

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

Lecture – 09
Multilayer Perceptron and Deep Neural Networks

Welcome and so, in the we will be continuing down with the from where we left in the last lecture, and that is on multi-layer perceptron's to deep neural networks, and this is where we would be introducing. So, while till the last lecture we had a good amount of briefing and a recap of the learning rules, and how to create down this single model and then a single perceptron and then a whole collection of perceptron's in terms of it is matrix and the matrix form of representing the data.

From there going down to the gradient and what happens with the gradient based learning rule and then trying to come down to the point of how do we calculate the gradient of the output itself. So, your Del del W of JW how it gets broken down into partial products over there, and using these partial derivative products how you can find out the total derivative of the network itself.

So, from there we would be entering into the multi layer perceptron model of a deep neural networks and how it works out.

(Refer Slide Time: 01:15)

The slide is titled "Gradient Computation" and features a diagram of a single neuron and a mathematical equation. The diagram shows three input nodes labeled x_1 , x_2 , and x_3 with weights w_1 , w_2 , and w_3 respectively, all pointing to a summation node \sum . A bias input of 1 is also shown with weight $w_0 = b$. The output of the summation node is y , which passes through a non-linear transfer function $f_{NL}(\cdot)$ to produce the predicted output \hat{p} . To the right of the diagram, the chain rule is expressed as $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = \frac{\partial J(\mathbf{W})}{\partial \mathbf{p}} \frac{\partial \mathbf{p}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{w}}$. Three blue boxes with arrows point to the terms in the equation: "Derivative of cost function" points to $\frac{\partial J(\mathbf{W})}{\partial \mathbf{p}}$, "Derivative of non-linear transfer function" points to $\frac{\partial \mathbf{p}}{\partial \mathbf{y}}$, and "Derivative of the linear network" points to $\frac{\partial \mathbf{y}}{\partial \mathbf{w}}$. The slide includes the NPTEL logo and text at the top, and a small video inset of the professor at the bottom right.

So, just to do a brief recap of where we left down in the last class, that was on the gradient computation part say I have 3 scalars x_1 , x_2 and x_3 and then I would like to map it down to another predictor scalar which is my \hat{p} , and then how this network was constructed is that I had 3 weights w_1 , w_2 and w_3 , they were all put into a linear summation block, and there was another added component of what is called as the bias.

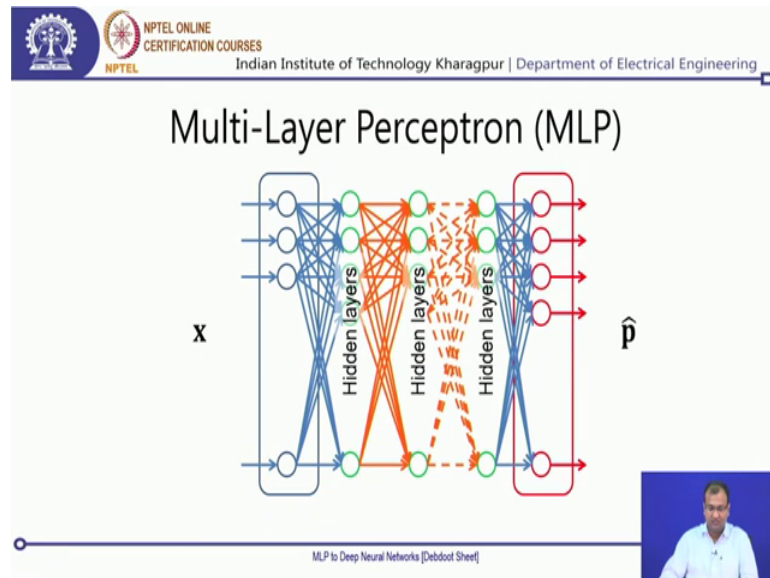
Now taking down all of these linear summations over there, and the bias together you get an output called as y , and that is mapped down through a non-linearity to your predicted output \hat{p} . Now while we have done the forward pass which is from x how to get down to \hat{p} , the question which we had raised is how to get down this derivative, and at first we are doing a partial product of different derivatives.

So, you take the derivative of the cost function which is derivative of J with respect to the output which is \hat{p} , then you take a derivative of \hat{p} or the predicted output with respect to y which is the output from the summation block, and you take a derivative of y with respect to w which is known as the derivative of the linear part of the network.

Now and how this was so, these were the 3 different parts where we left off in the last class. Now the point is that this kind of a computation is what holds true for just 1 single neuron and the next point is that if it is not just one single neuron, but you have a collection of neurons or something which is a deep neural network.

So, one network so, 1 bunch of neurons in one layer then you have another bunch of neurons in another layer, then another bunch of neurons in another layer typically what you called as a multi-layer perceptron due to it is multiple layers form over there.


(Refer Slide Time: 02:58)



So, that is exactly what we are speaking about. So, I have my bunch of inputs x which connects down to a set of intermediate nodes over there, that connects to another set of intermediate nodes and that subsequently to another set of intermediate nodes and finally, you get your final predictor which is a \hat{p} , where each of these intermediate nodes when you are connecting now is what is called as my hidden layer.


Now, the point is that we did find out how to get my output from when I just have one single neuron to connect. So, my inputs to my classification neuron and how to get a derivative, the question is that here also you will need to get down the derivative in its own way, but then in order to get this one you see that clearly there is not just one single connection which connects down the inputs to my output, but it passes through a multiple set of non-linear transformations along the depth itself.

(Refer Slide Time: 03:53)

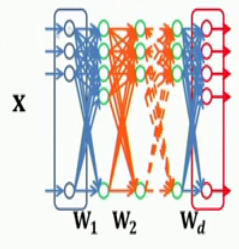


NPTEL ONLINE
CERTIFICATION COURSES

Indian Institute of Technology Kharagpur | Department of Electrical Engineering





Gradient Calculation



$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = \frac{\partial J(\mathbf{W})}{\partial \mathbf{p}} \frac{\partial \mathbf{p}}{\partial \mathbf{y}_d} \frac{\partial \mathbf{y}_d}{\partial \mathbf{w}_d}$$

$$\frac{\partial \mathbf{y}_d}{\partial \mathbf{w}_d} = \frac{\partial \mathbf{z}_{d-1}}{\partial \mathbf{y}_{d-1}} \frac{\partial \mathbf{y}_{d-1}}{\partial \mathbf{w}_{d-1}}$$

$$\frac{\partial \mathbf{y}_1}{\partial \mathbf{w}_1} = \mathbf{x}$$

MLP to Deep Neural Networks [Deeboot Sheet]

And that is where comes the major question which we have so, what we do in that case is something tricky. So, let us look into this small part of the network. So, what we do is say that I am looking at one of my particular layer which is say called as the d-th layer. Now for my a d-th layer what will I will have done is I can write it down in terms of this partial products which is del del p of J W, then I can have del del so del del y of p, which is my output from there.

Now that can be done as an extended product of del del W of y d which is output which is the linear part of summation which comes down to that particular plot. Now if I go down to my d minus 1th layer. So, this is my w d which is just connecting down my output to the d-th layer. Now if I go down to one layer before it now what we can see is that this del del y of sorry, del del W of y d.

So, which is the derivative of this linear part of this block with respect to the weights which are connecting these 2 this blocks over, there can also be written down in terms of a partial product of the output of these blocks, which are z s over here with respect to the partial derivative of the linear part of this one with respect to the block earlier it.

So, this is my d-th layer the output weights connect down to my d-th layer to my target output layer over there, this is my d minus 1th layer and this is the connections, which goes down. So, this is my w of d minus 1 and that is where my expansion happens. Now and similarly I keep on repeating this whole thing together on the chain and finally, what

I would get down is on the final part which is $\frac{\partial}{\partial W} y_1$ which is my first output layer over there and that incidentally is equal to whatever is my inputs over here. So, if you just look into your matrix form of representation of y and w in terms of $y = w$ and x which is you just have a linear product of the weight and the x , so that is our dot product which gives rise to this output over here y so; obviously, my output my derivative of the linear part of output with respect to my weight is going to give my input to it which is my x over here.

So, this is a typical way in which we calculate now our whole networks gradient over there. So, if I want that my total network has to be solved out. So, this is exactly what I would be doing in terms of my calculations so, you can typically look. So, now, that I do not have what is my input coming from here. So, what I would be doing is I do not know exactly what values are over here so, I will be again differentiating this with respect to this and that is what the chain rule keeps on doing. So, every time you have a $\frac{\partial}{\partial W}$ of y . So, you keep on going to the next previous one.

So, over here like this block that block will again be represented in terms of this dot product of 2 partial fractions, partial derivative multipliers over there. And these 2 partial derivative multipliers will again keep on going and subsequently the final point where it stops is a $\frac{\partial}{\partial W} y_1$ and that is equal to my input which is x . So, I believe this part is quite clear to you guys and quite intuitive actually not so, hard to calculate you know.

(Refer Slide Time: 07:30)

Gradient Calculation

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = \frac{\partial J(\mathbf{W})}{\partial \mathbf{p}} \frac{\partial \mathbf{p}}{\partial \mathbf{y}_d} \left(\frac{\partial \mathbf{z}_{d-1}}{\partial \mathbf{y}_{d-1}} \frac{\partial \mathbf{y}_{d-1}}{\partial \mathbf{w}_{d-1}} \right) \dots \left(\frac{\partial \mathbf{z}_2}{\partial \mathbf{y}_2} \frac{\partial \mathbf{y}_2}{\partial \mathbf{w}_2} \right) \mathbf{x}$$

$\nabla J(\mathbf{W})$ $\nabla(\text{net})$

The diagram shows the chain rule decomposition of the gradient. Four blue boxes with arrows point to parts of the equation: 'Derivative of cost function' points to $\frac{\partial J(\mathbf{W})}{\partial \mathbf{p}}$; 'Derivative of non-linear transfer function' points to $\frac{\partial \mathbf{p}}{\partial \mathbf{y}_d}$; 'Derivative of the perceptron' points to the product of derivatives from $\frac{\partial \mathbf{z}_{d-1}}{\partial \mathbf{y}_{d-1}}$ to $\frac{\partial \mathbf{z}_2}{\partial \mathbf{y}_2}$; and 'Input to the network' points to \mathbf{x} . A bracket under the last three boxes is labeled $\nabla(\text{net})$.

MLP to Deep Neural Networks [Debbot Sheet] 15

Now, the next point is that I have my final form of the whole derivative going down something like this. Now that I have this form going down, so my first part of it is what is called as a derivative for my cost function or known as grad of JW the gradient of my cost function. The second parts over there is my derivative of the non-linear transfer function.

Now the other part is the derivative of the perceptron itself and which together is what is called as a derivative of the network and finally, is the input to the network which is my \mathbf{x} . So, these together is what constitutes of any sort of a learning mechanism within a multi-layer perceptron or any kind of a deep neural network. So, what you will have to do is you will have to find out what is my derivative of my cost function, you will have to find out the derivative of the network which together consists of 2 parts of the derivative one is derivative of the non-linear transfer function, and derivative of the perceptron together, and you will have to find out what is that.

So, this is always known to you because you are just pushing in the input to the network, now by solving out this complete derivative over here is what we are able to get down as our neural network learning algorithm in terms of gradient descent, and that is where it goes down. So, that brings us to a very important aspect over here and which called as the existential criteria.

(Refer Slide Time: 08:58)

Existential Criteria

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{p}}$$

Derivative of cost function should exist

$$J(\cdot) = \|\mathbf{p} - \hat{\mathbf{p}}\|^2$$

✓

$$J(\cdot) = |\mathbf{p} - \hat{\mathbf{p}}|$$

✗

$$\frac{\partial \mathbf{p}}{\partial \mathbf{y}_d}, \frac{\partial \mathbf{z}_{d-1}}{\partial \mathbf{y}_{d-1}}, \dots$$

Derivative of non-linear transfer should exist

$$f_{NL}(\cdot) = \frac{1}{1 + e^{-z}}$$

✓

$$f_{NL}(\cdot) = \frac{1}{|z|}$$

✗

MLP to Deep Neural Networks [Deebot Sheet] 16

So, what this essentially means is that in order for the total derivative of the cost function to exist with respect to weight, you need to see that every single fraction of the derivative exists. So, every single part over there we were doing a chain rule of expansion. So, if every single component of the chain rule exists only in that case you would see that the total derivative of the network as well exists, and you can now learn on it in a perfect way.

So, my first component over there which is $\frac{\partial J}{\partial \mathbf{p}}$ of J is what is called as the derivative of the cost function, and that part should be existing. So let us take down a very simple example which is let us take down these 2 cost functions. So, the first one is what is called as the l_2 norm the second one is what is called as the l_1 norm, and let us see if it is derivative exists.

So, I would give you this just some moment to ponder on this one and like really contemplate on what do you think, that is the derivative exists for each of them, so what you can do is quite simple I mean you can just take a $\frac{\partial J}{\partial \mathbf{W}}$ of J over here. And then just find out whether the derivative can exist or not, now interestingly what will happen is definitely it does exist for the first case which is Euclidean norm. So, and then that is not so, hard to contemplate as well because I mean for any kind of an l_2 norm existing existence of a derivative is the pretty a straightforward case, now you have the second one. And this is where the fun is so, do you think that the derivative of this l_1

will exist as well or not just take down a few seconds over here, while I just wait. So, if you look into this one carefully the derivative value should not exist, and one of the reasons why this will not exist is that you will have some sort of a discontinuity at 0.

You see you have an l_1 norm or just a mod. So, $\text{mod of } p \text{ minus } p \text{ hat this } 1 \text{ over}$ there is basically a value which is always a non 0 value, and this hat does have a discontinuity at x equal to 0 and that is one of the reasons why this direct absolute summation will never have a derivative. Now let us look into the other part of the network, and that is the derivative of the non-linear this will be the derivative of the rest part of the network.

And now over here, one important point is that the derivative of this non-linear transfer function or f_{NL} that should also be existing otherwise your $\frac{\partial y}{\partial p}$ that will not come into existence whereas, $\frac{\partial y}{\partial z}$ that does not have any issues, because that keeps on expanding over and over, but your $\frac{\partial y}{\partial p}$ is something which needs to exist at every single non-linearity wherever you are putting now.

So, let us take 2 different cases of nonlinearities over here. So, you see the first one is a sigmoid function, which I have as a non-linearity the second one is a one by mod set or this is something like the inverse of the activity the input which goes down to the network itself, but absolute value of the input. And now the question is does the derivative of each of them exist or not.

So, let us give you some time you can calculate out the derivative of the first function which is your sigmoid non-linearity, I will give you a couple of seconds to yes calculate this part. So, if you look carefully over here, what you would see is that for the first case which is my sigmoid non-linearity the derivative does exist, and that is a perfectly differentiable function.


Whereas, look into the second part of that that again is something which is not differentiable, because of the discontinuity at x equal to 0 right. So, these are 2 some like really interesting facts because, like remember in the earlier class in the first weeks lecture where we were discussing about neural networks and how to make them. So, that is where you were exposed to this concept of what is a non-linear transfer function as well as like what are the different property. So, one property was definitely to make it bounded in some form, but we also mentioned that there are other properties and one of those important properties is that the cost function itself sorry not the cost function,

but the transfer function itself needs to be differentiable in its own term. The question was why do you think it needs to be differentiable is something which gives you an answer over here, and this is one of the major reasons why you need a transfer function to be differentiable from end to end over there. Until and unless a transfer function is differentiable the derivative of the complete network cannot exist and that is the reason why you cannot take transfer functions like say signum of x .

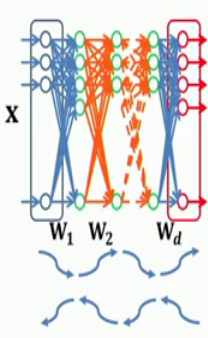
Because if you take a signum function that will again have it is a it is a like, plus 1 for any value which is greater than 0 and it is minus 1 for any value, which is less than 1 and it becomes sort of 0 in order to be made precise, but the point is at x equal to 0 you still see a discontinuity coming down into those function.

And that is the reason why these kind of functions cannot be made cannot be used into it because they are not differentiable, and cannot get our total compliance within the way of how a neural network is going to act and make it is learning in the reverse way. And that brings us to the point of trying to visualize down our learning rule itself. So, now, say that I have a multi-layer perceptron, and I have my inputs x s which connect down through intermediate weights w_1 up to w_d and go down to my final output which is \hat{p} .

(Refer Slide Time: 14:56)



Learning Rule



- Step1: Forward x to obtain \hat{p} net:forward()
- Step2: Compute $J(\cdot)$ criterion:forward()
- Step3: Compute $\nabla J(\cdot)$ criterion:backward()
- Step4: Compute $\nabla net(\cdot)$ net:backward()
- Step5: Update W updateParameters()
- Step6: Go to Step1 if $J(\cdot) > \epsilon$
- Step7: End

MLP to Deep Neural Networks [Debbot Sheet] 17

And the way of how we are doing down is something of this sort and the first step which is called as the forward pass of the network. And this is something similar to what you

had done in your laboratory classes in the last week, where you well learning down how to actually learn down a perceptron model, and for classification purposes we had used a similar kind of a concept over there as well.

So, you would do a first would be a forward pass of this x in order to obtain your \hat{p} . Now that you have your forward pass and you have obtained some \hat{p} over there within a particular given epoch. The next point is that I need to compute out my J , which is my cost function the way of computing this J , or the cost function is that I have my output my predicted output which is \hat{p} . I have my ground truth of my predictors which is p , and then I need to find out what is the Euclidean distance and that was my straight forward way.

Now Euclidean distance for finding out my cost function is not just the only way though we are just sticking down to a very basic form over here, because you have seen that your cost function also needs to be differentiable, and if the cost function is not differentiable, then it does not work. So, we will be going down with more detail cost functions in a bit later down on this course, where we will be bringing in very specific cost functions.

Which are designed for classification will bring in very specific cost functions which are designed for regression problems. So, they would eventually come down, but as of now let us stick down to the very basic form of the Euclidean cost function. So, the next part is that you would be computing whatever is your cost coming down for a given epoch, and whatever combination of weights you have now.

Once you find out your cost function you would be finding out what is the gradient of this cost function or the $\nabla_{\hat{p}}$ of JW that is the first part which was my ∇ of g or the gradient of J now. Once I have my gradient of J the next part is to compute my gradient of the network or ∇ of network, and this gradient of the network is what is the gradient of my \hat{p} , with respect to w and w with respect to w , this was my net.

So, my intermediate points over there for all of these hidden layers where what they are giving me some outputs called a z so, what I was doing is I would in my grad off net is what is my \hat{p} of y with respect to w , and then that gets again remolded in terms of the output z with respect to t with respect to y s and eventually this keeps on going down the line. So, once I have this part also computed and this is very straightforward to compute

down by solving out those sets of equations know, where it comes down the utility of most of the libraries for deep learning is that you do not need to explicitly sit down, and calculate these 1 on pen, and paper or you will not even have to write down a separate the bunch of codes in order to design them.

If you are using very standard forms of cost functions and very standard forms of non-linear a transfer functions over here, then calculating this and this part so, grad of J and guard of net is a very straightforward activity to be undertaken. Now by any of those standard deep neural network libraries say the library which we are going to use down in the next class.

Onwards is what is called as pytorch and that has very standard ways of doing it so, but there you would be in standard doing something like this that you do a forward pass you get down your output from there you will be calculating what is your error in terms of cost from there you find out what is the gradient of the cost from, there you will be next computing out the gradient of the network. Once you have the gradient of the network then you will be doing at update w, and this update w is something which will happen in the reverse way as you had seen in these arrows. So, once you invoke update w, then it would be going down updating all the weights from the output side to the input side together.

And then next is to continue again with step number 1 which is do a forward pass of x obtain p, and then repeat all the steps together until and unless you find that your cost function J over there is so, this will keep on continuing till your cost function J is above a certain threshold value which is epsilon. So, the moment you go below this threshold value. So, you can set empirically set this values.

So, this threshold value can be say something like 10^{-3} 10^{-4} 10^{-5} or even as small as so it can be larger values as well. So, they can be one 2 something of that sort it all depends on the problem which you are handling, but then so, often I do get this question that how do we choose this value of epsilon. Now that is something which well be covering down a bit subsequently in the later classes by coming down to an understanding of the dependence of data to the architecture of the network to the kind of non-linear transfer functions you use and what is the nature of the cost function which you are using over there. So, while certain kind

of cost functions they have a like large dynamic range. So, they some kind of cost functions and have a value in the range of say 1 to 10 some of them have a value in the range of 100 to 10000 some of them, have value which are in order of 10^3 or 10^4 .

So, this epsilon has to be chosen based on what is the nature of the cost function you are using and what is the nature of the non-linear transfer function you are using, and before that it is like really hard to tell down until, and unless we enter in to that. So, while we will be entering into these very practical experiments, we would becoming known to them 1 by 1 and in the lab sessions to understand how these are to be used and finally, the end is that once you are below your certain threshold of error, you can just go and stop the whole learning process.

So, what typically happens is that within your learning over there in while you will be writing down your codes, you would be coming across some very well-known terms, and then say net is basically a pointer which is used for defining network and so, if you look onto the screen over here, will be getting them out.

So, basically net is a variable which defines it is a pointer to the data structure of how to define this neural network a net colon forward is basically what executes this part of it, which is to get down your \hat{p} . Next is a criterion colon forward which executes whatever is the output of this J. The next part is a criterion colon backward.

So, these parameters which go as arguments over here are not indicated over here. So, these are the ones which we will be studying down in the lab classes itself, but it is not so hard to guess down what these parameters would be so, if doing a net colon forward. So, what I need is the parameter over here which goes down is just this x over here, the output of this will be; obviously, \hat{p} . So, now, if I want to do a criterion colon forward I will have to give down \hat{p} , and what is the actual state of p or the ground truth these 2 will be the input to the criterion function.

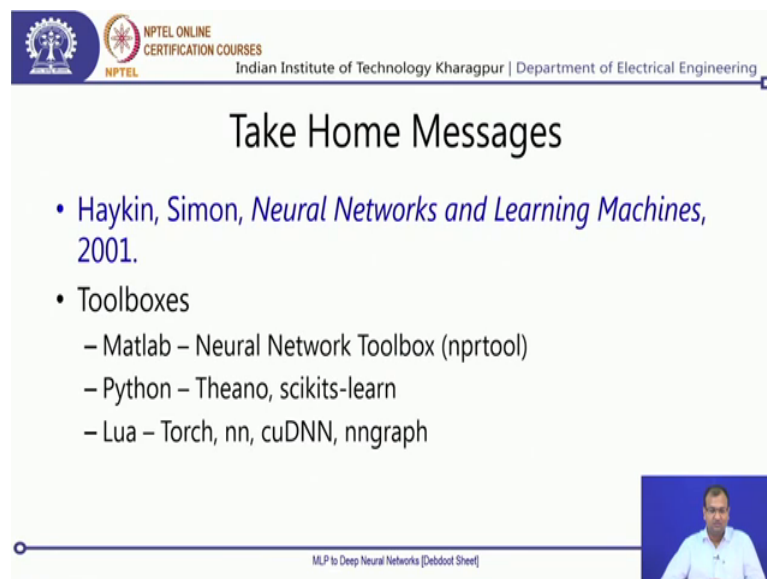
Next is when you find out the gradient of the criterion itself, for the gradient of cost function, that is what is evaluated using something called as a backward operator, and this backward operator what it would be needing now is basically the so I know this pointer to this whole thing. So, what it needs is basically whatever is my predicted output and over here so, what is my cost function. Now the next one is the net colon backward

and this is what will be needing this output and this input together. So, it is not so hard to get it down, because if you go back to the earlier slides into the equations you would see from there how this relates.

And now once this is done the next part is that you need to update parameters of the network. So, this input over here is the this network the pointer of the network net over here, and these 2 backward operators or these 2 gradients which are calculated and then that together will update down w following the update rules, and this will be updating all of these weights not just one of the width.

So, each weight so it goes in a subsequent fashion for this one updates, then this updates and this is what would finish off what is called as one single iteration or one epoch of it, and eventually you can now trace down your error and then either decide to stop or you can continue.

(Refer Slide Time: 23:14)



The slide is a presentation slide from NPTEL Online Certification Courses. It features the NPTEL logo and the text 'NPTEL ONLINE CERTIFICATION COURSES' and 'Indian Institute of Technology Kharagpur | Department of Electrical Engineering'. The main title is 'Take Home Messages'. Below the title, there is a bulleted list of references and toolboxes. In the bottom right corner, there is a small video inset showing a man speaking. At the bottom center, there is a small text box that says 'MLP to Deep Neural Networks [Debdoot Sheet]'.

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

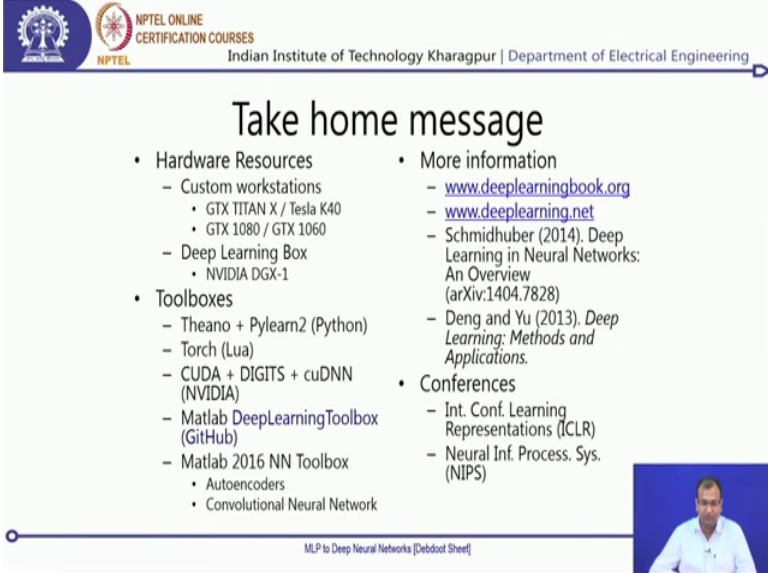
MLP to Deep Neural Networks [Debdoot Sheet]

So, this brings us all most to the end of trying to understand a deep neural network from a multi-layer perceptron model, and at the end of it what I would like really suggest you is that you can do much more detailed reading from the textbook on neural networks and learning machines by Simon Haykin and for toolboxes side of it while for neural networks it is like the most simplest thing, which a lot of people get started with is the neural network tool box within mat lab which is for pattern recognition. So, that is the

NPR tool you can alternatively also look into Theano and Scikits learn in python and Lua for torch.

And other aspects we would be doing down though the labs with Pytouch which is basically a port off so, it is very recently ported out version of torch which can walk down with python environments, and makes it much more easier because of the rest of the script availability within python, and in is based on something called as a dynamic graph based architecture which allows us to compute these forward passes, and the gradients much easily in a computationally attractive form.

(Refer Slide Time: 24:27)



The slide is titled "Take home message" and is part of an NPTEL Online Certification Course. It lists various resources and information for students. The slide includes a header with the NPTEL logo and the text "NPTEL ONLINE CERTIFICATION COURSES Indian Institute of Technology Kharagpur | Department of Electrical Engineering". The main content is organized into three columns of bullet points. The first column lists hardware resources and toolboxes. The second column lists more information, including websites and research papers. The third column lists conferences. A small video inset of a speaker is visible in the bottom right corner of the slide.

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

Take home message

- Hardware Resources
 - Custom workstations
 - GTX TITAN X / Tesla K40
 - GTX 1080 / GTX 1060
 - Deep Learning Box
 - NVIDIA DGX-1
- Toolboxes
 - Theano + Pylearn2 (Python)
 - Torch (Lua)
 - CUDA + DIGITS + cuDNN (NVIDIA)
 - Matlab DeepLearningToolbox (GitHub)
 - Matlab 2016 NN Toolbox
 - Autoencoders
 - Convolutional Neural Network
- More information
 - www.deeplearningbook.org
 - www.deeplearning.net
 - Schmidhuber (2014). Deep Learning in Neural Networks: An Overview (arXiv:1404.7828)
 - Deng and Yu (2013). *Deep Learning: Methods and Applications*.
- Conferences
 - Int. Conf. Learning Representations (ICLR)
 - Neural Inf. Process. Sys. (NIPS)

MLP to Deep Neural Networks [Debbot Sheet]

And finally, if you are like really into getting down into more deeper understanding. So, the typical suggestion is that they do not tend to these networks do not tend to work down. So, great on laptops you might experience heating problems as well. So, the best point is that get on a custom workstation. So, I mean a very competitively built up workstation can be achieved with GTX 1060 or a GTX 1080, and 1080 TI machines. So, this in Indian rupees this would be costing in less than a bracket of 1 lakh rupees to get a total desktop raised and setup. You can get done much more professional ones with the titan x or tesla version or you can even think of buying down any of these, for tool boxes these are again multiple of these toolboxes, and just to for a revision these are the different reading sources from where you can read down.

And these are the 2 major conferences which where you would see down most of these advancements in the field coming down. So, with that we come to an end on deep neural networks and multilayer perceptron's on the lecturing, and theory in the next class we will be doing a lab session where you would be going to a good walk through, and getting to execute it on your side or as well as on the remote clusters to which you can gain access under certain kind of academic licenses as well. So, with that thanks and stay tuned for the rest of the classes as well.