Neural Networks for Signal Processing - I

Prof. Shayan Srinivasa Garani

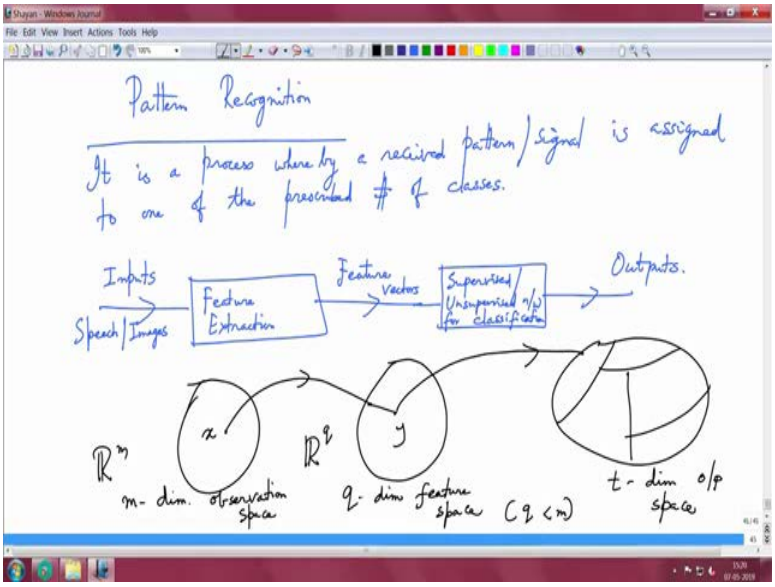
Department of Electronics Systems Engineering

Indian Institute of Science, Bengaluru

Lecture – 08

Perceptron 1

(Refer Slide Time: 00:29)



Next, we will discuss the topic pattern recognition. What exactly is pattern recognition? It is the process by which a received pattern, or signal, is assigned to one of a predetermined number of classes. Pattern recognition is a well-established field, extensively studied over the last 50 to 60 years in signal processing.

A classic example of a pattern recognition task involves recognizing spoken numbers. Imagine people are speaking isolated words like "one, two, three, four, five, six, seven, eight, nine, zero." The task is to form these into numerical sequences like "123" or "145." As these words are spoken, the challenge is whether a machine can accurately recognize that the spoken "one" corresponds to the numeral 1, "zero" to the numeral 0, and so on.

For instance, consider speaking through a phone and inputting an account number or other details. In this scenario, the machine should be able to automatically recognize and understand the numbers from spoken speech. How can we accomplish this? The machine must be capable

of recognizing these spoken numbers and then performing subsequent actions or controls based on that recognition.

This is an important application of AI, namely pattern recognition. So, how does one approach pattern recognition? First, we begin with inputs, which typically consist of signals such as speech, images, videos, and so forth. The initial step involves extracting features from these inputs, a phase known as feature extraction. This process yields feature vectors, which are then fed into a supervised or possibly unsupervised network for classification, ultimately producing outputs.

Delving deeper into this process, let's consider a signal x residing in an m-dimensional observation space, denoted as R^m. From this space, we transition to a feature space of dimension q, denoted as R^q, where typically q is less than m. Let's denote this transformation as y. From y, we proceed to an output space, possibly with t-dimensional outputs, represented as R^t.

So, at this stage, when transitioning from the observation space to a feature space, we undergo a form of dimensional reduction. Consider a simplistic example in speech recognition: we start with a band-limited speech signal of 4 kHz. After sampling this signal, we extract features such as Mel-frequency cepstral coefficients (MFCCs) or other feature sets, which reside in a lower-dimensional space compared to the input signal.

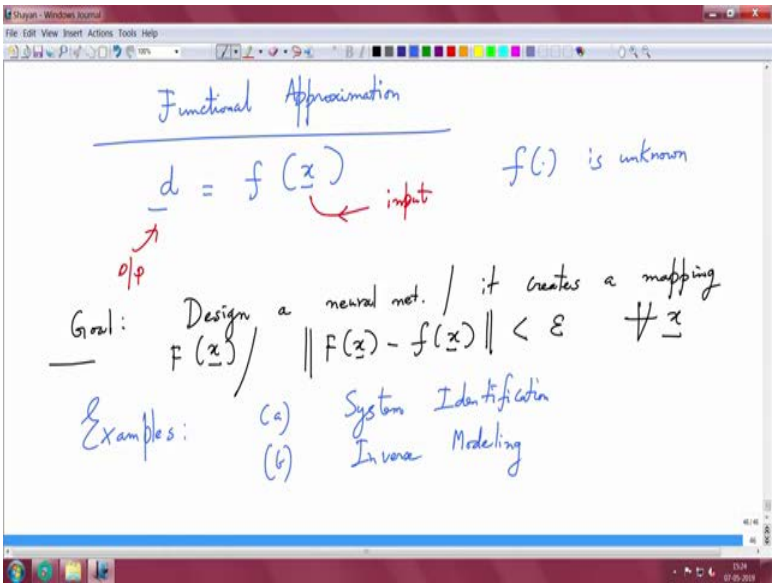
These feature vectors are then inputted into a neural network, which may operate in a supervised or unsupervised mode, I mean, our focus in this course being neural networks. The neural network processes these features, mapping them to output space, where it either associates them with a desired response in supervised learning or naturally clusters them in unsupervised learning.

In a supervised scenario, for instance, we associate the feature vector with a specific label, while in unsupervised learning, the network tessellates the feature vectors based on similarities. Once trained with test inputs, the network can effectively classify new inputs based on the patterns it has learned. This illustrates the foundational workings of a pattern recognition engine.

Let's delve into functional approximation. Suppose we have an input vector x and an output d, and there exists an unknown mapping f between them. When we input x into f, it generates d.

However, we don't know the exact form of f; we only have observed mappings like $f(x_1) = d_1, f(x_2) = d_2$,, and so forth.

(Refer Slide Time: 06:47)



The challenge here is to design a neural network that constructs a mapping, denoted as $\mathcal{F}(x)$, such that for all x, $\mathcal{F}(x)$ closely approximates f(x) within a specified tolerance ϵ under some norm measure. This problem of functional approximation is crucial because f could be non-linear, time-varying, or otherwise complex. The neural network, employing non-parametric methods, aims to approximate f with $\mathcal{F}(x)$.

Examples of such functional approximation problems include system identification and inverse modeling, where understanding the relationship between inputs and outputs can lead to effective modeling and prediction in various applications.

Certainly, inverse modeling can be seen as akin to system identification, as both involve discerning the nature of a system. However, we distinguish them into two categories, A and B. For instance, when dealing with a nonlinear plant and aiming to understand the system for control applications, we approach it as a system identification problem. Inverse modeling, on the other hand, comes into play when we consider scenarios like channel equalization, where achieving perfect inverse isn't feasible but using a neural network can approximate the desired outcome within an acceptable margin of error.

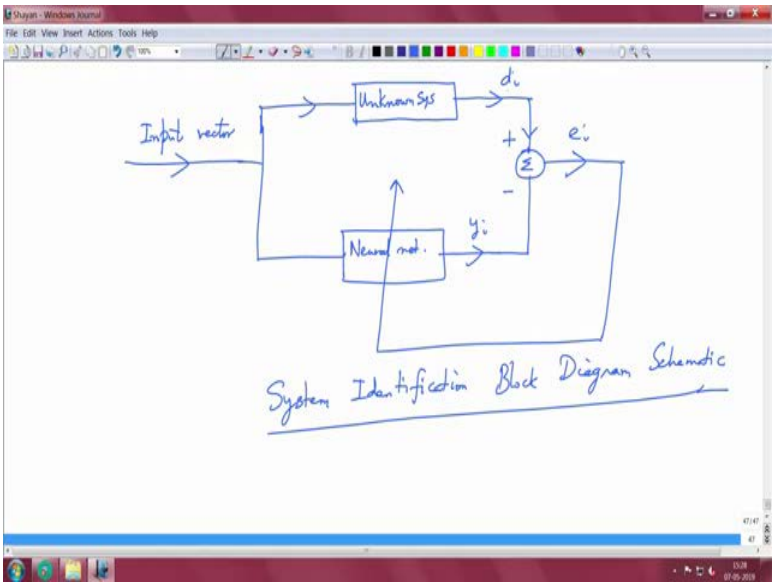
This is a crucial task. In applications such as communications, there are many facets to

consider. One crucial aspect in neural networks is that unlike in statistical signal processing techniques, there is no prior model assumption. Instead, we focus on parameters like biases and weights within the neural network framework. These parameters are learned directly from the data, without making any assumptions about their prior distribution. This makes neural networks inherently non-parametric.

In contrast, statistical signal processing techniques are parametric because they assume a specific form for the density function and estimate parameters like maximum likelihood or maximum a posteriori probability. This distinction between parametric and non-parametric approaches is fundamental.

Now, let's delve into the system identification problem and examine the fundamental components that constitute this process.

(Refer Slide Time: 11:24)



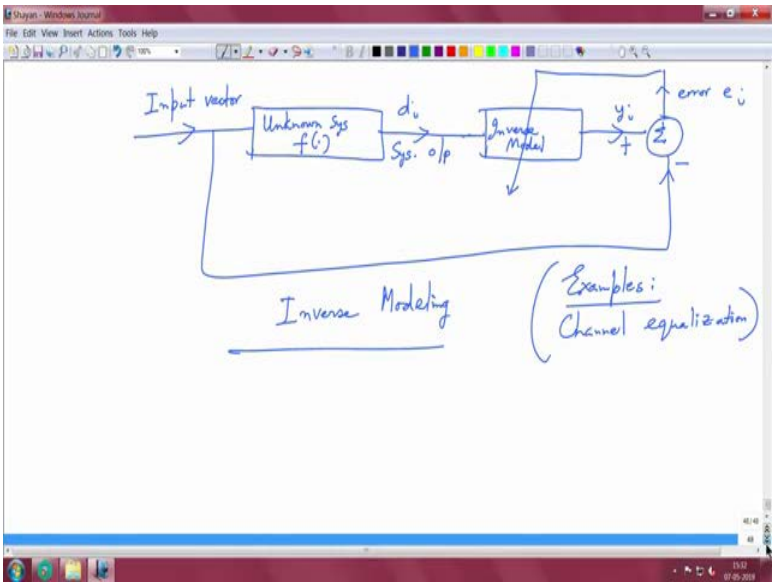
Let me outline the architecture and then explain its functioning.

Here is a schematic block diagram of a system identification task. We start with an input vector that is fed into an unknown non-linear system, which generates desired responses d_i. Simultaneously, this input vector is also fed into a neural network, which produces desired responses y_i.

We compute the difference between d_i and y_i, which gives us the error signal. This error signal, processed through an objective function, guides the adaptation of the network. If you're

knowledgeable about this, you might consider whether d_i and y_i need to be scalars or vectors. They can indeed be vectors, leading to an error vector. To handle this, we typically convert the error vector into a scalar using an appropriate norm measure, which then governs the network's adjustments under specific regulatory conditions. We'll delve deeper into these specifics as we carefully define the problem. This provides a qualitative understanding of how a system identification task is effectively executed. Similarly, we encounter inverse modeling.

(Refer Slide Time: 14:19)



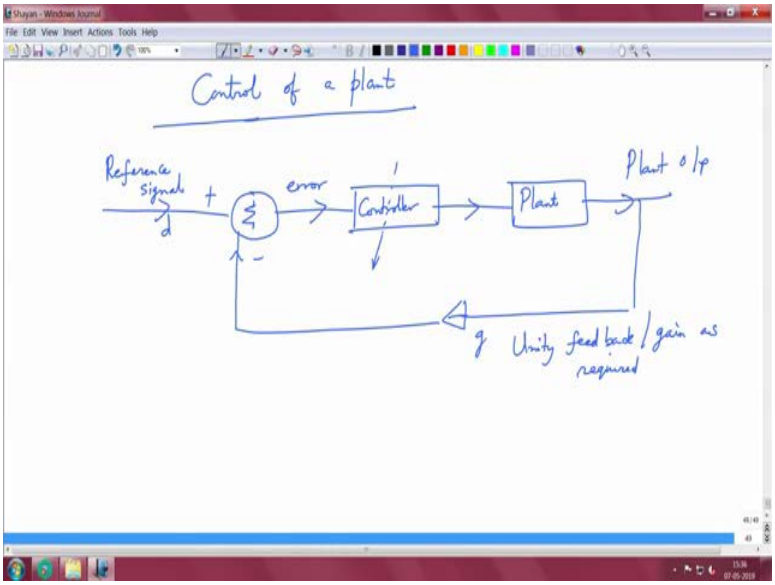
We start with an input vector that enters an unknown system, denoted as f. This system produces outputs d_i, which represent the system's response. Crucially, the input vector also feeds into a linear combiner where, on one path, it combines with y_i, and on the other, it stands alone. The resulting deviation e_i serves as the driving force for this model.

The goal here is to effectively model this unknown system. Ideally, y_i should closely approximate the input vector. This desire for y_i to mimic the input vector reflects our modeling objective. Essentially, the composition of mappings between the unknown system and the inverse model should ideally nullify each other, forming the core principle of inverse modeling.

Examples illustrating inverse modeling include applications like channel equalization within communication systems or scenarios involving echo cancellation. These applications underscore the practical importance and versatility of inverse modeling techniques in various engineering domains.

So, what happens with channel equalization? Imagine I have a signal that travels through an unknown channel. I don't know the characteristics of this channel, but I need to reverse its effects. The idea is to construct an inverse model that closely approximates a delta function alongside the channel composition. The key lies in understanding the specific input vectors and the desired outcomes, which are highly dependent on the particular case. This diagram offers a high-level view of the inverse model because our objective is to accurately replicate the original input signal at the output after passing through this model. Essentially, this encapsulates the challenge of inverse modeling.

(Refer Slide Time: 17:28)



Next, let's discuss control. I'll sketch out the block diagram again, similar to what we did earlier, and then delve into each block and its function. Imagine we want to regulate a plant. This involves taking certain inputs from a controller, which is influenced by the error signal. The reference signal serves as our target, but due to the plant and controller's inherent characteristics, the actual signal may differ slightly. Our goal, however, remains to regulate the plant effectively.

The error signal represents the deviation from the reference signal and guides the feedback loop controlled by the controller. We aim to regulate this transfer function through the controller, ensuring that the controlled signal sent to the plant, despite disturbances and other factors, aligns closely with our desired output. This is a general overview, and depending on the application, the controller could potentially be driven by a neural network.

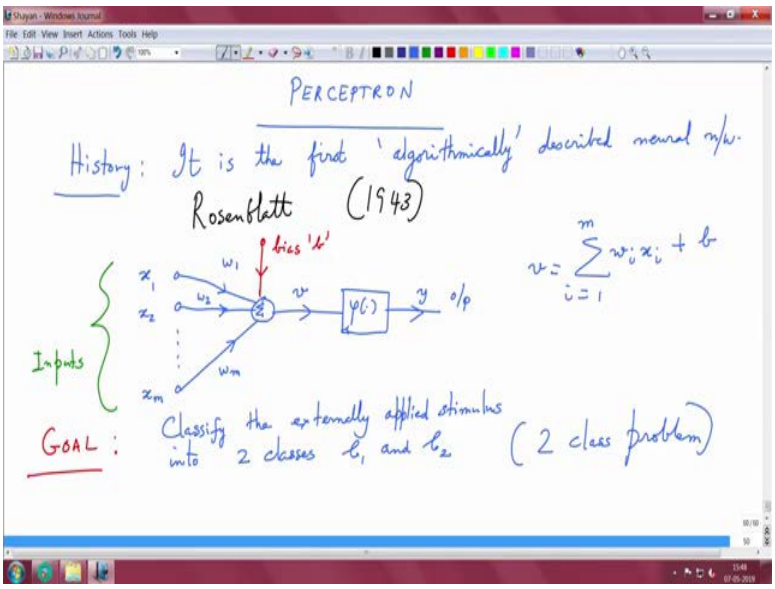
This controller is where adaptation comes into play, potentially utilizing a neural network-driven approach. This schematic illustrates the control of a plant, not botanical plants, but rather industrial facilities. For instance, consider the Haber process used in ammonia manufacturing, where precise regulation of gas concentration and flow is crucial.

The controller's function hinges on error feedback and adaptation mechanisms. I'll denote this arrow here to signify adaptation driven by the controller's error perception. In real-world applications, there are numerous instances where such control mechanisms are essential.

As we've observed, we encounter pattern association, pattern recognition, function approximation, and plant control, all of which are non-linear problems. Unlike traditional linear signal processing with LTI (Linear Time-Invariant) systems, where we deal with rational transfer functions to analyze poles and zeros, these non-linear systems lack such characteristics.

Instead, we're dealing with non-linear mappings that defy simple characterization by poles and zeros. We often resort to piecewise linear approximations to understand these mappings, enabling us to analyze them in terms of LTI models under appropriate approximations. Fundamentally, this realm constitutes a non-linear signal processing challenge, focusing on the learning and understanding of complex mappings.

(Refer Slide Time: 22:54)



So, this summarizes our overview of learning tasks and serves as an introduction to the objectives achievable through learning procedures and algorithms in this neural network

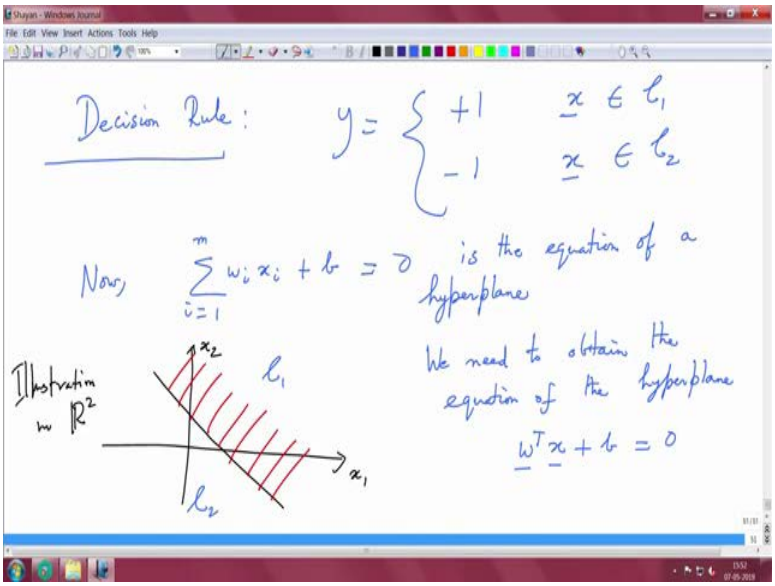
course. Moving forward, our next module will focus on the perceptron. It's the foundational algorithm you'll begin with, marking the start of our journey with the perceptron. Let's dive into the perceptron, which serves as the initial neural network model you'll explore in this course.

The history of the perceptron dates back to 1943 when it was first algorithmically described by Rosenblatt, hence sometimes referred to as Rosenblatt's perceptron. Now, let's revisit our earlier discussion on the linear combiner and activation function. We have inputs x_1, x_2, ..., x_m, connected to the linear combiner via weights w_1, w_2, ..., w_m, and a bias b. The local receptive field then passes through an activation function ϕ to produce the output y.

Recall that v is given by $v = \sum_{i=1}^{m} w_i x_i + b$.

Our objective here is to classify externally applied stimuli into two classes, namely c_1 and c_2. This is a binary classification problem where we aim to categorize incoming inputs into one of these two classes.

(Refer Slide Time: 26:20)



If we delve deeper, what we're seeking is a decision rule denoted by y, where y = 1 if vector x belongs to class c_1, and y = -1 if x belongs to class c_2. Essentially, there are only two possibilities: $y = 1 \; for \; x \in c_1$ and $y = -1 \; for \; x \in c_2$.

To establish this decision boundary, we set $\sum_{i=1}^{m} w_i x_i + b = 0$, which defines an equation of a hyperplane. To visualize, consider a simple x_1 x_2 plane where one side represents c_1 and the

other c_2.

In general, the equation $w^T x + b = 0$ describes this hyperplane. To ensure clarity, this illustration is in R^2, but the general form extends to higher dimensions. We can augment the bias b as an additional attribute, making w a vector of coordinates from 1 to m+1. Thus, $w^T x + b = 0$, encapsulates the decision rule, where b acts like a variable akin to w_i, albeit multiplied by 1.

All I have are the data points x, and I know they are linearly separable. This means I can establish a decision rule: y = +1 when x belongs to c_1 and y = -1 when x belongs to c_2. With this decision rule in mind, I aim to derive the equation of the hyperplane, which involves estimating and obtaining w.

This marks the beginning of our journey where we'll delve into the algorithm and discuss the proof of convergence in the upcoming module.