Neural Networks for Signal Processing-I

Prof. Shayan Srinivasa Garani

Department of Electronic System Engineering

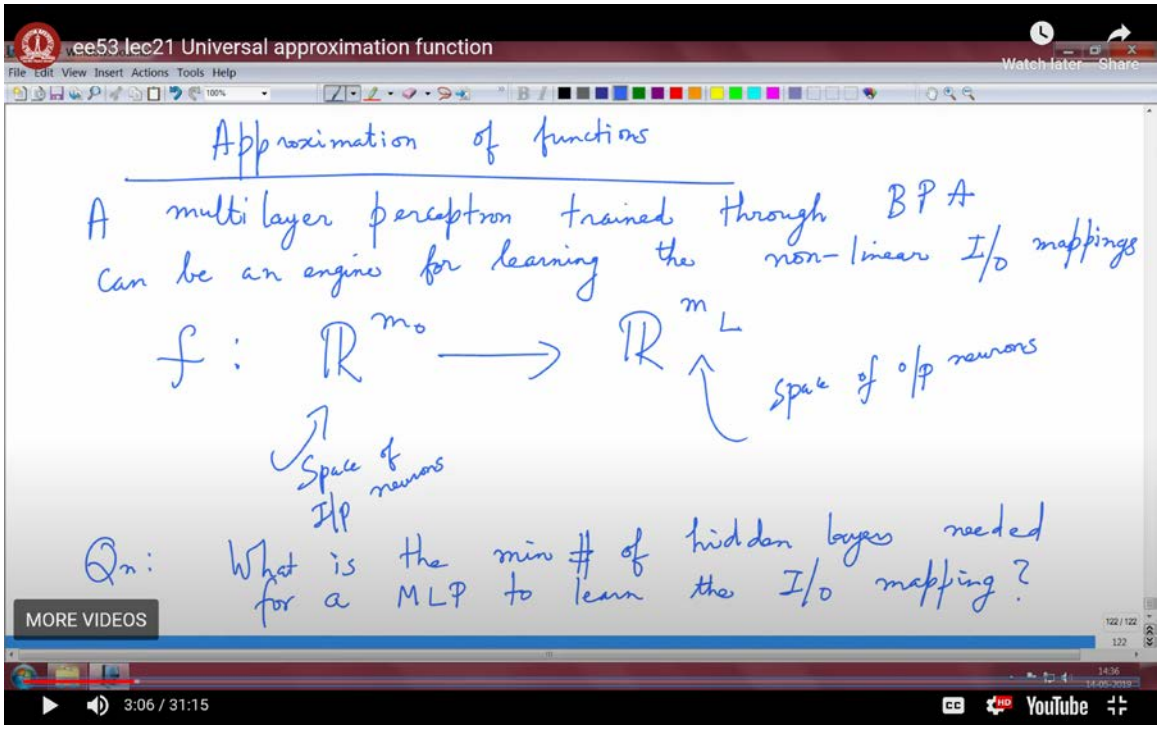
Indian Institute of Science, Bengaluru

Lecture – 21

Universal Approximation Function

Let's dive into the topic of function approximation. A multilayer perceptron, trained using the backpropagation algorithm, is a powerful tool for learning input-output mappings, particularly when dealing with non-linear mappings. While linear mappings can be represented with an affine transformation and handled by simpler perceptrons, our focus here is on non-linear input-output mappings. This means that if we have inputs in R^{m_0}, we can think of mapping these inputs to R^{m_l}, where m_l represents the space of output neurons. Essentially, this involves mapping between two different Euclidean spaces.

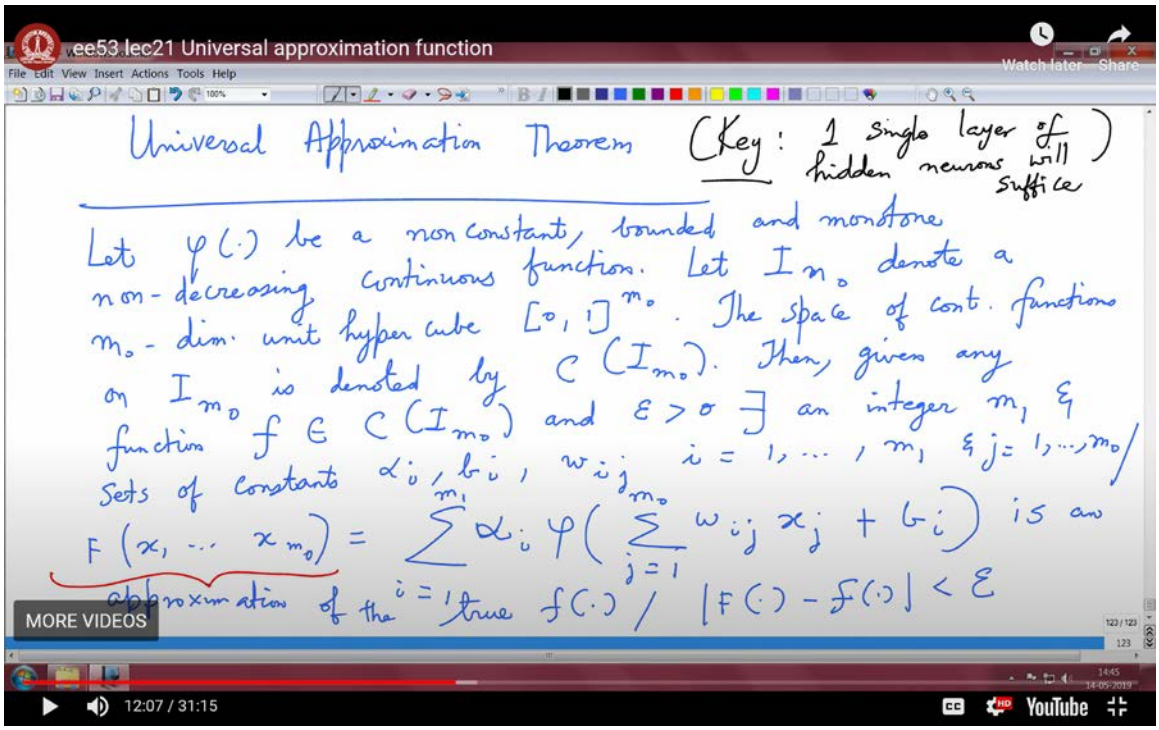
(Refer Slide Time: 03:06)



A critical question that arises algorithmically is: what is the minimum number of hidden layers required for a multilayer perceptron to effectively learn a given input-output relationship? In a machine learning or supervised learning context, where we provide inputs and their corresponding desired outputs, we need to determine the necessary architecture to learn these mappings.

Fortunately, there is an answer to this question, and it is provided by the Universal Approximation Theorem. Allow me to state this theorem, and I will also provide references for you to explore the proof in more detail. We may cover this proof in a future module if time permits.

(Refer Slide Time: 12:07)



The Universal Approximation Theorem is stated as follows:

Let φ be a non-constant, bounded, monotone non-decreasing continuous function. Let I_{m_0} denote an m_0-dimensional unit hypercube. The space of continuous functions on I_{m_0} is denoted by $C(I_{m_0})$, representing all continuous functions over this unit hypercube. Given any function f that belongs to this space, there exists a small positive ϵ and an integer m_1,

along with sets of constants α_i, b_i, and W_{ij} for i = 1, …, m_1 and j = 1, …, m_0, such that the function described by these variables can approximate f within an ϵ margin. In other words, the absolute difference between this approximating function and the true function is within ϵ.

This result essentially guarantees that with the appropriate number of hidden layers and neurons, a multilayer perceptron can approximate any continuous function on a given unit hypercube to any desired degree of accuracy.

Let's delve into the details of function approximation using a multilayer perceptron (MLP). If you observe the form of the approximating function provided, you will see that it involves some scalar parameters α_i, a non-linear function φ, and weights that scale the inputs. Additionally, there's a bias term added before the non-linear function is applied. This setup represents a local receptive field in which each of the M_1 non-linear functions contributes to the approximation of the true function.

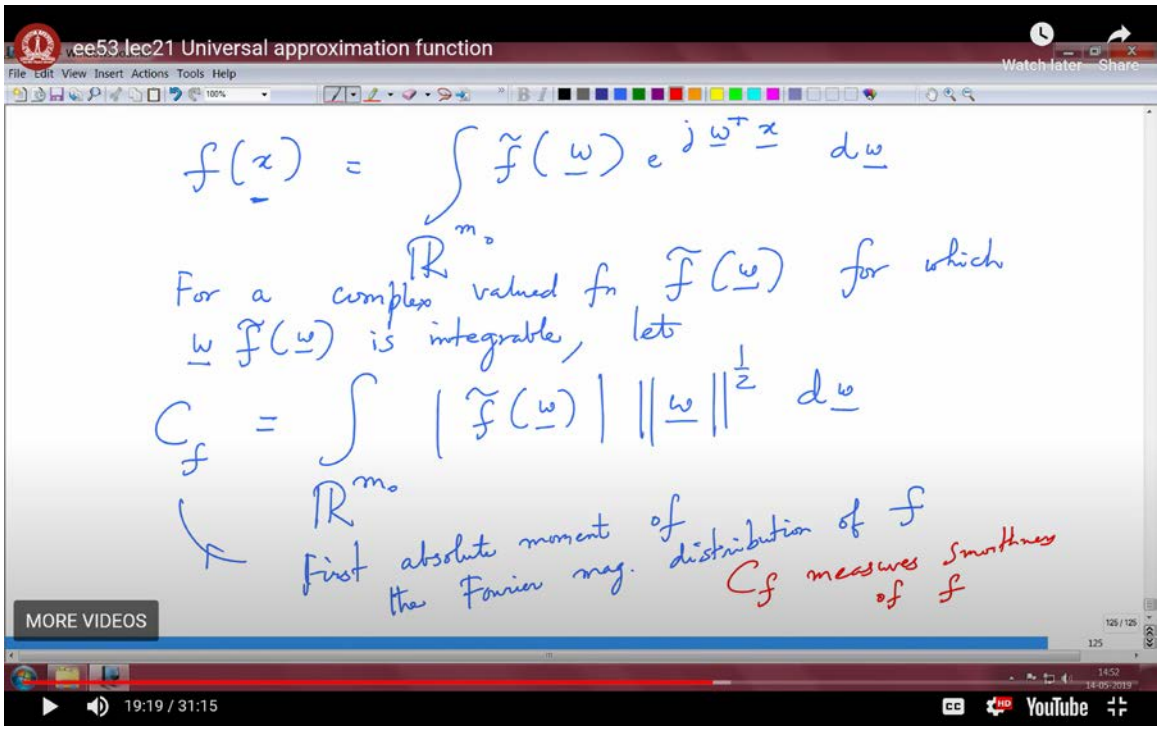
To approximate a continuous function, it’s crucial to note that one single layer of hidden neurons is sufficient. The universal approximation theorem asserts that this single hidden layer can approximate any continuous function in an m_0-dimensional space. This is a key insight: you don’t need multiple layers; a single hidden layer will do the job. This simplifies the learning process, focusing on a single layer of neurons to achieve the desired function approximation. However, the theorem doesn’t address the optimality of a single layer in terms of learning time, ease of implementation, or generalization, especially when the input process has memory or other complexities.

The theorem merely states that having M_1 neurons in the hidden layer is sufficient but does not specify how the number of neurons should relate to the density of data points or other factors. The essence of the theorem is that with a single hidden layer, you can approximate any continuous function.

The proof details of this theorem are elaborated in specialized papers, which will be provided for further reading. I may also cover the proof later in the course if time permits.

Now, let's turn to the topic of approximation error bounds. Barron, in 1993, explored the approximation properties of multilayer perceptrons using sigmoid functions and a single hidden layer. He demonstrated that when given a set of data points sampled from a function, the network provides an approximating function. For a function f representing the data points (the true function), the network produces an approximating function, which we can denote as f.

(Refer Slide Time: 19:19)



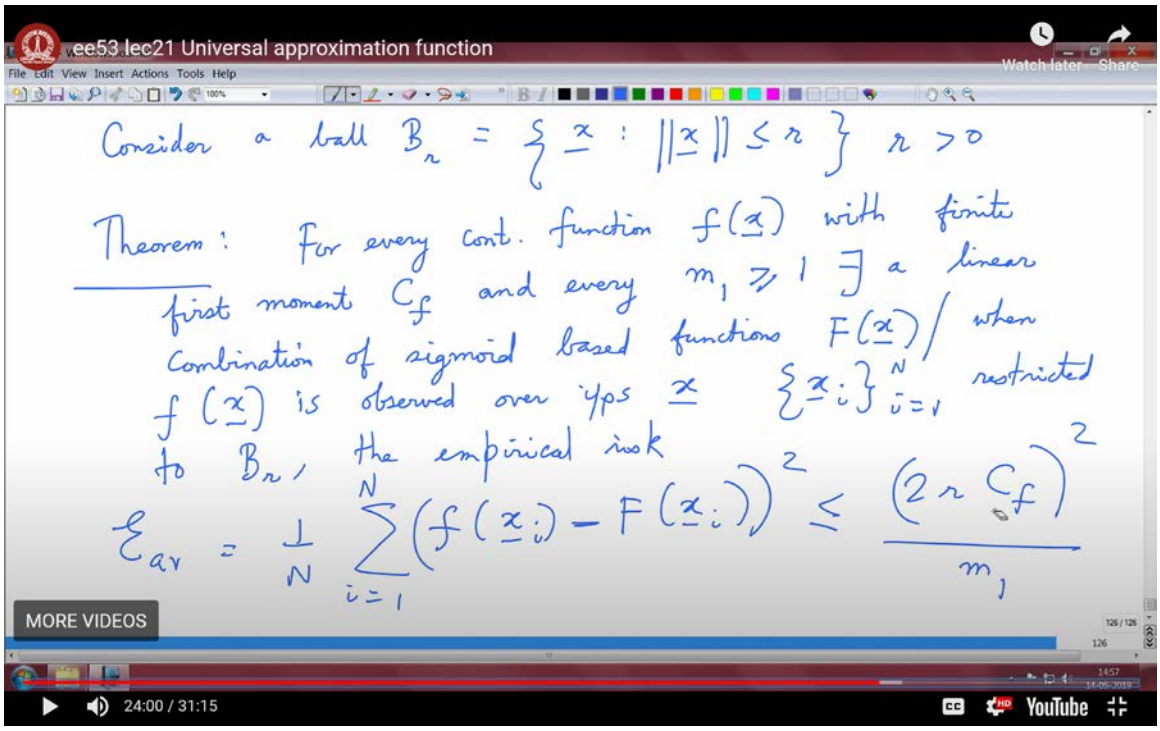
When the network encounters test data, the approximating function F serves as an estimator for the target function values at new points. Thus, $f \approx \hat{f}$ represents the approximation of the true function.

To frame this in the context of Fourier transforms, let $\tilde{f}(w)$, where w is a vector, denote the multidimensional Fourier transform of f(x), with x being a data vector in R^{m_0}. Here, w represents the frequency vector (in a 2D plane, for instance, it would be ω_1 and ω_2 or w_1 and w_2). The function f(x) can be recovered using the inverse Fourier transform, provided that $w \cdot \tilde{f}(w)$ is integrable.

Let us define c_f as the first absolute moment of the Fourier magnitude distribution of f, given by the specified equation.

This parameter c_f represents the first absolute moment of the Fourier magnitude distribution of the function f, and it essentially measures the smoothness of f. In other words, c_f quantifies how smooth the function f is.

(Refer Slide Time: 24:00)



Barron's theorem provides a bound on the empirical risk based on this parameter c_f. To delve into the details:

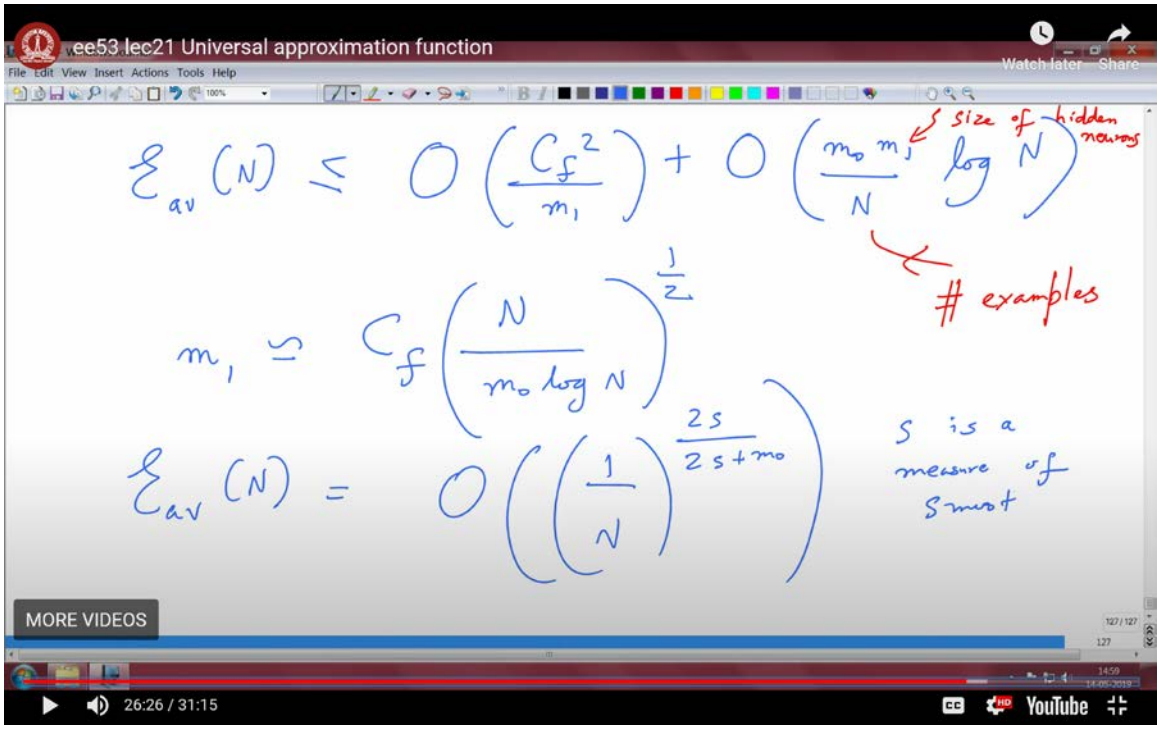
Consider a ball B_r, defined as the set of all points x such that the norm of x is within a radius r, where r is positive. Barron's theorem states the following: For any continuous function f(x), which acts on the data vector and has a finite first moment c_f, and for every $M_1 \geq 1$ (indicating at least one unit in the hidden layer), there exists a linear combination of sigmoid-based functions that can approximate f(x). This means that if f(x) is observed over a set of inputs x, specifically within the ball B_r, the empirical risk is given by:

$$\text{Empirical Risk} \leq \frac{2rc_f^2}{M_1}$$

Here, the empirical risk is defined as the squared error between the true function f(x) evaluated at data points and the approximating function F(x). This result is significant because it provides a bound on the empirical risk based on the parameter c_f, the number of hidden neurons M_1, and the radius r.

In practical terms, if we have a set of data points and know the values of M_1 and c_f (which can be estimated from the Fourier transform of the true function), we can compute the empirical risk bound. This bound helps determine the number of hidden neurons needed, making this theorem particularly useful.

(Refer Slide Time: 26:26)



Barron further simplified this bound in his paper. The quantity is of the order:

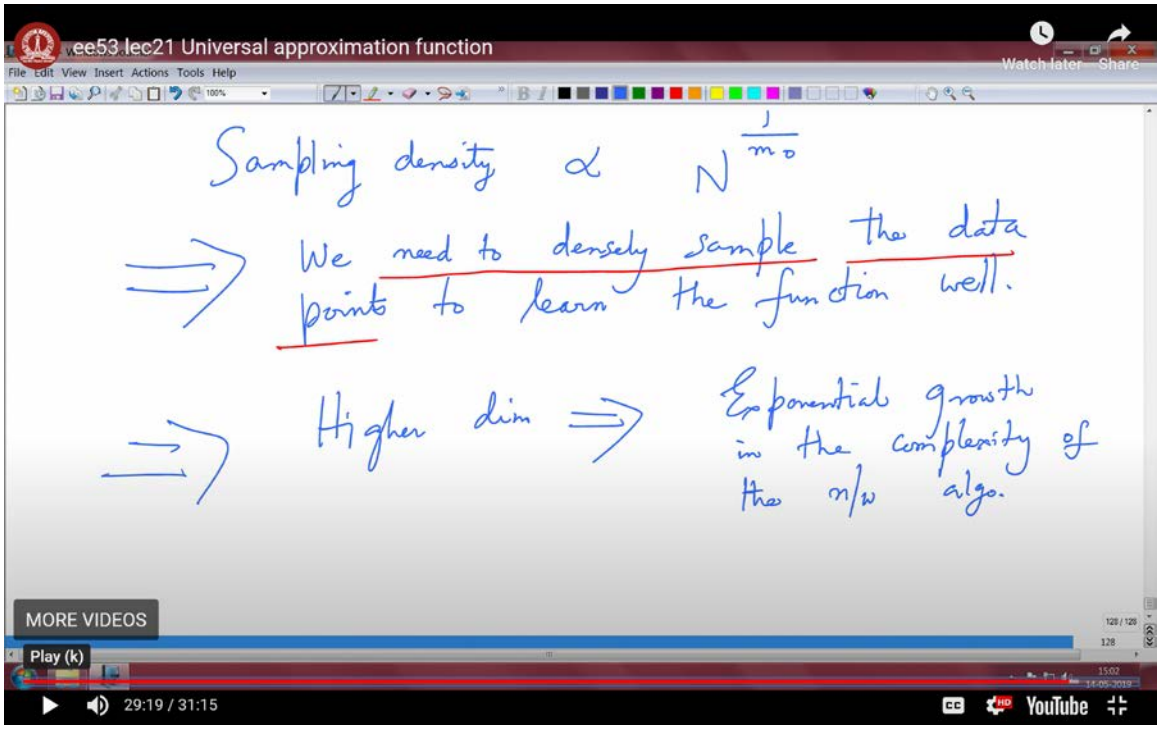
$$\frac{c_f^2}{M_1} + \frac{M_0 M_1}{N} \log N$$

where N is the number of examples or data points, M_1 is the number of hidden neurons, and M_0 is the dimension of the input space. Simplifying this, Barron showed that the number of hidden neurons M_1 is approximately:

$$M_1 \approx \frac{c_f \cdot N}{M_0 \log N}$$

This indicates that there is a dependency between the error rate and the input dimension. In essence, the average error can be simplified to show how it relates to c_f and M_0, providing a measure of smoothness and input dimension impact.

(Refer Slide Time: 29:19)



The sampling density required to effectively learn a function is proportional to the m_0-th root of the number of data samples. In simpler terms, this means that as the number of data points increases, the density of sampling needs to increase as well. Specifically, in a higher-dimensional space, we need to densely sample the data points to accurately capture the function's behavior. However, this necessity for dense sampling comes with a significant

drawback: the complexity of the network or algorithm grows exponentially with higher dimensions.

This exponential increase in complexity when dealing with higher dimensions is known as the "curse of dimensionality." Despite this challenge, higher dimensions can be beneficial as they offer greater generalization capabilities and improved pattern separation. In essence, higher dimensions help us better distinguish between different patterns. However, the need for dense sampling in higher dimensions also leads to an exponential rise in complexity, creating a trade-off between sampling density and computational feasibility.

We will revisit this issue when discussing Cower's theorem, which will provide further insights into how higher dimensions impact function learning, input-output mapping, and generalization abilities for pattern separation and classification problems.

So the core ideas related to the backpropagation algorithm: a single layer of hidden neurons can provide effective generalization for input-output mappings. This is a key takeaway, but in practical scenarios, multiple layers are often used. This is because the optimal number of nodes in the hidden layer for achieving a good approximation is not straightforward to determine. Therefore, a cascade of hidden layers is employed to approximate the function more effectively.

We will stop here and will delve into additional technical aspects of the algorithm, such as the Jacobian and Hessian, in the upcoming discussion.