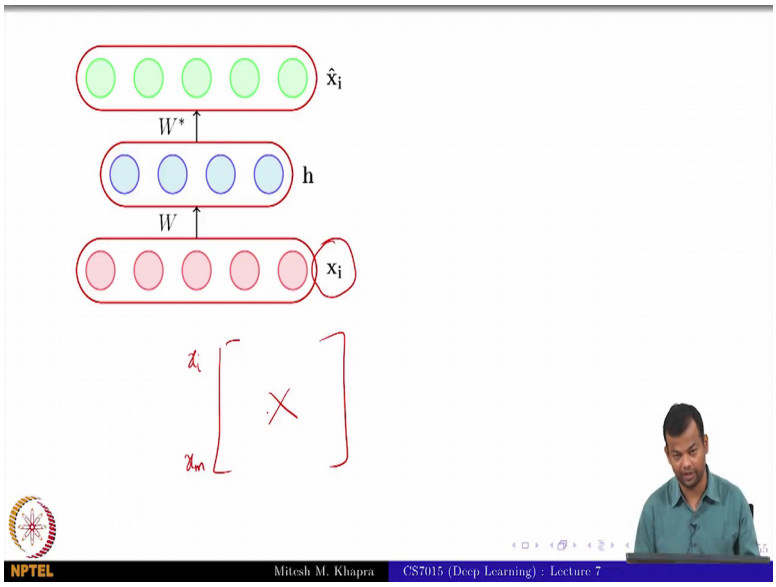
Deep Learning
Prof. Mitesh M Khapra
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 07
Autoencoders and relation to PCA, Regulation in autoencoders, Denoising autoencoders, Sparse autoencoders, Contractive autoencoders

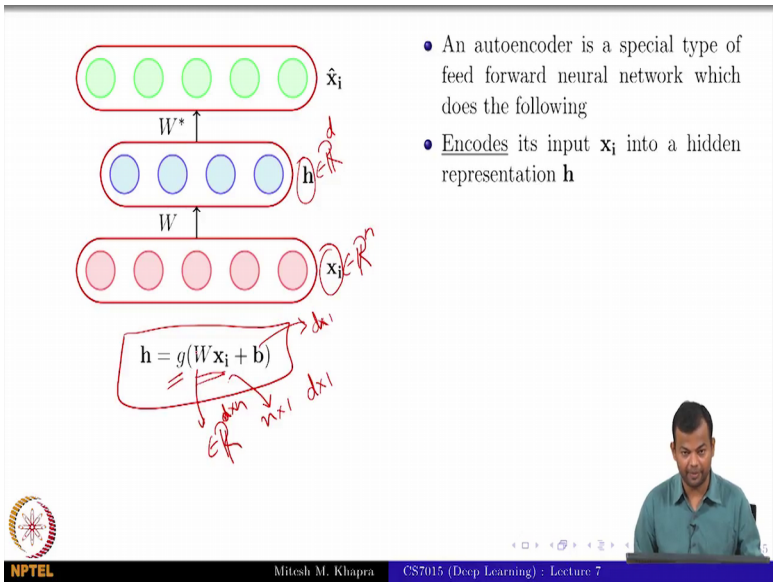
Welcome to lecture 7 of the course on Deep Learning CS 7015. In this lecture we are going to talk about auto encoders and we will focus on their relation with PCA. Then talk about regularization in auto encoders, wherein we will look at denoising auto encoders, sparse auto encoders and contractive auto encoders. So, let us begin with the introduction to auto encoders what they are.

(Refer Slide Time: 00:36)



So, this is what a typical auto encoder looks like. And as you can see this is very much like a feed forward neural network you have an input which is x i. So, you are given some training data you are given some ie samples x i to x m. So, this is your training matrix x which we have seen in the previous lectures. So, this is one of those training inputs x i, and then you have a hidden layer and then an output layer. So, let us look at what is the configuration of the hidden layer and what does the output layer actually try to do.

(Refer Slide Time: 01:08)



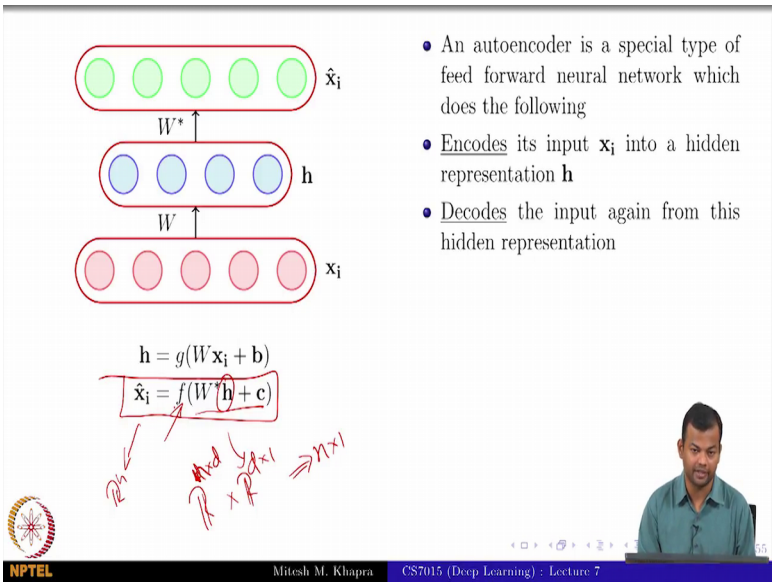
So, it is a very special type of a feed forward neural network. What it does is it encodes with input x i to a hidden representation h ok, and it uses an encoded function to do this. So, this is what the encoded function does. It first does a linear transformation.

So, W is a matrix and x i is a vector and you again have the bias b as a vector right. So, let us look at these dimensions right, so let us try to fix some dimensions. So, suppose x i belongs to R n that is what we have been considering throughout the course. So, far and let us say h belongs to R d.

So, it is a d dimensional representation. So in that case what would W be yeah. So, W would be R n cross another d cross n right. So, it will multiply with the n cross 1 vector which is x i and give you a t cross 1 output right. And similarly the b would also be d cross 1, and then on top of that you have this non linearity g, which will be operating at element wise just as we had seen earlier. So, it could be any of the sigmoid functions the logistic or tannish and so on.

So, the end result is you have taken an input x i and encoded into a hidden represent h by using a linear transformation first, and then a non-linear transformation right. So, I refer to W x plus b as a linear transformation, because it is a matrix multiplication. Now once you have constructed this hidden representation.

(Refer Slide Time: 02:41)



the job of the decoder or the latter half of the feed forward neural network which is this half is to take this encoded representation. And then try to reconstruct x again from it.

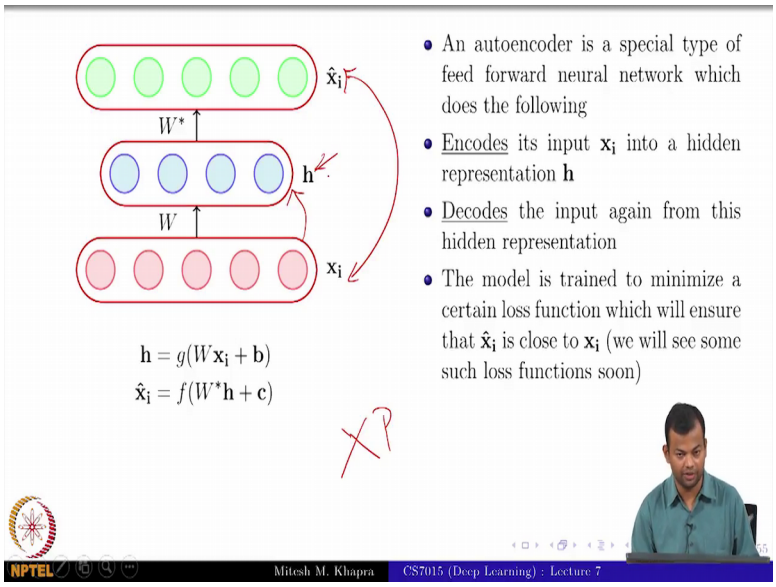
So, again let us first look at the equation. So this is the equation for the decoder, where again you first take the hidden representation do a linear transformation and then you again have some function on non-linearity on top of it right. So, we will see what this function can be so we will refer to it as f for, now we will not say whether this is sigmoid or linear or what kind of a function it is, we will come back to it later on.

So, now let us again look at these dimensions. So what is x i x i is again R n and your h was R d. So, you have to go from a d dimensional input to an n dimensional output. So, again your W star is going it to be R d cross sorry R n cross d. So, it will multiply with a d cross 1 vector and give you an n cross 1 output right. And that will pass through some function and it will give you x i hat which is a reconstruction of x i.

So, why are we trying to do this right we took an input x i, we computed it is hidden representation by doing some non-linear and linear transformation and then again we are trying to reconstruct x i hat. So, why are we trying to do this? So reason we are doing this is that we want to learn, what are the most important aspects or most important characteristics of the input data x i right. So, if you compute a hidden representation H, which is presumably smaller than your original input data.

And from that hidden representation if you are able to reconstruct x i right. Then that would mean that this hidden representation captures everything that is required or everything that is yeah everything that is required to reconstruct x i from x i from the original input right.

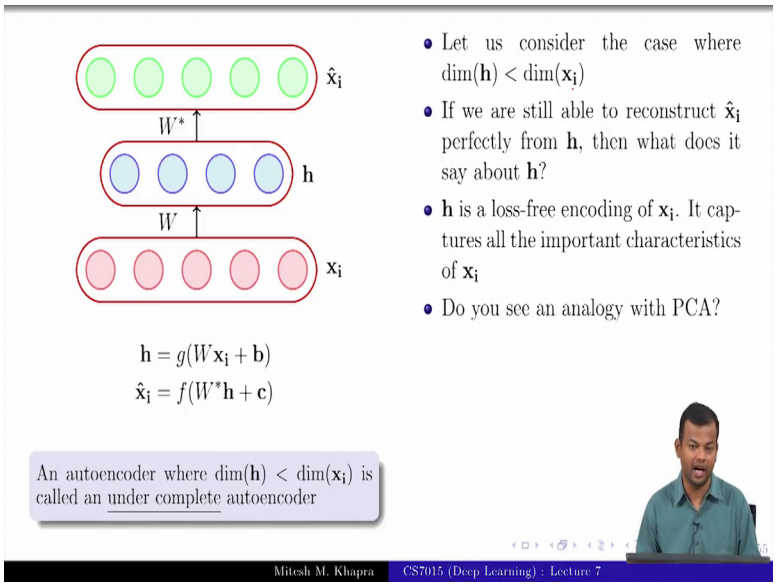
(Refer Slide Time: 04:37)



So, the model will be trained to minimize the difference between x i and x i hat. So, you want to make sure that after passing through this bottleneck which is the hidden representation you are able to reconstruct x i, and the reconstructed output is very close to the original input right.

So, can you see an analogy with PCA, where you are trying to find this hidden representation or this most important elements of the original input x i. So, there we had used this linear transformation where we are taking the original input x. And transformed it to a new basis and we had used that basis for representing the original input right. So, something similar is happening here, we are using this hidden representation h to represent our original input.

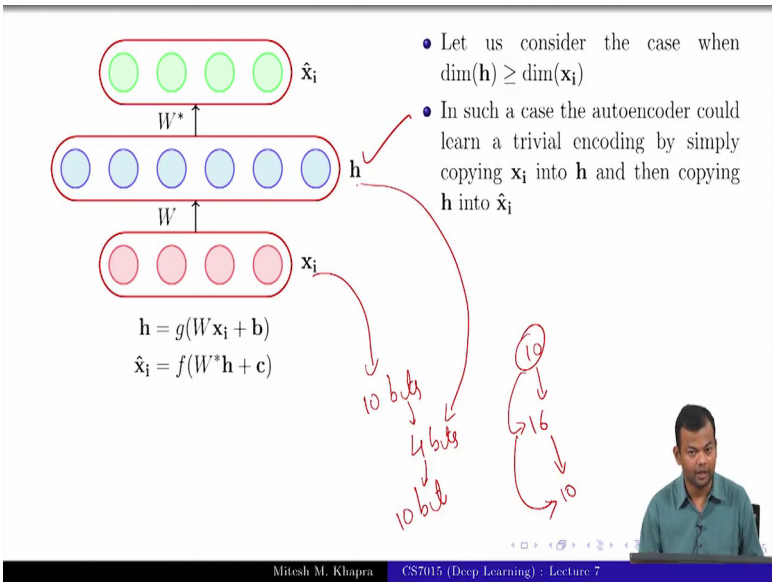
(Refer Slide Time: 05:22)



Now, let us consider a few cases the first cases when the dimension of h is less than the dimension of x i. In this case as I was trying to say earlier if we are still able to reconstruct x i hat perfectly from h. Then what does it say about h it tells us that h is a loss free encoding of x i. It captures all the important characteristics of x i write just repeating what I had said on the previous slide. And now you can see an analogy with PCA because h has all the important characteristics required from the original input data.

So, it has probably got rid of all the noise or all the low variance dimensions or the correlated dimensions and so on. And this is just the compact representation, which is as good as the original representation and from there you can reconstruct the original representation. And such an auto encoder where the dimension of the hidden representation is less than the dimension of your original input is known as an under complete auto encoder.

(Refer Slide Time: 06:17)



Now, let us look at the other case where the dimension of the hidden representation is greater than the dimensional of the original input, such an auto encoder is I will tell you what it is called. So, we will we are looking at the case where the dimension of the hidden representation is greater than the dimension of the original input right.

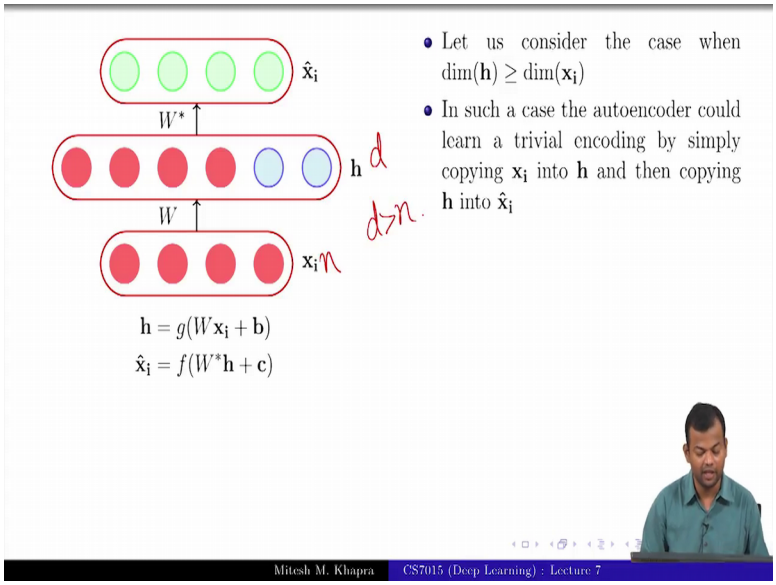
So, now in such a case the auto encoder could learn a very trivial encoding by simply copying x i into h and then copying h into x i right, so think of this from a compression point of view right. So, now, suppose you have 10 bits initially right, and then you want to somehow compress it and store it only in 4 bits.

And now this 4 bits should be such that it captures everything that was there in the original 10 bits because you would want to reconstruct the original input again right. So, this is what we do typically when we compress any of our files right we have a larger file we compress into a smaller information while making sure that everything important is there. So, that whenever I want to recover it, I can just recover it from there right

So, this is definitely a hard task, but now what I am doing in this auto encoder is that I had 10 bits, I am actually giving it more bits now because the dimension of h is greater than the dimension of the input. And then from these 16 bits, I want to reconstruct the 10 bits now this is a very trivial task right because all I could do is copy these 10 bits into the first 10 bits here leave the remaining 6 blank. And then from those 10 bits just

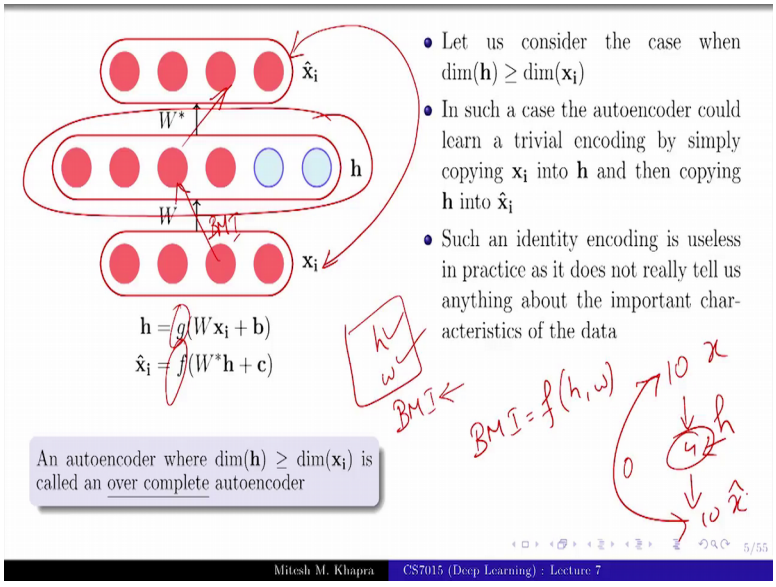
reconstruct the input that is very, very trivial if you give me more storage and what I originally needed, then definitely I can easily reconstructed right.

(Refer Slide Time: 07:58)



So, this looks very trivial and this is what it could do right just copy the input to the first the n bits.

(Refer Slide Time: 08:12)



So, this was n and this was d and we are looking at the case where d is greater than n. So, it will just copy the input to the first n bits and then just take it back to the output just as i

said in the case of you have 10 bits 16 bits and then again 10 bits it is very trivial to do this.

So, such an identity encoding is useless because you are just not running any important characteristics of the data, your h is almost the same as x i it also has all the useless information that x i had. In fact, it has slightly more because it has these blank units also, but this is not really useful right why would you want to actually learn such a hidden representation right. So, it is not clear why would you want to do that so we will take a look at it we will come back to why this is important.

So, such an auto encoder is known as an over complete auto encoder because it has the hidden representation has morenumber of neurons as compared to the original input, now let us look at a case where this would actually be important right. So, this is a very rough intuition for why you would want an over complete auto encoder right.

So, let us consider the case where you have as input 1 of the features that you are looking as BMI. So, suppose you are trying to find out whether the person is likely to get a certain disease or not right. So, whether he would have a heart attack or whether he would have a diabetes, would have diabetes and so on. And you are looking at various parameters or various medical parameters of that person and one of them could be height one of them could be weight and one of them could be BMI.

Now, for whatever reason you have not computed the height and weight and you have only looked at the BMI. So, now, what has happened in your input and all of you know that BMI is actually body mass index which is a function of the height and the weight.

So, now, what has happened is that in your original input there was already this compact the your feature space is already compact, because you would actually look at you should have actually looked at both the features height and weight, but for some reason you have only computed BMI and you could think of various some other correlated features, which are functions of many other features, but you do not look at all those features and just this final function of those features right.

So, now in that case if suppose your prediction is that this person has or has a high likelihood of being of high likelihood of having diabetes at some point in his life. Then

you would want to know whether it was the height, or whether it was the weight which was responsible for this.

So, in your original input your features are actually entangled and you would like to disentangle them right. So, you would want to go from this smaller feature space to a larger feature space where some of these entangled features get is disentangled. So, in those cases we reach an over complete auto encoder; however, the problem still remains that there is no reason why the machine should actually learn to disentangle these features it could still just simply copy the BMI here and then copy it back here right.

So, that is why when you are dealing with over complete auto encoders you will have to do something special to prevent this kind of identity encoding. So, as you just take the input and copy it to the hidden layer, and then copy it back to the output. So, we will look at what kind of special treatment you need to do to prevent these kind of identity representations.

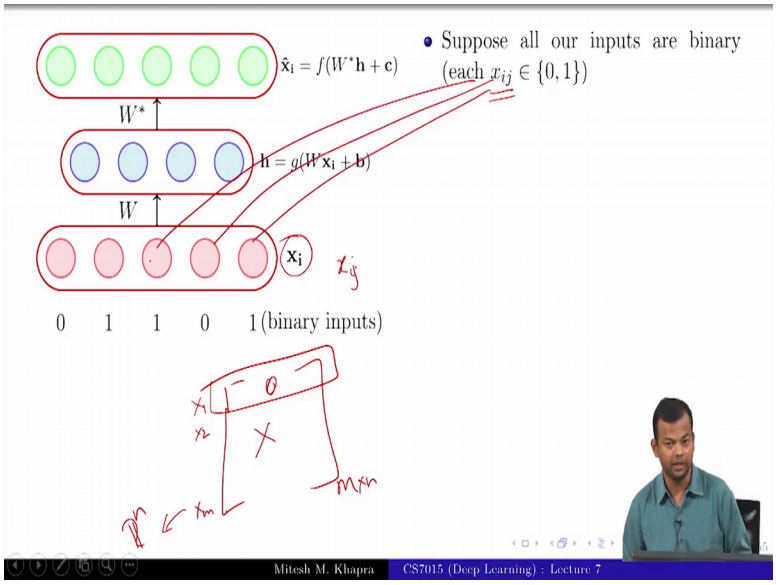
(Refer Slide Time: 11:24)

Here is the road ahead. So, first we will talk about the choice of f x i g x i right. So, we did not say anything about what these functions f and g have to be. So, we will talk about those and then we will talk about the loss function. So, I have just told you so far that we will train this model in a way that x is very close to x i hat right. And I have argued that if we are able to actually achieve that that x i hat is the same as x i in which case

presumably, presumably the loss would be 0. That means, our hidden representation has captured all the important characteristics of the original data.

Same as in the analogy of 10 bits to 4 bits to again 10 bits right. If I am able to reconstruct this without any error that means, loss is 0 then these 4 bits or the hidden representation of my original x was actually able to capture everything that was important in x. So, that it can reconstruct x again as x hat without losing any information right. So, that is the loss function that we would want now what is the actual mathematical formulation for this loss function that is what we will see next.

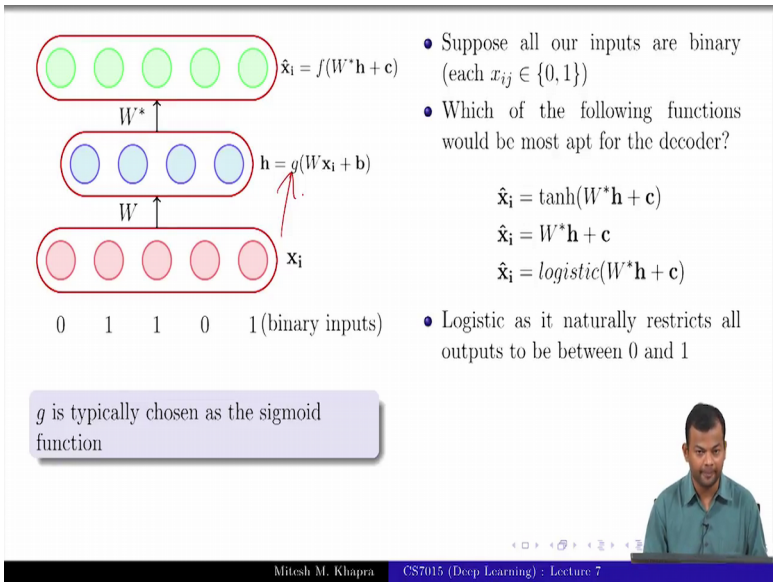
(Refer Slide Time: 12:35)



So, first let us start with the choice of f, f and g. So, we will consider 2 case 2 cases 1 case when your inputs are binary and the second case when your inputs are actually real numbers right. So, the first we will look at the binary case. So, now, just some notation clarification. So, remember our original data was this matrix x which was m cross n. That means, you had x 1 x 2 up to x m and each of these was R n right.

So, now when I am referring to the entire row or entire data instance I will use bold x i, as I have circled here. And I want to refer to 1 of the elements of this guy, then I will use this notation x i j same as what I have written here. So, what I am saying is that each of these x i js actually is a binary variable.

(Refer Slide Time: 13:25)



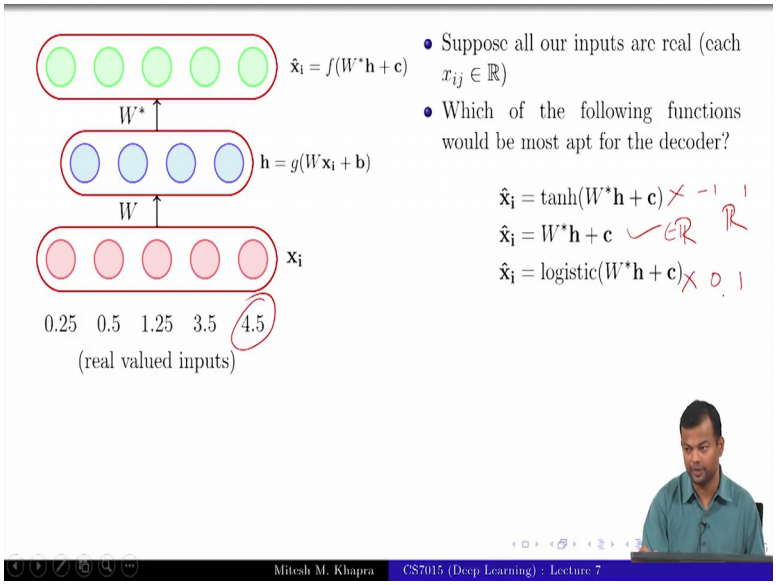
Now, which of the following functions would be most appropriate for the decoder? So, remember was the input was binary. That means, your output also has to be binary you do not want to produce numbers arbitrarily large belonging to any or want do not want to produce any real number you want to produce numbers which lie between 0 to 1. So, in such a case what would be an appropriate loss function or sorry what would be an appropriate function for the decoder. So, remember I am asking you what would f be.

So, I am giving you 3 choices it should be tan h or just a linear decoder or a logistic function. Which of these would be most appropriate logistic why would that be because it will make sure that your outputs are between 0 to 1, tan h would give you outputs between minus 1 to 1. But you do not want that because your inputs were between 0 to 1. So, when you are reconstructing; obviously, you want outputs between 0 to 1, you do not 1 minus 1 to 1. And linear of course, can give you any real number which is not what you want right.

So, if you produce any arbitrary real number like hundred and so on, your loss is going to be very high because your inputs were just 0 to 1 and you are producing these arbitrary real numbers which are very different from what your input was. So, in this case the logistic function makes the most appropriate choice. And g is typically that means, the encoded function is typically again chosen as a sigmoid function. So, it could either be

the logistic function or the tan h function right. So, the there is you could choose any of these as the encoder function fine.

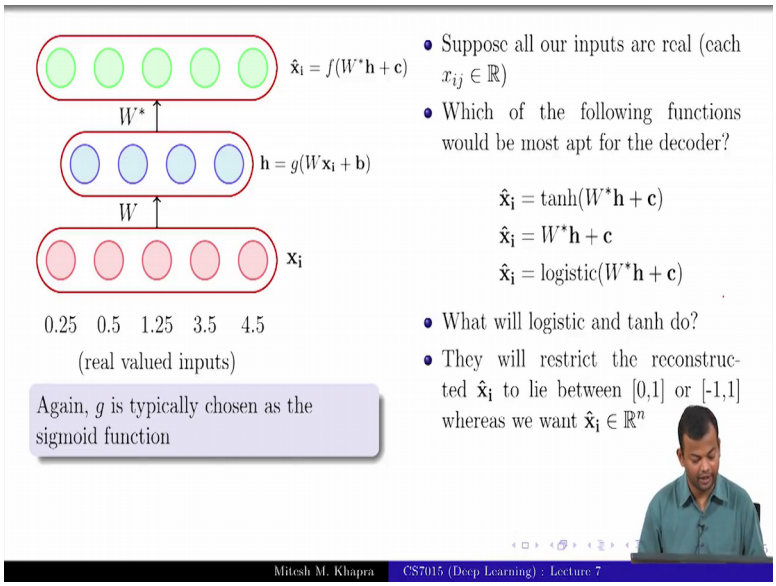
(Refer Slide Time: 15:01)



Now, let us consider the other case where your inputs are real valued. That means, when you reconstruct something you should again produce real values. That means, your function f should take whatever is the input given to it and map it to some real numbers right. So, that is what we want from this function f earlier in the binary case we wanted it to map it to binary numbers right. So, that is the difference that we have now.

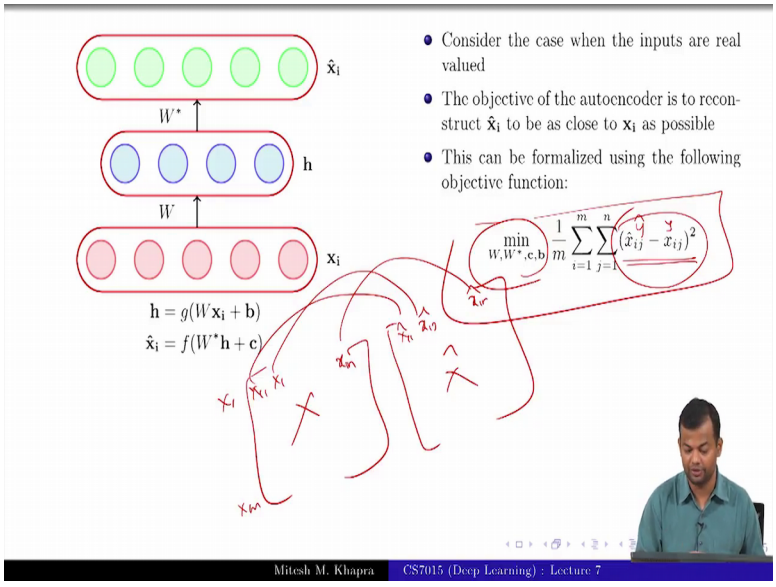
So, in this case which of the following would be appropriate? The second one right because tan h does not make sense because it will just produce minus 1 to 1, but you want to produce any possible real number because some of these are actually higher than 1 greater than 1. Linear would be fine because it will produce any real number logistic is again not fine because it will produce numbers between 0 to 1.

(Refer Slide Time: 15:51)



So, the logistic and tan h as I said would clamp the output to certain ranges. So, that is not appropriate hence you should choose the linear function and again in this case also g is typically chosen as the sigmoid function fine. So, the next thing that we look at is the choice of the loss function.

(Refer Slide Time: 16:10)



And again we will consider both the cases, where a case the first case is the inputs are real valued and the second case is when the inputs are binary.

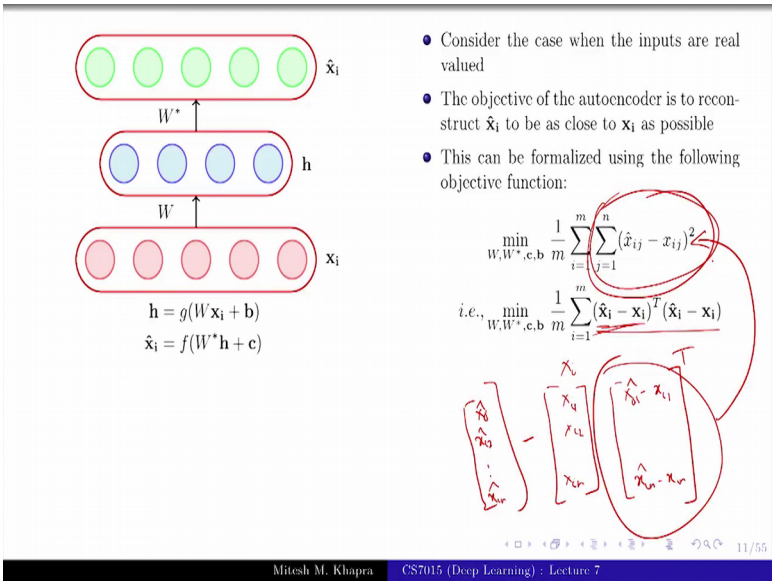
So, let us look at the real case first. Now here the objective of the auto encoder is to reconstruct x i hat to be as close to x i as possible. Now we have actually seen this before, so something similar before when we were talking about regression. So, now, you want to produce real valued outputs and they should match your real valued inputs. So, what is an appropriate loss function that you can choose the squared error loss function right.

So, what does this actually capture? It says that for all my input data x 1 to x m. For each of these dimensions x 1 to up to x 1 n right. I want to make sure that my original input, I will have a similar x hat reconstructed where I will have x 1 one hat x 1 2 hat and x 1 n hat right.

So, I want to make sure that each of these pairs of variables are actually similar. And I can capture that by ensuring that the squared error loss between the i jth entry in my output is the same as this or sorry. Rather I could capture the squared error loss between the i jth entry in the output and the input ok. That is what this function is trying to capture straightforward similar things we have seen while we were doing regression.

Except that there we had y hat and y, but here we are just trying to reconstruct the input. So, there is no y here we just have the x. And the parameters of the objective function are of course, all the variables or all the parameters that we have in a network, which is W W star c and b.

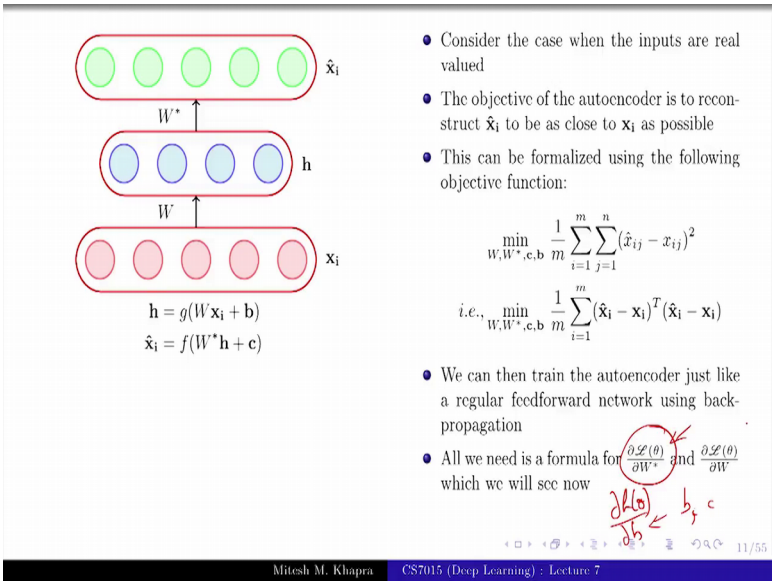
(Refer Slide Time: 17:56)



And the matrix or the vector way of writing this is the following. So, we have x i. So, what I am looking at here is I have gotten rid of this summation and I am just written it in vector form. So, let me just explain what this means. So, this is what x i would look like right. So, this would be x i 1 x i 2 up to x i n this is the vector. And then you have the x i hat vector which is going to be x i 1 hat x i 2 hat up to x i n hat right.

So, taking the difference between these two vectors that is what this term is. So, what you will get is essentially x i 1 hat minus x i 1 up to x i n hat minus x i n right. And then you are taking the dot product of this vector with itself which will essentially give you this summation right. So, the dot product of this vector with itself is actually going to be this summation, it is going to be the sum of the squares of the elements of this vector and that is exactly what we wanted.

(Refer Slide Time: 19:06)

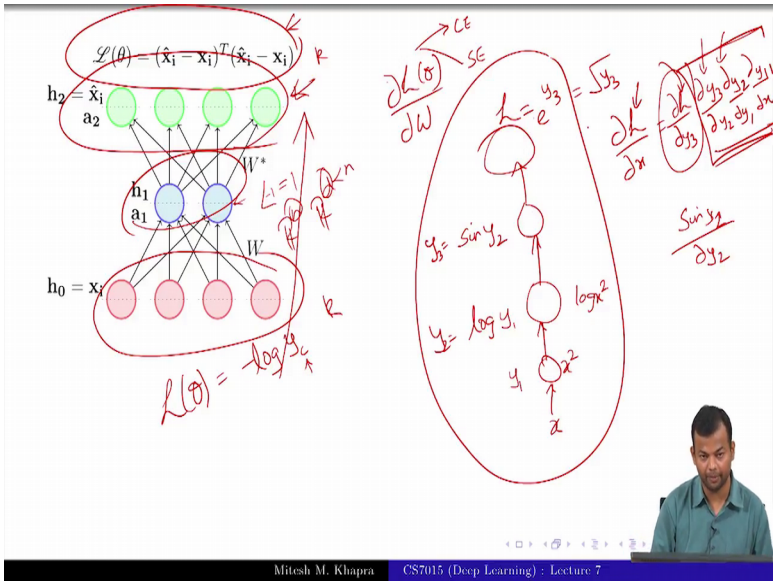


So, this is a more compact vectorial way of writing the same thing. And now we can just train the auto encoder by treating it as a regular feed forward neural network this is just a like any other feed forward neural network you have find the loss function.

And you can just use back propagation to treatment right, but and in this case all we will need is a formula for the gradient of the loss function with respect to with your parameters which are W and W star, I have again ignore the biases and the bias is here b and c. So, we will also need dou l theta by dou b and dou l theta by dou c right. So, these

two gradients also you will need, but these are generally the easier ones to handle if you know how to compute this the b and c are very easy.

(Refer Slide Time: 19:48)



So, let us look at this now what we need for back propagation as I said we will need this gradient right. All these four gradients but let us focus on 1 of these. Now we have already done back propagation and we have looked at arbitrary neural feed forward neural networks here right. We did not have we just said that there are L hidden layers and in this case L is equal to 1 right, or other we had said there was l minus 1 hidden layers and the lth layer was the output right.

So, in this case l minus 1 is equal to 1; that means, there is just 1 hidden layer. So, it does not matter we had actually derived it for the general case when l is equal when the number of hidden layers is l minus 1 and here we just have 1 eight n layer. So, it is much more simpler than what we had learnt. And even for the number of neurons in the each of these layers we are just assumed general that it could be R n. And in this case we would have some R d, which is less than n or it could even be greater than n right.

So but it does not matter because whatever algorithm we had or whatever equations we had derived for back propagation. They did not care about what this n or d was we had just derive it in general terms right, and the same for the output layer right.

We did not assume any number of inputs any number of neurons in the output layer we again said that it has some k neurons, but there the cache is in the earlier case when we had derived back propagation. We were dealing with classification and we had these k classes that we want to predict at the output.

And in which case our loss function was actually the cross entropy or the negative log likelihood function right. Where we were trying to maximize the probability of the correct class out of the k given classes, but here our loss function is slightly different it is actually this squared error loss between the input and the output.

So, now given this difference in the loss function does it mean that everything that we learn in the previous lecture on back propagation we just have to throw it all away because now there is a new loss function? That means, my gradients are going to be very different from what I had derived for the back propagation loss, where I was looking at the cross entropy loss as compared to the squared error loss. So, does it mean that, I will have to throw away all the hard work that we had done in that course in that lecture or can we reuse something from them we can reuse right.

So, let us look at what we can reuse and I will just give you an intuitive explanation for that. So, you can think of this as a composite function right. And you are taking your input passing it through a lot of functions and then arriving at the output, and then your loss function is actually a function of the output itself.

So, what we have is something like this right we have a situation like this. That you had an input x you computed some function of it say x square right. So, I will call this as y 1 then you computed some other function of it say y 1 say log of y 1 right. So, they this was log of y 1. So, in effect it is actually log of x square because y 1 is equal to x square and then some other function and then finally, you had the output. So, you had this other function which was sign of I am calling this y 2. So, say this was sign of y 2 and finally, you had this function which was e raise to y 3.

So, you have a very complex composite function of your original input right and this is your final output function that you are considering which is e raise to y 3. Now the way you would do this is if you want to take the gradient of d l with respect to your input d x right. In that case what would you do is you just apply the chain rule you will right.

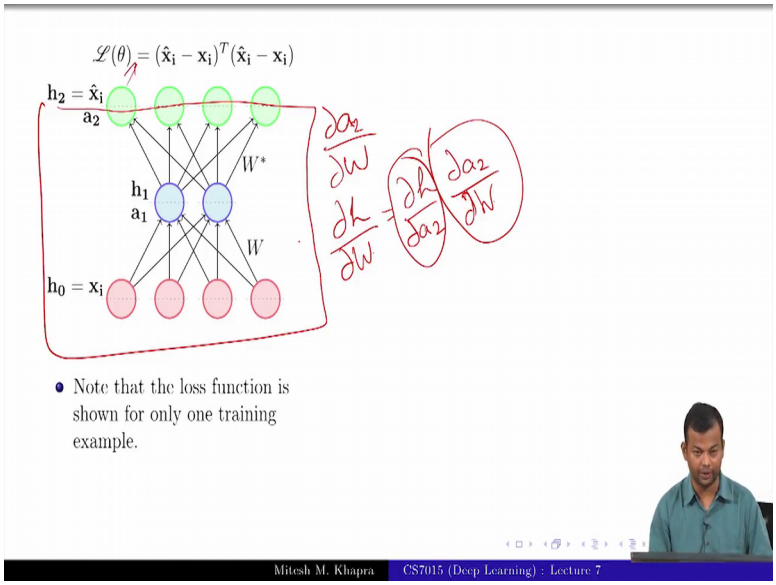
It as dou l by dou y 3 dou y 3 by dou y 2 dou y 2 by dou y 1 and then dou y 1 by dou x right. And this is something very similar that we are done in the back propagation lecture we had constructed this chain and then we had attacked every element of this chain and derived how to deal with that right derived an neat expression for that.

Now, the question which I am asking you is that in that lecture we had assumed a certain l. And that l was actually cross entropy, but in this lecture I have actually changed the l, what I am saying is the l is actually equal to the squared error loss. Now does that mean that I have to throw away all this work that I had done no right?

So, even in this example if you look at it suppose I change this function from e raise to y 3 to say square root of y 3. So, I have just changed my l, but notice all of these other guys are going to remain in the same, because y 3 is still sign of y 2. So, that the derivative of y 3 with respect to y 2 is not going to change. Even though I have changed the output function the loss function everything else is going to be remain in the same right.

So; that means, all these portions I could just reuse from the time when I had computed for this chain. I just need to rework on this final expression and plug it in right. So, that is why all the work that we had done in the case of back propagation will not go to waste in particular everything that we had done.

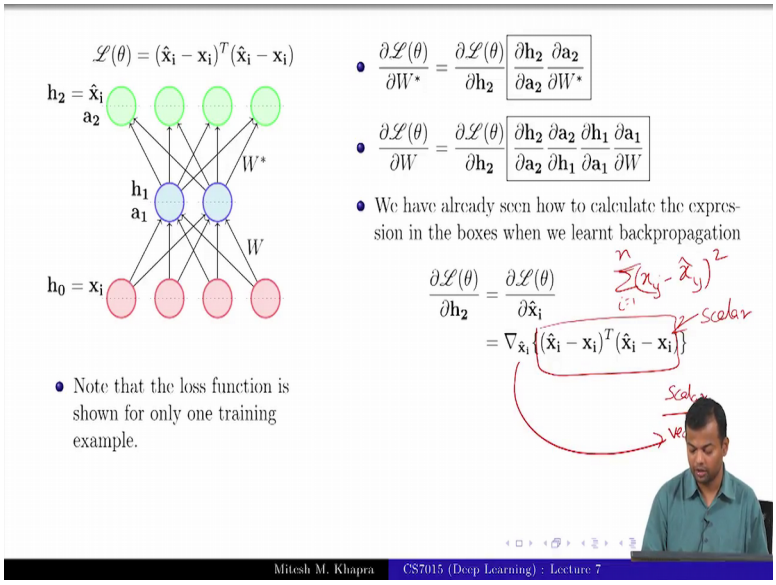
(Refer Slide Time: 24:45)



So, let me just go to the next slide. So, in particular everything that we had done for this portion of the network right which is actually dou a 2 all the way up to dou W right. So, if ok, so let me write it like this I want dou l by dou W. So, I can write it compactly as dou l by dou a 2 and then dou a 2 by dou W right.

So, this portion is not going to change because I am not change any of the functions here, I have just assumed sigmoid or logistic or the same kind of network. The only thing I have changed is something at the output layer. So, I will just need to recomputed this and the rest of it can be reused right. So, that is the intuition which I wanted to give you.

(Refer Slide Time: 25:27)



And that is exactly what is written on this slide. So, I am written it as dou l theta by dou W star that is the first gradient I have interested in. And I could write it as dou l theta by dou h 2 dou h 2 and dou a 2 by W star right.

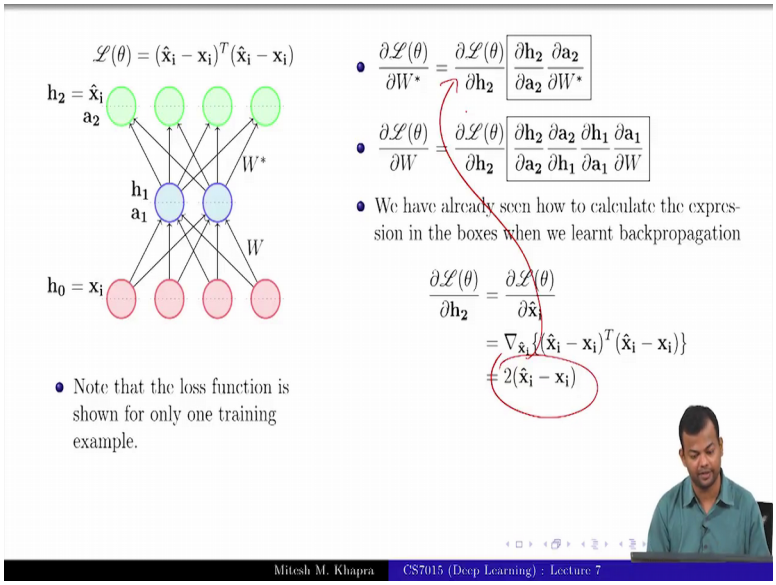
Now, this portion as I was trying to say is something that we have already seen in the back propagation lecture, and nothing has changed in the network in that part. So, you can just reuse it as it is and this portion is something that we need to recomputed right that is the only thing that we need to recomputed and plug it into our back propagation code or the algorithm, which we had in the previous lecture. And similarly if you want to do dou l theta by dou W it is the same idea here that you could write it as the following chain. And this part of the chain you already know how to compute from the back propagation lecture.

All you need to do is change the loss function and just try to find the derivative of the loss function with respect to your output layer which is h 2. That is the final thing that you have changed just as in my toy example I had changed e raise to y 3 to square root of y 3 right. That is the similar change that I am trying to do here fine.

So, all we need do is dou l theta by dou h 2, but dou h 2 is the same as x i hat right because that is my output and my output I am calling it as x i hat. So, I need to take actually the derivative of this. So, I am just using the vector form here I could have also written it as this summation over i equal to 1 to n x i j minus x hat i j the 2 whole square right. I could have also written it as am I just writing it as the vector here in the vector form here right, but this quantity ultimately is going to be a scalar because it is a dot product between 2 vectors which is the scalar.

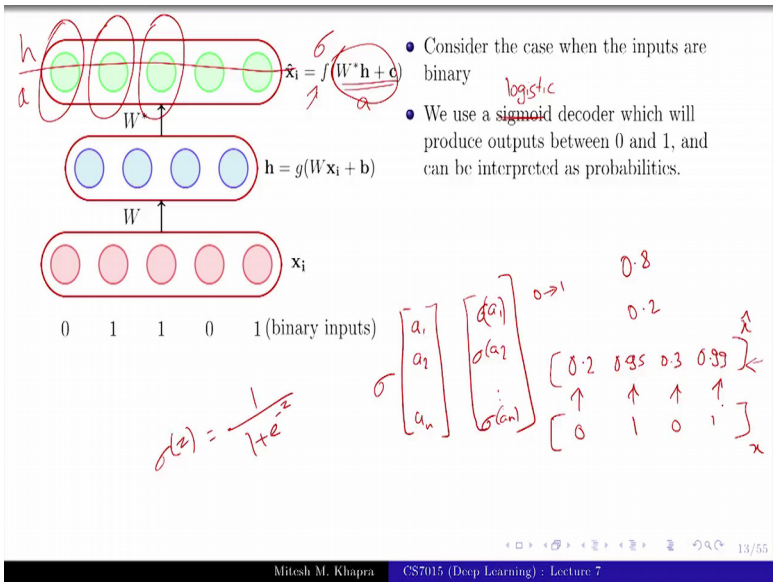
So, what I am doing here is taking the derivative of a scalar with respect to this vector. So, what is that derivative going to be? It is going to be a vector.

(Refer Slide Time: 27:20)



And I am just, so we have similar stuff in the past. So, you can actually easily work this out. So, this will actually turn out to be the following vector which is to times x i hat minus x i right. So, this is very simple I have just computed this and all I need to do is go back and change my back propagation code. And change this derivative of the loss function with respect to the output clear and the rest of the code I can just reuse it as it is. So, now similarly ah, so we have both of these ready.

(Refer Slide Time: 27:50)



Now, let us look at the other case when we have binary inputs ok. This is the most more this is something different that we will have to do here. So, we will now look at the second case where the inputs are binary. So, first we look at case when the inputs were real numbers, and hence your outputs also needed to be real numbers.

Now, we look at the case where inputs are binary and hence your outputs also need to be binary. Now here, so each of these guys is actually a sigmoid functions. So, it is in or rather if you look at the output you could divide into two parts. So, this is the pre activation and this is the activation right. So, your this is actually the pre activation and this is the activation right.

So, this activation is actually chosen as the sigmoid function or the actually the logistic function not the sigmoid function of course, logistic is the sigmoid function, but the logistic function which was 1 over 1 plus e raise to minus z right. So, logistic of z is equal to 1 over 1 plus e raise to minus z. And remember that this sigmoid function was element wise.

That means, this is a is a vector it has elements a 1 a 2 up to a n and then you know apply the sigmoid to it you get h, which is going to be sigmoid of a 1 sigmoid of a 2 and sigmoid of a n right. So, it is just the sigmoid applied to every element of the activation layer. That means, every element of this vector which have circled.

So, now in this case your outputs are going to be between 0 to 1 right, because your inputs were also between 0 to 1 and your sigmoid or the logistic function is going to give you clamped outputs between 0 to 1. So, since this is between 0 to 1 we could actually interpret it as probabilities right. So, we could say that whatever you are reconstructing is actually telling you that with 0.8.

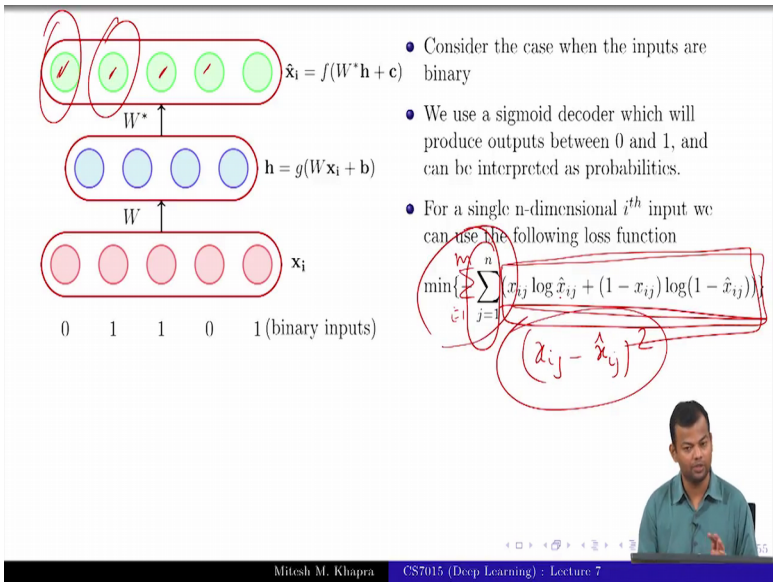
Suppose, the reconstruction value is 0.8, then you could think of it that with probability 0.8 it is telling you that the output should have been 1 right. And if it tells you that the output is 0.2 if the sigmoid gives an output as 0.2. Then you could think of it that with probably 0.2 the output was actually 0 or rather the input was 0 because an input is the same as the output right.

So, that is one way of interpreting it and this way of interpreting it why does it make sense. So, we will just look at that right, so before at if I do not give you this interpretation. And remember that the sigmoid is going to produce values between 0 to 1, but not necessarily 0 and 1 right it will try to be as close to 0 when the input is 0, but it could also produce 0.05 and so on. And when the input is point nine it could also produce something like 0.95.

So, at the output also you are going to get these vectors which are of which would look something like this right. And suppose you are input was 0 1 0 1 now can think of a suitable loss function for this yeah. So, again these are two vectors these are x hat and x. So, once again you could have just gone with the with a squared error loss right, you could have taken the squared error difference between these two and you could have been fine right.

So, that is definitely one way of going about it, but whenever we are looking at these binary inputs. And whenever this probabilistic interpretation is possible we tend to do something better which is look at the cross entropy loss instead of looking at the squared error loss. So, I am not saying that the square error loss is wrong in this case, but you could also use this cross entropy loss. And in practice for our binary inputs the cross entropy loss often works much better than squared error loss right.

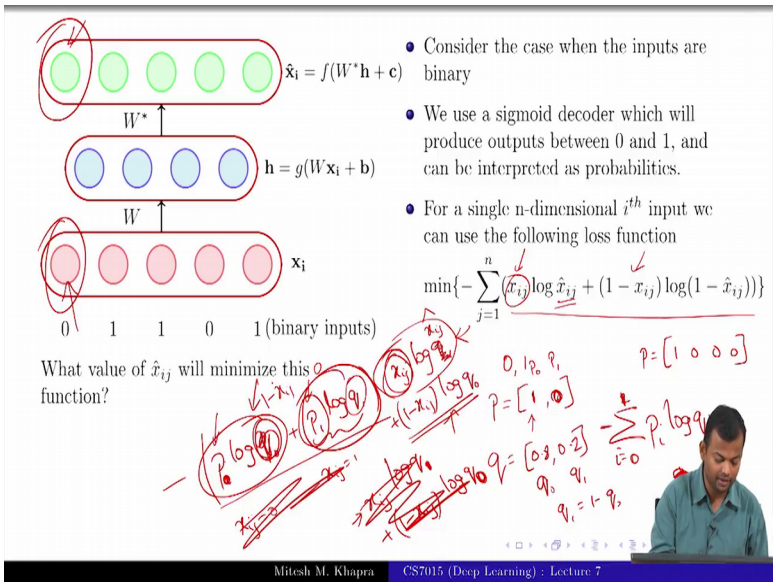
(Refer Slide Time: 31:31)



So, let us see what I mean by the cross entropy loss. So, remember that you have n outputs right. That is why this summation let us not worry too much about what is written inside for the time being I will explain that, but that is the I just want to explain the summation first. So, what you are saying is that for each of these green guys at the output you are going to make some loss. And you just want to some over that loss that is what we are trying to see.

Now, ideally you could have just written it as just done what you had done before and written this entire replace this entire box by this squared error loss. And that would have been just fine right of course, there should have also have been this summation i equal to 1 to m here, because you are going over all the m training instances and for each of the m training instances you are trying to minimize this loss. So, this 2 summations followed by this squared error loss would just have been fine.

(Refer Slide Time: 32:26)



, but instead of that I have this something special here. So, let us look at what this special quantity is. And now for that remember that I am trying to interpret each of these inputs as a binary random variable. I am saying that they can take values 0 or 1.

So, I can think of it that when I am given that this value is 0. I can write it as this deterministic probability distribution where I have p. And the probability mass is entirely concentrated out on this 0 value and my the probability mass on the value 1 is 0. This is something similar to what we had done earlier when we are given these labels suppose it was apple, orange, mango and banana. And the class label was given to us that this is an apple. Then we could still write it as the probability distribution where all the mass was concentrated on apple and everything else was here.

So, I am saying something similar here right. So, you could think of it that two possible values can occur here 1 and 0 and if I tell you this is 0 right. Then I am telling you that with probability 1 into it is 0 and with probability 0 it is 0. So, I still write it as a probability distribution now the same thing I can have at the output. So, for this unit when I am trying to reconstruct it and if I produce the output as 0.2 then I can or rather let us say 0.8 then I can say that with 0.8 probability I am predicting 0 and with 0.2 I am predicting a 1 right.

So, now I can think of this again as two probability distributions. And once I some have two probability distributions I know that cross entropy is the right or a better loss

function to look at right. And what is cross entropy actually in this case it would be given by summation i equal to 1 to 2 right or rather i equal to 0 to 1 because if the those are the values it can take p of i right into log of q i plus yeah. So, p of i into log of q i that is how I can write it.

So, let me just since there are only two terms I can just expand this summation right. So, I can write it as p i or rather p 0 log of q 0 plus of course, it is a minus sign here, this is a minus sign at the out p 1 log of q 1. I can just open up because there are only two terms. So, I can write it as this is that fine.

Now, also I know that there is this relation between p 0 and p 1 right. That p 0 is actually 1 minus p 1 yeah. Similarly, you have this relation between q 0 and q 1 that q 1 is equal to 1 minus q 0 because the sum is going to be 1 now let us look at this sum right. So, in the binary case this sum becomes interesting because. Now suppose your input x i j right which is the entity that I am looking at, suppose that was equal to 0. In which case all the probability mass would be concentrated on p 0 and p 1 would actually be equal to 0, which means the second term would display.

On the other hand if x i j is equal to 1 then the reverse situation what happen, that everything would be concentrated on p 1. That means, p 1 is equal to 1 and this guy would become 0 because p 0 is going to be 0 right. So, there is this another way of writing it that you could day that instead of x instead of writing p 0 and p 1 you could just write it as x i j into log q 0 plus 1 minus x i j into log of q 1.

So, now let us look at it again, so when x i j is 0 first which is the same which happened here just an (Refer Time: 36:20) in same thing right. Because, whenever s x i j is 0 p 0 is equal to sorry sorry it should have been q 1 and log q 0 sorry i made a mistake here. So, it have been x i j into log q. So, or rather let me just rewrite it.

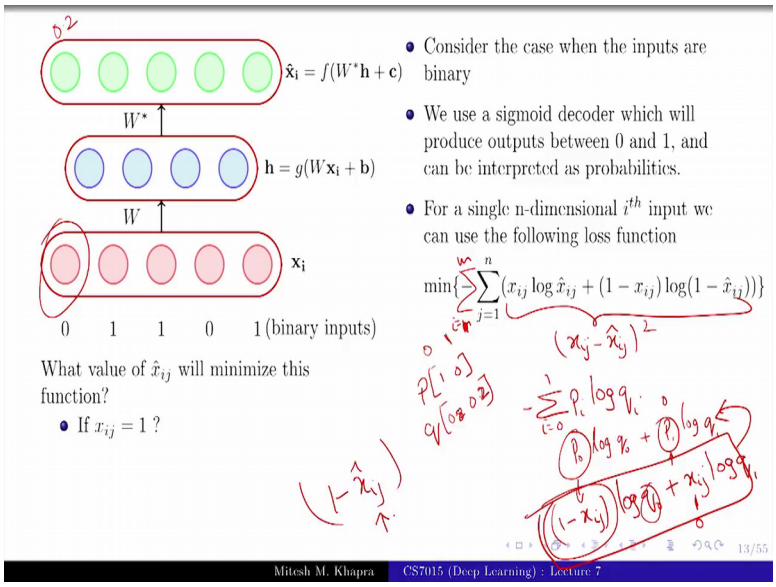
So, this is going to be actually I can write it as, I look at this term first. So, I can write it as x i j into log q 1 and then the second term I am going to write it as 1 minus x i j into log of q 0 right. And then I am going to simplify this further, but let see what is the consequence of this.

So, now whenever x i j is equal to 1 this term will remain and the second term will disappear and that is exactly what was happening in our original formula right. So, this is

just an equivalent way of writing your x i j is equal to 0 this term will disappear, but this term will remain; that means, log q 0 will remain this is exactly what was happening in our original formula right.

So, that is so now, I have given you why a I can replace p 0 and p 1 or rather p 1 and p 0 by x i j and 1 minus x i j. And now I can make a similar argument for x hat i j also. So, I can think of q 0 as whatever s predicted at the output right sorry I can treat q 1 as whatever is predicted out 1 output. So, whatever my sigmoid function predicts I can think of it as it is predicting the probability of getting a 1 right. So, it is just predicting the heads probability or the probability of getting 1. So, I can instead q i q 1 i can write it as x i j hat, and similarly instead of q 0 I can write as 1 minus x i hat i g right. So, did you get that so these become very messy.

(Refer Slide Time: 38:13)



So, let me just clean this up and I will just go over this again right.

So, what I was trying to tell you is that in the ideal case you could have just replaced this by the squared error loss, but since you are dealing with binary inputs you can do something better because you can interpret the outputs as probabilities. So, when you get a 0.2 here you can interpret it as it is telling you that the probability of this unit being 1 is 0.2, it is very less. And that is the same as saying that the probability of this unit being 0 is 1 right.

So, you can interpret this as a probability. Now if you think of it that way then you can say that at the input you are actually given a probability distribution. So, which tells you that in the first case your probability distribution looks like 1 0 right, because all the mass is focused on value 0 because your input is 0 at that case. And now suppose your output was 0.2 right and 0.2 is what you are treating as a probability of. So, this is the probability of 1 this is the probably of sorry this is the probability of 0 oops and this is the probability of 1.

So, if your output is predicting 0.2. That means, it is predicting 0.8 for 0 and 0.2 for 1. Now if you think of it this way then you can capture the loss function between these two guys using the cross entropy formula. Which is going to be summation i equal to 0 to 1 p i log q i is that fine ? And now I just said that the since there are only two terms I can just write it as p 0 log q 0 plus p 1 log q 1.

Then I focused on this relation between p 0 p 1 and your input. So, whenever your input is 0 your p 0 is going to be 1. So, then I can just replace p 0 by 1 minus my input right. So, if the input is 0 then this guy is going to be 1 and that is exactly what this expression is also going to be.

So, I can write it as 1 minus x j log q 0 and similarly for this second guy. Whenever input is 1 this guy is going to be 1, whenever my input is whenever my input is 1 this p 1 is going to be 1 whenever my input is 0 this p 1 is going to be 0. So, I can just replace p 1 by log by x i j and now you can see that this expression evaluates to the same as this expression right, you can substitute value of x i j 0 or 1 you will get the corresponding p 0 p 1 which would be 1 or 0 depending on what your input was and these two expressions will evaluate to the same thing.

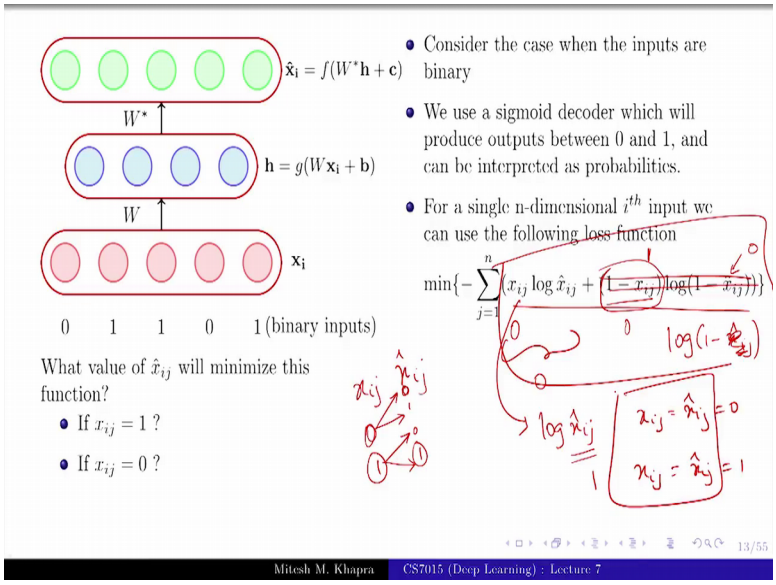
So, just as I replaced the ps by x is x i js, I can similarly replace by a the qs by x i j hats right because once again q 0 is nothing, but 1 minus whatever my output was predicted because whatever is predicted I am treating as the probability of getting a 1. So, 1 minus that is going to be the probability of getting a 0. So, that is what q 0 is and similarly q 1 I can replace by x hat i j. And so that is exactly what I have done in this expression here.

So, now this expression every term in these n terms captures the cross entropy for that particular random variable right. So, this is the original distribution p for this random variable. This is the predicted distribution q for this random variable, and I have just told

you that this the cross entropy between these two distribution can be written in this simple form as the function of x i j and x hat i j.

So, this is the standard thing to do when you are dealing with Bernoulli random variables. So, you can go back and read up a bit about it ah, but for now I guess with this explanation it should (Refer Time: 42:15) to know why this expression is used. And remember that I am not telling you that this squared error function was bad. I am just telling you that instead of the squared error function cross entropy loss function works better when you are dealing with binary inputs fine.

(Refer Slide Time: 42:34)



So, with that let us pursuit and the another we have looking at it is the following you can now look at this expression. And tell me when is this expression going to be minimized. So, we have x i j and x hat i j you can see that this expression will be minimized only when x i j or rather x hat i j is equal to x i j right. So, now, x i j could take value 0 or 1 and now x hat i j could take 0 1 or 0 1.

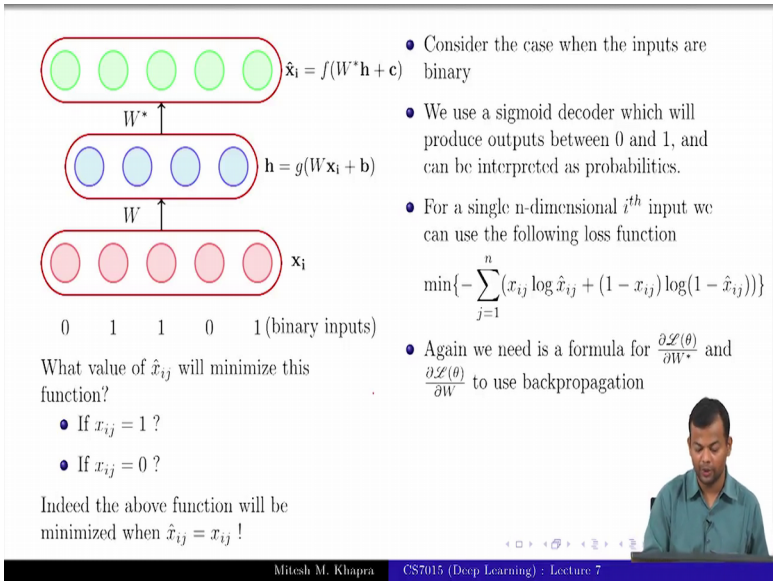
So, you can see that for these two combinations the value is going to be minimized only when x hat i j is actually equal to x i j. That means, if x hat if x i j was 0 then x hat i j should also be 0. And similarly in this case also if x i j was 1. Then the expression will be minimized only when x hat i j is equal to 1. So, let us see this so suppose x i j was 0. That means, this term is going to go to 0, but this term is going to remain and now if you are x hat i j was not equal to 0.

Then you will get some log of 1 minus x hat i j as the loss right, but if x hat i j was also 0 then you would get log of 1 which is 0. So, this whole expression would then evaluate to 0 which is the minimum possible value for this expression right.

So; that means, if x i j is 0 then this expression will be minimized only when x hat i j is also equal to 0. Similarly, if x hat i j sorry if x i j is 1 then this 1 minus 1 will give you 0. So, this term is going to disappear, but this term will remain. So, this will just be log of x hat i j because x i j is equal to 1.

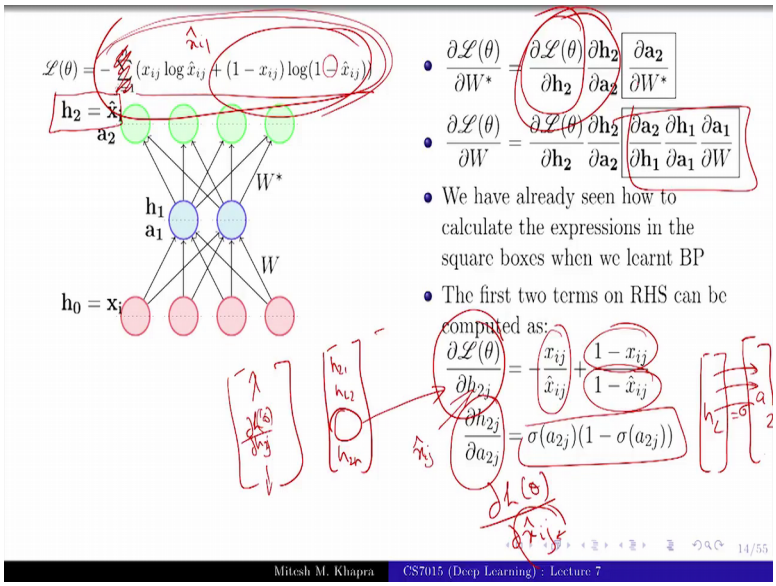
Now, if x hat i j is also equal to 1 then this is become log of 1 which is 0. That means, again this expression will attain it is minimum value when x hat i j is equal to x i j is equal to 1 right. So, this expression now attain it is minimum value in two cases when x i j is equal to x hat i j is equal to 0 or when x i j is equal to x hat i j is equal to 1. So, compactly I can say that this expression will attain it is minimum when x i j is equal to x hat i j, that is why this loss function makes sense.

(Refer Slide Time: 44:54)



Now, again we have this problem that we want to use back propagation to train this network. And once again for back propagation we will need the following gradients the gradients of the loss function with respect to W and W star. This is what we are going to need and I am going to make this same argument again that whatever hard work you had done in the back propagation lecture you can just reuse all of it.

(Refer Slide Time: 45:19)



because the only thing your changing is this final loss function. So, you just need to compute the gradients with respect to this loss function and everything else is going to remain the same right.

So, that is exactly what I am going to do on this slide. So, whatever is in the boxes here these two boxes that is something that you have already computed. And now what I am going to compute is the stuff which is outside the boxes, so let us look at that. So, I am interested in computing this dou l theta by dou h 2 this is the derivative of a scalar quantity with respect to a vector, say it is going to be a vector. And I am going to follow our usual recipe which is h 2 is actually equal to h 2 1 h 2 2 up to h 2 n.

So, I am going to consider any of these guys which is h 2 j, I am going to compute the derivative of the loss function with respect to this one entry and since I have that I am going to construct the entire gradient right. So, now, I will have this dou l theta by dou h 2 j right and once I have that expression I am just going to generalize it to all the other entries in this vector.

So, let us look at that expression first. So, now, if you look at this actually it does not have an h 2 j right, but we know h 2 j is the same as x hat j or rather x hat i j right for the ith input it is going to be x hat i j. Because h 2 is equal to x hat i ok, you can just see that the top left corner of the slides say x 2 is equal to x hat i. So, this is nothing dou l theta by dou x hat i j.

So, now I want to take the derivative of this quantity with respect to 1 particular x i j, and remember that this quantity has the sum which is indexed over j. So, j goes from 1 to n I am looking at 1 particular j. So that means, if I expand this sum of all the js possible the derivative with respect to all, but 1 is going to be 0, because they do not depend on this particular j. So, if I am looking at z equal to 3 then the term which has x hat i 1 is going to the derivative of that term is going to be 0.

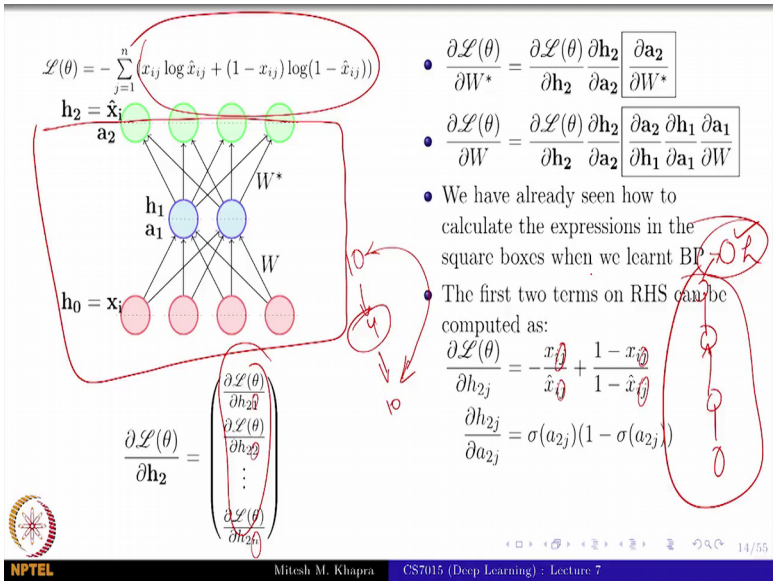
So, for all these terms in the expression only that term where a j is equal to the j which I am considering is going to remain k. So that means, only one term in the summation would remain and for that one term, so let me just rid of the summation right. So that means, only one term in the summation would remain, I am trying to find the derivative of this quality x which has a lot x i js with respect to x i j.

So, now this is of the form a log x, so the derivative would a over x right. So, that is exactly what I have written here and similarly for the second guy this is 1 minus a into log of 1 minus x. So, the derivative is going to be 1 minus a over 1 minus x and of course, there is this minus sign here which will then get adjusted appropriately right. So, that is how this expression has been completely that is very straight forward and now as you need the derivative of h 2 j with respect to a 2 j.

So, remember that h 2 is equal to sigmoid of a 2 which means it is just an element wise sigmoid right. So, I just need to compute the derivative of the jth entry of h 2 with respect to the jth entry of a 2 all the other derivatives are going to be 0 because they do not depend on that particular entry of a 2. So, now, that is just going to be sigmoid of a 2 into 1 minus sigmoid of a 2 right.

So, I have computed these two quantities I can just plug it then back into back propagation code. The rest of the code is going to remain the same and I have the gradients ready with me right.

(Refer Slide Time: 49:14)



And as I said once I have this one guy I can just extend it I can just generalize it. So, I just had these js here right for h 2 j, so I can just replace the j by 1 2 up to n and I will get the same expression.

So, that is the end of module 1 where we introduced auto encoders. What we showed is that they are actually just like any other feed forward neural network accept that they have this special objective. That they want to reconstruct the input and the reason they want to reconstruct the input is they about to first create a bottle neck which is this h hidden representation. And then try to reconstruct from there and just as I gave you that compression analogy that you have this 10 bits you want to compress it to 4 bits and then reconstruct the entire input again.

So, this will happen only if these 4 bits capture everything that is required or the most important characteristics of your original input right. And then we could have a loss function which tries to capture the difference between my original input and my reconstructed input.

Now, we argued that this loss function will be dependent on the nature of your input. So, for the real inputs it was straight forward we just said that we can use the squared error loss function for the binary inputs we actually did something special. We said that we can actually use the cross entropy, and then we had this funny way of writing the cross entropy which was this x i into log of x hat and 1 minus x i into log of 1 minus x hat.

And just gave you some intuition that that is the same as writing p log a pi log or rather p 0 log q 0 plus p 1 log q 1 write and the I just gave you some explanation for doing that.

You can go back and check on how do you write the cross entropy for Bernoulli random variables and you will see that this expression makes sense. And once we had this expression computing the gradients was easy. So, the other thing that we relied on is that in the back propagation lecture we had taken care of everything up to this point and in this lecture we have actually changed the loss function.

So, 1 loss function was the sum of squared, squared loss errors and the other loss function was the sum of sum of cross entropies whereas, in the back propagation lecture we had only dealt with cross entropy by the case that we made is that sense you have this chain. I know all you have done is change the last function in the chain right, you have changed this l function all the other functions you have not changed.

You can just reuse the computations from these or you can just use the code that you had written for these in the back propagation assignment. And you just need to change this last guy to adjust for the change in the output layer or the change in the loss layer. So, with that we will end the introduction to auto encoders, there we have done we have actually covered how to train an auto encoder using back propagation. So, that is where we will end.