Neural Networks for Signal Processing-I

Prof. Shayan Srinivasa Garani

Department of Electronic System Engineering

Indian Institute of Science, Bengaluru

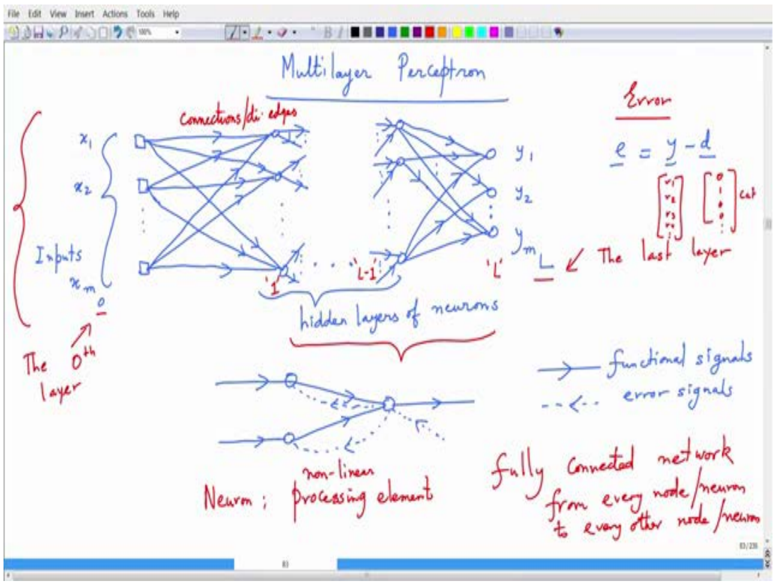
Lecture – 16

Multi-Layer Perceptron 1

Let's dive into this module, where we will explore multi-layer perceptrons (MLPs). In our previous discussions on perceptrons, we focused on solving pattern classification problems using linear decision boundaries. However, for problems that require non-linear decision boundaries and multi-class separation, we need to turn to feedforward neural networks, specifically multi-layer perceptrons.

In this module, we'll delve into the intricacies of multi-layer perceptrons, starting with how to construct such networks. We'll also derive the backpropagation algorithm, which is fundamental to training MLPs. Our exploration will include different modes of operation for this algorithm, such as batch mode and online mode. Additionally, we'll discuss practical strategies to effectively implement and utilize these networks.

(Refer Slide Time: 01:38)



Here is a schematic of the multi-layer perceptron. We start with inputs x_1, x_2, …, x_m, where m represents the number of input features. Although we are not including the bias term here for simplicity, it can be incorporated as we delve deeper into the algorithm. For now, let's assume that we have inputs x_1, x_2, …, x_m, with layer 0 being the input layer.

Each of these inputs is connected to the neurons in the subsequent layers through directed edges, forming a directed graph. These neurons are organized into what we call hidden layers. For example, layer 1 is the first hidden layer, layer 2 the second, and so on. The final layer, which we refer to as layer L, is the output layer.

In the most general form, we can have a fully connected network where every node or neuron in one layer is connected to every node in the next layer. In neural information processing terminology, neurons are sometimes referred to as processing elements. In this context, each neuron acts as a non-linear processing element.

Returning to the architecture, we have connections represented by directed edges from each node to every other node in the following layer. This flow of information between nodes is what characterizes the directed edges. The output layer consists of a set of nodes where the edges from the previous layer ultimately terminate.

The network is organized into a sequence of layers of neurons, with connections cascading from one layer to the next. Since there is no feedback from one node to another within the same layer, this is a feedforward network.

Each layer can have a variable number of neurons. For instance, layer 1 may contain m_1 neurons, layer 2 m_2 neurons, and so on. The final layer, layer L, can have m_L neurons. This flexibility allows us to tailor the network to specific needs without being restricted by a fixed number of neurons in each layer.

Our objective is to formulate a criterion for error measurement. The error vector, denoted as $\mathbf{e}$, is defined as $\mathbf{e} = \mathbf{y} - \mathbf{d}$, where $\mathbf{y}$ represents the actual outputs $y_1, y_2, \dots, y_{m_L}$, and $\mathbf{d}$ is the desired response vector corresponding to these outputs.

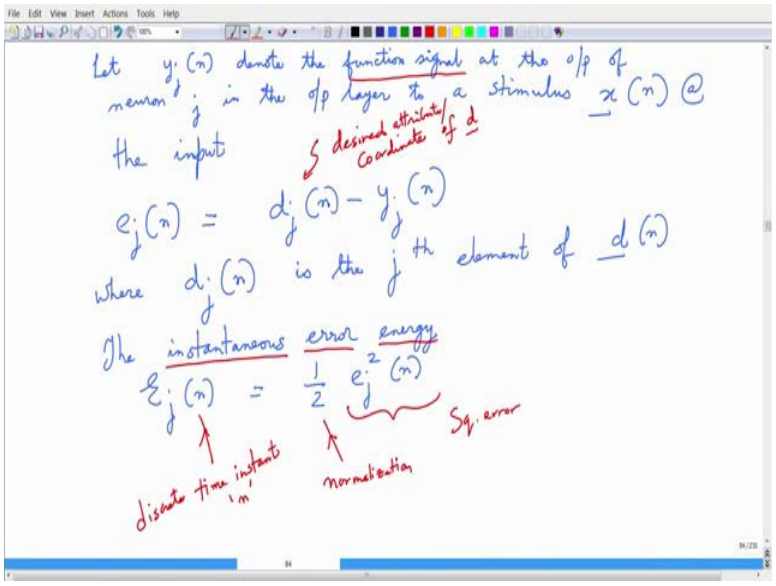
To elaborate, y_1, y_2, …, y_{m_L} are computed based on the connections within the network, given the inputs. Similarly, the desired responses are represented by the vector $\mathbf{d}$. For instance, if you are given an image of a cat in a specific orientation, you would extract feature vectors x_1, x_2, …, x_m (including x_0 for bias). The desired response for this cat might be a vector like [0, 1, 0, 1, 0, 0, 0], where '1' indicates the presence of a cat and '0' otherwise.

In contrast, the output vector **y** could contain real values like $r_1, r_2, r_3, r_4, \ldots$.

This example illustrates the mapping between output vectors and desired responses, which is the foundation of supervised learning.

In the context of signals, we use specific notations: bold blue edges represent functional signals, while dotted directed lines indicate error signals. It's crucial to understand why error signals are shown as moving backward. During computation, we first process inputs through the hidden layers to the output layer. After obtaining the outputs, we measure the error, which is computed in the output layer. To adjust the weights accordingly, the error must be propagated backward from the output layer through the hidden layers to the weights connecting the inputs to the first hidden layer. This backward propagation of error signals is a fundamental principle for adjusting weights in the network, as illustrated by the dotted and directed edges in our legend.

(Refer Slide Time: 10:03)



Let $y_j(n)$ denote the function signal at the o/p of neuron j in the o/p layer to a stimulus $\underline{x}(n)$ @ the input

desired attribute/ Coordinate of $\underline{d}$

$$e_j(n) = d_j(n) - y_j(n)$$

where $d_j(n)$ is the j^{th} element of $\underline{d}(n)$

The instantaneous error energy

$$\mathcal{E}_j(n) = \frac{1}{2} \underbrace{e_j^2(n)}_{\text{Sq. error}}$$

discrete time instant 'n'

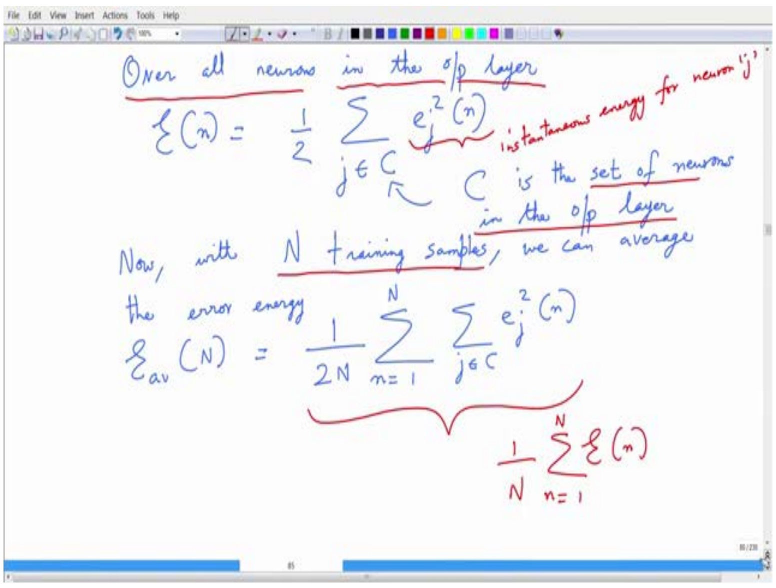
normalization

Let's proceed with our discussion. Let $y_j(n)$ represent the functional signal at the output of neuron j in the output layer at time step n. Each neuron in the output layer has a functional signal, and there is a corresponding desired response, which can be thought of as an attribute or coordinate. We denote y_j as the functional signal and d_j as the desired response or desired coordinate of the vector **d**. The error, denoted $e_j(n)$, is given by $d_j(n) - y_j(n)$. This error represents the deviation between the desired response and the actual functional signal.

Now, let's define the instantaneous error energy. It's called "instantaneous" because it pertains to time step n, and we must remember that this is a discrete time step. The instantaneous error energy is given by $\frac{1}{2}e_j^2(n)$. You might wonder why we include the factor of $\frac{1}{2}$. The reason is to simplify the process of taking derivatives. When we take the derivative of this instantaneous energy function with respect to the weights, the factor of $\frac{1}{2}$ ensures that the 2 in the squared term cancels out, making the derivative of the error function straightforward.

Additionally, squaring the error eliminates the concern about the sign of the error. This is crucial when using techniques based on quadratic optimization, where we are interested in minimizing the magnitude of the error rather than its sign. The squared error provides a measure of magnitude, which aligns with the goals of optimization techniques that rely on quadratic error metrics.

(Refer Slide Time: 13:05)



Over all neurons in the o/p layer

$$\mathcal{E}(n) = \frac{1}{2}\sum_{j\in C} \underbrace{e_j^2(n)}_{\text{instantaneous energy for neuron 'j'}}$$

C is the set of neurons in the o/p layer

Now, with N training samples, we can average the error energy

$$\mathcal{E}_{av}(N) = \underbrace{\frac{1}{2N}\sum_{n=1}^{N}\sum_{j\in C} e_j^2(n)}_{\frac{1}{N}\sum_{n=1}^{N}\mathcal{E}(n)}$$

Let's delve into the basics of computing the error energy for our multi-layer perceptron. To begin, we need to calculate the error energy for all the neurons in the output layer. This is done by summing the instantaneous error energy across all neurons in the output layer. The formula for this is:

$$E(n) = \frac{1}{2}\sum_{j\in C} e_j^2(n)$$

Here, $e_j(n)$ represents the instantaneous error for neuron j at time step n, and C denotes the set of neurons in the output layer. By summing these squared errors, we obtain the total instantaneous error energy for the network.

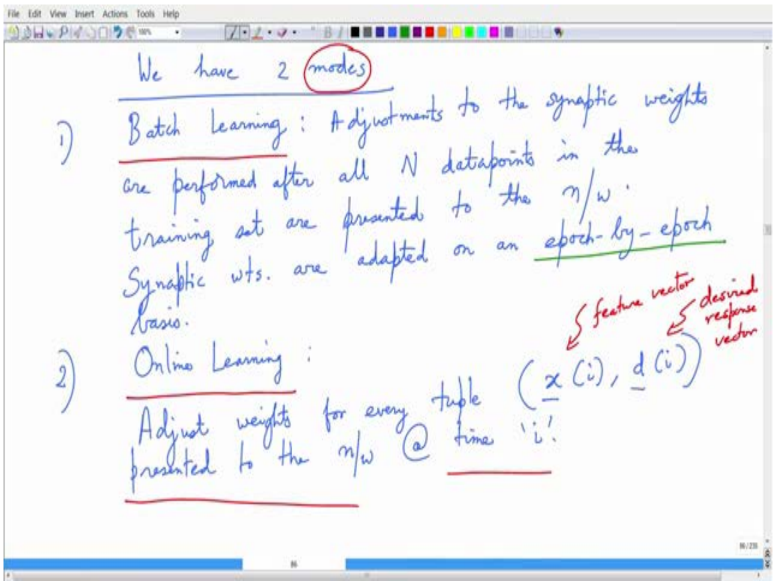
However, this calculation pertains to just one sample. To get a comprehensive measure of performance, we must consider all N training samples. Here, N represents the total number of training samples, while n refers to the discrete time step. We average the error energy over all N training samples to obtain the mean error energy. This is calculated as:

$$\text{Average Error Energy} = \frac{1}{N}\sum_{n=1}^{N} E(n)$$

This average error energy provides us with a performance metric for the training error of the network. The goal is to minimize this error with respect to the network's parameters to achieve the best possible estimates and ensure that the network has effectively learned the data's underlying patterns.

It's important to note that this error metric is quadratic in nature. Adopting a quadratic cost function simplifies the algebra and makes it easier to derive a training rule. This approach is beneficial for its mathematical tractability and, in many cases, provides a unique solution.

(Refer Slide Time: 16:50)



We have 2 modes

1) Batch learning : Adjustments to the synaptic weights are performed after all N datapoints in the training set are presented to the n/w. Synaptic wts. are adapted on an epoch-by-epoch basis.

2) Online Learning :

Adjust weights for every tuple $(\underline{x}(i), \underline{d}(i))$ presented to the n/w @ time 'i'.

Lastly, we will discuss the specific criterion for updating the neurons and the operation of the algorithm in more detail in the following sections.

Let's explore the two primary modes of learning in neural networks: online learning and batch learning. I'll begin with online learning, as it is more straightforward, and then we'll delve into batch learning.

Online Learning:

In online learning, weights are updated incrementally for each data point presented to the network. At each time instant i, a feature vector x_i and its corresponding desired response d_i are fed into the network. This means that every time a new example is presented, the network adjusts its weights based on this single example. This continuous updating process occurs with each new tuple of data, making online learning a dynamic and adaptive method.

You might wonder why online learning is handled this way. The key advantage of online learning is its adaptability to new data, as it updates the network weights immediately. However, this approach also has its drawbacks, which we will discuss later. Essentially, online learning updates weights with every data point, which can be both beneficial and challenging depending on the scenario.

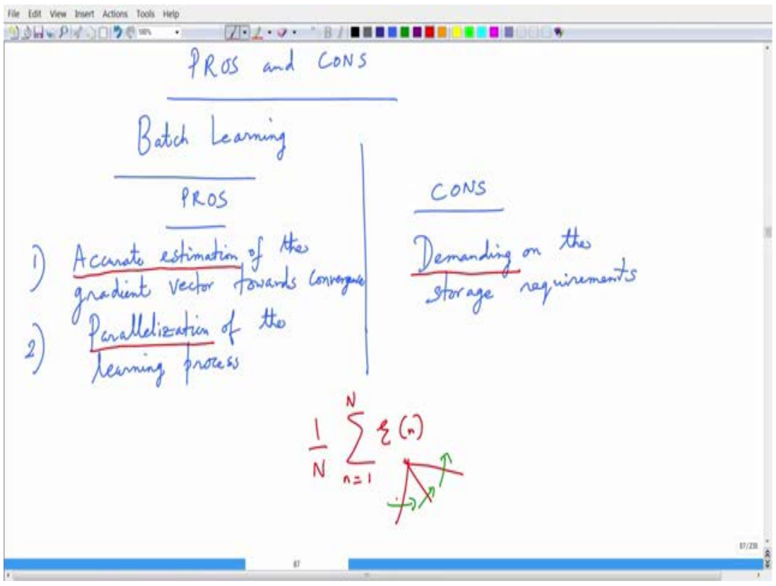
Batch Learning:

In contrast, batch learning involves adjusting the weights only after all data points in the training set have been processed. During batch learning, the entire set of N training examples is presented to the network in one pass, and then the weights are adjusted based on this complete dataset. This process is repeated on an epoch-by-epoch basis, where one epoch consists of a full pass through the training data. After each epoch, the weights are updated accordingly.

Batch learning typically involves multiple epochs, where the data may be shuffled before each pass to ensure the network does not learn any unintended patterns based on the order of the data. This is especially useful if the algorithms do not depend on the temporal sequence of the data. Shuffling the data can help improve the training process and avoid overfitting to specific sequences.

Both online and batch learning have their respective advantages and disadvantages. Online learning is beneficial for real-time applications where data continuously streams in, whereas batch learning can be more stable and effective for large, static datasets. The choice between these methods depends on the specific needs and constraints of the problem at hand.

(Refer Slide Time: 20:00)



PROS and CONS

Batch Learning

PROS

1) Accurate estimation of the gradient vector towards convergence

2) Parallelization of the learning process

CONS

Demanding on the storage requirements

$$\frac{1}{N}\sum_{n=1}^{N} \mathcal{E}(n)$$

Let's dive into the pros and cons of batch learning versus online learning, starting with batch learning.

Advantages of Batch Learning:

1. Accurate Gradient Estimation:

Batch learning provides a more accurate estimation of the gradient vector towards convergence. Since the network processes all examples or data points within one epoch, it computes an average error energy for that epoch. This comprehensive view allows for a better gradient estimation compared to online learning, where the gradient is updated per sample and can be quite noisy. The average error captured in batch learning leads to a more stable and precise gradient calculation.

2. Parallelization:

Batch learning facilitates the parallelization of the learning process. Since the error energy is averaged over all samples in the batch, this process can be distributed across multiple engines or processors. For instance, you can divide the computation of the error and its gradient among different processors, significantly speeding up the training process. This parallelization is not feasible with online learning due to its sample-by-sample approach, which does not provide an average error across all training samples.

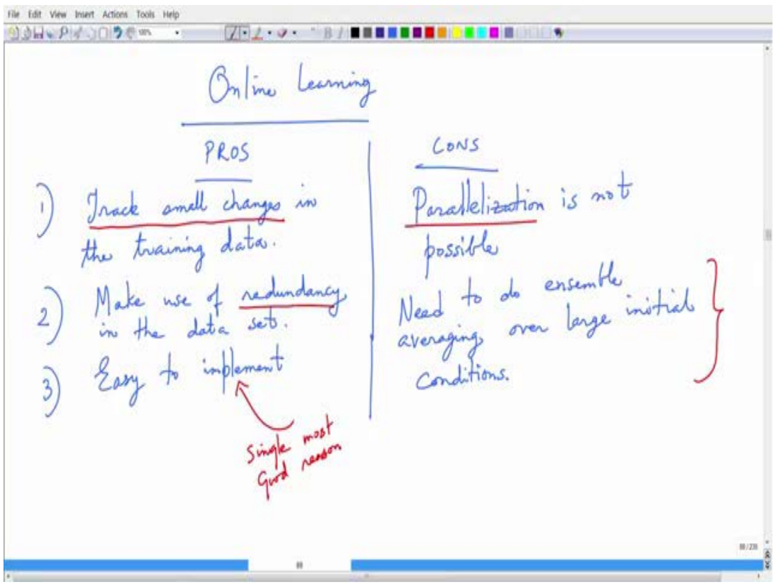
Disadvantages of Batch Learning:

1. High Storage Requirements:

One major drawback of batch learning is its high storage requirement. With modern AI applications often dealing with millions or even billions of records, storing all this data along with the weights can be very demanding. The need to process and store large datasets necessitates powerful storage solutions, such as GPUs or cloud-based infrastructure. This requirement for extensive storage and computation resources is a significant challenge in batch learning.

In summary, while batch learning offers precise gradient estimation and the benefits of parallelization, it comes with the challenge of substantial storage demands. As a result, many applications turn to GPU or cloud-based solutions to handle the large volumes of data and computation required.

(Refer Slide Time: 23:38)



Online Learning

PROS

1) Track small changes in the training data.

2) Make use of redundancy in the data sets.

3) Easy to implement

Single most Good reason

CONS

Parallelization is not possible

Need to do ensemble averaging over large initial conditions.

Let's delve into the details of online learning. We previously discussed how online updates are performed on a per-sample basis: each time an input vector and its corresponding desired vector are presented, the weights are updated immediately. This approach has both advantages and disadvantages.

Pros of Online Learning:

1. Tracking Small Changes:

Online learning is adept at tracking minor fluctuations in the data. Because updates occur at each step, the system can monitor subtle changes in the data and adapt the weights accordingly. This enables the network to respond dynamically to shifts in data patterns.

2. Utilizing Data Redundancy:

Online learning can exploit redundancy within the dataset. If a particular feature vector is highly redundant, it can be discarded to save time and storage costs. This is an advantage that batch learning lacks, as batch learning averages data and does not accommodate such flexibility.

3. Ease of Implementation:

One of the significant benefits of online learning is its simplicity in implementation. In scenarios where hardware constraints are a concern—such as in neuromorphic systems where power and area are limited—online learning's straightforward approach becomes highly valuable. Its ease of implementation makes it an attractive option for practical applications.

Cons of Online Learning:

1. Lack of Parallelization:

Online learning does not support parallelization effectively. Since updates occur one sample at a time, there is no opportunity to parallelize the process. Each sample requires an individual update to the weights, limiting the efficiency of processing large datasets.

2. Need for Ensemble Averaging:

Online learning often requires averaging over a large number of initial conditions to obtain a stable gradient estimate. Because the gradient updates are noisy and occur continuously, averaging over various initial conditions can help achieve a more accurate estimate. This need for extensive averaging can complicate the process, though its impact may vary depending on the specific application.

In summary, while online learning offers the benefits of adaptability, redundancy utilization, and ease of implementation, it faces challenges in parallelization and gradient estimation. These factors must be considered when designing and implementing online learning algorithms, whether for hardware or software applications.

Thank you.