

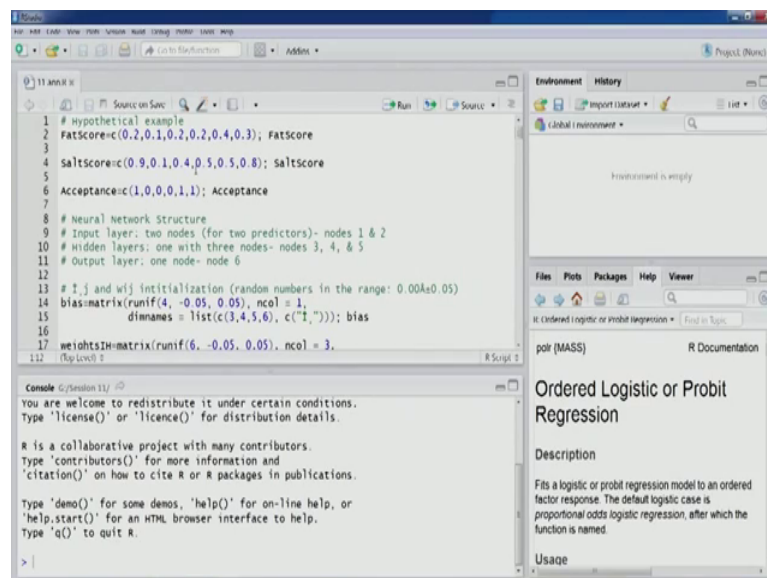
**Business Analytics & Data Mining Modeling Using R**  
**Dr. Gaurav Dixit**  
**Department of Management Studies**  
**Indian Institute of Technology, Roorkee**

**Lecture - 54**  
**Artificial Neural Network-Part II**

Welcome to the course Business Analytics and Data Mining Modeling Using R. So, in previous lecture we started our discussion on artificial neural network. So, we will continue that; so till now we have been we have been able to discuss the neural network, architecture, the background, multi layer feed forward, network and specific details about computations that are involved in input layer, hidden layers and output layer.

Now, what we will do; the this particular process, that is the computations that are involved in different layers. We will go through some of those things using an example using exercise in R. So, let us open R studio and the particular example this is a hypothetical one.

(Refer Slide Time: 01:09)



So, the example that we are going to is about is about a particular cheese combination. So, in a particular cheese we have these scores fat score and salt score. So, this is combination of is going to be tested by is has been tested by expert and the they indicate, whether that particular fat and salt score combination that particular fat and salt is combination is going to be accepted or not.

So, we have fat scores for every such experiment and the salt score and whether that was accepted or not. So, you can see we have just 6 observations. So, 6 observation in each of these vectors fat score, salt score and acceptance and this can be used to this is a small example, that we are going to use to understand certain computations that we discussed in the previous lectures.

(Refer Slide Time: 02:22)

```

1 # Hypothetical example
2 Fatscore=c(0.2,0.1,0.2,0.2,0.4,0.3); Fatscore
3
4 Saltscore=c(0.9,0.1,0.4,0.5,0.5,0.8); Saltscore
5
6 Acceptance=c(1,0,0,0,1,1); Acceptance
7
8 # Neural Network Structure
9 # Input layer: two nodes (for two predictors)- nodes 1 & 2
10 # Hidden layers: one with three nodes- nodes 3, 4, & 5
11 # Output layer: one node- node 6
12
13 # I, j and wij initialization (random numbers in the range: 0.00&0.05)
14 bias=matrix(runif(4, -0.05, 0.05), ncol = 1,
15             dimnames = list(c(3,4,5,6), c("I","j"))); bias
16
17 weightsIH=matrix(runif(6, -0.05, 0.05), ncol = 3,
18                  (Top Level)

```

Environment

Values
Fatscore num [1:6] 0.2 0.1 0.2 0.2 0.4 0.3

Console

```

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> Fatscore=c(0.2,0.1,0.2,0.2,0.4,0.3); Fatscore
[1] 0.2 0.1 0.2 0.2 0.4 0.3
>

```

Ordered Logistic or Probit Regression

Description

Fits a logistic or probit regression model to an ordered factor response. The default logistic case is proportional odds logistic regression, after which the function is named.

Usage

(Refer Slide Time: 02:31)

```

1 # Hypothetical example
2 Fatscore=c(0.2,0.1,0.2,0.2,0.4,0.3); Fatscore
3
4 Saltscore=c(0.9,0.1,0.4,0.5,0.5,0.8); Saltscore
5
6 Acceptance=c(1,0,0,0,1,1); Acceptance
7
8 # Neural Network Structure
9 # Input layer: two nodes (for two predictors)- nodes 1 & 2
10 # Hidden layers: one with three nodes- nodes 3, 4, & 5
11 # Output layer: one node- node 6
12
13 # I, j and wij initialization (random numbers in the range: 0.00&0.05)
14 bias=matrix(runif(4, -0.05, 0.05), ncol = 1,
15             dimnames = list(c(3,4,5,6), c("I","j"))); bias
16
17 weightsIH=matrix(runif(6, -0.05, 0.05), ncol = 3,
18                  (Top Level)

```

Environment

Values
Fatscore num [1:6] 0.2 0.1 0.2 0.2 0.4 0.3
Saltscore num [1:6] 0.9 0.1 0.4 0.5 0.5 0.8

Console

```

Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> Fatscore=c(0.2,0.1,0.2,0.2,0.4,0.3); Fatscore
[1] 0.2 0.1 0.2 0.2 0.4 0.3
> Saltscore=c(0.9,0.1,0.4,0.5,0.5,0.8); Saltscore
[1] 0.9 0.1 0.4 0.5 0.5 0.8
>

```

Ordered Logistic or Probit Regression

Description

Fits a logistic or probit regression model to an ordered factor response. The default logistic case is proportional odds logistic regression, after which the function is named.

Usage

So, let us create first variable fat score, so you can see fat score and the environment section we can see 6 observations; numeric vector. And then let us compute this salt score. So, again we can see 6 observations numeric vector and then the acceptance.

(Refer Slide Time: 02:39)

The screenshot shows the R Studio environment. The script editor contains the following code:

```

1 # Hypothetical example
2 Fatscore=c(0.2,0.1,0.2,0.2,0.4,0.3); Fatscore
3
4 Saltscore=c(0.9,0.1,0.4,0.5,0.5,0.8); Saltscore
5
6 Acceptance=c(1,0,0,0,1,1); Acceptance
7
8 # Neural network Structure
9 # Input layer: two nodes (for two predictors)- nodes 1 & 2
10 # Hidden layers: one with three nodes- nodes 3, 4, & 5
11 # Output layer: one node- node 6
12
13 # I,j and wij initialization (random numbers in the range: 0.00&0.05)
14 bias=matrix(runif(4, -0.05, 0.05), ncol = 1;
15             dimnames = list(c(3,4,5,6), c("I","J"))); bias
16
17 weightsIH=matrix(runif(6, -0.05, 0.05), ncol = 3;
18                  dimnames = list(c(3,4,5,6), c("I","J"))); weightsIH
19
20 # Weights for output layer
21 weightsOH=matrix(runif(1, -0.05, 0.05), ncol = 1;
22                  dimnames = list(c(6), c("O"))); weightsOH

```

The console shows the execution of the code:

```

> Fatscore=c(0.2,0.1,0.2,0.2,0.4,0.3); Fatscore
[1] 0.2 0.1 0.2 0.2 0.4 0.3
> Saltscore=c(0.9,0.1,0.4,0.5,0.5,0.8); Saltscore
[1] 0.9 0.1 0.4 0.5 0.5 0.8
> Acceptance=c(1,0,0,0,1,1); Acceptance
[1] 1 0 0 0 1 1

```

The environment pane shows the following variables:

Variable	Value
Acceptance	num [1:6] 1 0 0 0 1 1
Fatscore	num [1:6] 0.2 0.1 0.2 0.2 0.4 0.3
Saltscore	num [1:6] 0.9 0.1 0.4 0.5 0.5 0.8

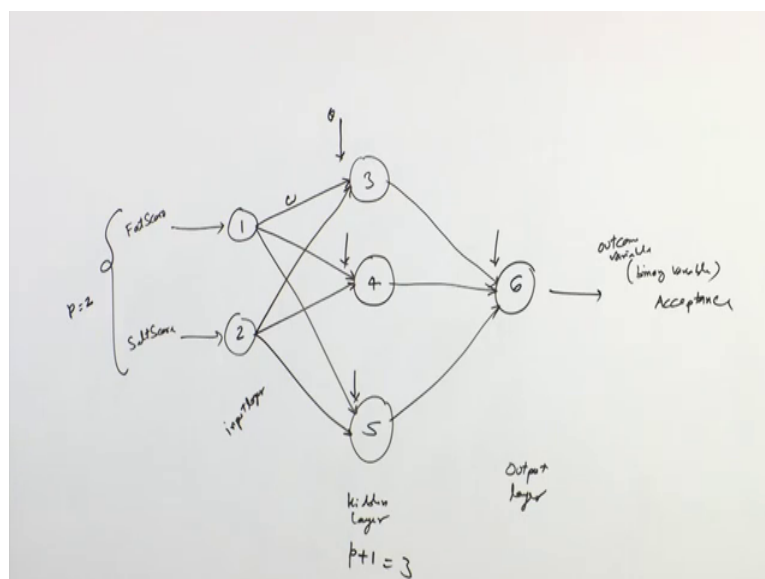
So, you can see salt score the acceptance 6 values 1 0 0 0 1 1. So, corresponding to each combination of fat and salt we have a value for acceptance, whether that was accepted or rejected.

Now, the typical example that the a specific example that we are going to follow is going to be based on this neural network structure. So, input layer will have two nodes, nodes 1 and 2, reason being we have two predictors here fat score and salt score. And hidden layers will have just one hidden layer with three nodes. So, that is one more than the number of predictors. So, p plus 1 n nodes in hidden layer.

So, they are going to be denoted by 3, 4 and 5. Then we have output layer of one node. So, that would be representing, because we have a you know binary variable here acceptance that is either 1 or 0, so one node for that; so that is denoted by node number 6.

So, if we want to draw the neural network architecture, that we have decided for this small exercises this one.

(Refer Slide Time: 03:55)



So, node 1 and 2, so node 1 and 2; so that is the values will come from this particular this particular predictor as we saw in our R environment, and then we have salt score and then these, so this is input layer. So, this is input layer now as we have discussed in previous lecture that input layer nodes all the all the input layer nodes they are going to provide input values to the next layer that is the first hidden layer. So, we had these predictors  $p$  value of  $p$  is 2.

So, typically around the number of predictors that we have; the same number of nodes we have input layer as we discussed and around the same number we have the number of nodes in the hidden layer. So, here we have just one hidden layer and the number of nodes that we are taking are  $p$  plus 1, otherwise here in this case three. So, this is no number 3, 4 and 5.

Now, as we talked about that from each node in the input layer we will have an arrow. So, this will provide a feed to this node as well as this node as well as this node. Similarly, from second you know from second node that is corresponding to the second predictor salt score will be providing feed to all three hidden layer nodes. So, you can see two arrows, because we had two predictors, two arrows coming to this particular this particular you know all the nodes of two arrows arriving at connecting to all the nodes in the hidden layer.

Then as we discussed that will have one node in the output layer. So, this is our output layer and this is the only hidden layer that we have. So, all the nodes in the hidden layer

are going to be connected to the output layer node the single output layer node that we have; so, they would be providing feed to the output layer single output layer node that we have, this is why we have just one node here. So, this is actually corresponding to the output variable that we have outcome variable, which is; which is binary variable in this case, which is binary variable in this case; which is acceptance. So, the feeds from all the hidden layer nodes would be provided, would be forwarded to this single output layer node. So, this is our typical neural network.

We will also have a bias values on each of the hidden layer nodes and output layer nodes. So, these are biased value. So, we will also have them. So, each of these nodes, we will have corresponding weights right. So, we will have these are bias values thetas and we will have weights for all the connected arrows. So, this is the typical neural network architecture that we are going to use for this example.

So, let us proceed. So, you can see first step is initialization. So, as you can see here in the comment that the theta and wij initialization that we have to perform first. So, this particular character this was actually theta; however, this probably this does not supported some problem here. So, this is being depicted by some other character, but this is; what we are talking about is the initialization of bias values that is thetas and weights. So, that is the typically that is first step.

So, bias values if we look at this particular function that we are using here is the matrix. So, we will compute a matrix of bias values, the first argument is of course, the data. So, we are using runif function to generate these random numbers. So, we are generating 4 random numbers, if we look at that we have three nodes in the hidden layer and one node in the output layer. So, we have 4 values we will have 4 bias values and therefore, I am generating 4 4 random numbers here.

And you can see the range of these numbers is minus 0.5 to 0.5. So, we will have that number of column is 1, so that is going to be representing all have all bias values. Dimension names are are 4, 3, 4, 5, 6, that is corresponding to node numbers 3, 4, 5, 6 for each of these node will have the row number and then we will have the 4 values corresponding values biased value. So, let us compute this.

(Refer Slide Time: 09:41)

```

4 SaltScore=c(0.9,0.1,0.4,0.5,0.8); SaltScore
5
6 Acceptance=c(1,0,0,1,1); Acceptance
7
8 # Neural network structure
9 # Input layer: two nodes (for two predictors)- nodes 1 & 2
10 # Hidden layers: one with three nodes- nodes 3, 4, & 5
11 # Output layer: one node- node 6
12
13 # i, j and wij initialization (random numbers in the range: 0.00&0.05)
14 bias=matrix(runif(4, -0.05, 0.05), ncol = 1,
15             dimnames = list(c(3,4,5,6), c("I","J"))); bias
16
17 weightsIH=matrix(runif(6, -0.05, 0.05), ncol = 3,
18                  dimnames = list(c(1,2), c(3,4,5))); weightsIH
19
20 weightsHO=matrix(runif(3, -0.05, 0.05), ncol = 1,
21                  dimnames = list(c(3,4,5,6), c("I","J"))); bias

```

```

> SaltScore=c(0.9,0.1,0.4,0.5,0.8); saltScore
[1] 0.9 0.1 0.4 0.5 0.8
> Acceptance=c(1,0,0,1,1); Acceptance
[1] 1 0 0 1 1
> bias=matrix(runif(4, -0.05, 0.05), ncol = 1,
+             dimnames = list(c(3,4,5,6), c("I","J"))); bias
+
+
+
+
+
+
3 -0.04593521
4 0.0202324
5 0.01181905
6 -0.04053740
>

```

So, you can see here; corresponding to node number 3, 4, 5, 6 which is which are being represented by row number here 3, 4, 5, 6 we have randomly initialized bias values which are from which range between minus 0.5 to 0.5.

Now, let us move forward to next step that is weights. So, first weights of arrows, which are connecting input layer nodes to hidden layer node; the single hidden layer node that we have. So, again here you would see that we have from we have two input layer nodes and three hidden layer nodes so; that means, we will have 2 into 3, 6 connecting arrows.

So, therefore, we will have to randomly initialize 6 values 6 of the weights first. So, again same range 6 values same range and we will have 3 columns 3 columns here corresponding to three nodes in the hidden layer nodes and we will have two rows corresponding to two nodes in the; input layer.

So, as you can see in the dimension names you can see first is for row names first element is for row names; first vector the second vector in this list is for column names. So, 1 and 2 2 nodes for input layer two and then three nodes in the hidden layer. So, in this fashion we will generate the randomly initialize the weights.

(Refer Slide Time: 11:17)

```

8 # Neural Network Structure
9 # Input layer: two nodes (for two predictors)- nodes 1 & 2
10 # Hidden layers: one with three nodes- nodes 3, 4, & 5
11 # Output layer: one node- node 6
12
13 # i, j and wij initialization (random numbers in the range: 0.00 to 0.05)
14 bias=matrix(runif(4, -0.05, 0.05), ncol = 1,
15             dimnames = list(c(3,4,5,6), c("1"))); bias
16
17 weightsIH=matrix(runif(6, -0.05, 0.05), ncol = 3,
18                  dimnames = list(c(1,2), c(3,4,5))); weightsIH
19
20 weightsHO=matrix(runif(3, -0.05, 0.05), ncol = 1,
21                  dimnames = list(c(3,4,5), c(6))); weightsHO
22
23 # Computing output value for first record
24
25

```

Console Output:

```

+ dimnames = list(c(3,4,5,6), c("1")); bias
1 1
3 -0.04593521
4 0.02022324
5 0.01181905
6 -0.04053740
+ weightsIH=matrix(runif(6, -0.05, 0.05), ncol = 3,
+                  dimnames = list(c(1,2), c(3,4,5))); weightsIH
1 3 4 5
1 0.04977568 -0.0294866 -0.02160449
2 -0.03303574 -0.0143802 0.02175594

```

Environment:

Object	Class	Attributes
bias	num	[1:4, 1] -0.0459 0.0
weightsIH	num	[1:2, 1:3] 0.0498 -0.
Acceptance	num	[1:6] 1 0 0 0 1 1
FatScore	num	[1:6] 0.2 0.1 0.2 0.2 ..
SaltScore	num	[1:6] 0.9 0.1 0.4 0.5 ..

Files Plots Packages Help Viewer

Ordered Logistic or Probit Regression

Description

Fits a logistic or probit regression model to an ordered factor response. The default logistic case is proportional odds logistic regression, after which the function is named.

Usage

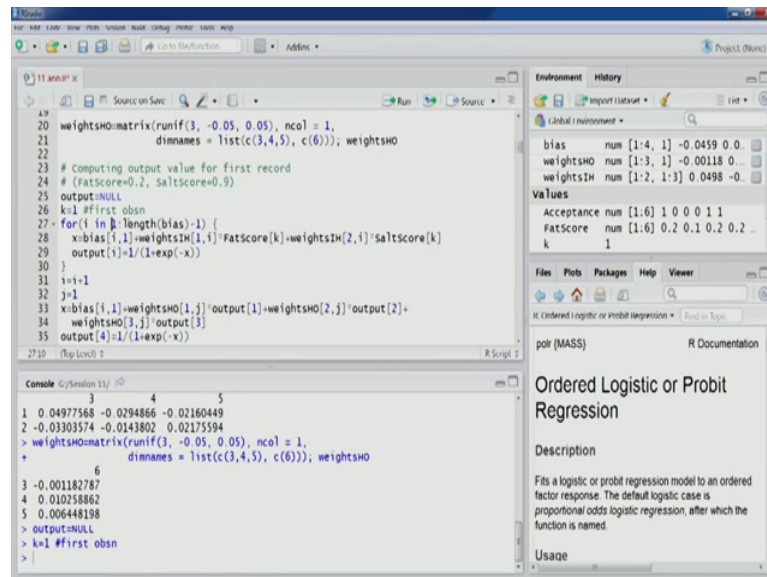
So, let us compute these values, you can see here in the output. So, row number correspond to input layer nodes that are 1 and 2, and the column names correspond to hidden layer nodes that are 3 and node number 3, 4 and 5 and you can see the values randomly initialized values in the range minus 0.5 to 0.5.

Then, let us move to next step that is initializing weight values that are that are there for the connections between hidden layer nodes to output layer nodes. So, we have three nodes in the hidden layer and just single hidden layer that we have; I am just single node in the output layer. So, we have 3 connecting rows 3 into 1, 3 connecting arrows and therefore, 3 weights we will have to initialize same you can see in this particular matrix function; 3 values will be in the same range one column, that is because we have just one node in the output layer.

So, you can see the dimension names, arrow names are corresponding to the hidden layer nodes that is 3, 4, 5 and the column name is corresponding to the output layer node that is 6 here.

(Refer Slide Time: 12:54)





So, let us compute this particular matrix and you can see in the output, that row numbers 3, 4, 5 responding to hidden layer nodes and the column name is 6, that is corresponding to our single output layer node that we have and these are the randomly initialized values.

So, once the random initialization has happened for all these  $w$ 's and  $\theta$ 's, then for a particular observation we can start certain computations. So, let us look at the first record that we have. So, first record that we have; you can have a look in the environment section as well, you can see fat score first value is 0.2 and salt score first value is 0.9. So, first record we have fat score as 0.2 and salt score as 0.9.

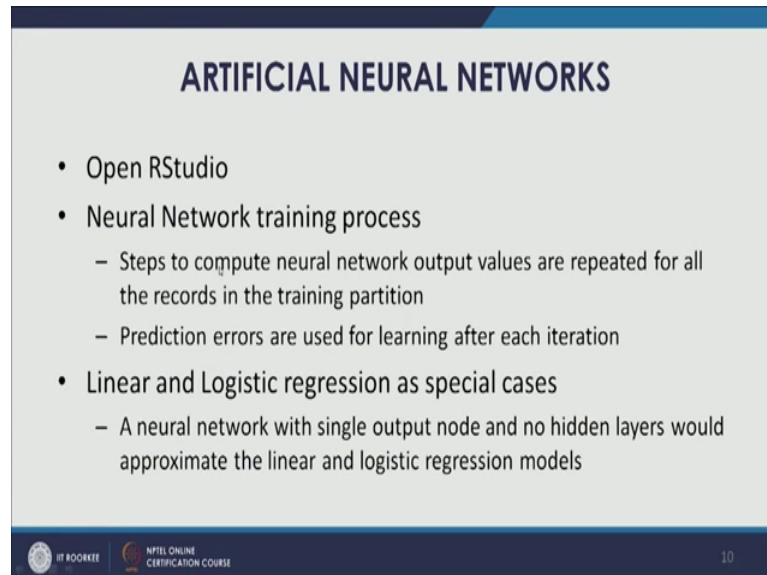
So, now, we will do certain computation. So, this is one variable output, where we are going to come store the output value values. So, any output values that are going to be processed after applying transfer function, they are going to be stored in this particular variable. So, let us initialize this marginal.

Now, because this is for first observation,  $k$  is 1; that is for first observation that is nothing, but to access the vectors of fat score and salt score, because the first value that we are taking here. So, let us initialize this now the loop; so this in loop runs from the you know all the all number of bias values and number of bias values and one less than the number of bias values. So, that is we can see that that, this is specifically for the hidden layer nodes we have three hidden layer nodes as you can see. So, this how we can



compute this. And now you can see the expression bias that expression that we saw in the slide. So, let us again have a look at that expression.

(Refer Slide Time: 14:35)

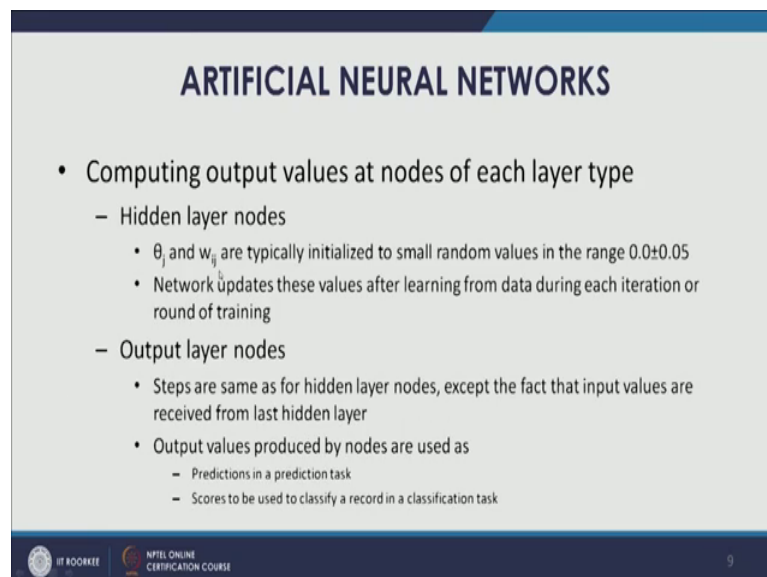


**ARTIFICIAL NEURAL NETWORKS**

- Open RStudio
- Neural Network training process
  - Steps to compute neural network output values are repeated for all the records in the training partition
  - Prediction errors are used for learning after each iteration
- Linear and Logistic regression as special cases
  - A neural network with single output node and no hidden layers would approximate the linear and logistic regression models

IT ROORKEE | NPTEL ONLINE CERTIFICATION COURSE 10

(Refer Slide Time: 14:36)



**ARTIFICIAL NEURAL NETWORKS**

- Computing output values at nodes of each layer type
  - Hidden layer nodes
    - $\theta_j$  and  $w_{ij}$  are typically initialized to small random values in the range  $0.0 \pm 0.05$
    - Network updates these values after learning from data during each iteration or round of training
  - Output layer nodes
    - Steps are same as for hidden layer nodes, except the fact that input values are received from last hidden layer
    - Output values produced by nodes are used as
      - Predictions in a prediction task
      - Scores to be used to classify a record in a classification task

IT ROORKEE | NPTEL ONLINE CERTIFICATION COURSE 9

This is we saw in a previous lecture.



(Refer Slide Time: 14:37)

## ARTIFICIAL NEURAL NETWORKS

- Computing output values at nodes of each layer type
  - Hidden layer nodes
    - Sum of bias value and weighted sum of input values received from previous layer is computed

$$\theta_j + \sum_{i=1}^p w_{ij}x_i$$

- Function g (referred as transfer function) is applied on this sum to produce the output values
- Transfer function could be a monotone function, for example:
  - Linear function:  $g(x) = bx$
  - Exponential function:  $g(x) = e^{bx}$
  - Logistic or sigmoidal function:  $g(x) = 1/(1+e^{-bx})$



8

So, you can see this is what we want to compute? We want to compute a weighted value. So, you can see theta value the bias value plus summation over all the predictors values the weighted average of all the predictors values.

So, we can see that bias value is being accessed, the bias value that we had initialized here right; starting from first value and then we can see weights then that we have initialized you can see the bias this one matrix first column first value right weights, again this was matrix; weights between input layer two hidden layer and you can see the you know a first row and then the we are going to through the i value is going to be 1. In the first iteration, so that is first column, then second column.

So, we would see in the output itself, if you go above here and bias values and then the weights IH; you can see the first column is corresponding to node number 3, second column is corresponding to node node number 4, and third column is corresponding to node number 5. So, you can see these column numbers are changing; however, we are dealing with the you know you know same row.

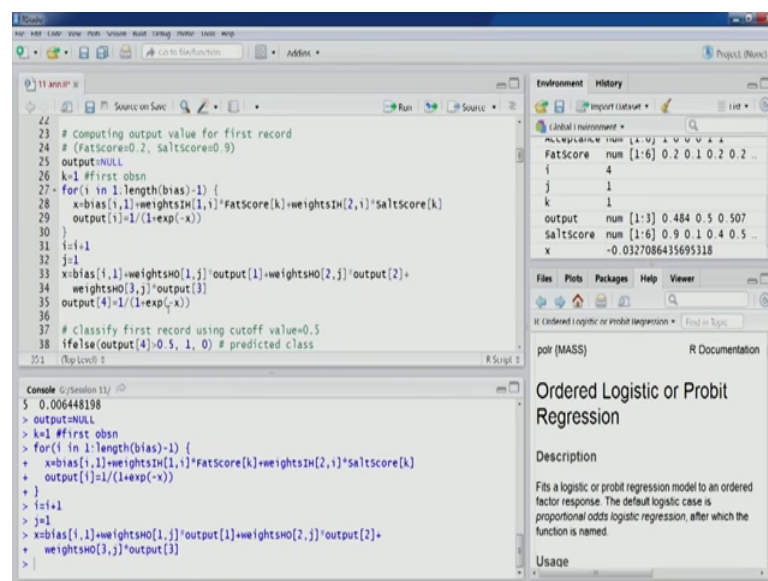
So, you can see row number 1 and here and for next one row number 2. So, first arrow, this is going to be the weight and then fat score and then, the second arrow this is the weight and for the same column and this is going to be computed. So, this expression is essentially being computed using this particular code and then the output values you can see the a logistic function here 1 divided by 1 plus exponential of minus x. So, we have

computed the logit value the logistic function and for all the nodes all the hidden layer nodes, this would be computed. So, let us run this loop.

Now, once this is computed, then we have one more output value to be computed that is for corresponding to the output layer node. So, let us increase increment the i value counter, and j also let us initialize this. Now you can see this particular code is for the output layer node biased value you can see here the bias values this was i we have already implemented.

So, therefore, the last pass value is going to be used here that is corresponding to the output layer node; then weight this is between hidden and output layer. So, the corresponding weight is being accessed from that matrix that we have and then that same expression the same expression that we have here is being evaluated here. So, let us compute this.

(Refer Slide Time: 17:49)



```

23 # computing output value for first record
24 # (Fatscore=0.2, saltscore=0.9)
25 output=NULL
26 k=1 #first obsn
27 for(i in 1:length(bias)-1) {
28   x=bias[i,1]+weightsIH[1,i]*Fatscore[k]+weightsIH[2,i]*saltscore[k]
29   output[i]=1/(1+exp(-x))
30 }
31 i=i+1
32 j=1
33 x=bias[i,1]+weightsHO[1,j]*output[1]+weightsHO[2,j]*output[2]+
34   weightsHO[3,j]*output[3]
35 output[4]=1/(1+exp(-x))
36
37 # classify first record using cutoff value=0.5
38 ifelse(output[4]>0.5, 1, 0) # predicted class

```

Console Output:

```

5 0.006448198
> output=NULL
> k=1 #first obsn
> for(i in 1:length(bias)-1) {
+ x=bias[i,1]+weightsIH[1,i]*Fatscore[k]+weightsIH[2,i]*saltscore[k]
+ output[i]=1/(1+exp(-x))
+ }
> i=i+1
> j=1
> x=bias[i,1]+weightsHO[1,j]*output[1]+weightsHO[2,j]*output[2]+
+ weightsHO[3,j]*output[3]
>

```

Environment:

Object	Class	Attributes
Fatscore	num	[1:6] 0.2 0.1 0.2 0.2 ..
i	int	4
j	int	1
k	int	1
output	num	[1:3] 0.484 0.5 0.507
saltscore	num	[1:6] 0.9 0.1 0.4 0.5 ..
x	num	-0.0327086435695318

Files | Plots | Packages | Help | Viewer

polr (MASS) R Documentation

### Ordered Logistic or Probit Regression

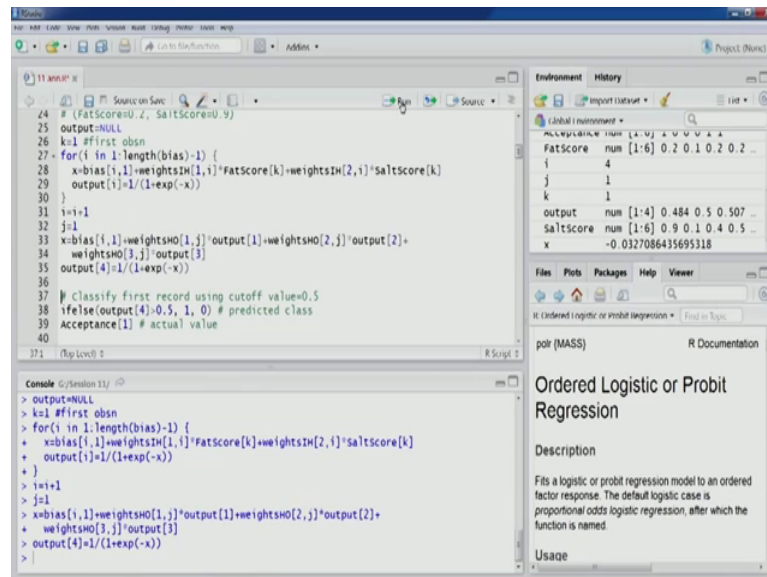
Description

Fits a logistic or probit regression model to an ordered factor response. The default logistic case is proportional odds logistic regression, after which the function is named.

Usage

Now, again we are using a logistic function to compute the value for the single output layer node.

(Refer Slide Time: 17:58)



So, now all the values output values have already been computed and; as you can also see when we computed the output value for the out single output layer node, the input values were the output of hidden layer you can see output 1 and output 2, output 3. In the loop you can see that input values were coming from input layer node, fat score here and salt score here.

These were the two nodes in the input layer and these values were being used. When we look at the you know output layer node, you can see the input values are coming are being used as the what output values of hidden layer are being becoming the input values for the output layer, and output 1 and output 2 and output 3 can be clearly seen here. And that is how we compute output for the output layer node. And once we have that value; so let us look at this value output 4; so the value that has been computed comes out to be 0.4918236.

(Refer Slide Time: 19:01)

```

27 for(i in 1:length(bias)-1) {
28   x=bias[i,1]+weightsIH[1,i]*Fatscore[k]+weightsIH[2,i]*Saltscore[k]
29   output[i]=1/(1+exp(-x))
30 }
31 i=i+1
32 j=1
33 x=bias[i,1]+weightsHO[1,j]*output[1]+weightsHO[2,j]*output[2]+
34   weightsHO[3,j]*output[3]
35 output[4]=1/(1+exp(-x))
36
37 # Classify first record using cutoff value=0.5
38 ifelse(output[4]>0.5, 1, 0) # predicted class
39 Acceptance[i] # actual value
40
41 # Model for hypothetical data
42 library(neuralnet)
43
3817 (Top Level) >

```

Environment History

Global Environment	num	[1:6]	0.2	0.1	0.2	0.2	...
Fatscore	num	[1:6]	0.2	0.1	0.2	0.2	...
i	4						
j	1						
k	1						
output	num	[1:4]	0.484	0.5	0.507	...	
Saltscore	num	[1:6]	0.9	0.1	0.4	0.5	...
x			-0.0327086435695318				

Files Plots Packages Help Viewer

Ordered Logistic or Probit Regression

Description

Fits a logistic or probit regression model to an ordered factor response. The default logistic case is proportional odds logistic regression, after which the function is named.

Usage

```

> for(i in 1:length(bias)-1) {
+   x=bias[i,1]+weightsIH[1,i]*Fatscore[k]+weightsIH[2,i]*Saltscore[k]
+   output[i]=1/(1+exp(-x))
+ }
> i=i+1
> j=1
> x=bias[i,1]+weightsHO[1,j]*output[1]+weightsHO[2,j]*output[2]+
+   weightsHO[3,j]*output[3]
+ output[4]=1/(1+exp(-x))
> output[4]
[1] 0.4918236
>

```

And, now as we know that the acceptance variable that outcome variable that we have that is; essentially a binary variable 0 and 1 whether that particular combination was accepted or not. So, this value; this score can be used you know compared with a cut off score that is we can take 0.5 as the cut off score and it can be compared to classify this observation, whenever this is the code that we are using.

(Refer Slide Time: 19:44)

```

27 for(i in 1:length(bias)-1) {
28   x=bias[i,1]+weightsIH[1,i]*Fatscore[k]+weightsIH[2,i]*Saltscore[k]
29   output[i]=1/(1+exp(-x))
30 }
31 i=i+1
32 j=1
33 x=bias[i,1]+weightsHO[1,j]*output[1]+weightsHO[2,j]*output[2]+
34   weightsHO[3,j]*output[3]
35 output[4]=1/(1+exp(-x))
36
37 # Classify first record using cutoff value=0.5
38 ifelse(output[4]>0.5, 1, 0) # predicted class
39 Acceptance[i] # actual value
40
41 # Model for hypothetical data
42 library(neuralnet)
43
391 (Top Level) >

```

Environment History

Global Environment	num	[1:6]	0.2	0.1	0.2	0.2	...
Fatscore	num	[1:6]	0.2	0.1	0.2	0.2	...
i	4						
j	1						
k	1						
output	num	[1:4]	0.484	0.5	0.507	...	
Saltscore	num	[1:6]	0.9	0.1	0.4	0.5	...
x			-0.0327086435695318				

Files Plots Packages Help Viewer

Ordered Logistic or Probit Regression

Description

Fits a logistic or probit regression model to an ordered factor response. The default logistic case is proportional odds logistic regression, after which the function is named.

Usage

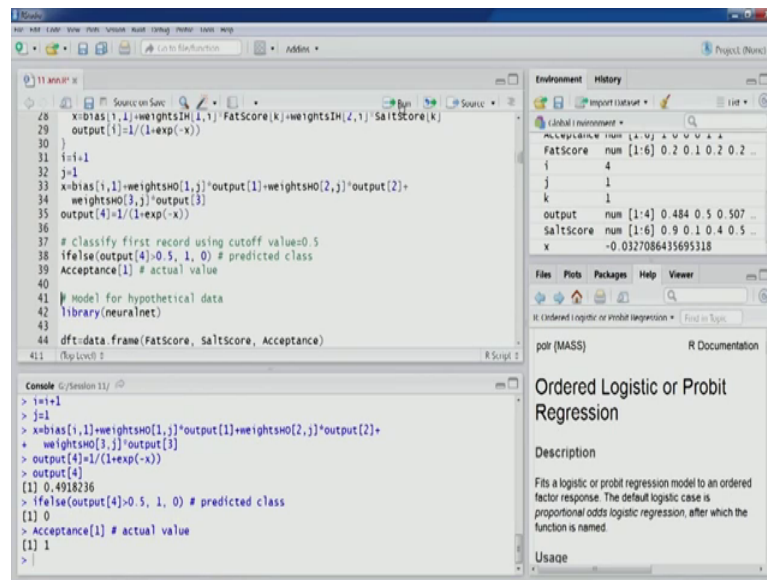
```

+   output[i]=1/(1+exp(-x))
+ }
> i=i+1
> j=1
> x=bias[i,1]+weightsHO[1,j]*output[1]+weightsHO[2,j]*output[2]+
+   weightsHO[3,j]*output[3]
+ output[4]=1/(1+exp(-x))
> output[4]
[1] 0.4918236
> ifelse(output[4]>0.5, 1, 0) # predicted class
[1] 0
>

```

So, if else if this output value is greater than 0.5, then classified to that at row 1; that means; accepted otherwise 0; so we can compute this value; so this comes out to be 0, because the value is 0.49 quite close to 0.5, but less than that. So, this has been classified as 0; however, quite close if you look at the actual value acceptance it was 1.

(Refer Slide Time: 19:58)



The screenshot shows the R Studio interface. The script editor on the left contains the following code:

```
28 x=bias[1,1]+weightsIH[1,1]*Fatscore[k]+weightsIH[2,1]*Saltscore[k]
29 output[1]=1/(1+exp(-x))
30 }
31 i=i+1
32 j=1
33 x=bias[1,1]+weightsHO[1,j]*output[1]+weightsHO[2,j]*output[2]+
34 weightsHO[3,j]*output[3]
35 output[4]=1/(1+exp(-x))
36 }
37 # Classify first record using cutoff value=0.5
38 ifelse(output[4]>0.5, 1, 0) # predicted class
39 Acceptance[1] # actual value
40 }
41 # Model for hypothetical data
42 library(neuralnet)
43
44 dft=data.frame(Fatscore, Saltscore, Acceptance)
45
```

The console on the bottom left shows the execution of the first iteration:

```
> i=i+1
> j=1
> x=bias[1,1]+weightsHO[1,j]*output[1]+weightsHO[2,j]*output[2]+
+ weightsHO[3,j]*output[3]
> output[4]=1/(1+exp(-x))
> output[4]
[1] 0.4918236
> ifelse(output[4]>0.5, 1, 0) # predicted class
[1] 0
> Acceptance[1] # actual value
[1] 1
>
```

The Environment pane on the right shows the following objects:

Object	Class	Attributes
Fatscore	num	[1:6] 0.2 0.1 0.2 0.2 ...
Saltscore	num	[1:6] 0.9 0.1 0.4 0.5 ...
Acceptance	num	[1:6] 0.484 0.5 0.507 ...
x	num	[1:6] -0.03270864 35695318

The right pane also shows the documentation for the `poir` function, titled "Ordered Logistic or Probit Regression".

So, the value that was computed here is 0.49. So, there is still some it was still not classified as you know class 1. So, therefore, what we can understand is; more iterations are to be performed, this neural network that we have this is what this was the first iteration that we had done.

So, therefore, more iteration have to be performed for us to be able to have the final model, which will have higher accuracy, higher classification accuracy. So, this was just first iteration. So, more iteration the network will learn something more from the data and then probably the performance will improve the actual you know results would start matching the actual values.

So, let us go back to our discussion. So, as you can see whatever discussion that we had in previous lecture the computations that we talked about in previous lecture, this particular expression as well. So, that we now saw how it are going to be performed you know in R environment R studio environment.

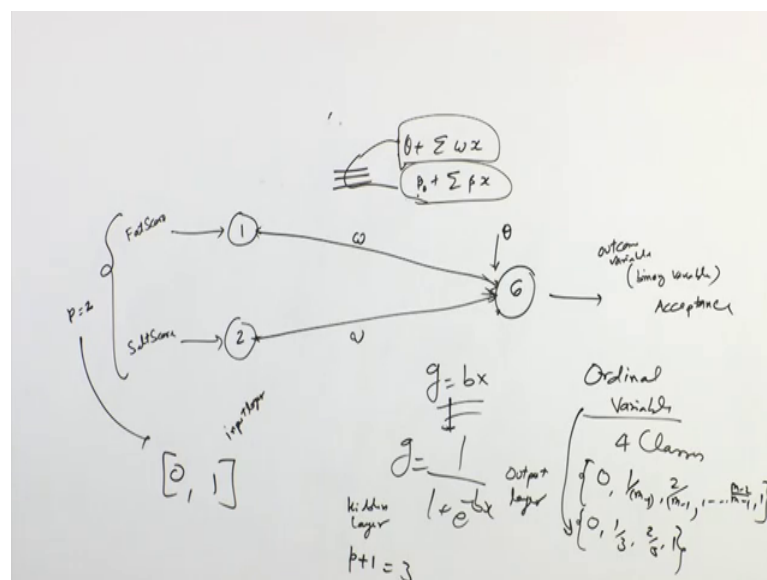
Now, let us now talk about the next important point here, we talked about in previous lecture that linear and logistic regression can be treated as a special cases of neural network, how is that how can that happen; so let us discuss. So, let us consider a neural network with single output node and no hidden layers and that would actually approximate the linear and logistic regression models right.

So, as we talked about that expression, that we use the expression that we use for weighing the inputs here right; the expression that we used here this  $\theta$  plus summation of  $w$  and  $x_i$  predictor values, this is quite similar to what we have in linear regression the  $\beta$  plus you know  $\beta_0$  plus here we have summation  $\beta$  and  $x$  this is quite similar to what we have in linear regression.

So, therefore, in that sense we can try and approximate we can try an approximate linear regression as a special case of neural network what we will do we will have 0 hidden layers. So, let us zero down layer single output node that we already have. So, if the same diagram we want to convert into a linear regression this is what we can probably do.

So, if we remove the hidden layer, if we remove the hidden layer the input layer nodes are going to be directly connected with the output layer node right; directly going to be connected with the output layer node, and if we are not using; if we are using the transfer function, so these arrows will also change this will also go. So, these arrows will also change.

(Refer Slide Time: 23:21)



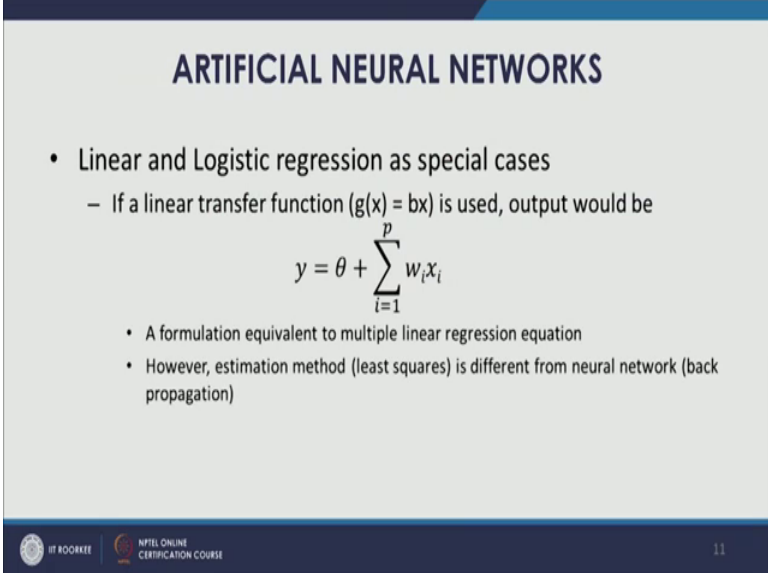
So, this will become there will be a feed to this from this node and there is going to be a feed to this. Let us remove some of these things. So, this is what we will have and this is  $\theta$  this is going to be. So, they are going to be weights. So, we have two predictors and one output variable and if you look at; now there is similarity, transfer function if  $g$  is



something like this, then the input values that receive the same are going to be transferred here and therefore, this is what we will have.

So, this will approximate what we talked about this will approximate the linear regression. So, as we can see in the slide as well if a linear transfer function  $g(x) = bx$  is used so; that means, the input values that we receive from predictors the same input values are going to be fed to the output layer node there are no hidden layer nodes, then the formulation is going to be equivalent to that of a linear linear regressions you can see these formulations.

(Refer Slide Time: 24:20)



**ARTIFICIAL NEURAL NETWORKS**

- Linear and Logistic regression as special cases
  - If a linear transfer function ( $g(x) = bx$ ) is used, output would be
$$y = \theta + \sum_{i=1}^p w_i x_i$$
  - A formulation equivalent to multiple linear regression equation
  - However, estimation method (least squares) is different from neural network (back propagation)

IT ROOKIE | NPTEL ONLINE CERTIFICATION COURSE | 11

So, this formulation will become equivalent to the formulation that we have in linear regression. However, the estimation method how do we estimate how you know we estimate the beta's in linear regression, that is typically done using least square that is different in case of neural network. Neural network we apply back propagation algorithm to estimate theta thetas and w.

So, theta and w's values are estimated using back propagation algorithm neural network; however, in linear regression these betas are estimated using least square. So, estimation method is different; otherwise, we can approximate the linear regression using neural network. So, therefore, we can say that in a way linear regression or a special cases of neural network.

The similar thing similar conceptualization we can do for logistic regression as well.

(Refer Slide Time: 25:44)

The slide is titled "ARTIFICIAL NEURAL NETWORKS" in a bold, dark blue font. Below the title, there is a bulleted list. The first bullet point is "Linear and Logistic regression as special cases". The second bullet point is "If a logistic transfer function ( $g(x) = 1/(1+e^{-bx})$ ) is used, output would be". Below this, the probability equation is shown: 
$$P(y = 1) = \frac{1}{1 + e^{\theta + \sum_{i=1}^p w_i x_i}}$$
. There are two more bullet points: "A formulation equivalent to logistic regression equation" and "However, estimation method (maximum-likelihood method) is different from neural network (back propagation)". At the bottom of the slide, there are logos for "IIT ROORKEE" and "NPTEL ONLINE CERTIFICATION COURSE", and the number "12" in the bottom right corner.

**ARTIFICIAL NEURAL NETWORKS**

- Linear and Logistic regression as special cases
  - If a logistic transfer function ( $g(x) = 1/(1+e^{-bx})$ ) is used, output would be
$$P(y = 1) = \frac{1}{1 + e^{\theta + \sum_{i=1}^p w_i x_i}}$$
    - A formulation equivalent to logistic regression equation
    - However, estimation method (maximum-likelihood method) is different from neural network (back propagation)

IIT ROORKEE NPTEL ONLINE CERTIFICATION COURSE 12

Suppose this transfer function is logistic function, if this transfer function is logistic function, then you would see that you would see that the neural network with zero hidden layer nodes zero hidden layers would again approximate the logistic regression equation. So, you can see as in the slide probability of a particular record belonging to class 1 and this is you know they are also in logistic regression also we use this logistic response function and we will have expression like this here.

So, that will; so this will actually approximate this will actually approximate the logistic regression. So, if the formulation is equivalent to what we have in logistic regression equation, how well just like linear regression; the estimation method is different there in logistic regression that is the maximum likelihood method that is typically used in logistic, and in neural network as I talked about typically back propagation is used. So, linear and logistic regression they both can be conceptualized as a special cases of neural network.

Let us move forward; so some more important points with respect to artificial neural network, so one is normalization.

(Refer Slide Time: 27:17)

# ARTIFICIAL NEURAL NETWORKS

- Normalization

- Scale of [0,1] is typically recommended for neural network models for performance purposes
- For numeric variables,

$$V_{norm} = \frac{V - \min(V)}{\max(V) - \min(V)}$$

- Binary variables (categorical variables with two classes)
  - Create dummy variables: set of values {0, 1}
- Nominal variables with m (>2) classes
  - Create m-1 dummy variables: set of values {0, 1}
- Ordinal variables with m (>2) classes
  - Map the values to the set {0, 1/(m-1), 2/(m-1), ..., (m-2)/(m-1), 1}



Typically, the amount of iterations that are performed in a neural network depending on the number of observations and the depending on the learning rates and other things that we will discuss later on. So, depending on that quite you know number of computations number of computations or computational intensity of a artificial neural network could be quite high. So, therefore, to boost the performance to get the converging convergence in neural network and to also get better performance; it is generally recommended that all the variables should be in the scale of 0, 1.

So, all the predictors; so we would like to have all the predictors in this scale 0 to 1. So, normalization, so we would be required to perform normalization. So, that all the variables are in that scale. So, for numeric variables as you can see in the slide the normalized variable  $V_{norm}$  could be computed in this fashion  $V$  minus minimum of  $V$  divided by  $\max V$  minus  $\min V$ . So, this will give us a normalized variable, which will have values in this particular range 0 to 1. So, this is for numeric variables.

If we talk about the binary variables the categorical variable with two classes; So, typically there is not much that we need to do; if we create dummy variables, so they will anyway have take values set of values is going to be 0 and 1. So, either 0 or 1 values is going to be taken for all the observations. So, binary variable will work just fine irrespective of whether they are ordinal or nominal. So, binary variables are going to be with this range this normalization.

We talked about the nominal variables with greater than two classes, then we can create  $m - 1$  dummy variables and because these are dummy variables again they will have this set of values 0 and 1. So, values could be going to be either 0 or 1. So, that is also ok.

Now, when we talk about the ordinal variables ordinal variables with  $m$  classes, where  $m$  is greater than 2, then we have to think about what can be done. So, typically the values can be mapped to this particular set of values. So, 0 comma 1 divided by  $m - 1$ , so if there are  $m$  classes; so we are dividing 1 by  $m - 1$ , then 2 divided by  $m - 1$ , then up to  $m - 2$  divided by  $m - 1$  and 1. So, this is to map the values.

So, if there are let us say; if there are 4 classes if we have an ordinal variable with 4 classes. So, we would like to as discussed in slide. So, you would see that, we would like to have in this fashion the values as mentioned in the slide. Now for four classes that scenario would be 0, 1 divided by 3; 2 divided by 3 and then 1.

So, these could be the four values for the ordinal classes, now the values that could be there in the variable, they will have to mapped to these four values right. And we have to change the variable type as ordinal and have these values. So, the scale will again be 0 to 1, and since this is going to be anyway ordinal variable. So, so the values are also going to be in this range and it can be used here.

So, these transformations can be performed or are actually recommended for in your work to achieve either the conversion of the network, conversion of the model or to even improve the performance. There are few more considerations more discussions point about artificial neural network, that we will discuss in the next lecture. So, we will stop here.

Thank you.