Neural Networks for Signal Processing-I

Prof. Shayan Srinivasa Garani

Department of Electronic System Engineering

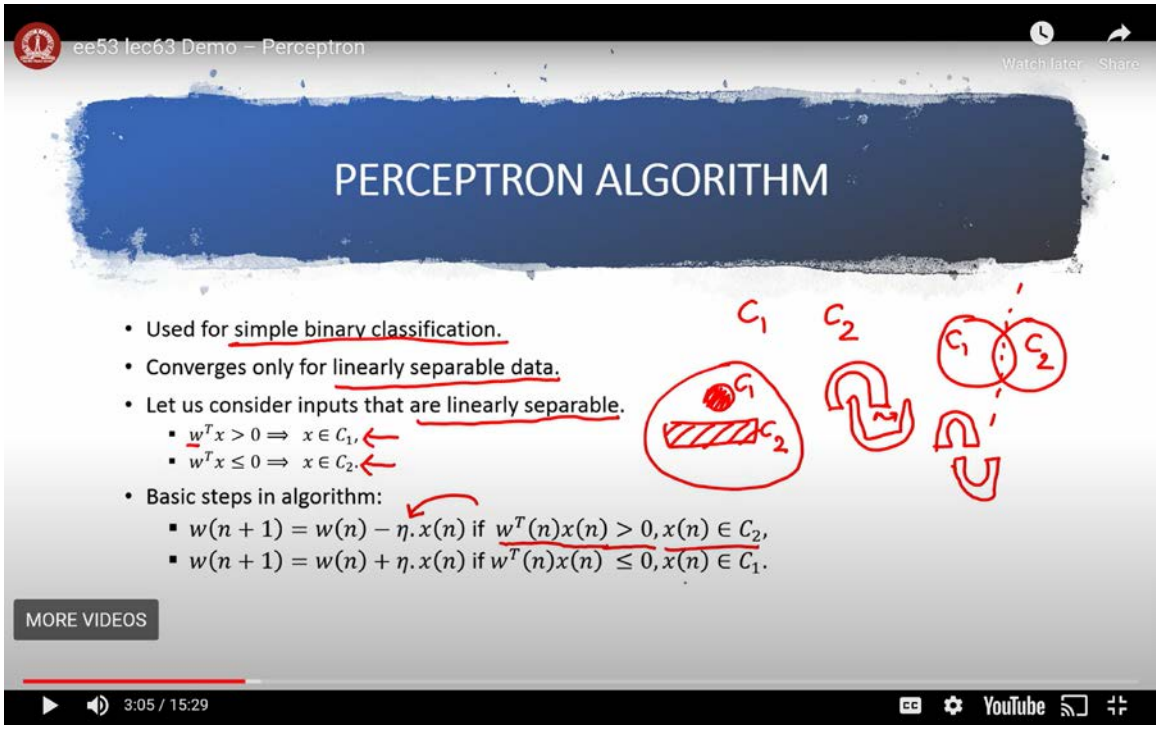
Indian Institute of Science, Bengaluru

Lecture – 63

Demo – Perceptron

Hello everyone, let's dive into a demonstration of the perceptron algorithm. As we have covered in this course, the perceptron algorithm is designed for solving simple binary classification problems. Specifically, it is used when we have two classes, C_1 and C_2, which need to be separated by a hyperplane.

(Refer Slide Time: 03:05)

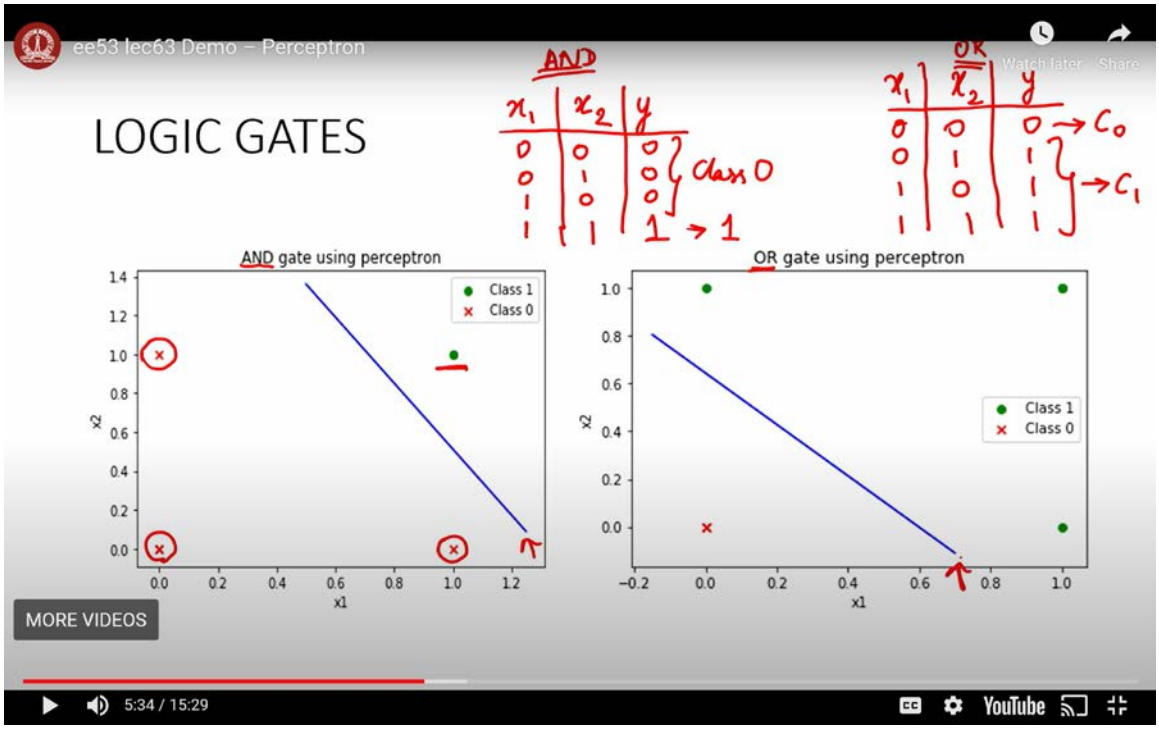


We also know that the perceptron algorithm converges only for linearly separable data. This means that if the data can be separated into C_1 and C_2 using a hyperplane, the

algorithm will work. However, if the data is not linearly separable, meaning there is no single hyperplane that can separate C_1 from C_2, the perceptron algorithm will fail.

Let's consider a scenario where our input data is linearly separable. In such a case, the perceptron algorithm can find a weight vector w such that $\boldsymbol{w}^T x > 0$ for all inputs x belonging to class C_1, and $\boldsymbol{w}^T x \leq 0$ for all inputs belonging to class C_2. For example, if we have two classes of data, C_1 and C_2, where C_1 is represented as one set of points and C_2 as another, and these classes are visually separable by a hyperplane, the perceptron algorithm will successfully find this separating hyperplane.

(Refer Slide Time: 05:34)



However, if the data is not linearly separable, such as when the classes form complex shapes like the "two moons" problem, the perceptron algorithm struggles. If the distance between the two classes increases and they become linearly separable, then the perceptron algorithm can find a suitable weight vector.

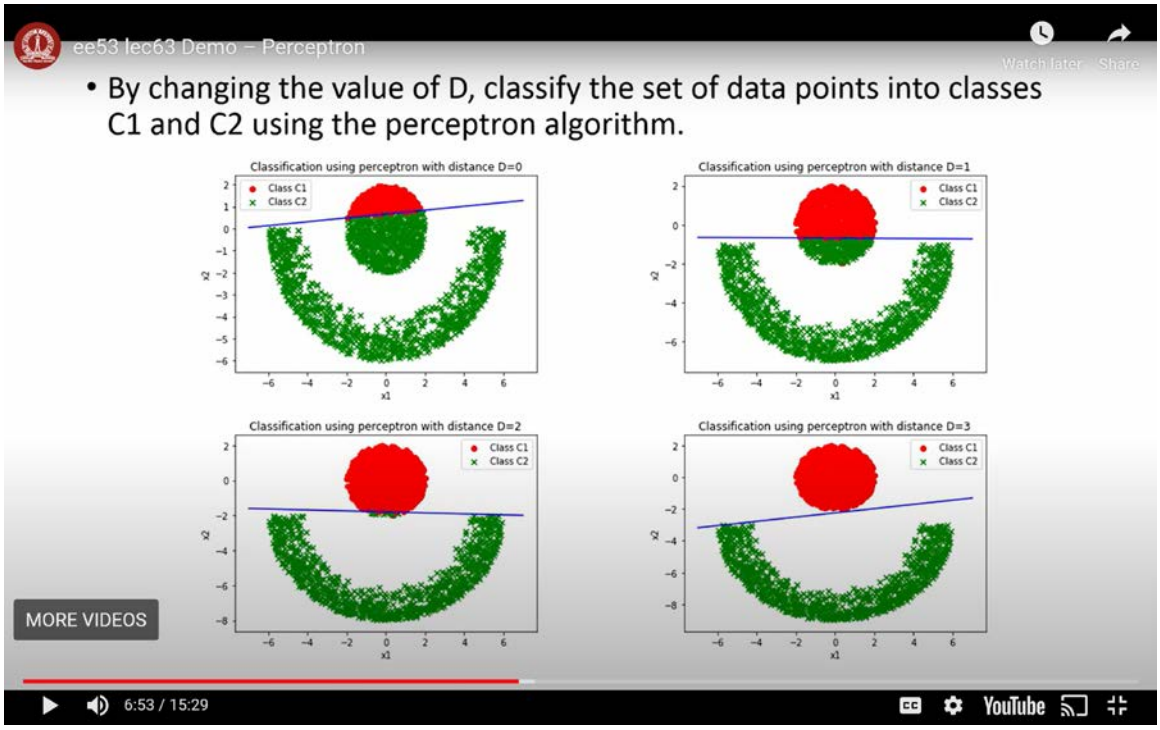
Assuming the data is linearly separable, the perceptron algorithm involves the following basic steps:

1. Compute the inner product between the weight vector and the input vector.

2. If the inner product is greater than 0 for an input that actually belongs to class C_2, this input is misclassified. To correct this, we update the weight by subtracting a term proportional to the learning rate multiplied by the input vector.

3. Conversely, if an input belongs to class C_1 but the inner product is less than or equal to 0, this input is also misclassified. Here, we need to increase the weight by adding a term proportional to the learning rate multiplied by the input vector.

(Refer Slide Time: 06:53)



These two steps are fundamental to the perceptron algorithm, where weight adjustments are made based on whether the classification is correct or not.

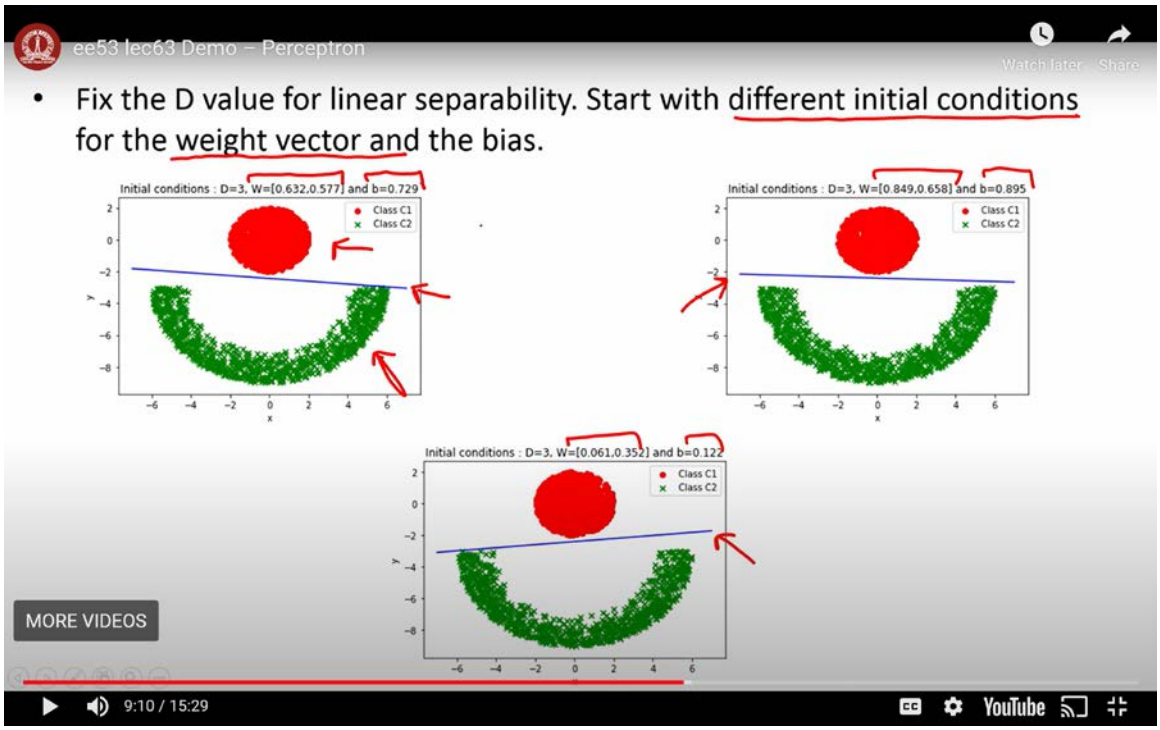
Now, let's examine a simple example using logic gates. Consider the AND gate, with inputs x_1 and x_2 and the corresponding output label y. The truth table for the AND gate is as follows:

- For inputs (0, 0), (0, 1), and (1, 0), the output is 0.

- For the input (1, 1), the output is 1.

We classify the inputs (0, 0), (0, 1), and (1, 0) as class 0 and the input (1, 1) as class 1. If we plot these points, we see that (0, 0) is class 0, (0, 1) is class 0, (1, 0) is class 0, and (1, 1) is class 1.

(Refer Slide Time: 09:10)



We observe that the perceptron algorithm effectively identifies a hyperplane that can linearly separate the two classes. Now, let's apply this to the OR gate. Referring to the truth table for the OR gate, we see that the classes are indeed linearly separable. Specifically, for inputs x_1 and x_2, with the output y, the output is 0 only for the input (0, 0), and 1 for all other inputs. Thus, (0, 0) belongs to class 0, while (0, 1), (1, 0), and (1, 1) belong to class 1.

In both the AND and OR gate cases, because the data is linearly separable, the perceptron algorithm successfully converges and finds a hyperplane that separates the two classes.

Now, let's explore a more complex problem. Consider a dataset with two classes: C_1 and C_2. Here, C_1 is a set of 1,000 points forming a circular pattern, while C_2 is another set of 1,000 points arranged in a half-moon shape. The radius of C_1 is denoted as R_m, while the inner and outer radii of C_2 are R_{in} and R_{out}, respectively. The ratio of R_m to R_{in} to R_{out} is 1:2:3. You can use various combinations such as 1, 2, 3 or 1, 3, 6, 9 to generate this data.

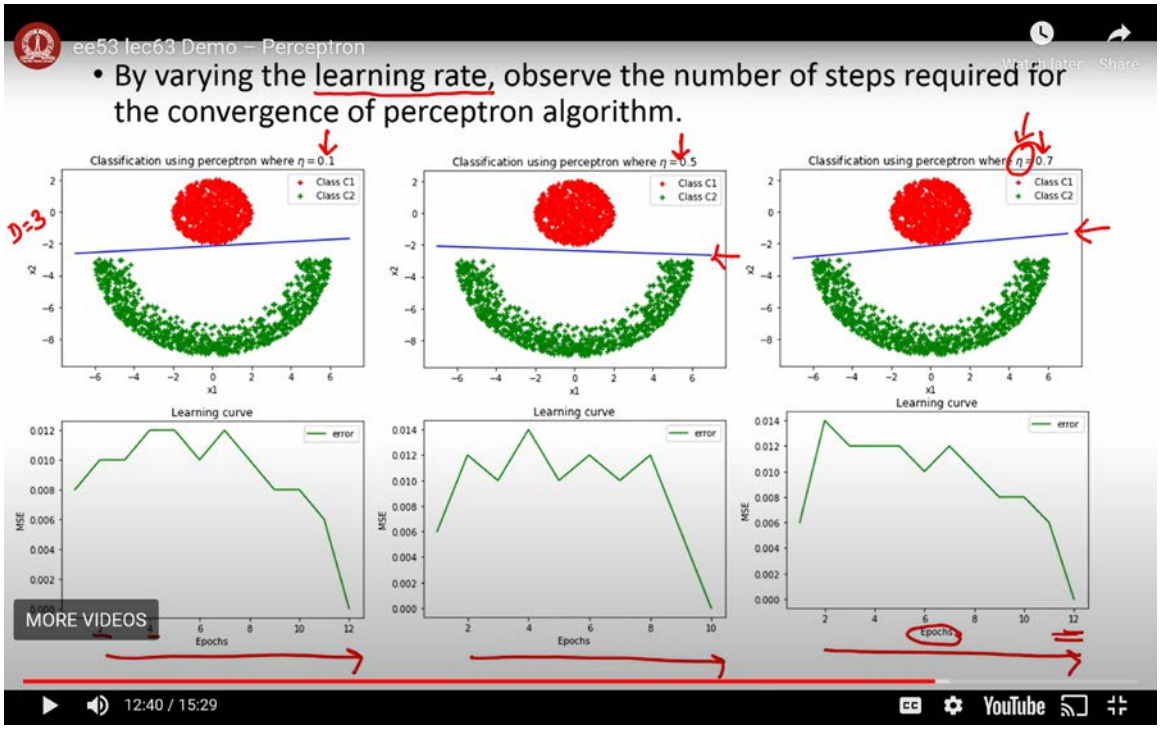
(Refer Slide Time: 10:43)

The parameter D controls the separation between the center of C_1 and the starting point of C_2. Let's examine how varying D affects the data's separability.

- When D = 0, meaning the circles overlap, the data is not linearly separable. Consequently, the perceptron algorithm cannot uniquely classify the data points into C_1 and C_2.
- Similarly, with D = 1, the data remains non-linearly separable, and thus, achieving 100 percent accuracy is not possible.
- For D = 2, the situation is the same; the data is still not linearly separable.

However, when D = 3, the data becomes linearly separable. This allows the perceptron algorithm to successfully identify a hyperplane that can classify the data points into C_1 and C_2 with 100 percent accuracy.

(Refer Slide Time: 12:40)

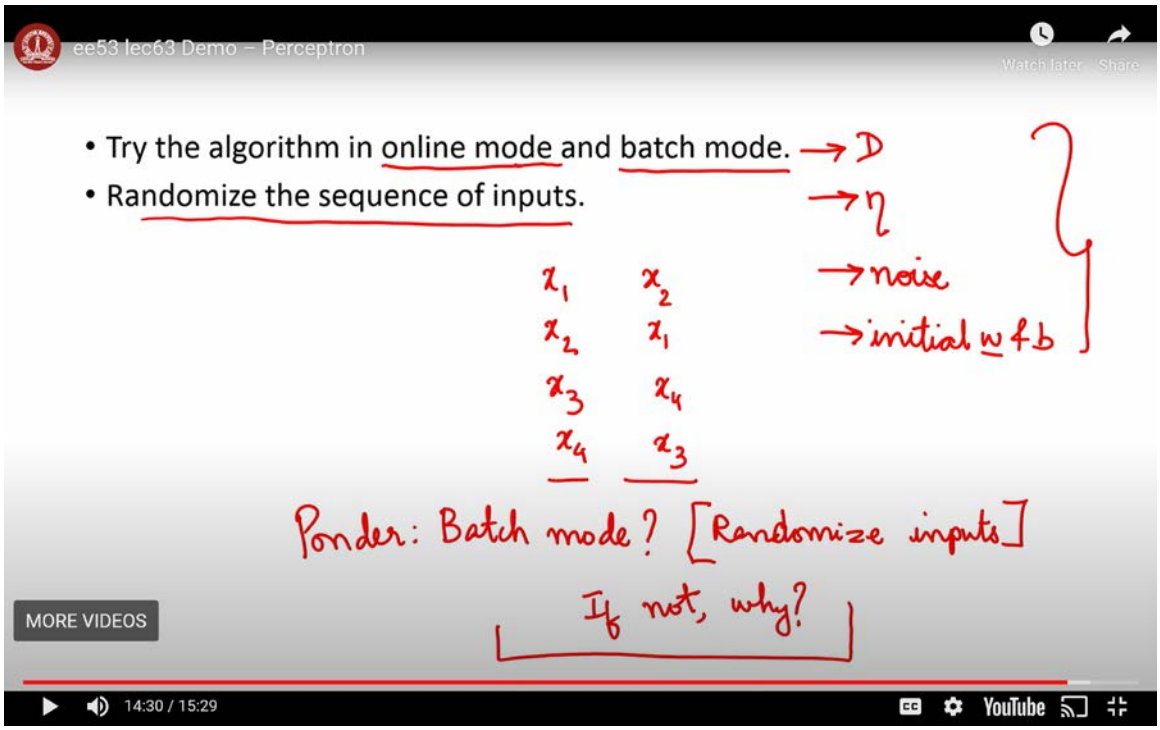


Having established that D = 3 ensures linear separability, let's perform additional experiments. With D fixed at 3, we will explore how different initial conditions for the weight vector affect the results. In our initial experiment, we have chosen specific weights and biases.

We know that the data is linearly separable, and thus, we can generate multiple hyperplanes. Given a specific weight vector and bias, the perceptron algorithm can successfully converge to a hyperplane that separates the two classes. In our examples, using different combinations of weights and biases, we consistently find a hyperplane. This demonstrates that the hyperplane obtained is not unique for a given dataset. You can experiment with various weight and bias combinations to see the different equations for the hyperplanes you can obtain.

Now, let’s introduce Gaussian noise to both classes. For each class, we add Gaussian noise with a mean of 0 and a variance ranging from 1 to 3. In the first case, with noise of mean 0 and variance 1, the data becomes highly noisy and no longer linearly separable. As a result, the perceptron algorithm fails to converge and cannot find a hyperplane that separates the two classes. Despite the data initially being linearly separable with D = 3, the addition of noise causes the data points to mix, disrupting separability.

(Refer Slide Time: 14:30)



Increasing the variance from 1 to 3 exacerbates this issue, causing even greater overlap between the classes. Consequently, the perceptron algorithm is unable to identify a hyperplane that can distinctly classify the data points into their respective classes.

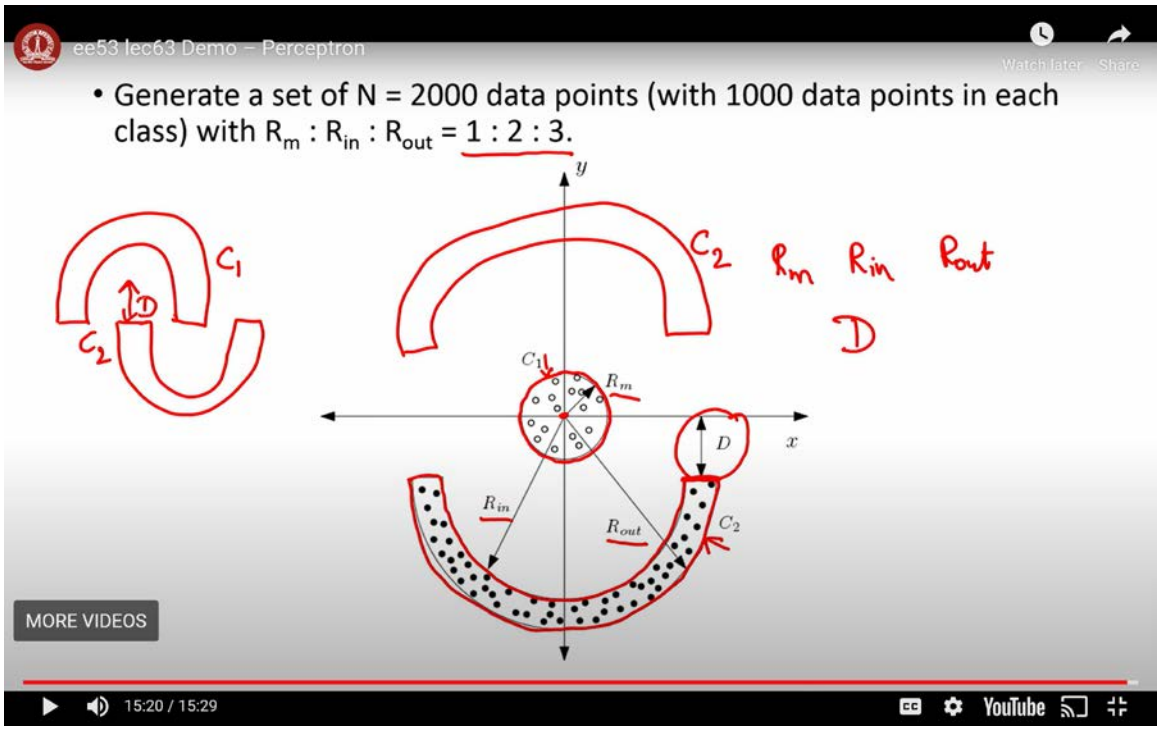
This highlights a limitation of the perceptron algorithm: it struggles to find a separating hyperplane in the presence of noise.

Next, let’s examine the impact of varying the learning rate on the algorithm's performance. For this experiment, we will first set the learning rate (η) to 0.1. Since the data is linearly separable with D = 3, the perceptron algorithm should converge. We observe that it takes

12 steps for the network to converge and find a hyperplane that separates the classes. Now, we will investigate how increasing the learning rate affects the convergence process.

Let's consider the impact of varying the learning rate η on the perceptron algorithm. When we increase η to 0.5, we observe that the perceptron algorithm can still find a hyperplane to classify the data points. In this case, it requires just 10 steps for convergence. This suggests that as the learning rate increases, the number of steps or epochs required for convergence might decrease.

(Refer Slide Time: 15:29)



Next, let’s increase η further to 0.7. Again, the perceptron algorithm successfully identifies a hyperplane that separates the data points into their respective classes. However, we notice that the number of steps required for convergence has increased to 12. Despite what seems like an inverse relationship between the learning rate and the number of steps, the correlation between η and the epochs required for convergence is not straightforward. The learning rate does not directly influence the number of steps for convergence, as confirmed

by the proof provided in the course. You are encouraged to review this proof and experiment with different values of η to observe the effects on convergence.

Additionally, you can perform various experiments with the same dataset. For example, you can run the algorithm in both online and batch modes to compare the number of steps required for convergence. Experiment with different learning rates, noise levels, and initial conditions for weights and biases. Try randomizing the sequence of inputs presented to the perceptron. For instance, if you initially provide the sequence x_1, x_2, x_3, x_4, try presenting them in a different order, such as x_2, x_1, x_4, x_3. Investigate whether the order of inputs affects the algorithm's performance, especially in batch mode.

Experimenting with these variations will deepen your understanding of the perceptron algorithm and help you determine which settings yield the best performance. You can also explore different values for the parameters R_m, R_{in}, and R_{out}, as well as varying the distance D between classes. Consider whether the data remains linearly separable under these different conditions or determine the minimum distance required for separability. Engaging in these experiments will enhance your familiarity with the perceptron algorithm. Thank you for your attention.