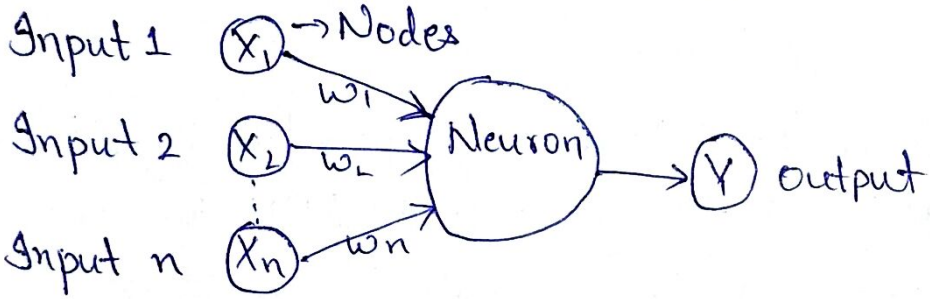
ML UNIT-2

Artificial Neural Networks-1

(1) Introduction:

- The term 'Artificial Neural Network (ANN)' is derived from biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the network.
- These neurons are known as nodes.



Characteristics of ANN:

- In ANN have more neurons that work as a processing elements.
- With the help of weighted connections, all processing elements are connected.
- Implemented learning process to acquire knowledge
- All data are distributed by connections.
- ANN work as mathematical model.

• A neural network contains the following three layers:

i) Input layer – The activity of i/p units represents the raw information that can feed into the n/w.

ii) Hidden layer –

• It is used to determine the activity of each hidden unit.

• The activities of i/p units & weights depends on connections b/w the i/p & hidden units.

• There may be one or more hidden layers.

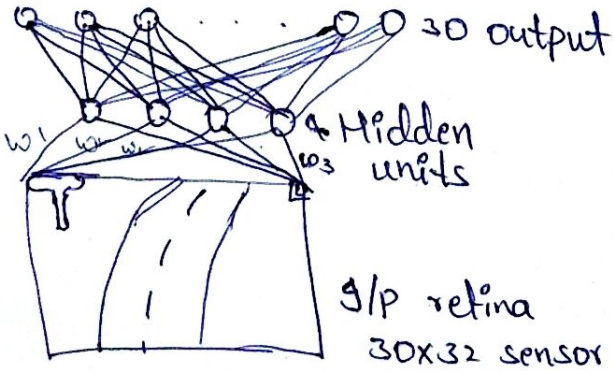
iii) Output layer: The behavior of the o/p units depends on the activity of the hidden units & the weights b/w the hidden & o/p units.

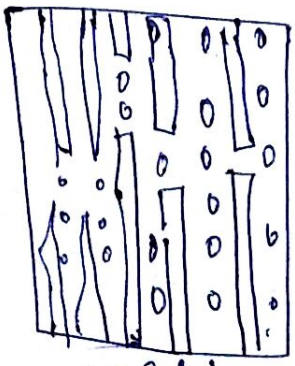
(2) Neural network representation:-

• A protypical example of ANN learning is provided by Pomerleau's (1993) system ALVINN, which uses a learned ANN to steer an autonomous vehicle driving at normal speeds on public highways.

• The i/p to neural n/w is a 30×32 grid of pixel intensities obtained from a forward-pointed camera mounted on the vehicle.

• The n/w o/p is the direction in which the vehicle is steered.





• Each node (i.e circle) in n/w diagram corresponds to o/p of a single n/w unit, & the lines entering the node from below are its inputs.

• There are 4 hidden units that receive inputs directly from all of the 30×32 pixels in the image. These are called "hidden" units becoz

their o/p is available only within the n/w & is not available as part of global n/w o/p.

- Each of these 4 hidden units computes a single real-valued o/p based on a weighted combination of its 960 inputs.
- These hidden unit o/p's are then used as inputs to a second layer of 30 'output' units.
- Each o/p unit corresponds to a particular steering direction, & the o/p values of these units determines which steering direction is recommended most strongly.
- The diagram on right side of figure depicts the learned weight values associated with one of the four hidden units in this ANN.

(3) Appropriate problems for neural network learning:-

- ANN learning is well-suited to problems in which the training data corresponds to noisy, complex sensor dat, such as inputs from cameras and microphones.
- The Backpropogation algorithm is the most commonly used ANN learning technique. It is appropriate for problems with the following characteristics:

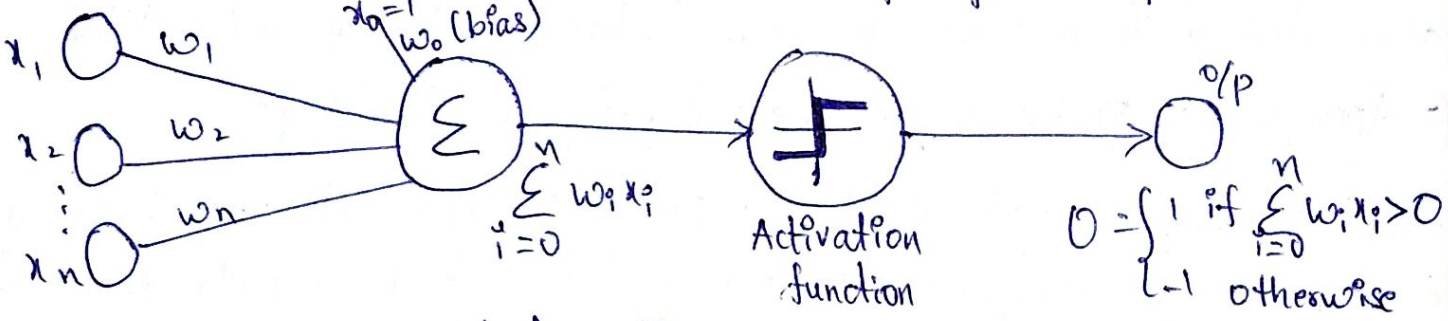
→ Instances are represented by many attribute-value pairs.
→ The target function output may be discrete-valued, real-valued or a vector of several real or discrete-valued attributes.
→ The training examples may contain errors.
→ Long training times are acceptable.
→ Fast evaluation of the learned target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

④ Perceptions:-

- A perceptron unit is used to build the ANN system.
- A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs '1' if the result is > some threshold & '-1' otherwise.
- I/ps's x_1 through x_n, o/p $O(x_1, \ldots, x_n)$ computed by the perceptron is:

$$O(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

- Where each 'w_i' is a real-valued constant or weight, that determines the contribution of i/p x_i to perceptron o/p.



- Here, the i/ps are accepted till the actual o/p = target o/p, if not equal then modify the perceptron weights. This process is repeated, until the perceptron classifies all training examples correctly.
- Weights are modified at each step according to perceptron training rule, which revises the weight 'w_i' associated with i/p 'x_i' according to the rule.

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

where

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

η = learning rate $\simeq 0.1$, t = target o/p, o = actual o/p, x_i = i/p associated with w_i

Perceptron training rule:-

- Algorithm:

Perceptron_training_rule(X, η)
initialize w($w_i \leftarrow$ an initial (small) random value)

```
repeat
    for each training instance (x, tx) ∈ X
        compute the real o/p ox = Activation(Summation(w.x))
        if(tx ≠ ox)
            for each wi
                wi ← wi + Δwi
                Δwi ← n(tx - ox)xi
            end for
        end if
    end for
until all training instances in x are correctly classified
return w.
```

- The perceptron rule finds a successful weight vector when the training examples are linearly separable, it can fail to converge if the examples are not linearly separable.
- A second training rule, called "Delta rule", is designed to overcome this difficulty.

Gradient Descent & Delta Rule:

- The key idea behind the delta rule is to use gradient descent to search the hypothesis space of possible weight vector to find the weights that best fit the training examples.
- This rules is important becoz gradient descent provides the basis for the Backpropagation algorithm, which can learn NW's with many interconnected units.
- To derive a weight learning rule for linear units ($O(\vec{x}) = \vec{w}.\vec{x}$), by specifying a measure for the training error of a hypothesis (weight vector), relative to training examples.

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - O_d)^2$$

where D = set of training examples
t_d = target o/p of d
O_d = o/p for linear unit
$\vec{w}$ = weight vector.

(5) Multilayer networks & the back-propagation algorithm:-

Gradient descent algorithm:

Gradient-Descent(training-examples, n)

Each training example is a pair of the form $(\vec{x}, t)$, where $\vec{x}$ is the vector of i/p values & 't' is target o/p value. 'n' is learning rate

- Initialize each w_i to some small random value.
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $(\vec{x}, t)$ in training_examples, Do
    - i/p the instance $\vec{x}$ to unit & compute the o/p o
    - For each linear unit weight w_i, Do

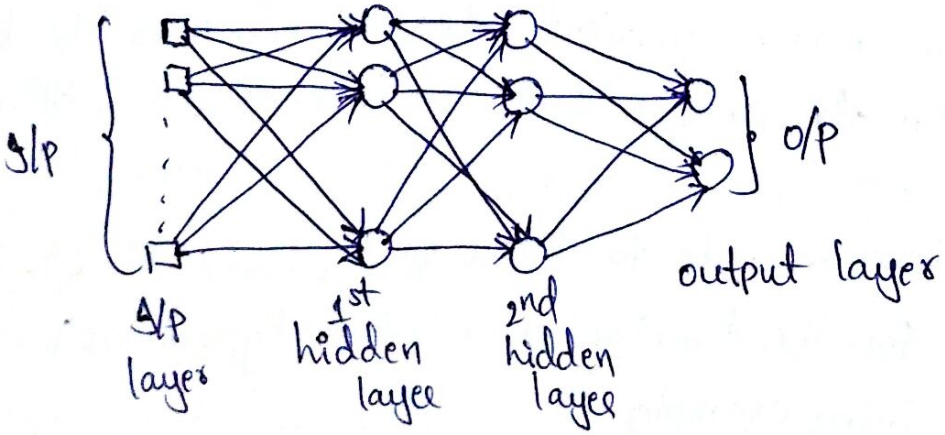
$$\Delta w_i \leftarrow \Delta w_i + n(t-o)x_i$$

  - For each linear unit weight w_i, Do

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

(5) Multilayer networks & the back-propogation algorithm:-

- A neural network that consists of multiple hidden layers is known as multi-layer neural n/w.
- Multilayer networks using gradient descent algorithm.

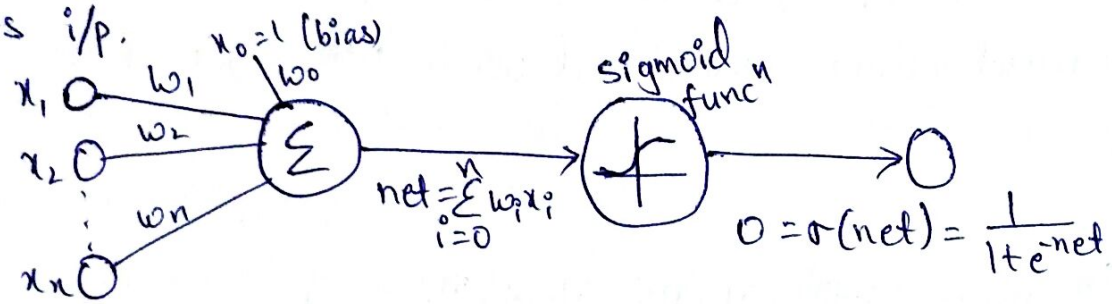


- In perceptron its discontinuous threshold makes it undifferentiable & hence unsuitable for gradient descent.

Differentiable Threshold unit:

. Sigmoid unit is a smoothed, differentiable threshold function.

. Like the perceptron, the sigmoid unit first computes a linear combination of its i/p's, then applies a threshold to the result. In the case of the sigmoid unit, however, the threshold o/p is a continuous func^n of its i/p.



. The sigmoid unit computes its o/p 'o' as $o = \sigma(\vec{w} \cdot \vec{x})$

$$\sigma(y) = \frac{1}{1+e^{-y}}$$

. σ is often called the sigmoid func^n, or the logistic func^n. Its o/p ranges b/w 0 & 1, increasingly monotonically with its i/p.

. It maps a very large i/p domain to small range of o/p's, it is often referred to as the squashing func^n of unit. The sigmoid func^n has the useful property that its derivative is easily expressed in terms of its o/p.

. The func^n 'tanh' is also sometimes used in place of sigmoid func.

BackPropagation Algorithm:

. Backpropagation is a neural n/w algorithm that is used to correct the weights & train the machine learning model.

. Backpropagation algorithm learns the weights for a multilayer n/w, given a n/w with a fixed set of units & interconnections.

. It employs gradient descent to attempt to minimize the squared error b/w the network o/p values & the target values for these o/ps.

• We are considering networks with multiple o/p units rather than single unit as before, we begin by redefining 'E' to sum the errors over all of the n/w o/p units.

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in o/p's} (t_{kd} - o_{kd})^2$$

• where o/p's is the set of o/p units in the n/w, & t_{kd} and o_{kd} are the target & output values associated with k^{th} o/p unit & training example 'd'.

Algorithm:

Back propagation(training-example, η, n_{in}, n_{out}, n_{hidden})

Each training example is a pair of form $(\vec{x}, \vec{t})$, where $\vec{x}$ is vector of n/w i/p values, $(\vec{t})$ is vector of target n/w o/p values.

η is learning rate, n_{in} is no. of n/w i/p's, n_{hidden} is no. of units in hidden layer & n_{out} is no. of o/p units.

The i/p from unit 'i' into unit 'j' is denoted by x_{ji} & the weight from unit 'i' to unit 'j' is denoted by w_{ji}

• Create a feed-forward n/w with n_{in} i/p's, n_{hidden} hidden units & n_{out} o/p units.

• Initialize all n/w weights to small random numbers.

• Until the termination cond^n is met, Do

  • For each $(\vec{x}, \vec{t})$, in training examples, Do

    Propagate the i/p forward through the n/w:

    i) I/p the instance $\vec{x}$, to n/w & compute o/p o_u of every unit 'u' in the n/w

    Propagate the errors backward through the n/w:

    ii) For each n/w o/p unit k, calculate its error term δ_k

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

    iii) For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h (1 - o_h) \sum_{k \in o/p's} w_{h,k} \delta_k$$

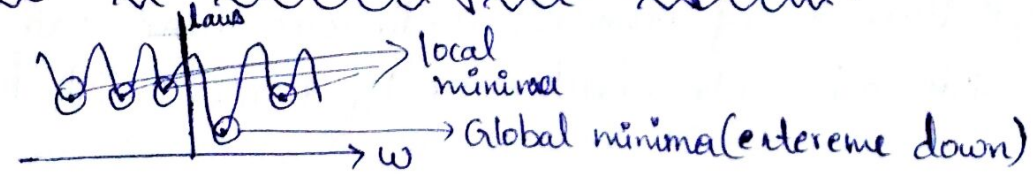
iv) Update each n/w weight w_{ji}

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

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

ANN - 2

(1) Remarks on Back-Propagation algorithm:



i) Convergence & Local Minima:

- Backpropagation over multilayer networks is only guaranteed to converge toward some local minimum in 'E' & not necessarily to the global minimum error.
- When gradient descent falls into a local minimum with respect to one of these weights, it will not necessarily be in a local minimum with respect to other weights.
- A 2nd perspective on local minima can be gained by considering the manner in which n/w weights evolve as the no. of training iterations increases.

Common heuristics to attempt to alleviate the problem of local minima include:

- Add a momentum term to weight-update rule. Momentum can sometimes carry the gradient descent procedure through narrow local minima.
- Train multiple n/w's using the same data, but initializing each n/w with different random weights.
- If the different training efforts lead to different local minima, then the n/w with best performance over a separate validation data caset can be selected.

ii) Representational power of FeedForward networks:

Three rules of feedforward n/w.

a) **Boolean functions**: Every boolean func^n can be represented exactly by some n/w with two layers of units, although the no.of hidden units required grows exponentially in worst case with no.of n/w i/p's

b) **Continuous functions**: Every bounded continuous func^n can be approximated with arbitrarily small error by a n/w with two layers of units.

c) **Arbitrary functions**: Any func^n can be approximated to arbitrary accuracy by a n/w with 3 layers of units. Again, the o/p layer uses linear units, the two hidden layers use sigmoid units, & the no.of units required at each layer is not known in general.

iii) Hypothesis Space search and Inductive Bias:

- Hypothesis space is the n-dimensional Euclidean space of the n n/w weights. This hypothesis space is continuous, in contrast to the hypothesis spaces of decision tree learning & other methods based on discrete representations.
- Inductive bias is depends on interplay b/w the gradient descent search & the way in which the weight space spans the space of represent able functions. However, one can roughly characterize it as smooth interpolation b/w data points.

iv) Hidden layer Representations:

- One intriguing property of backpropagation its ability to discover useful intermediate representations at the hidden unit layers inside the n/w.

v) Generalization, Overfitting & Stopping Criterion:

- Backpropagation is susceptible to overfitting the training examples at the cost of ↓ing generalization accuracy over other unseen examples.

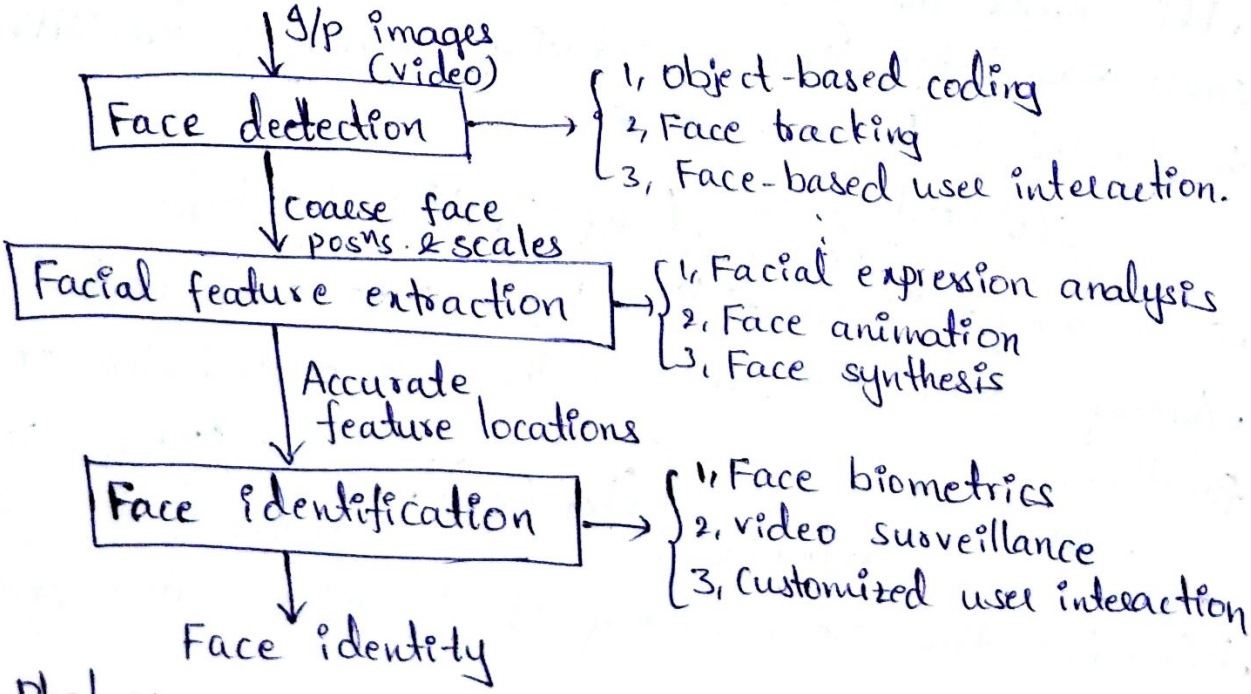
. Several techniques are available to address the overfitting problem for backpropagation learning. One approach, known as weight decay, is to ves each weight by some small factor during each iteration.

. One of the most successful methods for overcoming the overfitting problem is to simply provide a set of validation data to the algorithm in addition to the training data.

(2) An illustrative example: Face recognition:

. ANN have been used in the field of image processing and pattern recognition.

. A general face recognition system includes many steps: face detection, feature extraction, & face recognition. In the recent years, different architectures & models of ANN were used for face detection and recognition.



Eg: Google Photos:

. Google photos has had facial recognition technology for a few years now, With it, you can let Google scan your photo library to help identify & tag people who appear in your photos.

. Not only can you see all pics & video's you've taken of a person or pet, you can also use that in search to find specific snaps like "me in Vienna" or "[Person] & [other person/cat/dog]" or "[person] and food" for example.

. But Google's algorithms aren't infalliable & there are instances when you have to remove certain pics for wrong identification, or merge what it thinks are two different faces. Photos is aware of that & has been asking users on web to help it improve its facial recognition.

③ Advanced topics in ANN's :-

i) Alternative Error Functions :-

. Adding a penalty term for weight magnitude. We can add a term to E' that ↑es with the magnitude of the weight vector.

. This causes the gradient descent search to seek weight vectors with small magnitudes, there by reducing the risk of overfitting. One way to do this is to redifine E as

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in o/p's} (t_{kd} - o_{kd})^2 + \gamma \sum_{i,j} w_{ji}^2$$

. Each weight is multiplied by constant upon each iteration.

. Adding a term for errors in the slope, or derivative of the target function.

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in o/p's} \left[(t_{kd} - o_{kd})^2 + \mu \sum_{j \in i/p's} \left(\frac{\partial t_{kd}}{\partial x_d^j} - \frac{\partial o_{kd}}{\partial x_d^j} \right)^2 \right]$$

. Minimize the cross entropy of the n/w with respect to the target values.

$$- \sum_{d \in D} t_d \log o_d + (1 - t_d) \log (1 - o_d)$$

ii) Alternative error minimization procedures:

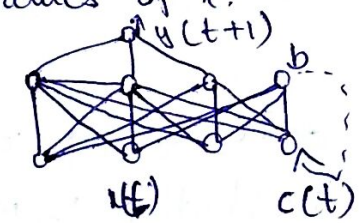
. One optimization method, known as line seach, involves a different approach to choosing the distance for the weight update.

. A second method, that builds on the idea of line seach, is called the conjugate gradient method. Here, a sequence of line searches is performed to search for a minimum in the error surface.

iii) Recurrent network:

. Recurrent n/w's are artificial neural n/w's that apply to time series data & that use o/p's of n/w units at time t as the i/p to other units at time t+1.

. One limitation of such a n/w is that the prediction of y(t+1) depends only on x(t) & cannot capture possible dependencies of y(t+1) on earlier values of x.



iv) Dynamically Modifying n/w structure:

. One idea is to begin with a n/w containing no hidden units, then grow the n/w as needed by adding hidden units until the training error is reduced to some acceptable level.

. The cascade-correlation algorithm is one such algorithm. Cascade-Correlations by constructing a n/w with no hidden units.

Evaluation Hypothesis

(1) Motivation:

. It is important to evaluate the performance of learned hypotheses as precisely as possible.

→ One reason is simply to understand whether to use the hypothesis

→ A second reason is that evaluating hypothesis is an integral

component of many learning methods.

Two key difficulties arise while learning a hypothesis and estima-ting its future accuracy given only a limited set of data:

i) Bias in the estimate: Estimator accuracy for future is poor which learned from training examples. To get unbiased estimate test the hypothesis on some set of examples chosen independently of the training examples.

ii) Variance in the estimate: With unbiased set measured accuracy vary from true accuracy. The smaller the set of test examples, the greater the expected variance.

(2) Estimation Hypothesis accuracy:-

. When evaluating a learned hypothesis we are most often intrested in estimation the accuracy with which it will classify future instances.

. Generally 2 Questions arise -

i) Given a hypothesis 'h' & a data sample containing 'n' ex. drawn at random according to the distribution D, what is the best estimate of the accuracy of 'h' over future instances drawn from the same distribution?

ii) What is the probable error in this accuracy estimate?

Sample error & true error:

Def: The sample error (denoted $error_S(h)$) of hypothesis 'h' w.r.t target func^n 'f' & data sample 'S' is

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

where 'n' is no. of examples in 'S', & the quantity $\delta(f(x), h(x))$ is '1' if $f(x) \neq h(x)$ & '0' otherwise.

Def: The true error (denoted $error_D(h)$) of hypothesis 'h' w.r.t target function 'f' & distribution 'D', is the probability that 'h' will misclassify an instance drawn at random according to D.

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

Here, the notation $\Pr_{x \in D}$ denotes that the probability is taken over the instance distribution 'D'.

Confidence intervals for discrete-valued hypothesis:-

Suppose we wish to estimate the true error for some discrete 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 & independent of 'h', according to the probability distribution V. ($n \geq 30$)

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

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

→ Given no other information, the most propable value of $error_D(h)$ is $error_S(h)$.

→ With approximately 95% probability, the true error $error_D(h)$ lies in the interval.

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

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

: 95% confidence interval.

Z_N different constant is used to calculate the N% confidence interval.

(3) Basics of sampling theory:-

i) Basic definitions:

. A random variable can be viewed as the name of an experiment with a probabilistic outcome. Its value is the outcome of the experiment.

. A probability distribution for a random variable 'Y' specifies the probability $P_r(Y=y_i)$ that 'Y' will take on the value 'y_i', for each possible value 'y_i'.

. The expected value, or mean, of a random variable Y is $E[Y] = \sum_i y_i P_r(Y=y_i)$. The symbol μ_Y is commonly used to represent E[Y].

. The variance of a random variable is $Var(Y) = E[(Y-\mu_Y)^2]$. The variance characterizes the width or dispersion of the distribution about its mean.

. The Binomial distribution gives the probability of observing 'r' heads in a series of 'n' independent coin tosses, if the probability of heads in a single toss is p.

. A N% confidence interval estimate for parameter 'p' is an interval that includes 'p' with probability N%.

ii) Error estimation & estimating binomial proportions:

→ The probability function $P(r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r}$

If the random variable 'X' follows a binomial distribution, then:

. The probability $P_r(X=r)$ that X will take on the value 'r' is given by P(r).

. The expected, or mean value of X, E[X], is $E(X) = np$

. The variance of X, Var(X), is $Var(X) = np(1-p)$

. The standard deviation of X, σ_X is $\sigma_X = \sqrt{np(1-p)}$

iii) Binomial distribution:

. The general setting to which the binomial distribution applies is:

i) There is a base, or underlying, experiment (e.g: toss of the coin) whose outcome can be described by a random variable, say Y. The random variable 'Y' can take on two possible values (e.g., Y=1 if heads, Y=0 if tails).

b) The probability that Y=1 on any single trial of the underlying experiment is given by some constant p, independent of the outcome of any other experiment. The probability that Y=0 is (1-p). Typically p, is not known in advance, & the problem is to estimate it.

iv) Mean and variance:

. Two properties of a random variable that are often of interest are its expected value (also called its mean value) & its variance.

. The expected value is the average of the values taken on by repeatedly sampling the random variable.

Def: Consider a random variable 'Y' that takes on the possible value $y_1, \ldots, y_n$.

. The expected value of Y, E(Y), is $E[Y] \equiv \sum_{i=1}^{n} y_i P_r(Y = y_i)$

Def: The variance of a random variable Y, var[Y], is

$$var[Y] \equiv E[(Y - E[Y])^2]$$

Def: The standard deviation of a random variable Y, σ_Y is

$$\sigma_Y \equiv \sqrt{E[(Y - E[Y])^2]}$$

By binomial distribution:

$$var[Y] = np(1-p) \qquad \sigma_Y = \sqrt{np(1-p)}$$

v) Estimators, Bias and Variance:

$$error_S(h) = \frac{r}{n} \qquad error_D(h) = p$$

. Where 'n' is no. of instances in sample 'S', 'r' is no. of instances from 'S' misclassified by h, & 'p' is probability of misclassifying a single instance drawn from D.

④ A general approach for deriving confidence intervals:-

. The general process includes the following steps:

i) Identify the underlying population parameter 'p' to be estimated, for example, $error_D(h)$.

ii) Define the estimator Y (e.g: $error_S(h)$). It is desirable to choose a minimum variance, unbiased estimator.

iii) Determine the probability distribution D_Y that governs the estimator Y, including its mean & variance.

iv) Determine the N% confidence interval by finding thresholds 'L' & 'U' such that N% of mass in the probability distribution D_Y falls b/w 'L' & 'U'.

Central Limit theorem: The central limit theorem states that the probability distribution governing $\bar{Y}_n$ approaches a normal distri-bution as $n \to \infty$, regardless of distribution that governs the underlying random variables Y_i. Furthermore, the mean of distributi-on governing $\bar{Y}_n$ approaches 'μ' & S.deviation approaches $\frac{\sigma}{\sqrt{n}}$

Theorem: Consider a set of independent, identically distributed random variables $Y_1 \ldots Y_n$ governed by an arbitrary probability distribution with mean 'μ' & finite variance σ^2. Define the sample mean, $\bar{Y}_n \equiv \frac{1}{n} \sum_{i=1}^{n} Y_i$.

There as $n \to \infty$, the distribution governing $\frac{\bar{Y}_n - \mu}{\frac{\sigma}{\sqrt{n}}}$ approaches a Normal distribution, with zero mean & S.deviation = 1.

. The central limit theorem is very useful fact, becoz it implies that whenever we define an estimator i.e the mean of some sample (e.g. $error_S(h)$ is mean error), the distribution governing this estimator can be approximated by normal distribution for large 'n'.

(5) Difference in error of two hypotheses:

. Consider the case where we have two hypotheses 'h_1' and 'h_2' for some discrete valued target function. Hypothesis 'h_1' has been tested on a sample 'S_1' containing 'n_1' randomly drawn examples, and 'h_2' has been tested on an independent sample 'S_2' containing 'n_2' examples drawn from the same distribution.

$$d \equiv error_D(h_1) - error_D(h_2)$$

$$\hat{d} \equiv error_{S_1}(h_1) - error_{S_2}(h_2)$$

$\hat{d}$ gives an unbiased estimate of d, that is $E[\hat{d}] = d$.

. For large n_1 & n_2 (e.g.: both > 30), both $error_{S_1}(h_1)$ and $error_{S_2}(h_2)$ follow distribution that are approximately normal. Because the difference of two normal distributions is also a Normal distribution, $\hat{d}$ will also follow a distribution i.e approximately normal, with mean d.

. Variance is the sum of the variances of errors =

$$\sigma_{\hat{d}}^2 \approx \frac{error_{S_1}(h_1)(1 - error_{S_1}(h_1))}{n_1} + \frac{error_{S_2}(h_2)(1 - error_{S_2}(h_2))}{n_2} \quad \text{---(1)}$$

. N% confindence interval estimate for d is $\hat{d} \pm z_N \sigma$

$$\hat{d} \pm z_N \sqrt{\frac{error_{S_1}(h_1)(1 - error_{S_1}(h_1))}{n_1} + \frac{error_{S_2}(h_2)(1 - error_{S_2}(h_2))}{n_2}}$$

where z_N is constant.

Hypothesis Testing:

. Suppose we measure the sample errors for 'h_1' & 'h_2' using two independent samples 'S_1' & 'S_2' of size 100 & find that $error_{S_1}(h_1) = 0.30$ and $error_{S_2}(h_2) = 0.20$, hence the observed difference is $\hat{d} = 0.10$.

. Note the probability $P_r(d > 0)$ is equal to the probability

that $\hat{d}$ has not overestimated d' by more than 0.10, Put another way, this is the probability that $\hat{d}$ falls into the one-sided interval $\hat{d} < d + 0.10$. since d' is the mean of the distribution governing $\hat{d}$, we can equivalently express this one-sided interval as $\hat{d} < \mu_{\hat{d}} + 0.10$. ~~since d' is mean of the distribution governing~~

Using eq① we find that $\sigma_{\hat{d}} \approx 0.061$, so we can re-express the interval as approximately $\hat{d} < \mu_{\hat{d}} + 1.64\sigma_{\hat{d}}$

⑥ Comparing learning algorithms:-

. Comparing the performance of 2 learning algorithms L_A & L_B rather than 2 specific hypothes.

. To determine 1 method is "on average" is to consider the relative performance of 2 algorithms averaged over all the training sets of size 'n' over distribution 'D'.

. Other one is expected value of the difference in their errors.

$$E_{S \subset D}[error_D(L_A(S)) - error_D(L_B(S))]$$

where $L(S)$ denotes the hypothesis o/p by learning method 'L' when given the sample 'S' of training data & where the subscript $S \subset D$ indicates that the expected value is taken over samples 'S' drawn according to the underlying instance distribution D.

. Limited sample D_0, divides into training set S_0, Test set T_0. Algorithms L_A and L_B.

. Quantity is $error_{T_0}(L_A(S_0)) - error_{T_0}(L_B(S_0))$

. Do into disjoint training sets and test sets.

i) Partition the available data D_0 into 'k' disjoint subsets $T_1, T_2, \ldots, T_k$ of equal size, where this size is atleast 30.

ii) For 'i' from 1 to k, do
use 'T_i' for the test set, & the remaining data for training set 'S_i'

$$S_i \leftarrow \{D_0 - T_i\}$$
$$h_A \leftarrow L_A(S_i)$$
$$h_B \leftarrow L_B(S_i)$$
$$\delta_i \leftarrow error_{T_i}(h_A) - error_{T_i}(h_B)$$

iii) Return the value $\bar{\delta}$, where

$$\bar{\delta} \equiv \frac{1}{k} \sum_{i=1}^{k} \delta_i$$

. One way to improve on the estimator is to repeatedly partition the data D_0 into disjoint training & test sets & to take the mean of the test set errors for these different experiments.

. Tests where the hypotheses are evaluated over identical samples are called paired tests.

a) Paired t Tests:

. Consider the following estimation problem:

→ We are given the observed values of a set of independent, identically distributed random variables $Y_1, Y_2, \ldots, Y_k$

→ We wish to estimate the mean 'μ' of the probability distribution governing these Y_i.

→ The estimator we will use is the sample mean $\bar{Y}$

$$\bar{Y} \equiv \frac{1}{k} \sum_{i=1}^{k} Y_i$$

. This problem of estimating the distribution mean 'μ' based on the sample mean $\bar{Y}$ is quite general.

. The test 't' applies, in which the task is to estimate the sample mean of a collection of independent, identically & normally distributed random variables.

. Confidence interval => $\mu = \bar{Y} \pm t_{N,k-1} s_{\bar{Y}}$

where $s_{\bar{Y}}$ is estimated s. deviation of sample mean

$$s_{\bar{Y}} \equiv \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^{k} (Y_i - \bar{Y})^2}$$

where $t_{N,k-1}$ is constant analogous to our earlier z_N.

b) Practical considerations:

. In practice, the problem is that the way to generate new 'Si' is to resample Do, dividing it into training & test sets in different ways. The 'δi' are not independent of one another in this case, because they are based on overlapping sets of training examples drawn from the limited subset Do of data, rather than full distribution 'D'.

. k-fold method in which Do is partitioned into k disjoint, equal-sized subsets. In this k-fold approach, each example from Do is used exactly once in a test tset, & k-1 times in a training set 'A'.

. In contrast, the k-fold method is limited by total no. of examples, by the use of each example only once in a test set, & by our desire to use samples of size atleast 30.

. In contraast, the test sets generated by k-fold cross validat -ion are independent becoz each instance is included in only one test tset.