

Deep Learning for Visual Computing
Prof. Debdoot Sheet
Department of Electrical Engineering
Indian Institute of Technology, Kharagpur

Lecture – 04
Neural Networks for Visual Computing

So, Good morning and we will be starting with today's lecture, and that is on neural networks for visual computing. So what essentially this would cover down as while tell the previous two lectures and the one lab which we have covered we have learned how to operate on images, some basic operations using the classical way. So, as you start with any kind of a visual computing task, and say that my computing tasks over here is the recognition tasks.

So, you would be starting down with the taking an image, then expressing an image in terms of its features and that is basically to compress down all the information which you have in that big corpus of pixel space available to you. Now from that when we eventually go down as you have seen that there are features which you have extracted out.

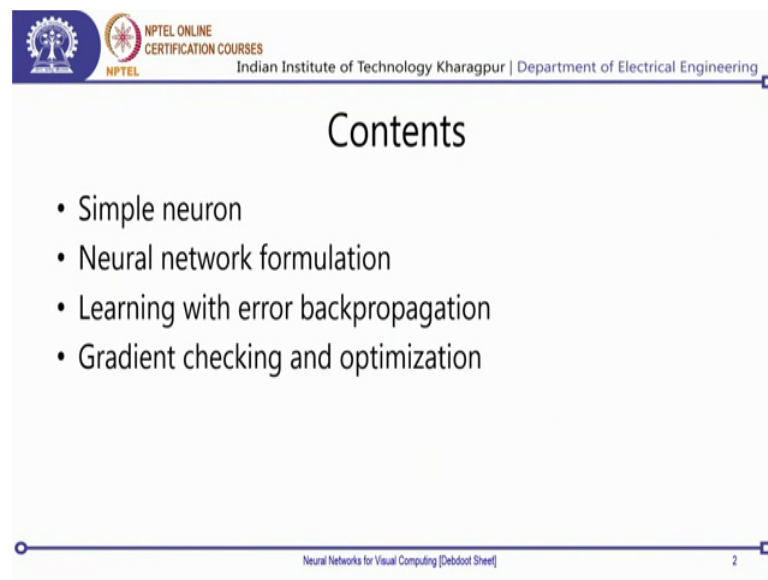
The next question which comes down to you is can be classified and you know in order to classify them the simple definition which you have learnt on some of the related other subjects as well as what we had defined in the first few lectures was that. You need to be able to relate certain features to a certain kind of an object category and this kind of a relationship is what is called as a classification problem.

Now in order to make it even simpler so, what it would essentially mean is that if I have an image represented in terms of some different parameter say $x_1 \times x_2 \times x_3 \times x_4$, and these are all may be scalar parameters. Now if I arrange these scalar parameters into sort of a matrix that is what we would call down as a vector or in the standard parlance of our definitions we would also be calling this as a feature vector.

Now once you have that feature vector given to you how do I associate a feature vector to one single categorical label, and that single categorical label may be whether it is a cat, dog, horse or rose, flower, bus any anything of that sort. And this whole associating a whole set of these attributes collected from a feature to something of a categorical

variable is what is called as classification, and where we stand down is that neural networks as we know, when they came down in the early they all started down from the perspective of visual computing itself, and now from that perspective here is where we start down so.

(Refer Slide Time: 02:34)



What today's lecture will contain down as in simple form called as we will be starting down with a simple neuron model, and from there we will go down to the neural network formulation, and then we go down to something called as a learning with the read back provision and gradient checking and optimizations. Now what essentially this means is that what we would do is we would define what a neuron is so, as in a neural network you would always have a neuron.

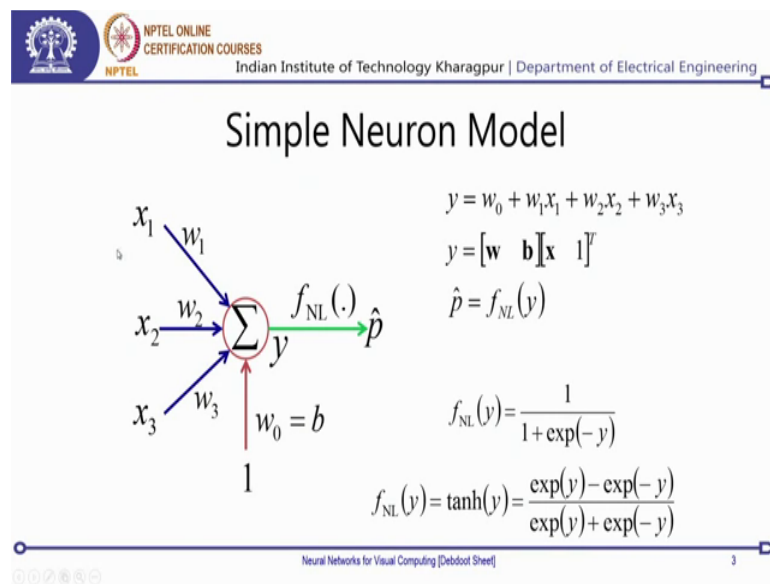
So, then what do you call something by neuron what is its mathematical definition, how do we learn to relate down something from feature space which are certain numbers to another number which is a categorical variable which you get down. So, from a simple neuron we will enter into what is called as a neural network or in some simple terms if I have some sort of a network like behavior which is one parameter connected onto multiple parameters or multiple parameters connected to one parameter, and it can even go down by a successive stages of parametric connections which goes down. And finally, is what enters into what we call as the learning.

And in this learning phase we have learning with aided back propagation, and what this does is that we define something as a neural network and then say that this neural network is able to classify images and as we go down over there. Now one important fact which you will have to take care over here is that as we are learning over here.

then that would mean that somehow with experience going by that definition somehow with experience which we are gaining we will be able to do that task much better so; that means, necessarily that as we show it more number of samples of features, and which are associated number of categorical variables over there, then this whole network over here which I call as a neural network would be able to really associate any unknown feature sample coming down to it to a categorical variable and classifier.

And in order to achieve this learning you will have to go down through the way of gradient checking and optimizations, and then how what rule display. So, as we go on eventually you will get down to no more in details about them.

(Refer Slide Time: 04:47)



So, a simple neuron model is something which is defined like this. So, we have the mathematical equation, but let us get down into what it looks like. So, say I have three different variables x_1 , x_2 and x_3 , and these can be three features, so features of an image as in say brightness of the image over here is the contrast of the image.

And say over here is the average entropy of the image. So, we can have three different features over there for one single image given down. Now once we have these 3 so, each of them is a scalar value as you see over here because brightness of the image is scalar, now then the contrast of the image x_2 that is also a scalar well and entropy of the image x_3 is also scalar.

And now together what we can do is we can associate a categorical variable say \hat{p} over here which is called as a predicted class or predicted a categorical category label variable over there. Now we can associate each of these scalars with certain vector, and then we can sort of sum them up so, x_1 will be multiplied by a weight w_1 , x_2 will be multiplied by a weight w_2 x_3 will be multiplied by a weight w_3 .

And then on top of it what we add is a certain scalar value with the dc offset which is also called as a bias. So, essentially and then put it into a summation such that I get this term y over here, and then my output over here. So, the form of y is something which is written down as w_{naught} plus $w_1 x_1$ plus $w_2 x_2$ plus $w_3 x_3$. So, if I have some n number of variables over here, and I can have n number of bits associated with each of them and then this will be a $n + 1$ term summation which comes down over here.

Now from there as we see we can also write this kind of a form in terms of a matrix representation and that sort of a matrix representation is what is given down over here. So, what w is basically it is a matrix of collection of all of these bits and b is basically. So, over here, as in since we have only one predictor variable this is supposed to be a scalar value bias, and that is equal to w_{naught} itself, and x is another vector of all the scalars which are arranged in terms of a matrix over there.

So, this is a matrix form of representation now once I have that one what I can do is I can relate this y to my predicted class level in terms of some sort of a non-linear function, and we will say that is an f_{NL} . Now this f_{NL} can have two different forms some common forms are something like this. The first one which is also called as a sigmoid form of non-linearity, the second one is what is called as the tan hyperbolic form of non-linearity, now what it essentially does is that this my x s over here I do not have any sort of a control over my x so, these can be anything from minus infinity to plus infinity for the purpose of simplicity we keep down the fact that let these be real valued numbers and

not complex valued numbers, that does assist us in getting down a mathematical tractability to the whole problem.

Now, all of these w 's for me they can also be some weight some scalar weights which can vary in an open range, and open range in a sense that they can be from minus infinity to plus infinity. Now that I have these also open ranged then what I get down from this y over here can be an open range problem. So, that can be anything from minus infinity to plus infinity, but this \hat{p} over here if that has to be a categorical label.

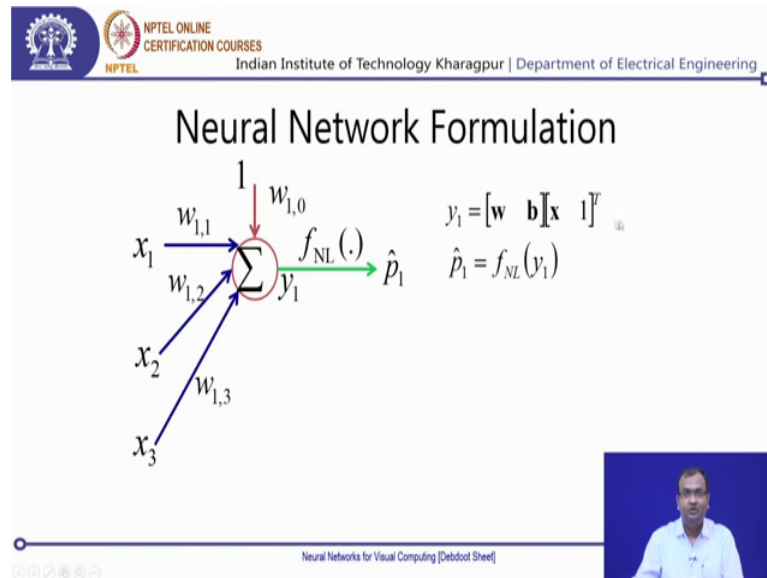
So, maybe a true false question or like I want to classify it into two classes, whether this is a ball or this is not a ball; it says 0 and 1 problem over here, now if this y comes to me that can be anything from minus infinity to plus infinity which would typically make a real challenge over here. So, you will need to some sort of like map it down into a 0 to 1 problem. Now one way in which you can map it down to a 0 to 1 problem is that just put a threshold over there say that if the value is greater than 0 make it 1 if the value is less than 0 make it 0 that obvious.

And then it is pretty much possible there is no harm in that, but what we do is we also make use of these kind of functions say a sigmoid non-linearity. So, what it would do is that as my y becomes tending towards minus infinity this is a value which goes to 0 and as it tends to as plus infinity or a very high number this is a value it goes to 1. So, typically my sigmoid function over here gives me a value in the range of 0 to 1.

Now in the same way as I go with my second non-linearity which is called as a tan hyperbolic. So, you can put down your values of y is varying from minus infinity to plus infinity, and you can very intuitively see that as the value of y goes down to minus infinity this value tends to minus 1 as the value goes to plus infinity this value tends to plus 1. And so there is the you can while you can as well use down a simple threshold which is say just give down the argument that if it is greater than 0 make it one less, than 0 make it 0 yeah, but you can also be using this kind of an argument over here.

Now as we go down a bit later on we would find out why this is an argument which is like these two kind of nonlinearities are something which are preferred for making that decision. So, from there let us get into now that we know about a simple neuron model the next point is how we construct a neural network.

(Refer Slide Time: 10:22)



And now, for that let us look into this one. So, if I have 3 scalar values over here so, what I can do is I can associate it to some different number of patterns, which I want to different number of predictors, which I want to do. So, it may be that based on these three features so, x_1 being the average brightness x_2 being the contrast, and x_3 being the average entropy on the image. I want to classify whether that is a ball in the image yes or no and whether the image is a photograph or image is a painted one, maybe two different ones.

So, p_1 is the standard problem of whether there is a ball in the image or there is not a ball in the image. And now I can write it down in this form so, what happens here is that as you see that these weights now got down subscripted dually. Now with it with this dual weight subscription what happens technically is that the first subscript over here is the target pattern or total the target class to which I want to classify, and the first subscript is the one which connects down which feature is being connected to which target neuron over here.

So, that is how it is done so the feature x_2 connected to this class predictor p_1 is via through the weight of $w_{1,2}$ accordingly. So, this is what it you can get done for y_1 and p_1 in terms of an equation.

(Refer Slide Time: 11:49)

Neural Network Formulation

$$\begin{array}{c}
 x_1 \\
 x_2 \\
 x_3
 \end{array}
 \begin{array}{c}
 \xrightarrow{w_{2,1}} \\
 \xrightarrow{w_{2,2}} \\
 \xrightarrow{w_{2,3}}
 \end{array}
 \begin{array}{c}
 1 \\
 \sum \\
 y_2
 \end{array}
 \begin{array}{c}
 \xrightarrow{w_{2,0}} \\
 \xrightarrow{f_{NL}(\cdot)} \\
 \hat{p}_2
 \end{array}$$

$$\hat{p}_1$$

$$y_2 = [\mathbf{w} \ \mathbf{b}] \mathbf{x} + 1$$

$$\hat{p}_2 = f_{NL}(y_2)$$

Neural Networks for Visual Computing [Debdoot Sheel]

Now, if I get down another parameter p_2 and that is what I was saying that do you have another different thing to predict and that may be that whether it is a natural image or this was a sketched image. Now for that you will have a similar set of equation which you get down over here, now you can clearly see that using the same set of features by just varying down the weights over here, you can make two different classification outputs two different questions can be asked and their outputs can be designed over here.

(Refer Slide Time: 12:17)

Neural Network Formulation

$$\begin{array}{c}
 x_1 \\
 x_2 \\
 x_3 \\
 x_j
 \end{array}
 \begin{array}{c}
 \xrightarrow{w_{2,1}} \\
 \xrightarrow{w_{2,2}} \\
 \xrightarrow{w_{2,3}} \\
 \xrightarrow{w_{k,j}}
 \end{array}
 \begin{array}{c}
 1 \\
 \sum \\
 y_2 \\
 \sum \\
 y_k
 \end{array}
 \begin{array}{c}
 \xrightarrow{w_{2,0}} \\
 \xrightarrow{f_{NL}(\cdot)} \\
 \hat{p}_2 \\
 \xrightarrow{f_{NL}(\cdot)} \\
 \hat{p}_k
 \end{array}$$

$$\hat{p}_1$$

$$y = [\mathbf{w} \ \mathbf{b}] \mathbf{x} + 1$$

$$\hat{\mathbf{p}} = f_{NL}(y)$$

Neural Networks for Visual Computing [Debdoot Sheel]

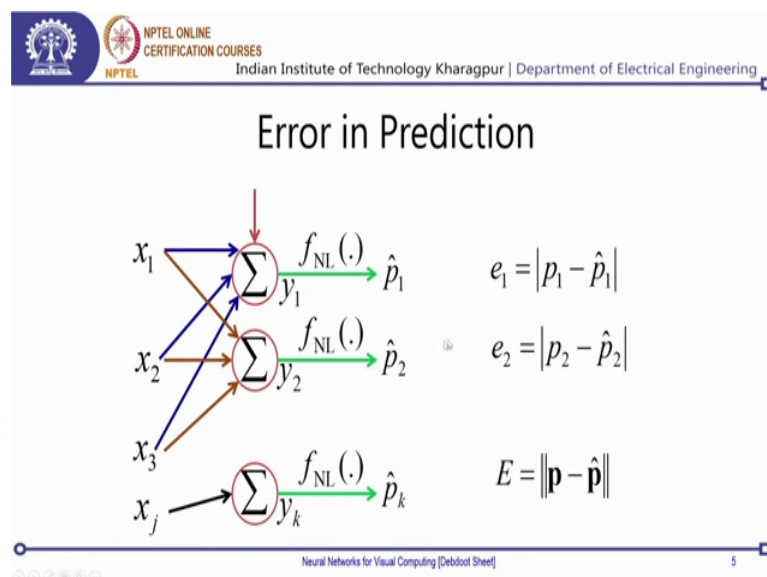
Now, similarly I can extend it to some n number of some k number of neurons over here. So, on the input side my j th neuron will be linking on to my k th output neuron why are the via the weight of kj and this is a generalized formulation which you would get done. Now what happens after this is that if you arrange all of these scalars which are y is in terms of matrix arrangement, because each is independent.

So, you can arrange you would see that all these weights also start getting down into a matrix arrangement. So, all these weights are initially row vectors and all of these y s together is what forms a column vector. So, you can stack down all of these row vectors now once you stand on all the row vectors you get down a rectangular matrix over there of w and b .

And that is this with this matrix combination which you see over here in this equation, and then accordingly your predicted neurons will also be stacked into one single matrix and that is called as a p , while this non-linearity which you see that non-linearity was applied on a scalar, and that is why this can be extended on to a matrix valued form, and then so, a scalar non-linearity function can be applied anywhere on a vector valued function and that will give you the same sort of a vector output which comes down over here.

Now essentially what this helps you is that you can relate down some j number of input neurons to some k number of output neurons in in straight simple terms.

(Refer Slide Time: 13:42)



Now, from there once you are able to relate it down, next what comes down is that I will have certain sort of error when I am able to relate it down. It means that so using these three features and some combination of weights which are present over here, I will be able to predict whether that is a ball or not.



Now, for every image whether there is a ball or not there is a true value which I know, and there is a value which is coming down from this neuron itself. Now the difference between the true value say in the first case there was a ball, but it predicted there was not a ball. So, there is it is an error it is a clear case of an error so, but here what I would get done is that the error value is 1.

In the other way round where say there was not the ball so, this 1 was 0, but it predicted that there was a ball that is also an error. In case both of them predicted that there was a ball and the ground truth is also a ball, then it is you do not have an error if it predicts it there was not at the ball, and the ground truth also says that there is not a ball, it means that it is a correct case. So, similarly we will have it for the second predictor as well, now if you see all of these predictors are independent of each other.

So, it means that the errors are also independent of each other. Now in that case typically what we do is that instead of trying to take down a sort of direct summation over that the best way is to actually take a Euclidean distance between the predicted vector over here, this \hat{p} that becomes a matrix, now and the actual ground truth which is so between your p hat and your p .

So, this will give me a single scalar value and that is what is my error E , now the whole statement of learning over here, is in a sense that I should be somehow able to get down a network such that. This error over here is supposed to become now supposed to come down to 0. Now what essentially happens in that case is we use a method called as error back propagation.

(Refer Slide Time: 15:47)



NPTEL ONLINE
CERTIFICATION COURSES
Indian Institute of Technology Kharagpur | Department of Electrical Engineering

Error Backpropagation

x_1	p_1	\hat{p}_1	$J(\mathbf{W}) = \sum_n \ p_n - \hat{p}_n\ $
x_2	p_2	\hat{p}_2	
x_3	p_3	\hat{p}_3	
\vdots	\vdots	\vdots	
x_n	p_n	\hat{p}_n	

$\mathbf{W} = \arg \min_{\mathbf{W}} \{J(\mathbf{W})\}$

$$\mathbf{W}^{(k+1)} = \mathbf{W}^{(k)} - \frac{\partial}{\partial \mathbf{W}^{(k)}} J(\mathbf{W})$$

Neural Networks for Visual Computing [Deeboot Sheet] 6

So, what it would do is say I have a set of observations x_1 . So, this 1 over here is no more related to one particular feature, but that one which we are putting down over here in this relationship is actually which is related down to which particular sample number I am speaking about. So, what I do is that I take one image the first image for which I know the ground truth for all the predictions I want to do and I have my predicted output from my network coming down.

I take the second image which is x_2 and its features are x_2 and I have my ground truth, and I have my predicted value.

Similarly I keep on doing such that I have n number of images in something which is called as a training set. So, what happens in a training set in this kind of a problem which is a supervised learning problem is that you have a set of images some n number of images and for each image you also know what is the class label given to it. So, here we were asking down two questions, whether there is a ball in the image or not and whether this is a natural image or it is a rendered image or hand drawn image kind of thing. So, there are 2 vectors over here which I there is a 2 dimensional vector, or 2 parameters which I want to predict down to class levels.

So, that should also be known to me so, there are n number of such images on which this is given down and that is what is called as the training set. Now on the other side I am going to predict out all of this with a certain given form of my weights over there. Now

initially what I would do is I would start with a neural network in which all of those weights are randomly initialized.

So, there are some random values now once I start over there, so I would be able to get down this difference. So, this difference is coming down for each, so I get the Euclidean distance for each sample. So, for x_1, x_2, x_3 similarly it up to x_n I get down this difference coming down for n number of samples. Once I get down this difference coming down for all the n number of samples, I can take a simple algebraic summation over this whole dataset and that will give me the error in the data set.

Now I write down that in terms of something called as a $J(W)$ or a cost function J in terms of this parameter W , the reason why we do that is if you look carefully into this one. So, my \hat{p} 's are what are dependent on all my excess over here, but these x values they do not change in the whole data set right, the only thing which changes within the network.

Now which can impact this whole function is the weights of the network and that is why this J is written in terms of $J(W)$. So, we will eventually get into that part as well, now the whole objective is that I want to get down a particular value of W , which is the only thing variable and adjustable within my neural network. Such that my cost function over here is minimum, and this has to be minimum when you need to have done the minimum error.

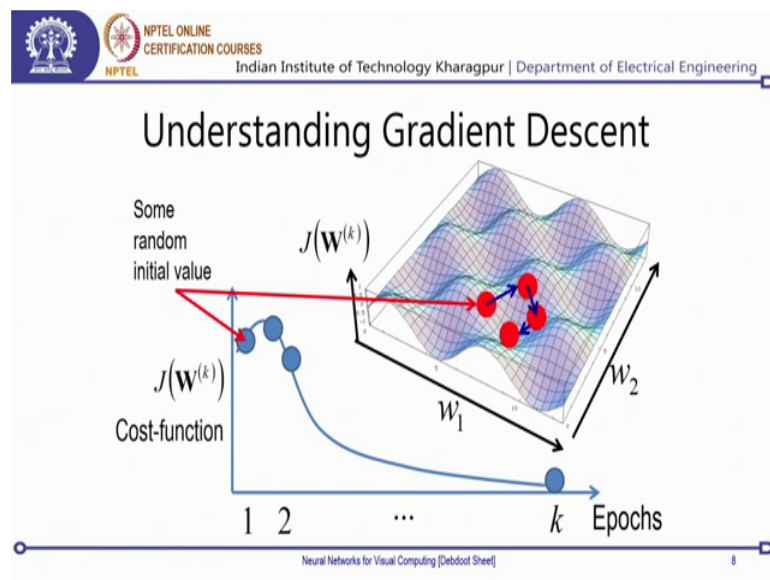
So, as you can see if all p 's match all the \hat{p} 's, you would get down your minimum error over there as you get down your minimum error in this case this has to be 0 so, your $J(W)$ is 0. So, what I would do is I want to get down this weight W somehow coming down, such that my total error over here becomes down to 0, and what we do is we use this particular form of something which is called as a gradient decent way of solving this one.

So, the problem the statement as it goes down is something like this that you initially start with a random guess of W within a k th iteration. So, my k at the start of it will be $k=0$. So, at $k=0$ I start with some W over here, now with that W I will be able to get down my these predictions over here, p_1 to \hat{p}_n from there I would get down my $J(W)$.

And accordingly I can solve out this differential equation over here which is a partial derivative of the cost function with respect to my weight at that particular instant. Now once I am able to get down this partial derivative, I will just subtract it up over there and get down my new estimate of weight which is my W of k plus 1.

Now with this W of k plus 1 I would again start with the whole process, I would get down all of these predictions from there, I would get down get down as the $J W$ for W k plus 1, and then I would iterate it over such that at some point of time I would reach down this minimum value of W and then just stop over here. Now, essentially through this whole thing what we are doing is what is called as gradient descent learning.

(Refer Slide Time: 20:07)



So, in this gradient descent learning what happens is that you have your W and, if you look over this is what the gradient is over there, now as you see this negative sign. So, this is something which is going opposite to the gradient or what we called as descending against the gradient. So, there is a whole decent with this along the direction of this gradient which goes down.

Now in that what essentially happens is something like this that as I have my different epochs varying from 0 till k , so these changes. So, as my k varies from 1 to 3 and goes up to k over there. So, my weights will be changing and accordingly my cost function over here would be changing, and I would be getting down this some sort of a curve in

terms over here and finally, when it when I reach the minimum point over here this is my stopping criteria, this is what I would observe now.



Once we see that this is what we have observed the interesting part is that lets look into what happens in the weight space itself. So, if I plot down to different plots over here say that I have a very simple neuron; we just associate 2 features to one single output to give it much simpler. So, that you have just a 2 dimensional space over here, and this 3rd axis over here of the 3rd dimension is the cost function.

Now as we see we start with some random initial value over there, and with this random initial value. So, we will have a point on the weight space as well as in this space of epochs versus my cost. Now accordingly I update, so I go down to a next new value of W_1 and W_2 new value based on my update rule over there on the gradient descent. Now with that I will get down a different value of cost coming down over here, accordingly I get down to my next one, and then it keeps on updating and as this whole update happens this will come down to a point which would go down to this minima.

And now, if you can clearly see the interesting part is that this looks like an egg gasket design or where you have a lot of minimize and maximize which are just spread around, and this is a pure case which can exist I mean in in most of our neural networks this is the major fallacy which comes down. And as we go down deeper and deeper we will come down to a much better understanding of why this fallacy exists. Now typically the objective is that maybe you are at some value of w which is not exactly the minima.

So, in case you are at the minimum value of W in the first epoch itself you get J W is equal to minimum or equal to 0, basically the absolute minimum, and then you do not have to worry about it. In case you are at some other value which is more than that then it would eventually scroll down and come down just to this minima and that is your learning problem. So, with this the basics of (Refer Time: 22:45) actually comes to an end and we will be doing it, in the next lecture where we have a lab session with a understanding of how to implement this one, and then through that implementation you will be able to relate down the ways of connecting these input images via certain features to or class of outputs.

(Refer Slide Time: 23:08)




NPTEL ONLINE
CERTIFICATION COURSES
Indian Institute of Technology Kharagpur | Department of Electrical Engineering

Take Home Messages

- Haykin, Simon, *Neural Networks and Learning Machines*, 2001.
- Toolboxes
 - Matlab – Neural Network Toolbox (nprtool)
 - Python – Theano, scikits-learn
 - Lua – Torch, nn, cuDNN, nnggraph

Neural Networks for Visual Computing [Debdoot Sheel]



Now finally, if you want to read more in details about these neural networks and the best book to go through is actually Simon Haykins book on neural networks, and learning machines, and for toolboxes well we would be using cycads learn on Python, you are also welcome to use down Matlab with the neural network toolbox which is has a much better gui, and given the fact that a lot of people are most experienced on Matlab you might be able to use it much faster.

But we would be sticking down to our options of doing it on python the other implementations are; obviously, based on Lua torch which also has acceleration with cu DNN as well and the next one on which we will be doing our deep learning things in the next week onwards is what is called as pytorch, and pytorches basically this original library which was implemented on Lua that got full completely ported on to for integration with Python.

So, it is a syntax and the base library of torch which is one of the fastest ones as of date, integrated within python to work it out. So, that is all I have for this particular lecture and just keep on tuned for the next one where we do some hands on as well, so with that.

Thanks.