Neural Networks for Signal Processing-I

Prof. Shayan Srinivasa Garani

Department of Electronic System Engineering

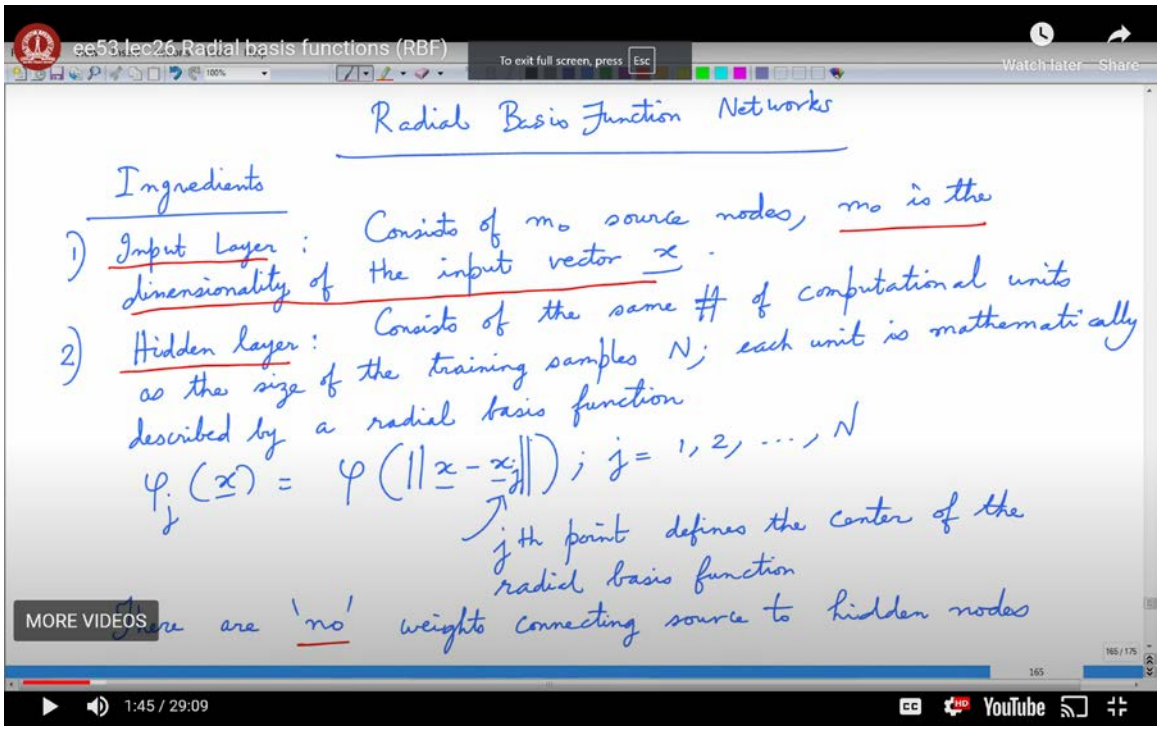
Indian Institute of Science, Bengaluru

Lecture – 26

Radial Basis Functions (RBF)

Hello, in the last lecture, we learned about radial basis functions. Now, let's extend the concept of radial basis functions to networks. How do we design a radial basis function network, and what are its key components? Typically, we have an input layer, a hidden layer, and an output layer. However, this network differs from the traditional neuron-based computational unit network, and we will explore these differences as we delve into the details.

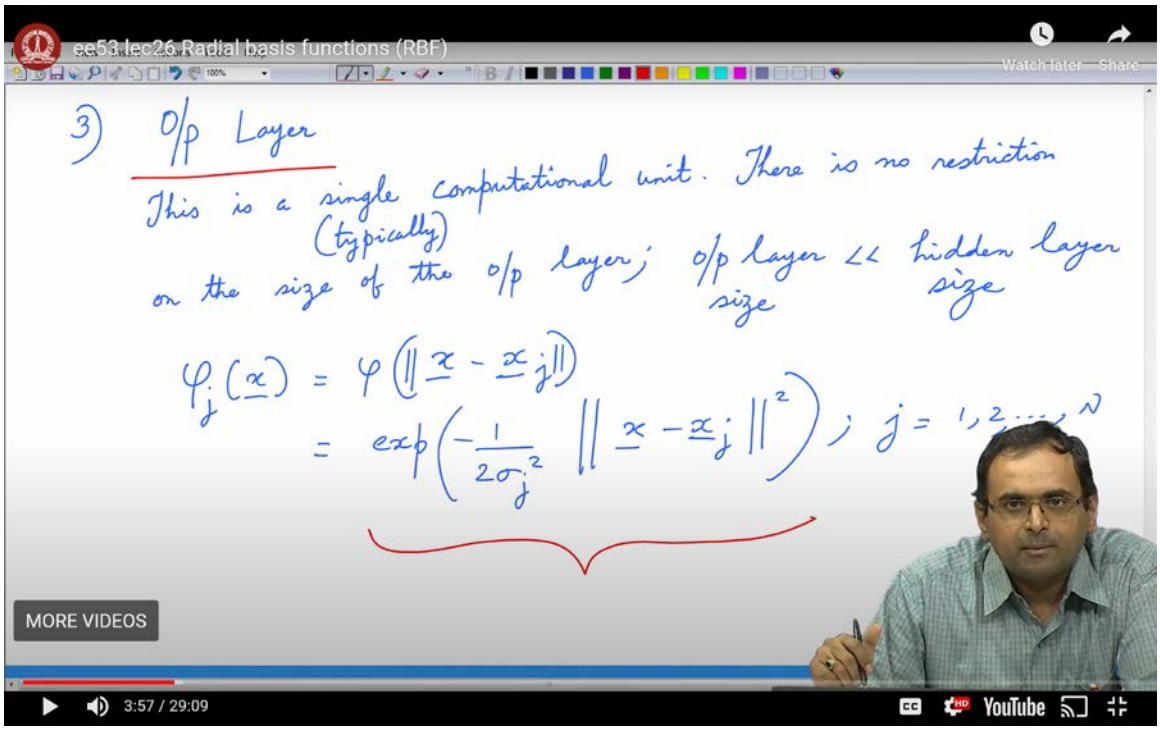
(Refer Slide Time: 01:45)



The input layer consists of M_0 source nodes, where M_0 is the dimensionality of the input vector x. The hidden layer, in a typical radial basis function network, consists of the same number of computational units as the size of the training samples. However, this is not always practical due to computational intensity. For now, let's assume this is true and proceed, later arriving at a simplification.

Each unit in the hidden layer is mathematically described by a radial basis function $\varphi_j(x)$, given by $\varphi(|x - x_j|)$. Here, the center is the data point itself, indexed from 1 to N. This involves computing the pairwise distance between every data point and its center, essentially calculating the distance metric across all points.

(Refer Slide Time: 03:57)



Unlike the multilayer perceptron, there are no weights connecting the source and hidden nodes in a radial basis function network. The output layer is a single computational unit, though its size can vary. Typically, it consists of just one node.

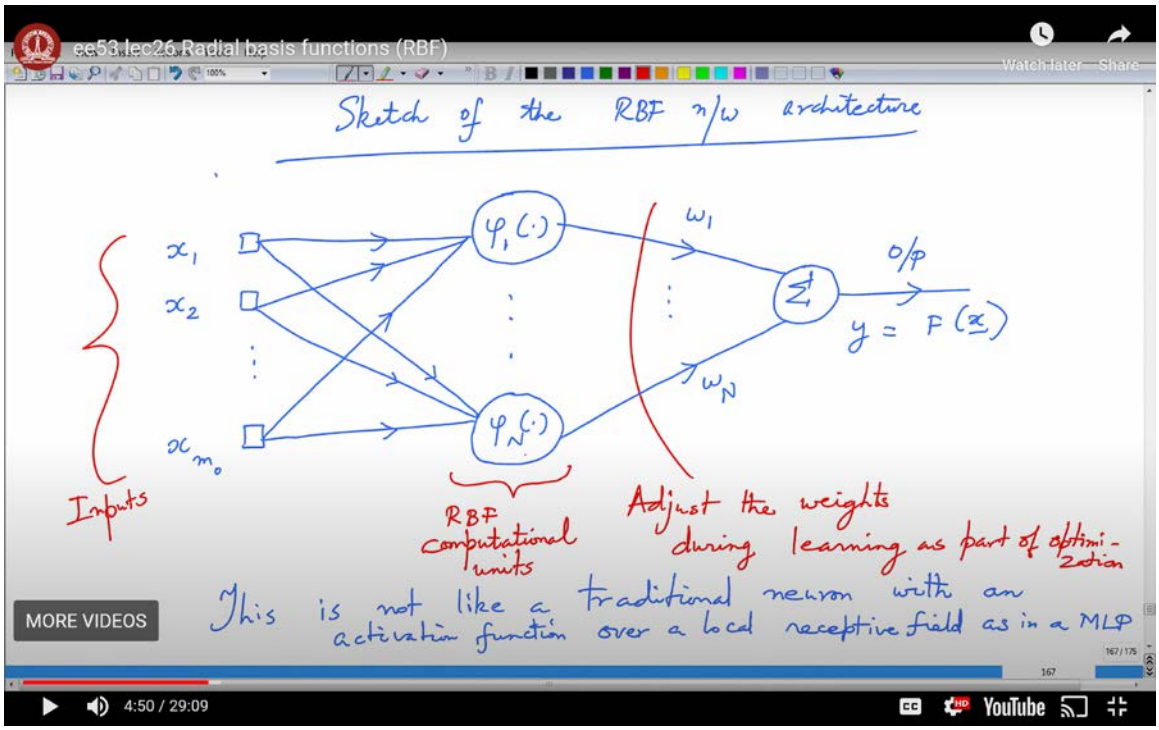
The computational units in the hidden layer are essentially these φ functions. Assuming we use the Gaussian function, it takes the form:

$$\phi_j(x) = \exp\left(-\frac{1}{2\sigma_j^2}|x - x_j|^2\right).$$

This formula highlights the Gaussian nature of the function, where σ_j is the standard deviation, and $|x - x_j|$ represents the Euclidean norm between the input vector x and the center x_j. This is how we define and understand the structure and components of a radial basis function network.

This is the form of our function φ_j, and since we assumed that the number of units in the hidden layer is equal to the number of data points, this goes from j = 1 to N. Now, let's delve deeper into sketching the architecture based on our ideas. We have a set of inputs in an M_0-dimensional space, x_1 to x_{M_0}. These inputs are directly linked to each of the hidden units without any weights, just a direct connection with a value of 1.

(Refer Slide Time: 04:50)



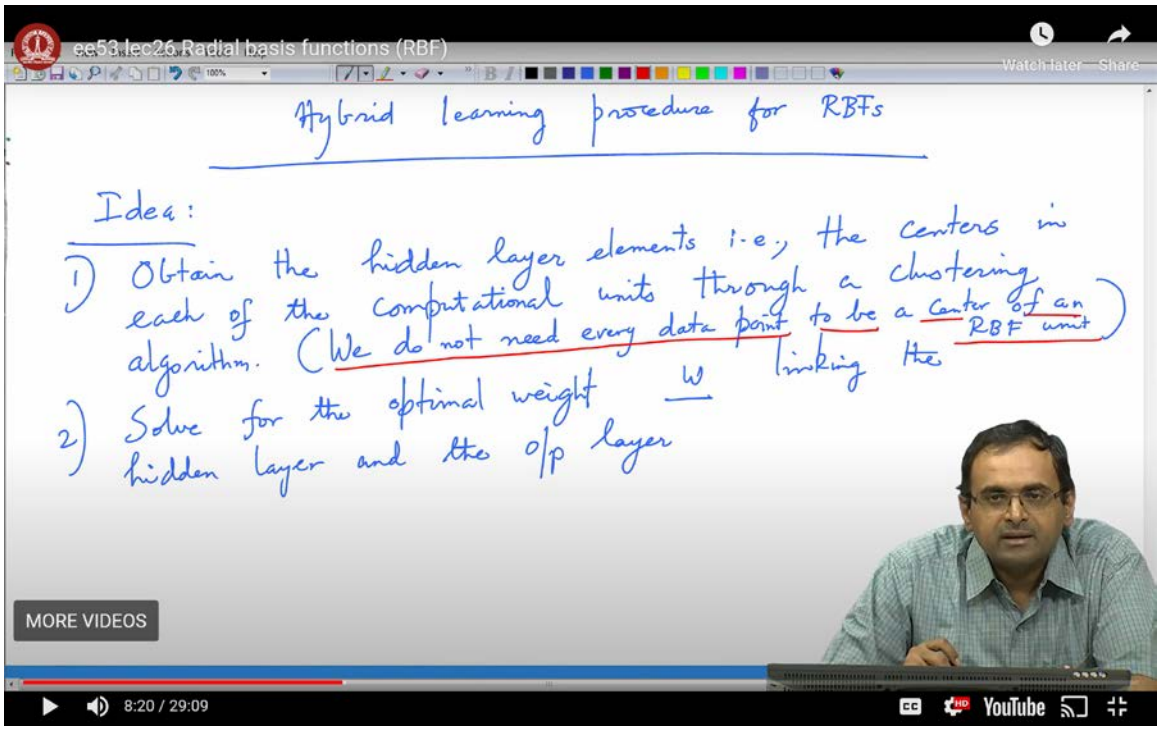
In this architecture, we have Gaussian units from φ_1 to φ_n, and we discussed previously that each φ_j has this specific form. The hidden nodes are connected to the output using a set of weights, denoted as W_1 through W_n. The n-th node of the hidden layer is connected

to the output via W_n, and the first node is connected via W_1, and so on. At the output, we form a linear combination of these weights.

It's important to note that this differs from a traditional neuron, which has an activation function over a local receptive field. This is a key distinction between the RBF unit and the typical McCulloch-Pitts neuron we discussed earlier.

This is a basic sketch of the RBF architecture. The critical aspect of the learning process is learning these weights, which are optimized during the training process. Since we know the output of the RBF network and the desired response, we can optimize these weights using an objective function. We can obtain the optimal weights by inverting the φ matrix and multiplying it by the desired vector, as we discussed earlier.

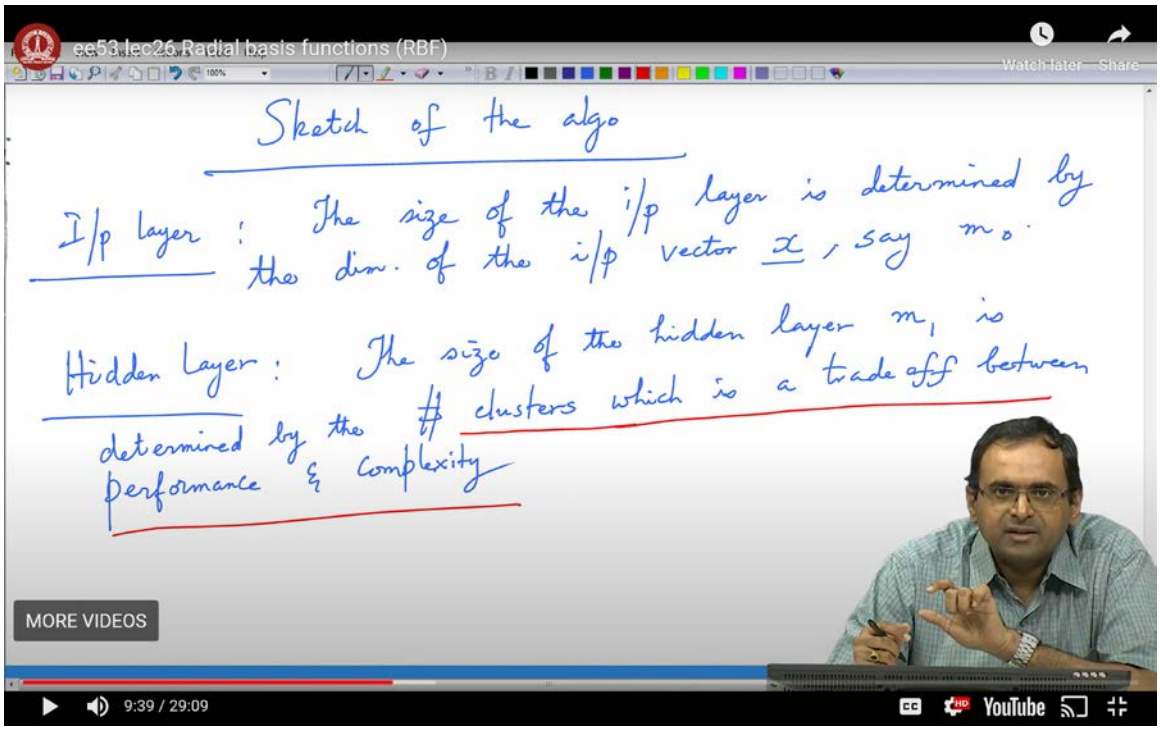
(Refer Slide Time: 08:20)



To evolve a hybrid learning procedure for the RBFs, the idea is to obtain the hidden layer elements, or centers, for each computational unit through a clustering algorithm. Given a set of data points, we can arrive at clusters K, which is significantly fewer than the number

of data points N. This reduces computational complexity by limiting the number of hidden units.

(Refer Slide Time: 09:39)



The next step is to solve for the optimal weights W linking the hidden layer to the output layer. The only unknowns are these weights, which we solve for given the set of data points and appropriately chosen K clusters. Here is a sketch of this hybrid learning algorithm:

1. Input Layer: Consists of inputs with dimension M_0. The size of the input layer is determined by the dimension of the input.

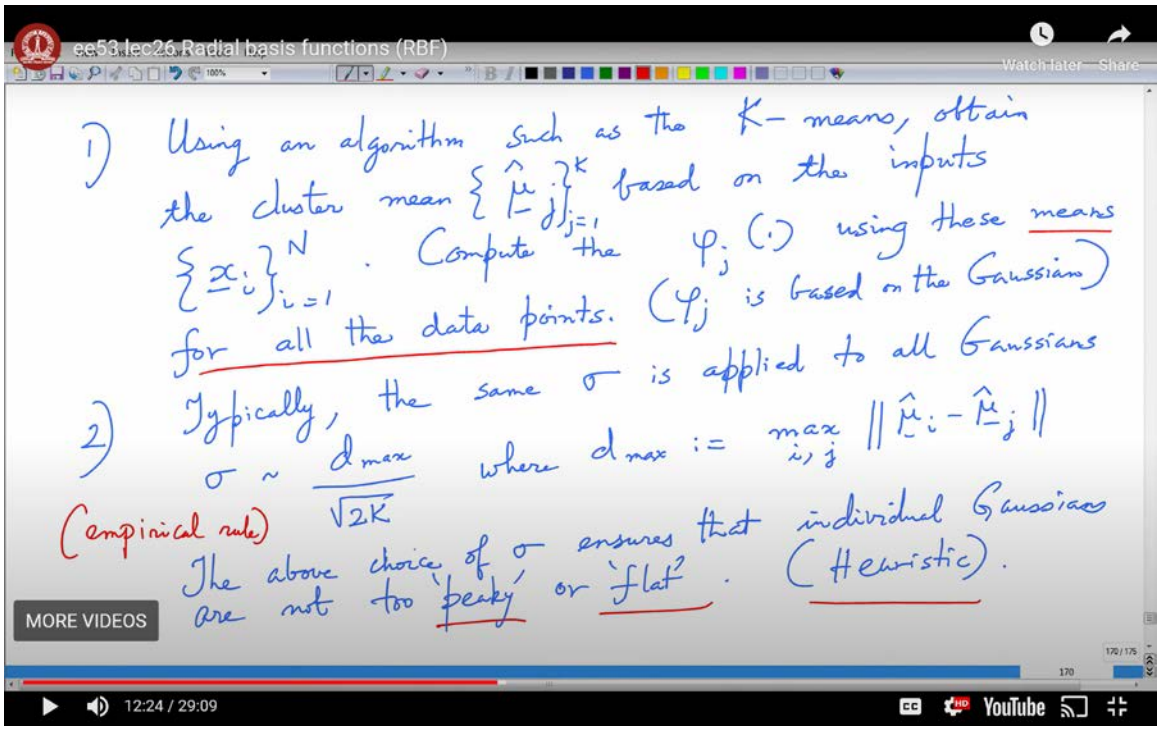
2. Hidden Layer: Contains computational units corresponding to the clusters obtained through a clustering algorithm.

3. Output Layer: A linear combination of the hidden layer outputs weighted by the optimized weights W.

By following this approach, we can efficiently train an RBF network to achieve the desired results with reduced computational complexity.

The hidden layers are composed of numerous Gaussian units, or radial basis function (RBF) computational units. The number of units is determined by clusters, which balance performance and complexity. As previously discussed, it is unnecessary to have N units corresponding to each data point. This reduction in computational demand comes with a tradeoff in performance. Ideally, selecting every data point as a center minimizes distortion, but this approach increases complexity.

(Refer Slide Time: 12:24)



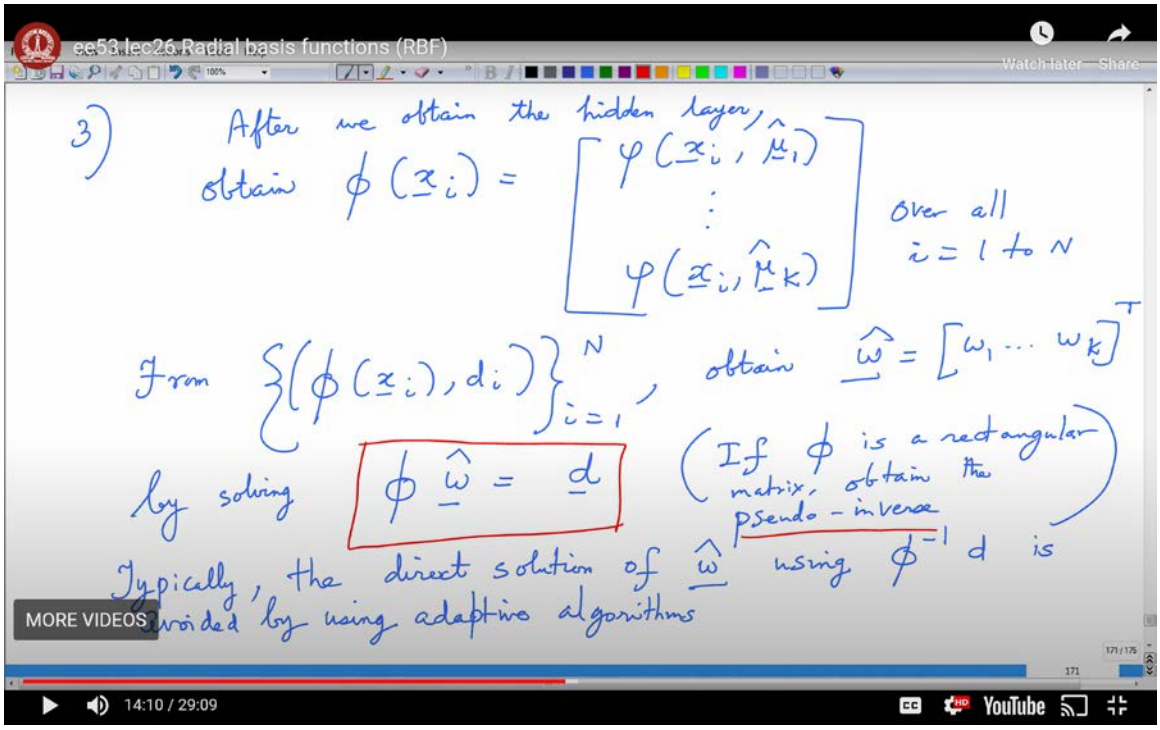
By using algorithms like k-means, we can determine cluster means μ_j (denoted as $\hat{\mu}_j$ to indicate estimates), for j ranging from 1 to K, where K represents the number of clusters based on the input data points x_i, for i ranging from 1 to n. These cluster means are then used to compute φ_j values for all data points. Once these computations are complete and the Φ matrix is formed, incorporating data points and clusters, we can combine these with weights and desired responses into a matrix equation and solve it.

A pertinent question is how to choose σ for these Gaussian units. One approach is to have a σ_j for each cluster, calculated from the cluster variance. However, this method can result

in either too narrow or too broad Gaussian distributions, depending on the centers. To address this, we can use a uniform σ for all Gaussian units, calculated as $d_{\max}/\sqrt{2K}$. This is an empirical heuristic where $d_{\max}$ is the maximum distance between any pair of cluster centers, and K is the number of clusters. This choice ensures that individual Gaussians are neither too narrow nor too broad.

Once these parameters are computed, the problem reduces to solving the matrix equation $\Phi\widehat{W} = d$, where d is the desired vector. The Φ matrix is formed from all data points i = 1 to n and clusters $\widehat{\mu_1}$ to $\widehat{\mu_K}$.

(Refer Slide Time: 14:10)



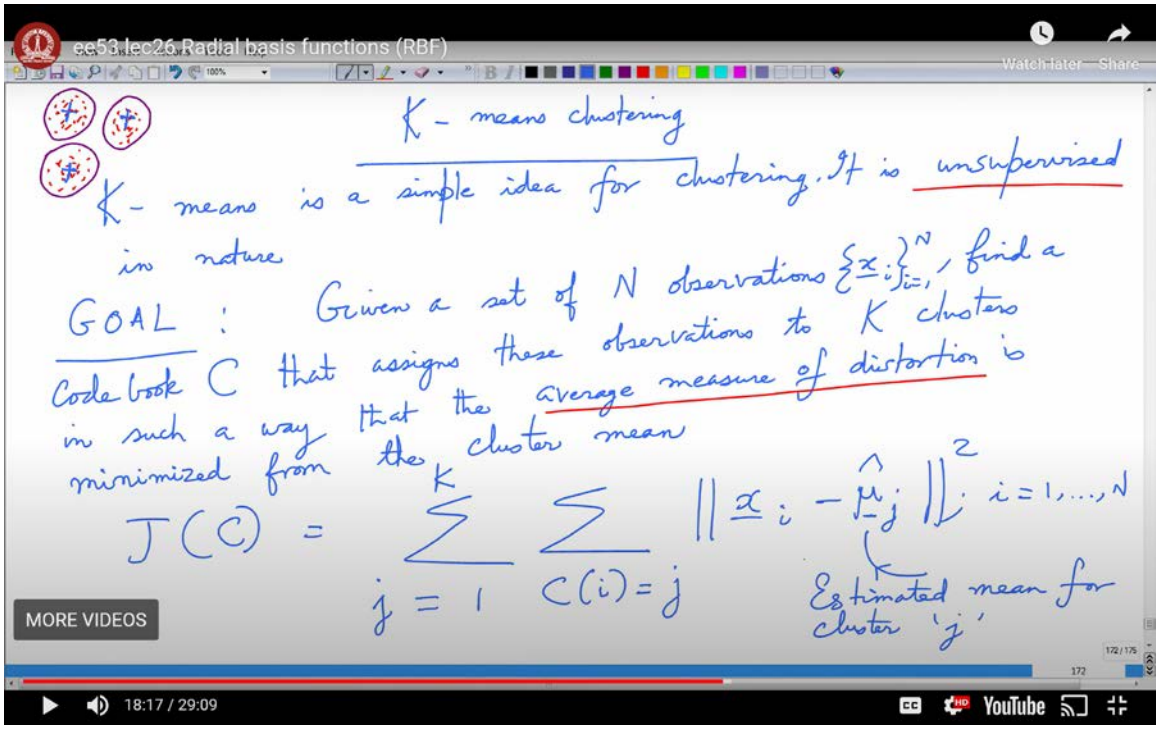
You may wonder whether Φ could be a rectangular matrix. It will be a square matrix if the number of functional units equals the number of data points (i and j both ranging from 1 to n). However, if the number of data points is n and the number of clusters is K, Φ will be rectangular. In this case, we must compute a pseudo-inverse to find the optimal $\widehat{W}$, as direct inversion is not feasible.

In such cases, we address these challenges using adaptive algorithms. One effective method is the recursive least squares algorithm, which enables us to solve for these weights optimally.

At this point, we haven't delved into the clustering algorithm, so let's discuss the K-means algorithm, a well-known and popular method in the machine learning community. K-means is one of the fundamental algorithms encountered in this field.

So, what exactly is K-means? It is a straightforward idea for clustering that operates in an unsupervised manner, meaning there is no desired response guiding the process. Given a set of data points, the objective is to cluster them based on a specific optimization function.

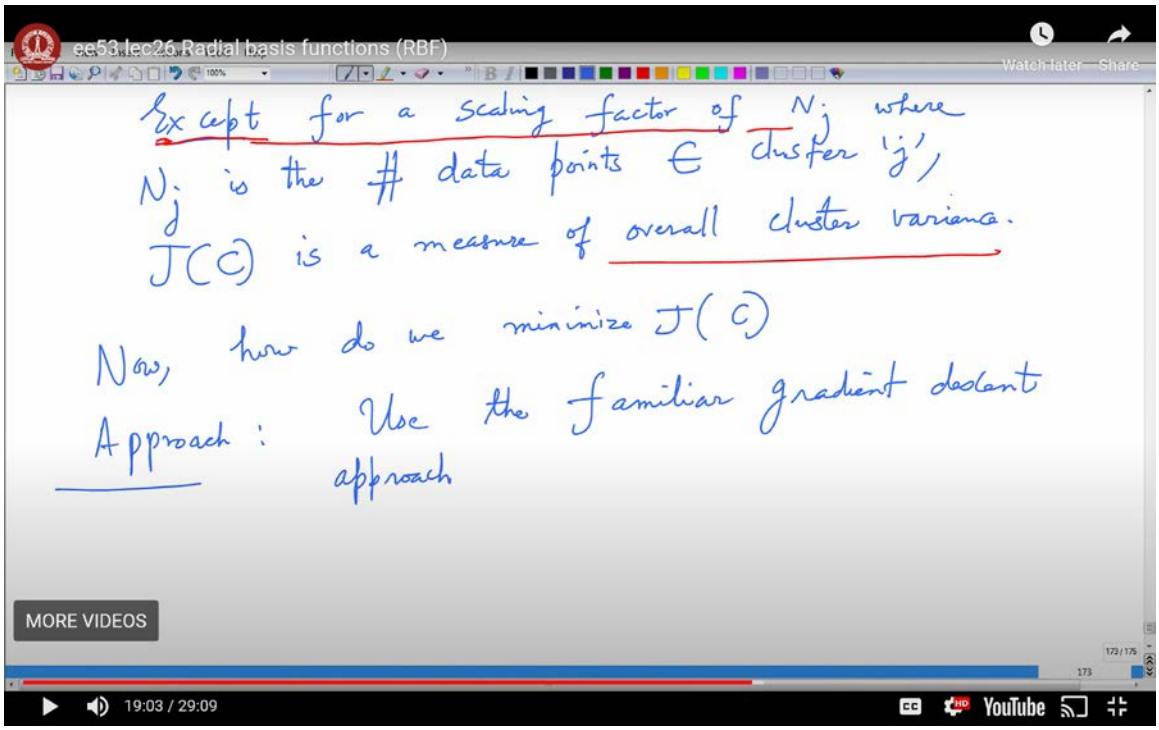
(Refer Slide Time: 18:17)



The goal of the K-means clustering algorithm is to find a codebook C that assigns n observations X_i (where i ranges from 1 to n) to k clusters. This assignment should minimize the average measure of distortion from the cluster mean. In other words, given a set of points, the algorithm aims to create clusters such that each cluster is represented by its mean, thus reducing distortion.

Imagine you have a set of points that naturally form three clusters. Instead of representing each cluster by individual data points, you represent them by a single label, typically the cluster mean, indicated by blue crosses. This concept is akin to a codebook, where each data point is associated with a cluster center or index, which can then be used for various purposes.

(Refer Slide Time: 19:03)



In signal processing, this concept is known as vector quantization. In the pattern recognition, neural networks, or machine learning communities, it is referred to as clustering, though they are essentially the same.

The cost J(C) is defined by this distortion function, which means we have k clusters (with j ranging from 1 to k) over which we compute the cluster variance or distortion. For each cluster, we have a data point x_i encoded to this cluster j, represented by the inner summation. This process computes the total distortion across all clusters.

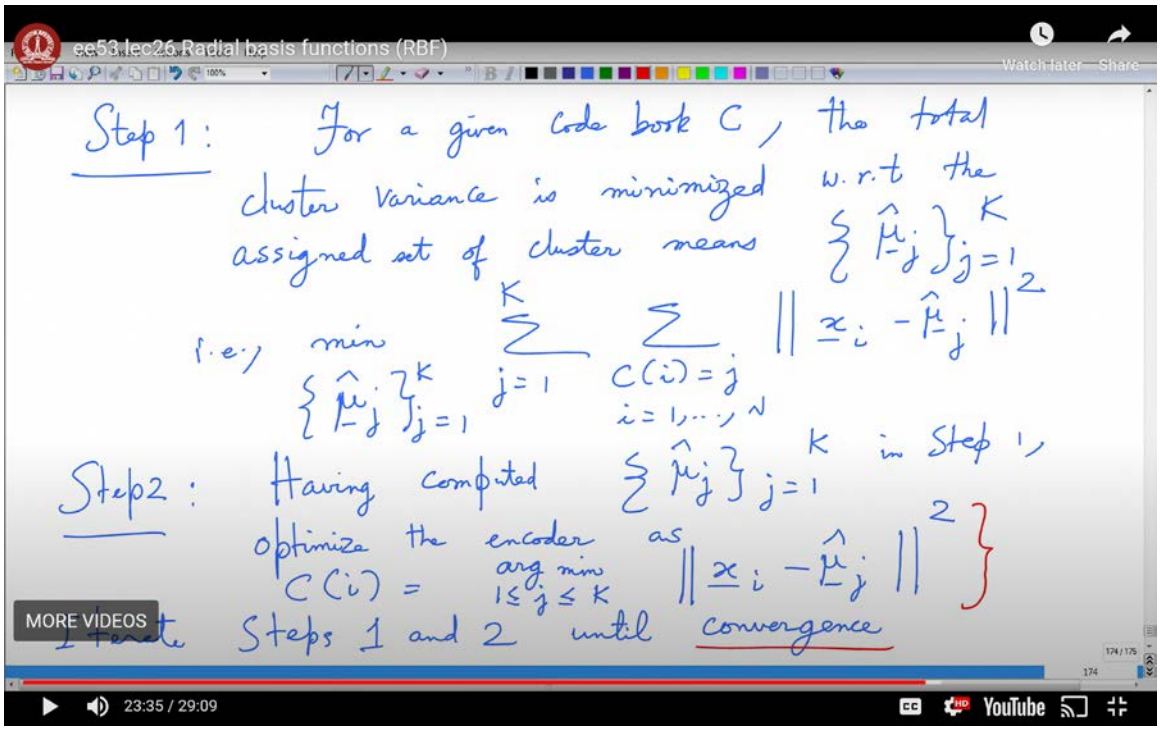
To tackle this problem, we follow a two-step process: computing the cluster mean μ_j and updating the codebook. Before detailing the algorithm, it's important to note that there is

no scale factor N_j, which is the number of data points belonging to cluster j. This approach measures the overall cluster variance without considering the number of points per cluster.

To formalize this approach, we use a familiar gradient descent method to minimize the distortion function. This involves iteratively adjusting the cluster centers to reduce the total distortion, thereby optimizing the clustering process.

The first step involves initializing the codebook C. You start with some random centers and then proceed to update them. The total cluster variance is minimized with respect to the assigned set of cluster means, $\hat{\mu}_j$. To minimize this cluster distortion, we use the familiar gradient descent approach: $\mu_{new} = \mu_{old} - \alpha \cdot \nabla_{\mu} J$, where α is the learning rate. This iterative update continues until convergence. Alternatively, you can directly derive the optimal $\hat{\mu}_j$ by taking the functional derivative and solving for it, but the adaptive method is typically easier.

(Refer Slide Time: 23:35)



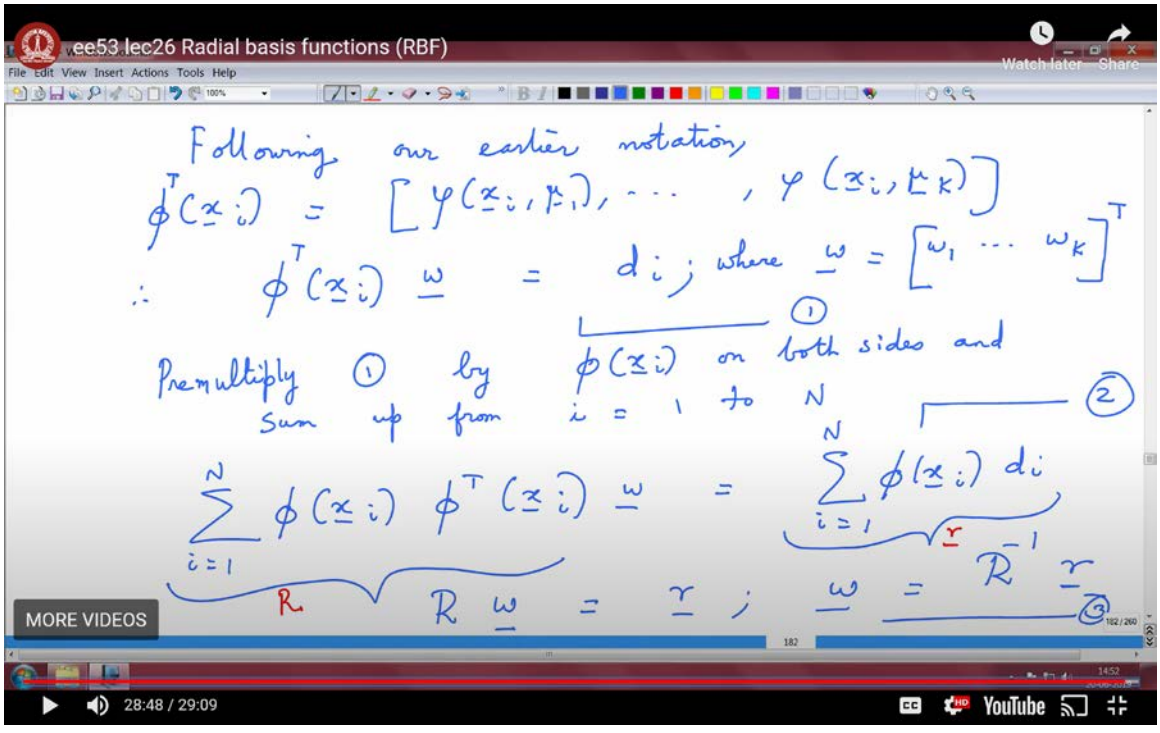
Once we obtain all the $\hat{\mu}_j$ (for j = 1 to k), we optimize the encoder to choose the label i for a data point by minimizing the distortion. For a given data point x_i, we compute the

distortion $x_i - \hat{\mu}_j$ for all j from 1 to k. We then select the j that gives the smallest distortion for x_i; this j is the cluster to which x_i is assigned. We iterate steps 1 and 2 until convergence.

This iterative process is necessary because we are adaptively estimating the cluster means $\hat{\mu}_j$. Convergence is reached when we achieve the minimum possible distortion, beyond which further reduction is not possible. It's important to note that when k equals N (the number of data points), there's no actual clustering, resulting in zero distortion. This scenario highlights the performance-complexity tradeoff in the RBF network: choosing k hidden units instead of using all data points as cluster centers is a more practical approach.

Having covered the key components of the radial basis function network, we need to address the recursive least squares algorithm for efficiently computing the weights. The weights can be determined either through direct matrix inversion or using an adaptive algorithm.

(Refer Slide Time: 28:48)



To set up the adaptive algorithm, let's delve into some details with our notation. Following our earlier notation, $\phi^T(x_i)$ represents $\phi(x_i, \mu_1), \phi(x_i, \mu_2), \ldots, \phi(x_i, \mu_k)$, where

$\mu_1, \mu_2, \ldots, \mu_k$ are the cluster centers determined by a clustering algorithm like K-means. We'll discuss the K-means algorithm next as a way to realize these clusters.

To compute $\phi(x_i, \mu_1), \ldots, \phi(x_i, \mu_k)$, we form the vector $\phi^T(x_i)$. This gives us $\phi^T(x_i) \cdot w = d_i$, where d_i is a scalar and $w = [w_1, \ldots, w_k]^T$ is the weight vector. Note that $\phi^T(x_i)$ is a row vector and w is a column vector, making their inner product a scalar d_i.

Next, we pre-multiply this equation $\phi^T(x_i) \cdot w = d_i/\phi(x_i)$ on both sides and sum over all data points. The result is:

$$\sum_{i=1}^{N} \phi(x_i)\phi^T(x_i)w = \sum_{i=1}^{N} \phi(x_i)d_i$$

The summation on the left-hand side forms the matrix R, and the summation on the right-hand side forms the vector r. Therefore, we have:

$$Rw = r$$

Where R is the matrix formed by $\sum_{i=1}^{N} \phi(x_i)\phi^T(x_i)$ and r is the vector $\sum_{i=1}^{N} \phi(x_i)d_i$. To find the weight vector w, we compute:

$$w = R^{-1}r$$

However, computing the inverse of R can be challenging, especially with a large number of data points, N. Even finding the pseudo-inverse for a rectangular matrix is computationally intensive. Therefore, we need a more efficient solution.

To address this, we will adopt the recursive least squares (RLS) approach for solving this matrix equation adaptively. In the next lecture, we will discuss the details of the RLS algorithm and how it can be implemented efficiently in an adaptive manner. With this we will close this lecture. Thank you.