MODULE - 3

SUPPORT VECTOR MACHINES AND RADIAL BASIS FUNCTIONS

Learning from Examples, Statistical Learning Theory, Support Vector Machines, SVM application to Image Classification, Radial Basis Function Regularization theory, Generalized RBF Networks, Learning in RBFNs, RBF application to face recognition.

Learning from examples and generalisation :

Supervised learning can play a key role in learning from examples. From this algorithm, useful information can be easily extracted from large datasets. The problem of learning from examples consecutively involves approximating functions from sparse and noisy data.

In supervised learning, netwrok is trained on a dataset of the form, $T = \{x_k , d_k \}^{Q}_{k=1}$ from k=1 to Q. Here training involves searching the relationship that underlie the data in such a way that machine can predict that is

$Y_{predicted}$ = f(unseen)

or machines should be made capable to generalise. It is observed that using MLP (multilayer perceptrons) with sufficient number of hidden neurons , it is possible to approximate a given function to any arbitratry degree of accuracy. When the training reduces MSE (mean squared error) over the training set, whose sufficiently low level and the network can predict output for the unseen inputs with considerable accuracy, we say that the network provides valid **generalizations** to unseen patterns (but this is not guaranteed)

The problem of **approximation function** involves fitting a set of weights of a neural network to a given training data. A network with too many weights (or free parameters) will fit the training data very accurately and will generalise poorly to unseen data.

On the other hand, the network with too few weights will also give poor generalizations as it will be unable to approximate even the training data. It is important to strike a balance between the number of training examples used to train a model and inherent complexity of a model as measured by the number of free parameters of the network.

The generalization ability of a machine is closely related to the capacity of the machine, in terms of familiar functions it can represent, as well as the dataset that is used for training.

When the space of representative functions and their capacity is large and when the dataset is small the model tends to overfit and generalize poorly. Given finite training data , one has to achieve a correct balance between accuracy and the capacity of the machine to learn a dataset without error.

Vapnik's theory centers around capacity control in which lies the key to good generalization. Vapnik's theory is **a form of computational learning theory, which attempts to explain the learning process from a statistical point of view**.

For a **simple model**, the variance between the slopes drawn from w_0,w_1 in the form of y = mx+c will always have low variance with respect to other slopes drawn from different batches of the training input and the functions computed will be different. The **true function** is a function that is ideal where the weights are tuned approximately or exactly and the function matches the input training example. (In other words, it's a function drawn from the desired outcome). The variance between the approximate functions of different training examples is very low, but the bias between the approximate function and the true function is greater. So **simple models have low variance and high bias**.

In case of a **complex model**, where the functions for each batch of the training example fine tune itself and fit the model well during the training phase will have a high variance with respect to other approximate functions that also is finely tuned to match the true function. But these models will have low bias as the approximate function is almost close or has unvarying distance compared to the true function. **Complex models have high variance and low bias**

The proper approach to achieve a good generalization is to trade off between achieving a good fit to the training data which maintains the fitted function to be smooth enough by limiting the complexity of the model, the key to good generalization lies in an understanding of **bias-variance dilemma**, or the **B-V dilemma**.

For the true function y = f(x), if F(x,w) is the approximated function modelled using the neural network then, the Bias term is given by

$$bias = E [F(x,w)] - f(x)$$

I.e esimated difference between the approximation function and true function. The expectation of the approximation function and the variance is given by,

$$variance = E[(F(x,w)) - F(x,w)]^2$$

where E[F(x,w)] is the average function value computed after estimating different approximate functions under different training datasets taken from the same process.

It can be shown that MSE (mean squared error)

$$MSE = E[(f(x)—F(x,w))]^2$$

or it is the bias² + variance.

1. In the context of statistical learning theory, write a note on empirical risk minimization defining various loss functions, structural risk minimization and VC dimension.

STATISTICAL LEARNING THEORY

Learning from the example is a stochastic process with training data being drawn from the two sets of variables input x_k and response d_k.

Relationship between X and D is probabilistic.The element from X defines a probability distribution on D.

Probability distribution P(x,d) defined on X-D space determines the probability of observing a training data point (x_k,d_k), the training data $T=\{x_k,d_k\}$ from k=1 to Q is actually generated by sampling the X-D space Q times in accordance with the distribution P(X,D).

Now learning from sample set is like searching for the approximate estimator function

F f: X->D

In order to succesfully solve a regression or classification task, learning machine learn an approximating function f(x,w) which is function of x,w

Empirical Risk Minimization :

In empirical risk minimization or (ERM), the expected value of loss is defined by the risk function that employs loss function, L.

$L(d,f(x,w)) = [d-f(x,w)]^2$

measures the error between desired value 'd' and predicted value f(x,w). When the joint probability distribution P(x,d) is known, the expected risk is defined as

$R(f) = E [L(d, f(x,w))]$

$= \int L(d, f(x,w))\, D\, P(x,d)$ (D = derivative, P means probability of x,d)

Now we denote f_0 as an optimal function that minimizes expected risk at optimal weight w_o as

$f_0 = f(x,w_0) = \text{argmin } R(f)$

Here f_0 is the ideal estimator (target function). Since P(x,d) is not known, f_0 cannot be found in practice. To solve this problem Vapnik suggested the ERM (Empirical Risk Minimization) principle which is an inductive principle that can be used to train the machine using a limited number of data samples at hand.

ERM generates a stochastic estimation of R using the training set T and this approximation is called as **Empirical Risk** and is denoted as

$$Re(f) = \frac{1}{Q} \sum_{k=1}^{Q} L(d_k , f(x_k,w))$$

As Q approaches infinity,

$$\lim_{Q \to \infty} Re(f) = R(f)$$

This means that as more samples are added, the approximation is better and the empirical risk is equal to the expected risk because now the machine has more details and number of samples to train from.

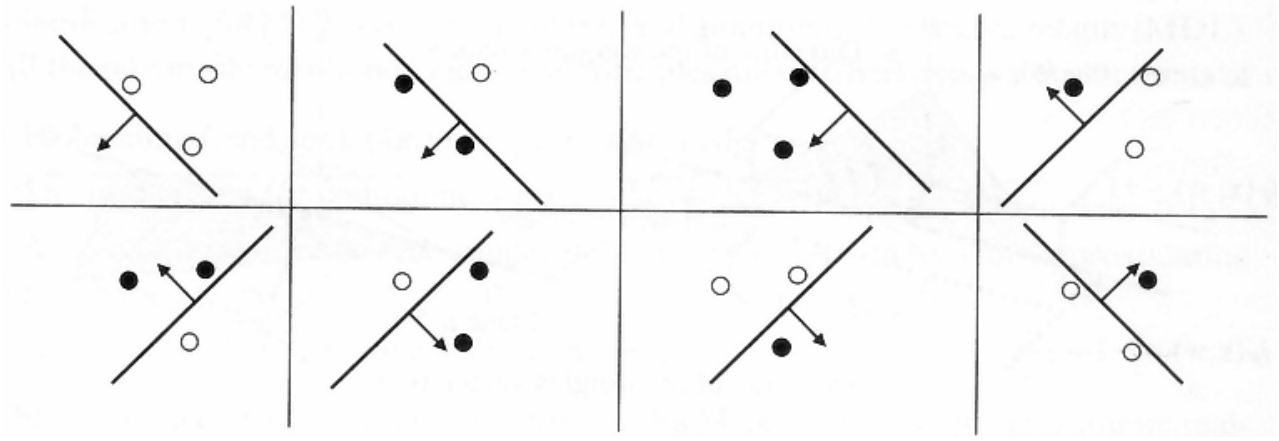
VAPNIK - CHERVONENKIS DIMENSION (VC DIMENSION):

Consider a set of functions F = {f(x,w)} that maps points from n dimension to a set {0,1} or {-1,1} where F is an **indicator function**. If one considers Q points each of these can be assigned a class of 0 or 1 randomly and lure Q points can be labelled in 2^Q different ways.

If the set of indicator function 'f' can correctly classify each of the possible 2^Q labellings then we say that set of points is **shattered** by F.

The VC dimensions of a set of functions essentially quantifies the intrinsic capacity of that set measured in terms of the largest number of data points that can be shattered by the set in question.

VC dimension (h) for a set of functions (F) is defined as the largest number of points that can be shattered by F. VC dimenesion talks about the complexity of the model. Here 'shattered' means classified/partitioned.



For 8 possible labelling of 3 data points, a straight line can correctly separate all 8 configurations as shown in the picture. If we increase the no. of points to 4, there are 16 possible ways to label the points and two of these 16 labellings cannot be correctly classified by a linear decision boundary.

Since the largest set of points in R^2 (2 dimension) that can be shattered is 3 we say that the VC dimension of the set of oriented straight lines is R^2 is 3.

The VC dimenstion of (h) = 3, thus provides us with the measure of capacity of linear decision function in space R^2 under consideration so in general

n=1 h=2 n=2 h=3

n= n , h= n+1

that is almost n+1 points in R^n can be shattered by a hyperplane. **VC dimension is equal to no. of free parameters (weights)**.

Structural Risk minimization (SRM) :

When a machine is trained on a given training set, the approximations generated are naturally biased towards those data points. It is necessary to ensure that the model chosen for representation of the underlying function has a complexity (capacity) that matches the dataset in question.

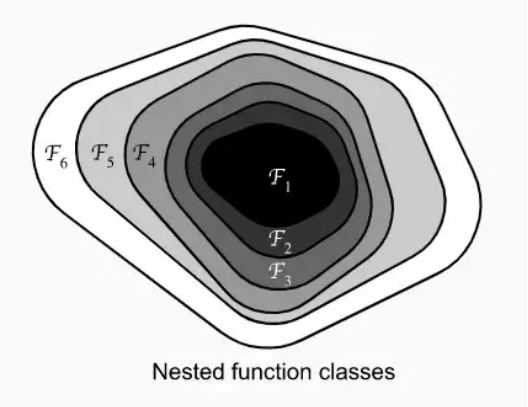
For binary classification, loss function which take on values either 0 or 1 for some $0 \leq \eta \leq 1$, the following bounded hlds with probability at least 1-η

$$R(f) \leq Re(f) + \sqrt{\frac{h(ln\frac{2Q}{h}+1) - ln(\frac{n}{4})}{Q}} \quad \text{—(1)}$$

where h=VC dimension , Q = no. of training set sample, Re(f) = ERM, R(f) = generalization error.

One of the important consequences of VC theory is that the difference between empirical and expected risk can be bounded in terms of VC dimension as in the eqn 1. In inequality eqn 1 suggests the way to achieve good generalisation, minimise the combination of empirical risk and complexity of the hypothesis space.

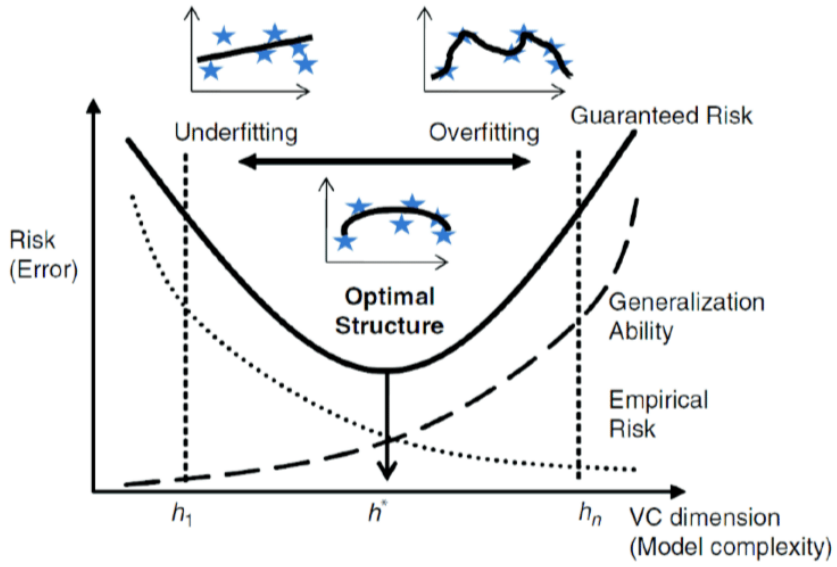
Since the space of duntions F where the loss L is defined is usually very large, the idea is to restrict the focus of learning to a smaller space. SRM defines nested sequences of functions of increasing complexity.



Nested function classes

where F_i might represent a multilayer feedforward neural network with i hidden neurons or a set of polynomials in one variable of degree i .

The nesting $\subset$ implies that each successive hypothesis space contains all proceeding less complex functions. Each hypothesis space has its own VC dimension.



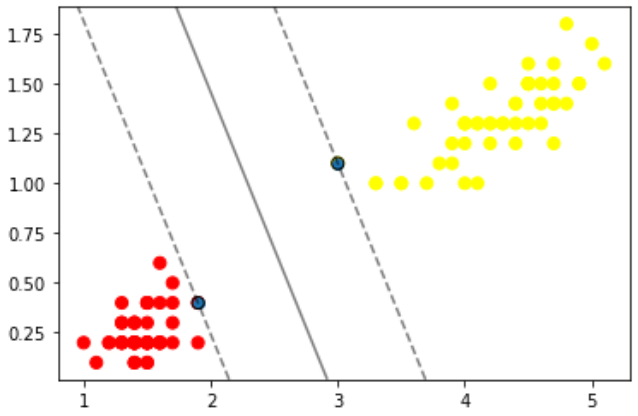
Expected risk $\leq$ empirical risk + VC confidence

$R(f) \leq Re(f) + f(Q,h , \eta)$

For increase in complexity or added free parameters, the model learns better and Re(f) reduces. The aim should be to find that balance where the model is neither too complex nor too simple.

2. Explain the SVM algorithm design objective, its advantages and derive the expression for total maximum margin in case of linearly separable patterns

SUPPORT VECTOR MACHINES :



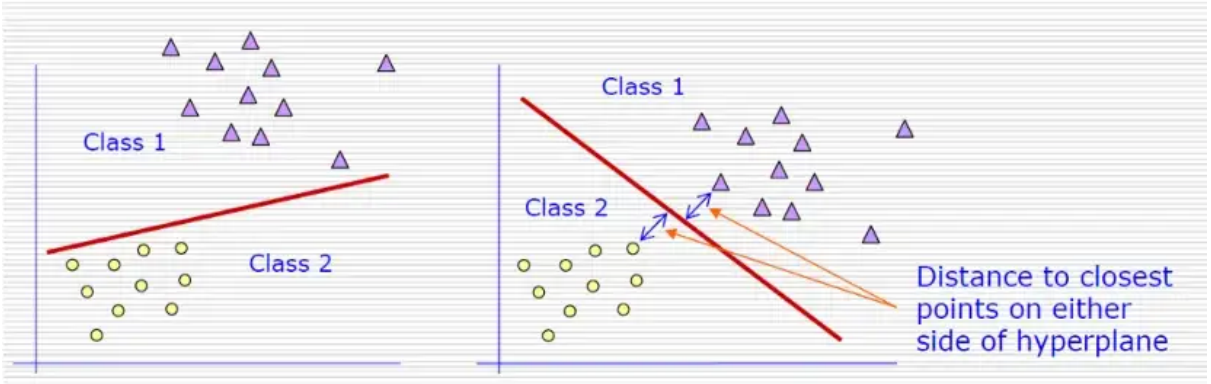
SVM has a firm frounding in the VC theory of statistical learning and essentially implements Structural Risk Management (SRM) .

For a linearly separable classes, the maximum margin hyperplane is given by assuming,

Let T = $\{x_k, d_k\}$ from k=1 to Q. where $x_i \in R^n$ (n dimension) and $d_k = \{-1,+1\}$

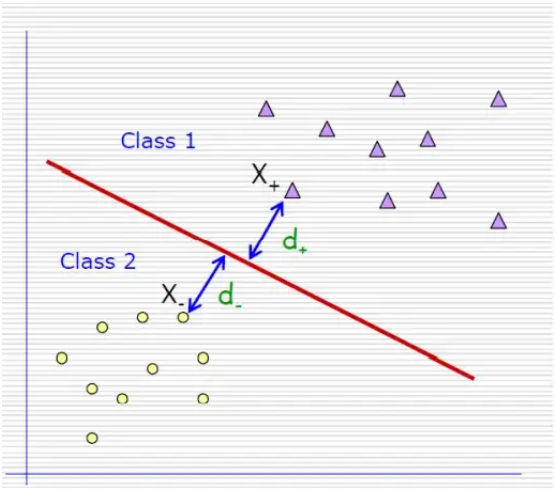
An indicator function B (bipolar signal function) is taken.

Considering samples of T that can be classified into one of the two classes C1 and C2.



Where C1 = +ve sample with $d_k = +1$ (d_+)

And C2 = -ve sample with $d_k = -1$ (d_-)



Note that linear separability implies that we can find an oriented hyperplane defined by a set of weight 'w' and bias w_0 which separates the positive samples from negative ones. For all the points on hyperplane, the equation of the hyperplane is taken as ,

$Wx + w_0 = 0$

Here, hypothesis space under consideration is a set of funtions $f(X,w_0,w) = sign(wx + w_0)$

There exists multiple solutions w = ws for which the classes C1 and C2 are linearly separable . In addition to achieving **correct classification**, the second objective in SVM is to **maximize the margins** from the separating hyperplane to the nearest +ve or -ve data points.

Definitions of margins :

Let d_+ and d_- be the perpendicular distance from the separating hyperplane to the closest data points taken from the classes C1 and C2 respectively. The total margin is the distance,

$d_+ + d_-$

Here the idea of margin tightens the classification criteria for the correct classification.

From the original formulation of hyperplane,

$wx+w_0 > 0$ $d_i = +1$ classified as class c1

$wx+w_0 < 0$ $d_i = -1$ classified as class c2

In SVM , the reformulated hyperplane equation,

$wx+w_0 > \Delta$ $d_i = +1$ classified as class c1

$wx+w_0 < \Delta$ $d_i = -1$ classified as class c2

The reformation is performed by introducing +ve constant delta Δ . This reformation constraints the hyperplane to a position such that for the closest data points X_+ and X_- ,

$wx+w_0 = +\Delta$

$wx+w_0 = -\Delta$

Depending upon which side of the hyperplane they are on. Without loss of generality we may set Δ =1 by choosing w and w_0.

The decision region transforms to,

$wx+w_0 = +1$ $d_i = +1$ classified as class c1

$wx+w_0 = -1$ $d_i = -1$ classified as class c2

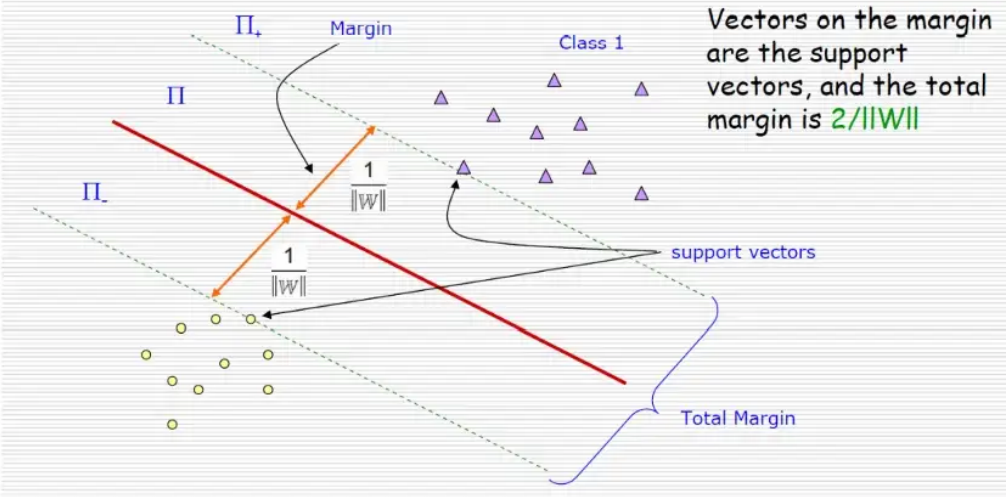
In a more comapct form, above two equations can be written as ,

$d_i (wx + w_0) -1 = 0$

taking assumptions.

$d_i (wx + w_0) = 1$ for $X = X_+$, $d_i = +1$

$d_i (wx + w_0) = 1$ for $X = X_-$, $d_i = -1$



Let X_{+1} be the datapoint from C1 closest to hyperplane Π and XΠ be the unique point on Π that is closest to X_{+1} ,

In SVM, the direction of w is normal to Π and here we are interested in maximizing

$$\left|\left|X_{+} - X_{\Pi}\right|\right|$$

From the defining equation of Π ,

$wx_{+} + w_0 = +1$

$wx_{\Pi} + w_0 = 0$

Subtracting these equations, we get

$w(x_{+} - x_{\Pi}) = 1$

This is further simplified as,

$$W \left(\left|\left|X_{+} - X_{\Pi}\right|\right| * \frac{W}{||W||} \right) = 1$$

Where $\left|\left|X_{+} - X_{\Pi}\right|\right| = \frac{1}{||W||}$

Similarly X_{-} to the point X_{Π} on the hyperplane is

$$\left|\left|X_{-} - X_{\Pi}\right|\right| = \frac{1}{||W||}$$

So the net margin $d_{+} + d_{-}$ is

$$\frac{1}{||W||} + \frac{1}{||W||}$$

$$= \frac{2}{||W||}$$

(*Here* $wx + w_0$ *has* w *= orientation of the hyperplane,* w_0 *= position of the hyperplane (+ve or -ve). These will control the hyperplane*)

3. SVM application to Image Classification.

SVM has a high generalization performance even in the absence of apriori knowledge. Since images will fall into more than one class one needs to adapt the classical binary SVM algorithm to work for the multi class case. Here C different hyperplanes are constructed for C different classes where each hyperplane separates the class in question from all other classes. Therefore, there are n different decision functions $y_j(X)$ where $1<j<C$ that get generated. These take the form,

$y_j(X) = \text{sign}(d_i K(X,X_i)+b)$. The class c_j assigned to point X is then

$J = \text{argmax } y_j(X)$

(i) Description of dataset : The data under consideration comprises images from the Corel Stock Photo collection with some 200 classes each with 100 images. Two databases were derived from the original collection- first was corel14 with 14 classes and 1400 images, second - Corel7 with seven classes and 2670 images.

Image classification in the present application is based on the idea of a color histogram. A color is represented by a point in a 3D color space- hue, saturation, luminance value (HSV) which is in direct correspondence with RGB space. Sixteen bits per color component are selected yielding a dimension of $16^3 = 4096$ for a histogram. For each input we have 96 x 64 pixels with 3 color components larger than 4096.

(ii) Selection of kernel : this is critical to the problem of maximizing performance of an SVM. the kernels employed are:

Polynomial : $K_p (X,Y) = (X.Y + 1)^p$

Gaussian : $K_g (X,Y) = \exp(-||X-Y ||^2)$.

Classifiers using g will have Gaussian basis functions classifiers and p will have polynomial functions. For gaussian functions, SVM acts like a radial basis function where SVM corresponds to its centers and weights and bias are chosen automatically. These yield excellent results when trained with non SVM methods.

(iii) Experiment :

Corel14 samples were divided into 924 training and 476 test samples. For Corel7, divided intp 1375 training and test samples each. For all kernels except the linear one, full separability was enforced on the training set. RBF kernels bring the error down by 30% percent. This is due to the excellent generalizability of SVMs.

Using a specific renormalization parameter value, support vectors obtained from the training set were tested on a validation set. The lamda renormalization parameter yielded best results. The error rate of linear SVMs is 30 percent higher than RBF based SVM but computational requirements are magnitude less than the RBF based SVM. This makes linear SVM attractive for such an application and used for histogram based image classification.

4. What are RBFs and Describe how RBF are utilized for interpolator

RADIAL BASIS FUNCTION :

Radial basis function is derived from Cover's Theorem of Separability of Patterns :
A complex pattern classification problem, cast in a high dimensional space non linearly is more likely to be linearly separabale then in a low dimensional space provided that the space is not densely populated.

Hidden neuron activations in RBFN are computed using an exponential of a distance measure (Euclidean distance) between input vectors and prototype vectors associated with hidden neurons.

Architecture of RBFN in the context of interpolation theory :
RBFN was originally introduced for the purpose of interpolation of data points on a finite training set T = $\{X_k , d_k\}_{k=1}^{Q}$. Then solving the exact interpolation problem, we have to search for a map "f" such that $f(x) = d_k$ k=1 to Q.

For the purpose of performing interpolation RBF assumes a set of Q basis functions :

$$\Phi(\left|\left| X - X_i \right|\right|)$$

Which are non linear and whose arguments involves a Euclidean distance measure || X - X_i ll , the map

$$f(X) = \sum_{i=1}^{Q} w_i \Phi(\left|\left| X - X_i \right|\right|)$$

Substitute the condition for exact interpolation

$X = x_k$

$$f(x_k) = \sum_{i=1}^{Q} w_i \Phi(\left|\left| X - X_i \right|\right|)$$

Equal to dk (because $f(x_k) = d_k$)

Let $D = [d_1 , d_2 , \ldots . d_Q]^T$

$W = [W_1 , W_2 , \ldots . W_Q]^T$

Then RBF in matrix form is written as :

$$\begin{bmatrix} \Phi(\left|\left| X_1 - X_1 \right|\right|) & \ldots\ldots\ldots\ldots & \Phi(\left|\left| X_1 - X_Q \right|\right|) \\ . & & \vdots \\ . & & \vdots \\ . & & \vdots \\ \Phi(\left|\left| X_1 - X_1 \right|\right|) & \ldots\ldots\ldots\ldots & \Phi(\left|\left| X_1 - X_Q \right|\right|) \end{bmatrix}$$

QxQ matrix

Therefore we can summarize that

$D = \Phi^T W$

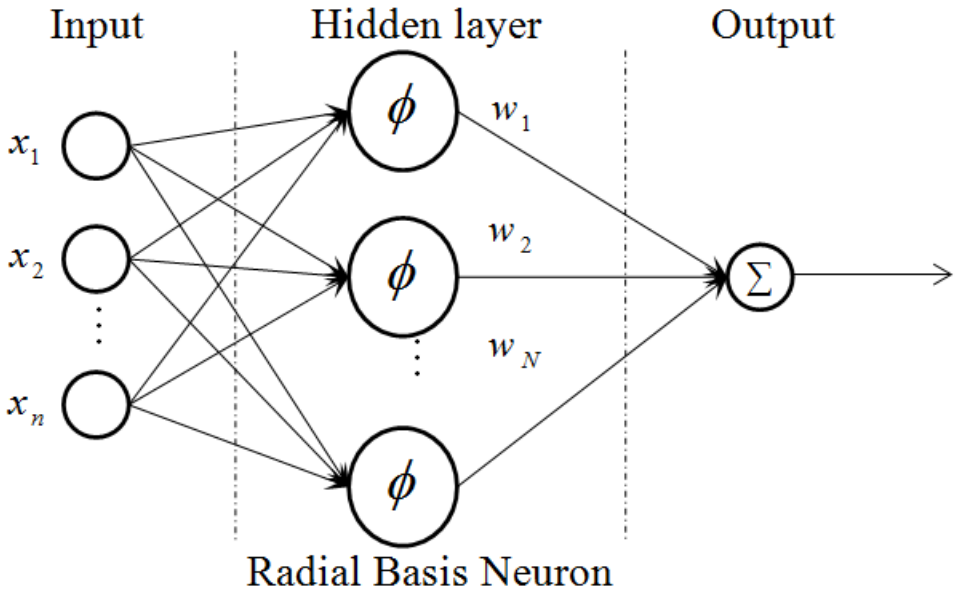
$W = \Phi^{-1} D$ where Φ is non singular

There are 3 functions for RBF used in ML :

1. Gaussian functions
2. Multiquadrics
3. Inverse multiquadrics

Architecture of RBF :

The hidden layers are in hidden space where nonlinear transformation happens.



5. Derive the weight computation matrix for designing an RBF based interpolator when fewer basis functions are used as compared to the number of training samples(Q).

Interpolation with fewer than Q basis function :

To formalise interpolation with fewer than Q basis function, consider interpolating a set of basis function with a number q<Q (q= no. of basis functions, Q = no. of training patterns)

The original approximation function :

$$f(X) = \sum_{i=1}^{Q} w_i \Phi(\| X - X_i \|) \quad \text{transforms to}$$

$$f(X) = \sum_{i=1}^{q} w_i \Phi(\| X - X_i \|)$$

To solve the approximation problem in a least square sense ,

$$\varepsilon = \frac{1}{2} \sum_{k=1}^{Q} (d_k - \sum_{i=1}^{Q} w_i \Phi(\left|\left| X - X_i \right|\right|))^2$$

Which compiles the square error at output neuron summing over all data points. To find the solution vector for w ,

$$\varepsilon = \frac{1}{2} \sum_{k=1}^{Q} (d_k - \sum_{i=1}^{Q} w_i \Phi(\left|\left| X - X_i \right|\right|))^2 - 2\, d_k\, w_i \Phi(\left|\left| X - X_i \right|\right|))$$

$$\frac{\partial \varepsilon}{\partial w_i} = \sum_{k=1}^{Q} \sum_{i=1}^{q} \Phi(\left|\left| X - X_i \right|\right|)^2 - d_k \Phi(\left|\left| X - X_i \right|\right|)$$

$$= \sum_{k=1}^{Q} \sum_{i=1}^{q} \Phi^2_{ki}\, w_i = \sum_{k=1}^{Q} d_k\, \Phi_{ki}$$

In matrix form

$$\Phi(w) = \Phi^T D \quad — (1)$$

Where

$$\Phi = \begin{bmatrix} \Phi_{11} ..... \Phi_{1q} \end{bmatrix}$$

$$\begin{bmatrix} \Phi_{Q1} ..... \Phi_{Qq} \end{bmatrix}$$

Equation can be further simplified to

$$W = (\Phi^T \Phi)^{-1}\, \Phi^T D$$

(By taking pseudo inverse of $\Phi^T \Phi$)

6. Describing regularization theory explains the weight computation procedure utilizing Green's function.

We define an ill posed problem as a problem that does not exist, not unique or continuously dependent on data.

Regularisation is an approximation solution for an ill posed problem. It requires constraints that impose a restriction on solution space. It imposes a smoothness constraint on an approximating set of functions.

Smoothness is necessary for the represented function to be robust to noise.

Regularization replaces standard error minimization by minimization of a regularization risk functional that comprises two terms.

First term is an error function- difference between training and solution.

Second term is a smoothness function that provides the smoothness of solution and is constraint imposed.

In regularization, we need to find the function close to data points in T and make it smooth.

$$R_r(f) = \sum_{i=1}^{Q} (d_{ki} - f(X_i))^2 - \lambda f[f]$$

Where $\lambda f[f] = ||p_f||^2$ = smoothness function

And p is the linear differential operator.

$$R_r(f) = \sum_{i=1}^{Q} (d_{ki} - f(X_i))^2 - \lambda ||p_f||^2 \quad \text{---(1)}$$

λ is the regularization parameter and controls the tradeoff between the first (closeness) and second term.

The smoothness function stabilises the network.

We introduce Green's functions G(x,y) which is symmetric and singular (x=y)

$$f(x) = \sum_{i=1}^{Q} w_i G(X,X_i)$$

Substituting this f(x) in equation (1) ,

$$R_r(f) = \sum_{i=1}^{Q} (d_{ki} - w_i G(X,X_i))^2 - \lambda ||p_f||^2$$

To solve for weights ,

we take $wi = \frac{1}{\lambda}(d_i - f(X_i)) \; k=1,..Q$

$$f(x) = \sum_{i=1}^{Q} w_i G(X,X_i) \text{ where } wi = \frac{1}{\lambda}(d_i - f(X_i)) \; k=1,..Q$$

To convert this to vector matrix form, we take

$F = (f(X_1),..f(X_Q))^T$

$W = (w_1 ,.. w_Q)^T$

$D = (d_1 \ldots d_Q)^T$

$$G = \begin{pmatrix} G(X_1, X_1) & \ldots & G(X_1,X_Q) \\ \vdots & & \vdots \\ G(X_Q, X_1) & \ldots & G(X_Q,X_Q) \end{pmatrix}$$

Where G is the Green's matrix anf is symmetric as $G = G^T$

So, in vector form,

$W = \frac{1}{\lambda}(D - F')$

$F' = GW$

7. Explain the methods employed to learn the centers and spreads of basis function networks

Learning in RBFNs :

Generalised RBF networks not only use fewer than Q basis functions (Q being no. of data points) but also permits the centres and spread of individual permits to differ from centres. In RBF, the weights are determined in case when the no. of basis functions q is less than Q no. of points and where the spreads and centres of those basis functions are no longer restricted to be at data point. Now we review some of the methods employed to learn the centres and spread of these basis functions. We assume that no. of basis functions are fixed and the central problem of determining centres and spreads of individual basis functions.

1) Random subset selection : select subset q from Q data points and fix q as centre of gaussian basis function. In semi random selection a basis function is usually placed at every r th datapoint. The spreads are assumed to be a function of maximum distance between chosen centres and 'q'. In other words, assuming that gaussian basis functions, centres are selected randomly at μ_i. We define the max distance α between any of the chosen centres as

$$\alpha = \max \left|\left| \mu_i - \mu_j \right|\right| \quad 1 \leq i \ , j \leq q$$

$$\text{Spread factor } \sigma = \frac{\alpha}{\sqrt{2q}}$$

2) Hybrid learning procedure : in hybrid learning procedure, we first determine the centers of basis fucntion using a clustering algorithm, such as kmeans clustering algorithm and then tune the hidden to output weights using LMS procedure . K Means clustering algorithm places centers if RBF in those regions of input space where the significant data are present, Given a training set T = $\{x_k , d_k\}$, Kmeans clustering algorithm functions are defined below

   1. Initialisation : decide 'q ' (no. of clusters) and randomize centre $(\mu_i)^q_{i=1}$. Then fix the learning rate $0< \eta < 1$.
   2. Sampling : select input vector X_k randomly find the index j of the closest centre

$$J = \text{argmin} \left|\left| X_k - \mu_i^k \right|\right| \text{ where } i \leq i \leq q$$

   3. Update only the closest centre using the centre update equation :

$$\mu_j^{k+1} = \mu_j^k + \eta(X_k - \mu_j^k)$$

Until no perceptible change in centre and spread factor σ is computed :

$$\sigma = \sqrt{\frac{\sum(X-\mu_i)^2}{Q}}$$

3) Supervised learning of centres :
   In this technique, to tune centres with spread factor gradient descent technique employed and gradient descent requires cost function to be minimised .

The cost function is defined as

$$\varepsilon = \frac{1}{2} \sum_{k=1}^{Q} (d_k - \sum_{i=1}^{Q} w_i G(\left\| X_k - \mu_i \right\|))^2$$

The free parameters include μ_1 to μ_q and the spread in the form of covariance matrix K_1 to K_Q and the weights W_1 to W_q .

In the above expression note that error function is convex with respect to μ and σ .

Centres and spreads which affects the error in a nonlinear fashion . Therefore there will be many local minima and error backpropagation algorithm will guarantee convergence only to the closest local minima.

$$\frac{\partial \varepsilon}{\partial w_i} = \sum_{j=1}^{Q} e_j^k G(\left\| X_j - \mu_j^k \right\|) \quad \text{i=1 to q}$$

$$\frac{\partial \varepsilon}{\partial \mu_i^k} = \partial w_i^k \sum_{j=1}^{Q} e_j^k \; G'(\left\| X_j - \mu_j^k \right\|) \; K^{-1|k}(\left\| X_j - \mu_j^k \right\|)$$

$$\frac{\partial \varepsilon}{\partial K^{-1|k}} = - w_i^k \sum_{j=1}^{Q} e_j^k \; G'(\left\| X_j - \mu_j^k \right\|) \quad (X_j - \mu_j^k) \; (X_j - \mu_j^k)^T$$

where $K^{-1|k}$ K^{-1} is the spread matrix of covariance matrix K and $G'(\left\| X_j - \mu_j^k \right\|)$ is the differentiation of rbf function . Finally the weight update equation is given by

$$W_i^{k+1} = w_i^k - \eta \frac{\partial \varepsilon}{\partial w_j^k}$$

Centre update equation is given as :

$$\mu_i^{k+1} = \mu_i^k - \eta \frac{\partial \varepsilon}{\partial \mu_i^k}$$

Spread factor :

$$K^{-1|k+1} = K^{-1|k} - \eta \frac{\partial \varepsilon}{\partial K^{-1|k}}$$

So for tuning the weights, these parameters need to be tuned.

8. Explain about a generalized RBF network.

GENERALISED RBFN :

When the training data is usually corrupted by noise, we can apply a regularization approach to overcome the problem. The regularization network described previously uses one basis function for each datapoint (this is in practical)

The only way to solve this problem is to choose no. of basis function considerably less than no. of datapoints.

In generalised RBFN along with the regularisation condition, fewer than Q basis functions are considered.

To reduce the no. of basis functions non data centres are used.

(centres q less than Q data points)

Now the interpolation function transforms to

$$f_a(X) = \sum_{i=1}^{q} w_i \, G(\left\| X - \mu_i \right\|)$$

Here centres μ_i and spread σ_i are determined using some learning technique or heuristic approach. Now the risk function that has to be minimised in GRBFN is

$$R_r(f_a) = \sum_{i=1}^{Q} [\, d_k - \sum_{i=1}^{q} w_i \, G(\left\| X_k - \mu_i \right\|) \,]^2 + \lambda \left\| P f_a \right\|^2$$

—(1)

Here f_a is the function we're trying to estimate , $G(\left\| X_k - \mu_i \right\|)$ is the general basis function and $\lambda \left\| P f_a \right\|^2$ is the generalisation term.

These can be represented in matrix forms.

$$R_r(f_a) = \text{II D - GW II}^2 + \lambda \, W^T G_0 W$$

To find the weight W,

$$\frac{\partial R(fa)}{\partial W} = 0$$

$$2(D-GW) + (-G) = 2\lambda G_o W$$
$$W = (G^T G + \lambda G_0)^{-1} \; G^T D$$

9. Write about RBFN application in face recognition.

Automatic face recognition from both still images and video is an important area of research and is now routinely used in both static matching applications (passport, credit card, driving licences) and real time applications (surveillance cameras).

It is a difficult research problem because face images are highly variable - changing with lighting, background conditions, facial hair, makeup and other dynamic factors. This can lead to challenges tackling the overfitting and overtraining problems.

An important factor of face recognition is the identification of the features to be used to represent a face in varying environments and the subsequent of those features to classify new facial data.

1. Extraction of features :

Since the dimension of the input is very high, we need to perform feature set reduction.

(i) Principal Component Analysis : a face image F_i is assumed to be a 2d nxn array of intensity values. Assume we are given Q images in the training set $F = \{F_1, F_2, ...F_Q\}$ belonging to C classes. To perform PCA, we define covariance matrix :

$$K = \frac{1}{Q} \sum_{i=1}^{Q} (F_i - F') (F_i - F')^T$$

Here F' is the summation of F_i through i=n. The Eigenface feature vector X_i are calculated by projecting the facial intensity vectors to the eigenface space:

$$X_i = P^T F_i$$

(ii) Fisher's Linear Discriminant : While PCA performs dimensionality reduction, it does not provide any information on inter class discrimination. FLD is an approach that provides this information. FLD is applied to eigenface feature vectors X_i by calculating scatter matrices between classes (S_B) and within-class (S_W) .

$$S_B = \sum_{k=1}^{C} q^k \; (X'^q - X') (X'^q - X')^T$$

$$S_W = \sum_{k=1}^{C} \sum_{i=1}^{q} (X'^q - X') (X'^q - X')^T$$

The feature vectors for images with the optimal discrimination are calculated as :

$$Z_i = E_0^T X_i$$

2. Determination of structure and initialization of the RBFN :

The number of inputs to the network is equal to the numbers of features, and the number of outputs is equal to the number of classes, C . The number of RBF units is appropriately incremented through an iterative procedure which repeatedly goes through all training samples checking for inter class containment and overlaps.

Centers of the RBF units are initialised through this procedure of iterative splitting using the mean of data vectors that are associated with the RBF unit.

For initialization of widths, overlaps to the nearest RBF are minimized, while trying to maintain generalization ability of the network.

3. Hybrid Learning Algorithm :

The model uses a two pass hybrid leaning procedure - weights from hidden RBF units to the output are adjusted using linear process - linear least squares (LLS)

Parameters of the RBF units are adjusted using a non linear process - gradient descent.

(i) Weight Adjustment : Given input Z_k and q hidden RBF neurons, the signal from output neuron j is defined as RBFN :

$$Y_j^k = \sum_{h=1}^{q} w_{hj} S_h^k$$

Or in vector form $Y = S^T W$. The optimal weight matrix is obtained by keeping the RBF parameters fixed and adjusting the weights to match the desired classification outputs using the LLS algorithm.

(ii) Gradient Descent for RBF Parameter Adjustment : The parameters of the RBF neurons are adjusted through gradient descent. The error function employed is the square error function and descent is obtained by partial derivative of the error with respect to the centers and spreads of RBF neurons.

During training the RBF neurons parameters are held fixed, the centres and spreads of RBF are adjusted using gradient descent . The process is repeated till the stopping criterion is reached .

4. The ORL Database :

The algorithm was tested on a database of facial images known as ORL (Olivetti Reasearch Laboratory at Cambridge University) Face Image Database. The database comprises 400 images of 40 individuals; some subjects, images were taken against dark homogenous backgrounds. 200 images were reserved for training and 200 for testing. Each person appeared in images 5 times. After initialization and training procedure was applied to RBFN and the testing and training patterns were exchanged one time and procedure repeated on different test sets.

The RBFN model has the best error rate based on 6 runs with feature dimensions being 25 and 30 respectively.

10. List out the differences between RBF and MLP

When deciding whether to use an RBF network or an MLP, there are several factors to consider. There are clearly similarities between RBF networks and MLPs:

Similarities

1. They are both non-linear feed-forward networks.
2. They are both universal approximators.
3. They can both be used in similar application areas.

It is not surprising, then, to find that there always exists an RBF network capable of accurately mimicking a specific MLP, and vice versa. However the two network types do differ from each other in a number of important respects:

RBF	MLP
1. RBF networks are naturally fully connected with a single hidden layer	1. MLPs can have any number of hidden layers and any connectivity pattern.
2. In RBF networks, the hidden nodes (i.e., basis functions) have a very different purpose and operation to the output nodes.	2. In MLPs, the nodes in different layers share a common neuronal model, though not always the same activation function.
3. In RBF networks, the argument of each hidden unit activation function is the distance between the input and the "weights" (RBF centres)	3. In MLPs it is the inner product of the input and the weights.
4. RBF networks are usually trained quickly one layer at a time with the first layer unsupervised, which allows them to make good use of unlabelled training data.	4. MLPs are usually trained iteratively with a single global supervised algorithm, which is slow compared to RBF networks and requires many learning parameters to be set well, but allows an early stopping approach to optimizing generalization.
5. RBF networks tend to use localised non-linearities (Gaussians) at the hidden layer to construct local approximations.	5. MLPs construct global approximations to non-linear input-output mappings with distributed hidden representations.
6. RBF networks tend to use localised non-linearities (Gaussians) at the hidden layer to construct local approximations.	6. MLPs construct global approximations to non-linear input-output mappings with distributed hidden representations
7. Generally, for approximating non-linear input-output mappings, RBF networks can be trained much faster.	7. An MLP may still allow easier optimization of generalization.