Neural Networks for Signal Processing-I

Prof. Shayan Srinivasa Garani

Department of Electronic System Engineering

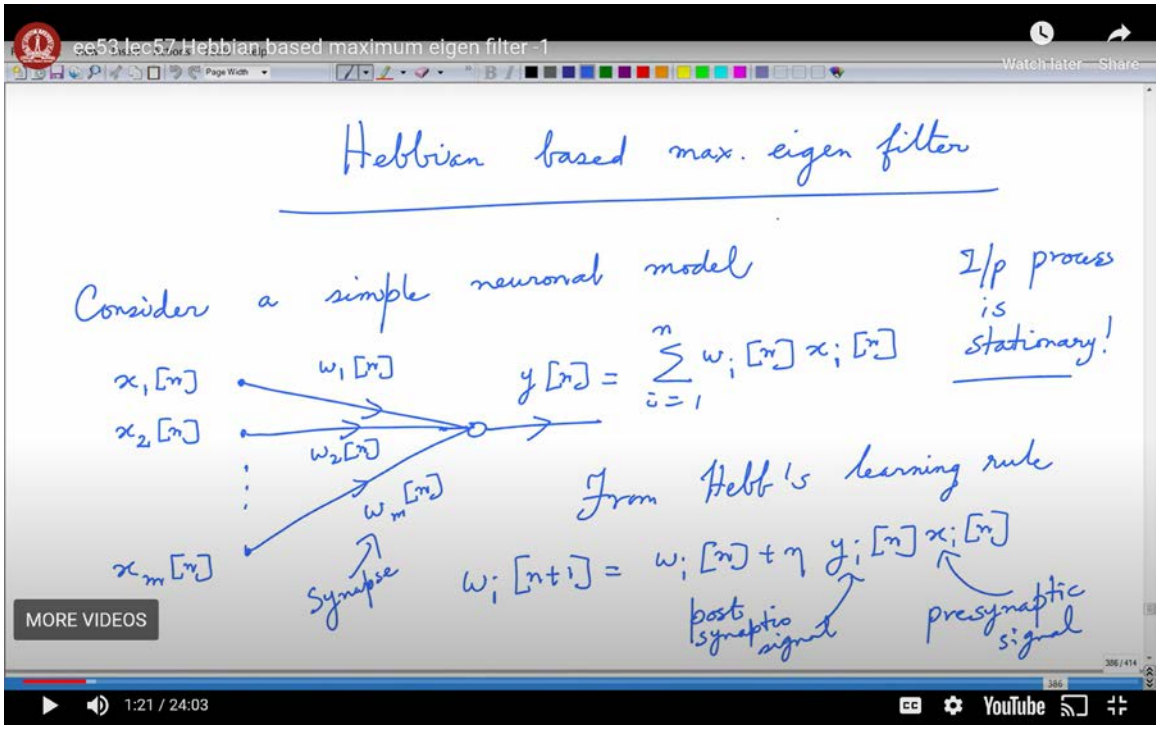
Indian Institute of Science, Bengaluru

Lecture – 57

Hebbian Based Maximum Eigen Filter -1

As we delve into the topics related to Principal Component Analysis (PCA) in the context of neural networks, let's explore how a single neuron, using a modified Hebbian update rule, can extract the principal eigenvector from a stationary input. This is a central concept driving our lecture.

(Refer Slide Time: 01:21)

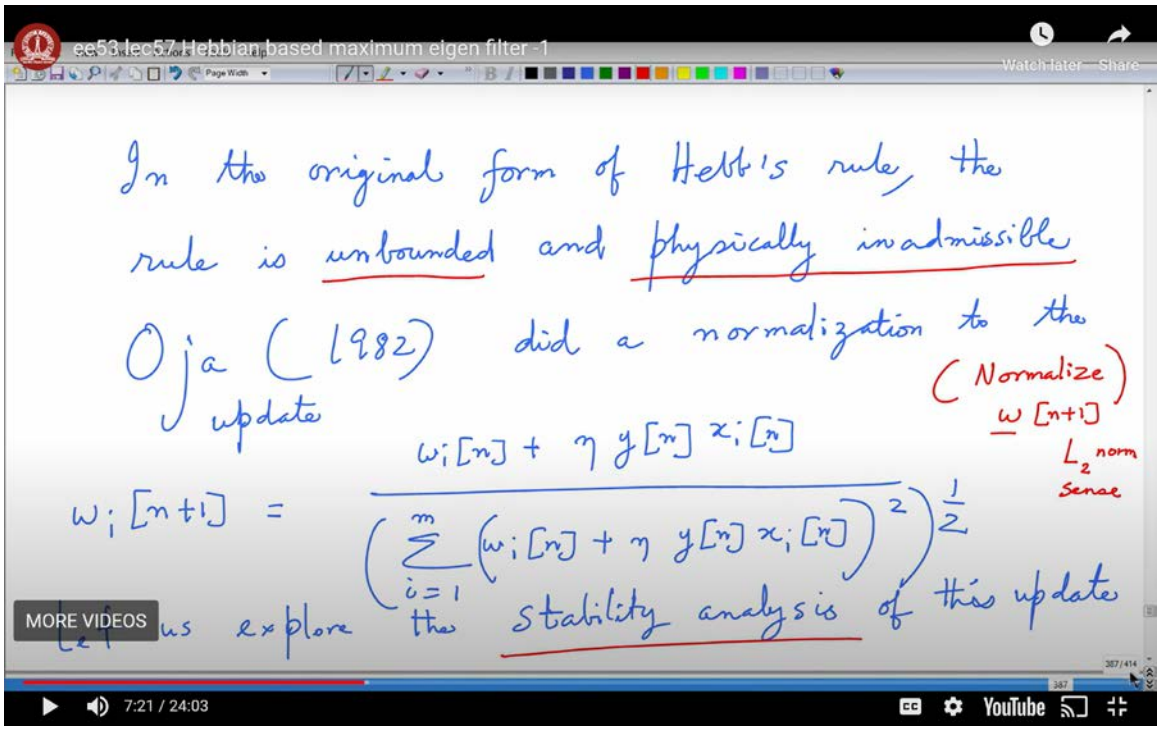


We begin with a simple neuronal model. Suppose we have an input vector x at time n, where x has m coordinates: x_1, x_2, …, x_m. The output y of the neuron is essentially the

projection of this input vector x onto the synaptic weight vector w. The weight vector w also has m components: w_1, w_2, …, w_m.

The output y is a linear combination of the weights, which can be interpreted as the projection of x onto w. We assume the input process is stationary, meaning that at any given time step, the statistical properties of the data vector remain constant. This is a crucial detail because it ensures that the data vectors we observe at time n have consistent statistical properties over time.

(Refer Slide Time: 07:21)



With these assumptions in place, we apply the Hebbian update rule. According to Hebb's principle, the synaptic weight update at time step n+1 is given by:

$$w_i(n+1) = w_i(n) + \eta \cdot y(n) \cdot x_i(n)$$

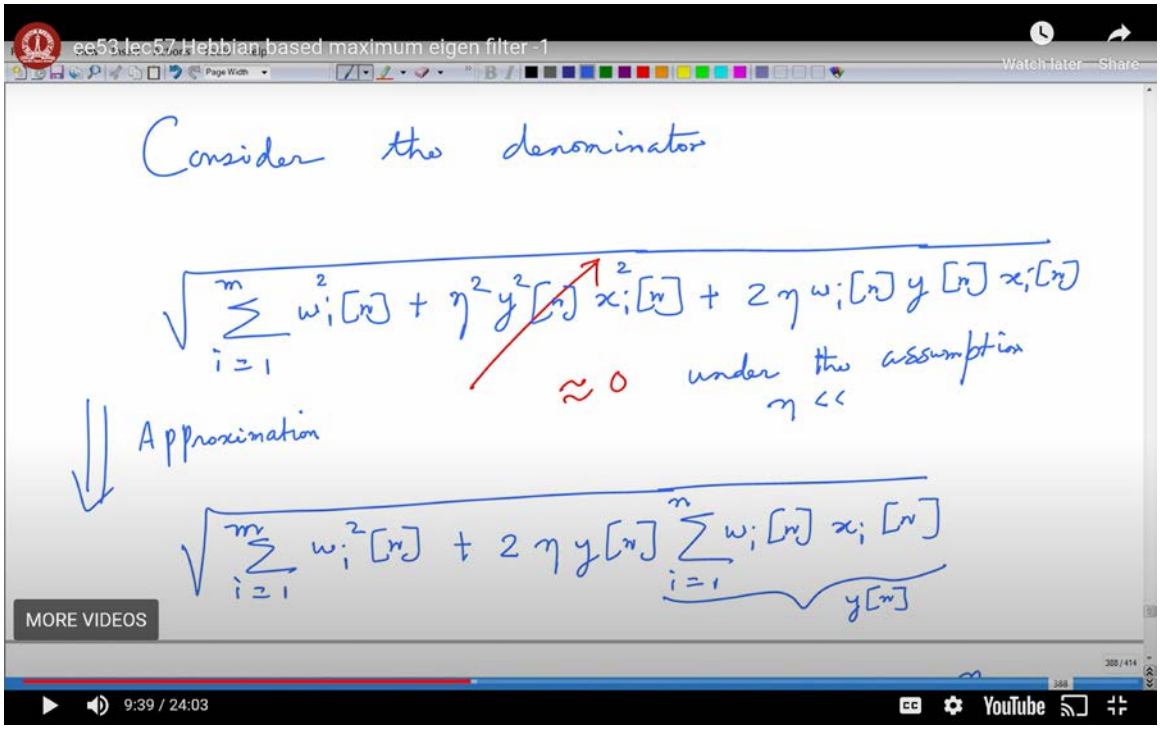
where η is the learning rate, a small positive quantity. Here, y(n) represents the output at time n, and $x_i(n)$ is the i-th component of the input vector at time n.

To simplify notation, we remove the subscript i for the output y since there is only one output. Therefore, the update rule becomes:

$$w(n+1) = w(n) + \eta \cdot y(n) \cdot x(n)$$

where x(n) represents the entire input vector and y(n) is the output at time n.

(Refer Slide Time: 09:39)



Before concluding, let's make a small correction: in the summation on the left side of the equation, we should use m rather than n to be consistent with the dimensions of the input vector, as depicted in the signal flow graph.

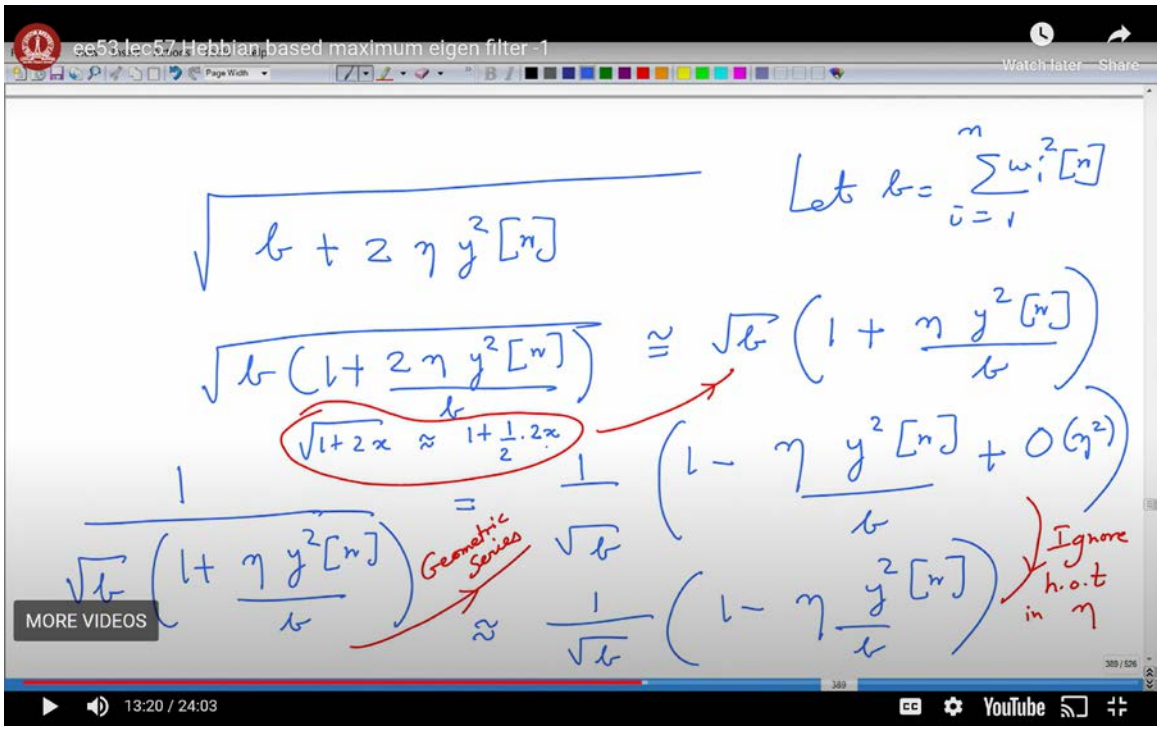
This update rule, as it stands, can become unbounded if x_i or y increases without bounds, leading to physically unrealistic results. Fortunately, a straightforward solution to this problem was proposed by Oya in 1982. The solution involves incorporating normalization into the update process. Specifically, after computing the update:

$$w_i(n+1) = w_i(n) + \eta \cdot y(n) \cdot x_i(n)$$

we normalize the weight vector by dividing it by the square root of the sum of the squares of the updated weights. This ensures that the weights do not grow indefinitely, maintaining the stability of the update process.

At each time step n+1, I normalize the weight vector, ensuring that each coordinate of the vector is scaled appropriately. This normalization is performed in the L_2 norm sense, meaning that the vector is scaled by its L_2 norm.

(Refer Slide Time: 13:20)



Next, let us examine the stability of this update rule and determine whether it provides a stable solution. We want to explore if this system remains stable and what insights can be derived from this analysis. Clearly, this is an adaptive system because the weights are updated at every time step n, and it appears to function as a nonlinear dynamical system. However, we will investigate whether it is truly nonlinear, which is the goal of our analysis.

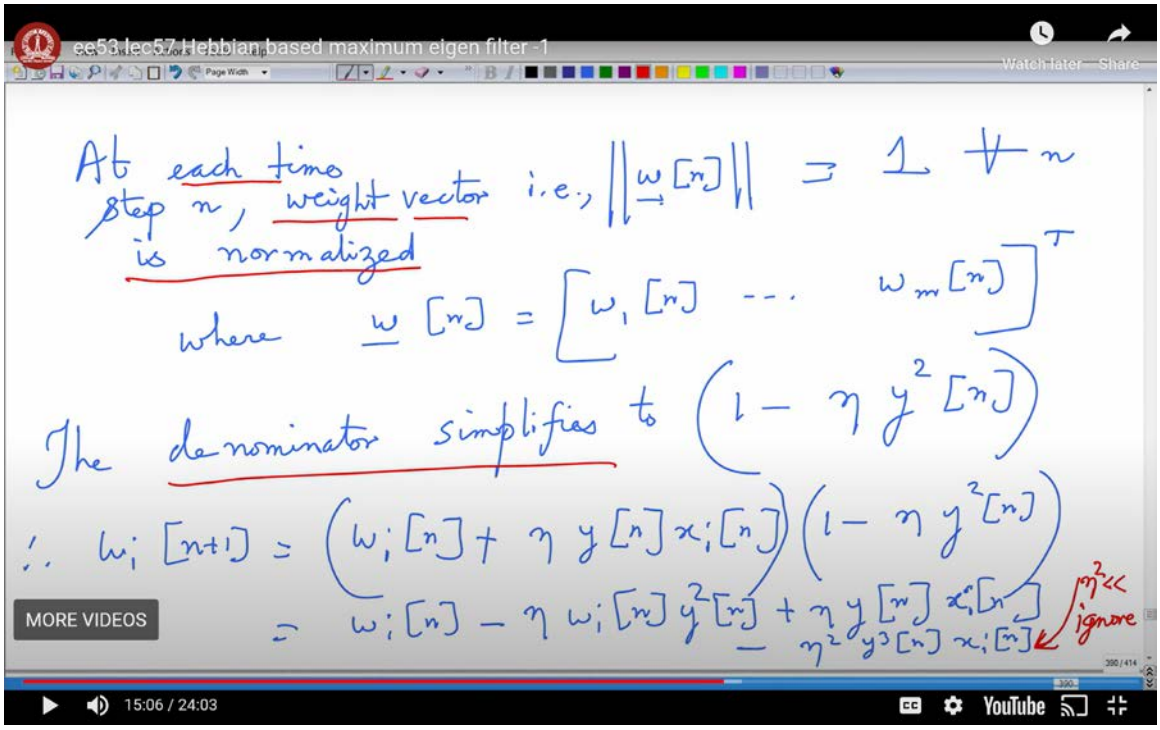
Consider the denominator of the normalization factor, which is:

$$\sqrt{\sum_{i=1}^{m}[w_i(n) + \eta \cdot y(n) \cdot x_i(n)]^2}$$

Expanding the square term inside the summation, we get:

$$[w_i(n) + \eta \cdot y(n) \cdot x_i(n)]^2 = w_i^2(n) + \eta^2 \cdot y^2(n) \cdot x_i^2(n) + 2\eta \cdot w_i(n) \cdot y(n) \cdot x_i(n)$$

(Refer Slide Time: 15:06)



Here, I expand $(a + b)^2$ into $a^2 + b^2 + 2ab$. Under the assumption that η is a small quantity, I can ignore the higher-order terms involving η^2 and higher. Thus, we approximate the expression as:

$$\sqrt{\sum_{i=1}^{m}[w_i^2(n) + 2\eta \cdot w_i(n) \cdot y(n) \cdot x_i(n)]}$$

Breaking this sum into two parts, we get:

$$\sqrt{\left(\sum_{i=1}^{m} w_i^2(n)\right) + 2\eta \cdot y(n) \cdot \left(\sum_{i=1}^{m} w_i(n) \cdot x_i(n)\right)}$$

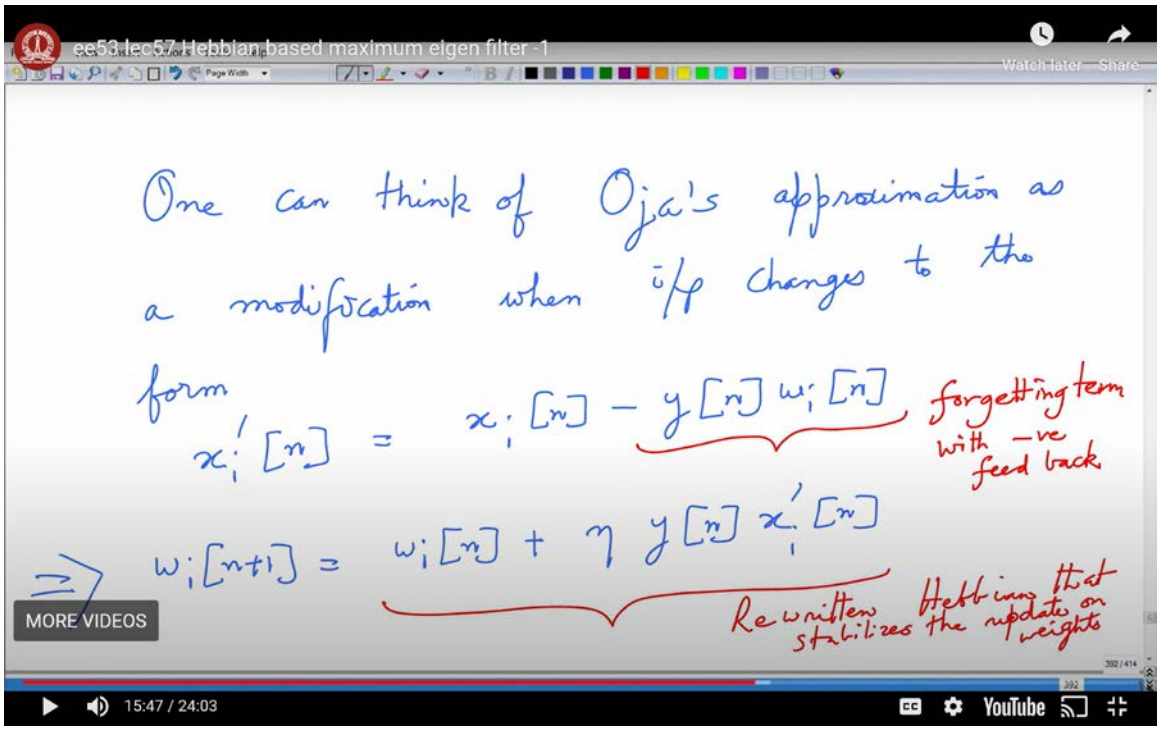
Let's denote $\sum_{i=1}^{m} w_i^2(n)$ by b. So the expression becomes:

$$\sqrt{b + 2\eta \cdot y^2(n)}$$

We can rewrite this as:

$$\sqrt{b\left(1 + \frac{2\eta \cdot y^2(n)}{b}\right)}$$

(Refer Slide Time: 15:47)



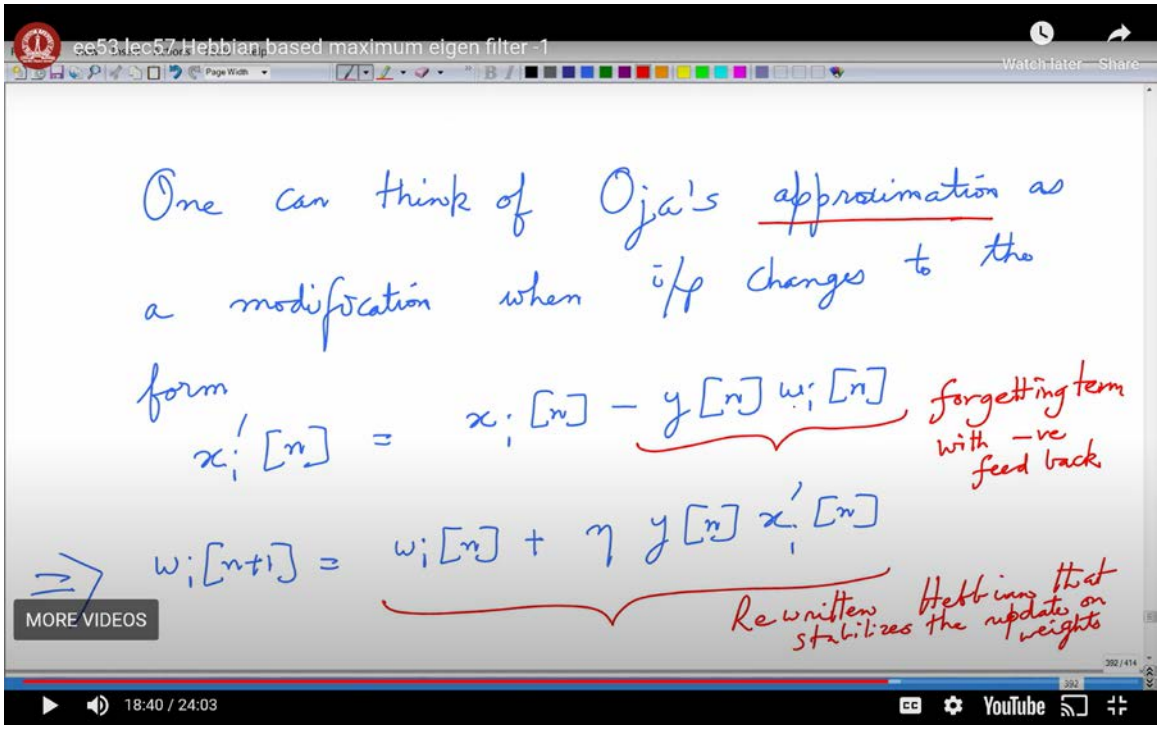
Applying the binomial approximation for the square root, which is $\sqrt{1+x} \approx 1 + \frac{1}{2}x$ for small x, we get:

$$\sqrt{b\left(1+\frac{2\eta\cdot y^2(n)}{b}\right)} \approx \sqrt{b}\left(1+\frac{\eta\cdot y^2(n)}{b}\right)$$

Thus, the denominator simplifies to:

$$\sqrt{b}\left(1+\frac{\eta\cdot y^2(n)}{b}\right)$$

(Refer Slide Time: 18:40)



Therefore, the ratio:

$$\frac{1}{\sqrt{b}\left(1+\frac{\eta\cdot y^2(n)}{b}\right)}$$

can be further simplified. Recognizing that $\frac{1}{1+x}$ can be approximated using the geometric series expansion as:

$$\frac{1}{1+x} \approx 1 - x$$

we get:

$$\frac{1}{\sqrt{b}\left(1 + \frac{\eta \cdot y^2(n)}{b}\right)} \approx \frac{1}{\sqrt{b}}\left(1 - \frac{\eta \cdot y^2(n)}{b}\right)$$

So, the denominator can be expressed approximately as:

$$\frac{1}{\sqrt{b}} - \frac{\eta \cdot y^2(n)}{b^{3/2}}$$

This adjustment allows us to address the stability and behavior of the update rule, providing insights into how the normalization affects the weight updates over time.

Ignoring higher-order terms in η, we focus on the linear term, which has the coefficient $\frac{y^2(n)}{b}$. Note that the binomial approximation should have been $1 + \frac{1}{2} \cdot 2x$, simplifying to 1 + x, rather than $1 + \frac{1}{2}x$.

Since b = 1 because we are normalizing the weight vector at every time step, the norm of W(n) is 1 for all n. Consequently, the denominator simplifies to $1 - \eta \cdot y^2(n)$. Here, y(n) represents the inner product of w with x, which is central to our analysis.

Now, if we use Oya's update rule, the weight at time step n+1 is given by:

$$w_i(n+1) = w_i(n) + \eta \cdot y(n) \cdot x_i(n)$$

In the numerator of Oya's update rule, and the denominator simplifies to:

$$1 - \eta \cdot y^2(n)$$

Expanding this, we get terms involving $w_i(n)$, η, and η^2. By ignoring higher powers of η (such as η^2 and above), we simplify the expression to:

$$w_i(n+1) = w_i(n) + \eta \cdot y(n) \cdot \big(x_i(n) - w_i(n) \cdot y(n)\big)$$

This is the simplified update rule up to the first power of η. The update rule shows how the weight vector is adjusted at each time step.

So, what does this mean for Oya's update? Initially, in the Hebbian update, the weight at time step n+1 is:

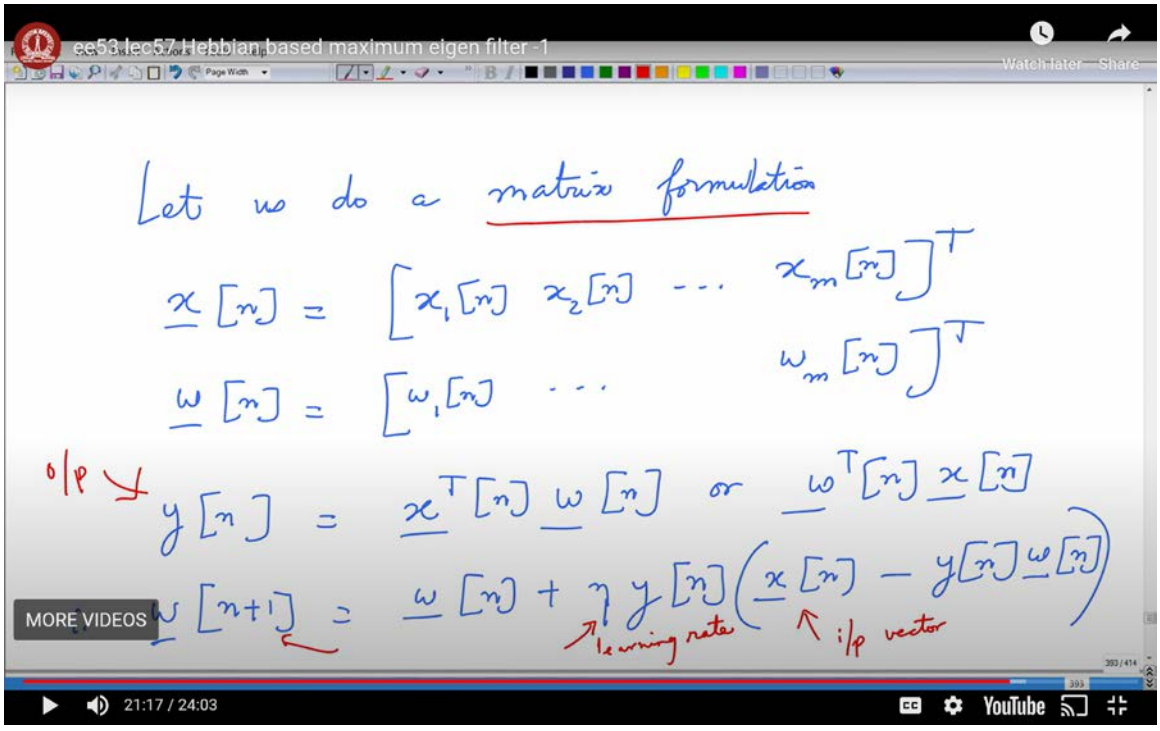
$$w_i(n+1) = w_i(n) + \eta \cdot (\text{correlation between input and output})$$

With the normalization step, it appears that the input is modified according to a forgetting factor. Specifically, if we consider:

$$x_i'(n) = x_i(n) - y(n) \cdot w_i(n)$$

Oya's normalization introduces a forgetting term into the input through negative feedback. This means that if $w_i(n)$ increases over time, the input is adjusted in a way that counteracts this growth. The feedback term y(n) is used to pull the weight back, preventing it from blowing up and ensuring stability.

(Refer Slide Time: 21:17)



Thus, the modified Hebbian update stabilizes the weights by incorporating this negative feedback mechanism. This approach effectively controls the growth of weights and ensures a stable solution, which would otherwise become unstable in the original Hebbian update.

Finally, while we previously discussed the scalar update for each synaptic weight, we now aim to update the weight vector as a whole in one step. This involves updating the entire weight vector simultaneously at a given time step, rather than updating each coordinate individually.

The weight vector update can be effectively expressed using matrix notation. If we consider x as a vector and w as the weight vector, we can cast the update rule into a matrix formulation. Let's delve into this approach.

Let,

$$x(n) = \begin{bmatrix} x_1(n) \\ x_2(n) \\ \vdots \\ x_m(n) \end{bmatrix}$$

represent our input vector with m coordinates, and let

$$w(n) = \begin{bmatrix} w_1(n) \\ w_2(n) \\ \vdots \\ w_m(n) \end{bmatrix}$$

be our weight vector, also with m components. The output y(n) is essentially the inner product of x and w, which can be expressed as either x^T w or w^T x. We will discuss which form is more useful depending on the algebraic manipulations required for matrix operations

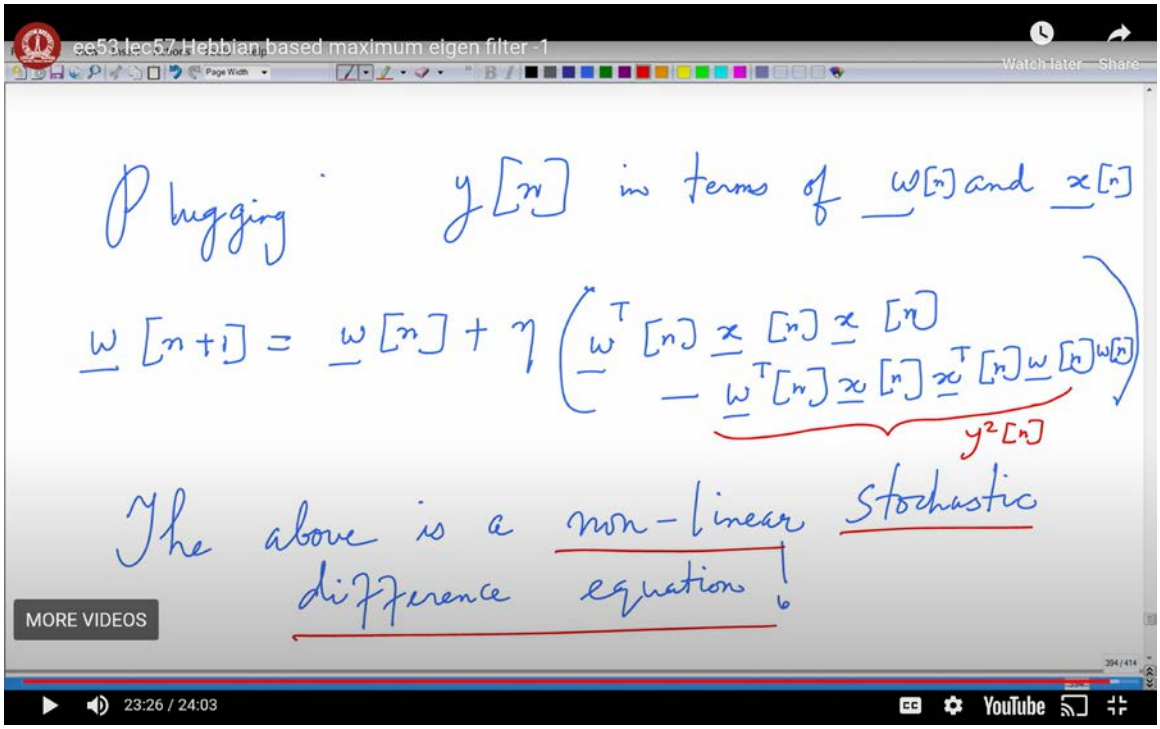
Both forms, x^T w and w^T x, are mathematically equivalent, but their usefulness becomes apparent when performing specific matrix operations. For clarity, we will use both forms as needed in our analysis.

The weight vector update at time step n+1 is given by:

$$w(n+1) = w(n) + \eta \cdot y(n)$$

where η is the learning rate, which we assume to be constant but could vary with time. Here, x(n) represents the input vector at time step n, and w(n+1) is the updated weight vector.

(Refer Slide Time: 23:26)



To simplify this, substitute y(n) in terms of w and x. We get:

$$w(n+1) = w(n) + \eta \cdot [w^T(n)x(n) \cdot x(n) - w^T(n)x(n) \cdot x^T(n)w(n)]$$

In this expression, y(n) is a scalar, represented as either w^T x or x^T w. To understand the advantages of these forms, recall that x x^T represents the outer product of x, and taking its expectation yields the correlation matrix. This leads to a more elegant quadratic form:

$$w^T(\text{correlation matrix})w$$

This quadratic form is more streamlined than arbitrary combinations of x and w.

The above equation represents a nonlinear stochastic difference equation. While normalization is straightforward to implement in practice and can be effective, a deeper analysis is required to understand its theoretical underpinnings. This rigor is essential in a graduate-level course. With this overview, we stop this module and will proceed with further topics in our course.