Neural Networks for Signal Processing – I

Prof. Shayan Srinivasa Garani

Department of Electronic Systems Engineering

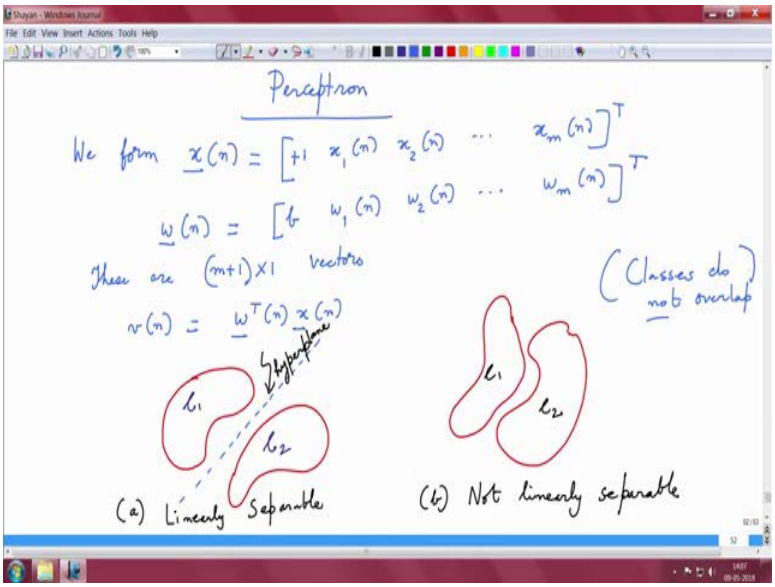
Indian Institute of Science, Bengaluru

Lecture – 09

Perceptron 2

We have just discussed the necessity of an equation for a hyperplane, which is expressed in the form $w^T x + bias = 0$. Now, let us delve into the process of devising an algorithm to derive this equation and subsequently demonstrate the validity of our approach.

(Refer Slide Time: 00:44)



To proceed, let us define some variables. We will construct a vector x_n, starting with a 1 as the first coordinate, followed by x_{1n}, x_{2n}, and so forth up to x_{mn}. This vector is then transposed, resulting in an (n+1)-dimensional vector.

In a similar fashion, we define the weight vector w_n, where the first coordinate is b, representing the bias. This is followed by w_{1n}, w_{2n}, up to w_{mn}, and is also transposed. Thus, these vectors are of dimension $(m + 1) \times 1$.

Here, n signifies that we are operating in an online mode, meaning we receive these vectors sequentially at each time step n. The time order is indicated by n.

Now that we have defined these two vectors, we can define $v_s(n)$ as the inner product between x_n and w_n. This can be expressed as $w_n^T x_n$.

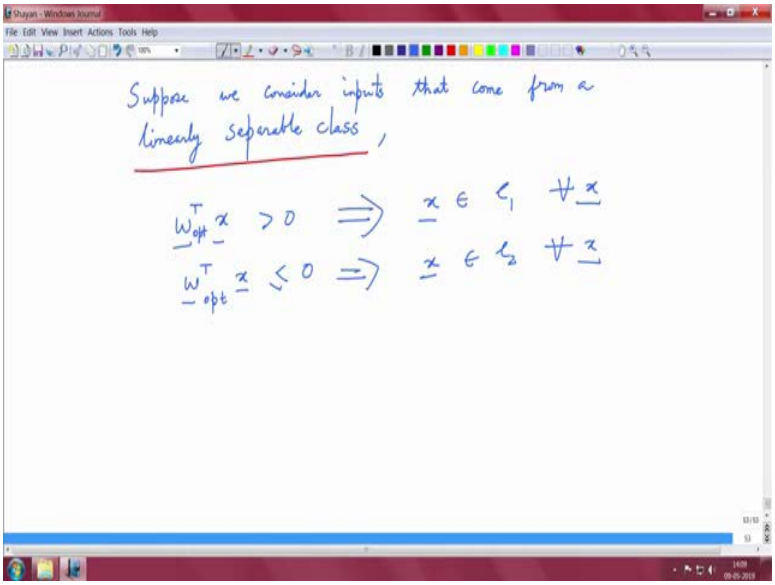
One crucial point to highlight is that, for the perceptron algorithm to function correctly, if you have a two-class problem, these classes must be linearly separable.

Let's visualize the concept of linear separability to better understand what is happening. Imagine two distributions representing two classes: class c_1 and class c_2. If we can find a hyperplane that successfully separates these two classes, then we can say that classes c_1 and c_2 are linearly separable. In other words, these classes are linearly separable if a hyperplane can be drawn to partition the space between them.

Now, consider another scenario where the two classes are nested within each other. Suppose c_1 is nested within c_2. In this case, it is impossible to draw a hyperplane that separates c_1 from c_2. Therefore, these classes are not linearly separable. However, we could use a curve to separate the two classes, assuming they do not overlap.

It is crucial to note that this explanation assumes that the classes do not overlap. If the classes do overlap, the concept of linear separability becomes meaningless.

(Refer Slide Time: 05:39)

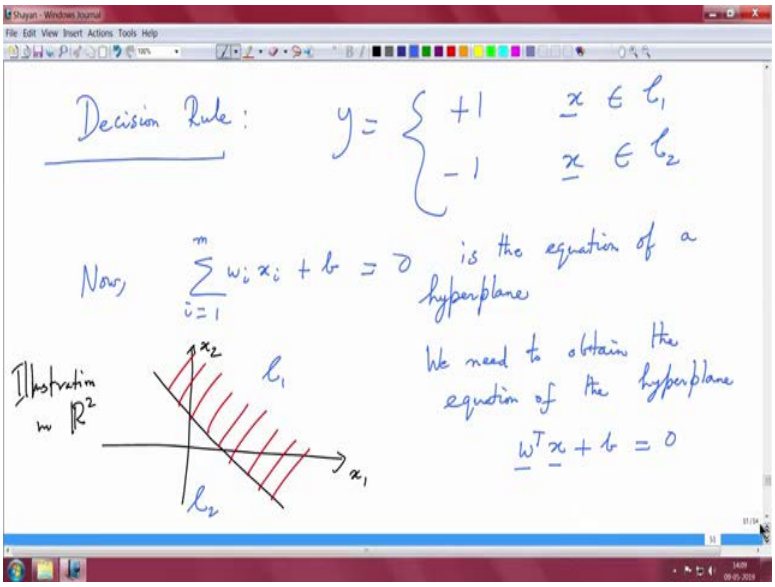


Now, let's delve into more concrete terms. Suppose we consider inputs from a linearly separable class. If we have $w^T x > 0$, there exists an optimal weight vector w_{optimal} such that

$w_{\text{opt}}^{T} x > 0$. This implies that x belongs to class c_1 for all such x.

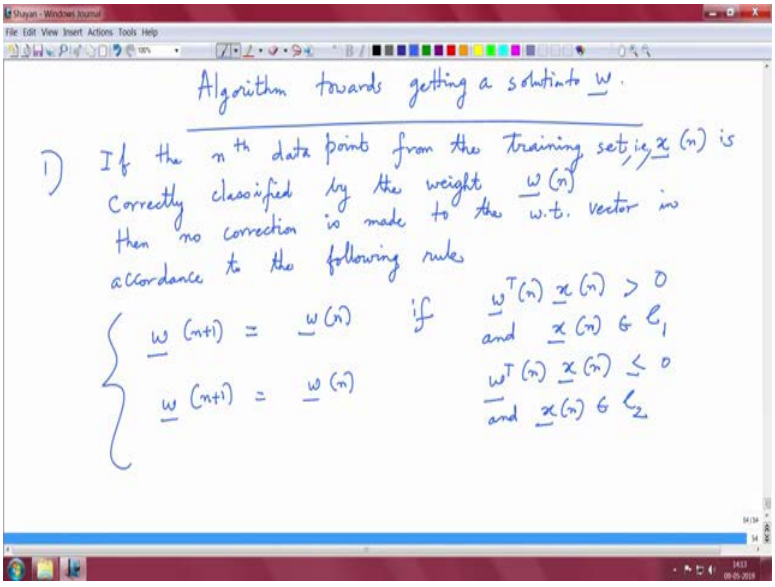
Conversely, if this inner product w_{optimal}^{T} x is less than or equal to 0, then x belongs to class c_2. This holds true for all x whose dot product with w_{optimal} is negative or zero.

(Refer Slide Time: 07:20)



This concept becomes clearer when we visualize it using a diagram. Imagine a hyperplane dividing the space: everything on one side of the hyperplane belongs to class c_1, while everything on the other side belongs to class c_2.

(Refer Slide Time: 07:44)



This equation effectively describes this geometric separation. To achieve the optimal value of w, we need an algorithm. When we talk about "optimal," we mean finding a solution that ensures linear separability. Let me outline this algorithm in two steps:

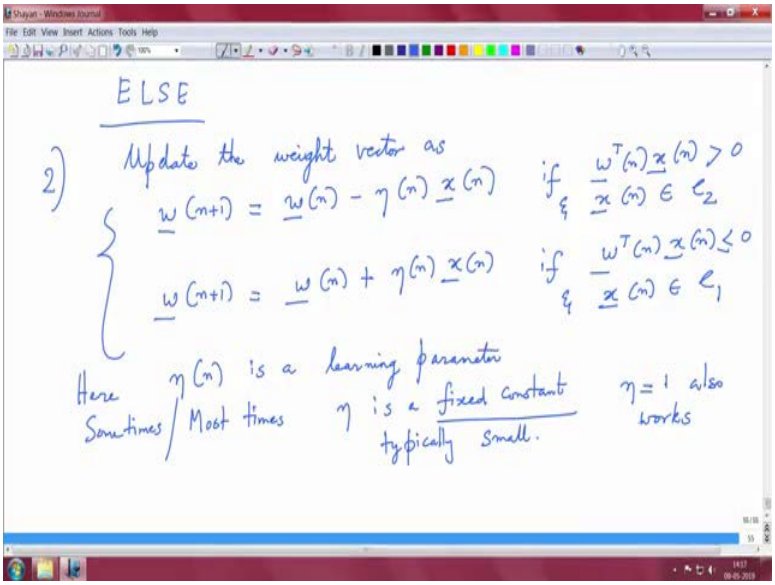
1. If the nth data point from the training set, x_n, is correctly classified by the weight w_n, no correction is made to the weight vector. Initially, there is some initialization, but we will discuss that after completing the procedure. The rule is as follows:

If $w_n^T x_n > 0$ and x_n belongs to class c_1, then $w_{n+1} = w_n$. In this case, we do not modify the weight because it is already correct.

Similarly, if the inner product $w_n^T x_n \leq 0$ and x_n belongs to class c_2, no adjustment is necessary. Though the first part of the statement implies the second, it is stated explicitly for clarity.

So, there is no need to change the weight if the data point lies naturally where it should. If the data point x_n lies on the side of the hyperplane where the inner product is positive and belongs to c_1, or the inner product is non-positive and it belongs to c_2, then the current weights are correct.

(Refer Slide Time: 11:47)



2. If the classification is incorrect, we must update the weight vector accordingly. So, if the data point is misclassified, we need to update the weight vector as follows. Let's explore this update rule:

-The weight adjustment at time n+1 is given by:

$$w_{n+1} = w_n - \eta_n x_n$$

where η_n can be a constant learning rate, if $w_n^T x_n > 0$ and x_n belongs to class c_2.

Conversely, if $w_n^T x_n \leq 0$ and x_n belongs to class c_1, the weight adjustment is given by:

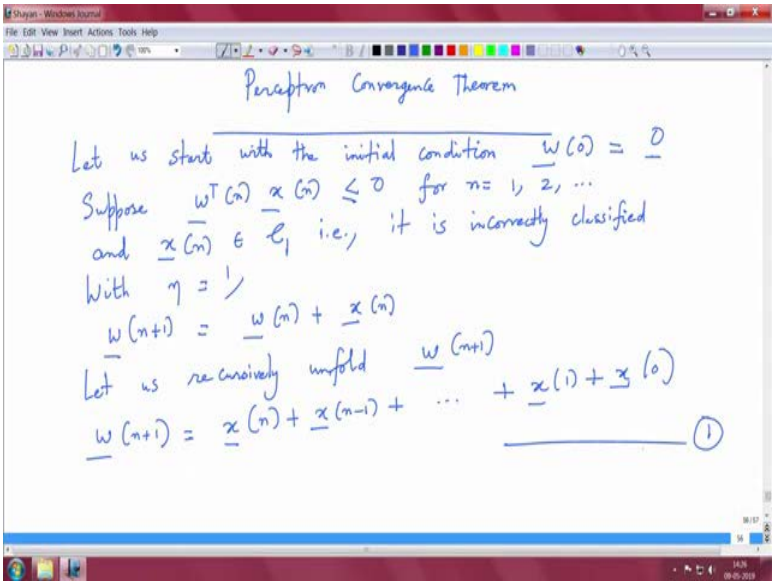
$$w_{n+1} = w_n + \eta_n x_n$$

To explain how this works, consider a situation where the inner product w_n^T x_n is greater than 0, but the data point actually belongs to class c_2 instead of c_1. This means the current weight should be reduced in the direction of x_n. Hence, we subtract $\eta_n x_n$, scaling it by a factor η_n.

Similarly, if the inner product is less than or equal to 0 and the data point belongs to class c_1, we need to add to the weight in the direction of x_n, scaled by η_n. So, the update involves adding η_n x_n in this case.

Here, η_n is a learning parameter, which is often fixed and independent of n. Typically, η is a small constant, and in some cases, it can even be set to 1 to simplify the algorithm's dynamics.

(Refer Slide Time: 16:14)



The idea behind this algorithm is straightforward, much like balancing a scale. If the weight is too high on one side, you adjust by redistributing the weight. This intuitive approach forms the basis of the perceptron algorithm.

One of the key questions now is whether we can prove that this algorithm converges for a linearly separable class. This will be our next step as we delve into the details of the proof.

To recap what we discussed earlier, I didn't specify the initial conditions at that time. The idea was to start with an arbitrary weight vector and adjust it based on whether the classification was correct or incorrect. If the classification was correct, no adjustment was needed. If it was incorrect, the weight would be adjusted by adding or subtracting a value depending on the inner product with the data point.

Now, let's establish the initial condition: we start with the weight vector at time step 0 being the zero vector, $w_0 = 0$.

Given this, suppose $w_n^T x_n$ is less than 0 for n = 1, 2, ..., and x_n belongs to class c_1. If x_n belongs to class c_1, the inner product should have been greater than or equal to 0. This discrepancy indicates that the data point is incorrectly classified by the weight at time step n.

Assuming a learning rate $\eta = 1$, we follow our update rule. What should we do? The weight at time n+1 is updated as follows:

$$w_{n+1} = w_n + x_n$$

This specific case considers x_n belonging to class c_1, but the same logic applies if x_n belongs to class c_2. The update rule will adjust accordingly, maintaining consistency without loss of generality.

Now, let’s unfold this recursively. The weight at time n+1 can be expressed as:

$$w_{n+1} = w_n + x_n$$

Expanding w_n:

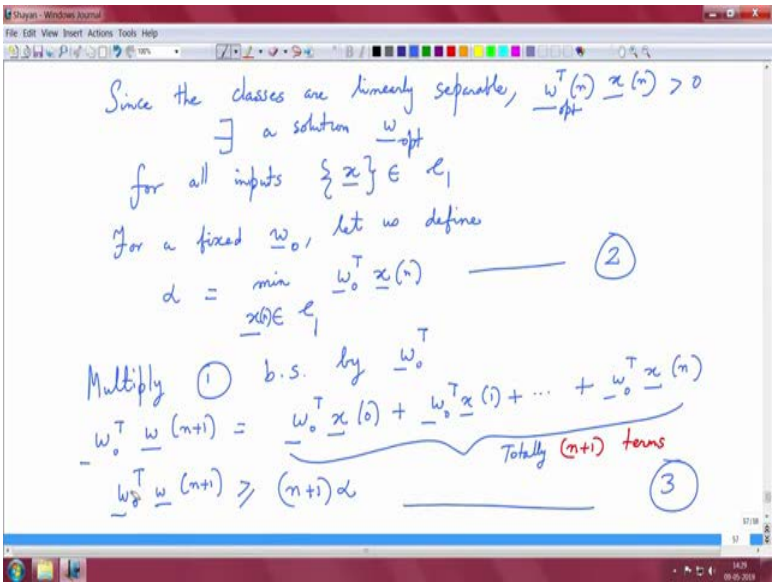
$$w_n = w_{n-1} + x_{n-1}$$

Continuing this process iteratively, we get:

$$w_{n+1} = x_n + x_{n-1} + \cdots + x_1 + x_0$$

It's important to note that $w_1 = w_0 + x_0$, and since $w_0 = 0$, the equation is accurate. Let's label this equation as Equation 1 for future reference.

(Refer Slide Time: 20:43)



Since the classes are linearly separable, there exists a solution w_0 (or w_{optimal}) such that $w_{\text{opt}}^T x > 0$, for all inputs x belonging to class c_1. This means that, given all inputs x from class c_1, there should exist an optimal hyperplane that satisfies this condition due to the linear separability of the classes.

For a fixed w_{opt} (which we'll denote as w_0 to avoid confusion with the subscript 0), let's define a quantity α as the minimum value of $w_{\text{opt}}^T x$ over all x in class c_1. We can label this as Equation 2.

Now, for illustration purposes, let's use an index n to indicate the time step when x is presented. This means we denote the data point at time n as x_n.

Next, let's multiply both sides of Equation 1 by w_0^T. It's important to note that for the optimal hyperplane, the time index is irrelevant, so we deliberately omit it. By multiplying Equation 1 by w_0^T, we get:

$$w_0^T w_{n+1} = w_0^T x_0 + w_0^T x_1 + \cdots + w_0^T x_n$$

Since there are n+1 terms, and if each of these terms is lower-bounded by α, we can state that:

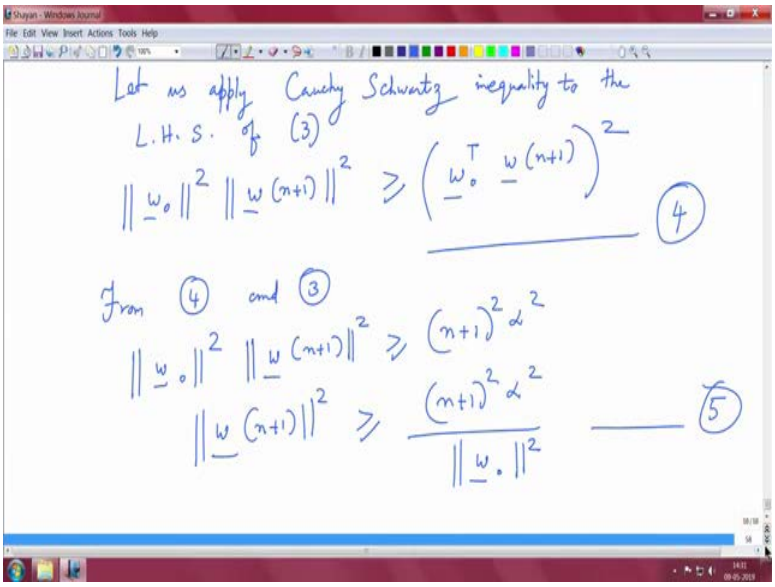
$$w_0^T w_{n+1} \geq (n+1)\alpha$$

This follows because α is the minimum value, so the sum of these n+1 terms is at least

$(n+1)\alpha$. Let's label this as Equation 3.

To proceed further, we need to relate this in terms of norms. Therefore, we must introduce the Cauchy-Schwarz inequality, which will help us establish the required relationship.

(Refer Slide Time: 26:50)



Let's apply the Cauchy-Schwarz inequality to the left-hand side of the equation we derived earlier, $w_0^T w_{n+1}$. For a detailed proof of the Cauchy-Schwarz inequality, you can refer to educational videos on YouTube. In fact, in the "Mathematical Methods and Techniques for Signal Processing" course, I have proved the Cauchy-Schwarz inequality, and you can review that proof, or consult any other reliable learning material.

Now, what do we get from applying this inequality? The Cauchy-Schwarz inequality tells us that:

$$|w_0|^2 |w_{n+1}|^2 \geq (w_0^T w_{n+1})^2$$

This is essentially stating that the product of the squared norms of w_0 and w_{n+1} is greater than or equal to the square of their inner product. Let's call this Equation 4. We aim to link this to our earlier bound involving (n+1) α.

From Equations 3 and 4, we have:

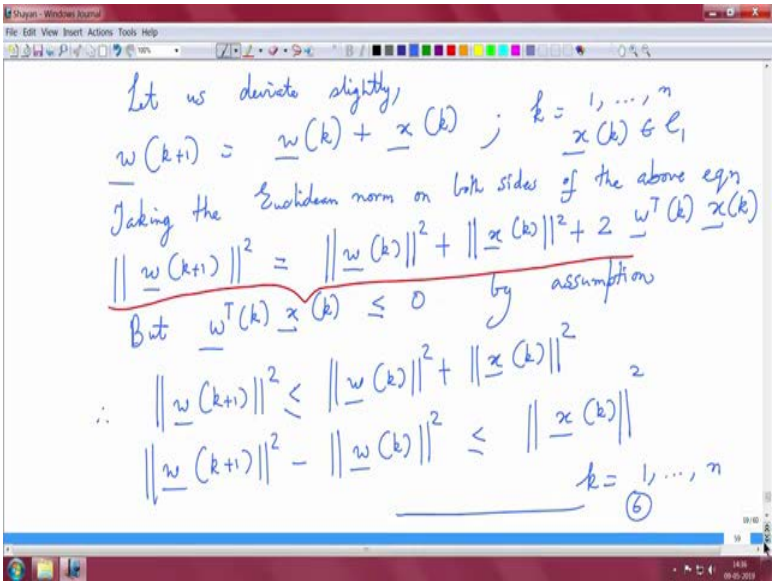
$$|w_0|^2 |w_{n+1}|^2 \geq (n+1)^2 \alpha^2$$

Rewriting this, we get:

$$|w_{n+1}|^2 \geq \frac{(n+1)^2\alpha^2}{|w_0|^2}$$

Here, it's important to note that w_0 is not the zero vector, avoiding a divide-by-zero scenario.

(Refer Slide Time: 30:03)



Next, let's consider the recursive relationship $w_{k+1} = w_k + x_k$ for k = 1, ..., n and all x_k belonging to class c_1. Taking the Euclidean norm on both sides of this equation, we get:

$$|w_{k+1}|^2 = |w_k|^2 + |x_k|^2 + 2w_k^T x_k$$

Since we are updating the weights based on the assumption $w_k^T x_k \leq 0$, the term $2\, w_k^T x_k$ is non-positive. Therefore:

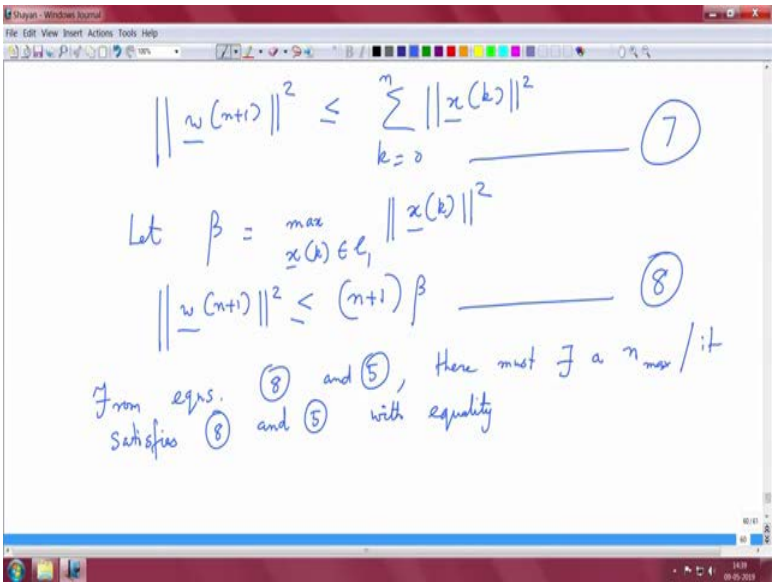
$$|w_{k+1}|^2 \leq |w_k|^2 + |x_k|^2$$

This provides a recursive relationship in terms of the norms of these vectors. Rearranging this, we get:

$$|w_{k+1}|^2 - |w_k|^2 \leq |x_k|^2$$

This inequality holds for k = 1, ..., n, as we sequentially process all data points.

(Refer Slide Time: 33:45)



Now, if we recursively expand this, it becomes straightforward. We will arrive at:

$$|w_{n+1}|^2 \leq \sum_{k=0}^{n} |x_k|^2$$

Let us call this Equation 6. This derivation is straightforward: taking each term from w_{n+1} to w_0, and noting that w_0 is the zero vector, we obtain:

$$|w_{n+1}|^2 \leq \sum_{k=0}^{n} |x_k|^2$$

Now, we need to bound this sum. Let β be defined as the maximum of $|x_k|^2$ over all x_k belonging to class c_1. By choosing the maximum, we establish an outer bound. Thus, we have:

$$|w_{n+1}|^2 \leq (n+1)\beta$$

Let's call this Equation 7.

From Equations 3 and 6, on one side we have:

$$|w_{n+1}|^2 \geq \frac{(n+1)^2\alpha^2}{|w_0|^2}$$

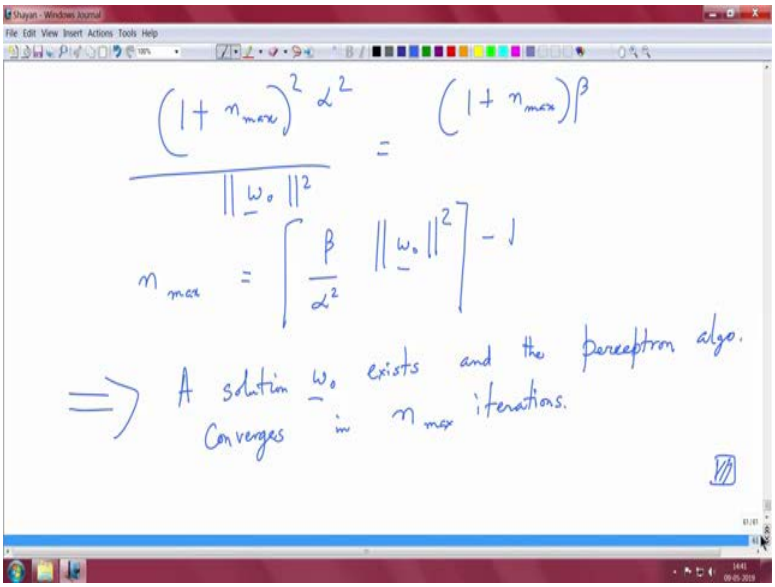
And from Equation 7, we have:

$$|w_{n+1}|^2 \leq (n+1)\beta$$

There must be a point where these two inequalities meet, implying equality. From Equations 6 and 7, there must exist an n_{max} such that these equations are satisfied with equality. Therefore, we equate the terms:

$$\frac{(n_{max}+1)^2\alpha^2}{|w_0|^2} = (n_{max}+1)\beta$$

(Refer Slide Time: 37:25)



Solving for n_{max}, we get:

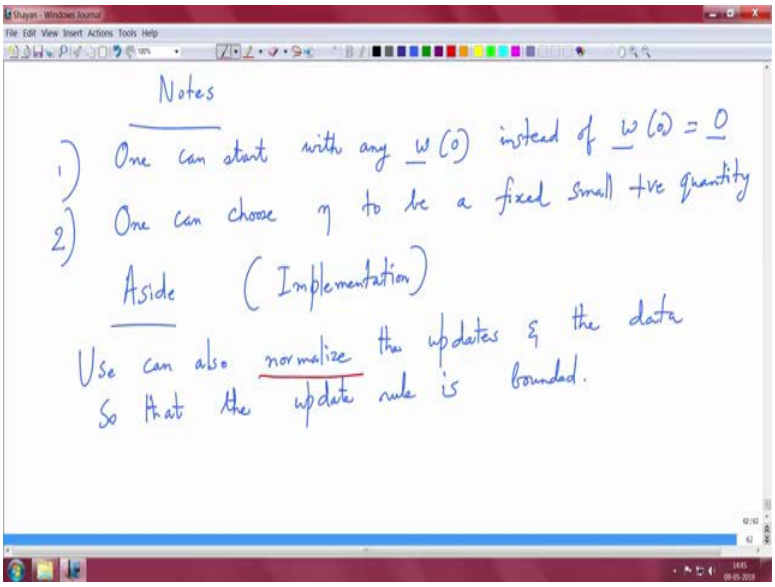
$$n_{max} = \left(\frac{\beta\,|w_0|^2}{\alpha^2}\right) - 1$$

Since n_{max} must be an integer, we round up to the nearest integer. This implies a solution w_0 exists and the perceptron algorithm converges in n_{max} iterations. This should be intuitively clear: because the classes are linearly separable, there must exist a hyperplane. By adjusting the weight vector by adding or subtracting a quantity iteratively, we are able to find a value of n_{max} that terminates the algorithm. This is a significant proof.

You may wonder how to start the algorithm. One can start with any initialization, such as $w_0 = 0$ or a random value, and iterate until convergence. Also, you may question the choice of η. While we fixed η to be 1, it can be adjusted to a small value, and the algorithm will still

converge.

(Refer Slide Time: 39:57)



So, a couple of points to note here: you have the flexibility to start with any initial vector instead of sticking to the all-zero vector, which simplifies updates. Starting with zero is convenient, but you can choose any starting point. Another consideration is the choice of η, which can be a fixed small positive quantity. Typically, η is chosen between 0 and 1, and the update process should function effectively within this range. If you find that updates exceed desired bounds, data normalization can be employed to guide the algorithm towards convergence.

From an implementation standpoint, especially with a large number of data points, normalizing both the updates and the data itself can help keep the update rule within manageable limits. This approach ensures that the algorithm remains stable and efficient. Additionally, shuffling the data before presenting it to the perceptron can further enhance the algorithm's performance by preventing any inherent order bias in the data presentation. These practices contribute to robust and effective implementation of the perceptron algorithm.

Let's delve into the practical aspects here. Imagine you're working with a set of all points, sifting through c_1 and ensuring correctness, then transitioning to c_2. To prevent the weights from escalating in a particular direction, you can shuffle the data points. For instance, in an update equation where you add points sequentially like $w_{n+1} = w_n + x_n$, arranging your data such that you consistently encounter such situations can be managed by shuffling.

Shuffling ensures a mix of positive and negative influences, thus keeping the updates bounded. The choice of weights is critical too; preventing linear increases and maintaining normalization are key considerations in the practical application of this algorithm. This approach ensures stability and effectiveness in handling different types of data distributions encountered in training scenarios.

The most effective way to understand the algorithm is by implementing it yourself. As you start coding and working with the data points, you'll gain a deeper grasp of the small algorithmic adjustments that complement the core concept. This hands-on approach not only solidifies your understanding but also highlights the practical nuances of the algorithm. With this, we conclude the proof of convergence for the perceptron algorithm. Next, we will explore the batch perceptron algorithm and look into its convergence proof.