Neural Networks for Signal Processing-I

Prof. Shayan Srinivasa Garani

Department of Electronic System Engineering

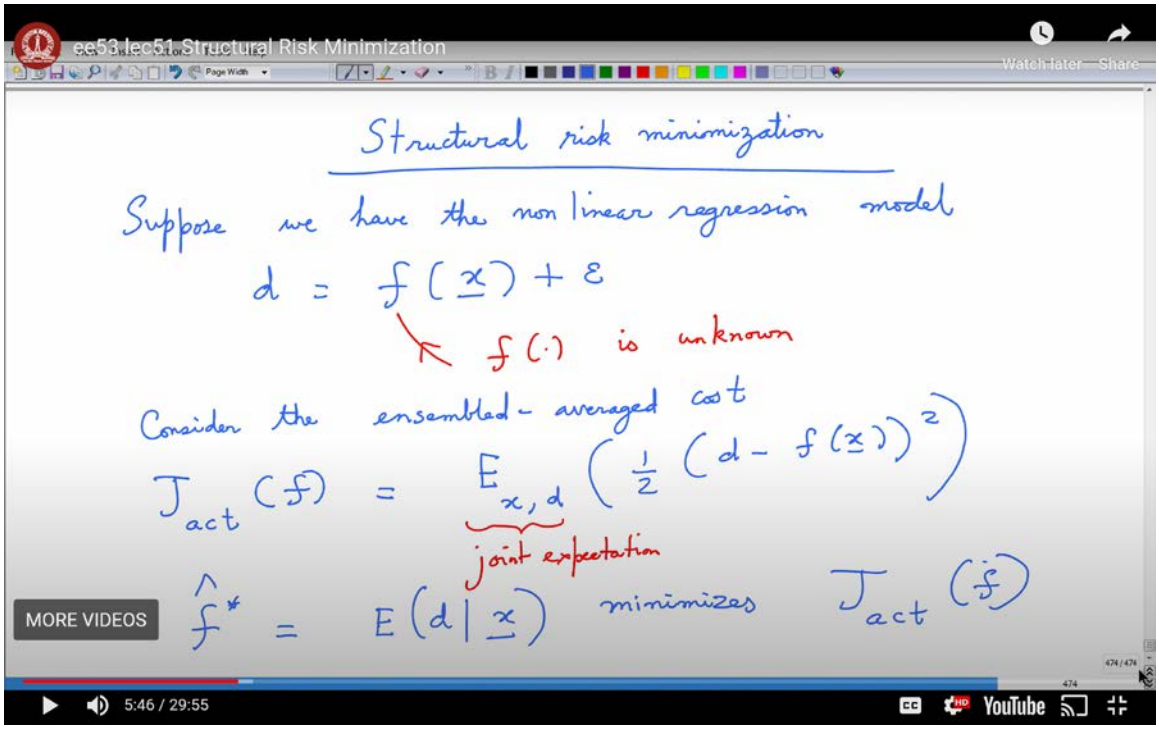
Indian Institute of Science, Bengaluru

Lecture – 51

Structural Risk Minimization

Let's delve into the concept of structural risk minimization. We start with a set of samples $\{x_i, d_i\}$, where x_i represents the data points and d_i is the corresponding desired response. Our goal is to determine a mapping such that $d_i = f(x_i)$, where f is an unknown, potentially non-linear mapping that we need to uncover.

(Refer Slide Time: 05:46)

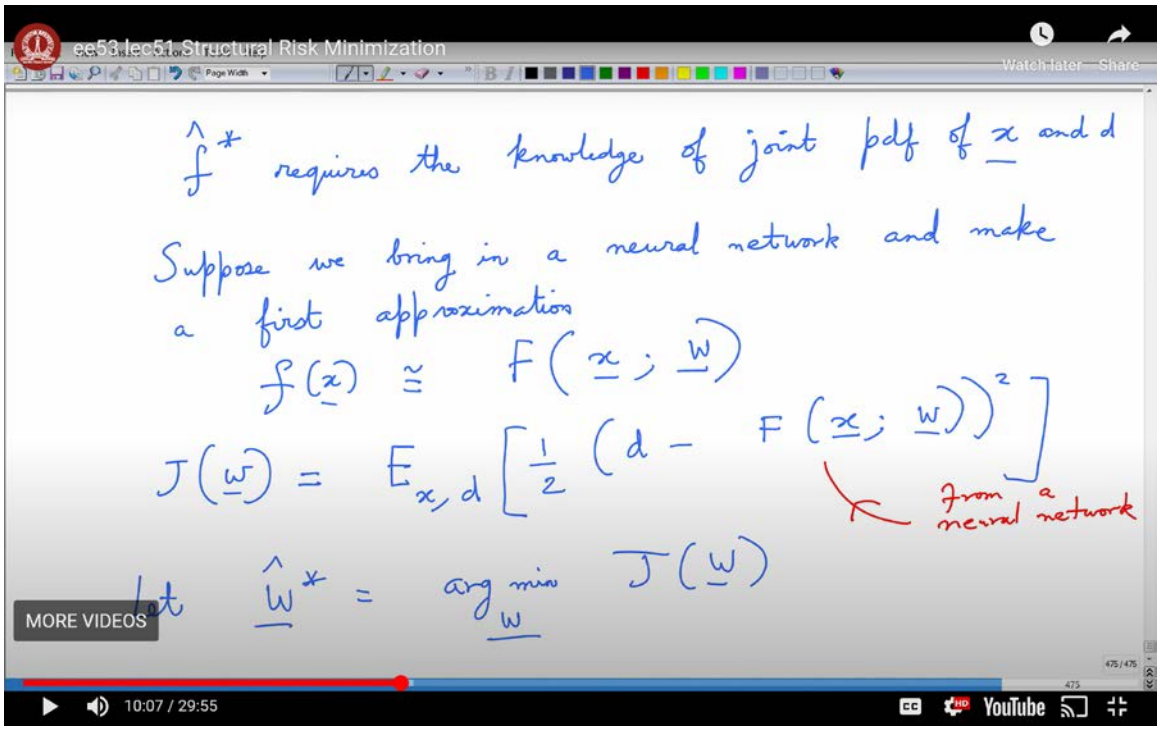


The challenge lies in figuring out this non-linear relationship, and as we've seen, neural networks offer a robust approach to achieve this. Various techniques, including

backpropagation algorithms, radial basis functions (RBFs), and Support Vector Machines (SVMs), can be employed to approximate these non-linear mappings effectively.

Once we've determined this mapping, we need to consider several factors: How does the error depend on the network's parameters? What is the approximation error? Is there any error in the estimate? And how is this error influenced by the number of data points and the network's complexity? These are critical considerations when solving such problems.

(Refer Slide Time: 10:07)



To explore these questions, let’s systematically work through the concept of structural risk minimization.

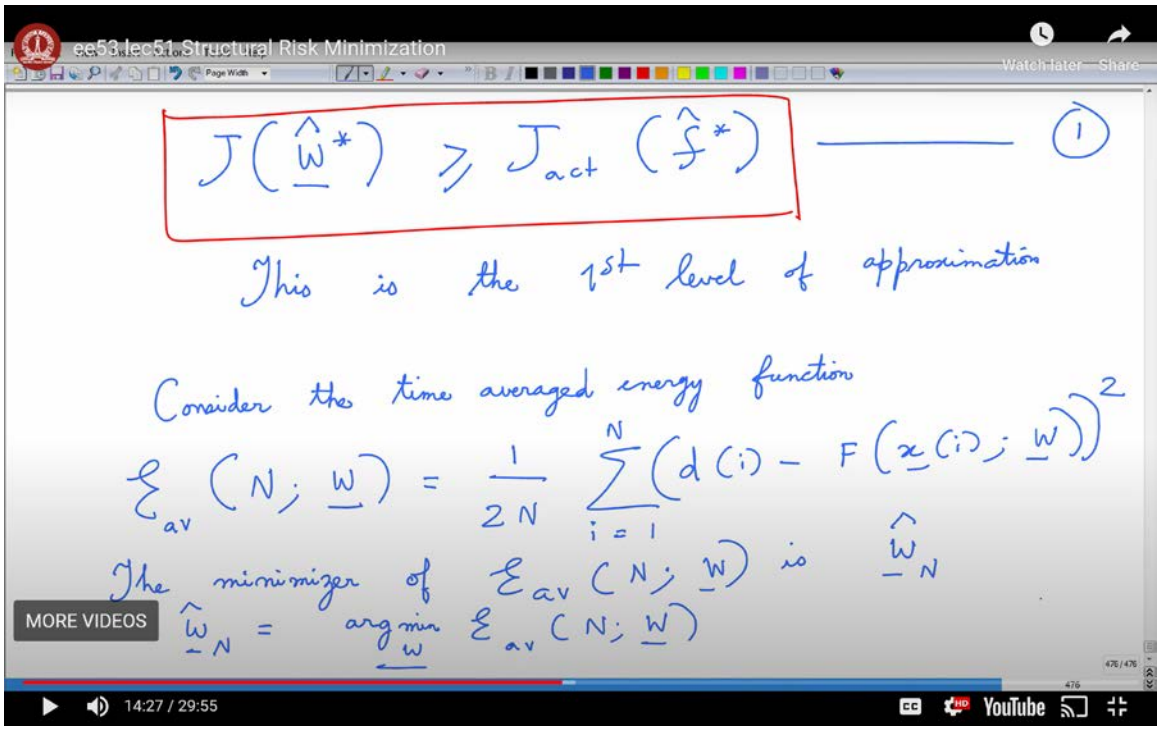
We begin with a non-linear regression model. Suppose we have such a model, as we discussed in a previous lecture, where $d = f(x) + \epsilon$. Here, f is the unknown mapping, x is the data point, d is the desired response, and ϵ represents the noise embedded in the response.

Now, let's consider the ensemble average cost, denoted as $J_{\text{actual}}(f)$. This cost is defined as the expectation over all pairs (x, d), and is mathematically expressed as:

$$J_{\text{actual}}(f) = E_{x,d}\left[\frac{1}{2}\left(d - f(x)\right)^2\right]$$

In this equation, the averaging is performed over the joint expectation of x and d, meaning we are calculating the mean squared error across all possible data pairs. Since the observations inherently contain noise, and we lack a precise statistical model for this noise, we proceed with averaging over the joint distribution of x and d.

(Refer Slide Time: 14:27)



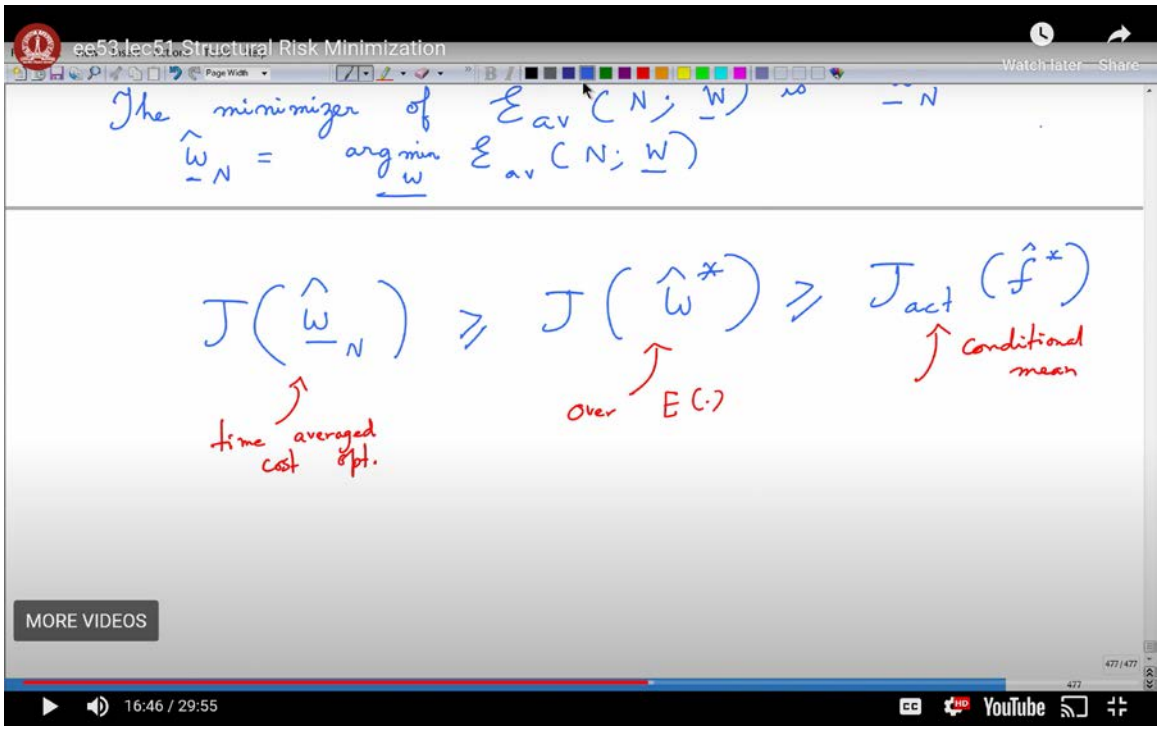
To compute the optimal estimate that minimizes $J_{\text{actual}}(f)$, we draw upon principles from basic statistical signal processing. The optimal function f^* that minimizes the cost function is the conditional expectation of D given X:

$$f^*(x) = E[D|X = x]$$

This expression represents the conditional mean of the desired response given the input. The intuition behind this is straightforward: if we're given a data point x, the best estimate for the corresponding response d is its conditional mean, which effectively minimizes the average error.

However, to determine this optimal function f^*, we need knowledge of the joint probability density function (PDF) of X and D. Unfortunately, in most practical scenarios, this joint density is unknown, and we typically do not have a parametric form for these PDFs.

(Refer Slide Time: 16:46)

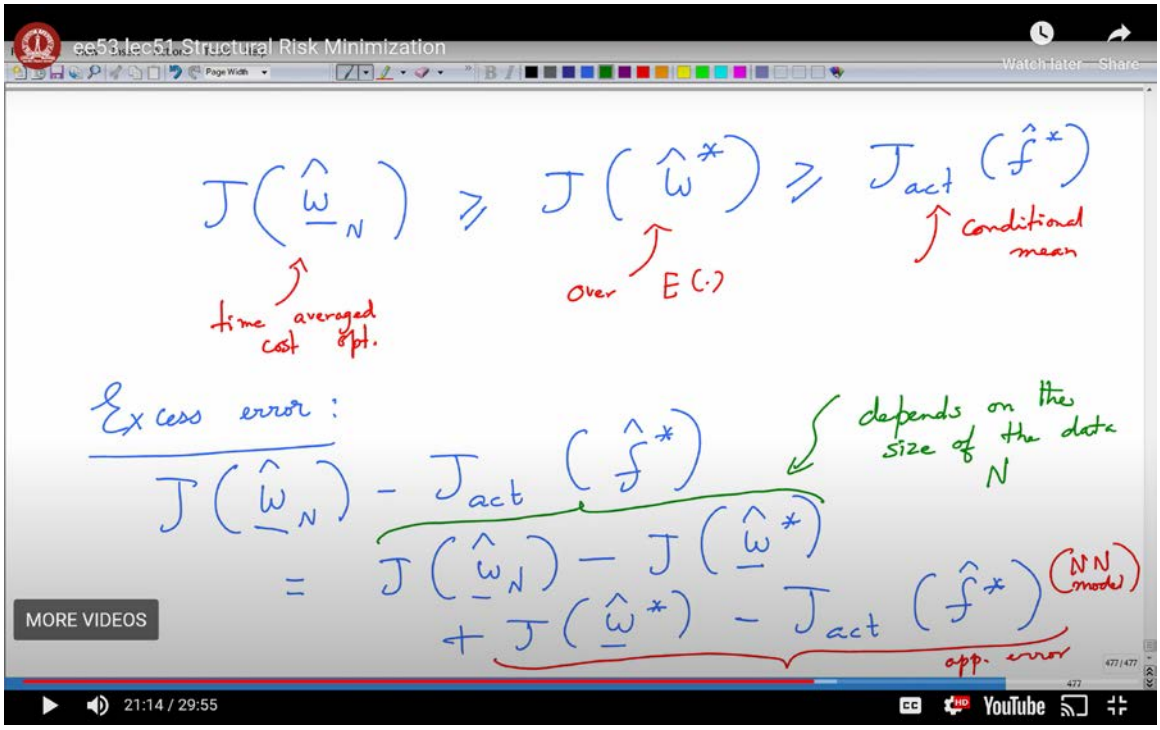


Given these constraints, we are compelled to make approximations and simplify the problem. This simplification allows us to proceed with an approach that approximates the optimal function f^*, even in the absence of detailed statistical knowledge about the underlying distributions.

Let's start by introducing the concept of using a neural network for our approximation. Suppose we bring in a neural network to make our first approximation. The function f(x), which is currently unknown, can be approximated by a neural network parameterized by a

vector $\boldsymbol{w}$. This network could be a multilayer perceptron utilizing a backpropagation algorithm, a recurrent network, or any other neural network architecture that fits the task at hand. The reason we approximate f(x) is because we need to learn the input-output mapping non-linearly, and a neural network is well-suited for this purpose.

(Refer Slide Time: 21:14)

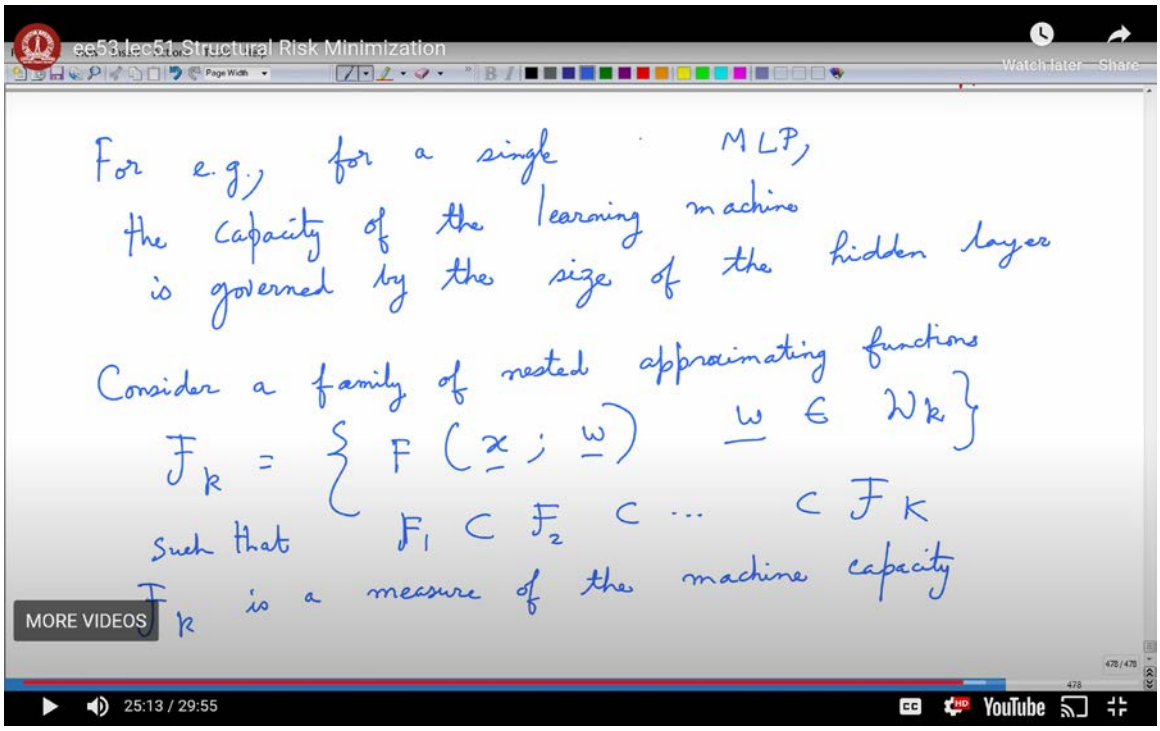


Let the function that performs this mapping be denoted as F(x, $\boldsymbol{w}$), where x represents the data points and *w* corresponds to the parameters (weights) of the neural network. Although this function is not a parametric form of the regression function in the traditional sense, the weights of the network are inherently embedded within this function.

Now, we can express the cost function J($\boldsymbol{w}$) as the expectation, specifically, the joint expectation over the data points. This expectation, however, replaces the unknown function f(x) with our approximated function F(x, $\boldsymbol{w}$), since we are using a neural network to achieve this approximation. The task now is to find the optimal parameters $\boldsymbol{w}$ that minimize this cost function.

Let $\boldsymbol{w}^*$ denote the optimal set of parameters, which we can estimate as the argument that minimizes J($\boldsymbol{w}$) over all possible choices of the weight vector. In other words, $\boldsymbol{w}^*$ represents the optimal weight vector that minimizes the cost function.

(Refer Slide Time: 25:13)



It's important to note that, because we're making an approximation using a neural network, even if we optimize the parameters with respect to this cost function, the resulting value J($\boldsymbol{w}^*$) will still be greater than or equal to the error obtained by estimating the function through the conditional mean. This is a fundamental inequality that reflects the limitation of our approximation.

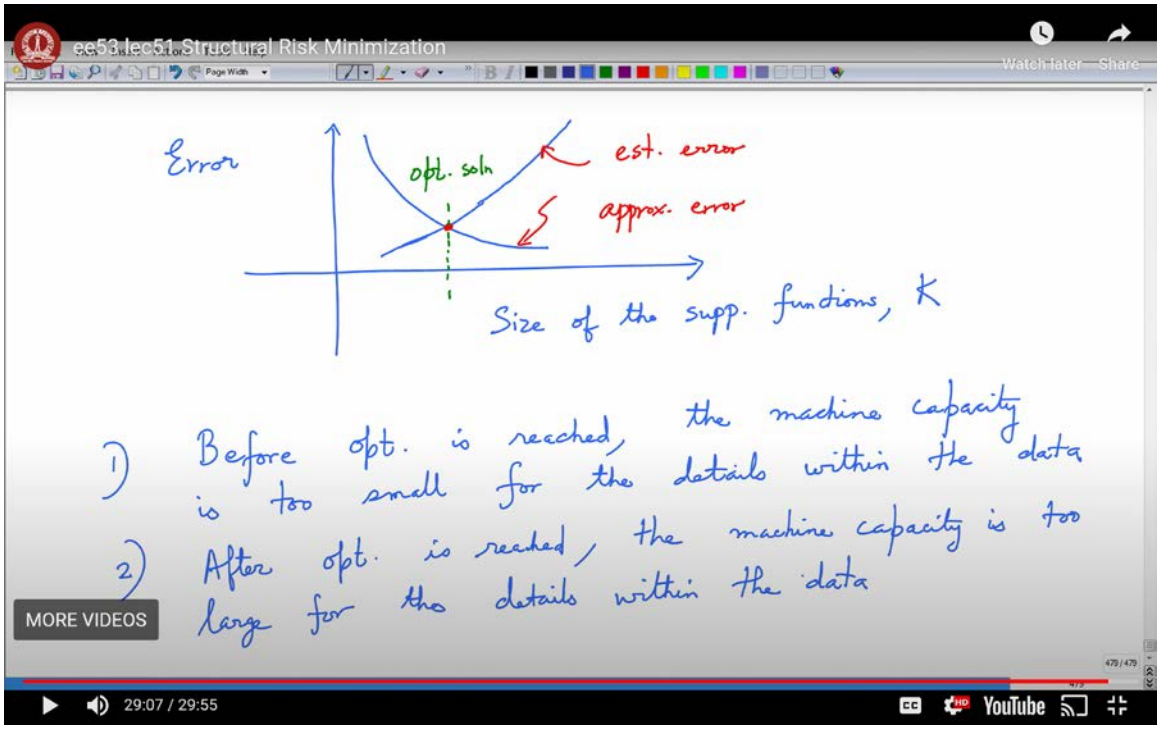
Next, let's consider this in more detail. The cost function J($\boldsymbol{w}$) still involves an expectation that needs to be computed over all possible pairs (x, d) from the joint probability density function (PDF). However, computing this expectation directly may be challenging because it would require an infinite number of samples.

To address this, we can make a further approximation by considering the time-averaged energy function instead. Specifically, we look at the time-averaged energy function E_{avg}, which is computed over n data points. This function is given by:

$$E_{\text{avg}}(w) = \frac{1}{2n}\sum_{i=1}^{n}\left(d_i - F(x_i, w)\right)^2$$

The optimal parameters $\hat{\boldsymbol{w}}_n$ that minimize this time-averaged energy function are found by minimizing the sum of the squared errors over all examples. This approach assumes a uniform distribution over the squared error, effectively replacing the expectation with an empirical average.

(Refer Slide Time: 29:07)



Clearly, when evaluating the cost using this optimal solution $\hat{\boldsymbol{w}}_n$, we expect the resulting error to be greater than or equal to J($\boldsymbol{w}^*$), especially if n is large. This inequality reflects the inherent trade-off in approximating the expectation with a finite sample average.

Thus, by making these approximations, first with a neural network and then with a time-averaged energy function, we arrive at a practical approach for minimizing the error in our non-linear mapping, acknowledging the limitations imposed by the approximations.

The relationship between the cost functions can be expressed as $J(\widehat{W_n}) \geq J(\widehat{W^*}) \geq J(f^*)$, where $J(\widehat{W_n})$ represents the cost function evaluated at the parameter set $\widehat{W_n}$, $J(\widehat{W^*})$ denotes the cost function evaluated at the optimal parameter set $\widehat{W^*}$, and $J(f^*)$ is the actual cost evaluated at the true function f^*. Here, f^* is the conditional mean function.

This relationship stems from the conditional mean, which involves an expectation and is calculated through time averaging. Essentially, if we had an infinite number of realizations, we would be able to determine the optimal parameter set $\widehat{W^*}$ by performing the expectation over the joint probability density function (PDF). With the joint PDF known, we could compute the optimal solution directly. However, in practice, since the joint PDF is unknown, we resort to time-averaged cost calculations.

In this case, the estimate obtained from the time-averaged cost function will always be greater than or equal to the cost function evaluated using the joint PDF with a neural network. This, in turn, will be greater than or equal to the actual cost evaluated at the conditional mean function f^*. This conceptual understanding highlights the hierarchy of approximations and the resulting cost evaluations.

Practically, the time-averaged cost function is more relevant since the joint PDF is not known. We begin with a neural network to determine our nonlinear mapping. While having the joint PDF could improve our approximation, the true function that minimizes the squared error is indeed the conditional mean function given X.

Now, due to this hierarchy of approximations, there is an excess error which can be quantified. This excess error is given by the difference between the cost function evaluated at $\widehat{W_n}$, the parameter set derived from the time-averaged cost, and the actual cost $J(f^*)$, which we aim to compute. This excess error can be decomposed into two parts.

Consider the cost function $J(\widehat{W_n})$. To analyze this, I add and subtract the cost evaluated at $\widehat{W^*}$, assuming we have the expectation. This approach is expressed in the following form. If we examine these two quantities, the term I've highlighted represents the approximation error, which depends on the neural network model used.

You might be wondering about the Universal Approximation Theorem, which asserts that any function can be approximated by a neural network. However, this theorem does not address the complexity involved. In practice, we must contend with this complexity constraint.

The complexity constraint and the specific architecture of the neural network model, whether it's a feedforward, feedback, or a combination, affect how we determine the optimal parameter set. This dependence introduces an error related to the assumptions made about the model. Additionally, there is another error due to the deviation between $J(\widehat{W_n})$ and $J(\widehat{W^*})$, which depends on the size of the data, denoted by N.

If the dataset is very large, the estimated error will be closer to the true error. Conversely, with a smaller dataset, we have a limited number of data points, leading to potentially higher error rates. This introduces a trade-off between approximation error and estimation error, which is a critical area of study.

To illustrate this further, consider the capacity of the learning machine. For instance, in a single-layer multilayer perceptron (MLP), the capacity to accurately classify or label data without errors largely depends on the size of the hidden layer. The capacity of the learning machine is thus significantly influenced by this layer's size.

Now, let’s consider a family of nested approximating functions denoted as f_k, where f_k depends on the data points and the parameter set W belonging to the space W_k. This family exhibits a containment property: $f_1 \subseteq f_2 \subseteq f_3 \subseteq \cdots \subseteq f_k$, where f_k measures the machine's capacity. A larger number of parameters typically allows the machine to solve the problem more accurately, reducing the approximation error. However, this increased capacity can also lead to overfitting the data, creating a trade-off that we need to address.

It's important to note that when I refer to a single-layer MLP, I mean a network with a single hidden layer. Thus, we are dealing with two components of error in this analysis.

When plotting error against the size of the supporting functions, where K represents increasing complexity, we observe that the approximation error decreases. However, before reaching the optimal point, there is a trade-off to consider. Let me clarify what happens with both the approximation error and the estimation error.

(Refer Slide Time: 29:42)

As the complexity of the machine increases, the approximation error continues to decrease. This is because a more complex model can better capture the underlying patterns in the data. However, beyond a certain point, increasing the complexity further leads to an increase in the estimation error. This is due to overfitting: the machine's capacity becomes too large relative to the amount of detail in the data. Essentially, the model starts to fit noise rather than the true underlying signal.

In the graph, the curve representing the approximation error trends downward, while the curve representing the estimation error trends upward. The "sweet spot," or optimal

solution, is found where these two curves balance. This balance represents the trade-off between approximation error and estimation error, which is crucial for effective data modeling and fitting.

To summarize, before reaching the optimal point, the machine's capacity is insufficient to capture the nuances in the data, necessitating increased complexity. After reaching the optimal point, the machine's capacity exceeds what is needed, leading to overfitting and increased estimation error. Thus, it is essential to find the optimal network size that balances these two errors.

When designing neural networks and determining their size, it is crucial to take these factors into account to achieve an optimal balance between approximation and estimation errors.