Introduction to Machine Learning
Prof. Sudeshna Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 27
Introduction

Good morning, welcome to today's class. This week we will talk about neural networks. So, neural networks is one of the most active topic of research in machine learning nowadays because of the capability of the neural networks to represent and learn highly complex and non-linear functions. We will see that neural network was inspired by the human brain; human beings are very intelligent and can do certain tasks extremely well and this inspired people to try to understand how human brain works.

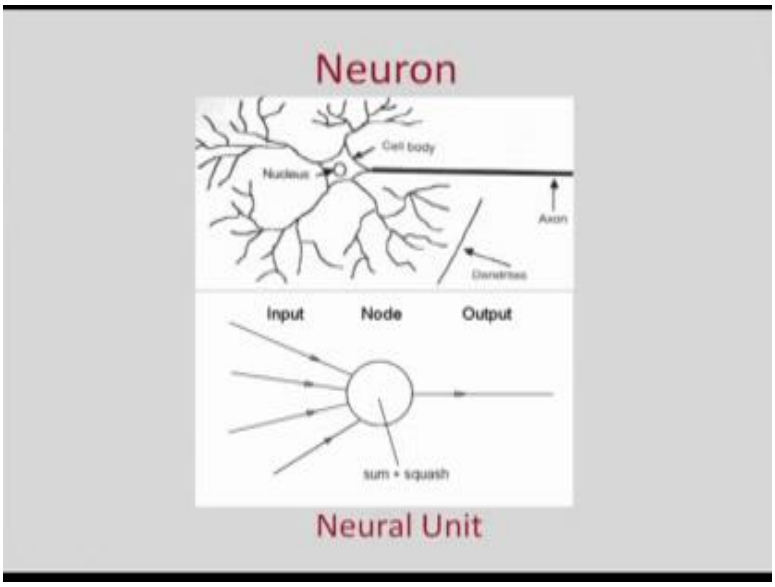
(Refer Slide Time: 01:07)

Human brain contains a number of neurons, of the order of tens of billions. So, tens of billions of neurons are there in the human brain which is highly interconnected and these individual neurons are simple computing units, but together they can perform very complex tasks. There are certain characteristics of neurons which have been incorporated while trying to form the architecture of neural networks. So, these characteristics are massive parallelism. There are many units which are individually simple, but they work

together in parallel to share, to achieve complex tasks.

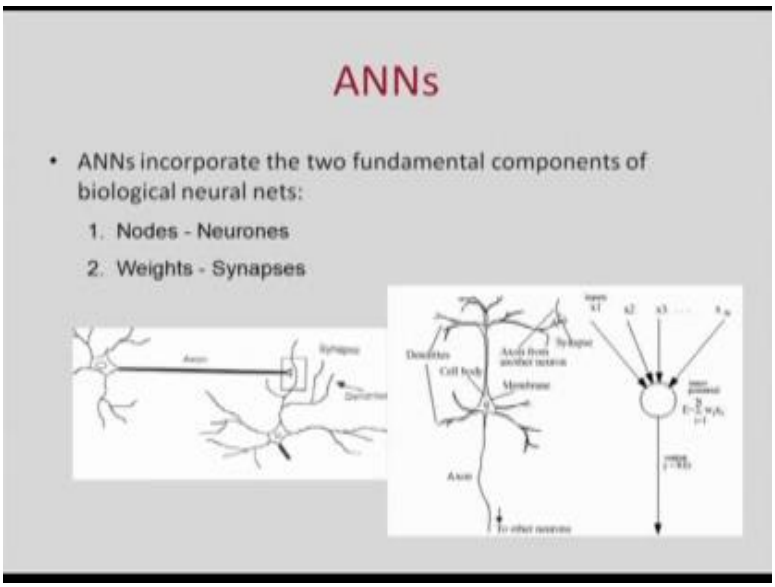
Number 2 is these units are highly interconnected with each other and through this interconnection, they can solve a task together and third is they can model distributed associative memory through weights in the sign update connections.

(Refer Slide Time: 02:42)



Now, if we look at the slide it shows the schematic diagram of a neuron. This is the neuron cell body which has the nucleus and these are the dendrons through which input is expected and this is the tail or the axon through which the output is given. So, this structure of the neuron is simulated by a neural network unit which has inputs and then some simple computation are the node and output, and the computation of the node first computes the weighted sum of the inputs and then applies a function it could be a squashing function like the sigmoid function or some other function. So, this is the node, this is the input and this is the output.

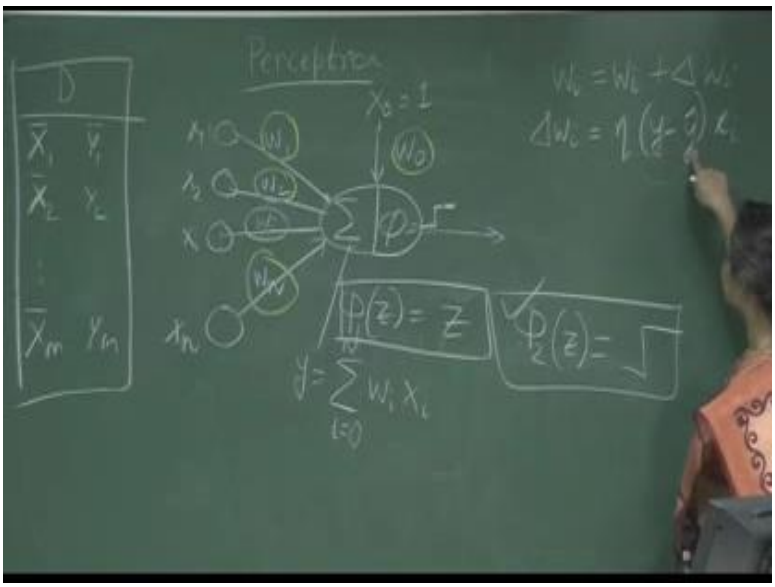
(Refer Slide Time: 03:41)



If you look at this particular diagram you can see that this is a neuron and through the axon that is the output. This output feeds into the input of another neuron through this synaptic connections. So, this is the synapse, this is the axon and these are the dendrites. So, through the dendrites the input is accepted and through the axon the output is transmitted through the electric impulses through the synaptic layer to the other neurons, and this is what inspired the neural network architecture.

Now, the exact neural network architecture is inspired by the human brain when people have come up with architectures. They may not be exactly similar always to how the human brain works, but this is the inspiration. In today's talk, we will talk about single layer neural networks. We have already talked about single layer units while talking about linear regression, while talking about logistic regression; nevertheless we will just go through it again. The basic in unit in a neural network is called a Perceptron.

(Refer Slide Time: 05:06)



Now, in a perceptron as we have seen, it has n inputs and let us denotes them by X 1, X 2, X n and these are the inputs to a perceptron and in this perceptron unit there are 2 parts; first a weighted summation of the input is computed. There is also another unit input called the bias and so this input is computed and this input is passed through another transfer function to the output and we can denote this transfer function by phi.

So, if you have a linear unit phi Z is just Z. So, just the input say summation is passed, this is what was happening in linear regression or this transfer function can take different forms, for example, sigma Z could be thresholding function. So, if we have a threshold and if the summation is greater than the threshold, you output 1 or the summation is less than the threshold, you output 0 or it could be some other non-linear function, for example, we will talk about the sigmoid function, the tan hyperbolic function. We have already talked about the sigmoid function when we talked about logistic regression. So, there are several transfer functions which are possible, but first let us look at the simplest type of perceptron which let us say the users are linear transfer.

So, at this point Y equal to sigma W i X i is computed. So, sigma W i X i, i equal to 1 to n plus this bias let us say, b this is computed. Another way of looking at it is that instead of writing b for the bias we can associate W 0 here and keep a fixed input X 0 defined to

be 1, in that case we can write this as Y equal to summation i equal to 0 to n. So, this is what is computed at the output of this unit and then depending on the value of phi if phi was identity this output will be transmitted, otherwise this phi is apply to 1 and as I said a second type of phi. So, phi 1 Z equal to Z let us say phi 2 Z is a thresholding function right. So, this thresholding function can be applied and the output will be given as 0 or 1, this is a basic architecture of a single perceptron.

Now, in a perceptron, this links are associated with which W 1, W 2, W n. Now, if you consider supervised learning, we have looked at different algorithms was different methods for supervised learning. If you use supervised learning using this neural network what we have is a set of training examples D and D comprises of X 1 Y 1, X 2 Y 2, X m Y m. So, these are the training examples that I have right. Now, based on the training example we want to train this network, what does training the network mean? Training the network means learning these weights W 0, W 1, W 2, W n. So, we want to learn the values of the weights W 0, W 1, W 2, W n given the training examples, so that this particular network has a good fetch to the training examples. So, we have a training algorithm for perceptrons.

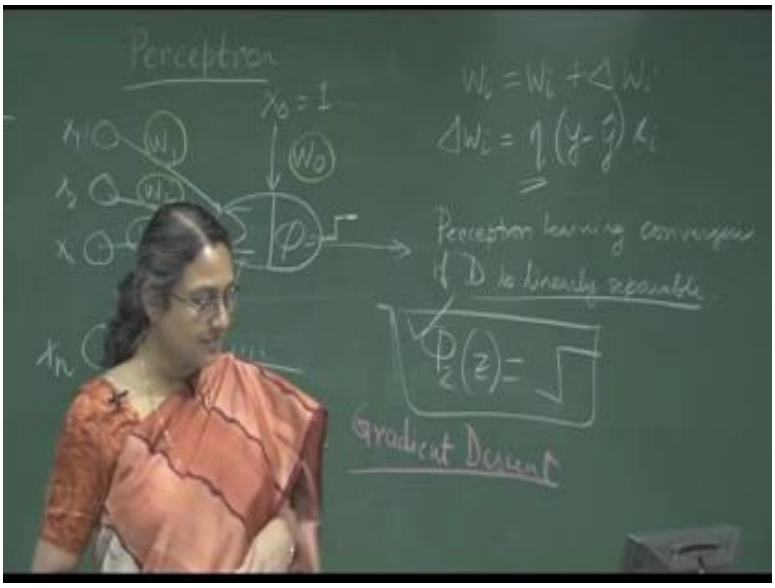
Let us first look at a very simple way of training a simple perceptron, which comprises of this threshold function. So, we will look at the perceptron training rule in perceptron training rule what we do is that initially, when we set up this network we have some initial values of the weights W 0, W 1, W n have some initial values of the weights and at each iteration we will update the weights. In the simplest training rule what we do is that we feed example 1 at a time to this network and based on how the network performs on the example we update the weights. Suppose, we feed X 1 to this network and we get an output right the output let us say if we pass through thresholding function output is 1 or 0 here also we have a classification problem with the output as 1 or 0.

So, if the output of the network is same as Y 1 then we do not need to change the weight, but if the output is different that is suppose the output should have been 1, but I am getting a 0 then the weights have to be updated. So, how we update the weight we update the weight as this is the initial weight of W I, initial value of W i plus delta W i and how is delta W i computed, we change W I, so that the output is more likely to be closer to the

target output and this is the training rule that is employed delta W i is eta times Y minus Y hat X i. So, this X i, this is a vector.

So, we feed a particular example let us say X 1 vector which has this different components X 1, X 2, X n corresponding to the different features and W i is corresponding to the feature X i. So, W i is corresponding to the feature X i and we change W i. So, that in this way eta Y minus Y hat X i, Y is the target output and Y hat is what you get through this network. If Y and Y hat are equal this term is 0. So, there would be no update otherwise you change W i based on Y minus Y hat times X i, where X i is the input to this network. So, this is how you change W i and if Y was 1, Y hat is 0 and if X i was 1 you can see that we are increasing the weight and eta is a factor which is called the learning rate, it controls how much you change based on 1 error. So, based on the error you are changing the weights.

(Refer Slide Time: 13:55)

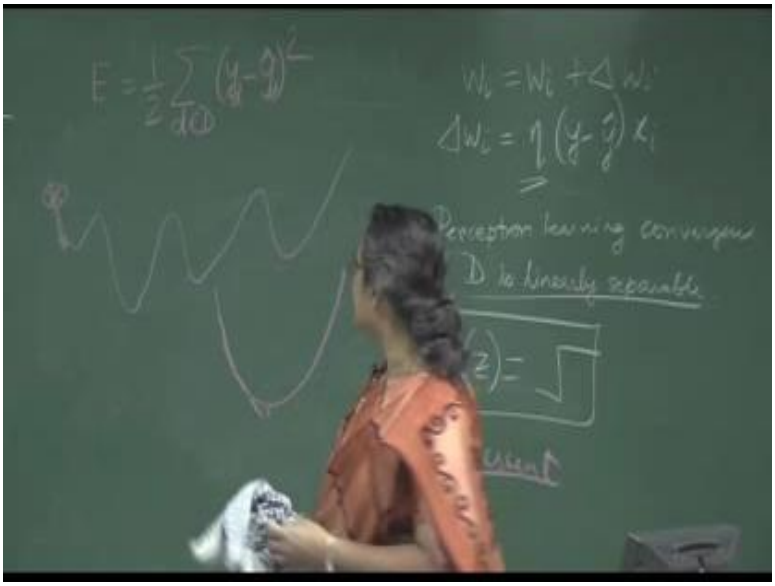


Now, this is a very simple the training rule and if you apply this particular algorithm by taking the examples one by one. It can be shown that this perceptron learning converges to a consistent model, if D, the training example is linearly separable. We have already talked about what we mean by linearly separable; that means, there is a linear decision surface that separates the positive and negative examples if such a linear separation

surface exists then by applying this perceptron training rule, the learning algorithm will converge to hypothesis which will separate the positive and negative examples, but if the data is not linearly separable then this perceptron learning algorithm will not converge, it will not work.

So, we have to look for alternate algorithms and we have already discussed about gradient descent. So, gradient descent algorithms can be used in the more general case. If you have a situation where you know there may not be a linear separability, but with respect to a particular value of the parameters, the weight values you can define the error of the network right and you can perform gradient descent on this error function to find that value of the parameters for which this error function is minimized. So, this is done by gradient descent which we have already discussed earlier.

(Refer Slide Time: 16:01)



In gradient descent what happens is that we first define an error function for example, here the error function can be defined as sigma Y minus Y hat whole square over all training examples. So, we can subscript it by Y d minus Y d hat. So, Y d is the actual value of the output for that training example and Y d hat is what you get through your network. So, this is the error with respect to network in some cases we put half in order to you know just have a nice form of the final output.

So, we have this error function and we want to minimize this error and how we minimize it is that we try to find the find the weights through a process called gradient descent. So, now, this error is a function which has a surface and we want to find the minima of this surface. So, there are could be of function like this and we want to find the minima.

Suppose, at a particular point we are at this point of the error surface. In gradient descent what you do is that you find the derivative at the error surface you rather find the partial derivative with respect to each and every weight, and based on that you find the weight of the you find the direction of the gradient and you take a step in the negative direction of the gradient. So, as to go towards the minima right in certain cases for single layer perceptron, this error surface will be convex or quadratic and there is a single minima and if you do gradient descent you are guaranteed to ultimately reach this minima.

But there are other cases when we talk about multilayer networks in the next class. We will see that the error surface is ill behaved, it is not necessarily convex and there can be some local minima which you can get started, but basic idea of gradient descent is you find the gradient and then you take a step towards the gradient. We have already talked about gradient descent, but this is important.

(Refer Slide Time: 18:33)

Now, we have seen 2 examples, one is when we have a linear. Now, one of the problems that will mention is that when we use this step function, this step function is not differentiable, we cannot do gradient descent. So, we have to take some other function which is differentiable, for example, the simple linear function phi 1 Z equal to Z this is differentiable and we can do a gradient descent on this which we already did earlier, and based on that we talked about gradient descent we talked about a gradient descent where you take this error function and by this we can get to the local minima we can find out the values of the weight, so that this is minimized.

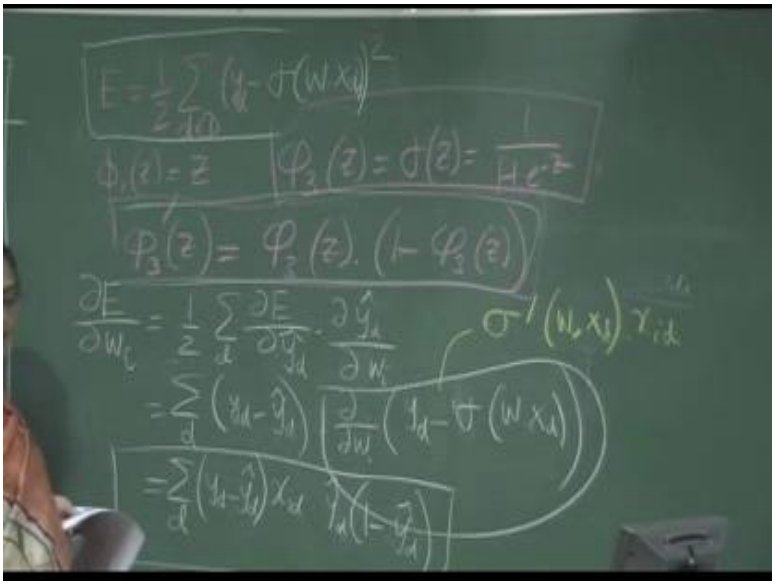
We can also do stochastic gradient descent where we take a single example from the training set at a time, define the error as Y minus Y hat whole square. Based on that we can we can take one example at a time based on this we change weight this gives us stochastic gradient descent which is passed right. So, we have already worked this out for linear neurons.

Now, the basic idea in gradient descent is that the delta W i that we compute is equal to minus eta del E by del W i. So, we take the partial derivative of the error E with respect to this weight and we find the direction of the gradient and we take a small negative step minus given by minus eta in this direction. This is how we change the weight at each iteration and if we do gradient descent on a linear function we get function like this what we get is, we will get for linear which we already worked out earlier it is a what we get is X i j Y j minus Y j hat. So, we get this and we get the training rule for using gradient descent which is actually similar to the training rule that we used for the perceptron.

So, we also saw in the previous week that for classification problem or even for other reasons which we will explain instead of. So, let me explain it now only, in the next class we will talk about multilayer networks. We will show that single layer networks can only handle linear decision surfaces, but if we want to capture non-linear functions we have to go for multilayer networks. Now, in multilayer networks we have these different networks connected with each other, but if we connect linear units with each other the combination will again be a linear function. So, in order to have a complex to able to represent complex functions, we want non-linear unit and we want non-linear units which a differentiable that is why we go for a transfer function which is differentiable

and non-linear.

(Refer Slide Time: 22:38)



And one of the transfer functions that we often use is the sigmoid function, which is given by 1 by 1 plus e to the power minus z. So, this is the logistic function or the sigmoid function which is one of the most popular transfer functions which you has been traditionally used for a neural networks.

So, if we use this logistic unit as the transfer function, we can figure out how to do gradient descent with this particular transfer function. So, let us say phi Z equal to 1 by 1 plus e to the power minus z. We can differentiate this which we already did earlier. So, and we found that the differentiation of this function can be written as phi 3 Z times 1 minus phi 3 z. So, this function is differentiable is differentiable and the result of the differentiation can be very simply expressed in terms of the value of the function itself.

Now, with respect to logistic function, these sigmoid functions, let us see how to compute the gradient. So, this is my error e equal to half, this is my error. So, and what is Y d hat Y d hat is in this case sigma W dot X d whole square. So, we have W dot X t which is the summation input and this pass through the sigmoid function. So, this is my error function. Now, if we take the partial derivative of the error function with respect to

the weight W i what we get is half sigma d del E by del Y d, we do a transfer of variables and chaining and del Y d by del W i.

Now, del E by del Y d, so what we get from here is simply summation over d. So, E equal to Y minus Y hat whole square. So, this will become 2 into Y minus Y d. So, we just get from this Y d minus Y d hat times del del W i of Y d minus this psi or this sigma apply to W dot X t. So, Y d has nothing to do with W i, this is del del W i apply to minus of sigma W dot X t. So, this gives us, this part gives us this sigma dash of W X d times X. So, this is W 1, W 1 X 1, W 2 X 2, W 3 X 3 of d and only one of those terms will contain W i. So, the rest of the terms are independent of W i.

So, corresponding to that term we get X i d. So, this is X i d terms times sigma dash of W dot X d which will give us summation over d Y d minus Y d hat corresponding to the dth training example then we have X i d. We have before that we have Y d, this is Y d times. So, wait let me write X i d before. So, this is times X i t times sigma dash W dot X d which by using this formula, this can be written as Y d hat into 1 minus Y d hat. So, this is partial derivative of E with respect to W i.

(Refer Slide Time: 28:41)

Now, based on this we can write the weight training rule as let me rub this out, so that I

have space. We can write delta W i equal to eta times sigma d Y d minus Y d hat Y d hat 1 minus Y d hat times X i d right. So, this is the training rule for sigmoid units and as we have already seen that we can use this, we can do a single layer logistic unit and find its weights, but as I have already told that the limitation of single layer neural network is that they can only represent linearly separable functions.

We have already looked at SBM they can only represent linearly separable function. Of course, in SBM what we can do is that we can try to represent non-linear function by transforming the features space and having a linear function in the transformed features space. What we will do instead in multilayer neural network is that we will try to represent non-linear function by stacking many of these units together which we will see in the next lecture.

Thank you.