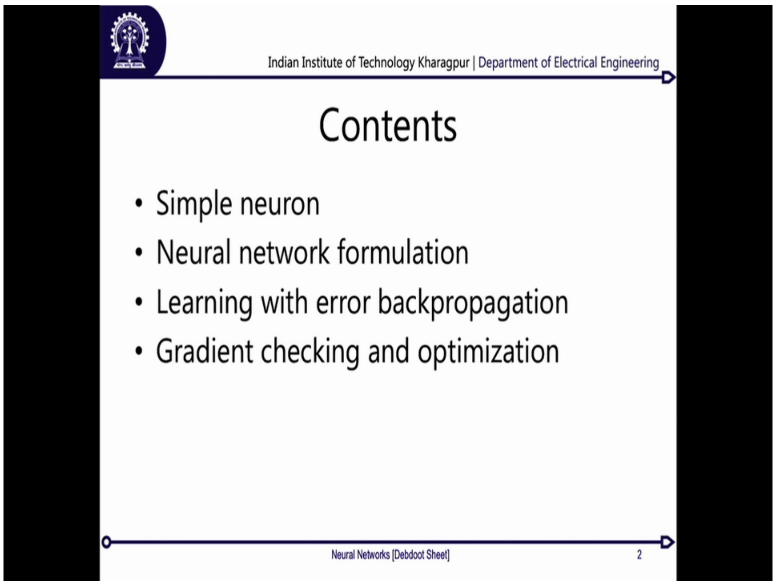
Introduction to Medical Imaging and Analysis Softwares

Professor Debdoot Sheet
Department of Electrical Engineering
Indian Institute of Technology Kharagpur

Module 3
Lecture No 13
Neural Networks for Segmentation and Classification

So welcome, to yet another exciting topic which we would be covering today. And this is on Neural Networks and very specifically aimed at medical image analysis. So while we have dealt a lot of topics about how features are extracted and what they can be describing over there and we had done our last class on using Random Forest and Decision Tree like learners as well for doing classification segmentation problem. So yet again another exciting one is can we use neural networks for doing the same thing as well? We will be starting with a very primitive introduction to what neural networks are and from there we will be proceeding on to how we are going to actually do it and there is a exciting demonstration as well using a very small snippet code on Matlab as well.

(Refer Slide Time: 1:10)


Indian Institute of Technology Kharagpur | Department of Electrical Engineering

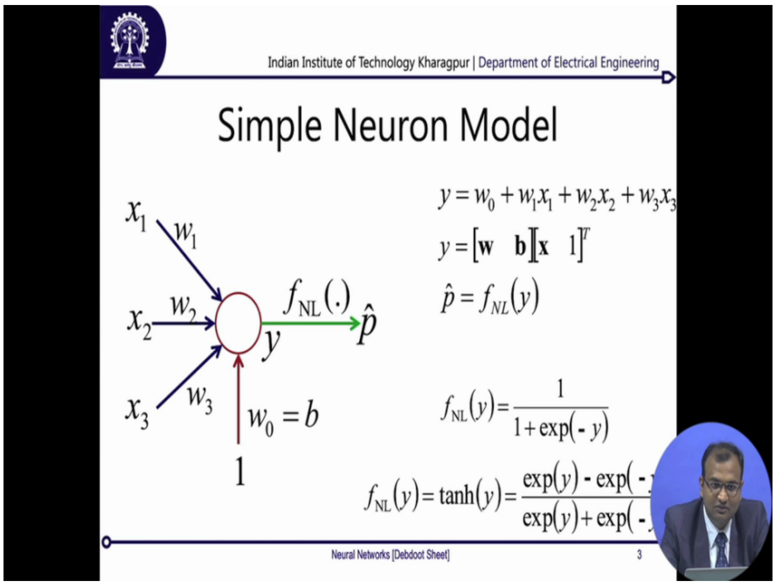
Contents

- Simple neuron
- Neural network formulation
- Learning with error backpropagation
- Gradient checking and optimization

Neural Networks [Debdoot Sheet] 2

So this talk is organized where I am going to speak about what a neuron model is, so mathematically how you define a neuron as in its own (()) (1:13) and from there I would be moving onto a say what is a neural network and what the neural network formulation looks like and from there I would be proceeding onto a very standard learning algorithm which is called as Red Backpropagation, which is as of now one of the most commonly used. So there are different ways in which how this error is backpropogated and how things are optimized. And then we have gradient checking and optimization, which is just optimization criteria as to when to stop and what is the final conformal configuration of a neural network where it will be giving you the best of classification or say regression problem.

(Refer Slide Time: 2:17)



Indian Institute of Technology Kharagpur | Department of Electrical Engineering

Simple Neuron Model

x_1
w_1
x_2
w_2
x_3
w_3
$f_{\text{NL}}(.)$
y
$\hat{p}$
$w_0 = b$
1

$y = w_0 + w_1x_1 + w_2x_2 + w_3x_3$

$y = [\mathbf{w} \quad \mathbf{b}][\mathbf{x} \quad 1]^T$

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

$f_{\text{NL}}(y) = \frac{1}{1+\exp(-y)}$

$f_{\text{NL}}(y) = \tanh(y) = \frac{\exp(y) - \exp(-}{\exp(y) + \exp(-}$

Neural Networks [Debdoot Sheet] 3

So without much of delay, let us start with how we define a simple neuron model. So see that you have a feature vector, which just has 3 dimensions, so I can represent a particular feature sample over there as a vector x which can have 3 different scalar values X1, X2 and X3. So these are basically 3 different components which make 3 different features. So now I take these once and then I want to predict a particular variable which is called as p and note a fact that I am putting a small hat on top of p because this is a predicted variable.

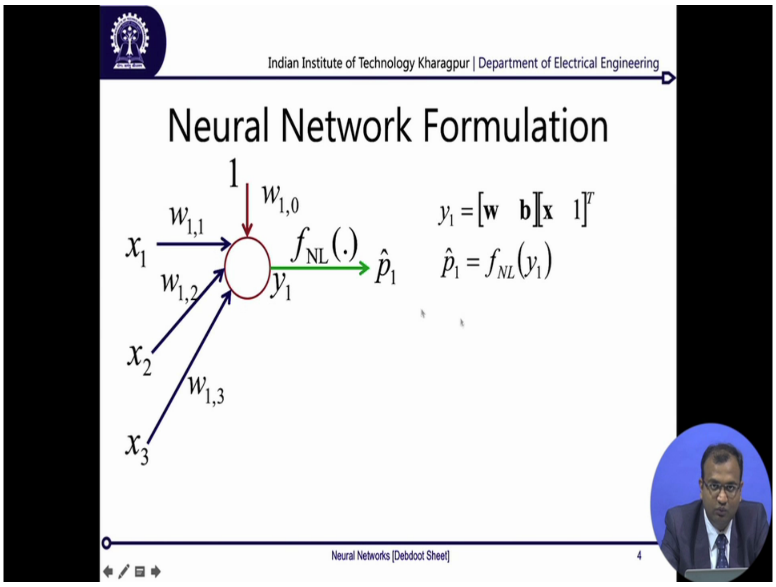
The true state of the variable will be the same value p without that particular hat and p stands because I am using it as a predictor constant over here. So now let us have a very simple model which would say that let us multiply each of these observation points over there with a certain weight. So X1 is multiplied by a weight called as w1, X2 is multiplied by a weight called as w2 and X3 is multiplied by a weight called as w3, ok. Now, we additionally take another weight called as W not or the bias and so you can also define this bias as unity multiplied by a weight W not, ok. In that case your W not to W1, 2, 3 they all formed on one simple vector which is called as W. Now if, I have a summer over here then my output of this summer would appear something like this which is y = W not + W1X1 + W2X2 + W3X3.

Now in terms of a matrix multiplication, I can just write it down as a scalar multiplication between a weight matrix Ws, I also take b as a bias term over there and I take a transpose of all of these access and unity. So this is going to give me a scalar, a dot product between them is going to give me a scalar value. Now, I apply certain sort of a transformation on top of this result in output over there, which is called as a nonlinear transformation and this will be mapping my output of this summer onto my predictor which is p hat. Now this nonlinear function can be of this particular form which is 1 by 1 + e to the power of - y, where y is the input given down to this nonlinear function.

And this has a very famous name this is called as the sigmoid non-linearity function. Now if you look here very carefully, so if the value of y is tending towards 0, so you would have basically e to the power of 0 which makes it 1 so the value at y = 0 for this function is 0.5. Now, if the value of y is a quantity which is very much negative so say - 1000, ok. now you get 1 by 1 + e to the power of - of - 1000, which makes it 1 by 1 + e to the power of 1000, which is a very large number compared to 1 and of, so in that case this nonlinearity fNL would actually tend towards e to the power of - 1000, where it goes and that is a very low number very close to 0 actually.

On the other way, if you see the upper bound of it, when y tends to a very large positive number then you will be having this whole thing e to the power of - y, where y is a large number tending actually towards the 0 and then this whole function over here will be of a form of 1. So you see that for negative values of y you have a saturation at 0, for very positive values of y you again saturate at 1, so even if your y becomes unbounded in some form this nonlinearity imposes that your final predictor is always bounded in the range of 0 to 1. Similarly there is another famous function which is called as tan hyperbolic and this also has a similar bound (()) (5:49) for it, it is not bounded in the range of 0 to 1, but it is bounded in the range of - 1 to + 1. So these are certain kind of nonlinear functions which we use so that our responses always remain bounded.

(Refer Slide Time: 6:07)


Indian Institute of Technology Kharagpur | Department of Electrical Engineering

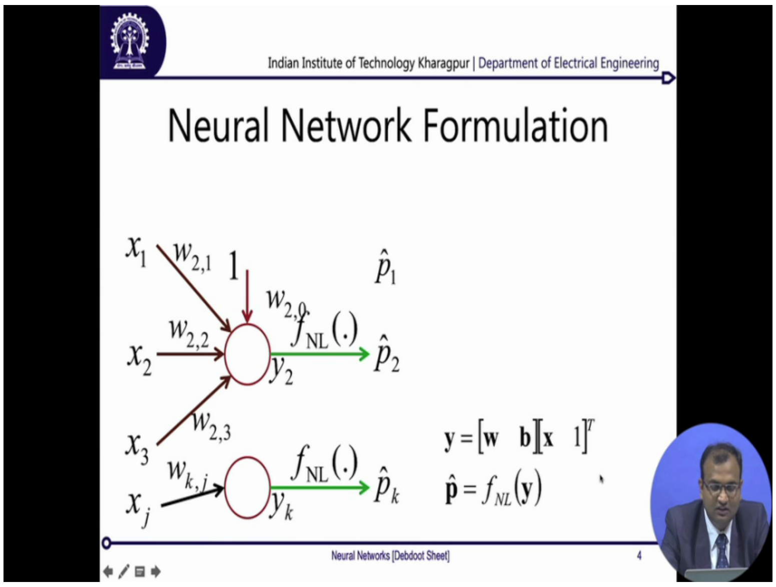
Neural Network Formulation

$$y_1 = [\mathbf{w} \quad \mathbf{b}][\mathbf{x} \quad 1]^T$$
$$\hat{p}_1 = f_{NL}(y_1)$$

Now from there let us look into how we can formulate a whole network using these kind of neuron models. So as of now what we were doing is we take down multiple such input parameters over there, we are able to predict only one of those output values. But my constant may be that I want to predict multiple of those output values and that is where a neural network will form down that I have a network being created.

So in that case so we take the same simple example that let there be just three different parameters over there and let there be one single neuron, which is mapping it to one predictor value p1. Now note, I have taken these weights no more as just W1, W2, W3 they are now written as W1, 1, W1, 2, W1, 3 and this basically stands the first element is the target where it is being mapped, the second element of the subscript is the source from where it is being mapped and this is a very standard convention how it goes. Then that is by the same reason that your output over here is also termed as y1, ok.

(Refer Slide Time: 7:22)


Indian Institute of Technology Kharagpur | Department of Electrical Engineering

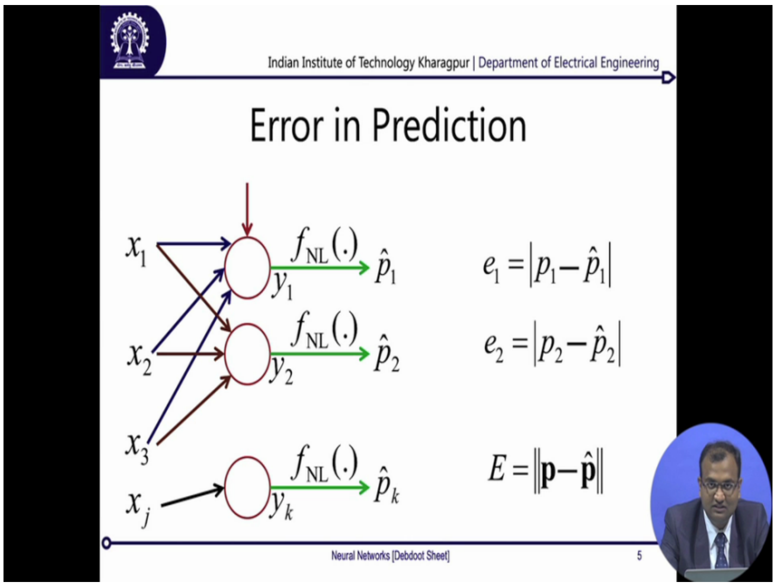
Neural Network Formulation

$$\mathbf{y} = [\mathbf{w} \quad \mathbf{b}][\mathbf{x} \quad 1]^T$$
$$\hat{\mathbf{p}} = f_{NL}(\mathbf{y})$$

Neural Networks [Debdoot Sheet] 4

Now in a similar way I can map it down to the second node as well and once I map it down to the second node I will be able to get down my response from the second node as well and this will be the form in which it would appear. Now let us say that I have j number of such input nodes and I would like to map it to k number of such output nodes. And in that case very typically, weight relation between the Jth input node to the kth output node will be mapped as W of k, j and you would a similar kind of nonlinearity and this is the basic form in which it appears.

(Refer Slide Time: 7:59)


Indian Institute of Technology Kharagpur | Department of Electrical Engineering

Error in Prediction

$$e_1 = |p_1 - \hat{p}_1|$$
$$e_2 = |p_2 - \hat{p}_2|$$
$$E = \|\mathbf{p} - \hat{\mathbf{p}}\|$$

Neural Networks [Debdoot Sheet] 5

Now, if we look at the total network at any point of time, this is how the picture of the whole network is going to look like. Now over there for my first input over here for my first set of inputs and my first element on this output over here, I will be able to find out certain difference which is called as the error, which is I am predicting the value of p1 as p1 hat and my actual value should have been p1. So there would be a difference between them, say this is 1 and this comes down somewhere around 0.8, so you have a difference of 0.2 or similar kind of cases might appear.

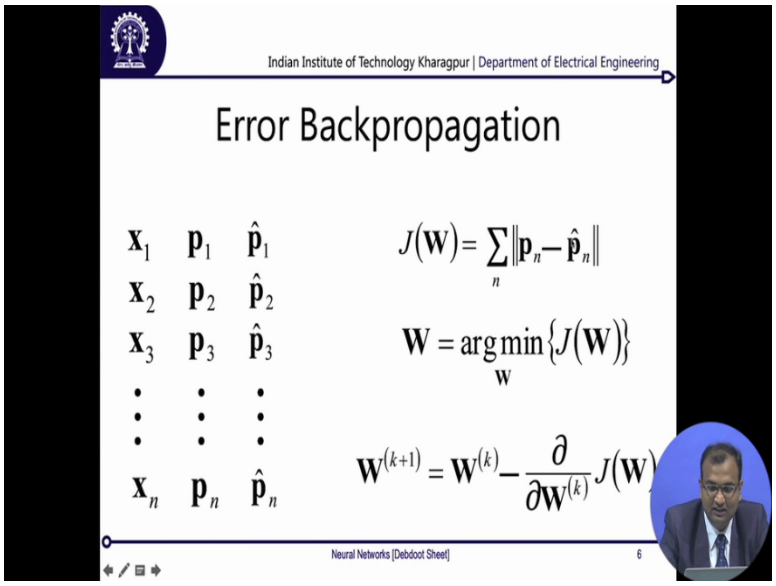
If this is also 1, this is also 1 you get an error of 0, that means you have done perfect mapping over there. So this function over here we just take down accumulation of these errors which come down. Now similarly, for the second node also you will be getting a similar quantity however, the question is you are going to get all of these errors for each of them, is there some way in which we can say about the cumulative error being predicted in this whole network. Now in that, what you can do is assuming that this is also a vector and you can actually compute the Euclidean distance between these two vectors. So you have a set of those actual state variables and you have a set of those predictions which you are getting down from this network. Take down the Euclidean distance that is going to be a scalar quantity, which will be a cumulative error of this network while it is predicting it out.

Now keep this in mind, this is a very important step because this is what we are going to subsequently use in the next stages as well. So this commodity over here is called as the mean squared error, which I am going to take down, ok. So if I am going to so I will come down to how we are getting the mean quantity over there, but this is as of now just a square root of the error which we are getting.

So the next term which we call is known as “Error Backpropogation”. Now what this does is pretty simple, say you have a network you put down you had put down certain values of those weights over there. Now you would actually want that this neural network performs in a way such that my error e, which I was computing in the last step, should be minimum that is my final goal and objective which I need to achieve.

However, the point is that how do I select what are the values of those weights such that this network will give me the minimum error and that is a question in hand. Now, we solve it using this method called as error backpropogation in which what we do is we just compute what is the final error coming down with a particular combination of these weights and then we will like to backpropogate it, or push it back into the network and then keep on refining the values of these Ws such that I will come down at a place, when after some point of time after refining it for a few number of iterations I should be at a position to get down my maximum performance over there or minimum error.

(Refer Slide Time: 11:02)


Indian Institute of Technology Kharagpur | Department of Electrical Engineering

Error Backpropagation

$$\begin{array}{ccc} \mathbf{x}_1 & \mathbf{p}_1 & \hat{\mathbf{p}}_1 \\ \mathbf{x}_2 & \mathbf{p}_2 & \hat{\mathbf{p}}_2 \\ \mathbf{x}_3 & \mathbf{p}_3 & \hat{\mathbf{p}}_3 \\ \vdots & \vdots & \vdots \\ \mathbf{x}_n & \mathbf{p}_n & \hat{\mathbf{p}}_n \end{array}$$

$$J(\mathbf{W}) = \sum_n \|\mathbf{p}_n - \hat{\mathbf{p}}_n\|$$

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

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

Neural Networks [Debdoot Sheet]
6

Now for that what I can do is I will not just be having one single observation of Xs, but I will be having a series of such observations. So say in a simple example I have a set of images on which I can say whether they are some sought of diagnostically abnormal or diagnostically normal. So we will be later on the class, when we do the coding exercises, we would be taking a very simple example in which we are going to take down my cross copic images of white blood cells and we would be predicting whether each of this WBC images over there are cancerous of a particular kind, which is acutely probabilistic leukemia or they are not at all leukemic in any way and they are perfectly normal.

So this is what we are going to do, so we will be just given images, so corresponding to each image I can extract a set of features, so that will be for my first my image my set of features will be X1. So that image will have a state variable which is either it is normal or abnormal 0 or 1 and network would be predicting a certain value. For the second image I will be having my features, I will be having the state variable over there.

Now make a note over there that I have generically used a vector notation for p, but these can be scalars as well. So I can put them as class labels of 0 and 1, or I can keep them as a binary tuple class in which it will be the first one is the true or false for the first class and the second element over there is a true and false of the second class. So if it is class 1, this will be appearing in a form as 10, if it is class 2 it will be appearing in a form of 01, so that can also be a way in which this can be represented, similarly I have for all the n number of images.

Now what I do is, I write cleverly something called as the cost function, which is j of omega and this is where this submission over this error comes in play. So what I do is typically for a set of weights, I will be pushing all of these training samples over there and I would be finding out what is my difference of error computed for each of the sample, then I take a sum of all of those errors because that is going to represent my cumulative error in performance of the network over all the samples over which I have as of now pushed forward through the network.

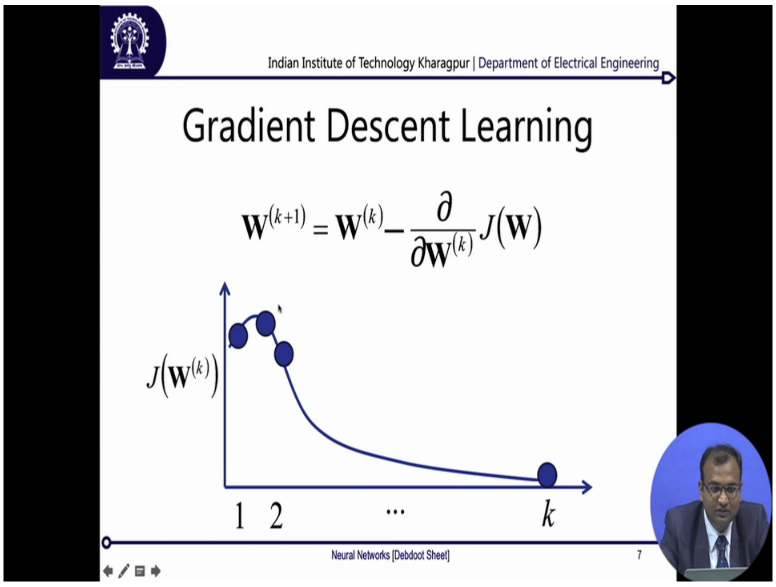
Now from there my objective is, somehow I should be able to minimize this cost function which is the error, because if I am able to get minimum error across all the samples I am using for training, I would be able to get the network which has maximum performance accuracy over there. Now for that we write down formally our whole objective as the argument of W in which I get a minimum condition for my cost function coming up over there.

Now in order to achieve that what we do is called as gradient descent learning and this whole learning paradigm is defined something like this. We have another variable called as an epoch or k over here, so what we do is for a certain epoch whatever is my weights I am going to subtract the derivative of the change of this cost with respect to the weights over there.

Now remember one thing that j of w is a function of pn and pn hat, now pn is a constant but pn hat is actually a function of all of these Xs, because your predictor was over there. Now, when you have a function of all of these Xs and Ws over there, so this function is actually dependent on both x and w. So I can take a derivate of this whole thing with respect to w, so this becomes a finitely solvable problem given the point that each function, which is functionally constituting simple elements over this variable should also be having a finite derivative only in that case.

And as you see the all the earlier equations, which we have used they are submission and there is a certain amount of nonlinearity which is continuous and differentiable throughout the range. Now that is why this comes down as a differentiable quantity and you can always keep on solving. Now the question is now that I get it, how do I actually train my network which is objective which we are trying to achieve over here during the process of training?

(Refer Slide Time: 15:22)


Indian Institute of Technology Kharagpur | Department of Electrical Engineering

Gradient Descent Learning

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

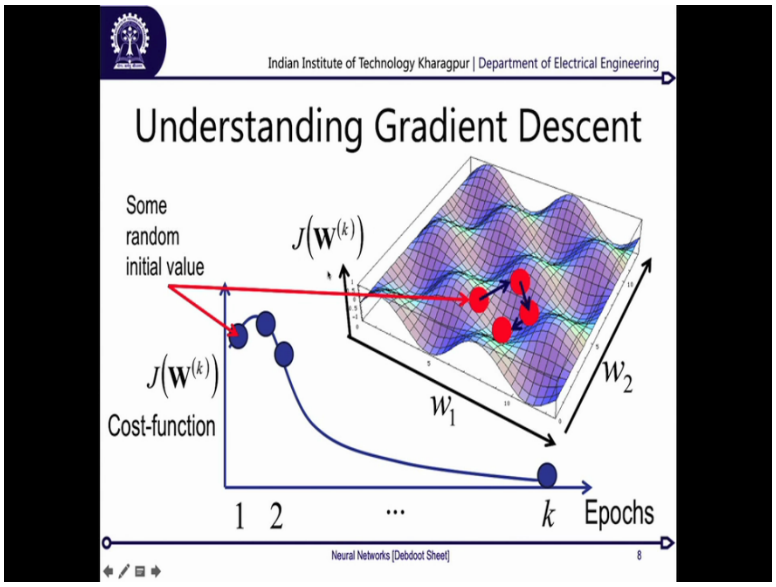
$J(\mathbf{W}^{(k)})$
1
2
...
k

Neural Networks [Debdoot Sheet] 7

So what we do is, we take the same equation over here, now we would start with some sort of an initial guess of this Wk, say that is a random guess at which I land down with the configuration called as W1, and at this W1 I would be getting a certain error which is over here. Now based on this error and solving out these derivate, I can update the value of Wk, now having updated the value of Wk there is a chance that I would land up somewhere here and this will be my error position, based on that I am again going to update it, so at each iteration I am going to compute my cumulative error over all the samples I am using for training.

And this way I keep on repeating till I land down at a position k, which is the final point where I can actually stop and beyond this the error is not going to minimize any further. Now this scale over here of change of case is called as epochs and this comes down from the fact that within each epoch or within a small age of time I am going to have all the possible changes being taken care off over of there, so I use all the samples which are present over there in order to compute my errors and update my network and then I keep on repeating across epochs.

(Refer Slide Time: 16:39)


Indian Institute of Technology Kharagpur | Department of Electrical Engineering

Understanding Gradient Descent

Neural Networks [Debdoot Sheet] 8

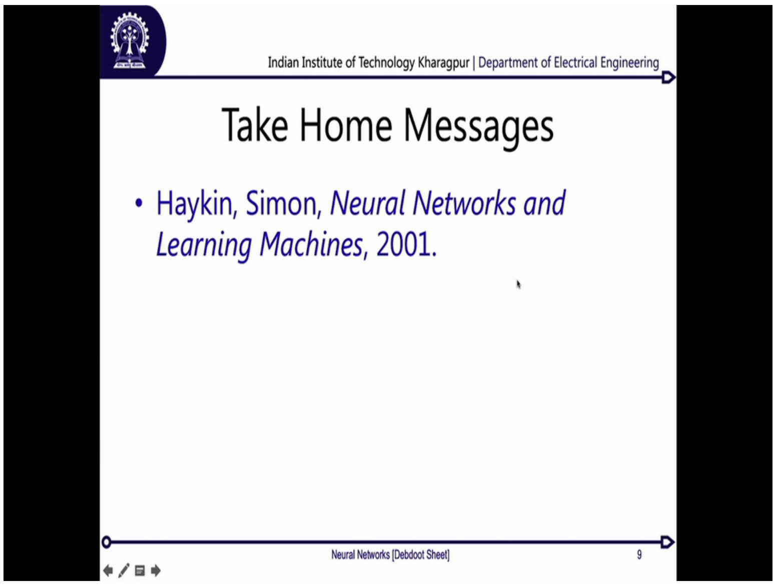
Now let us look at a fun part over here which is how this gradient decent would actually work. Now we were looking into this graph over here which is about the cost function varying with the number of epochs, but we were never trying to look into what these weights what was happening to the space of these weights because that is where the fun is located actually.

Now what happens over there is we will be starting with some random guess, we have just two plots over here and we are going too looked a simultaneously into two both the plots. Now, at the first epoch I have this first initial guess, where I start now that is a pure random guess. Now based on that I am going to update, so I actually got increase in my cost function and here I am going to change my position appropriately because my Ws are also going to change so it is a different combination of Ws.

Now from there I moved down to another position from there I keep on moving till the kth position, when I get down somewhere over here. And this it now what would happen is if you are going to change or across all of these values of Ws say you are empirically instead of training the whole network what you do now is you take a network and then you possible point out all the possible values of Ws and then you can actually get a surface plot over there. So say W1 is varying in the range of - 500 to + 500 and Wt is again varying in the range of - 500 to + 500. So I just changed these values in discrete steps over there and then I would be getting a 2D matrix which this height of this whole thing will be giving me this surface over this cost function over there.

Now what typically happens is that as you are going to look at a point over there and then you find out the gradient, this gradient was actually del W of J(W), so that is the first derivative of this cost function with respect to the weights itself. So it is going to follow the slope along this weight surface and come down to the point wherever this cost is minimum. So that is where this concept of gradient descent comes from, it does not comes from the concept that this graph is a epoch versus cost graph but this is a gradient it is going to descent along the gradient inside this surface plot of the cost function itself. So on the plot, on the particular function of the cost function versus the weight is going to follow the gradient along that and come down to a position where this error is going to be minimum and that is what this whole training is all about.

(Refer Slide Time: 19:13)



Indian Institute of Technology Kharagpur | Department of Electrical Engineering

Take Home Messages

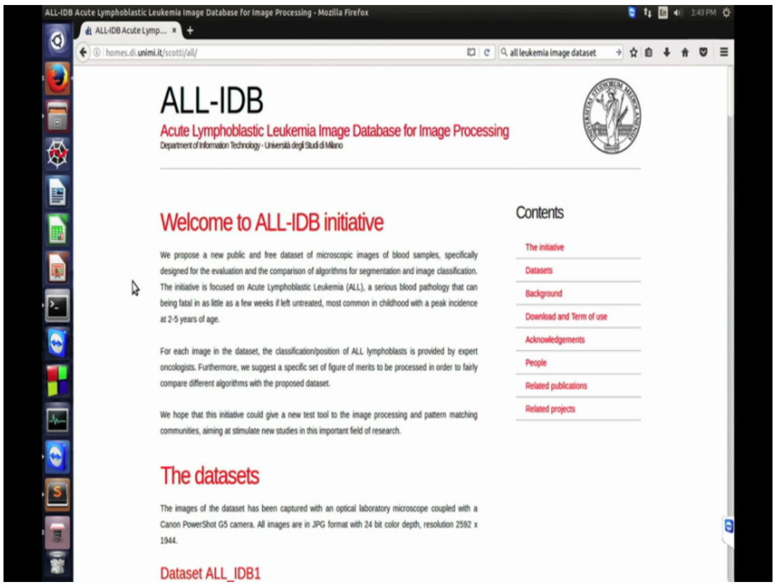
- Haykin, Simon, *Neural Networks and Learning Machines*, 2001.

Neural Networks [Debdoot Sheet] 9

Now with that we are finishing off with the theory and as a take home message what I would do is I have a pointer you can have a look into this book called as Neural Networks and Learning Machines by Simon Haykin, which is a very famous book on Neural Networks, so please have a look through it for understanding theory in much more details. And if you would like to play your hands around then I have a list of these toolboxes. In Matlab, we have the stuff called as Neural Network Toolbox the simple,nd is “nprtool”, it is a GUI based and we will be doing a second subsequent experiment on this one itself.

On Python you can use the toolbox called as Theano and there is another scientific programming language called as Lua on which a particular library for mathematical intensive programming is Torch and within that you can use certain modules called as nn, cuDNN and nngraph which we will be looking in fact in the subsequent lecture on Deep Neural Networks as well as how to play around with them in solving a similar kind of problem, so with that we would be proceeding next to the demonstration on how to use these once for a library practical problem yeah.

(Refer Slide Time: 20:22)



ALL-IDB

Acute Lymphoblastic Leukemia Image Database for Image Processing

Department of Information Technology - Università degli Studi di Milano

Welcome to ALL-IDB initiative

We propose a new public and free dataset of microscopic images of blood samples, specifically designed for the evaluation and the comparison of algorithms for segmentation and image classification. The initiative is focused on Acute Lymphoblastic Leukemia (ALL), a serious blood pathology that can being fatal in as little as a few weeks if left untreated, most common in childhood with a peak incidence at 2-5 years of age.

For each image in the dataset, the classification/position of ALL lymphoblasts is provided by expert oncologists. Furthermore, we suggest a specific set of figure of merits to be processed in order to fairly compare different algorithms with the proposed dataset.

We hope that this initiative could give a new test tool to the image processing and pattern matching communities, aiming at stimulate new studies in this important field of research.

Contents

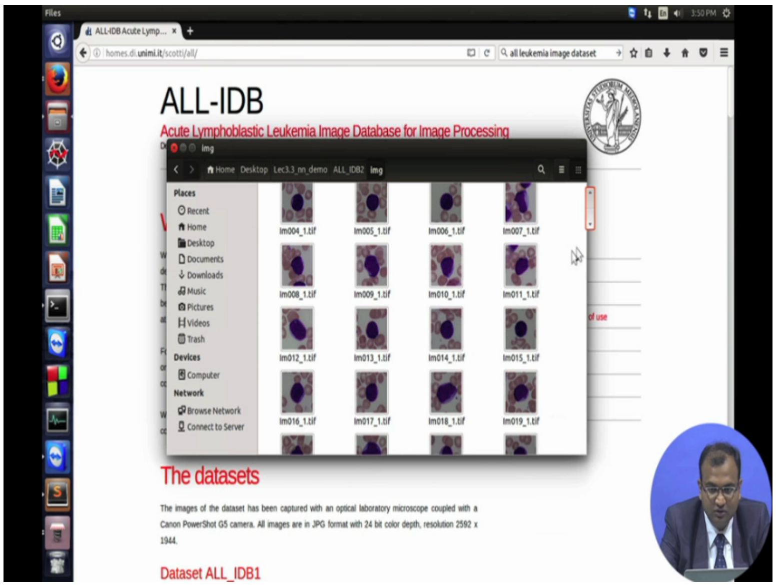
The datasets

The images of the dataset has been captured with an optical laboratory microscope coupled with a Canon PowerShot G5 camera. All images are in JPG format with 24 bit color depth, resolution 2592 x 1944.

Dataset ALL_IDB1

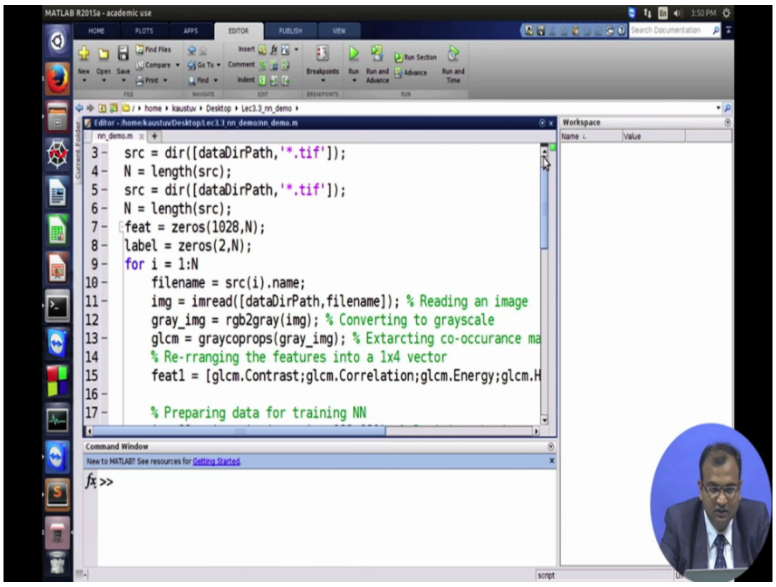
So this is about the experiment we will be doing today, so I would be using the ALL-IDB dataset for this thing. So as I was telling you this particular dataset is about leukemic images and this is a particular condition of cancers within the blood in which your lymphoblast which are a type of white blood corpuscles, they actually get cancerous and this whole challenge is about being able to classify each WBC image into whether they belong to the ALL class or they do not belong.

(Refer Slide Time: 21:02)



So you can get your dataset by signing into the terms and conditions of use over here and download the same thing. So for my purpose I actually have the whole dataset downloaded with me and this is how these images look like over here. So typically you will be having each of these images which are sort of WBC and what they do is they have actually marked each of them with a file name which has an underscore 1, or underscore 0s. So these 0s are the once which are perfectly normal and the 1 which wherever you have a underscore 1 written they are the once which are leukemic in nature and the problem is very simple, you have this image patches and from these images patches you just have to classify whether this one belongs to an abnormal class or a normal class.

(Refer Slide Time: 21:48)

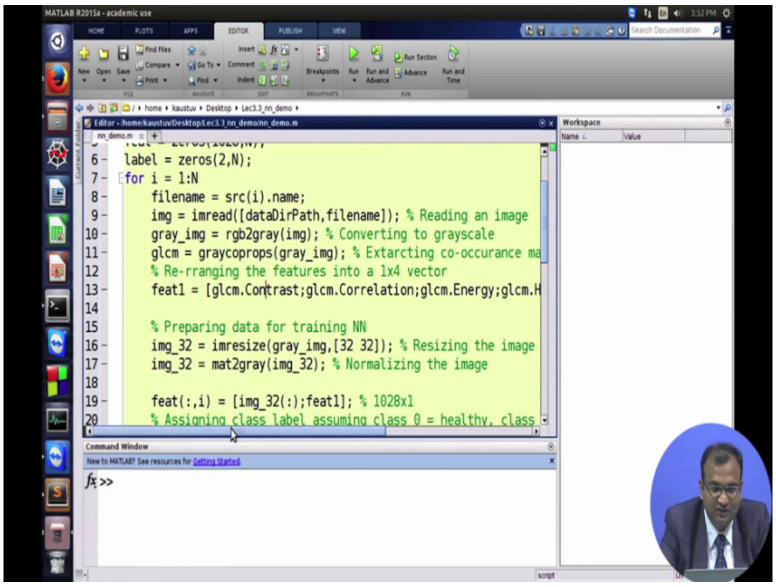


```
src = dir([dataDirPath,'*.tif']);
N = length(src);
src = dir([dataDirPath,'*.tif']);
N = length(src);
feat = zeros(1028,N);
label = zeros(2,N);
for i = 1:N
    filename = src(i).name;
    img = imread([dataDirPath,filename]); % Reading an image
    gray_img = rgb2gray(img); % Converting to grayscale
    glcm = graycoprops(gray_img); % Extarcting co-occurance ma
    % Re-rranging the features into a 1x4 vector
    feat1 = [glcm.Contrast;glcm.Correlation;glcm.Energy;glcm.H

    % Preparing data for training NN
```

```
fx >>
```

Let us look into our Matlab code which does that, so I have a neat script over here, thanks to one of the TS who helped me prepare this one. So what we do is typically I am just doing a bit of book keeping over here which is I get down to the path where my data is kept and then I am reading down all the images, which are present over there using this directory function and then just looking into how many images did I read over there. Now from there what we do is typically that I try to create down two different vectors over here, one is this feature vector which is all my Xs and the other one is my labels matrix over there. Now, you remember that in my feature vector what I have is basically 1028 such columns and N of such rows, so for each of them I will be having 1028 different features which are being extracted.

(Refer Slide Time: 23:17)

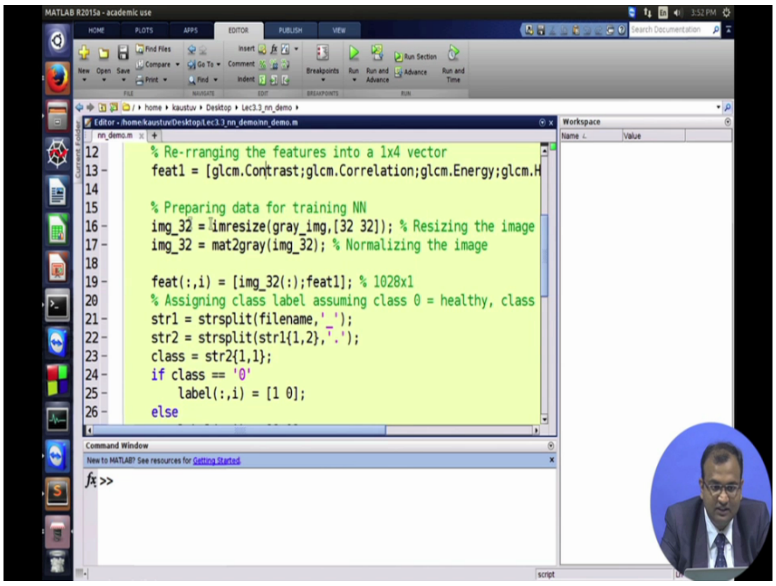


```
label = zeros(2,N);
for i = 1:N
    filename = src(i).name;
    img = imread([dataDirPath,filename]); % Reading an image
    gray_img = rgb2gray(img); % Converting to grayscale
    glcm = graycoprops(gray_img); % Extarcting co-occurance ma
    % Re-rranging the features into a 1x4 vector
    feat1 = [glcm.Contrast;glcm.Correlation;glcm.Energy;glcm.H

    % Preparing data for training NN
    img_32 = imresize(gray_img,[32 32]); % Resizing the image
    img_32 = mat2gray(img_32); % Normalizing the image

    feat(:,i) = [img_32(:);feat1]; % 1028x1
    % Assigning class label assuming class 0 = healthy, class
```

```
fx >>
```

Now, on labels I did tell you that it can be a 01 classification problem first class second class and how I can write down is a some sought of a tuple. So I can have for the first class I can have a 10 combination, for the second class I can have a 01 combination. And that is the way in which we are going to use the labels over here as well. So what I do is I read down one of these images and then after I have been able to read convert them onto a grey scale image and from there I find out certain co-occurrence matrix properties. So we will be computing a grey scale co-occurrence matrix of these features and then from there we will be finding out what is the contract of that co-occurrence matrix the co-relation energy homogeneity and these kind of factors coming up.

(Refer Slide Time: 23:29)



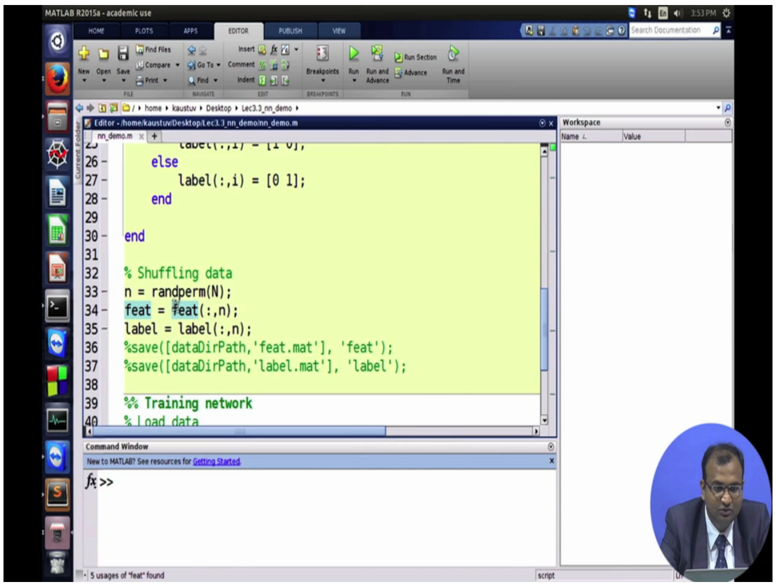
```
    % Re-rranging the features into a 1x4 vector
    feat1 = [glcm.Contrast;glcm.Correlation;glcm.Energy;glcm.H

    % Preparing data for training NN
    img_32 = imresize(gray_img,[32 32]); % Resizing the image
    img_32 = mat2gray(img_32); % Normalizing the image

    feat(:,i) = [img_32(:);feat1]; % 1028x1
    % Assigning class label assuming class 0 = healthy, class
    str1 = strsplit(filename,'_');
    str2 = strsplit(str1{1,2},'.');
    class = str2{1,1};
    if class == '0'
        label(:,i) = [1 0];
    else
```

```
fx >>
```

Now once that is done what I do is that you can actually now resize this whole image itself into a patch of 32 cross 32 and once that whole thing is done, now what you will be doing is you also along with this co-occurrence matrices, you push down that whole thing whole image itself as features over there. So each pixel of the image is also a feature as of now. Now, later on in the class we will again understand in a bit of more advance lectures we will get into fact like how this also helps over here. Initially it might appear a bit confusing from the fact that we were already extracting features and then why do I pushed down the whole thing, but there is a beauty which happens with that.

Now what we do after that is so this particular book keeping over there is just to find out the class labels because you do not have a separate file which says which image belongs to which class but it was the file name where the whole encoding was done. So this is just to find out which belongs to the first class and which belongs to the second class. Now from there once this is done we have already prepared our data.

(Refer Slide Time: 24:32)

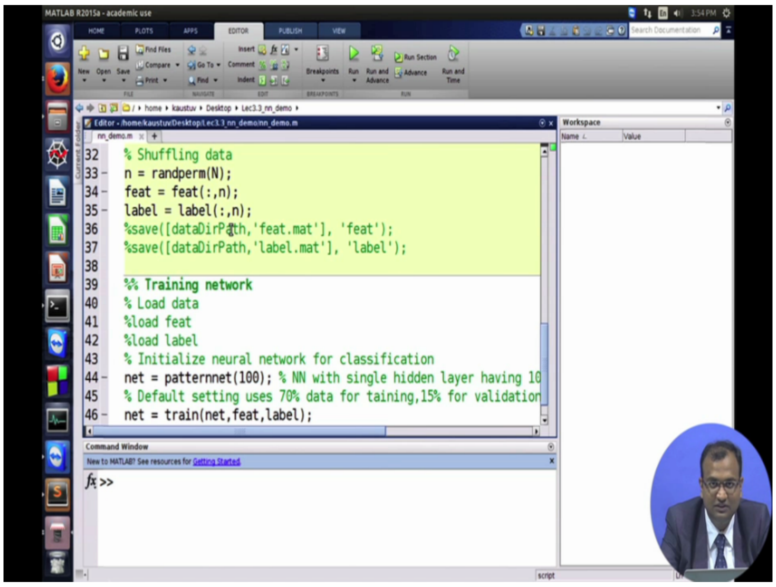


```
            label(:,i) = [1 0];
        else
            label(:,i) = [0 1];
        end

end

% Shuffling data
n = randperm(N);
feat = feat(:,n);
label = label(:,n);
%save([dataDirPath,'feat.mat'], 'feat');
%save([dataDirPath,'label.mat'], 'label');

%% Training network
% Load data
```

```
fx >>
```

Now the next part is over here what we do is, we randomly shuffle out the data and the reason for doing all of these shuffling is that we do not want to typically have a certain kind of ordering. So if you look at those images, the first few images were all belonging to the abnormal class and the last few images were all belonging to normal class. So now if I am somehow if I look at the whole gradient descent problem over there, so first I will be training it being very much bias towards the positive class, after that very much bias to the negative class which is normal, I do not want this kind of a bias creeping into my system and that is why we just we juggling around with the order in which they come.

(Refer Slide Time: 25:36)



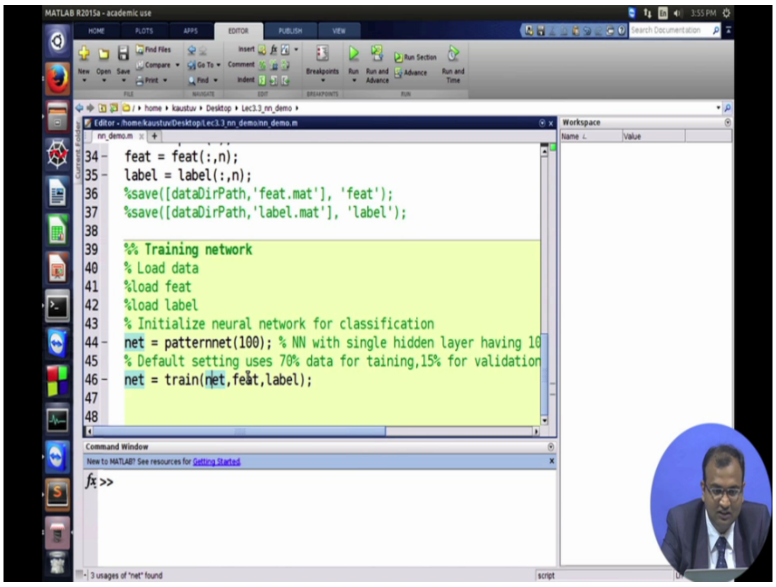
```
% Shuffling data
n = randperm(N);
feat = feat(:,n);
label = label(:,n);
%save([dataDirPath,'feat.mat'], 'feat');
%save([dataDirPath,'label.mat'], 'label');

%% Training network
% Load data
%load feat
%load label
% Initialize neural network for classification
net = patternnet(100); % NN with single hidden layer having 10
% Default setting uses 70% data for taining,15% for validation
net = train(net,feat,label);
```

```
New to MATLAB? See resources for Getting Started
fx >>
```

So there will be may be the first one or two of them from are positive class, the next three are from negative class, the next one is again positive, next one negative, so in any random order in which it comes, which will smooth down the way in which errors are computed and how the weights are trained over there. Now, once I have actually made all of this into one single linear array after this random permutation, now what I can do is this is one place where we were just trying to save it, you may choose to save the feature vectors as well for use sometime later on because every time you do a “randperm” it is going to randomly permuted in a different way. So you might like to preserve the way in which you are ordering was there. Now, what we do is here is where we start defining the network and it is a very simple,nd actually in Matlab, where you can define this as a pattern classification network we just has 100 neurons in one single hidden layer.

So, if I just want to change I can just make a, 10 so that would make down two different layers in which the first one has 100 neurons, the second one has 10 neurons over there. And by default this whole thing over here is where it is going to use just sigmoidal activation functions. You can again play around with them by just going into net dot activation function = and change all of these properties, you can just look into the documentation on much more details.

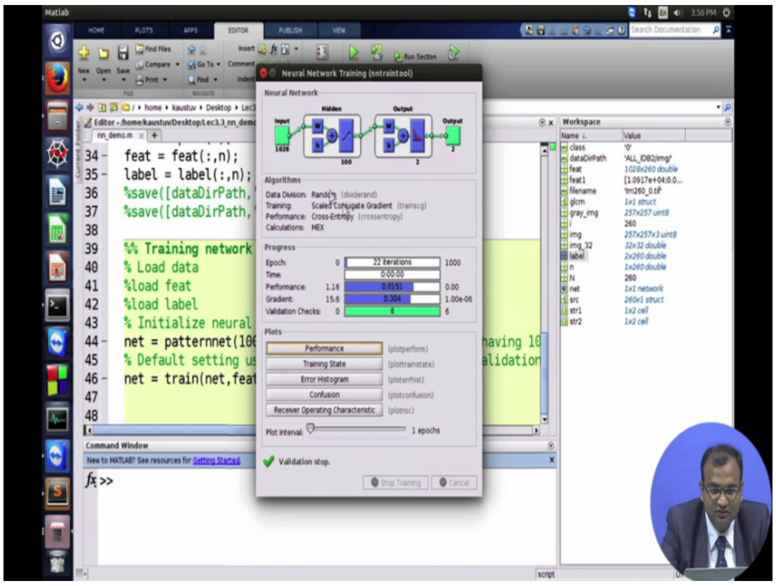
(Refer Slide Time: 26:36)



```
feat = feat(:,n);
label = label(:,n);
%save([dataDirPath,'feat.mat'], 'feat');
%save([dataDirPath,'label.mat'], 'label');

%% Training network
% Load data
%load feat
%load label
% Initialize neural network for classification
net = patternnet(100); % NN with single hidden layer having 10
% Default setting uses 70% data for taining,15% for validation
net = train(net,feat,label);
```

And then I start with the training in which what I do is I feed the network which is randomly initialized, I feed all the features which it has to use for it and I feed all the labels where it will be using in order to find out. So let us do a very simple run over this, since this is divided into modular cell so what I would be doing is I would first evaluate the first section which is about finding out the features. So now that we have all of the features coming down over here in this feeds and I also have my labels coming down over here in this label matrix, ok.

(Refer Slide Time: 27:29)

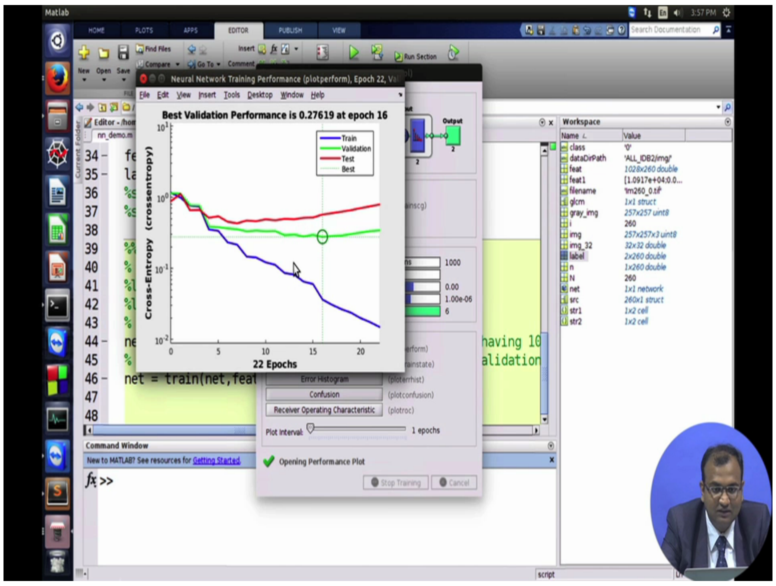


```
Neural Network Training (nntraintool)

Algorithms
Data Division:  Random  (dividerand)
Training:       Scaled Conjugate Gradient  (trainscg)
Performance:    Cross-Entropy  (crossentropy)
Calculations:   MEX

Progress
Epoch:              0     22 iterations   1000
Time:                     0:00:00
Performance:        1.16  0.0151          0.00
Gradient:           15.6  0.304           1.00e-06
Validation Checks:  0     6               6

Plot Interval:  1 epochs

Validation stop.
```

```
Workspace
Name         Value
class        '0'
dataDirPath  'ALL_IDB2/img/'
feat         1028x260 double
feat1        [1.0917e+04;0.0...
filename     'Im260_0.tif'
glcm         1x1 struct
gray_img     257x257 uint8
i            260
img          257x257x3 uint8
img_32       32x32 double
label        2x260 double
n            1x260 double
N            260
net          1x1 network
src          260x1 struct
str1         1x2 cell
str2         1x2 cell
```

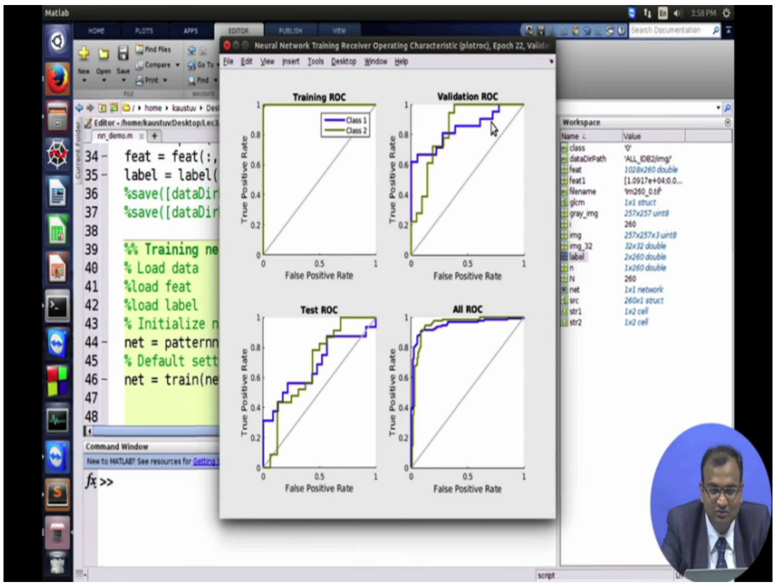
So I basically have 260 such samples over which this whole thing has been computed. Now with that I start with defining a network and let us train the whole network. So you see how fast this whole training has happened down, so this is where this whole network was created and you have a very graphical output about it, so we took down 1028 dimensional input and then we fed it down to 100 hidden layers and from there it was mapped down to an output which was just two vector output. And you can easily see that within this was set down for 1000 iterations over there in just 22 iterations or 22 epochs it has actually come down to the minimum point where it is supposed to saturate, ok. Now, let us look into that epoch versus performance graph and the training over here was based on other energy function which is called as cross entropy.

(Refer Slide Time: 28:10)


Neural Network Training Performance (plotperform), Epoch 22
Best Validation Performance is 0.27619 at epoch 16
Train
Validation
Test
Best
Cross-Entropy (crossentropy)
22 Epochs
net = train(net,feat
Opening Performance Plot
Stop Training
Cancel
Workspace

And you basically have 3 different graphs over here now what this graphs do is basically this blue one is called as the training graph so we were dividing it into 3 different batches of samples again and in that few of these samples were used for finding out the error over there for backpropogation. Based on backpropogating and updating the whole thing, we used another set which is called as a validation set on which we were finding out where should we stop down at a particular point of time. And there is another set which is called as a test set, which is basically used for testing over there when we do not use that error from the testing in order to do any updates or any changes onto the system. So this is one of them, you can actually click on each of them and have a different look. So you can open up the receiver updating characteristics and that could typically be showing you further three different sets.

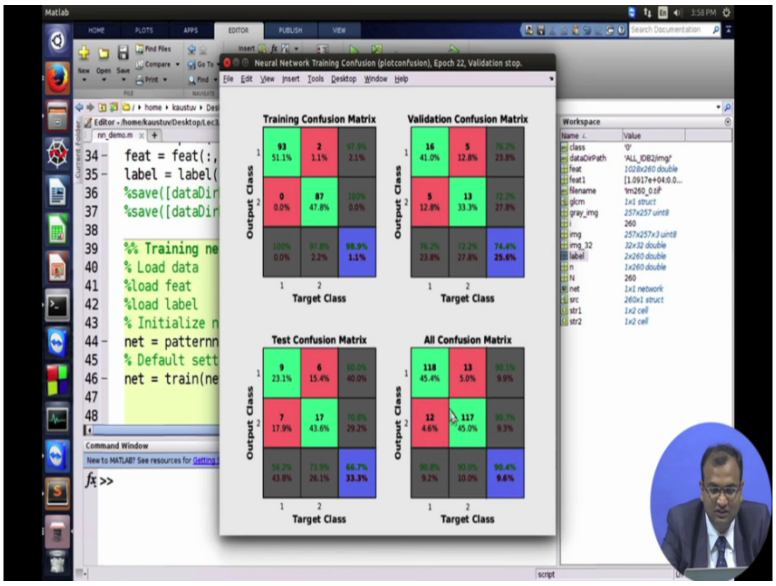
(Refer Slide Time: 29:02)


Neural Network Training Receiver Operating Characteristic (plotroc), Epoch 22, Valida

```
feat = feat(:,
label = label(
%save([dataDir
%save([dataDir

%% Training ne
% Load data
%load feat
%load label
% Initialize n
net = patternn
% Default sett
net = train(ne
```

So for the training one what is the ROC, for validation what is the ROC and for test what is the ROC and you would also be getting a 2 + problem over here and for that reason you have 2 different ROC matrix also coming down. Now from there we can also be looking at the confusion matrix which makes you very clear about how many false positives and false negatives, true positives and true negatives you are going to encounter and what is the total performance of the system as such.

(Refer Slide Time: 29:25)


Neural Network Training Confusion (plotconfusion), Epoch 22, Validation stop.

Training Confusion Matrix

Output Class	Target 1	Target 2	
1	93 51.1%	2 1.1%	97.9% 2.1%
2	0 0.0%	87 47.8%	100% 0.0%
	100% 0.0%	97.8% 2.2%	98.9% 1.1%

Validation Confusion Matrix

Output Class	Target 1	Target 2	
1	16 41.0%	5 12.8%	76.2% 23.8%
2	5 12.8%	13 33.3%	72.2% 27.8%
	76.2% 23.8%	72.2% 27.8%	74.4% 25.6%

Test Confusion Matrix

Output Class	Target 1	Target 2	
1	9 23.1%	6 15.4%	60.0% 40.0%
2	7 17.9%	17 43.6%	70.8% 29.2%
	56.2% 43.8%	73.9% 26.1%	66.7% 33.3%

All Confusion Matrix

Output Class	Target 1	Target 2	
1	118 45.4%	13 5.0%	90.1% 9.9%
2	12 4.6%	117 45.0%	90.7% 9.3%
	90.8% 9.2%	90.0% 10.0%	90.4% 9.6%

Now if you look over here, the total number of errors which is on the cross diagonal elements which are the false positives and the false negatives, you actually have a much lower quantity over there just 5 percent and 6 percent and 4.5, 6 percent of it. Majority of it which is about

95.4 percent is actually a true classification; about 90.5 percent is actually very true classification which happens. So you can see that very simple neural network we could be training it down just with 100 neurons on one single go and you have a 90 percent accuracy on classifying these patches of WBCs into either ALL or they are not, so with that we come to a conclusion about Neural Networks and that is all, thank you.