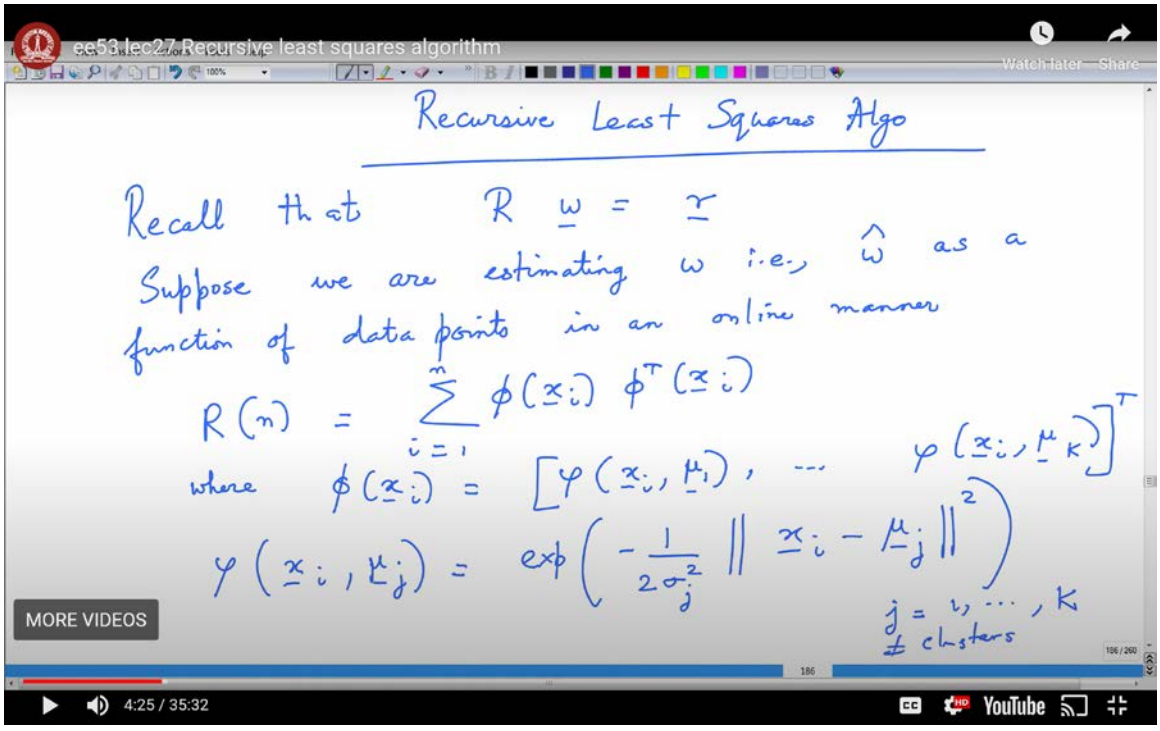
Neural Networks for Signal Processing-I
Prof. Shayan Srinivasa Garani
Department of Electronic System Engineering
Indian Institute of Science, Bengaluru

Lecture – 27
Recursive Least Squares Algorithm

In our discussion of the hybrid learning algorithm, I mentioned the necessity of the K-means algorithm for computing the cluster centers. Additionally, I pointed out the significant challenge posed by the matrix inversion when dealing with large datasets. As the number of data points increases, the complexity of inversion becomes a major issue. To address this, we require an adaptive algorithm, and one effective method is the Recursive Least Squares (RLS) algorithm. This algorithm is a standard tool in adaptive signal processing, but here I'll explain how it applies specifically to this problem. Understanding this technique will be invaluable when you encounter practical applications in the field.

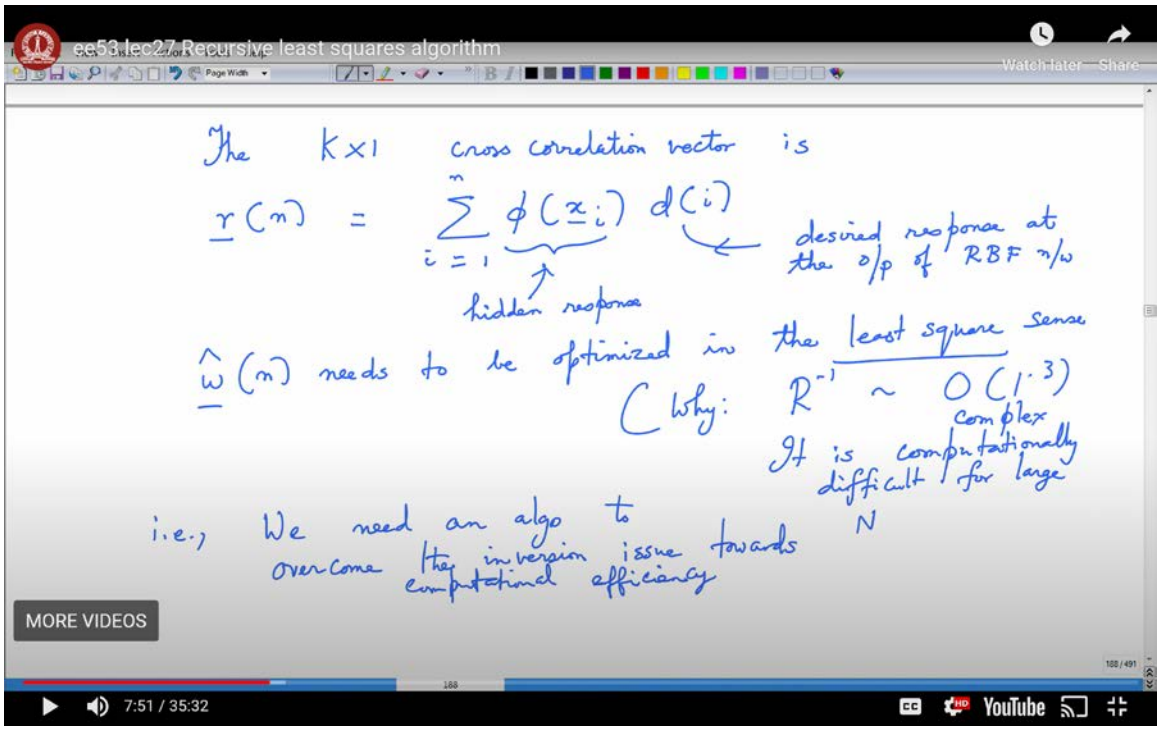
(Refer Slide Time: 04:25)



Let's recall the matrix equation we previously discussed: $Rw = r$, where we have already defined all these terms earlier. To reiterate, the matrix $\boldsymbol{R}$ as a function of the time index n is given by:

$$R(n) = \sum_{i=1}^{n} \phi(x_i)\phi^T(x_i)$$

(Refer Slide Time: 07:51)



Here, $\varphi(x_i)$ is the vector function evaluated at the data point x_i, with $\varphi(x_i, \mu_j)$ being the individual components corresponding to the cluster means μ_1 through μ_k.

To clarify, the expression for $\varphi(x_i, \mu_j)$ is:

$$\phi(x_i, \mu_j) = \exp\left(-\frac{1}{2\sigma_j^2}|x_i - \mu_j|^2\right)$$

where j runs from 1 through k, consistent with the number of clusters.

Previously, I referred to $\boldsymbol{R}$ without specifying it as $R(n)$, summing from i = 1 to N, where N is the total number of data points. However, we don't have to be confined to this batch processing approach. We can estimate $R(n)$ iteratively as new data points become available, in an online fashion.

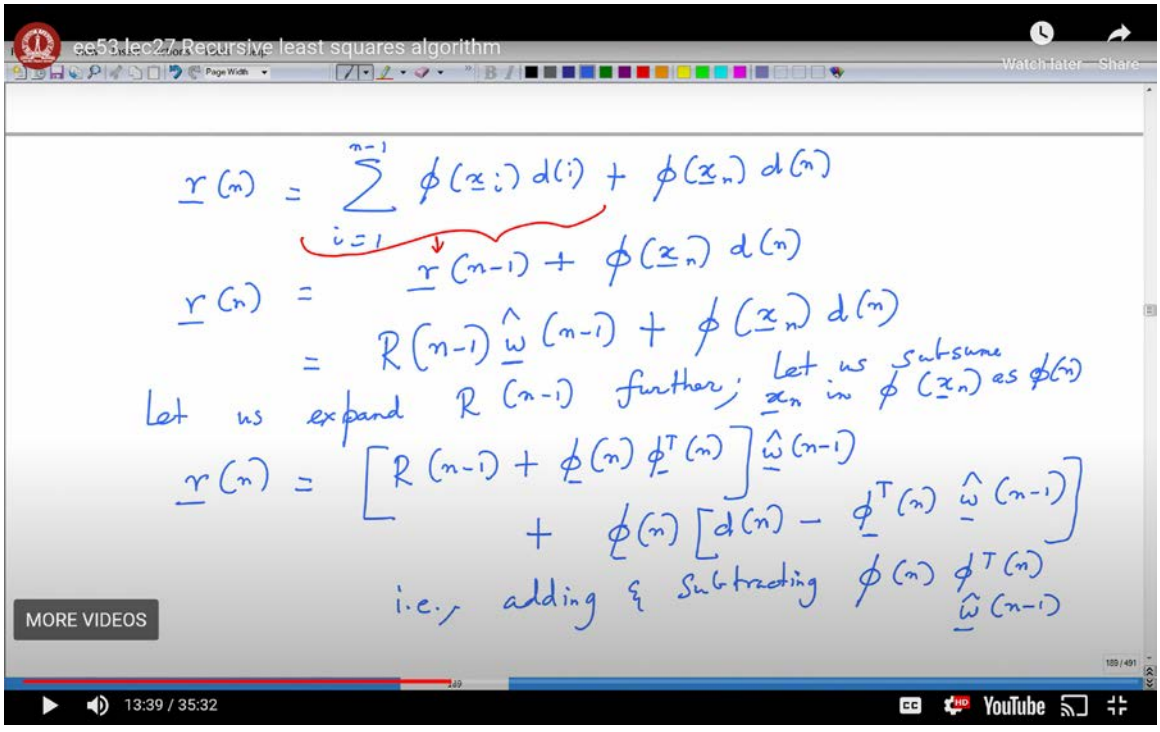
Moving on to the cross-correlation vector, denoted by $\boldsymbol{R}(n)$, this vector is defined as:

$$r(n) = \sum_{i=1}^{n} \phi(x_i) d_i$$

where d_i represents the desired response at the output of the Radial Basis Function (RBF) network, and $\varphi(x_i)$ represents the hidden layer's response.

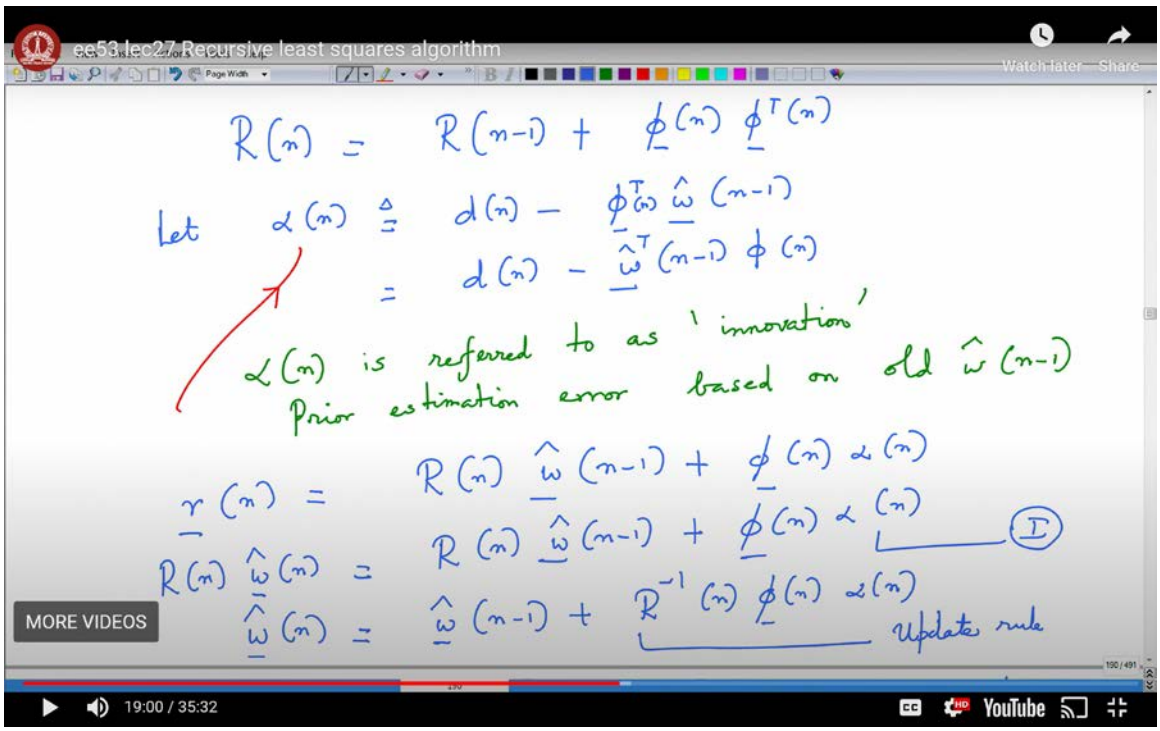
The goal is to optimize the weight vector $\hat{w}(n)$ in the least squares sense. The reason for this is straightforward: the computational complexity of inverting the matrix $\boldsymbol{R}$ is on the order of $O(n^3)$, where n is the number of data points. This high complexity makes direct inversion computationally expensive and impractical for large datasets.

(Refer Slide Time: 13:39)



This challenge is precisely why adaptive algorithms like RLS are so crucial. To clarify a point I made earlier, the dimension of R is actually k × k, not n × n. I mentioned the order of complexity as $O(n^3)$, but to be more accurate, it should be $O(k^3)$. This is because $\varphi(x_i)$ $\varphi^T(x_i)$ produces a k × k matrix, and when summing over all data points, we're effectively adding up these k × k matrices.

(Refer Slide Time: 19:00)



Thus, the matrix inversion process is of order k^3, consistent with the size of the matrix we're dealing with. Let's delve into the details.

Consider r(n), which is defined as:

$$r(n) = \sum_{i=1}^{n-1} \phi(x_i)d_i + \phi(x_n)d_n$$

Here, I've split the original summation $\sum_{i=1}^{n} \phi(x_i)d_i$ into two parts: the first n-1 terms and the last term separately. This can be rewritten as:

$$r(n) = r(n-1) + \phi(x_n)d_n$$

This expression also applies to our matrix R(n). Fortunately, we know the solution for R(n), which is in terms of the matrix R and the weight vector w. We can estimate r(n-1) based on this and express it as:

$$r(n-1) = R(n-1)\hat{w}(n-1) + \phi(x_n)d_n$$

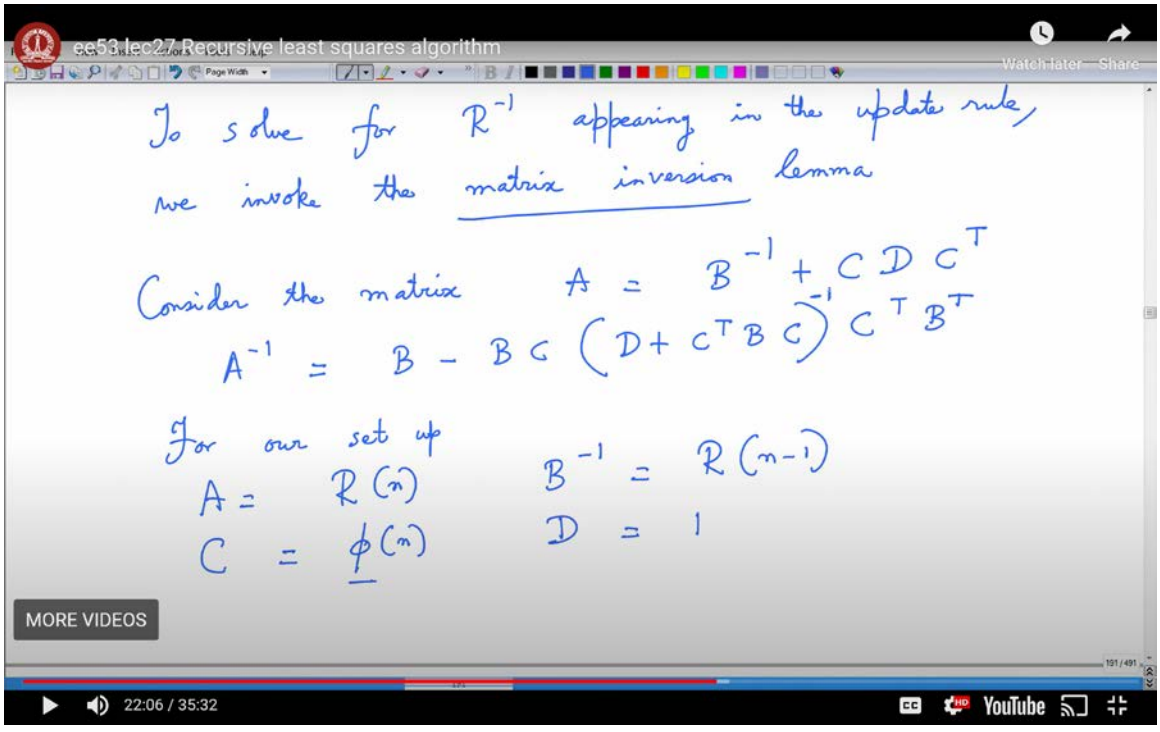
This is straightforward, but we can also expand r(n-1) further by adding and subtracting certain terms to help us with the recursive formulation.

So, let's expand r(n-1) a bit more carefully. We have:

$$r(n) = r(n-1) + \phi(x_n)\phi^T(x_n)\hat{w}(n-1) + \phi(x_n)d_n$$

For simplicity, let's denote $\varphi(x_n)$ as $\varphi(n)$, meaning that whenever I refer to $\varphi(n)$, it's understood to be evaluated at x_n. This way, we can avoid repeating the x_n notation throughout the equations.

(Refer Slide Time: 22:06)



Now, what I'm doing here is expanding r(n) as follows:

$$r(n) = r(n-1) + \phi(n)\phi^T(n)\hat{w}(n-1) + \phi(n)d_n - \phi^T(n)\hat{w}(n-1)$$

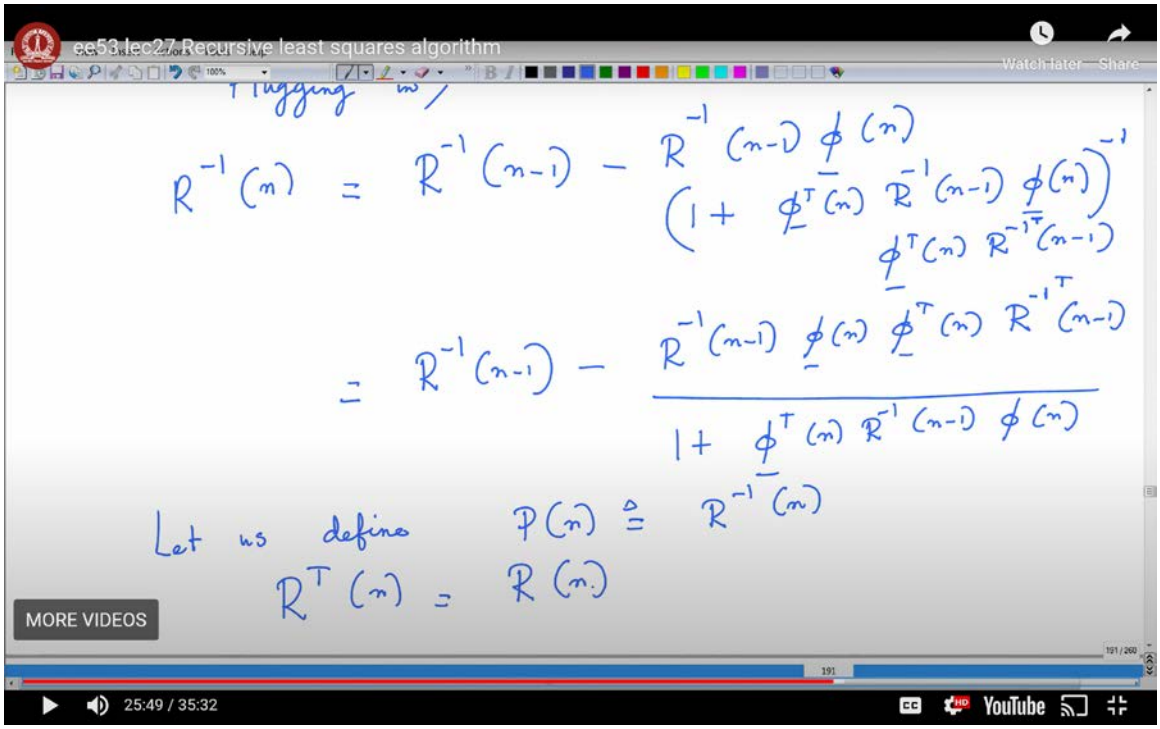
Here, I've added and subtracted the term φ(n) φ^T(n) $\hat{\boldsymbol{w}}$(n-1) to simplify the equation.

Next, we also have a recursion for the matrix R(n), given by:

$$R(n) = R(n-1) + \phi(n)\phi^T(n)$$

Again, x_n is subsumed as index n to simplify the notation.

(Refer Slide Time: 25:49)



Now, let's define the term α(n) as:

$$\alpha(n) = d_n - \phi^T(n)\hat{w}(n-1)$$

Since α(n) is a scalar, and $\phi^T(n)\hat{w}(n-1)$ is equivalent to $\hat{w}(n-1)\phi^T(n)$, we can rewrite it in this form.

This term α(n) is often referred to as the "innovation term" because it represents the estimation error based on the previous value of $\hat{w}$. Given the current response d_n and the function φ(n), we compute this term based on the prior estimate of $\hat{\boldsymbol{w}}$. It's called "innovation" because it reflects the improvement over the prior estimate.

Continuing, r(n) can be expressed as:

$$r(n) = R(n)\hat{w}(n-1) + \phi(n)\alpha(n)$$

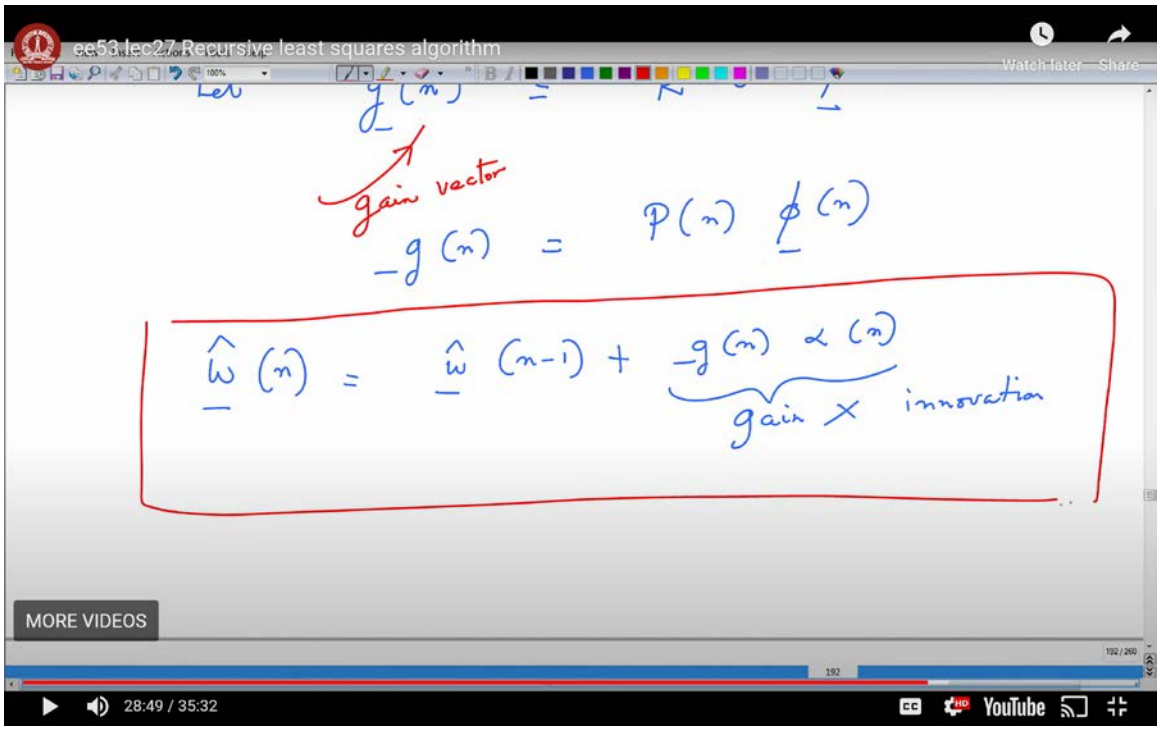
Where α(n) is the scalar, φ(n) is a vector, and everything is consistent.

Finally, this vector r(n) can be rewritten as:

$$r(n) = R(n)\hat{w}(n) = r(n)\hat{w}(n-1) + \phi(n)\alpha(n)$$

This equation ties together the recursive update of the weight vector $\hat{\boldsymbol{w}}$(n) in the context of our adaptive algorithm.

(Refer Slide Time: 28:49)



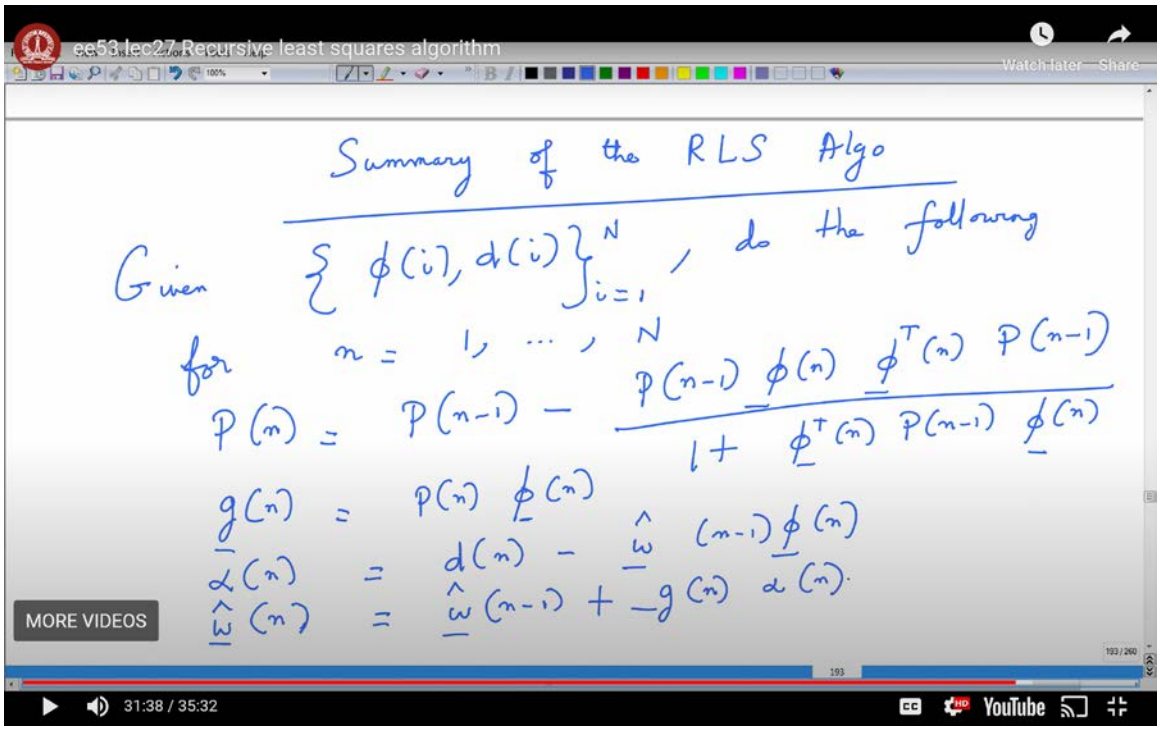
So, our update equation for $\hat{\boldsymbol{w}}$(n) can be expressed as follows: If we pre-multiply this equation, let's refer to it as Equation 1, $by R^{-1}(n)$ on both sides, we can simplify it to this form:

$$\hat{w}(n) = \hat{w}(n-1) + R^{-1}(n)\phi(n)\alpha(n)$$

Here, $\hat{w}(n)$ is the updated weight vector at time step n, and the term $R^{-1}(n)\phi(n)\alpha(n)$ provides the adjustment based on the current data. On the slide, there's a minor typo where $\hat{\boldsymbol{w}}$ should be transposed, and this correction ensures that the expression forms an inner product, resulting in a scalar.

Now, you might be wondering, "Aren't we still dealing with matrix inversion? What exactly have we simplified?" Fortunately, we can invoke the Matrix Inversion Lemma, which is particularly useful in efficiently computing the inverse of a matrix under certain conditions.

(Refer Slide Time: 31:38)



To apply this lemma, consider a matrix A defined as:

$$A = (B^{-1} + CDC^T)$$

where B, C, and D are matrices conforming to the rules of matrix multiplication. Given this setup, the inverse of A can be computed as:

$$A^{-1} = B - BC(D + C^T BC)^{-1} C^T B^T$$

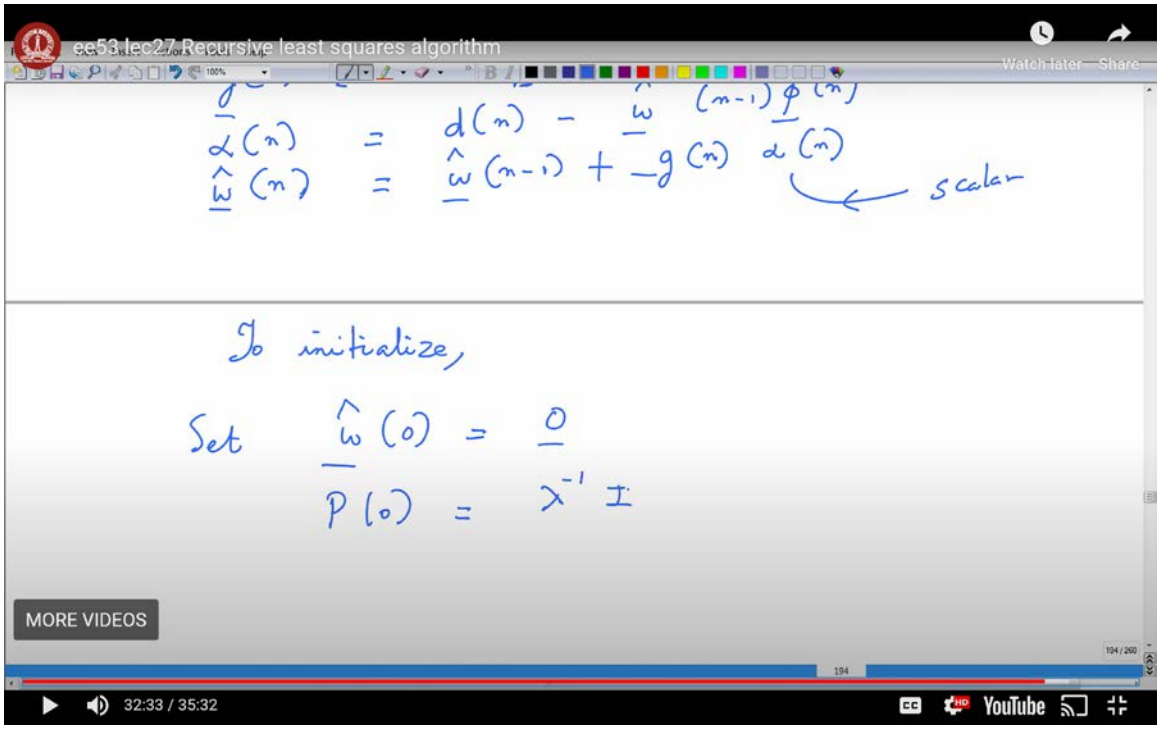
This result can be found in any standard linear algebra textbook. In our context, we define:

- A = R(n)
- $B^{-1} = R(n-1)$
- $C = \phi(n)$
- D = 1

Now, applying the Matrix Inversion Lemma, $R^{-1}(n)$ can be expressed as:

$$R^{-1}(n) = R^{-1}(n-1) - R^{-1}(n-1)\phi(n)[1 + \phi^T(n)R^{-1}(n-1)\phi(n)]^{-1}\phi^T(n)R^{-1}(n-1)$$

(Refer Slide Time: 32:33)



This expression simplifies by recognizing that the term $[1 + \phi^T(n)R^{-1}(n-1)\phi(n)]$ is a scalar, which moves to the denominator:

$$R^{-1}(n) = R^{-1}(n-1) - \frac{R^{-1}(n-1)\phi(n)\phi^T(n)R^{-1}(n-1)}{1 + \phi^T(n)R^{-1}(n-1)\phi(n)}$$

To simplify further, let's define a new matrix P(n) as $R^{-1}(n)$, which helps us avoid the cumbersome inverse notation. Thus, P(n) becomes:

$$P(n) = P(n-1) - \frac{P(n-1)\phi(n)\phi^T(n)P(n-1)}{1 + \phi^T(n)P(n-1)\phi(n)}$$

This definition allows us to manage the inversion process more efficiently.

Lastly, we note that the matrix transpose property $R^T(n) = R(n)$ implies that:

$$R^{-1}(n-1)^T = R^{-1}(n-1)$$

Hence, the inversion and transpose operations maintain consistency, and we can confidently rewrite the expression in this simplified form, making it easier to work with and computationally more efficient.

So, the transpose of $R^{-1}(n-1)$ is simply $R^{-1}(n-1)$, which we denote as P(n-1). Now, let's define the term g(n) as $R^{-1}(n)\phi(n)$, and we refer to this as the gain vector. Therefore, the gain vector g(n) can be expressed as P(n) φ(n). With this, we revisit our update equation for $\hat{w}(n)$:

We started earlier with the equation:

$$\hat{w}(n) = \hat{w}(n-1) + R^{-1}(n)\phi(n)\alpha(n)$$

Since $R^{-1}(n)$ is essentially P(n) and P(n) φ(n) gives us the gain vector g(n), our update equation for $\hat{w}(n)$ becomes:

$$\hat{w}(n) = \hat{w}(n-1) + g(n)\alpha(n)$$

Here, it's crucial to ensure that all vectors are properly indicated with subscripts, as there was a slight oversight in the notation earlier.

With this equation established, we're now prepared to summarize the Recursive Least Squares (RLS) algorithm. Let's break down the steps:

We are provided with φ_i and d_i, where d_i represents the desired response for the radial basis function (RBF) corresponding to each input. The function $\varphi(i)$ is evaluated at the data points x_i. Here's what we need to do for each time step n, from 1 to N:

1. Initialize: Start with $\widehat{\boldsymbol{w}}(0) = 0$ (a zero vector) and $P(0) = \lambda^{-1} I$, where λ is a small positive constant and I is the identity matrix.

2. Recursively update:

- Compute P(n) using:

$$P(n) = P(n-1) - \frac{P(n-1)\phi(n)\phi^T(n)P(n-1)}{1 + \phi^T(n)P(n-1)\phi(n)}$$

- Compute the gain vector g(n) as:

$$g(n) = P(n)\phi(n)$$

- Compute the innovation term $\alpha(n)$ as:

$$\alpha(n) = d(n) - \widehat{w^T}(n-1)\phi(n)$$

Here, $\widehat{w^T}(n-1)\phi(n)$ is a scalar.

- Update $\widehat{\boldsymbol{w}}(n)$ using:

$$\widehat{w}(n) = \widehat{w}(n-1) + g(n)\alpha(n)$$

This recursive approach for updating both $\widehat{\boldsymbol{w}}(n)$ and P(n) ensures computational efficiency, especially as the algorithm progresses over multiple time steps. The initialization step is crucial as it sets up the algorithm for convergence.

This method, facilitated by the Matrix Inversion Lemma, significantly simplifies the computation, particularly for large data sets. The lemma provides an elegant solution to the otherwise challenging task of matrix inversion, which is essential in the hybrid learning procedure for Radial Basis Function Networks (RBFNs).

To summarize the entire algorithmic process:

1. Determine Cluster Centers: Start by determining the cluster centers, which involves choosing the number of RBFs. This can be achieved by performing a clustering operation, such as the k-means algorithm, to obtain the cluster means.

2. Solve for Weights: After determining the cluster centers, solve for the weights that connect the layer of RBFs to the output. This requires inverting a matrix that is dependent on the basis functions evaluated at the data points and the cluster means.

For a small network with just a few basis functions, matrix inversion is straightforward. However, with large data sets, the inversion process becomes computationally intensive. By utilizing the RLS algorithm, we can efficiently handle this complexity.

In conclusion, this procedure simplifies the learning algorithm for RBFNs, making it manageable even for large-scale problems. I encourage you to try implementing this algorithm on a data set to get a hands-on understanding of its effectiveness. This concludes our summary, and I hope this explanation clarifies the process.