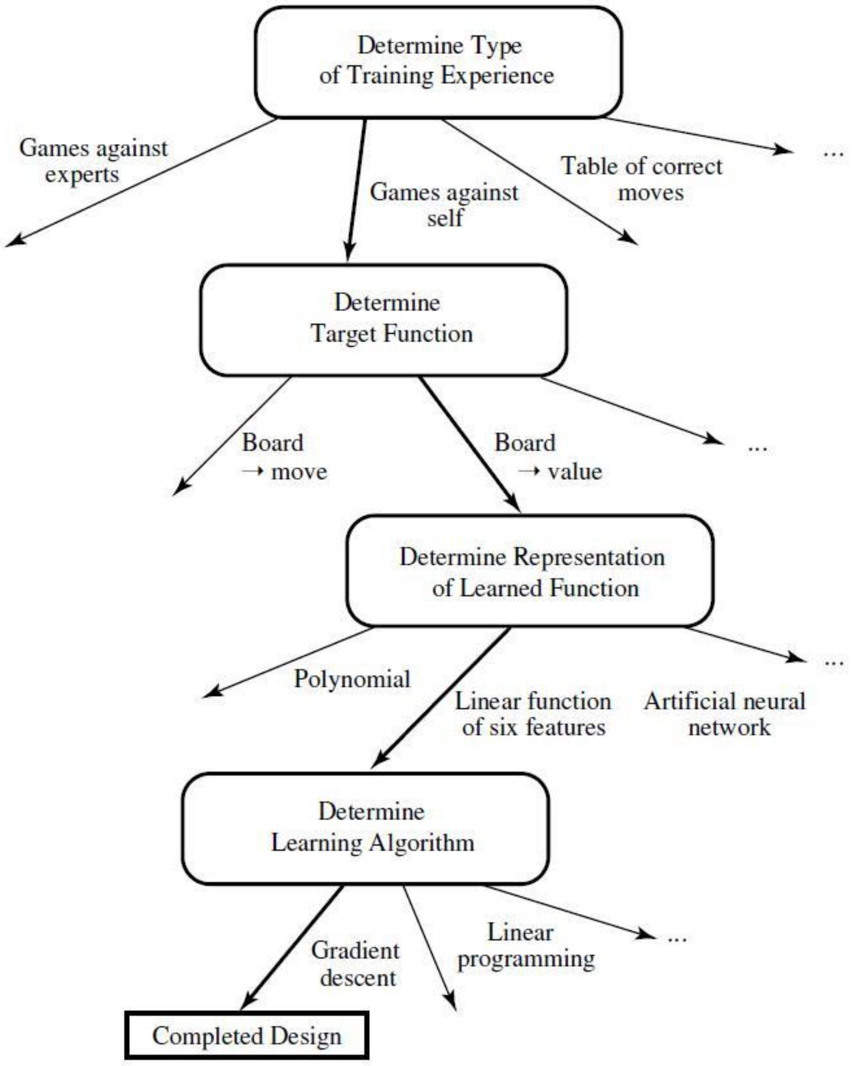
1. **The Performance System** is the module that must solve the given performance task by using the learned target function(s). It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output.

2. **The Critic** takes as input the history or trace of the game and produces as output a set of training examples of the target function

3. **The Generalizer** takes as input the training examples and produces an output hypothesis that is its estimate of the target function. It generalizes from the specific training examples, hypothesizing a general function that covers these examples and other cases beyond the training examples.

4. **The Experiment Generator** takes as input the current hypothesis and outputs a new problem (i.e., initial board state) for the Performance System to explore. Its role is to pick new practice problems that will maximize the learning rate of the overall system.

The sequence of design choices made for the checkers program is summarized in below figure


Determine Type of Training Experience
Games against experts
Games against self
Table of correct moves
...
Determine Target Function
Board → move
Board → value
...
Determine Representation of Learned Function
Polynomial
Linear function of six features
Artificial neural network
...
Determine Learning Algorithm
Gradient descent
Linear programming
...
Completed Design

Q2.b

General-to-Specific Ordering of Hypotheses

Consider the two hypotheses

h_1 = (Sunny, ?, ?, Strong, ?, ?)

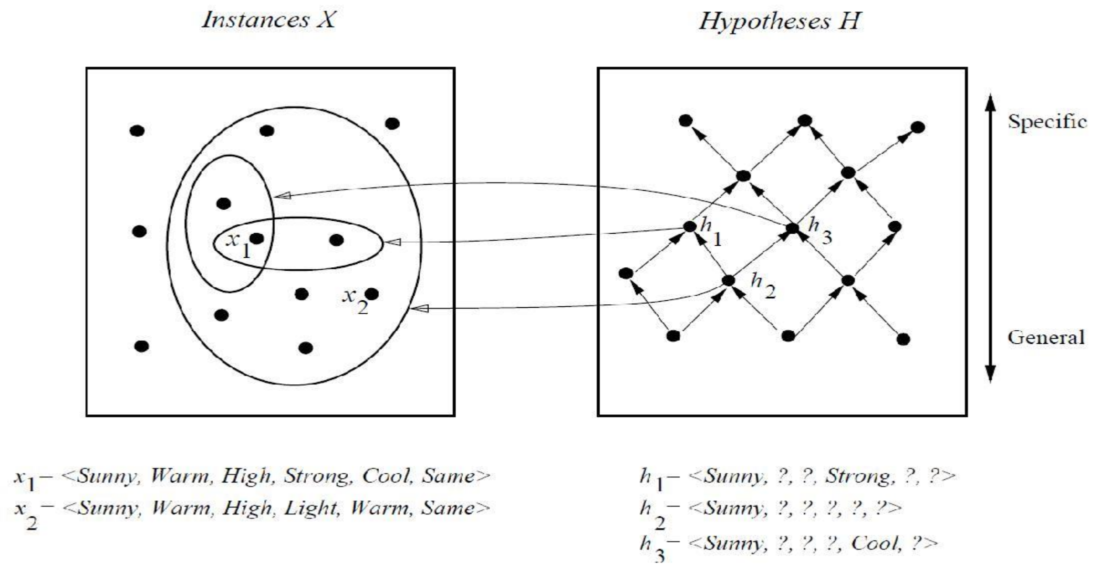
h_2 = (Sunny, ?, ?, ?, ?, ?)

- Consider the sets of instances that are classified positive by h_l and by h_2.
- h_2 imposes fewer constraints on the instance, it classifies more instances as positive. So, any instance classified positive by h_l will also be classified positive by h_2. Therefore, h_2 is more general than h_l.

Given hypotheses h_j and h_k, h_j is more-general-than or- equal do h_k if and only if any instance that satisfies h_k also satisfies h_i

Definition: Let h_j and h_k be Boolean-valued functions defined over X. Then h_j is ***more general-than-or-equal-to*** h_k (written $h_j \geq h_k$) if and only if

$$(\forall\ x \in X)\ [(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$$



Instances X
Hypotheses H
Specific
General
x_1
x_2
h_1
h_2
h_3
x_1 – <Sunny, Warm, High, Strong, Cool, Same>
x_2 – <Sunny, Warm, High, Light, Warm, Same>
h_1 – <Sunny, ?, ?, Strong, ?, ?>
h_2 – <Sunny, ?, ?, ?, ?, ?>
h_3 – <Sunny, ?, ?, ?, Cool, ?>

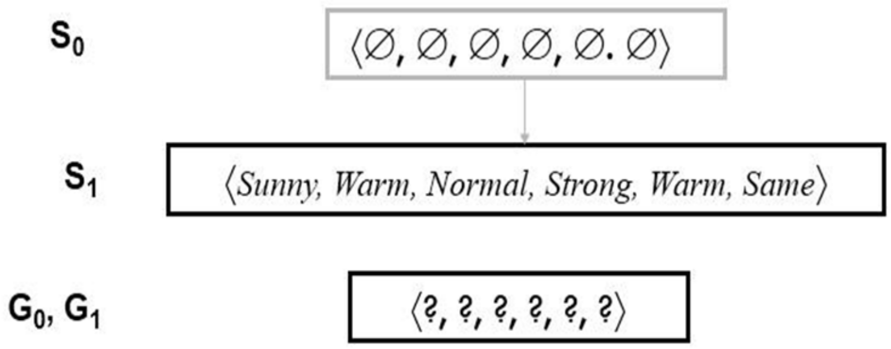
Q2.c

CANDIDATE-ELIMINTION algorithm begins by initializing the version space to the set of all hypotheses in H;

Initializing the G boundary set to contain the most general hypothesis in H G0 ?, ?, ?, ?, ?, ?

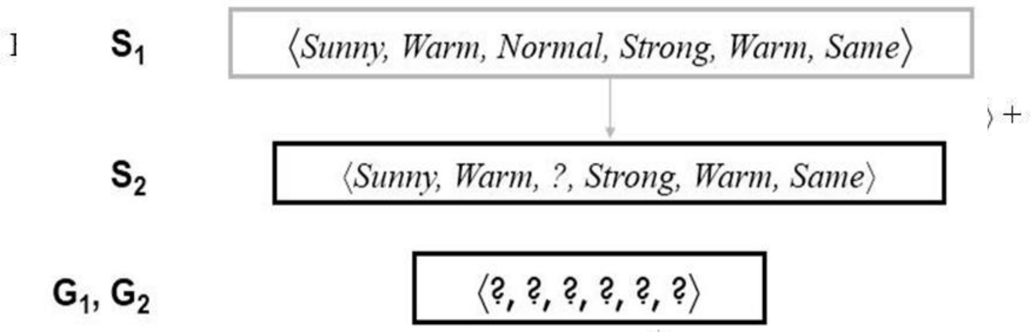
Initializing the S boundary set to contain the most specific (least general) hypothesis S0 □□, □, □, □, □, □□

For training example d,
⟨Sunny, Warm, Normal, Strong, Warm, Same ⟩ +

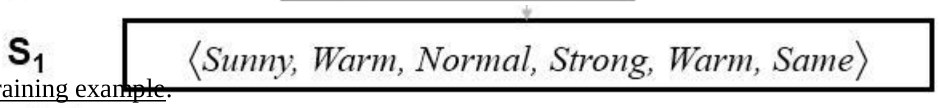


- □ When the second training example is observed, it has a similar effect of generalizing S further to S2, leaving G again unchanged i.e., G2 = G1 = G0

For training example d,
⟨Sunny, Warm, High, Strong, Warm, Same⟩ +

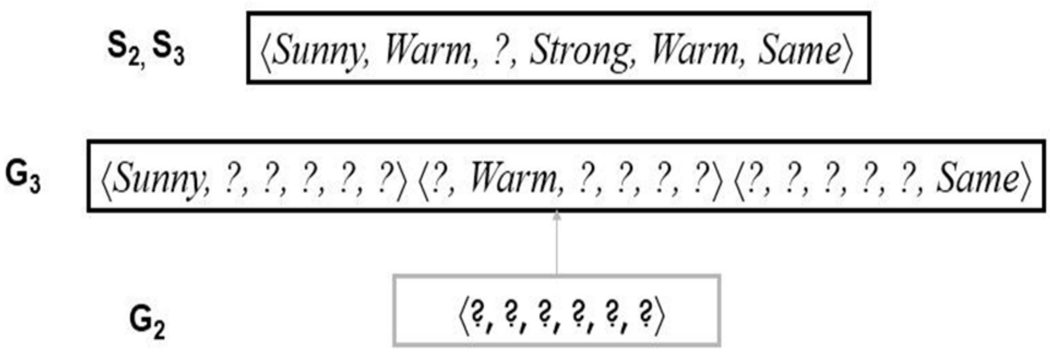


S_1
⟨Sunny, Warm, Normal, Strong, Warm, Same⟩
⟩ +
S_2
⟨Sunny, Warm, ?, Strong, Warm, Same⟩
G_1, G_2
⟨?, ?, ?, ?, ?, ?⟩



Consider the third training example.

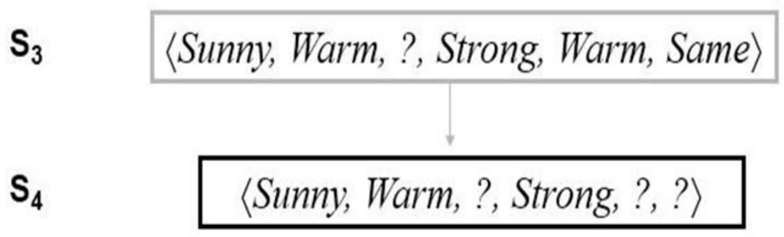
For training example d,
⟨Rainy, Cold, High, Strong, Warm, Change ⟩ –



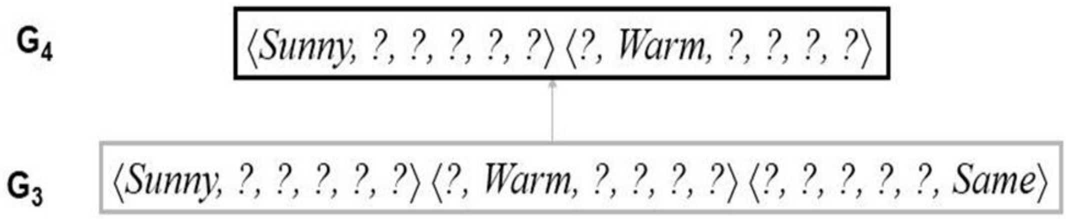
S_2, S_3
⟨Sunny, Warm, ?, Strong, Warm, Same⟩
G_3
⟨Sunny, ?, ?, ?, ?, ?⟩ ⟨?, Warm, ?, ?, ?, ?⟩ ⟨?, ?, ?, ?, ?, Same⟩
G_2
⟨?, ?, ?, ?, ?, ?⟩

Consider the fourth training example

For training example d,
⟨Sunny, Warm, High, Strong, Cool Change⟩ +

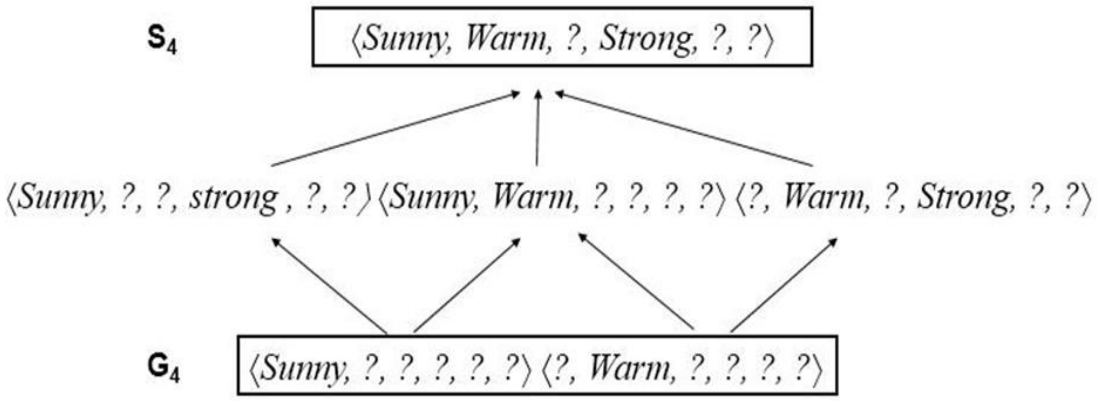


S_3
⟨Sunny, Warm, ?, Strong, Warm, Same⟩
S_4
⟨Sunny, Warm, ?, Strong, ?, ?⟩



G_4
⟨Sunny, ?, ?, ?, ?, ?⟩ ⟨?, Warm, ?, ?, ?, ?⟩
G_3
⟨Sunny, ?, ?, ?, ?, ?⟩ ⟨?, Warm, ?, ?, ?, ?⟩ ⟨?, ?, ?, ?, ?, Same⟩

After processing these four examples, the boundary sets S4 and G4 delimit the version space of all hypotheses consistent with the set of incrementally observed training examples.



S_4
⟨Sunny, Warm, ?, Strong, ?, ?⟩
⟨Sunny, ?, ?, strong , ?, ?⟩
⟨Sunny, Warm, ?, ?, ?, ?⟩
⟨?, Warm, ?, Strong, ?, ?⟩
G_4
⟨Sunny, ?, ?, ?, ?, ?⟩ ⟨?, Warm, ?, ?, ?, ?⟩

Q3.a **APPROPRIATE PROBLEMS FOR DECISION TREE LEARNING**

Decision tree learning is generally best suited to problems with the following characteristics:

1. ***Instances are represented by attribute-value pairs*** – Instances are described by a fixed set of attributes and their values

2. ***The target function has discrete output values*** – The decision tree assigns a Boolean classification (e.g., yes or no) to each example. Decision tree methods easily extend to learning functions with more than two possible output values.

3. ***Disjunctive descriptions may be required***

4. ***The training data may contain errors*** – Decision tree learning methods are robust to errors, both errors in classifications of the training examples and errors in the attribute values that describe these examples.

5. ***The training data may contain missing attribute values –*** Decision tree methods can be used even when some training examples have unknown values

Q3.b ID3(Examples, Target_attribute, Attributes)

Examples are the training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

```
• Create a Root node for the tree
• If all Examples are positive, Return the single-node tree Root, with label = +
• If all Examples are negative, Return the single-node tree Root, with label = -
• If Attributes is empty, Return the single-node tree Root, with label = most common value of
  Target_attribute in Examples
• Otherwise Begin
    • A ← the attribute from Attributes that best* classifies Examples
    • The decision attribute for Root ← A
    • For each possible value, $v_i$, of A,
        • Add a new tree branch below Root, corresponding to the test A = $v_i$
        • Let Examples $_{v_i}$, be the subset of Examples that have value $v_i$ for A
        • If Examples $_{v_i}$ , is empty
            • Then below this new branch add a leaf node with label = most common value
              of Target_attribute in Examples
            • Else below this new branch add the subtree
                  ID3(Examples $_{v_i}$, Targe_tattribute, Attributes – {A}))
• End
• Return Root
```

Q4:

(a) $\text{Entropy}(S) = -p_{\oplus}\log_2 p_{\oplus} - p_{\ominus}\log_2 p_{\ominus}$

$= -\frac{3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6} = 1.$

(b) $\text{Gain}(S, a_2) = \text{Entropy}(S) - \sum_{v}\frac{|S_v|}{|S|}\text{Entropy}(S_v)$

$= 1 - \frac{4}{6}\text{Entropy}(S_{a_2=T}) - \frac{2}{6}\text{Entropy}(S_{a_2=F})$

$= 1 - \frac{4}{6} - \frac{2}{6} = 0$

Q4.b

- The ID3 algorithm grows each branch of the tree just deeply enough to perfectly classify the training examples but it can lead to difficulties when there is noise in the data, or when the number of training examples is too small to produce a representative sample of the true target function. This algorithm can produce trees that overfit the training examples.

<u>Definition - Overfit:</u> Given a hypothesis space H, a hypothesis h ∈ H is said to overfit the training data if there exists some alternative hypothesis h' ∈ H, such that h has smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances

<u>Approaches to avoiding overfitting in decision tree learning</u>

2. Pre-pruning (avoidance): Stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data

3. Post-pruning (recovery): Allow the tree to overfit the data, and then post-prune the tree

Reduced-Error Pruning

- Reduced-error pruning, is to consider each of the decision nodes in the tree to be candidates for pruning

- ***Pruning*** a decision node consists of removing the subtree rooted at that node, making it a leaf node, and assigning it the most common classification of the training examples affiliated with that node

- Nodes are removed only if the resulting pruned tree performs no worse than-the original over the validation set.

- Reduced error pruning has the effect that any leaf node added due to coincidental regularities in the training set is likely to be pruned because these same coincidences are unlikely to occur in the validation set.

Rule Post-Pruning

Rule post-pruning is successful method for finding high accuracy hypotheses

- Rule post-pruning involves the following steps:
- Infer the decision tree from the training set, growing the tree until the training data is fit as well as possible and allowing overfitting to occur.
- Convert the learned tree into an equivalent set of rules by creating one rule for each path from the root node to a leaf node.
- Prune (generalize) each rule by removing any preconditions that result in improving its estimated accuracy.
- Sort the pruned rules by their estimated accuracy, and consider them in this sequence when classifying subsequent instances.

a. APPROPRIATE PROBLEMS FOR NEURAL NETWORK LEARNING

ANN learning is well-suited to problems in which the training data corresponds to noisy, complex sensor data, such as inputs from cameras and microphones.

ANN is appropriate for problems with the following characteristics:

1. Instances are represented by many attribute-value pairs.
2. The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.
3. The training examples may contain errors.
4. Long training times are acceptable.
5. Fast evaluation of the learned target function may be required
6. The ability of humans to understand the learned target function is not important.

Q5.b

The Perceptron Training Rule

The learning problem is to determine a weight vector that causes the perceptron to produce the correct + 1 or - 1 output for each of the given training examples.

To learn an acceptable weight vector

- Begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example.
- This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly.
- Weights are modified at each step according to the perceptron training rule, which revises the weight w_i associated with input xi according to the rule.

$$w_i \leftarrow w_i + \Delta w_i$$

Where,

$$\Delta w_i = \eta(t - o)x_i$$

Here,
t is the target output for the current training example
o is the output generated by the perceptron
η is a positive constant called the ***learning rate***

- The role of the learning rate is to moderate the degree to which weights are changed at each step. It is usually set to some small value (e.g., 0.1) and is sometimes made to decay as the number of weight-tuning iterations increases.

Gradient Descent and the Delta Rule

- If the training examples are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept.
- The key idea behind the delta rule is to use ***gradient descent*** to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.

To understand the delta training rule, consider the task of training an unthresholded perceptron. That is, a linear unit for which the output ***O*** is given by

$$o = w_0 + w_1x_1 + \cdots + w_nx_n$$

$$O(\vec{x}) = (\vec{w} . \vec{x}) \qquad \text{equ. (1)}$$

To derive a weight learning rule for linear units, specify a measure for the ***training error*** of a hypothesis (weight vector), relative to the training examples.

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \qquad \text{equ. (2)}$$

Where,

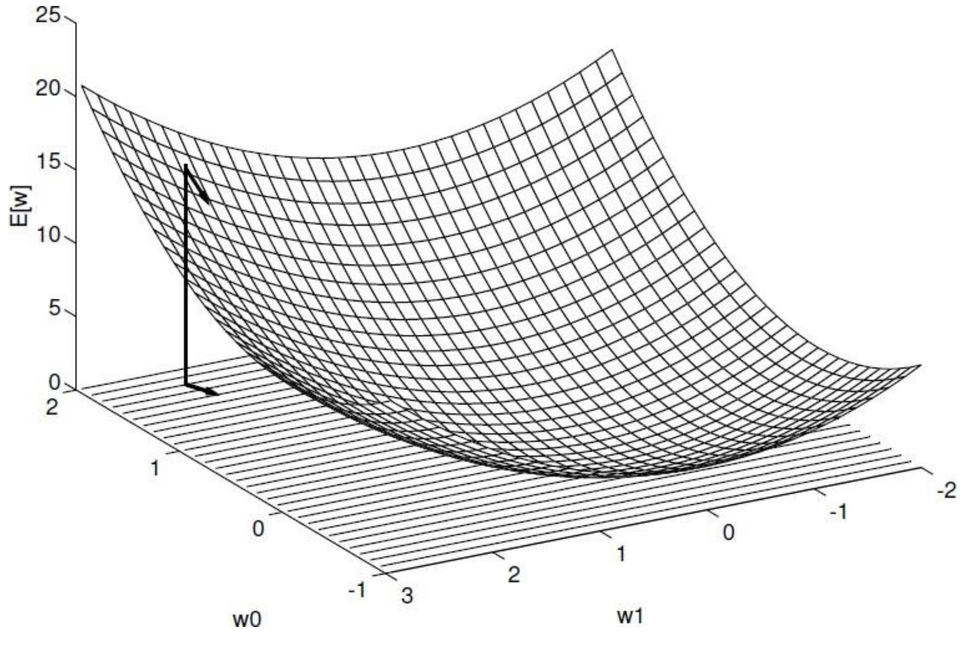
- D is the set of training examples,
- t_d is the target output for training example d,
- o_d is the output of the linear unit for training example d

E ($\vec{w}$) is simply half the squared difference between the target output t_d and the linear unit output o_d, summed over all training examples.

Q6.a.

Visualizing the Hypothesis Space

- To understand the gradient descent algorithm, it is helpful to visualize the entire hypothesis space of possible weight vectors and their associated E values as shown in below figure.
- Here the axes w_0 and w_l represent possible values for the two weights of a simple linear unit. The w_0, w_l plane therefore represents the entire hypothesis space.
- The vertical axis indicates the error E relative to some fixed set of training examples.
- The arrow shows the negated gradient at one particular point, indicating the direction in the w_0, w_l plane producing steepest descent along the error surface.
- The error surface shown in the figure thus summarizes the desirability of every weight vector in the hypothesis space



- Given the way in which we chose to define E, for linear units this error surface must always be parabolic with a single global minimum.

Gradient descent search determines a weight vector that minimizes E by starting with an

arbitrary initial weight vector, then repeatedly modifying it in small steps.
At each step, the weight vector is altered in the direction that produces the steepest descent along the error surface depicted in above figure. This process continues until the global minimum error is reached.

6.

BACKPROPAGATION (*training_example, η, n_{in}, n_{out}, n_{hidden})*

Each training example is a pair of the form ($\vec{x}$, t), where (x) is the vector of network input values, (t) and is the vector of target network output values.

η is the learning rate (e.g., .05). n_i, is the number of network inputs, n_{hidden} the number of units in the hidden layer, and n_{out} the number of output units.

The input from unit i into unit j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}

- Create a feed-forward network with n_i inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers
- Until the termination condition is met, Do
    - For each ($\vec{x}$,   ), in training examples, Do

Propagate the input forward through the network:

1. Input the instance $\vec{x}$, to the network and compute the output o_u of every unit u in the network.

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. For each hidden unit *h*, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

Where

$$\Delta w_{ji} = \eta \delta_j x_{i,j}$$

Propagate the errors backward through the network:

7.

- Each observed training example can incrementally decrease or increase the estimated probability that a hypothesis is correct. This provides a more flexible approach to learning than algorithms that completely eliminate a hypothesis if it is found to be inconsistent with any single example

- Prior knowledge can be combined with observed data to determine the final probability of a hypothesis. In Bayesian learning, prior knowledge is provided by asserting (1) a prior probability for each candidate hypothesis, and (2) a probability distribution over observed data for each possible hypothesis.

- Bayesian methods can accommodate hypotheses that make probabilistic predictions

- New instances can be classified by combining the predictions of multiple hypotheses, weighted by their probabilities.

- Even in cases where Bayesian methods prove computationally intractable, they can provide a standard of optimal decision making against which other practical methods can be measured.

8.a

Consider the problem of learning a *continuous-valued target function* such as neural network learning, linear regression, and polynomial curve fitting

A straightforward Bayesian analysis will show that under certain assumptions any learning algorithm that minimizes the squared error between the output hypothesis predictions and the training data will output a *maximum likelihood (ML) hypothesis*

- Learner L considers an instance space X and a hypothesis space H consisting of some class of real-valued functions defined over X, i.e., $(\forall h \in H)[h : X \rightarrow R]$ and training examples of the form $<x_i,d_i>$
- The problem faced by L is to learn an unknown target function $f : X \rightarrow R$
- A set of m training examples is provided, where the target value of each example is corrupted by random noise drawn according to a Normal probability distribution with zero mean ($d_i = f(x_i) + e_i$)
- Each training example is a pair of the form (x_i, d_i) where $d_i = f(x_i) + e_i$.
  - – Here $f(x_i)$ is the noise-free value of the target function and e_i is a random variable representing the noise.
  - – It is assumed that the values of the e_i are drawn independently and that they are distributed according to a Normal distribution with zero mean.
- The task of the learner is to *output a maximum likelihood hypothesis* or a *MAP hypothesis assuming all hypotheses are equally probable a priori*.

Using the definition of h_{ML} we have

$$h_{ML} = \underset{h \in H}{argmax}\ p(D|h)$$

Assuming training examples are mutually independent given h, we can write P(D|h) as the product of the various $(d_i|h)$

$$h_{ML} = \underset{h \in H}{argmax} \prod_{i=1}^{m} p(d_i|h)$$

Given the noise e_i obeys a Normal distribution with zero mean and unknown variance σ^2, each d_i must also obey a Normal distribution around the true targetvalue $f(x_i)$. Because we are writing the expression for P(D|h), we assume h is the correct description of f.

Hence, $\mu = f(x_i) = h(x_i)$

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i - \mu)^2}$$

Maximize the less complicated logarithm, which is justified because of the monotonicity of function p

$$h_{ML} = \underset{h \in H}{argmax} \sum_{i=1}^{m} \ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

The first term in this expression is a constant independent of h, and can therefore be discarded, yielding

$$h_{ML} = \underset{h \in H}{argmax} \sum_{i=1}^{m} -\frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Maximizing this negative quantity is equivalent to minimizing the corresponding positive quantity

$$h_{ML} = \underset{h \in H}{argmin} \sum_{i=1}^{m} \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

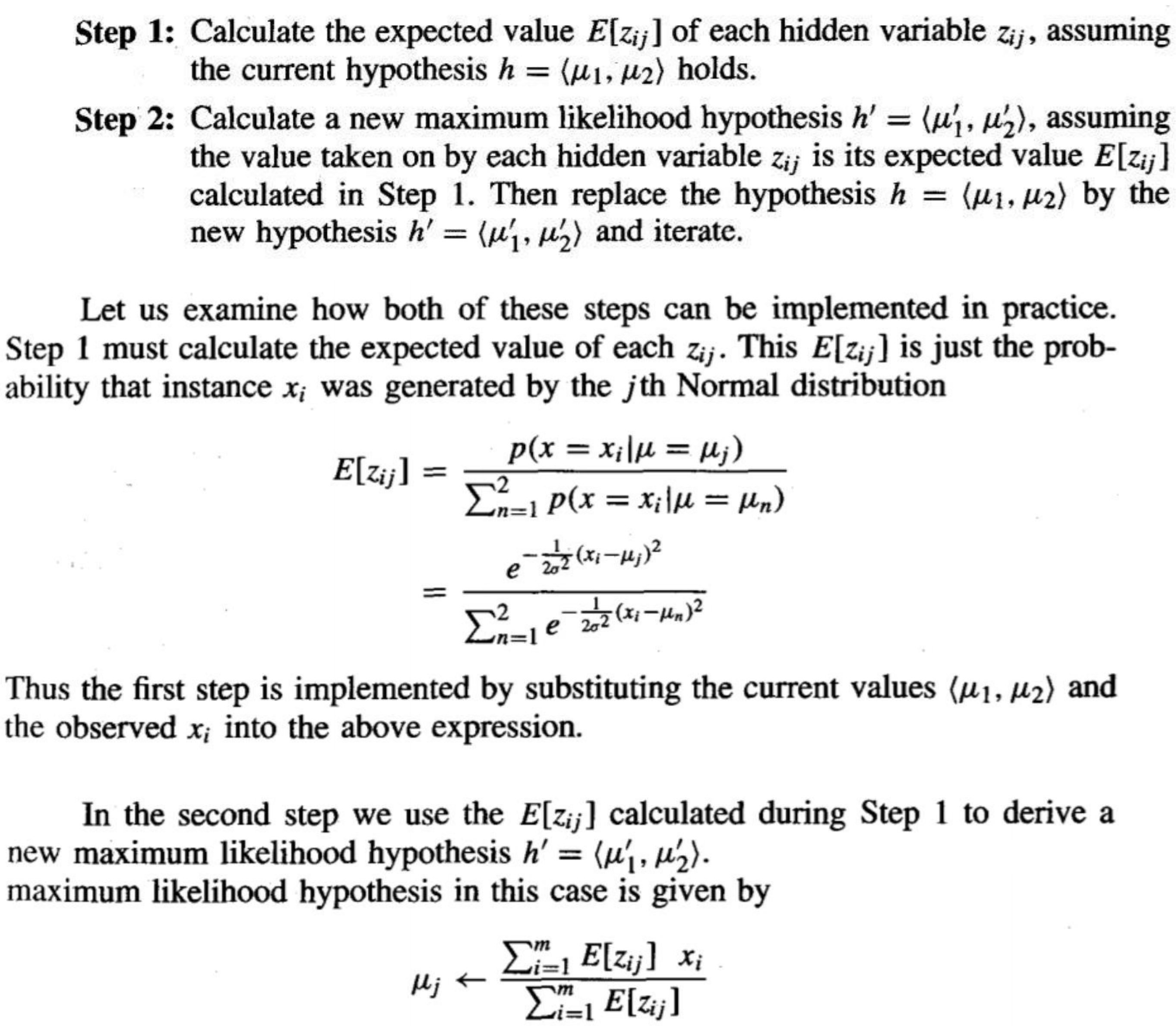
Finally, discard constants that are independent of h.

$$h_{ML} = \underset{h \in H}{argmin} \sum_{i=1}^{m} (d_i - h(x_i))^2$$

Thus, above equation shows that the maximum likelihood hypothesis h_{ML} is the one that minimizes the sum of the squared errors between the observed training values d_i and the hypothesis predictions $h(x_i)$

8.b

EM algorithm



Step 1: Calculate the expected value $E[z_{ij}]$ of each hidden variable z_{ij}, assuming the current hypothesis $h = \langle \mu_1, \mu_2 \rangle$ holds.

Step 2: Calculate a new maximum likelihood hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$, assuming the value taken on by each hidden variable z_{ij} is its expected value $E[z_{ij}]$ calculated in Step 1. Then replace the hypothesis $h = \langle \mu_1, \mu_2 \rangle$ by the new hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$ and iterate.

Let us examine how both of these steps can be implemented in practice. Step 1 must calculate the expected value of each z_{ij}. This $E[z_{ij}]$ is just the probability that instance x_i was generated by the jth Normal distribution

$$E[z_{ij}] = \frac{p(x = x_i | \mu = \mu_j)}{\sum_{n=1}^{2} p(x = x_i | \mu = \mu_n)}$$

$$= \frac{e^{-\frac{1}{2\sigma^2}(x_i - \mu_j)^2}}{\sum_{n=1}^{2} e^{-\frac{1}{2\sigma^2}(x_i - \mu_n)^2}}$$

Thus the first step is implemented by substituting the current values $\langle \mu_1, \mu_2 \rangle$ and the observed x_i into the above expression.

In the second step we use the $E[z_{ij}]$ calculated during Step 1 to derive a new maximum likelihood hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$.
maximum likelihood hypothesis in this case is given by

$$\mu_j \leftarrow \frac{\sum_{i=1}^{m} E[z_{ij}] \ \ x_i}{\sum_{i=1}^{m} E[z_{ij}]}$$

9.a

Sample Error –

The sample error of a hypothesis with respect to some sample S of instances drawn from X is the fraction of S that it misclassifies.

Definition: The sample error (***error$_s$(h)***) of hypothesis h with respect to target function f and data sample S is

$$error_S(h) \equiv \frac{1}{n} \sum_{x \in S} \delta(f(x), h(x))$$

True Error –

The true error of a hypothesis is the probability that it will misclassify a single randomly drawn instance from the distribution **D**.

Definition: The true error (error$_{\mathbf{D}}$(h)) of hypothesis h with respect to target function f and distribution **D**, is the probability that h will misclassify an instance drawn at random according to **D**.

$$error_{\mathcal{D}}(h) \equiv \Pr_{x \in \mathcal{D}} [f(x) \neq h(x)]$$

9.b

Suppose we wish to estimate the true error for some discrete valued hypothesis h, based on its observed sample error over a sample S, where

- The sample S contains n examples drawn independent of one another, and independent of h, according to the probability distribution **D**
- n ≥ 30

- Hypothesis h commits r errors over these n examples (i.e., $error_S(h) = r/n$).

Under these conditions, statistical theory allows to make the following assertions:

1. Given no other information, the most probable value of $error_D(h)$ is $error_S(h)$
2. With approximately **95% probability**, the true error $error_D(h)$ lies in the interval

Example:

$$error_S(h) \pm 1.96\sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

Suppose the data sample S contains n = 40 examples and that hypothesis h commits r = 12 errors over this data.

- The ***sample error*** is $error_S(h) = r/n = 12/40 = 0.30$
- Given no other information, ***true error*** is $error_D(h) = error_S(h)$, i.e., $error_D(h) = 0.30$
- With the 95% confidence interval estimate for $error_D(h)$.

$$error_S(h) \pm 1.96\sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

$= 0.30 \pm (1.96 * 0.07)$ $= 0.30 \pm 0.14$

10 a.

Hypothesis Testing

- Evaluates 2 mutual exclusive statement on population using sample data.
- Steps:
  1. Make initial Assumptions

  2. Collect Data

  3. Gather Evidence to Reject or accept NULL hypothesis

- What is the probability that
  error $_D$(h1) > error $_D$(h2)

COMPARING LEARNING ALGORITHMS

- Comparing the performance of 2 learning algorithms LA and LB.
- A reasonable way to define "on average" is to consider the relative performance of these 2 algorithms averaged over all the training sets of size n over Distribution D.

$$\underset{S\subset \mathcal{D}}{E}[error_{\mathcal{D}}(L_A(S)) - error_{\mathcal{D}}(L_B(S))]$$

Where ,

L(S) : Hypothesis output of learning method L when given the sample S of training data .

Here S c D : The expected value is taken over samples S drawn according to the underlying instance distribution D.

10.b

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 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.