statinfer

Neural Networks in R

Venkat Reddy

Statinfer.com

Data Science Training and R&D

Training on

Data Science

Bigdata Analytics

Machine Learning

Predictive Modelling

Deep Learning

Corporate Training

Classroom Training

Online Training

Contact us

info@statinfer.com

venkat@statinfer.com

Note

- This presentation is just my class notes. The course notes for data science training is written by me, as an aid for myself.
- The best way to treat this is, as a high-level summary; the actual session went more in depth and contained detailed information and examples
- Most of this material was written as informal notes, not intended for publication
- Please send questions/comments/corrections to info@statinfer.com
- Please check our website statinfer.com for latest version of this document

-Venkata Reddy Konasani

(Cofounder statinfer.com)

Contents

Contents

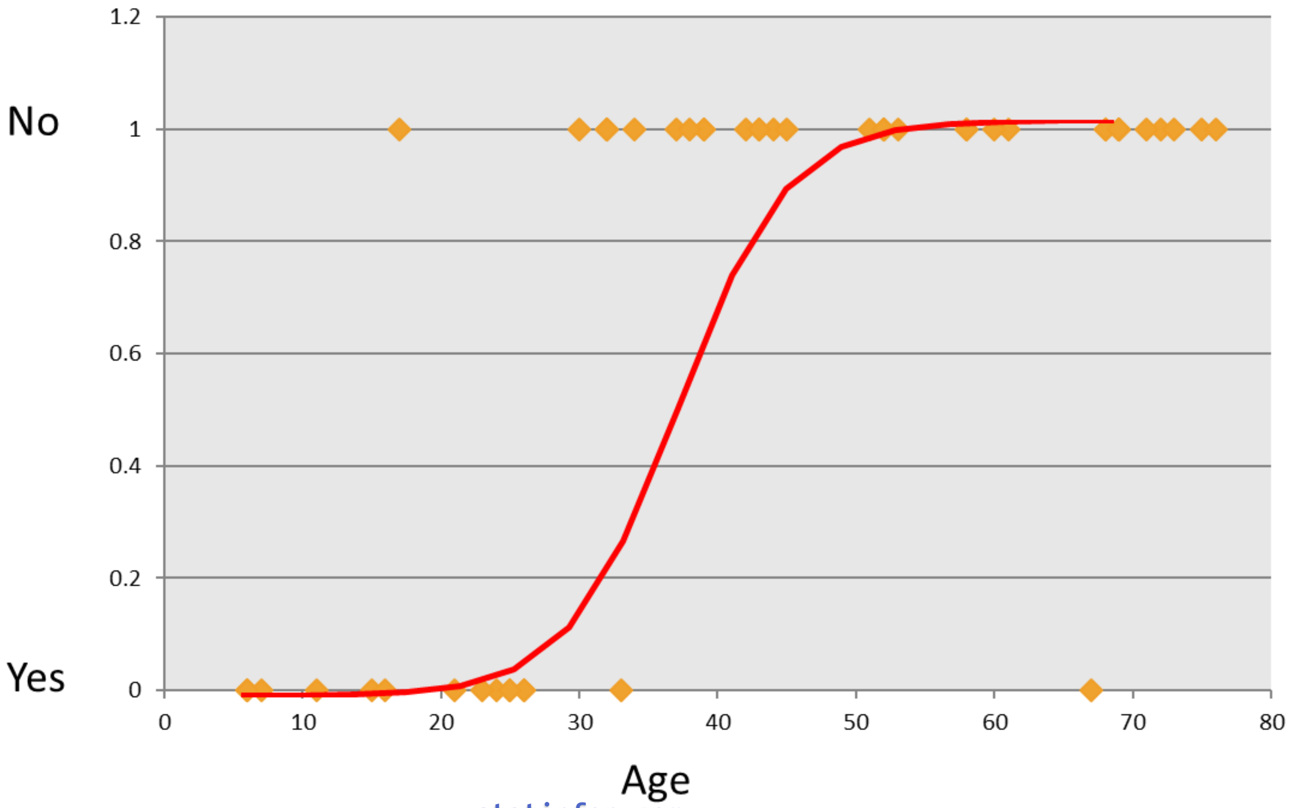
- Neural network Intuition
- Neural network and vocabulary
- Neural network algorithm
- Math behind neural network algorithm
- Building the neural networks
- Validating the neural network model
- Neural network applications
- Image recognition using neural networks

statinfer

Recap of Logistic Regression

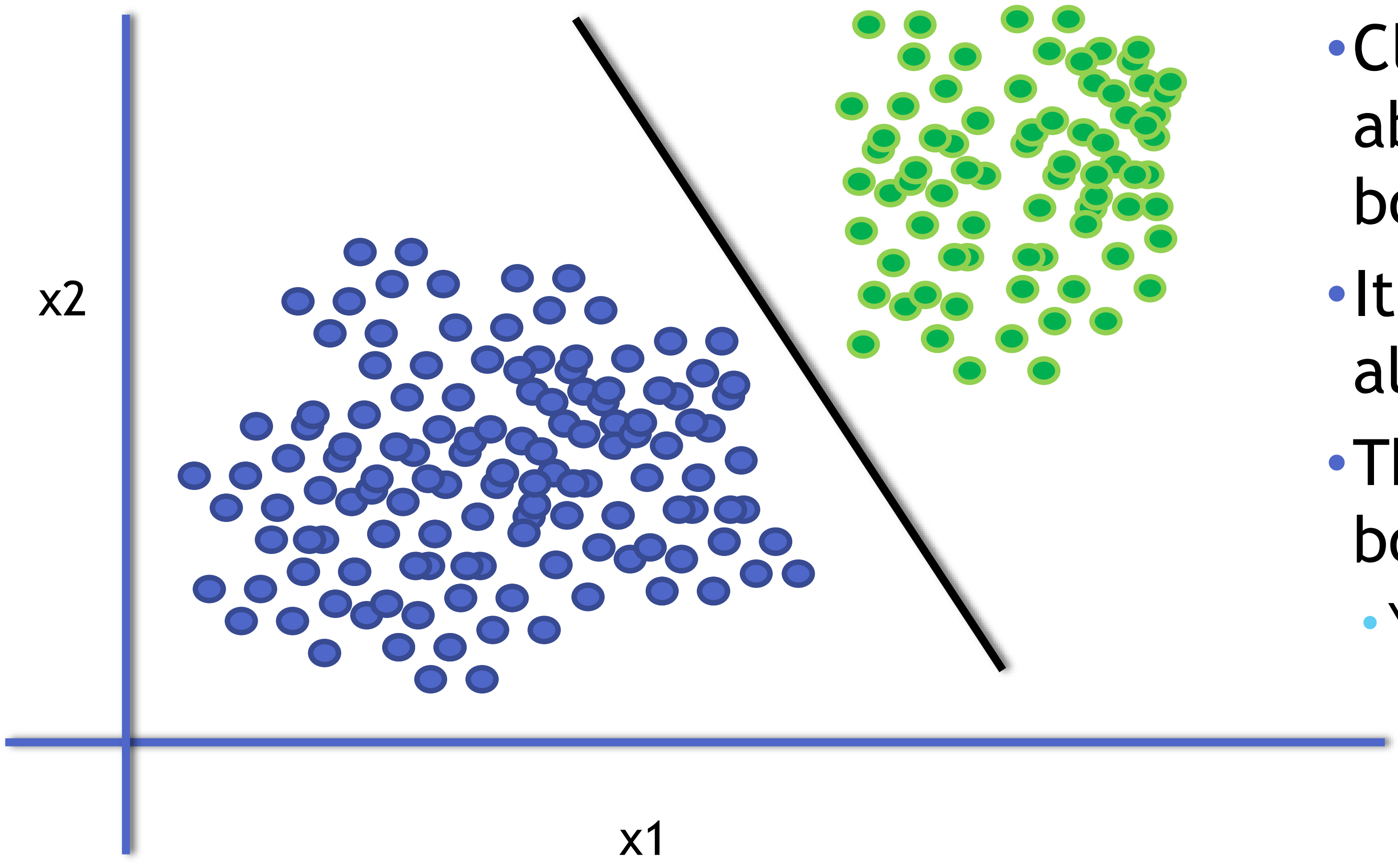
Recap of Logistic Regression

- Categorical output YES/NO type
- Using the predictor variables to predict the categorical output



Decision Boundary

Decision Boundary – Logistic Regression



- The line or margin that separates the classes
- Classification algorithms are all about finding the decision boundaries
- It need not be straight line always
- The final function of our decision boundary looks like
  - Y=1 if $w^Tx+w_0>0$; else Y=0

Decision Boundary – Logistic Regression

- In logistic regression, Decision Boundary can be derived from the logistic regression coefficients and the threshold.
  - Imagine the logistic regression line $p(y)=e^{(b0+b1x1+b2x2)}/1+exp^{(b0+b1x1+b2x2)}$
  - Suppose if p(y)>0.5 then class-1 or else class-0
    - $\log(y/1-y)=b_0+b_1x_1+b_2x_2$
    - $\text{Log}(0.5/0.5)=b_0+b_1x_1+b_2x_2$
    - $0=b_0+b_1x_1+b_2x_2$
    - $b_0+b_1x_1+b_2x_2=0$ is the line

Decision Boundary – Logistic Regression

- Rewriting it in mx+c form
  - $X_2=(-b_1/b_2)X_1+(-b_0/b_2)$
- Anything above this line is class-1, below this line is class-0
  - $X_2>(-b_1/b_2)X_1+(-b_0/b_2)$is class-1
  - $X_2<(-b_1/b_2)X_1+(-b_0/b_2)$ is class-0
  - $X_2=(-b_1/b_2)X_1+(-b_0/b_2)$ tie probability of 0.5
- We can change the decision boundary by changing the threshold value(here 0.5)

statinfer

LAB: Logistic Regression and Decision Boundary

LAB: Logistic Regression

- Dataset: Emp_Productivity/Emp_Productivity.csv
- Filter the data and take a subset from above dataset . Filter condition is Sample_Set<3
- Draw a scatter plot that shows Age on X axis and Experience on Y-axis. Try to distinguish the two classes with colors or shapes (visualizing the classes)
- Build a logistic regression model to predict Productivity using age and experience
- Create the confusion matrix
- Calculate the accuracy and error rates

LAB: Decision Boundary

- Draw a scatter plot that shows Age on X axis and Experience on Y-axis. Try to distinguish the two classes with colors or shapes (visualizing the classes)
- Build a logistic regression model to predict Productivity using age and experience
- Finally draw the decision boundary for this logistic regression model

Code: Logistic Regression

```
Emp_Productivity_raw <- read.csv("D:\\Google Drive\\Training\\Datasets\\Emp_Productivity\\Emp_Productivity.csv")

#############################################
####Sample-1
#############################################
Emp_Productivity1<-Emp_Productivity_raw[Emp_Productivity_raw$Sample_Set<3,]
```

```
> dim(Emp_Productivity1)
[1] 74  4
> names(Emp_Productivity1)
[1] "Age"          "Experience"   "Productivity" "Sample_Set"
> head(Emp_Productivity1)
   Age Experience Productivity Sample_Set
1 20.0        2.3            0          1
2 16.2        2.2            0          1
3 20.2        1.8            0          1
4 18.8        1.4            0          1
5 18.9        3.2            0          1
6 16.7        3.9            0          1
>
> table(Emp_Productivity1$Productivity)

 0  1
33 41
>
> ####The clasification graph Sample-1
> library(ggplot2)
> ggplot(Emp_Productivity1)+geom_point(aes(x=Age,y=Experience,color=factor(Productivity),shape=factor(Productivity)),size=5)
```

Code: Logistic Regression

Code: Logistic Regression

```
> ###Logistic Regerssion model1
> Emp_Productivity_logit<-glm(Productivity~Age+Experience,data=Emp_Productivity1, family=binomial())
> Emp_Productivity_logit

Call:  glm(formula = Productivity ~ Age + Experience, family = binomial(),
    data = Emp_Productivity1)

Coefficients:
(Intercept)          Age   Experience
    -8.9361       0.2763       0.5923

Degrees of Freedom: 73 Total (i.e. Null);  71 Residual
Null Deviance:      101.7
Residual Deviance: 46.77        AIC: 52.77
> coef(Emp_Productivity_logit)
(Intercept)         Age  Experience
 -8.9361114   0.2762749   0.5923444
```

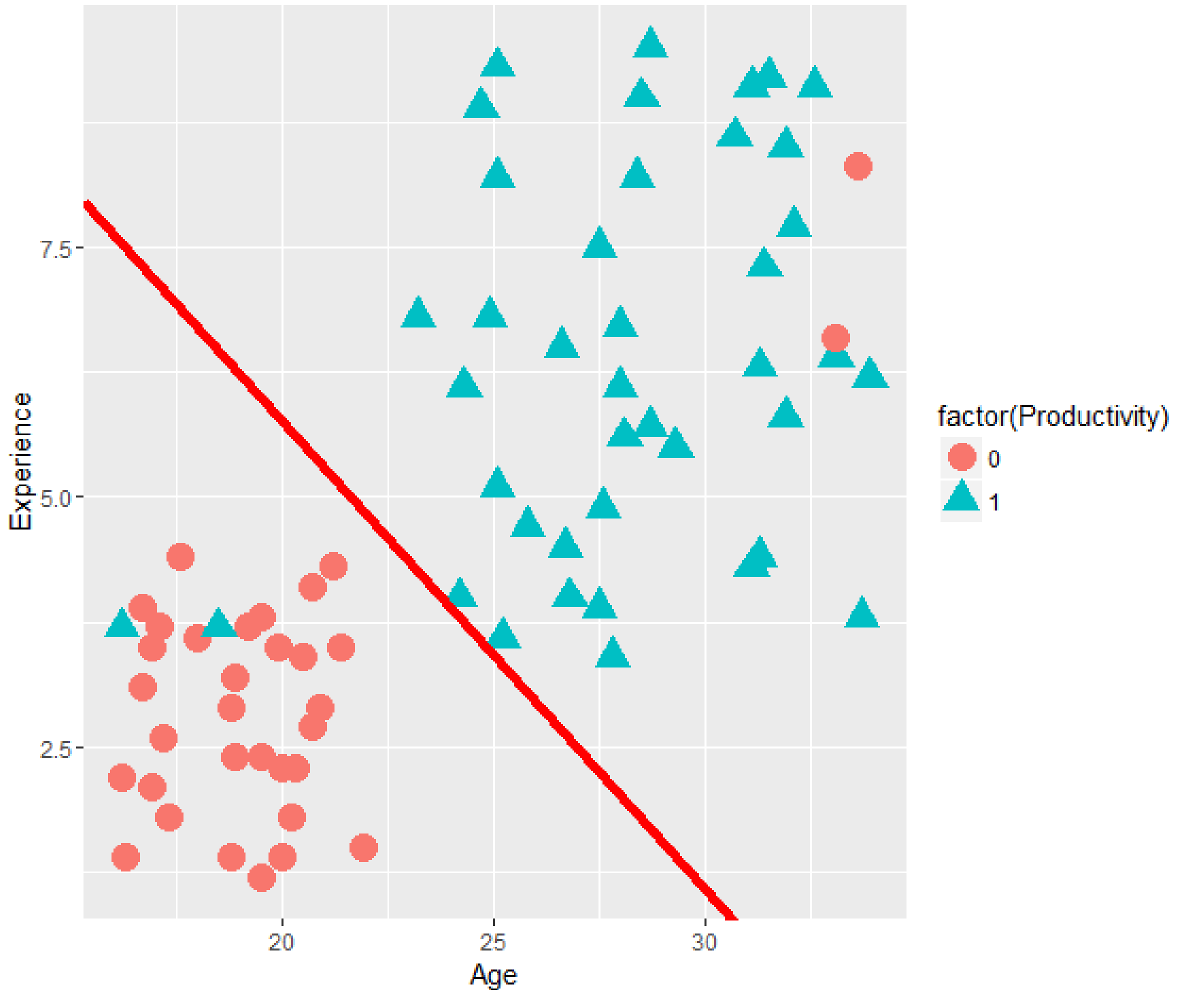
Code: Logistic Regression

```
> ####Accuracy of the model1
> predicted_values<-round(predict(Emp_Productivity_logit,type="response"),0)
> conf_matrix<-table(predicted_values,Emp_Productivity_logit$y)
> conf_matrix

predicted_values  0  1
               0 31  2
               1  2 39
>
> accuracy<-(conf_matrix[1,1]+conf_matrix[2,2])/(sum(conf_matrix))
> accuracy
[1] 0.9459459
```

Code: Decision Boundary

```
> Emp_Productivity_logit

Call:  glm(formula = Productivity ~ Age + Experience, family = binomial(),
    data = Emp_Productivity1)

Coefficients:
(Intercept)          Age   Experience
    -8.9361       0.2763       0.5923

Degrees of Freedom: 73 Total (i.e. Null);  71 Residual
Null Deviance:      101.7
Residual Deviance: 46.77        AIC: 52.77
> coef(Emp_Productivity_logit)
(Intercept)         Age  Experience
 -8.9361114   0.2762749   0.5923444
>
> slope1 <- coef(Emp_Productivity_logit)[2]/(-coef(Emp_Productivity_logit)[3])
> intercept1 <- coef(Emp_Productivity_logit)[1]/(-coef(Emp_Productivity_logit)[3])
>
> library(ggplot2)
> base<-ggplot(Emp_Productivity1)+geom_point(aes(x=Age,y=Experience,color=factor(Productivity),shape=factor(Productivity)),size=5)
> base+geom_abline(intercept = intercept1 , slope = slope1, color = "red", size = 2) #Base is the scatter plot. Then we are adding the de
cision boundary
```

Code: Decision Boundary



statinfer

New representation for logistic regression

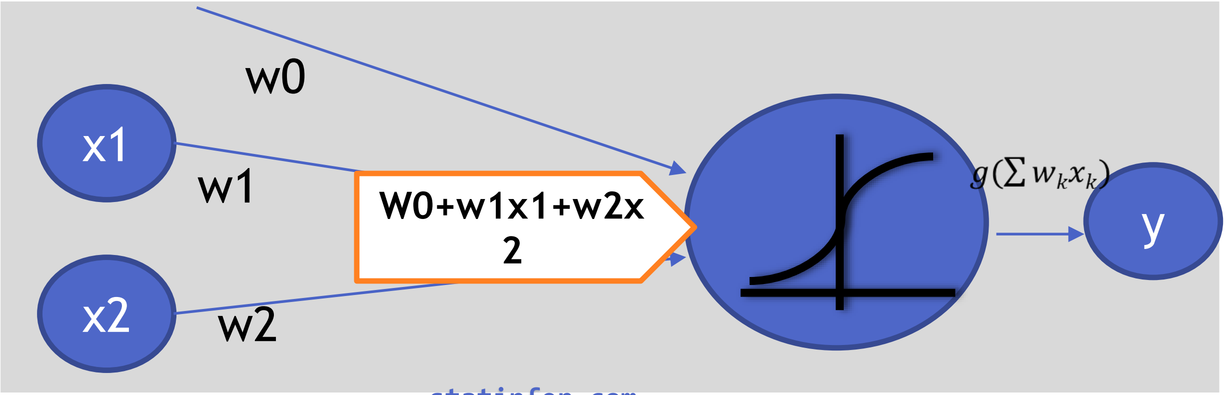
New representation for logistic regression

$$y = \frac{e^{\beta 0+\beta 1x1+\beta 2x2}}{1+e^{\beta 0+\beta 1x1+\beta 2x2}}$$

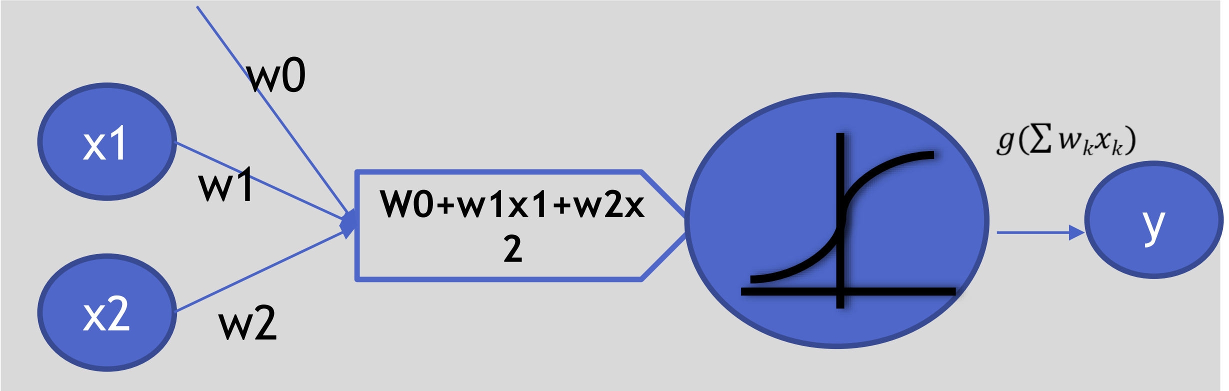
$$y = \frac{1}{1+e^{-(\beta 0+\beta 1x1+\beta 2x2})}$$

$$y = g(w_0 + w_1x_1 + w_2x_2) \quad where \quad g(x) = \frac{1}{1+e^{-x}}$$

$$y = g(\sum w_k x_k)$$



Finding the weights in logistic regression



$$out(x) = g(\sum w_k x_k)$$

The above output is a non linear function of linear combination of inputs – A typical multiple logistic regression line

$$We\ find\ \ w\ to\ minimize\ \sum_{i=1}^{n}[y_i - g(\sum w_k x_k)]^2$$

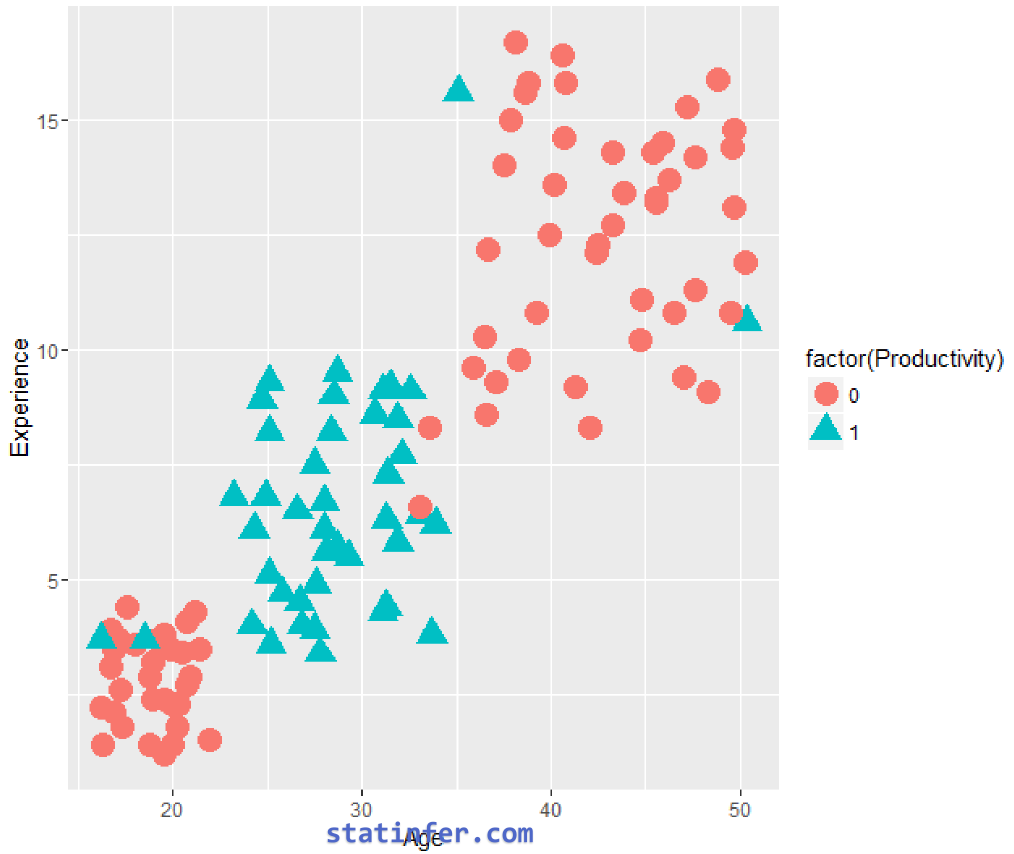
LAB: Non-Linear Decision Boundaries

LAB: Non-Linear Decision Boundaries

- Dataset: “Emp_Productivity/ Emp_Productivity_All_Sites.csv”
- Draw a scatter plot that shows Age on X axis and Experience on Y-axis. Try to distinguish the two classes with colors or shapes (visualizing the classes)
- Build a logistic regression model to predict Productivity using age and experience
- Finally draw the decision boundary for this logistic regression model
- Create the confusion matrix
- Calculate the accuracy and error rates

Code: Non-Linear Decision Boundaries

```
####The clasification graph on overall data
library(ggplot2)
ggplot(Emp_Productivity_raw)+geom_point(aes(x=Age,y=Experience,color=factor(Productivity),shape=factor(Productivity)),size=5)
```



Code: Non-Linear Decision Boundaries

```
> ###Logistic Regerssion model for overall data
> Emp_Productivity_logit_overall<-glm(Productivity~Age+Experience,data=Emp_Productivity_raw, family=binomial())
> Emp_Productivity_logit_overall

Call:  glm(formula = Productivity ~ Age + Experience, family = binomial(),
    data = Emp_Productivity_raw)

Coefficients:
(Intercept)          Age   Experience
    0.44784     -0.01755     -0.06324

Degrees of Freedom: 118 Total (i.e. Null);  116 Residual
Null Deviance:      155.7
Residual Deviance: 150.5        AIC: 156.5
```

Code: Non-Linear Decision Boundaries

```
> slope2 <- coef(Emp_Productivity_logit_overall)[2]/(-coef(Emp_Productivity_logit_overall)[3])
> intercept2 <- coef(Emp_Productivity_logit_overall)[1]/(-coef(Emp_Productivity_logit_overall)[3])
>
>
> ####Drawing the Decision boundary
>
> library(ggplot2)
> base<-ggplot(Emp_Productivity_raw)+geom_point(aes(x=Age,y=Experience,color=factor(Productivity),shape=factor(Productivity)),size=5)
> base+geom_abline(intercept = intercept2 , slope = slope2, colour = "blue", size = 2)
```

Code: Non-Linear Decision Boundaries

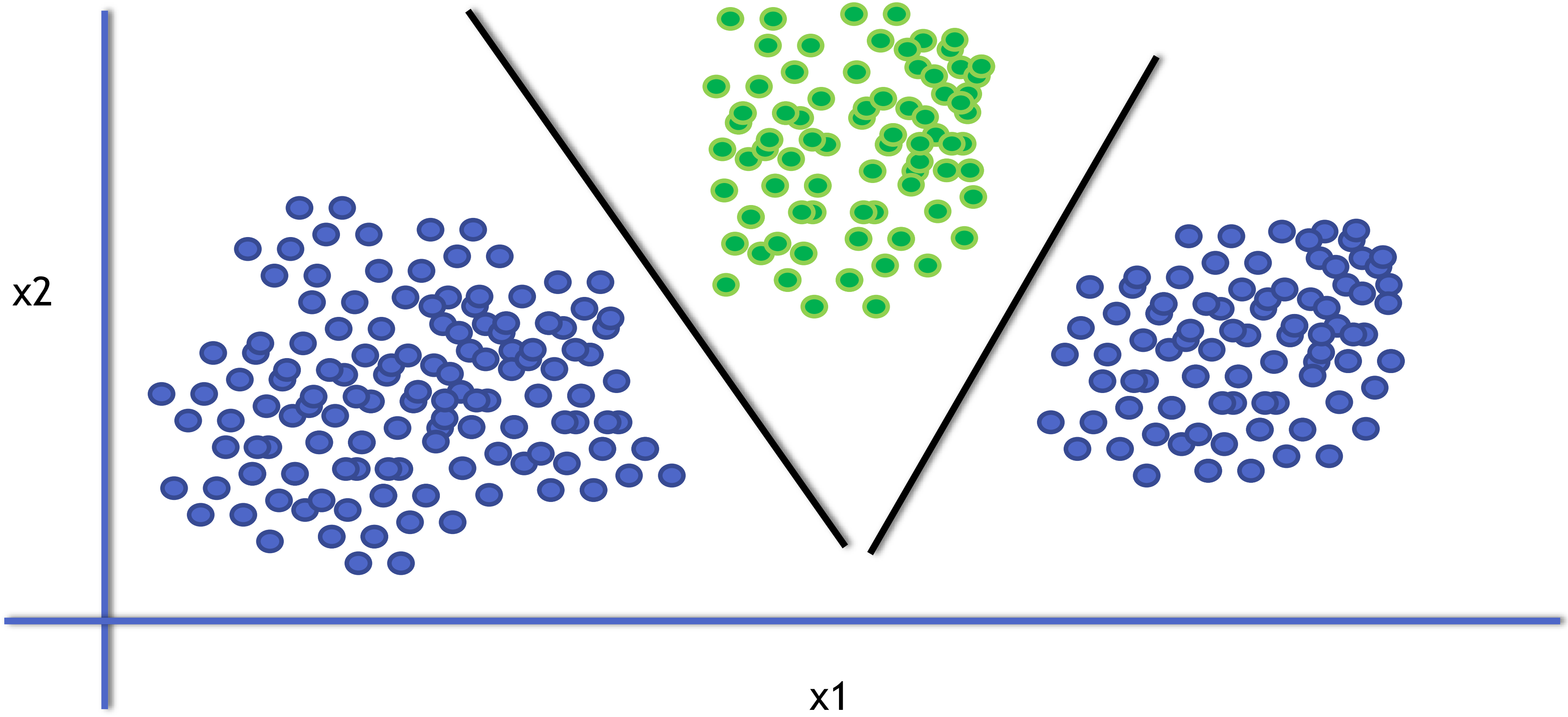
Code: Non-Linear Decision Boundaries

```
> ####Accuracy of the overall model
> predicted_values<-round(predict(Emp_Productivity_logit_overall,type="response"),0)
> conf_matrix<-table(predicted_values,Emp_Productivity_logit_overall$y)
> conf_matrix

predicted_values  0  1
               0 69 43
               1  7  0
>
> accuracy<-(conf_matrix[1,1]+conf_matrix[2,2])/(sum(conf_matrix))
> accuracy
[1] 0.5798319
```

statinfer

Non-Linear Decision Boundaries-Issue

Non-Linear Decision Boundaries

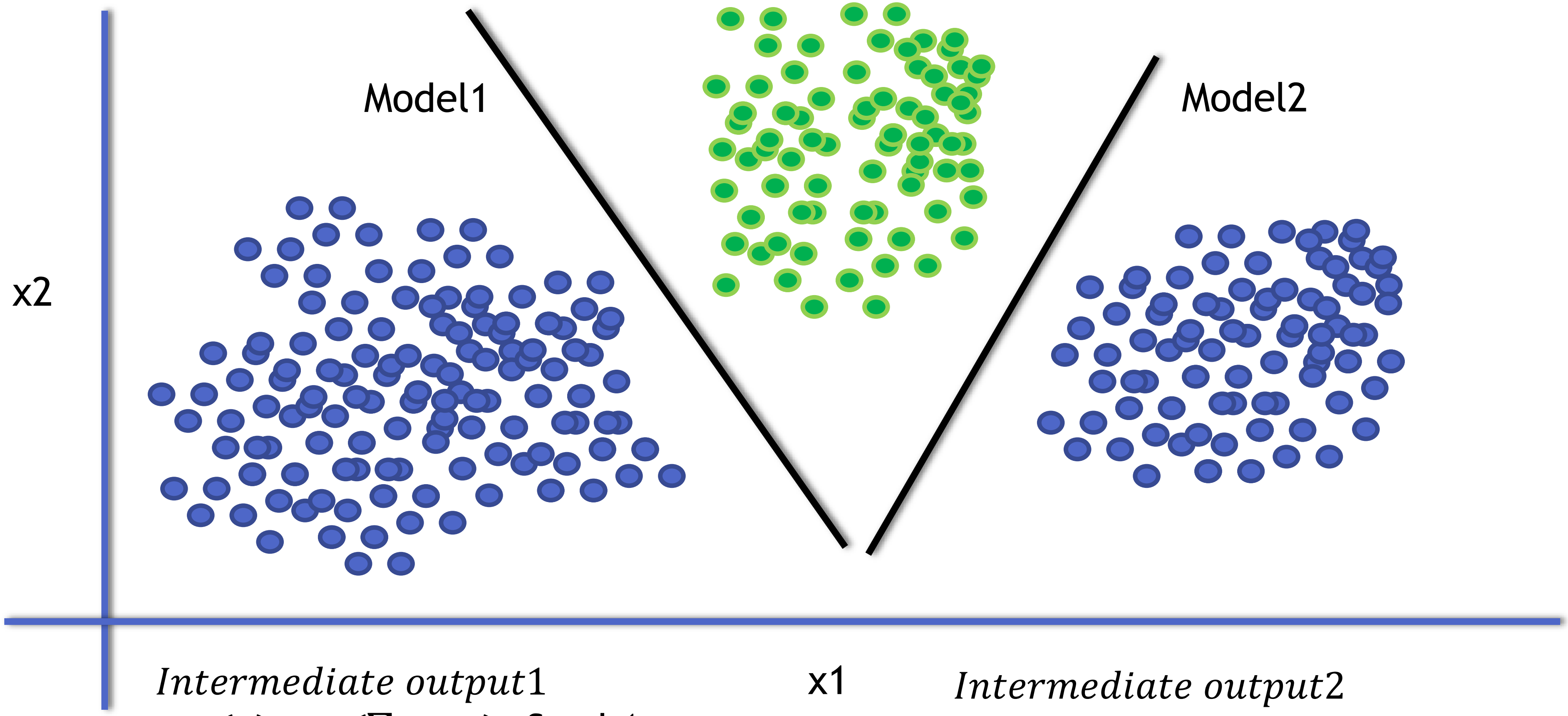


Non-Linear Decision Boundaries-issues

- Logistic Regression line doesn’t seam to be a good option when we have non-linear decision boundaries

Non-Linear Decision Boundaries-Solution

Intermediate outputs

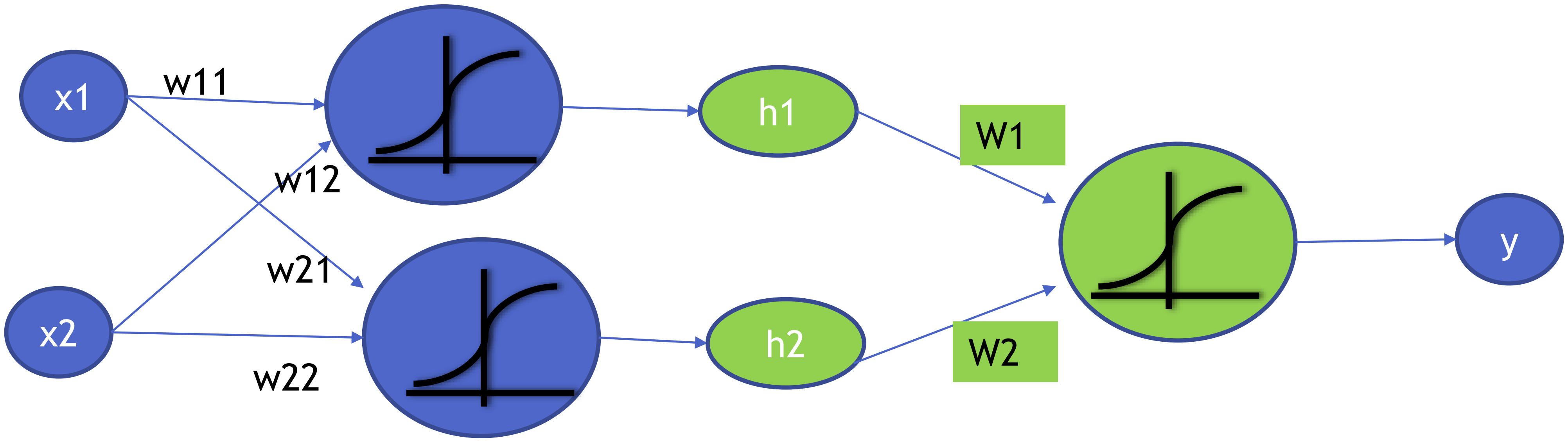


$Intermediate\ output1$
$out(x) = g(\sum w_k x_k)$,Say h1

$Intermediate\ output2$
$out(x) = g(\sum w_k x_k)$, Say h2

The Intermediate output

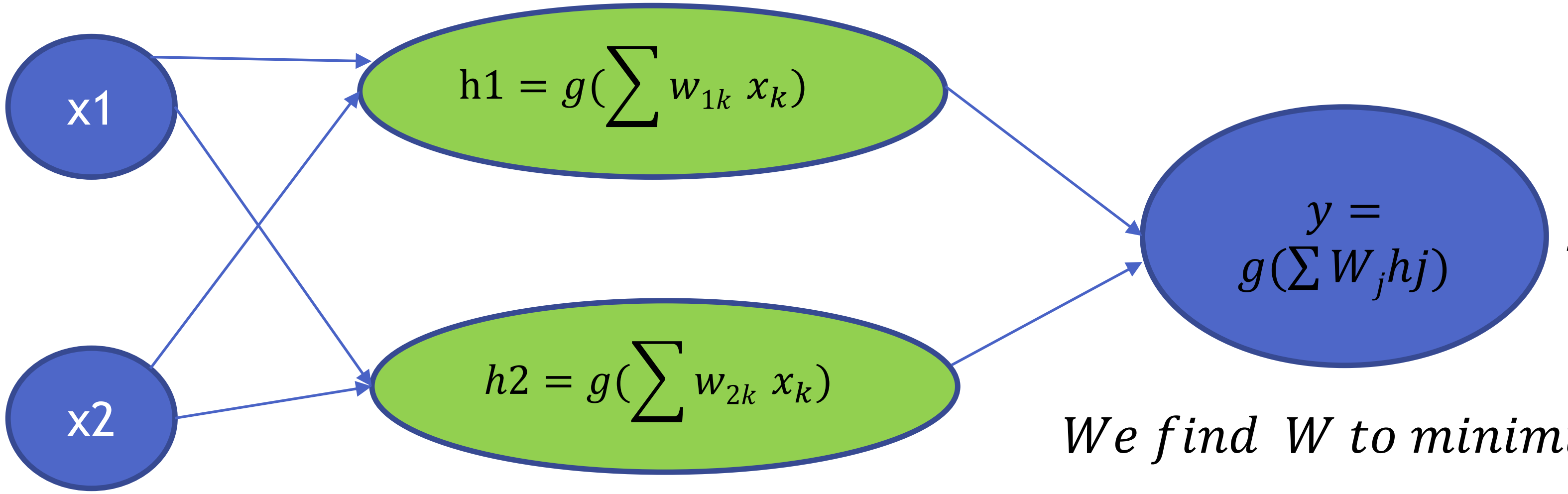
- Using the x's Directly predicting y is challenging.
- We can predict h, the intermediate output, which will indeed predict Y



Finding the weights for intermediate outputs

$Intermediate\ output1$
$h1 = out(x) = g(\sum w_{1k}\ x_k)$

$We\ find\ \ w_1\ to\ minimize\ \sum_{i=1}^{n}[h_{1i} - g(\sum w_{1k}\ x_k)]^2$



$Final\ output$
$y = out(h) = g(\sum W_j hj)$

$We\ find\ \ W\ to\ minimize\ \sum_{i=1}^{n}[y_i - g(\sum W_j h_{ji})]^2$

$Intermediate\ output2$
$h2 = out(x) = g(\sum w_{2k}\ x_k)$

$We\ find\ \ w_2\ to\ minimize\ \sum_{i=1}^{n}[h_{2i} - g(\sum w_{1k}\ x_k)]^2$

statinfer

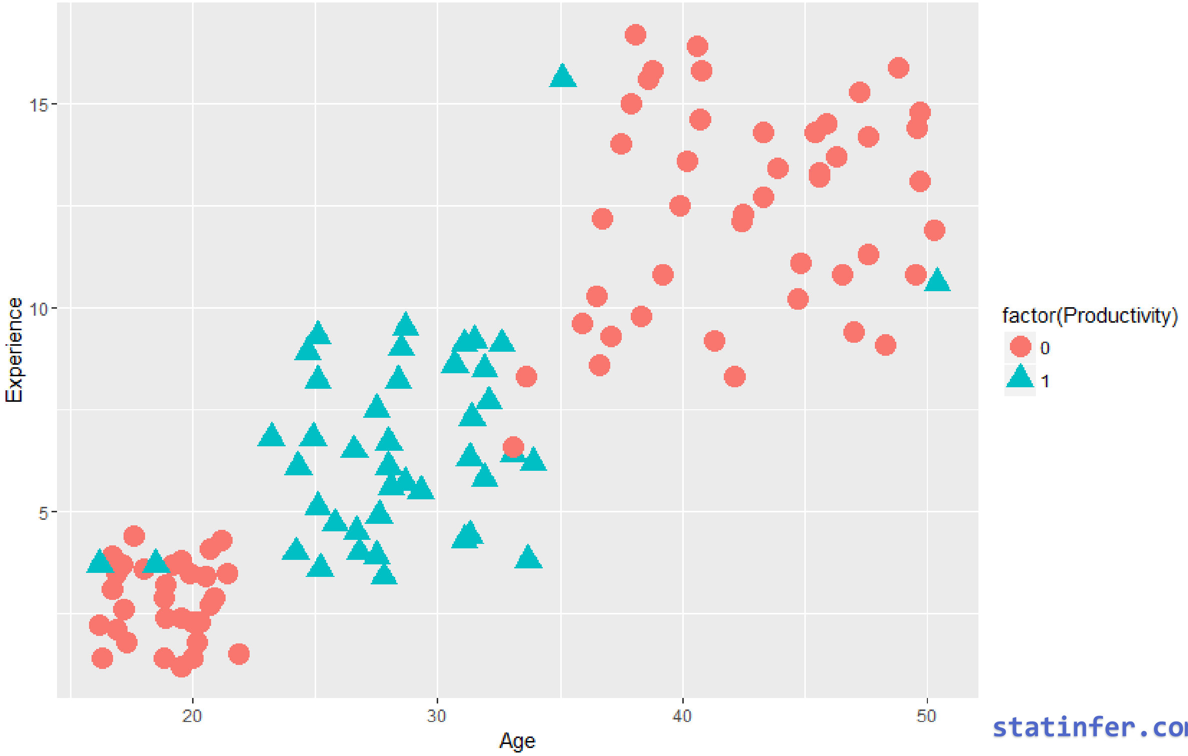
LAB: Intermediate output

LAB: Intermediate output

- Dataset: Emp_Productivity/ Emp_Productivity_All_Sites.csv
- Filter the data and take first 74 observations from above dataset .
- Build a logistic regression model to predict Productivity using age and experience
- Calculate the prediction probabilities for all the inputs. Store the probabilities in inter1 variable
- Filter the data and take observations from row 34 onwards.
- Build a logistic regression model to predict Productivity using age and experience
- Calculate the prediction probabilities for all the inputs. Store the probabilities in inter2 variable
- Build a consolidated model to predict productivity using inter-1 and inter-2 variables
- Create the confusion matrix and find the accuracy and error rates for the consolidated model

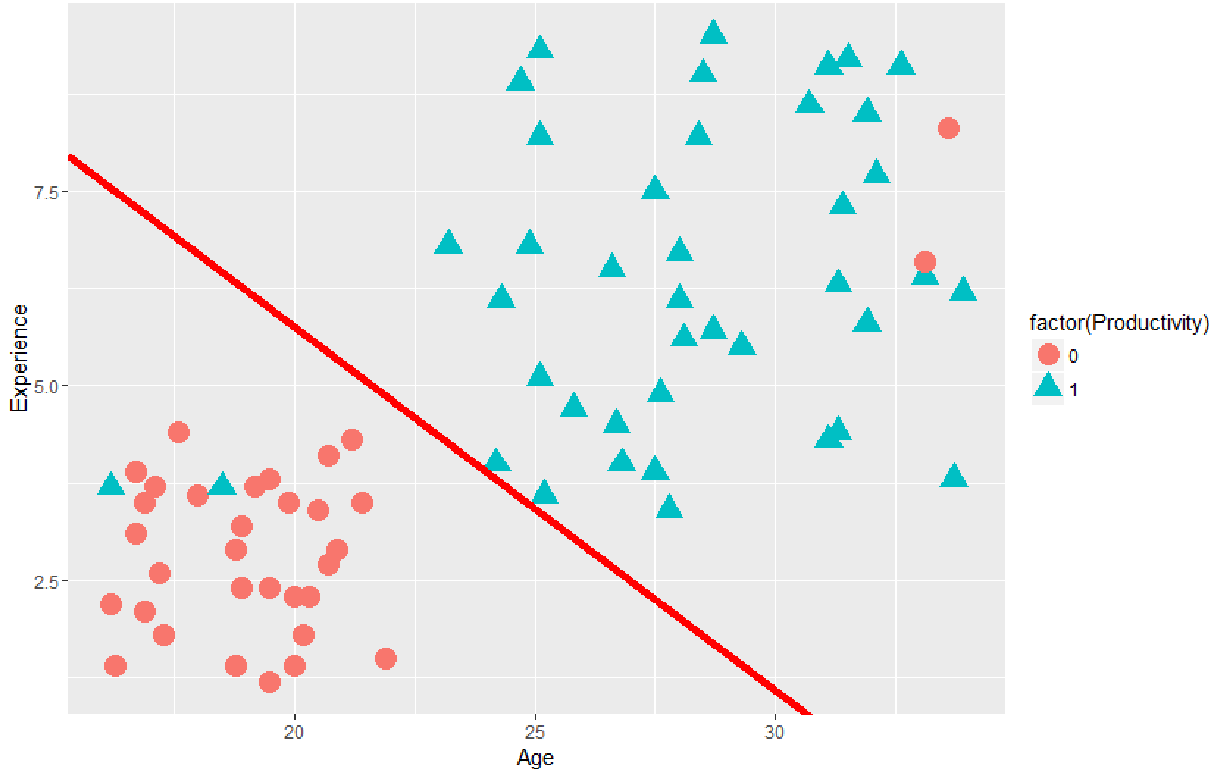
Code: Intermediate output

```
####The clasification graph on overall data
library(ggplot2)
ggplot(Emp_Productivity_raw)+geom_point(aes(x=Age,y=Experience,color=factor(Productivity),shape=factor(Pro


####The clasification graph Sample-1
library(ggplot2)
ggplot(Emp_Productivity1)+geom_point(aes(x=Age,y=Experience,color=factor(Productivity),shape=factor(Produc
```



Code: Intermediate output

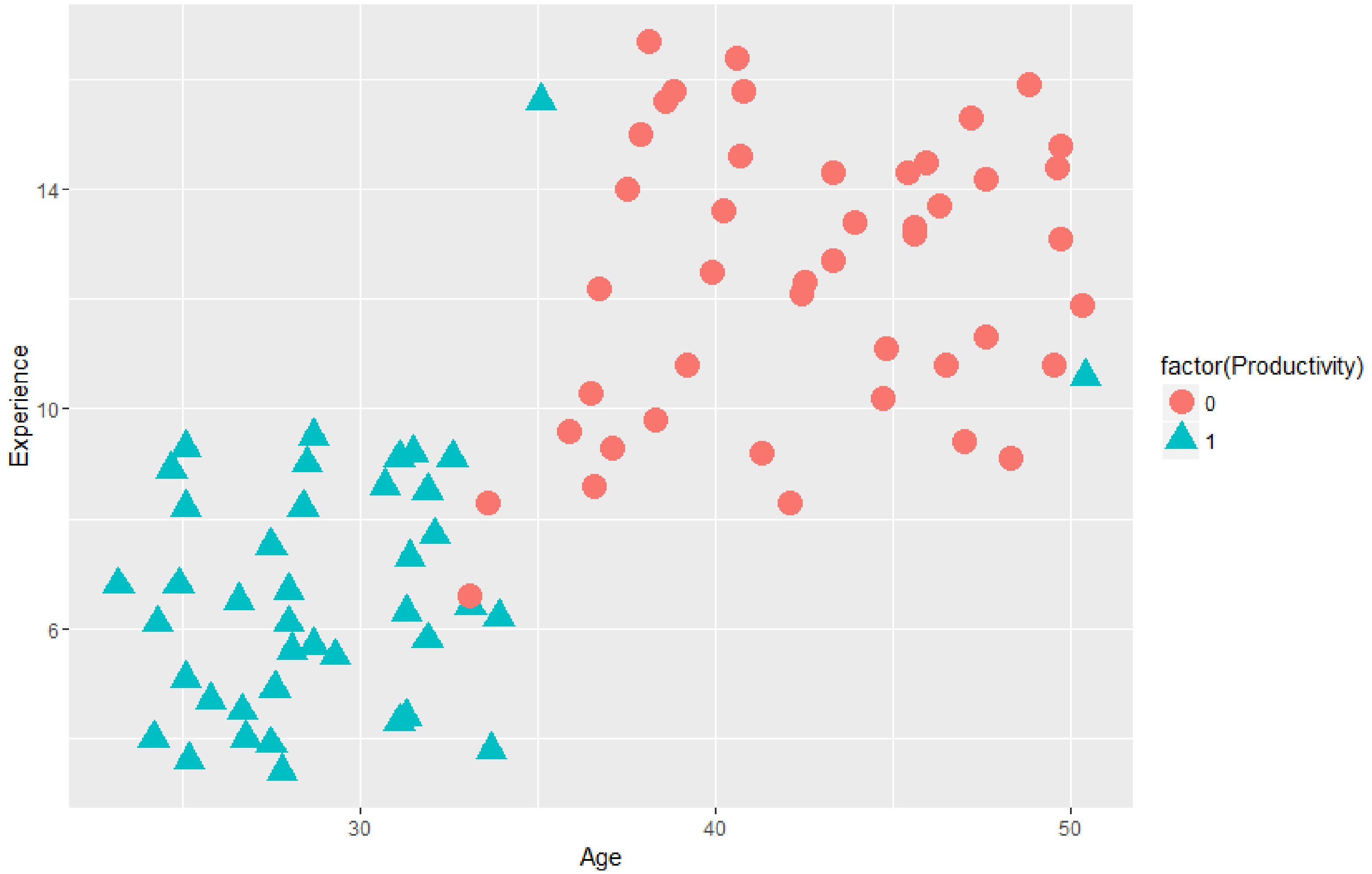
```
####Decision boundary for model1 built on Sample-1
library(ggplot2)
base<-ggplot(Emp_Productivity1)+geom_point(aes(x=Age,y=Experience,color=factor(Productivity),shape=factor(
base+geom_abline(intercept = intercept1 , slope = slope1, color = "red", size = 2) #Base is the scatter pl
```



Code: Intermediate output

```
#### sample-2
############################################

Emp_Productivity2<-Emp_Productivity_raw[Emp_Productivity_raw$Sample_Set>1,]

####The clasification graph
library(ggplot2)
ggplot(Emp_Productivity2)+geom_point(aes(x=Age,y=Experience,color=factor(Productivity),shape=factor(Produc
```

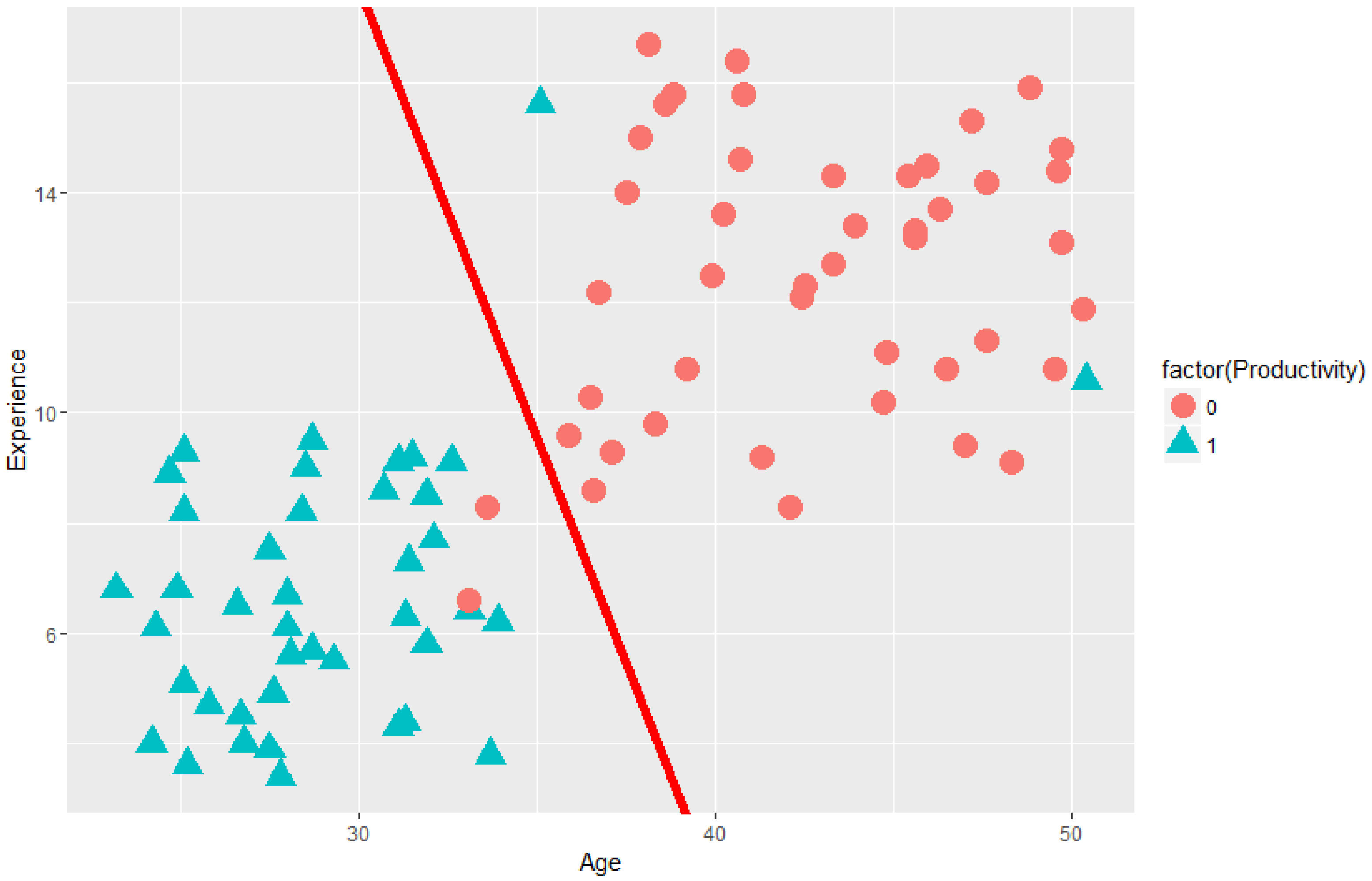
Code: Intermediate output



Code: Intermediate output

```
###Logistic Regerssion model2 built on Sample2
Emp_Productivity_logit2<-glm(Productivity~Age+Experience,data=Emp_Productivity2, family=binomial())
Emp_Productivity_logit2

coef(Emp_Productivity_logit2)
slope3 <- coef(Emp_Productivity_logit2)[2]/(-coef(Emp_Productivity_logit2)[3])
intercept3 <- coef(Emp_Productivity_logit2)[1]/(-coef(Emp_Productivity_logit2)[3])


####Drawing the Decison boundry
library(ggplot2)
base<-ggplot(Emp_Productivity2)+geom_point(aes(x=Age,y=Experience,color=factor(Productivity),shape=factor(
base+geom_abline(intercept = intercept3 , slope = slope3, color = "red", size = 2)
```

Code: Intermediate output



Code: Intermediate output

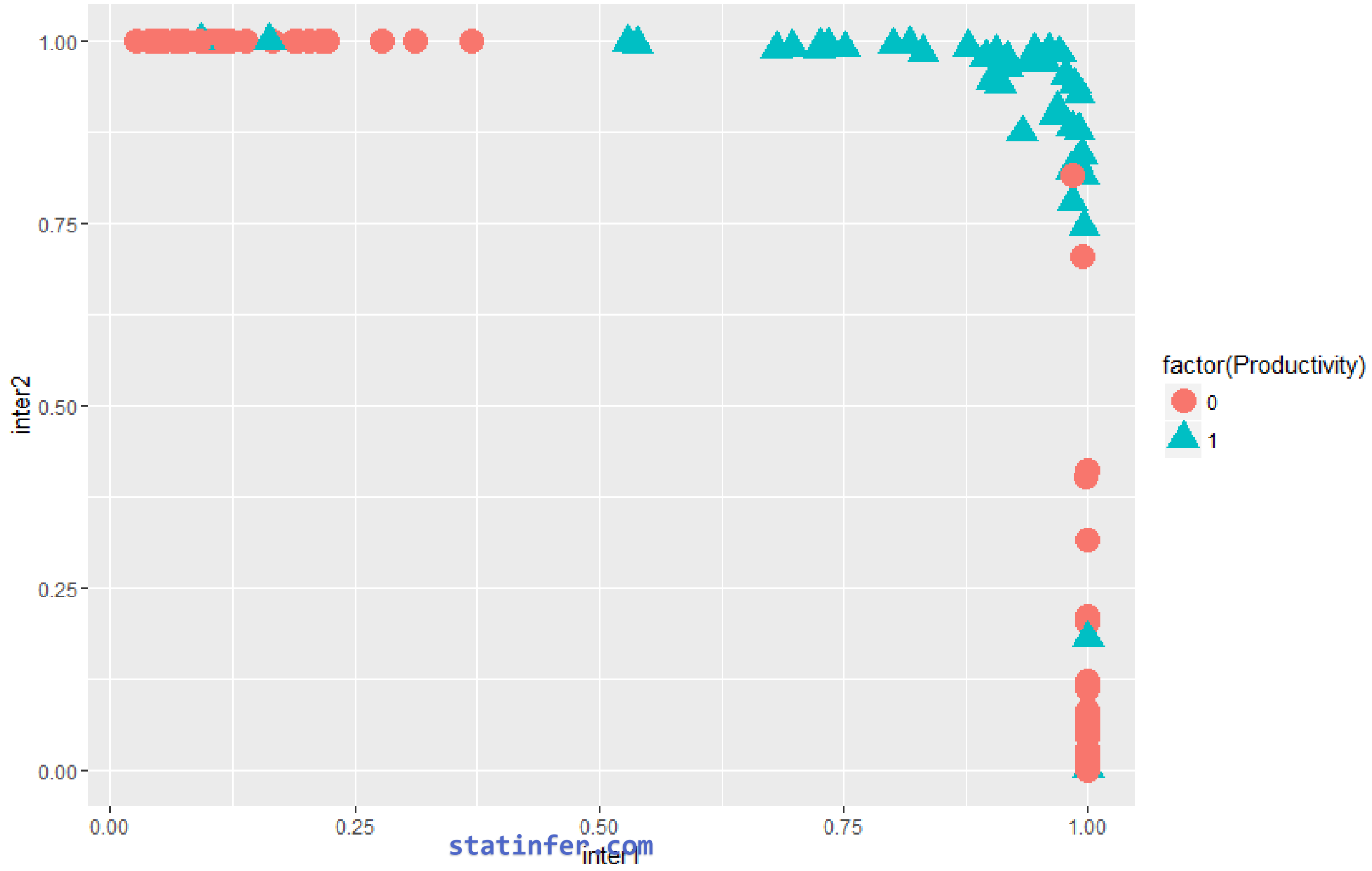
```
> ####Accuracy of the model2
> predicted_values<-round(predict(Emp_Productivity_logit2,type="response"),0)
> conf_matrix<-table(predicted_values,Emp_Productivity_logit2$y)
> conf_matrix

predicted_values  0  1
               0 43  2
               1  2 39
>
> accuracy<-(conf_matrix[1,1]+conf_matrix[2,2])/(sum(conf_matrix))
> accuracy
[1] 0.9534884
```

Code: Intermediate output

```
> #The Two models
> Emp_Productivity_logit

Call:  glm(formula = Productivity ~ Age + Experience, family = binomial(),
    data = Emp_Productivity1)

Coefficients:
(Intercept)          Age   Experience
    -8.9361       0.2763       0.5923

Degrees of Freedom: 73 Total (i.e. Null);  71 Residual
Null Deviance:      101.7
Residual Deviance: 46.77        AIC: 52.77
> Emp_Productivity_logit2

Call:  glm(formula = Productivity ~ Age + Experience, family = binomial(),
    data = Emp_Productivity2)

Coefficients:
(Intercept)          Age   Experience
    16.3184      -0.3994      -0.2440

Degrees of Freedom: 85 Total (i.e. Null);  83 Residual
Null Deviance:      119
Residual Deviance: 34.08        AIC: 40.08
>
> #The two new coloumns
> Emp_Productivity_raw$inter1<-predict(Emp_Productivity_logit,type="response", newdata=Emp_Productivity_raw)
> Emp_Productivity_raw$inter2<-predict(Emp_Productivity_logit2,type="response", newdata=Emp_Productivity_raw)
>
> head(Emp_Productivity_raw)
   Age Experience Productivity Sample_Set     inter1    inter2
1 20.0        2.3            0          1 0.11423230 0.9995775
2 16.2        2.2            0          1 0.04080461 0.9999096
3 20.2        1.8            0          1 0.09202657 0.9995949
4 18.8        1.4            0          1 0.05152147 0.9997899
5 18.9        3.2            0          1 0.13955234 0.9996608
6 16.7        3.9            0          1 0.11793035 0.9998329
```

Code: Intermediate output

```
> ####Clasification graph with the two new coloumns
> library(ggplot2)
> ggplot(Emp_Productivity_raw)+geom_point(aes(x=inter1,y=inter2,color=factor(Productivity),shape=factor(Producti
vity)),size=5)
```

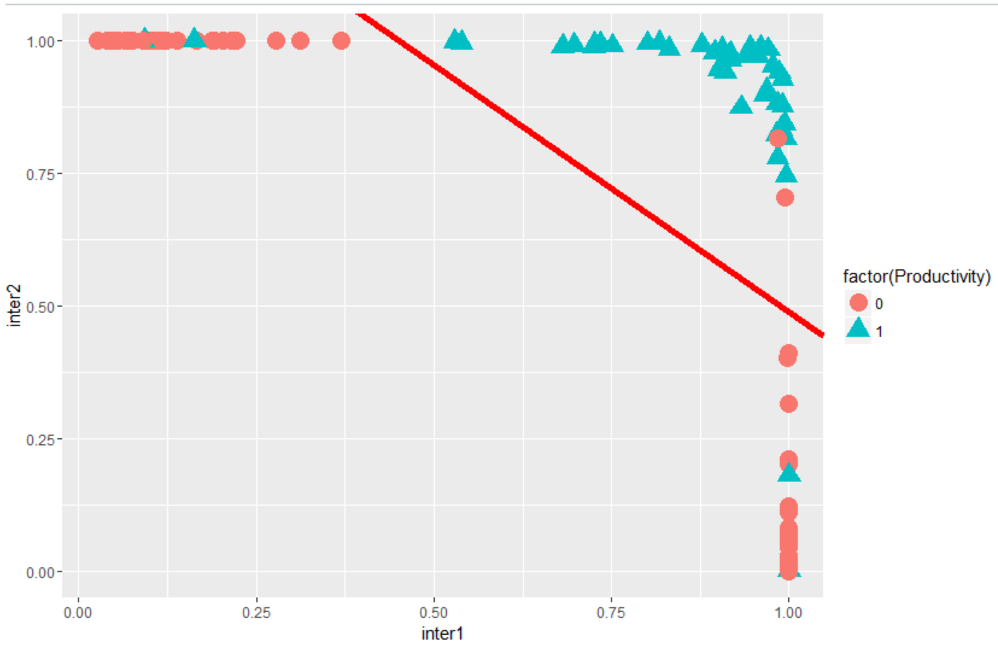


Code: Intermediate output

```
> ###Logistic Regerssion model with Intermediate outputs as input
> Emp_Productivity_logit_combined<-glm(Productivity~inter1+inter2,data=Emp_Productivity_raw, family=binomial())
> Emp_Productivity_logit_combined

Call:  glm(formula = Productivity ~ inter1 + inter2, family = binomial(), 
    data = Emp_Productivity_raw)

Coefficients:
(Intercept)       inter1       inter2  
    -12.213        8.019        8.598  

Degrees of Freedom: 118 Total (i.e. Null);  116 Residual
Null Deviance:      155.7 
Residual Deviance: 49.74        AIC: 55.74
```

Code: Intermediate output

```
####Drawing the Decison boundry
slope4 <- coef(Emp_Productivity_logit_combined)[2]/(-coef(Emp_Productivity_logit_combined)[3])
intercept4<- coef(Emp_Productivity_logit_combined)[1]/(-coef(Emp_Productivity_logit_combined)[3])


library(ggplot2)
base<-ggplot(Emp_Productivity_raw)+geom_point(aes(x=inter1,y=inter2,color=factor(Productivity),shape=facto
base+geom_abline(intercept = intercept4 , slope = slope4, colour = "red", size = 2)
```



Code: Intermediate output

```
> ####Accuracy of the combined
> predicted_values<-round(predict(Emp_Productivity_logit_combined,type="response"),0)
> conf_matrix<-table(predicted_values,Emp_Productivity_logit_combined$y)
> conf_matrix

predicted_values  0  1
               0 74  4
               1  2 39
>
> accuracy<-(conf_matrix[1,1]+conf_matrix[2,2])/(sum(conf_matrix))
> accuracy
[1] 0.9495798
```

Code: Intermediate output

```
> #The Final three models
> Emp_Productivity_logit

Call:  glm(formula = Productivity ~ Age + Experience, family = binomial(), 
    data = Emp_Productivity1)

Coefficients:
(Intercept)          Age   Experience  
    -8.9361       0.2763       0.5923  

Degrees of Freedom: 73 Total (i.e. Null);  71 Residual
Null Deviance:      101.7 
Residual Deviance: 46.77        AIC: 52.77
> Emp_Productivity_logit2

Call:  glm(formula = Productivity ~ Age + Experience, family = binomial(), 
    data = Emp_Productivity2)

Coefficients:
(Intercept)          Age   Experience  
    16.3184      -0.3994      -0.2440  

Degrees of Freedom: 85 Total (i.e. Null);  83 Residual
Null Deviance:      119 
Residual Deviance: 34.08        AIC: 40.08
> Emp_Productivity_logit_combined

Call:  glm(formula = Productivity ~ inter1 + inter2, family = binomial(), 
    data = Emp_Productivity_raw)

Coefficients:
(Intercept)       inter1       inter2  
    -12.213        8.019        8.598  

Degrees of Freedom: 118 Total (i.e. Null);  116 Residual
Null Deviance:      155.7 
Residual Deviance: 49.74        AIC: 55.74
```

Code: Intermediate output

```
> #Acccuracies of all the models till now
> accuracy_all_data
[1] 0.5798319
> accuracy_Sample1
[1] 0.9459459
> accuracy_Sample2
[1] 0.9534884
> accuracy_intermediate_Step
[1] 0.9495798
```

statinfer

Neural Network intuition

Neural Network intuition

$$\textit{Final output}$$
$$y = out(h) = g(\sum W_j h_j)$$

$$h_j = out(x) = g(\sum w_{jk} x_k)$$

$$y = out(h) = g(\sum W_j g(\sum w_{jk} x_k))$$

- So h is a non linear function of linear combination of inputs – A multiple logistic regression line
- Y is a non linear function of linear combination of outputs of logistic regressions
- Y is a non linear function of linear combination of non linear functions of linear combination of inputs

$$We\ find\ \ W\ to\ minimize\ \sum_{i=1}^{n}[y_i - g(\sum W_j h_j)]^2$$

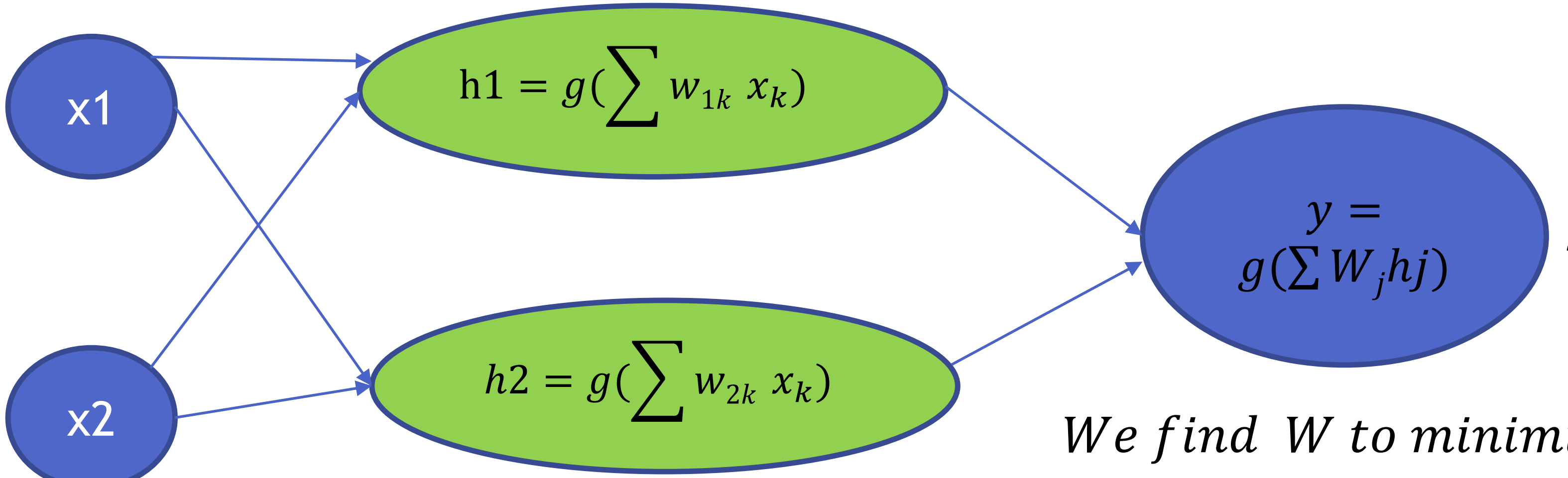
$$We\ find\ \{W_j\}\ \&\ \{w_{jk}\}\ to\ minimize\ \sum_{i=1}^{n}[y_i - g(\sum W_j g(\sum w_{jk} x_k))]^2$$

Neural networks is all about finding the sets of weights {Wj,} and $\{w_{jk}\}$ using **Gradient Descent Method**

Neural Network intuition

$Intermediate\ output1$
$h1 = out(x) = g(\sum w_{1k}\ x_k)$

$$We\ find\ \ w_1\ to\ minimize\ \sum_{i=1}^{n}[h_{1i} - g(\sum w_{1k}\ x_k)]^2$$



$Final\ output$
$y = out(h) = g(\sum W_j hj)$

$$We\ find\ \ W\ to\ minimize\ \sum_{i=1}^{n}[y_i - g(\sum W_j h_{ji})]^2$$

$Intermediate\ output2$
$h2 = out(x) = g(\sum w_{2k}\ x_k)$

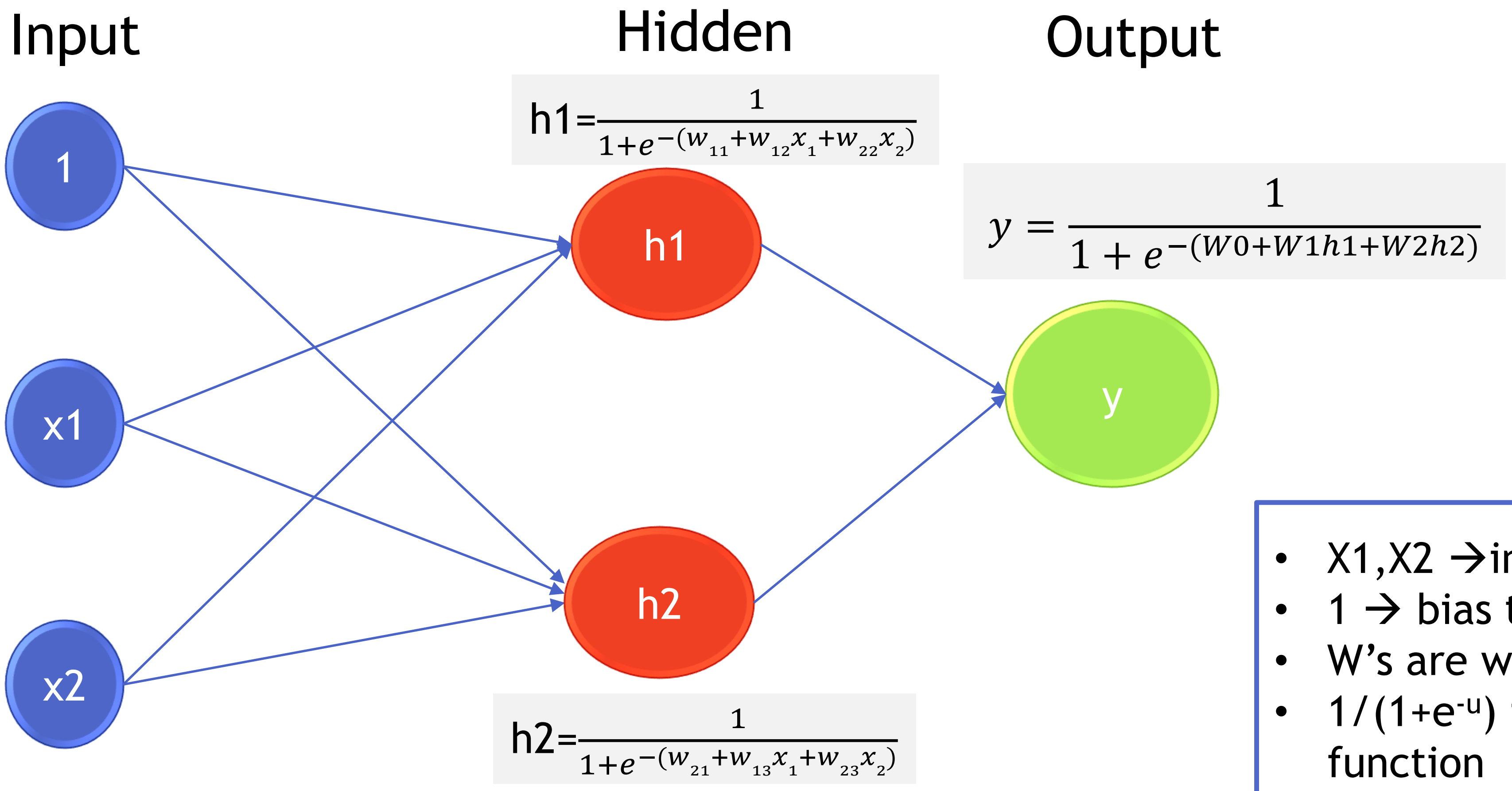
$$We\ find\ \ w_2\ to\ minimize\ \sum_{i=1}^{n}[h_{2i} - g(\sum w_{1k}\ x_k)]^2$$

The Neural Networks

- The neural networks methodology is similar to the intermediate output method explained above.
- But we will not manually subset the data to create the different models.
- The neural network technique automatically takes care of all the intermediate outputs using hidden layers
- It works very well for the data with non-linear decision boundaries
- The intermediate output layer in the network is known as hidden layer
- In Simple terms, neural networks are multi layer nonlinear regression models.
- If we have sufficient number of hidden layers, then we can estimate any complex non-linear function

statinfer

Neural network and vocabulary



- X1,X2 → inputs
- 1 → bias term
- W’s are weights
- $1/(1+e^{-u})$ is the sigmoid function
- Y is output

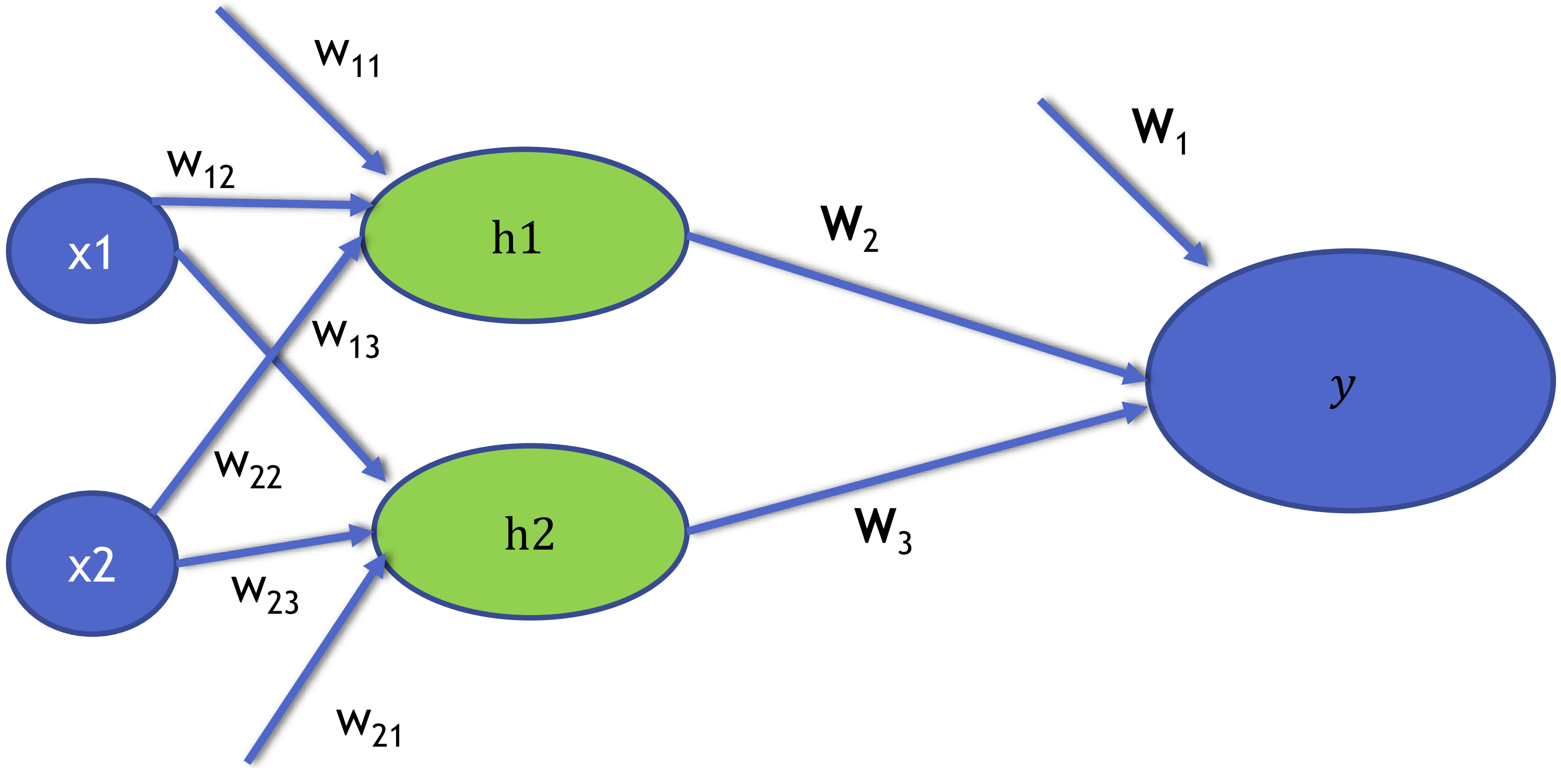
Why are they called hidden layers?

- A hidden layer “hides” the desired output.
- Instead of predicting the actual output using a single model, build multiple models to predict intermediate output
- There is no standard way of deciding the number of hidden layers.

statinfer

The Neural network Algorithm

Algorithm for Finding weights

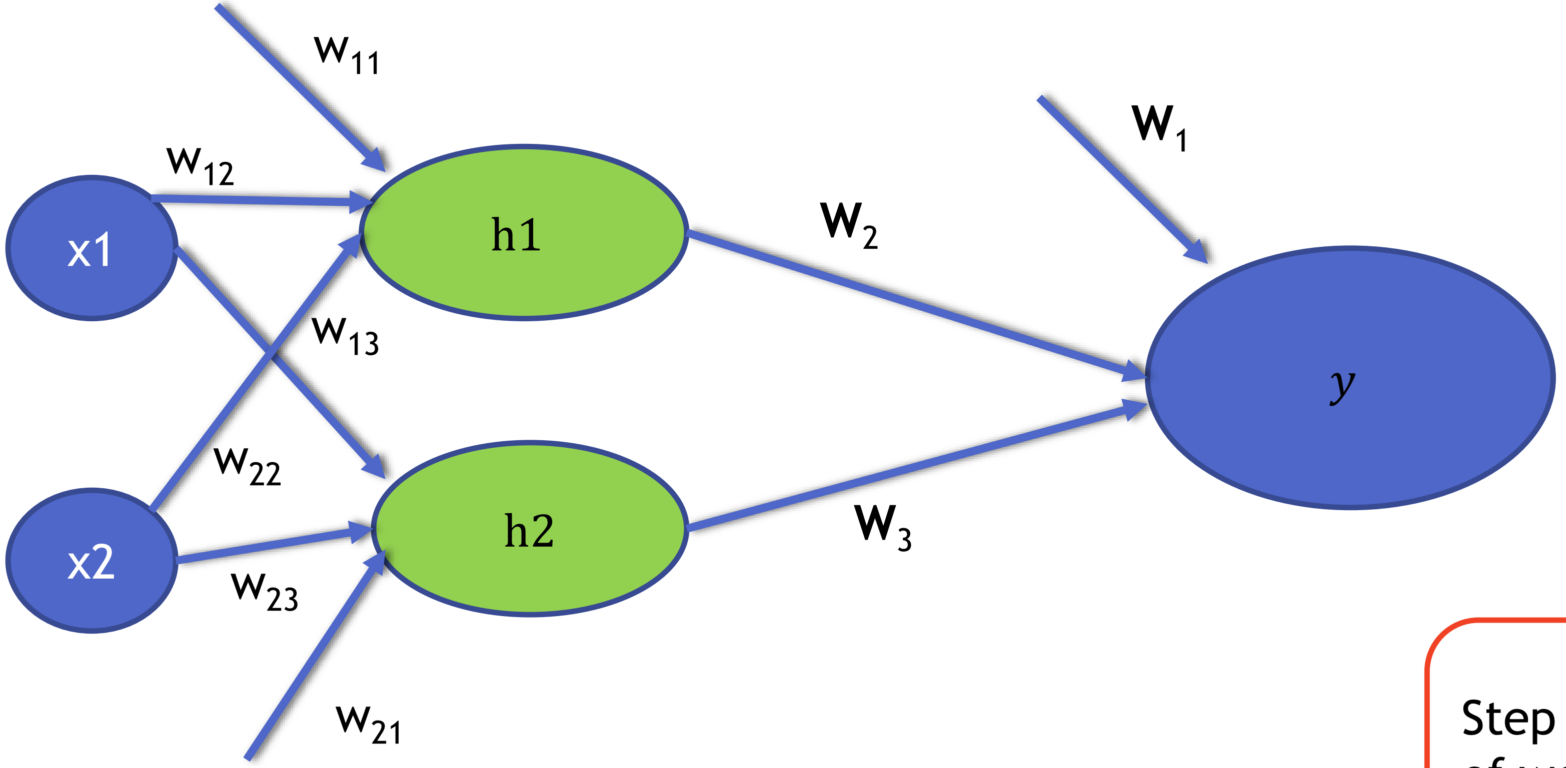


- Algorithm is all about finding the weights/coefficients
- We randomly initialize some weights; Calculate the output by supplying training input; If there is an error the weights are adjusted to reduce this error.

The Neural Network Algorithm

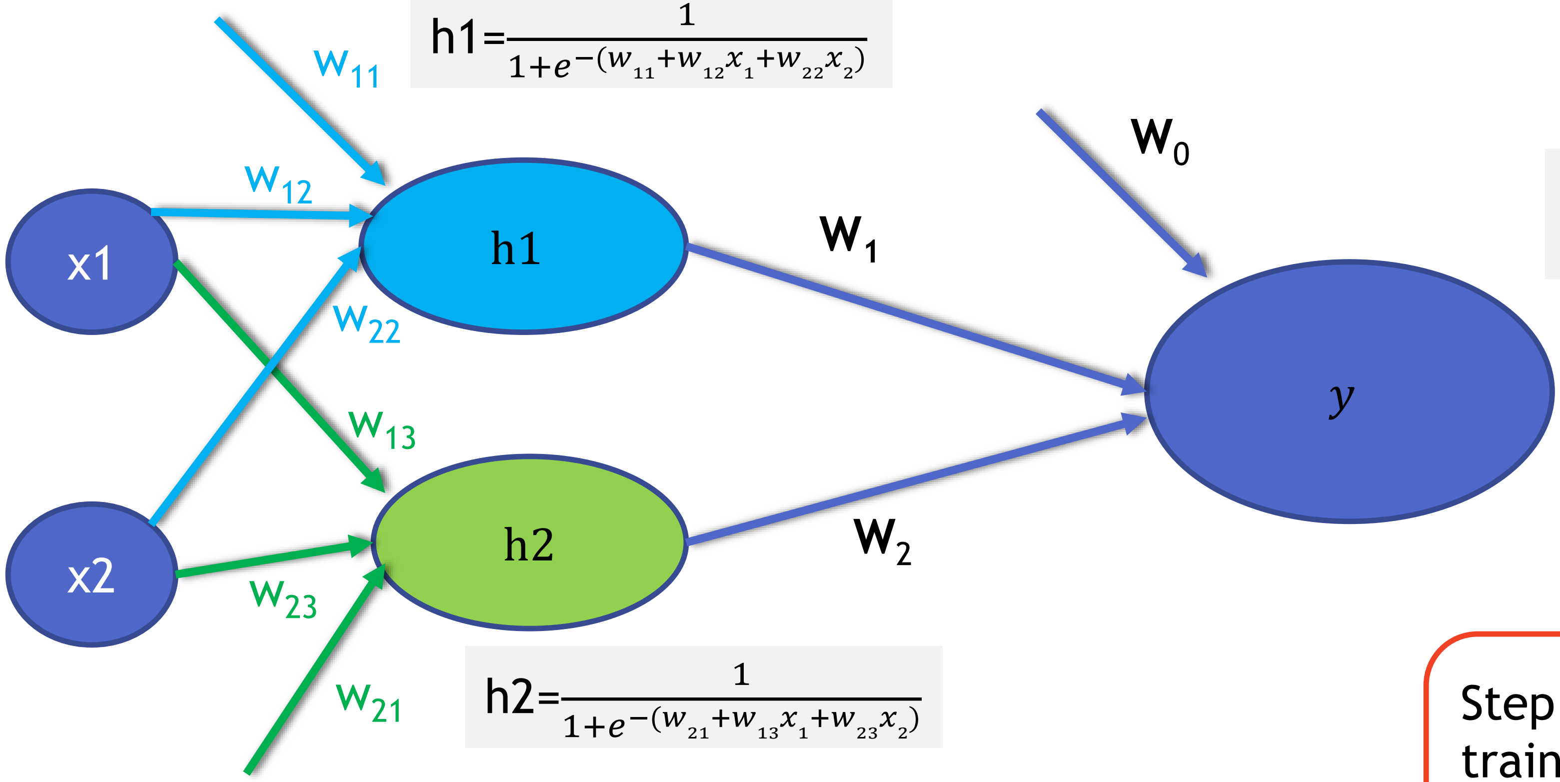
- **Step 1: Initialization of weights:** Randomly select some weights
- **Step 2 : Training & Activation:** Input the training values and perform the calculations forward.
- **Step 3 : Error Calculation:** Calculate the error at the outputs. Use the output error to calculate error fractions at each hidden layer
- **Step 4: Weight training :** Update the weights to reduce the error, recalculate and repeat the process of training & updating the weights for all the examples.
- **Step 5: Stopping criteria:** Stop the training and weights updating process when the minimum error criteria is met

Randomly initialize weights



Step 1: Initialization of weights: Randomly select some weights

Training & Activation



$$h1=\frac{1}{1+e^{-(w_{11}+w_{12}x_1+w_{22}x_2)}}$$

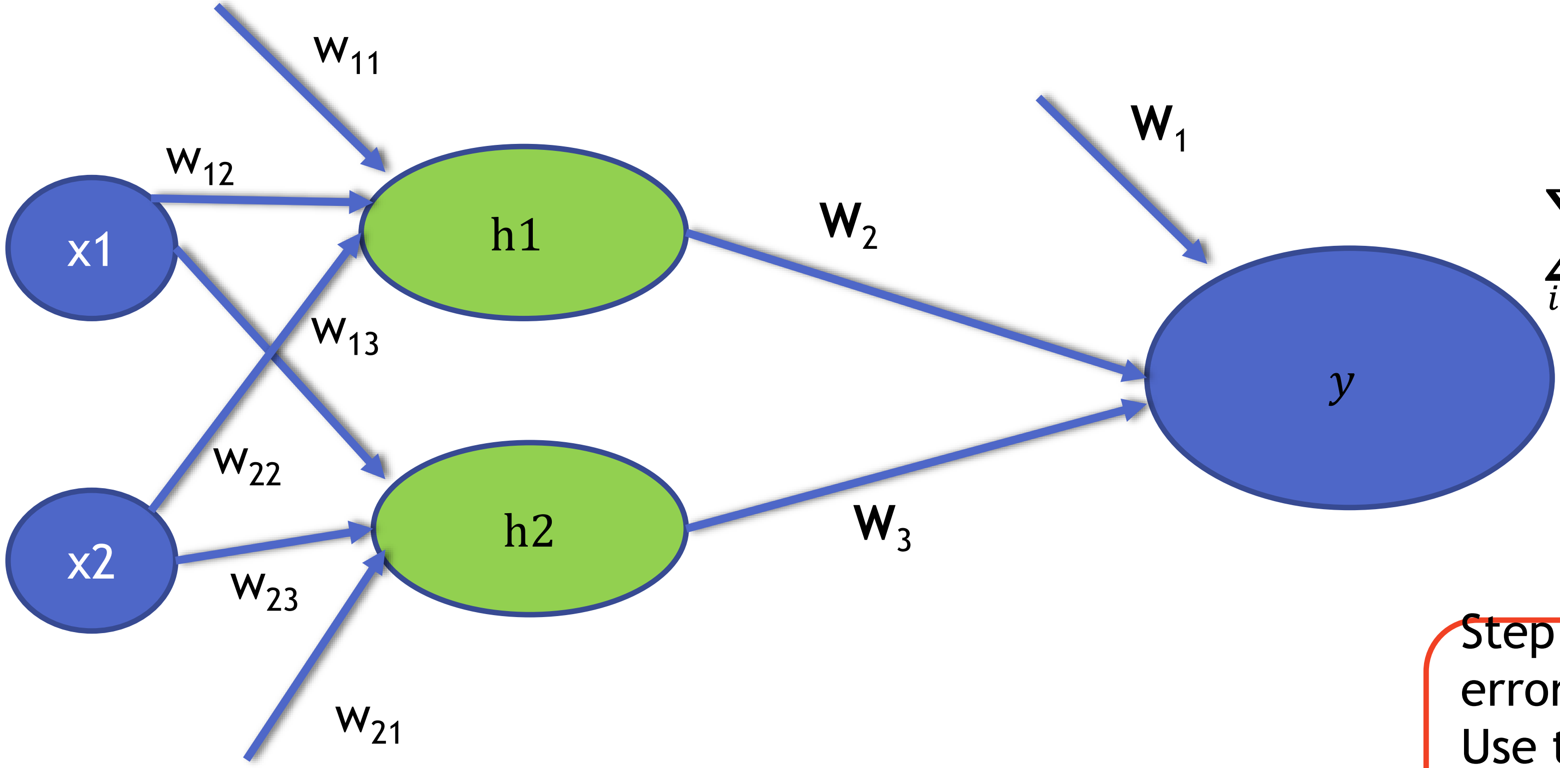
$$h2=\frac{1}{1+e^{-(w_{21}+w_{13}x_1+w_{23}x_2)}}$$

$$y=\frac{1}{1+e^{-(W0+W1h1+W2h2)}}$$

Step 2 : Input the training values and perform the calculations forward

Training input & calculations – **Feed Forward**

Error Calculation at Output



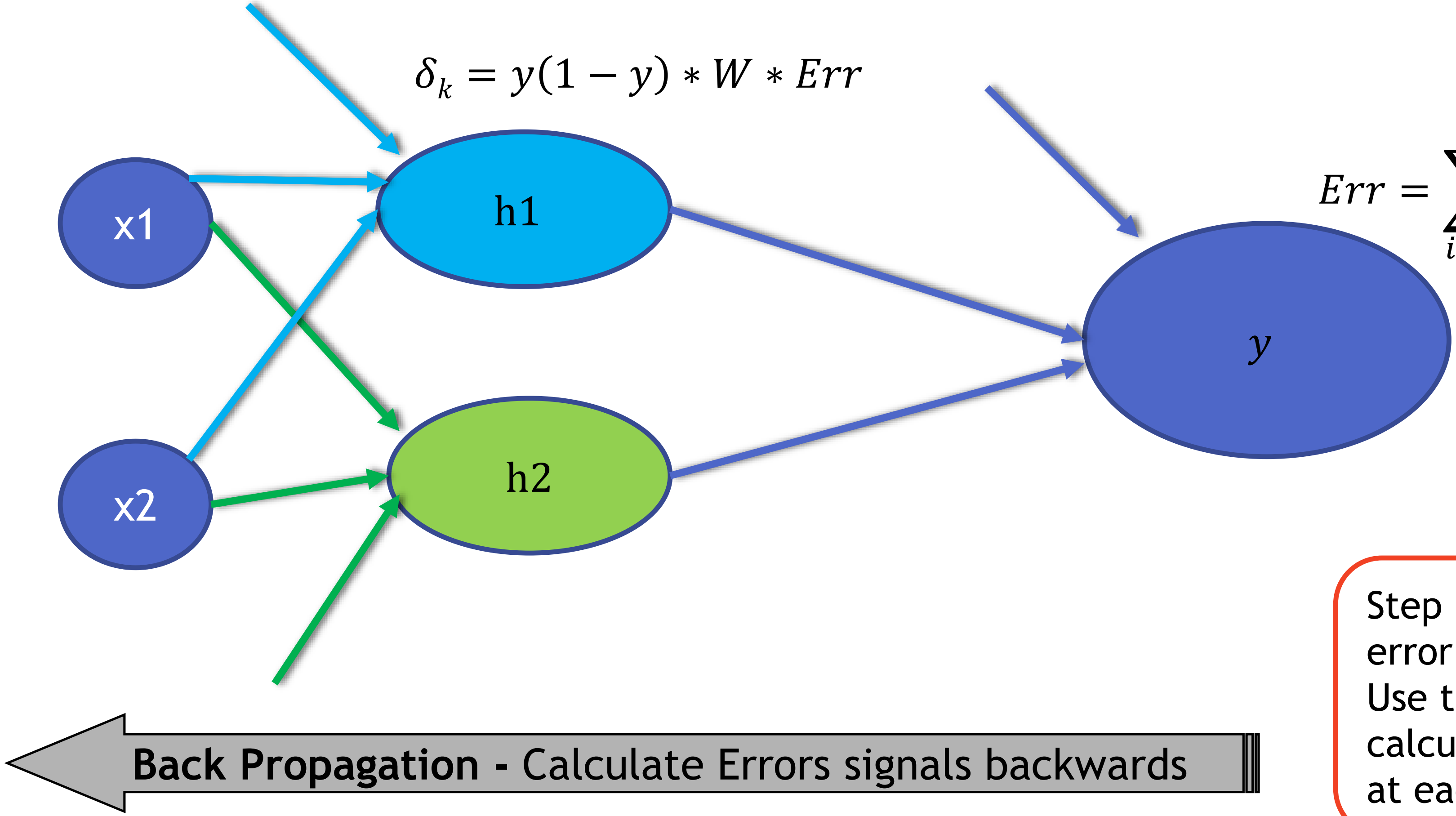
$$\sum_{i=1}^{n}\left[y_i - g\left(\sum_{k=1}^{m} w_k h_{ki}\right)\right]^2$$

Step 3: Calculate the error at the outputs. Use the output error to calculate error fractions at each hidden layer

Error Calculation at hidden layers

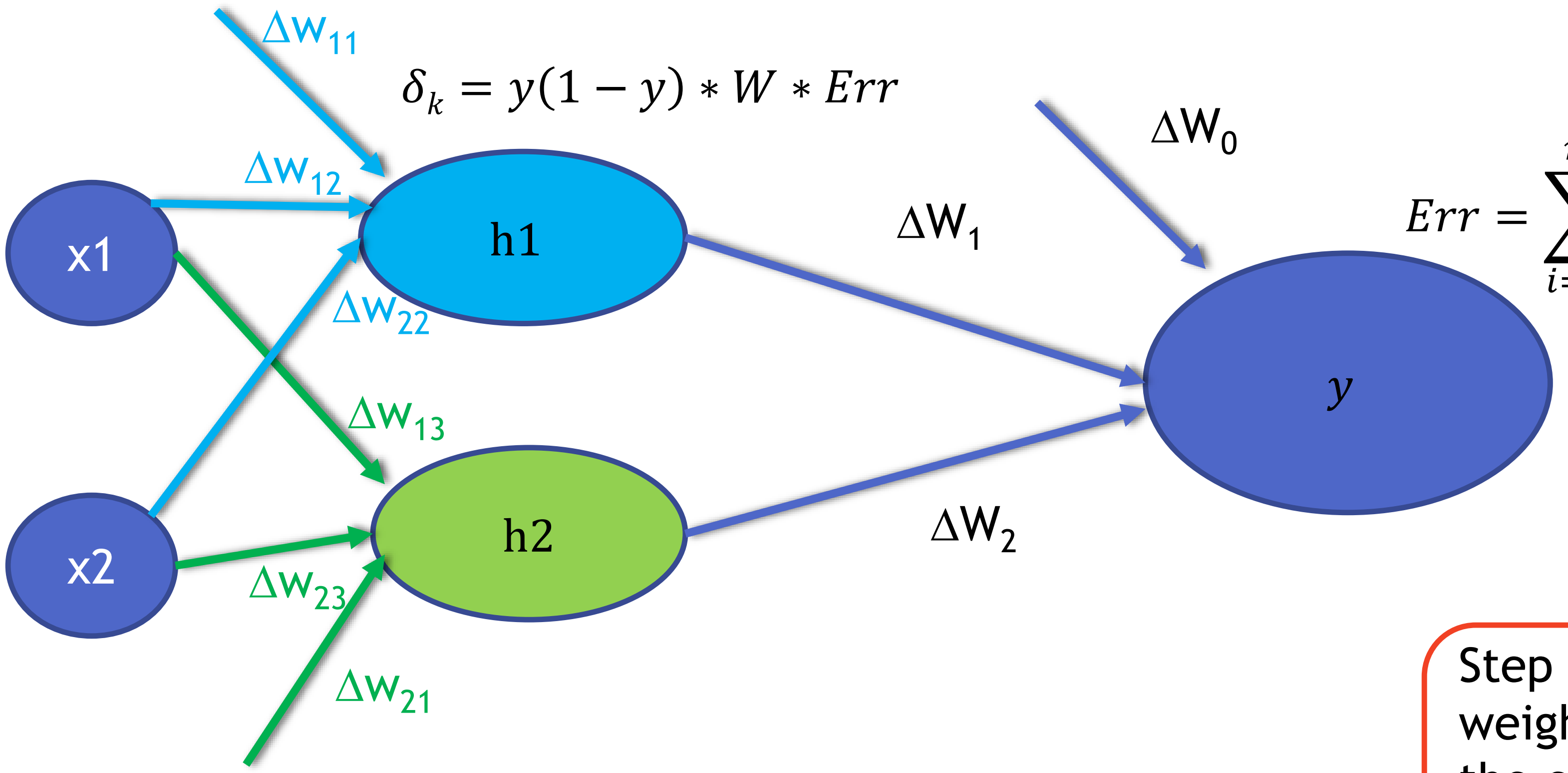
$$\delta_k = y(1-y) * W * Err$$

$$Err = \sum_{i=1}^{n}\left[y_i - g\left(\sum_{k=1}^{m} w_k h_{ki}\right)\right]^2$$



Step 3: Calculate the error at the outputs. Use the output error to calculate error fractions at each hidden layer

Calculate weight corrections

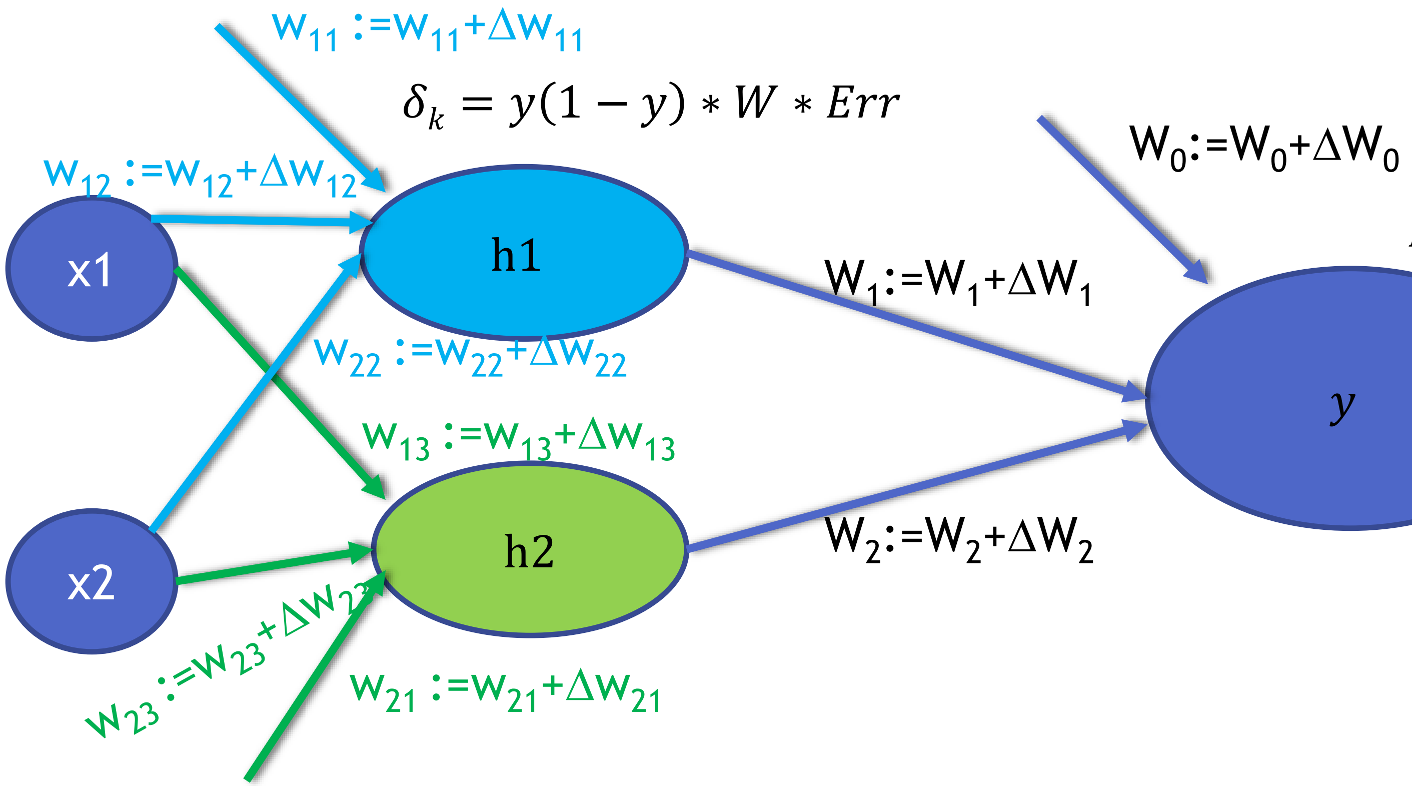


$$\delta_k = y(1 - y) * W * Err$$

$$Err = \sum_{i=1}^{n} \left[y_i - g\left(\sum_{k=1}^{m} w_k h_{ki} \right) \right]^2$$

Step 4: Update the weights to reduce the error, recalculate and repeat the process

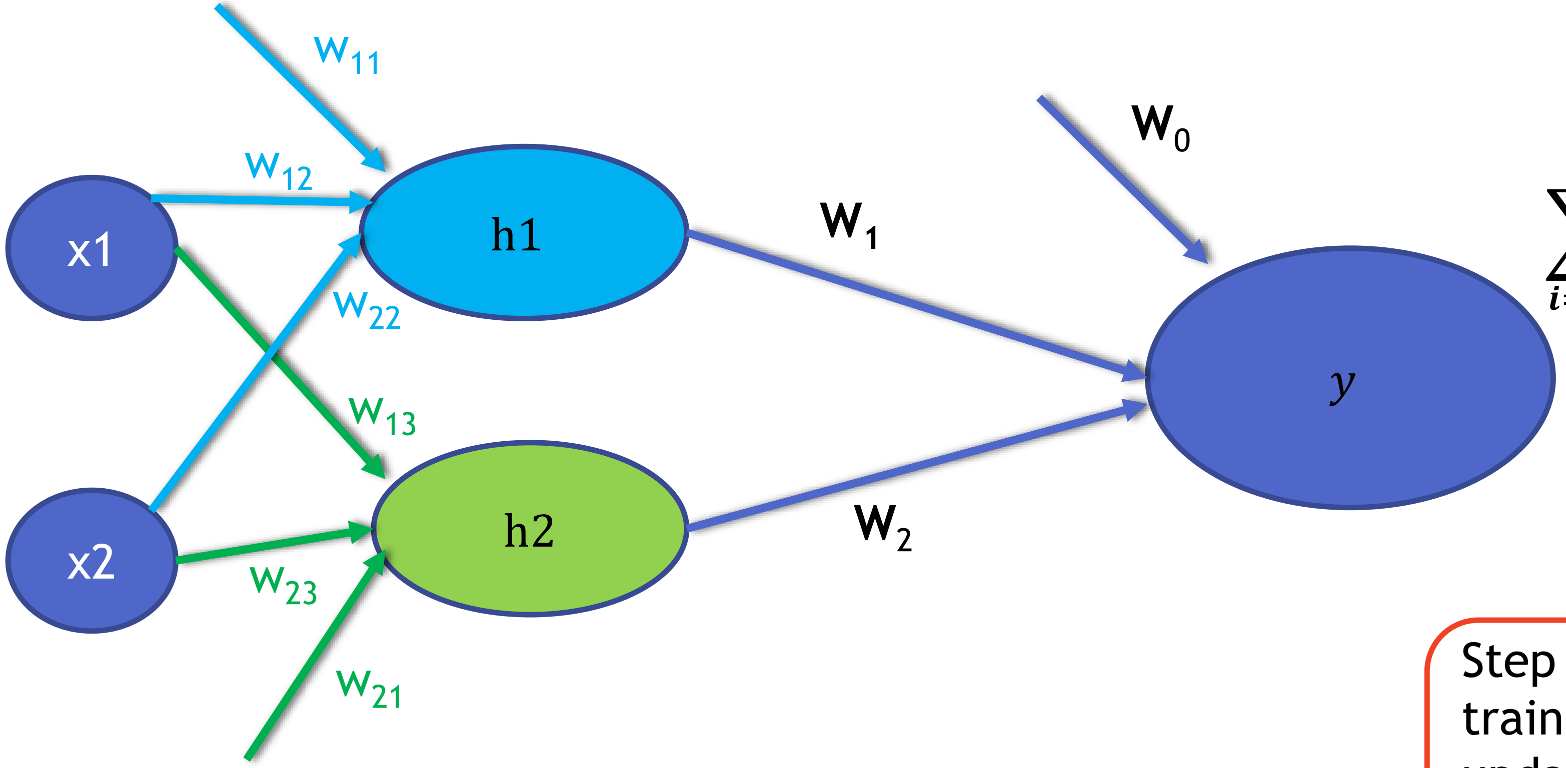
Update Weights



$$Err = \sum_{i=1}^{n} \left[y_i - g\left(\sum_{k=1}^{m} w_k h_{ki} \right) \right]^2$$

Step 4: Update the weights to reduce the error, recalculate and repeat the process

Stopping Criteria



$$\sum_{i=1}^{n}\left[y_i - g\left(\sum_{k=1}^{m} w_k h_{ki}\right)\right]^2$$

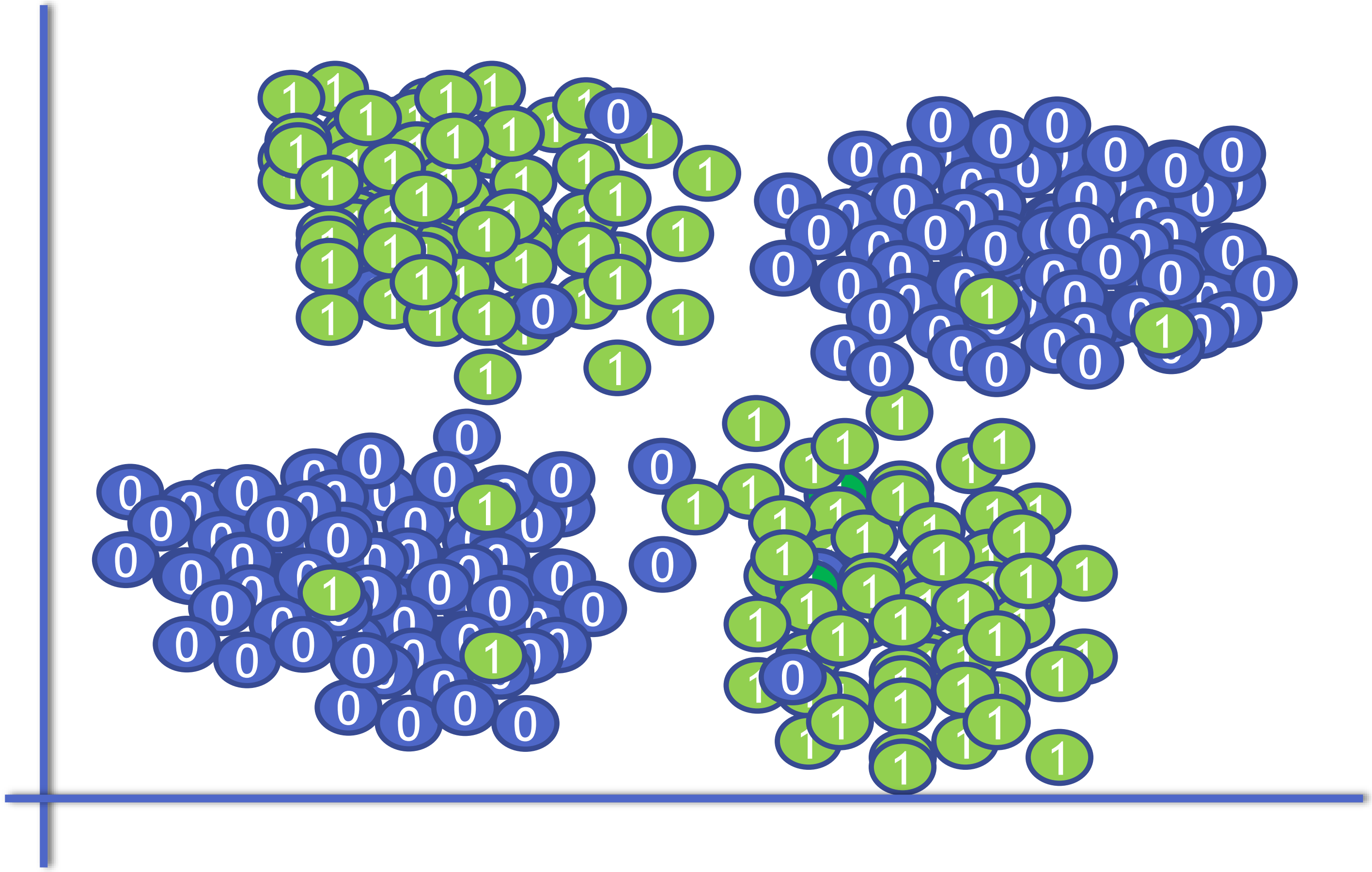
Step 5: Stop the training and weights updating process when the minimum error criteria is met

Once Again ….Neural network Algorithm

- Step 1: Initialization of weights: Randomly select some weights
- Step 2 : Training & Activation: Input the training values and perform the calculations forward.
- Step 3 : Error Calculation: Calculate the error at the outputs. Use the output error to calculate error fractions at each hidden layer
- Step 4: Weight training : Update the weights to reduce the error, recalculate and repeat the process of training & updating the weights for all the examples.
- Step 5: Stopping criteria: Stop the training and weights updating process when the minimum error criteria is met

Neural network Algorithm-Demo

Neural network Algorithm-Demo

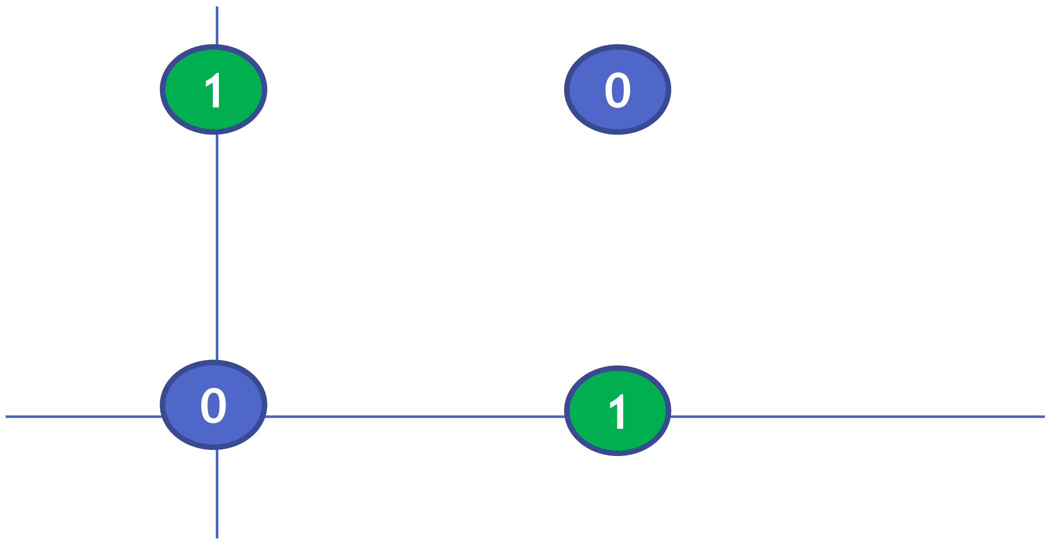


Looks like a dataset that can’t be separated by using single linear decision boundary/perceptron

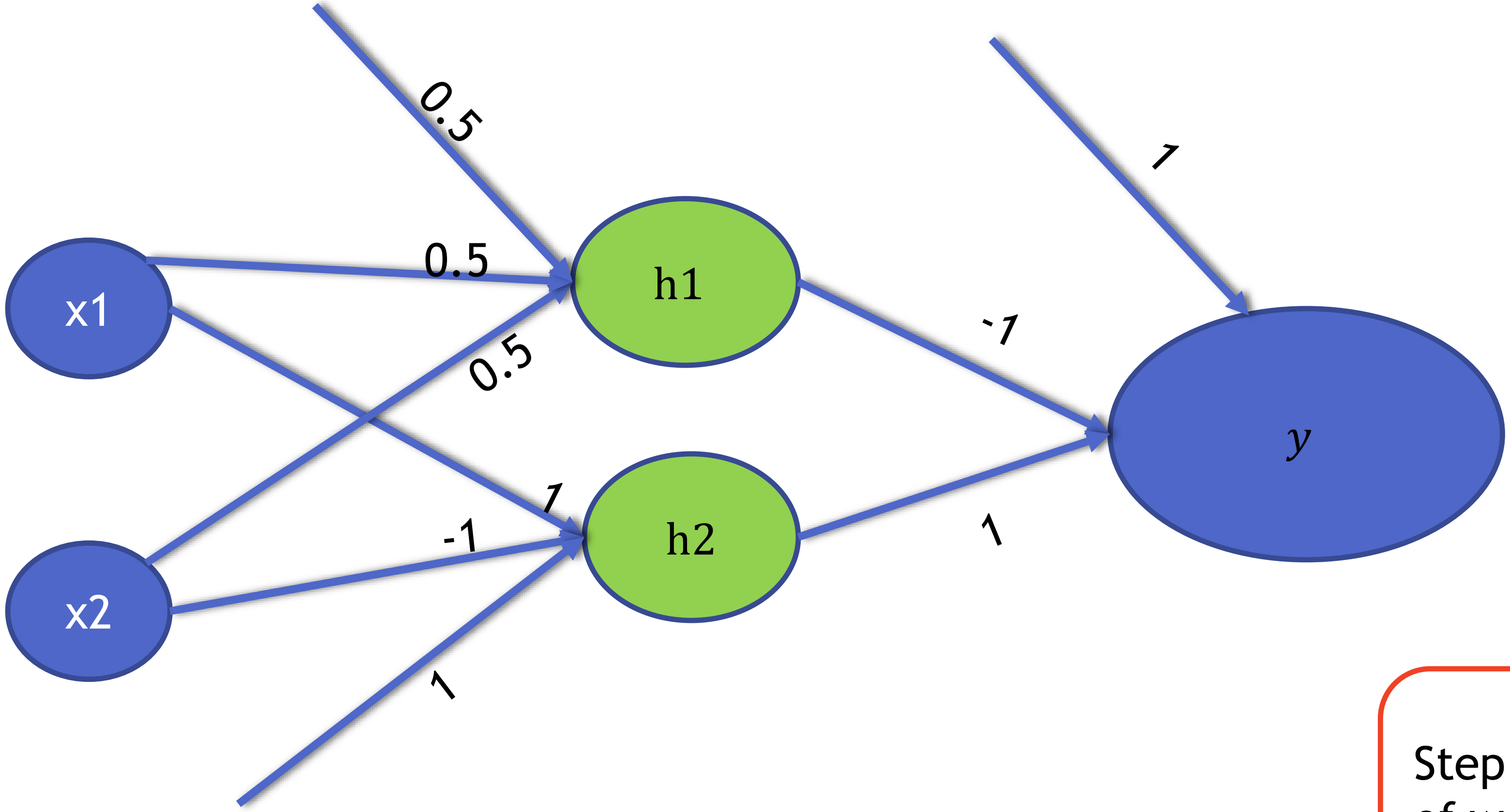
Neural network Algorithm-Demo

- Lets consider a similar but simple classification example
- XOR Gate Dataset

Input1(x1)	Input2(x2)	Output(y)
1	1	0
1	0	1
0	1	1
0	0	0



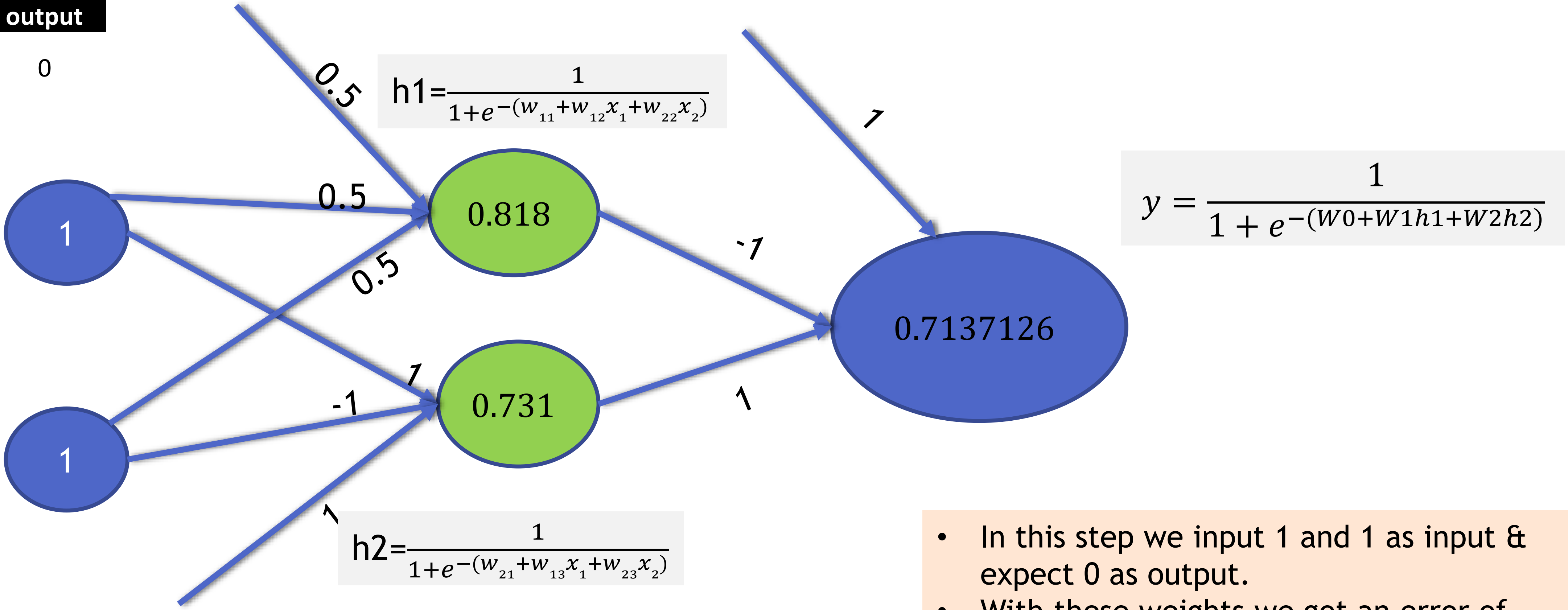
Randomly initialize weights



Step 1: Initialization of weights: Randomly select some weights

Activation

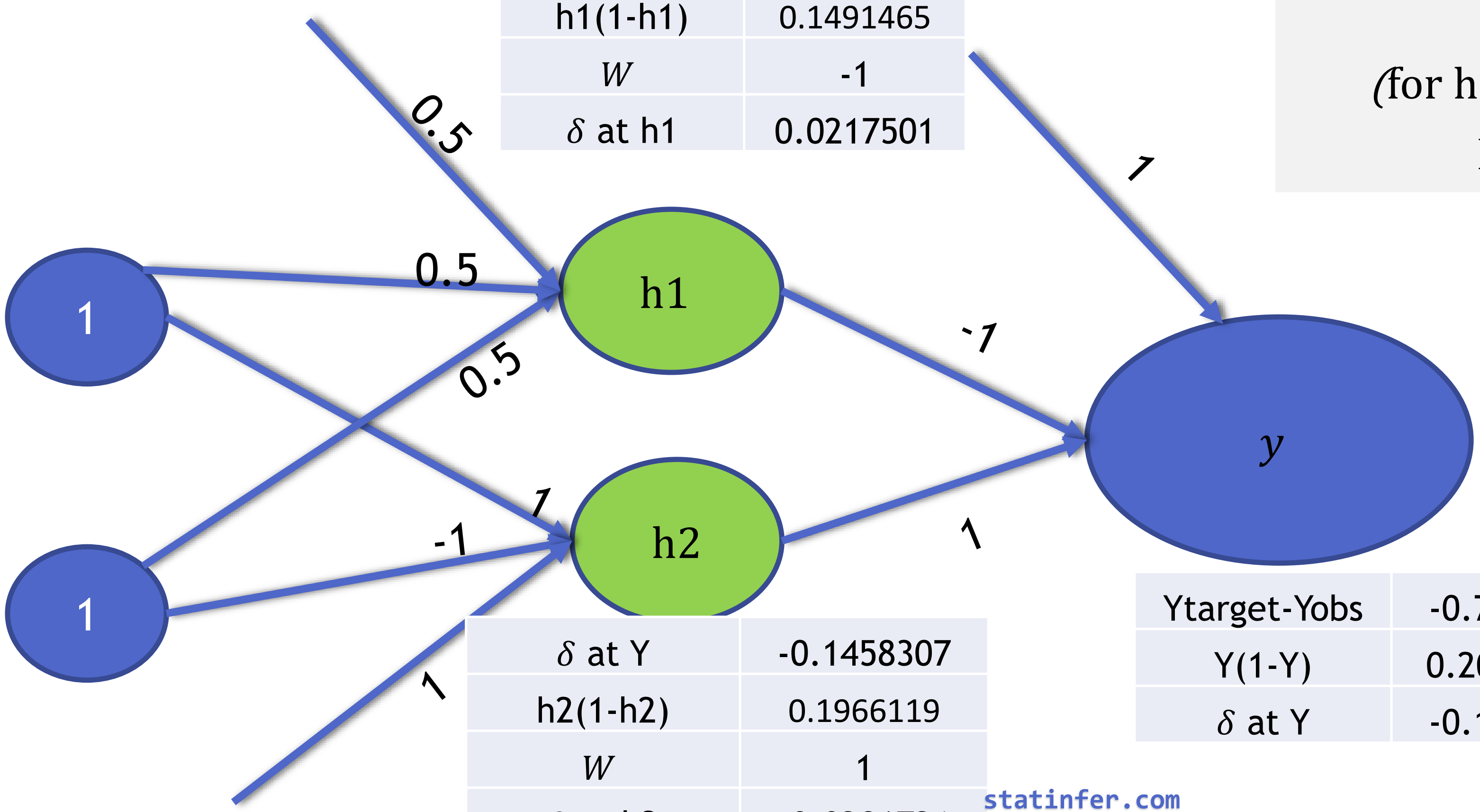
input1	input2	output
1	1	0



- In this step we input 1 and 1 as input & expect 0 as output.
- With these weights we got an error of -0.714 at output layer
- We need to adjust weights

Back-Propagate Errors

$$W_{jk} := W_{jk} + \Delta W_{jk}$$
$$where\ \Delta W_{jk} = \eta . y_j \delta_k$$
η is the learning parameter
$$\delta_k = y_k(1 - y_k) * Err$$
$$(\text{for hidden layers} \delta_k = y_k(1 - y_k) * w_j * Err)$$
Err=Expected output-Actual output

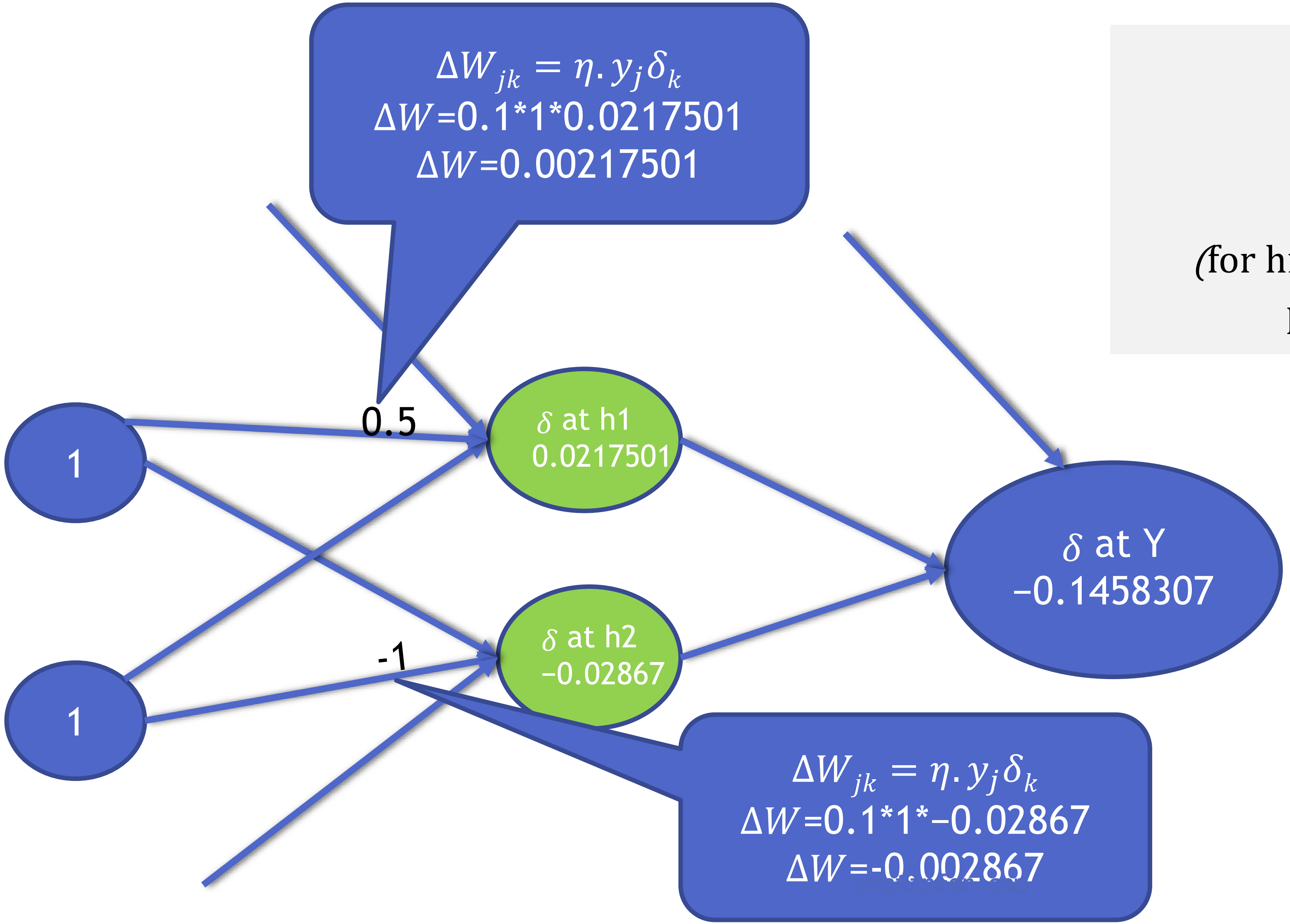


δ at Y	-0.1458307
h1(1-h1)	0.1491465
W	-1
δ at h1	0.0217501

δ at Y	-0.1458307
h2(1-h2)	0.1966119
W	1
δ at h2	-0.0286721

Ytarget-Yobs	-0.7137126
Y(1-Y)	0.20432693
δ at Y	-0.1458307

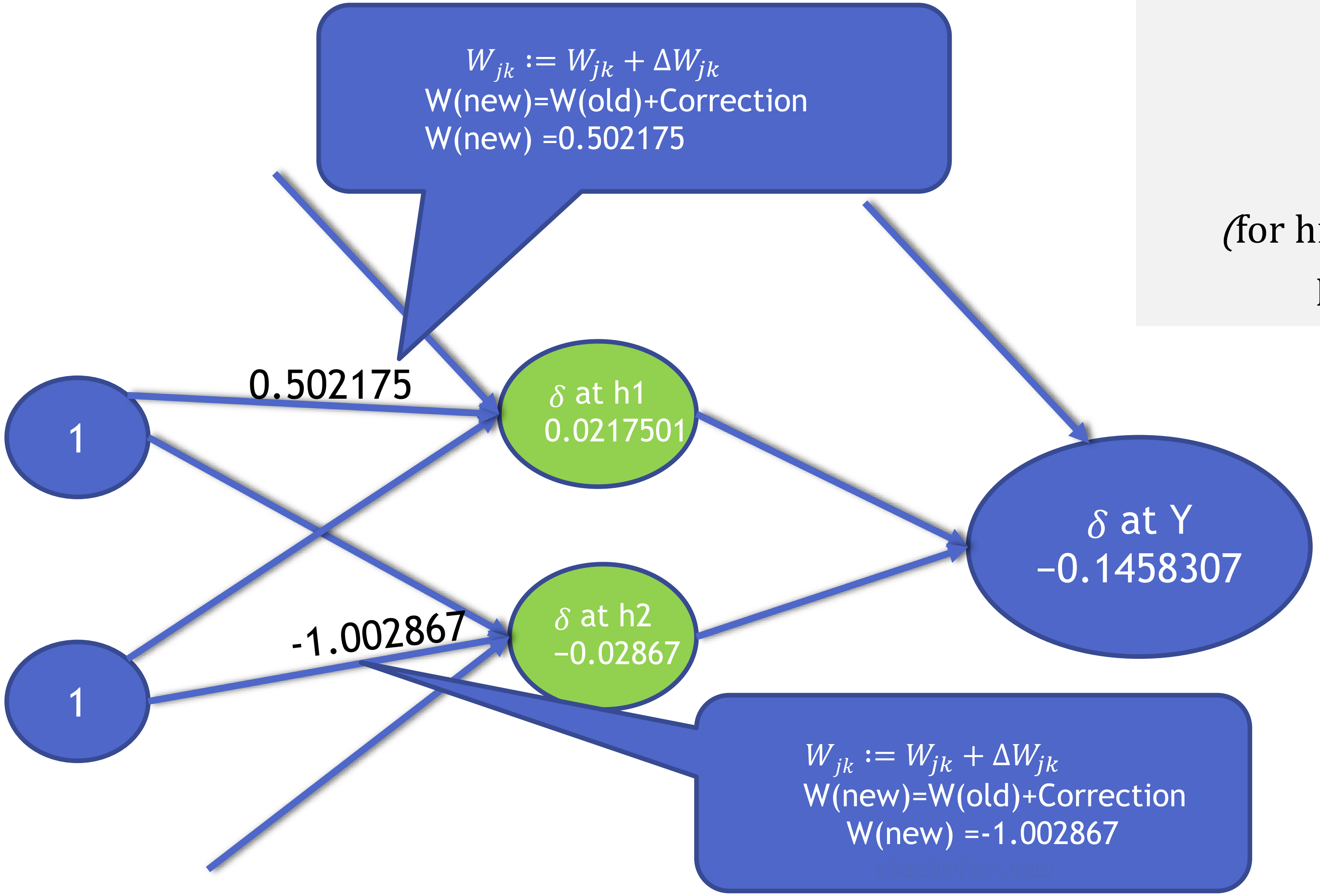
Calculate Weight Corrections



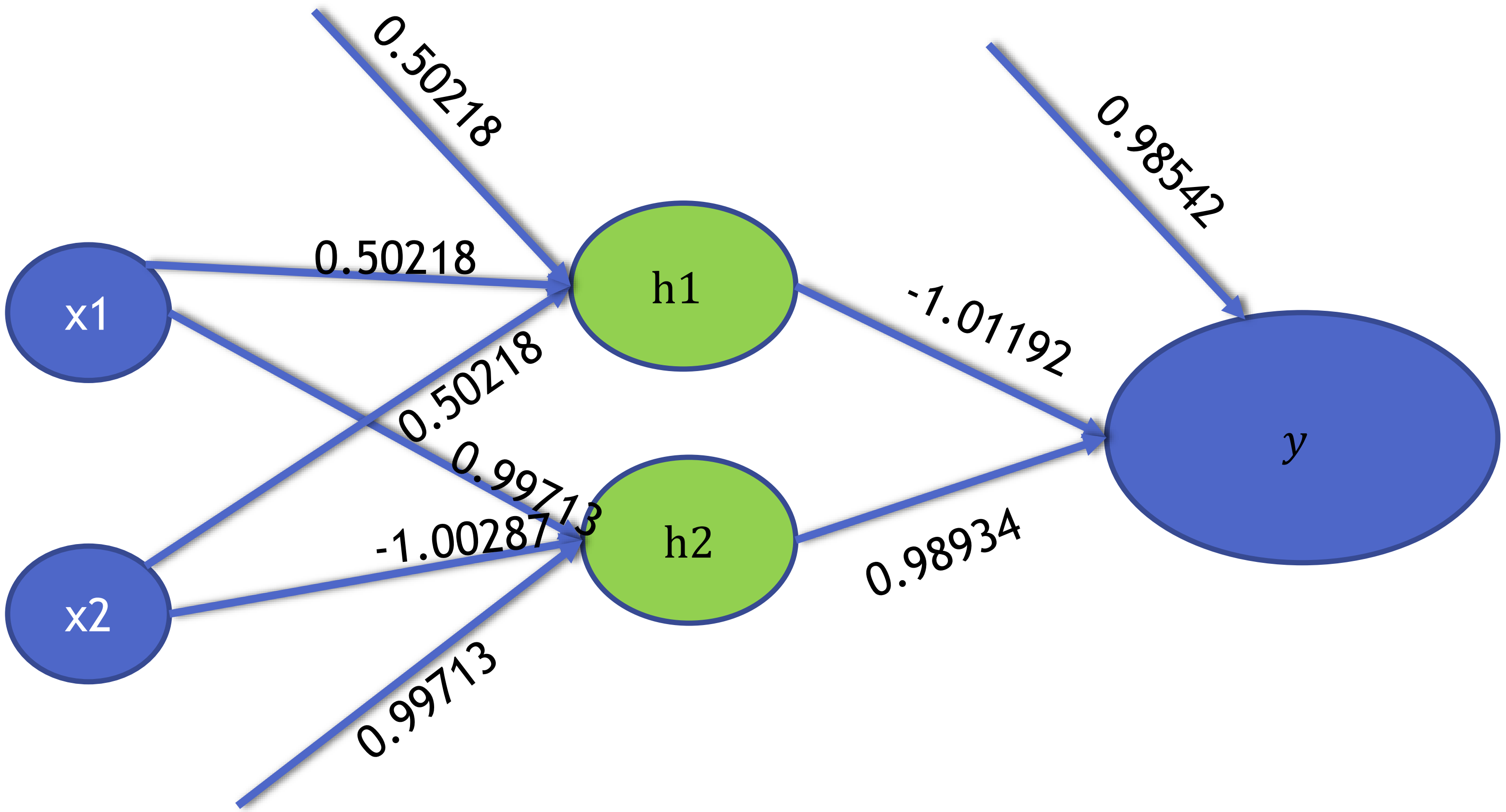
$$W_{jk} := W_{jk} + \Delta W_{jk}$$
$$where\ \Delta W_{jk} = \eta . y_j \delta_k$$
η is the learning parameter
$$\delta_k = y_k(1 - y_k) * Err$$
$$(for\ hidden\ layers\ \delta_k = y_k(1 - y_k) * w_j * Err)$$
Err=Expected output-Actual output

Updated Weights

$$W_{jk} := W_{jk} + \Delta W_{jk}$$
$$where\ \Delta W_{jk} = \eta . y_j \delta_k$$
η is the learning parameter
$$\delta_k = y_k(1 - y_k) * Err$$
$$(\text{for hidden layers}\ \delta_k = y_k(1 - y_k) * w_j * Err)$$
Err=Expected output-Actual output



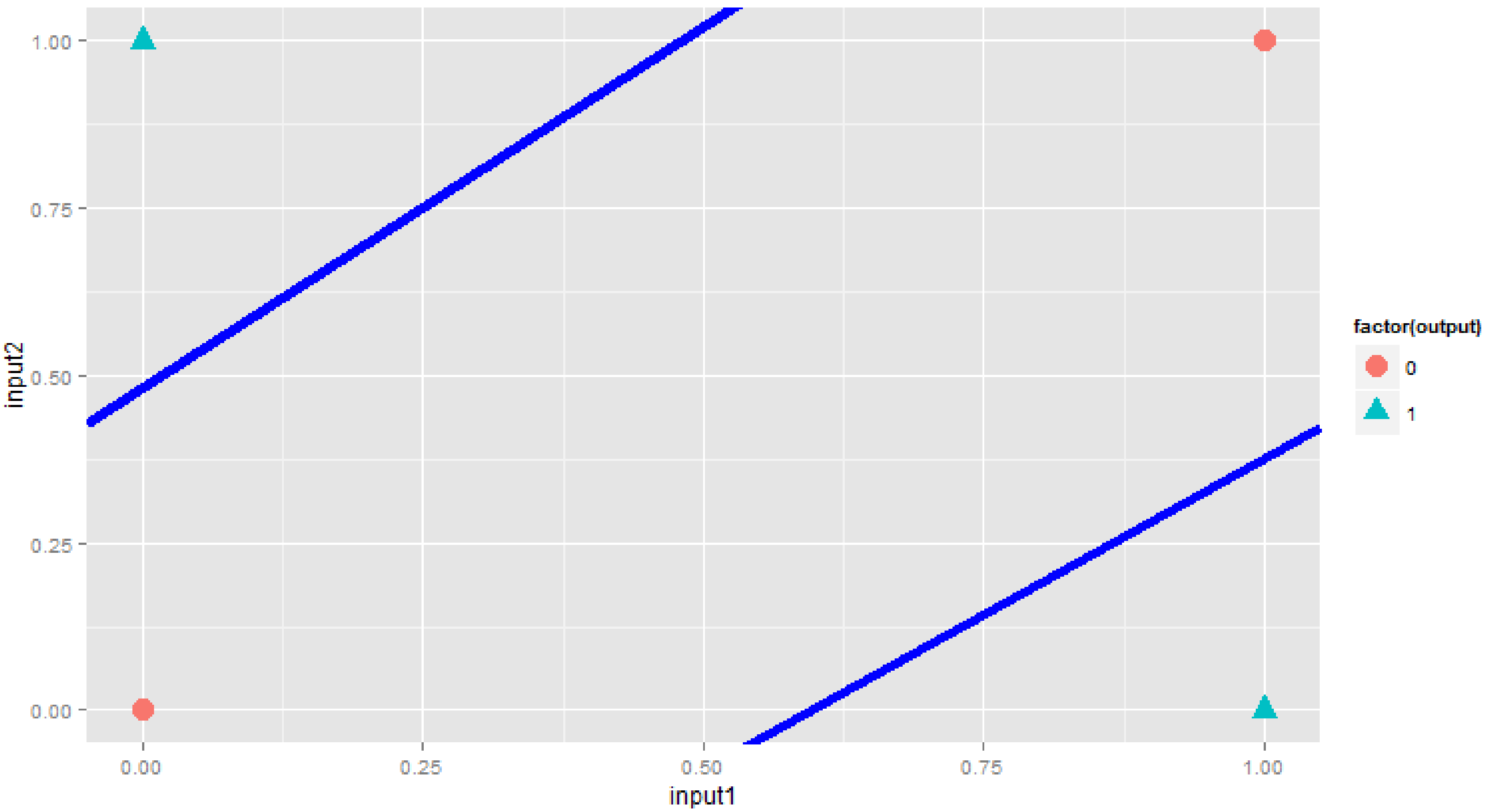
Updated Weights..contd



Iterations and Stopping Criteria

- This iteration is just for one training example (1,1,0). This is just the first epoch.
- We repeat the same process of training and updating of weights for all the data points
- We continue and update the weights until we see there is no significant change in the error or when the maximum permissible error criteria is met.
- By updating the weights in this method, we reduce the error slightly. When the error reaches the minimum point the iterations will be stopped and the weights will be considered as optimum for this training set

XOR Gate final NN Model



statinfer

Building the Neural network

The good news is..

- We don’t need to write the code for weights calculation and updating
- There readymade codes, libraries and packages available in R
- The gradient descent method is not very easy to understand for a non mathematics students
- Neural network tools don’t expect the user to write the code for the full length back propagation algorithm

Building the neural network in R

- We have a couple of packages available in R
  - net
  - neuralnet
- We need to mention the dataset, input, output & number of hidden layers as input.
- Neural network calculations are very complex. The algorithm may take sometime to produce the results
- One need to be careful while setting the parameters. The runtime changed based on the input parameter values

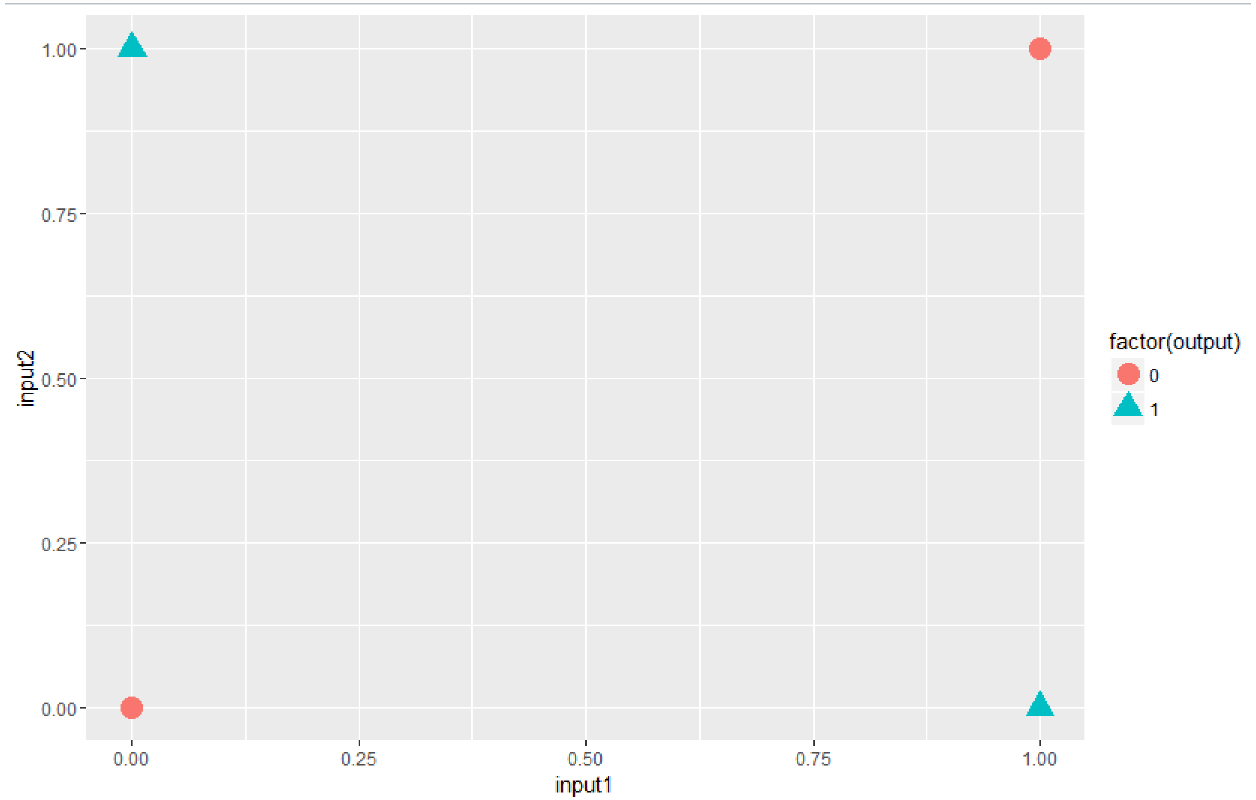
LAB: Building the neural network in R

LAB: Building the neural network in R

- Build a neural network for XOR data

Code: Building the neural network in R

```
xor_data <- read.csv("D:\\Google Drive\\Training\\Datasets\\Gates\\xor.csv")

#Graoh of data
ggplot(xor_data)+geom_point(aes(x=input1,y=input2,color=factor(output),shape=factor(output)),size=5)
```



R Code Options

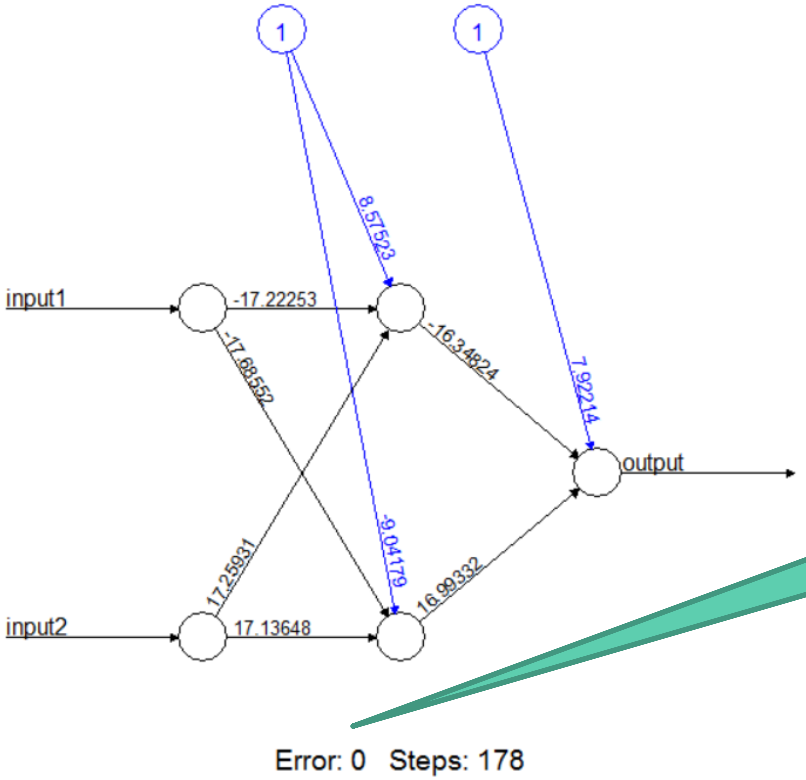
- `neuralnet(Productivity~Age+Experience,data=Emp_Productivity_raw, hidden=2, stepmax = 1e+07,  threshold=0.00001, linear.output = FALSE)`
- The number of hidden layers in the neural network. It is actually the number of nodes. We can input a vector to add more hidden layers
- Stepmax:
  - The number of iterations while executing algorithm.
  - Sometimes we may need more than 100,000 steps for the algorithm to converge.
  - Some times we may get an error “Alogorithm didn't converge with the default step max”; We need to increase the stepmax parameter value in such cases.
  - Additional info
    - One epoch one complete run of training data. If epoch=500 then algorithm sees the entire data set 500 times
    - One iteration is the number of times a "batch" of data passed through the algorithm(Steps). If batch size is same as full training data then iterations is equal to epochs

R Code Options

- Threshold
  - Connected to weights optimization on error function
  - By default, neuralnet requires the model partial derivative error to change at least 0.01 otherwise it will stop changing.
  - It can be used as a stopping criteria. If the partial derivative of error function reaches this threshold then the algorithm will stop.
  - A lower threshold value will force the algorithm for more iterations and accuracy.
- The output is expected to be linear by default. We need to specifically mention linear.output = FALSE for classification problems

Code: Building the neural network in R

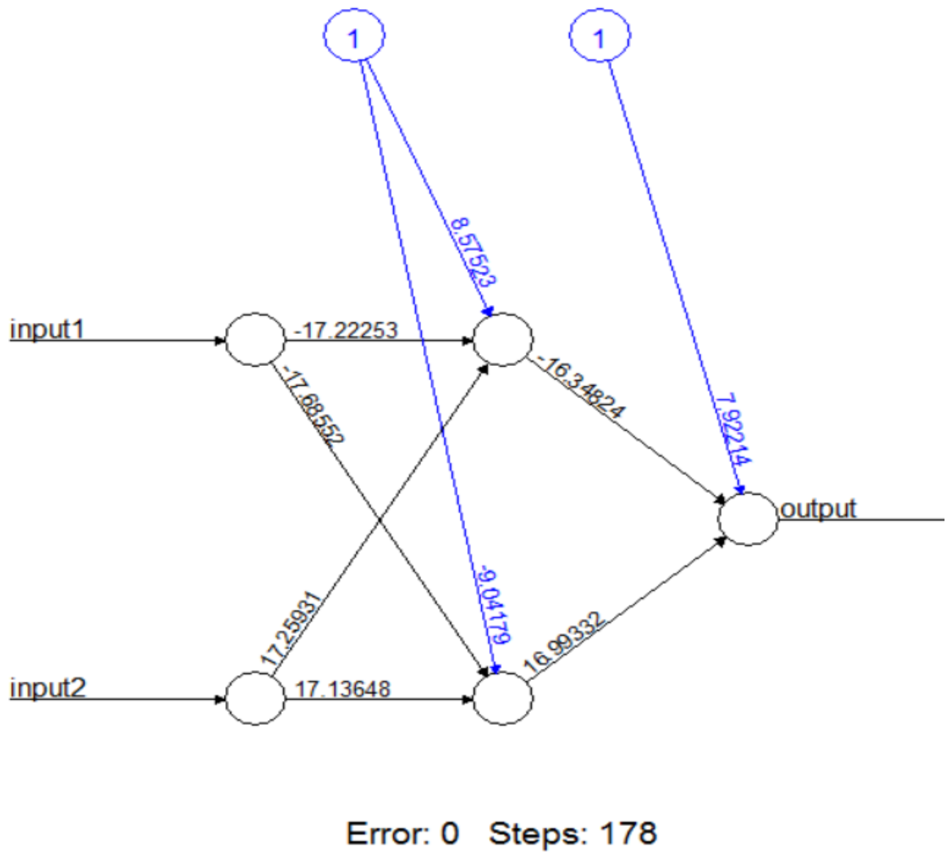
```
#Building Neuralnet
library(neuralnet)
xor_nn_model<-neuralnet(output~input1+input2,data=xor_data,hidden=2, linear.output = FALSE, threshold = 0.
plot(xor_nn_model)
```



Execute a couple of times to get zero error

Code: Building the neural network in R

```
> xor_nn_model$result.matrix
                                      1
error                0.000000013277397218
reached.threshold    0.000000007184425943
steps              178.000000000000000000
Intercept.to.1layhid1  8.575226977886682 53
input1.to.1layhid1   -17.222525222404101 44
input2.to.1layhid1    17.259306877649368 77
Intercept.to.1layhid2 -9.041785364430241 50
input1.to.1layhid2   -17.685518638634139 42
input2.to.1layhid2    17.136484678493776 10
Intercept.to.output    7.922139214924015 96
1layhid.1.to.output  -16.348237048568122 99
1layhid.2.to.output   16.993319597155156 94
```

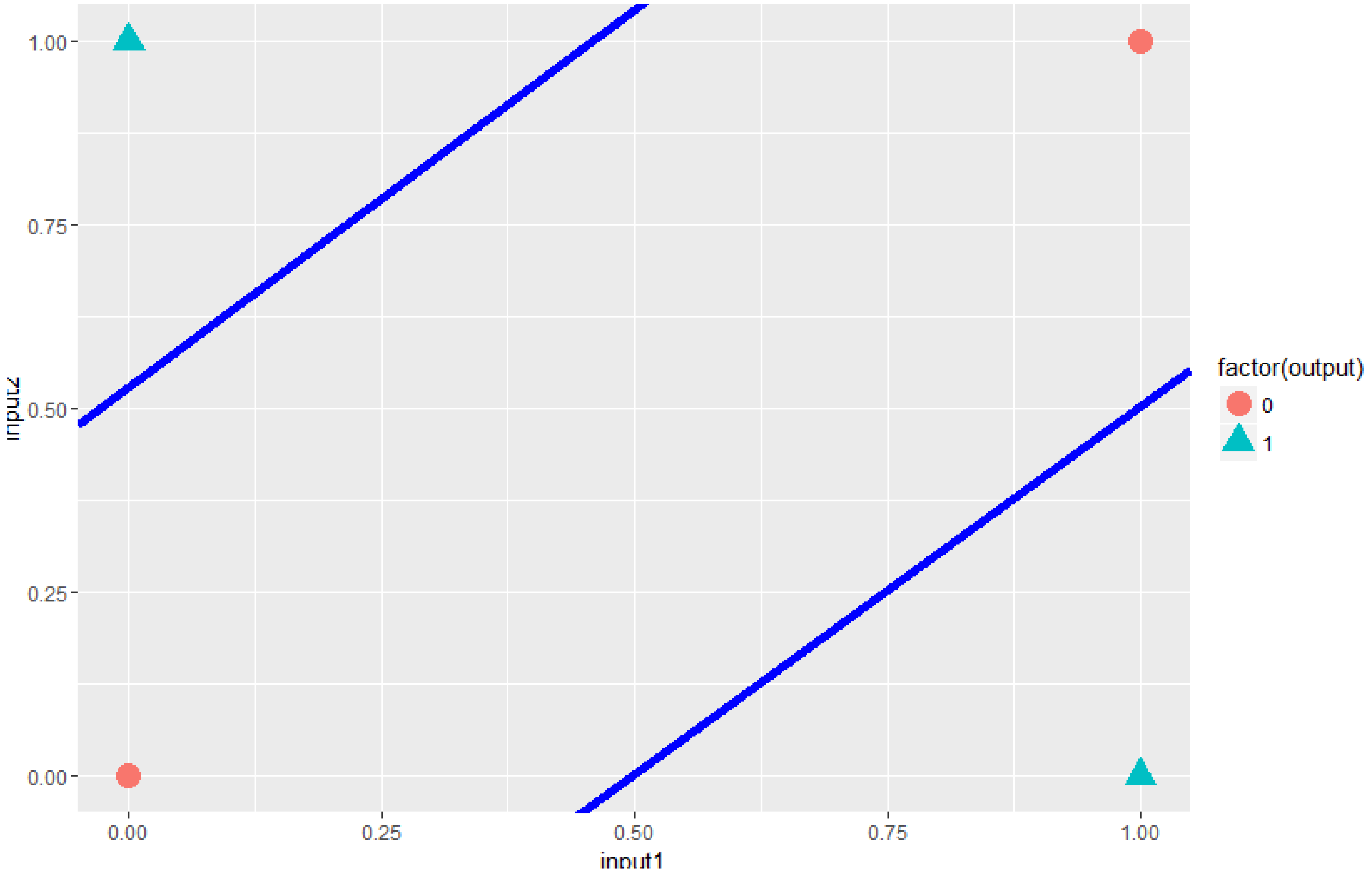



```
> round(xor_nn_model$result.matrix,5)
                              1
error                   0.00000
reached.threshold       0.00000
steps                 178.00000
Intercept.to.1layhid1   8.57523
input1.to.1layhid1    -17.22253
input2.to.1layhid1     17.25931
Intercept.to.1layhid2  -9.04179
input1.to.1layhid2    -17.68552
input2.to.1layhid2     17.13648
Intercept.to.output     7.92214
1layhid.1.to.output   -16.34824
1layhid.2.to.output    16.99332
```

Code: Building the neural network in R

```
#Decision Boundaries
m1_slope <- xor_nn_model$weights[[1]][[1]][2]/(-xor_nn_model$weights[[1]][[1]][3])
m1_intercept <- xor_nn_model$weights[[1]][[1]][1]/(-xor_nn_model$weights[[1]][[1]][3])

m2_slope <- xor_nn_model$weights[[1]][[1]][5]/(-xor_nn_model$weights[[1]][[1]][6])
m2_intercept <- xor_nn_model$weights[[1]][[1]][4]/(-xor_nn_model$weights[[1]][[1]][6])

####Drawing the Decision boundary

library(ggplot2)
base<-ggplot(xor_data)+geom_point(aes(x=input1,y=input2,color=factor(output),shape=factor(output)),size=5)
base+geom_abline(intercept = m1_intercept , slope = m1_slope, colour = "blue", size = 2) +geom_abline(inte
```

Code: Building the neural network in R

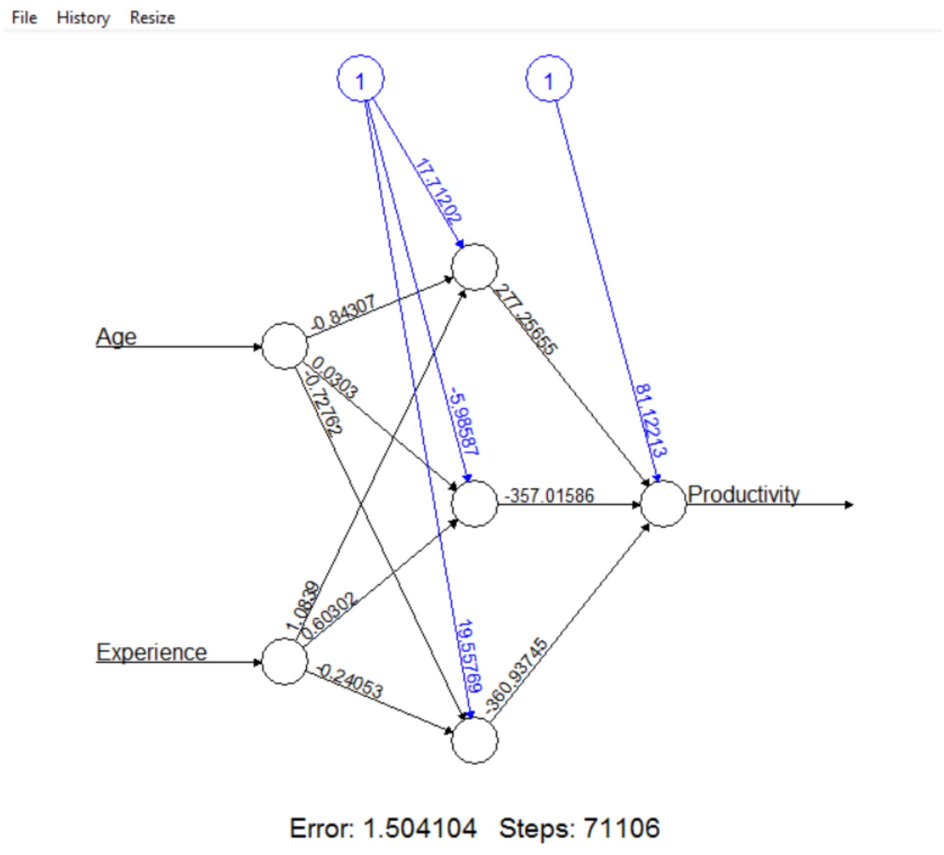


Lab: Building Neural network on Employee productivity data

- Dataset: Emp_Productivity/Emp_Productivity.csv
- Draw a 2D graph between age, experience and productivity
- Build neural network algorithm to predict the productivity based on age and experience
- Plot the neural network with final weights
- Increase the hidden layers and see the change in accuracy

Code: Neural network on Employee productivity data

```
> library(neuralnet)
>
> Emp_Productivity_nn_model1<-neuralnet(Productivity~Age+Experience,data=Emp Productivity raw, hidden=3,linear.o
utput = FALSE)
> plot(Emp_Productivity_nn_model1)
>
```



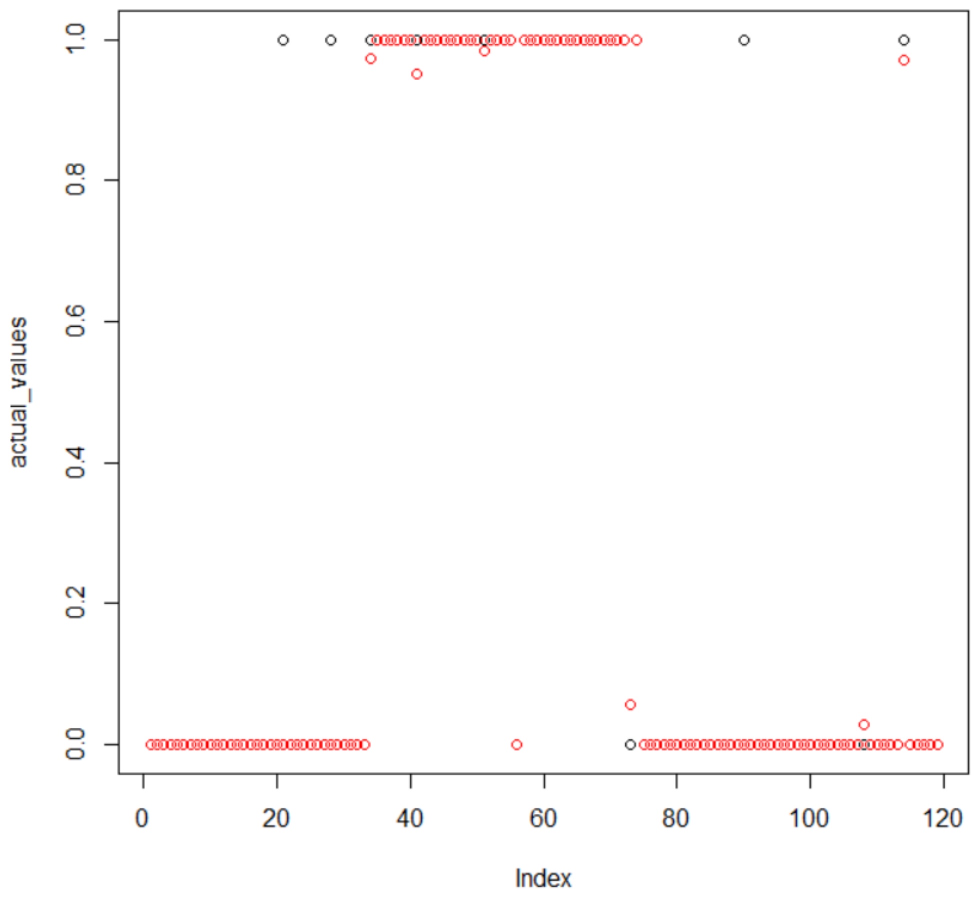
Code: Neural network on Employee productivity data

```
> ####Results and Intime validation
> actual_values<-Emp_Productivity_raw$Productivity
> Predicted<-Emp_Productivity_nn_model1$net.result[[1]]
> Predicted
                [,1]
1    6.027897831e-07
2    2.790358544e-04
3    3.704230941e-10
4    8.896327007e-07
5    4.493230845e-06
```

```
> #The root mean square error
> sqr_err<-(actual_values-Predicted)^2
> sum(sqr_err)
[1] 3.008207292
> mean(sqr_err)
[1] 0.02527905287
> sqrt(mean(sqr_err))
[1] 0.1589938768
```

Code: Neural network on Employee productivity data

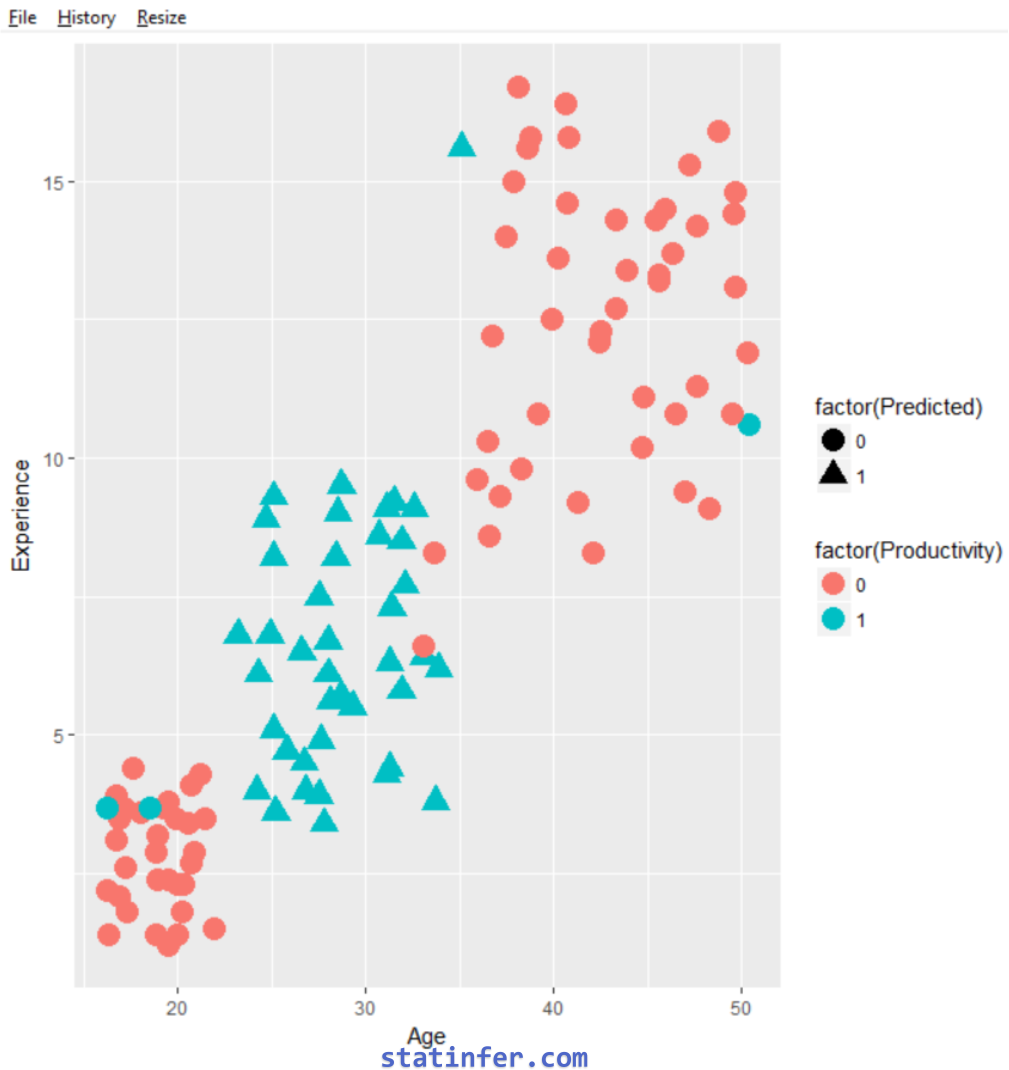
```
#Plottig Actual and Predicted
plot(actual_values)
points(Predicted, col=2)
```



Code: Neural network on Employee productivity data

```
> #Plottig Actual and Predicted using ggplot
> library(ggplot2)
> library(reshape2)
> act_pred_df<-data.frame(actual_values,Predicted)
> act_pred_df$id<-rownames(act_pred_df)
> act_pred_df_melt = melt(act_pred_df, id.vars ="id")
> ggplot(act_pred_df_melt,aes(id, value, colour = variable)) + geom_point()
>
> ##Plotting Actual and Predicted using ggplot on classification graph
>
> Emp_Productivity_pred_act<-data.frame(Emp_Productivity_raw,Predicted=round(Predicted,0))
> library(ggplot2)
> #Graph without predictions
> ggplot(Emp_Productivity_pred_act)+geom_point(aes(x=Age,y=Experience,color=factor(Productivity)),size=5)
>
> #Graph with predictions
> ggplot(Emp_Productivity_pred_act)+geom_point(aes(x=Age,y=Experience,color=factor(Productivity),shape=factor(
Predicted)),size=5)
```

Code: Neural network on Employee productivity data

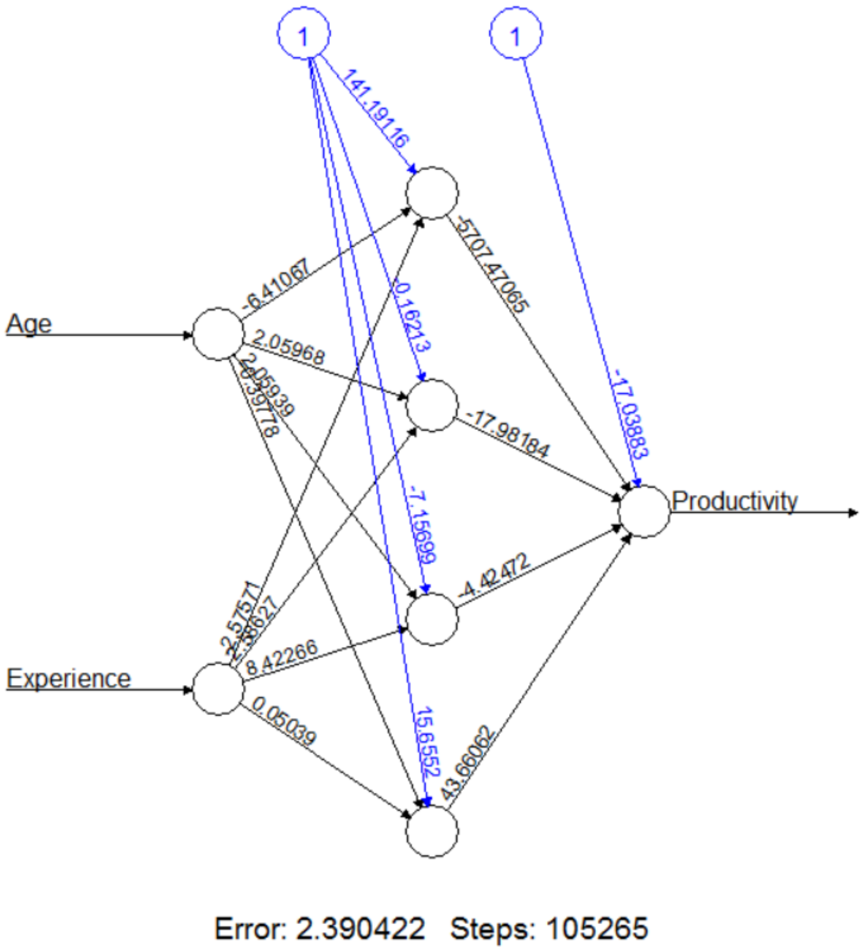


statinfer.com

Code: Neural network on Employee productivity data

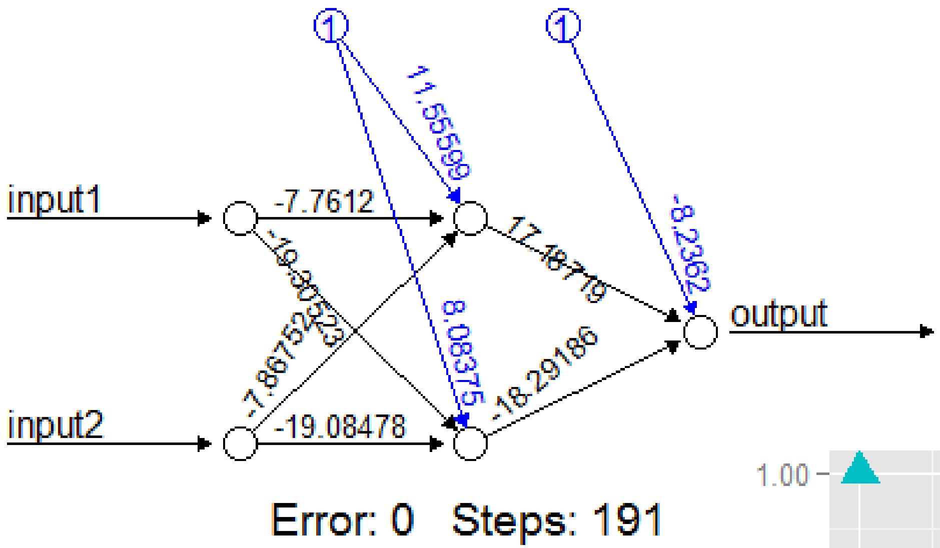
```
> ############
> #There is an issue with the local minimum. if you see the error is high, then you can rebuild the model
> Emp_Productivity_nn_model1<-neuralnet(Productivity~Age+Experience,data=Emp_Productivity_raw, hidden=3,linear
.output = FALSE)
> plot(Emp_Productivity_nn_model1)
>
>
> #########Further increasing hidden layers
> Emp_Productivity_nn_model3<-neuralnet(Productivity~Age+Experience,data=Emp_Productivity_raw, hidden=4,thresh
old=0.001, stepmax = 1e+07, linear.output = FALSE)
> plot(Emp_Productivity_nn_model3)
>
```

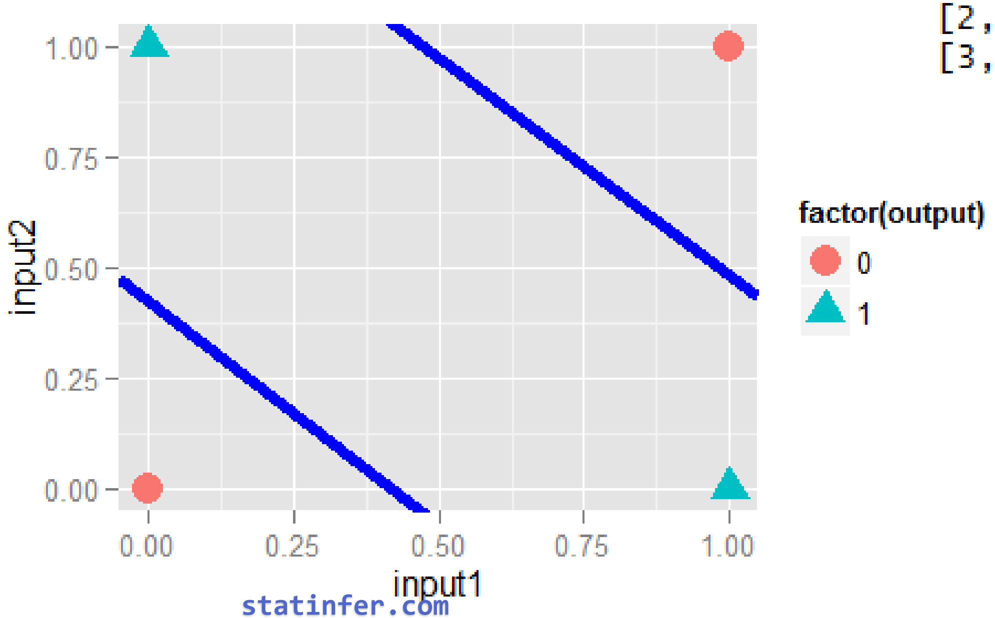
Code: Neural network on Employee productivity data



There can be many solutions

Set-1

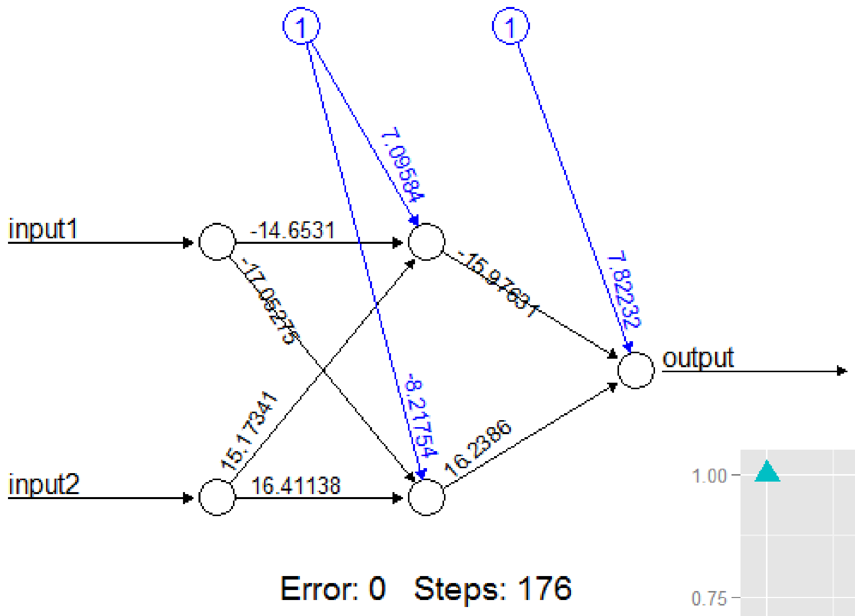


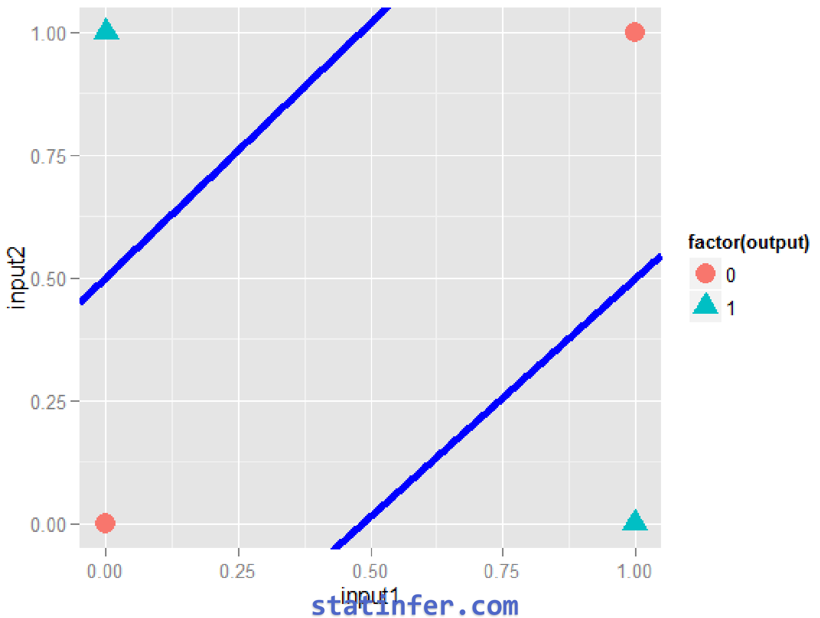



```
[[1]]
[[1]][[1]]
             [,1]          [,2]
[1,]  11.555990554   8.083746755
[2,]  -7.761201080 -19.305228584
[3,]  -7.867521277 -19.084777064

[[1]][[2]]
              [,1]
[1,]   -8.236203034
[2,]   17.187194824
[3,]  -18.291860830
```

There can be many solutions

Set-2

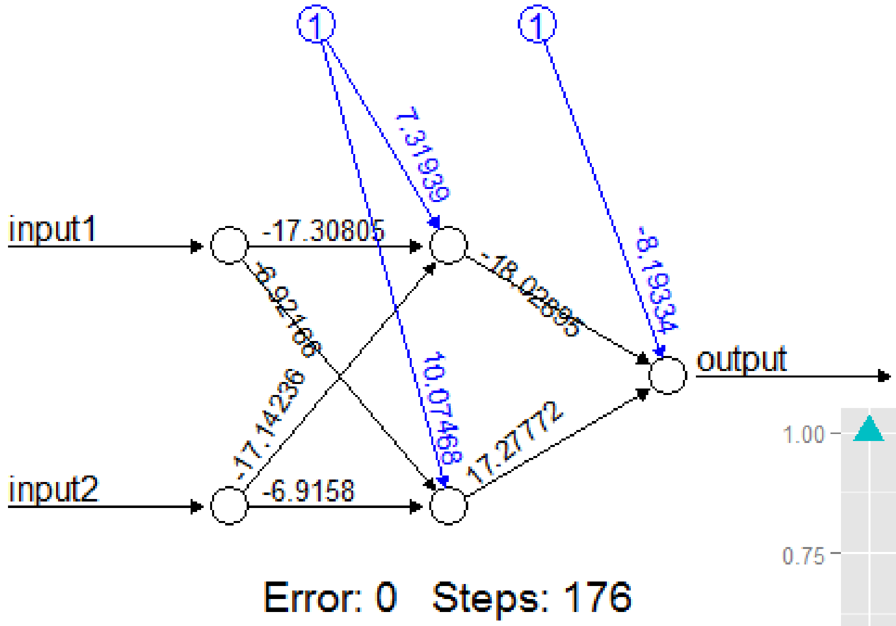


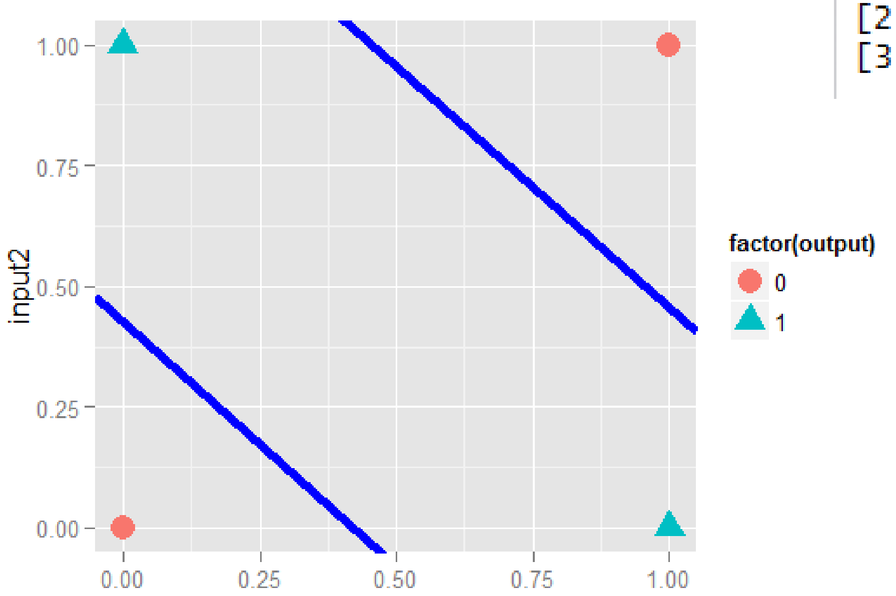



```
[[1]]
[[1]][[1]]
             [,1]         [,2]
[1,]   7.095840734  -8.217540628
[2,] -14.653102714 -17.052746020
[3,]  15.173411386  16.411376870

[[1]][[2]]
             [,1]
[1,]   7.822316873
[2,] -15.976309524
[3,]  16.238604904
```

There can be many solutions

Set-3



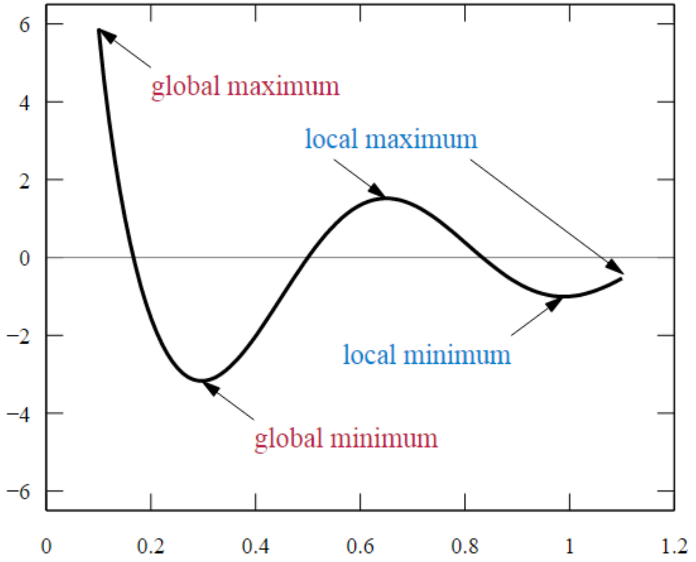


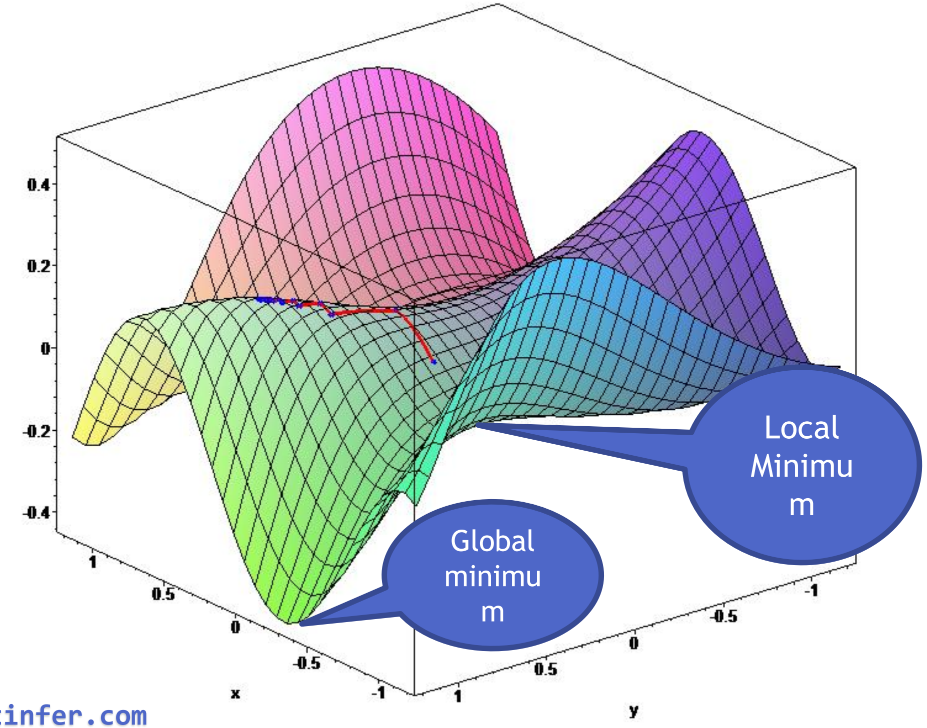

```
[[1]]
[[1]][[1]]
             [,1]         [,2]
[1,]   7.319389339 10.074676203
[2,] -17.308047777 -6.921658365
[3,] -17.142359203 -6.915800778

[[1]][[2]]
             [,1]
[1,]  -8.193335979
[2,] -18.028951740
[3,]  17.277723643
```

Local vs. Global Minimum

Local vs. Global Minimum

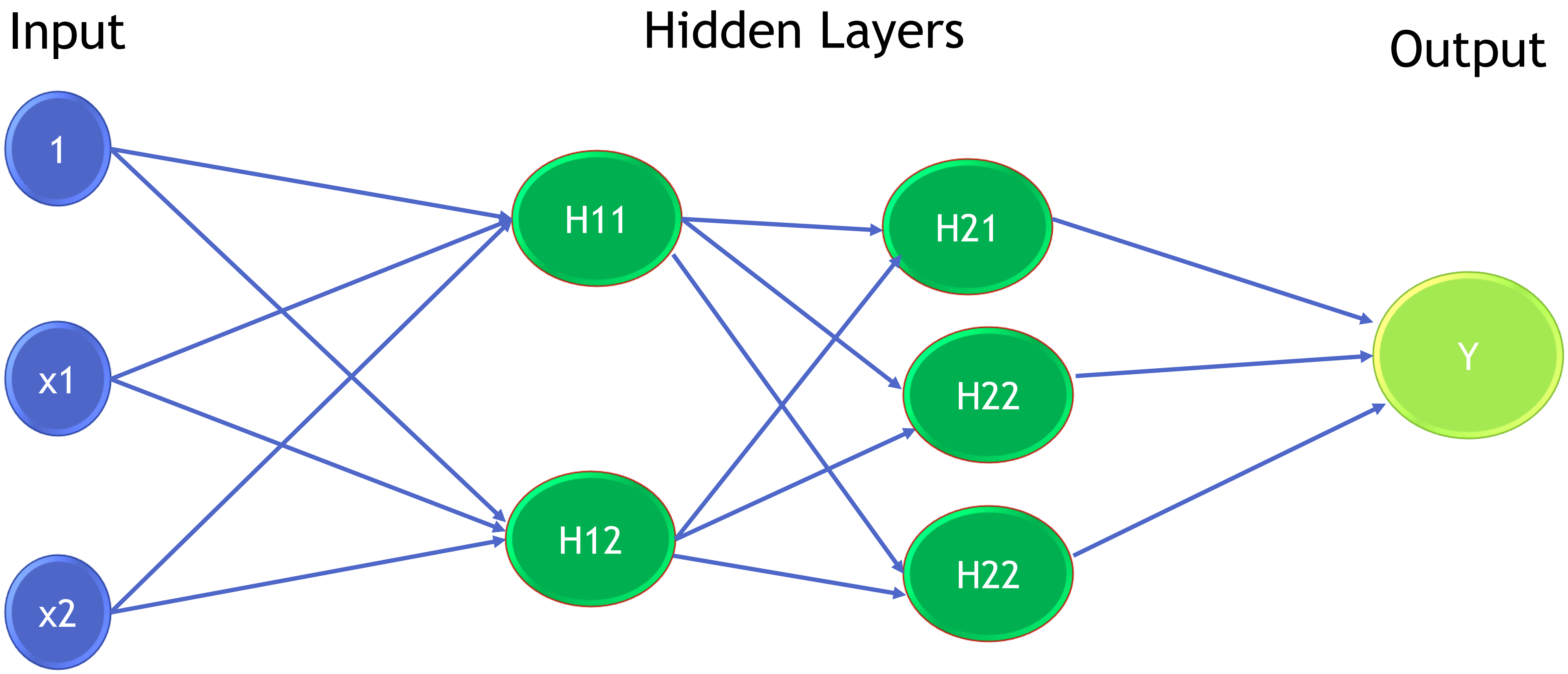
- The neural network might give different results with different start weights.
- The algorithm tries to find the local minima rather than global minima.
- There can be many local minima's, which means there can be many solutions to neural network problem
- We need to perform the validation checks before choosing the final model.



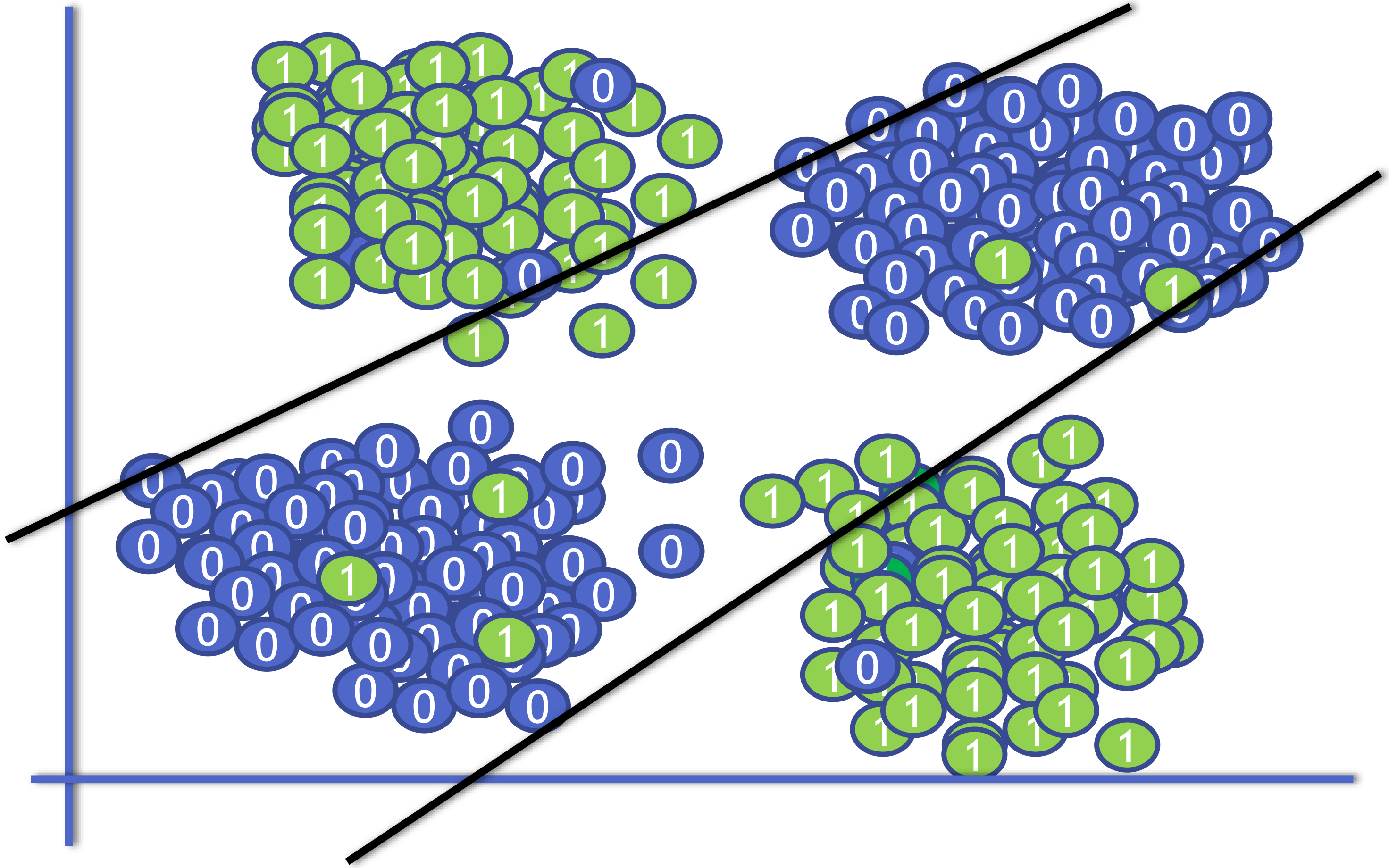


Hidden layers and their role

Multi Layer Neural Network



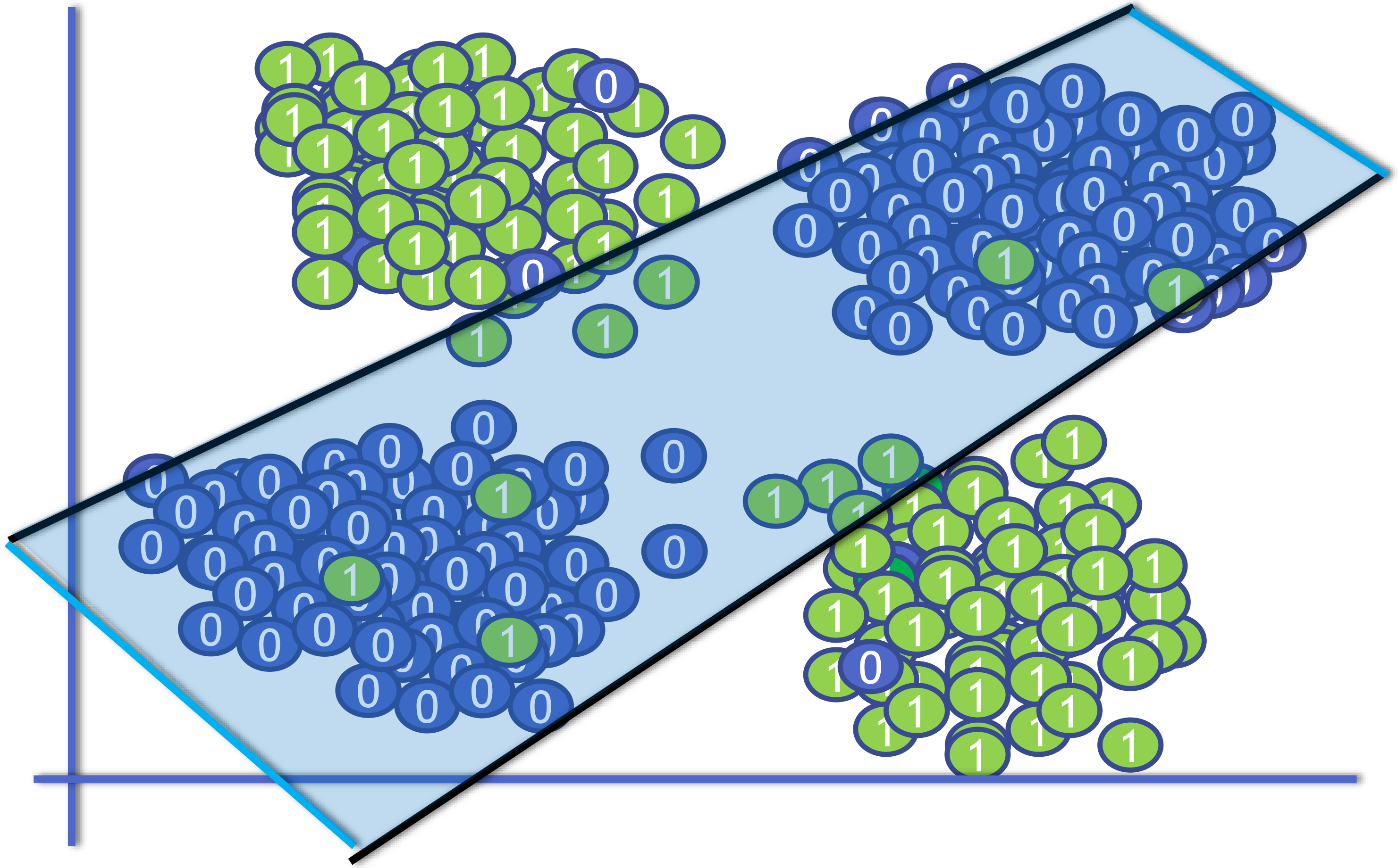
The role of hidden layers



- The First hidden layer
- The first layer is nothing but the liner decision boundaries
- The simple logistic regression line outputs
- We can see them as multiple lines on the decision space

![statinfer]

The role of hidden layers



- The Second hidden layer
- The Second layer combines these lines and forms simple decision boundary shapes
- The third hidden layer forms even complex shapes within the boundaries generated by second layer.
- You can imagine All these layers together divide the whole objective space into multiple decision boundary shapes, the cases within the shape are class-1 outside the shape are class-2

The Number of hidden layers

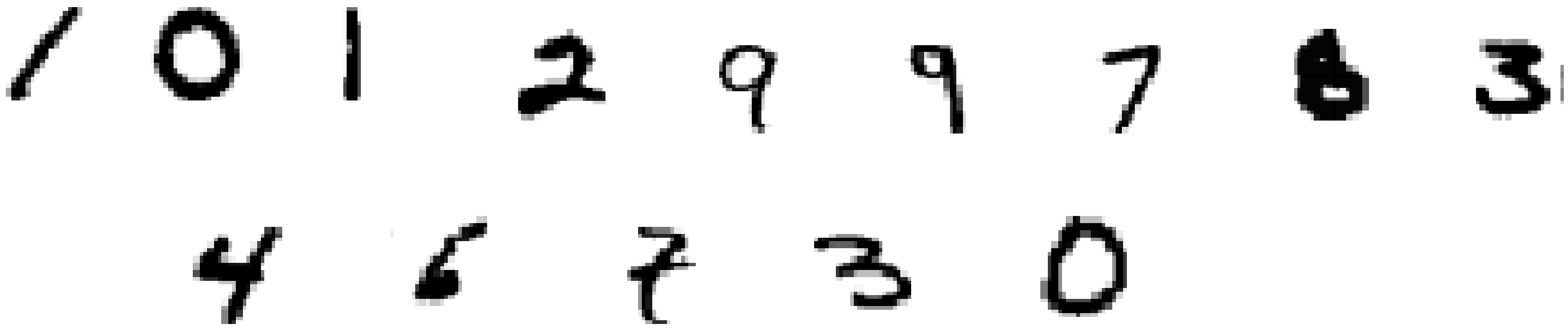
The Number of hidden layers

- There is no concrete rule to choose the right number. We need to choose by trail and error validation
- Too few hidden layers might result in imperfect models. The error rate will be high
- High number of hidden layers might lead to over-fitting, but it can be identified by using some validation techniques
- The final number is based on the number of predictor variables, training data size and the complexity in the target.
- When we are in doubt, its better to go with many hidden nodes than few. It will ensure higher accuracy. The training process will be slower though
- Cross validation and testing error can help us in determining the model with optimal hidden layers

LAB: Digit Recognizer

LAB: Digit Recognizer

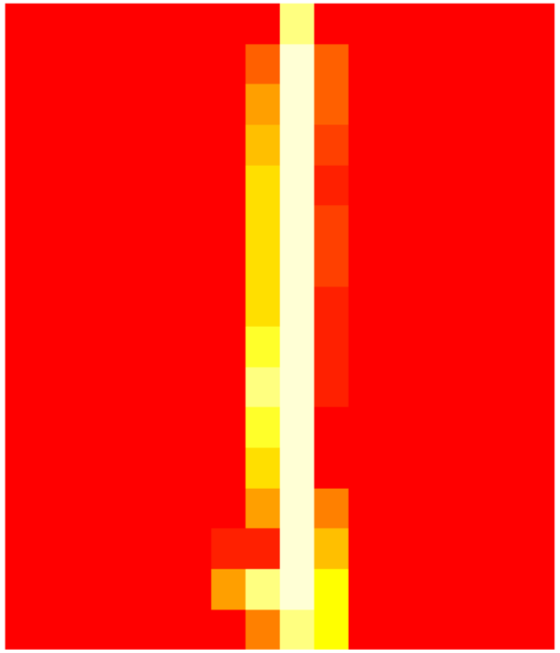
- Take an image of a handwritten single digit, and determine what that digit is.
- Normalized handwritten digits, automatically scanned from envelopes by the U.S. Postal Service. The original scanned digits are binary and of different sizes and orientations; the images here have been de slanted and size normalized, resultingin 16 x 16 grayscale images (Le Cun et al., 1990).
- The data are in two gzipped files, and each line consists of the digitid (0-9) followed by the 256 grayscale values.
- Build a neural network model that can be used as the digit recognizer
- Use the test dataset to validate the true classification power of the model
- What is the final accuracy of the model?



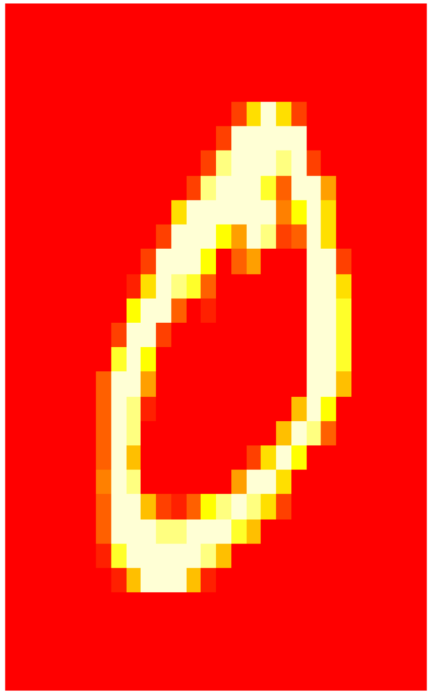
Code: Digit Recognizer

```
#Importing test and training data - USPS Data
digits_train <- read.table("D:\\Google Drive\\Training\\Datasets\\Digit Recognizer\\USPS\\zip.train.txt", quote="\"", comment.char="")
digits_test <- read.table("D:\\Google Drive\\Training\\Datasets\\Digit Recognizer\\USPS\\zip.test.txt", quote="\"", comment.char="")

dim(digits_train)

col_names <- names(digits_train[,-1])
label_levels<-names(table(digits_train$V1))

#Lets see some images.
for(i in 1:10)
{
data_row<-digits_train[i,-1]
pixels = matrix(as.numeric(data_row),16,16,byrow=TRUE)
image(pixels, axes = FALSE)
title(main = paste("Label is" , digits_train[i,1]), font.main = 4)
}
```

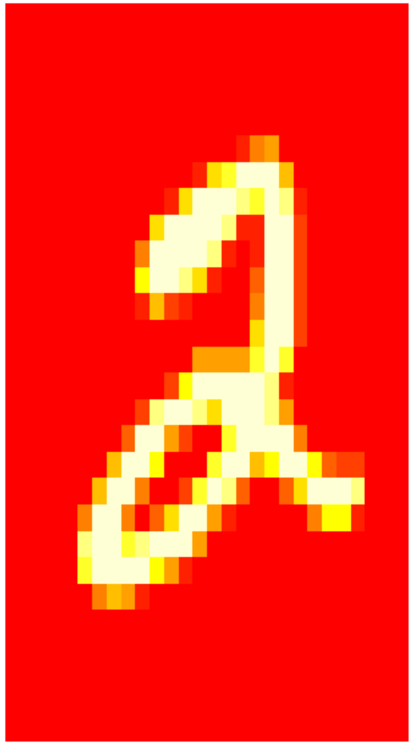
Code: Digit Recognizer



Label is 1



Label is 0



Label is 2

Code: Digit Recognizer

```
#####Creating multiple columns for multiple outputs
#####We need these variables while building the model
digit_labels<-data.frame(label=digits_train[,1])
for (i in 1:10)
    {
    digit_labels<-cbind(digit_labels, digit_labels$label==i-1)
    names(digit_labels)[i+1]<-paste("l",i-1,sep="")
    }

label_names<-names(digit_labels[,-1])

#Update the training dataset
digits_train1<-cbind(digits_train,digit_labels)
names(digits_train1)


#formula y~. doesn't work in neuralnet function
model_form <- as.formula(paste(paste(label_names, collapse = " + "), "~", paste(col_names,
collapse = " + ")))
```

Code: Digit Recognizer

```
########################The Model
pc <- proc.time()#Lets keep an eye on runtime

library(neuralnet)
Digit_model<-neuralnet(model_form, data=digits_train1, hidden=15,linear.output=FALSE)
summary(Digit_model)

proc.time() - pc

#######Prediction  on holdout data
test_predicted<-data.frame(compute(Digit_model,digits_test[,-1])$net.result)
```

Code: Digit Recognizer

```
> library(neuralnet)
> Digit_model<-neuralnet(model_form, data=digits_train1, hidden=15,linear.output=FALSE)

> pc <- proc.time()#Lets keep an eye on runtime
>
> library(neuralnet)
> Digit_model<-neuralnet(model_form, data=digits_train1, hidden=15,linear.output=FALSE)
> summary(Digit_model)
                    Length  Class      Mode
call                      5 -none-     call
response              72910 -none-     logical
covariate           1866496 -none-     numeric
model.list                2 -none-     list
err.fct                   1 -none-     function
act.fct                   1 -none-     function
linear.output             1 -none-     logical
data                    268 data.frame list
net.result                1 -none-     list
weights                   1 -none-     list
startweights              1 -none-     list
generalized.weights       1 -none-     list
result.matrix          4018 -none-     numeric
>
> proc.time() - pc
   user  system elapsed
 180.95    3.66  185.67
```

Code: Digit Recognizer

```
#######Prediction  on holdout data
test_predicted<-data.frame(compute(Digit_model,digits_test[,-1])$net.result)

########Collating all labels into a single column
pred_label<-0
for(i in 1:nrow(test_predicted))
  {
  pred_label[i]<-which.max(apply(test_predicted[i,],MARGIN=2,min))-1
    }
test_predicted$pred_label<-pred_label

###Confusion Matrix and Accuracy
library(caret)

confuse<-confusionMatrix(test_predicted$pred_label,digits_test$V1)
confuse
confuse$overall
```

Code: Digit Recognizer

```
> test_predicted$pred_label<-pred_label
>
> ###Confusion Matrix and Accuracy
> library(caret)
>
> confuse<-confusionMatrix(test_predicted$pred_label,digits_test$V1)
> confuse
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2   3   4   5   6   7   8   9
         0 341   0   8   7   2   9   4   0   1   0
         1   0 247   1   0   1   0   0   0   0   0
         2   2   5 165   7   6   1   2   4   3   1
         3   3   3   6 135   1  10   0   6   2   0
         4   3   3   5   1 165   3   5   4   5   4
         5   1   0   1   9   2 124   6   1  10   0
         6   5   5   1   1   5   6 153   0   1   0
         7   1   0   3   1   5   0   0 127   2   2
         8   3   1   8   4   8   3   0   1 141   1
         9   0   0   0   1   5   4   0   4   1 169

Overall Statistics

               Accuracy : 0.8804185
```

Code: Digit Recognizer

```
Statistics by Class:

                     Class: 0  Class: 1   Class: 2   Class: 3   Class: 4   Class: 5   Class: 6   Class: 7   Class: 8   Class: 9
Sensitivity         0.9498607 0.9356061 0.83333333 0.81325301 0.82500000 0.77500000 0.90000000 0.86394558 0.84939759 0.95480226
Specificity         0.9811893 0.9988526 0.98286346 0.98316133 0.98173769 0.98375744 0.98693522 0.99247312 0.98424769 0.99180328
Pos Pred Value      0.9166667 0.9919679 0.84183673 0.81325301 0.83333333 0.80519481 0.86440678 0.90070922 0.82941176 0.91847826
Neg Pred Value      0.9889908 0.9903299 0.98177802 0.98316133 0.98065229 0.98057205 0.99071038 0.98928189 0.98639085 0.99561163
Prevalence          0.1788739 0.1315396 0.09865471 0.08271051 0.09965122 0.07972098 0.08470354 0.07324365 0.08271051 0.08819133
Detection Rate      0.1699053 0.1230693 0.08221226 0.06726457 0.08221226 0.06178376 0.07623318 0.06327853 0.07025411 0.08420528
Detection Prevalence 0.1853513 0.1240658 0.09765820 0.08271051 0.09865471 0.07673144 0.08819133 0.07025411 0.08470354 0.09167912
Balanced Accuracy   0.9655250 0.9672293 0.90809840 0.89820717 0.90336884 0.87937872 0.94346761 0.92820935 0.91682264 0.97330277
> confuse$overall
      Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull AccuracyPValue  McnemarPValue 
  0.8804185351   0.8656984584   0.8654156673   0.8942997061   0.1788739412   0.0000000000            NaN 
```

Real-world applications

Real-world applications

- Self driving car by taking the video as input
- Speech recognition
- Face recognition
- Cancer cell analysis
- Heart attack predictions
- Currency predictions and stock price predictions
- Credit card default and loan predictions
- Marketing and advertising by predicting the response probability
- Weather forecasting and rainfall prediction

Real-world applications

- Face recognition :
  - https://www.youtube.com/watch?v=57VkfXqJ1LU
  - https://www.youtube.com/watch?v=xVQLBbXdVUY
- Autonomous car software
  - https://www.youtube.com/watch?v=gG72-SjwxAM

statinfer

Drawbacks of Neural Networks

Drawbacks of Neural Networks

- No real theory that explains how to choose the number of hidden layers
- Takes lot of time when the input data is large, needs powerful computing machines
- Difficult to interpret the results. Very hard to interpret and measure the impact of individual predictors
- Its not easy to choose the right training sample size and learning rate.
- The local minimum issue. The gradient descent algorithm produces the optimal weights for the local minimum, the global minimum of the error function is not guaranteed

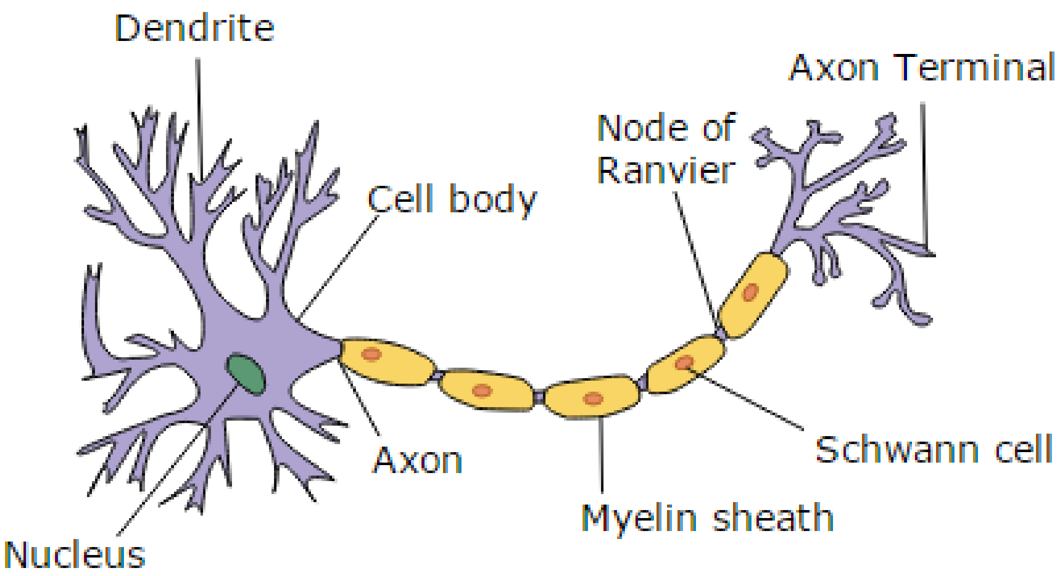
Why the name neural network?

statinfer

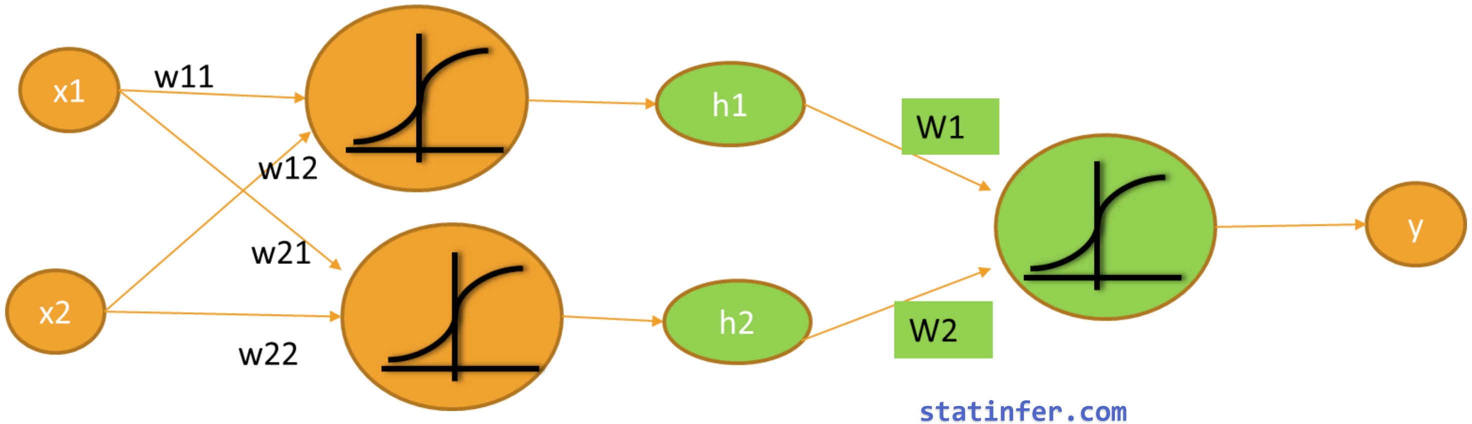
Why the name neural network?

- The neural network algorithm for solving complex learning problems is inspired by human brain
- Our brains are a huge network of processing elements. It contains a network of billions of neurons.
- In our brain, a neuron receives input from other neurons. Inputs are combined and send to next neuron
- The artificial neural network algorithm is built on the same logic.

Why the name neural network?



Dendrites→ Input(X)
Cell body → Processor(Σwx)
Axon → Output(Y)



Conclusion

statinfer

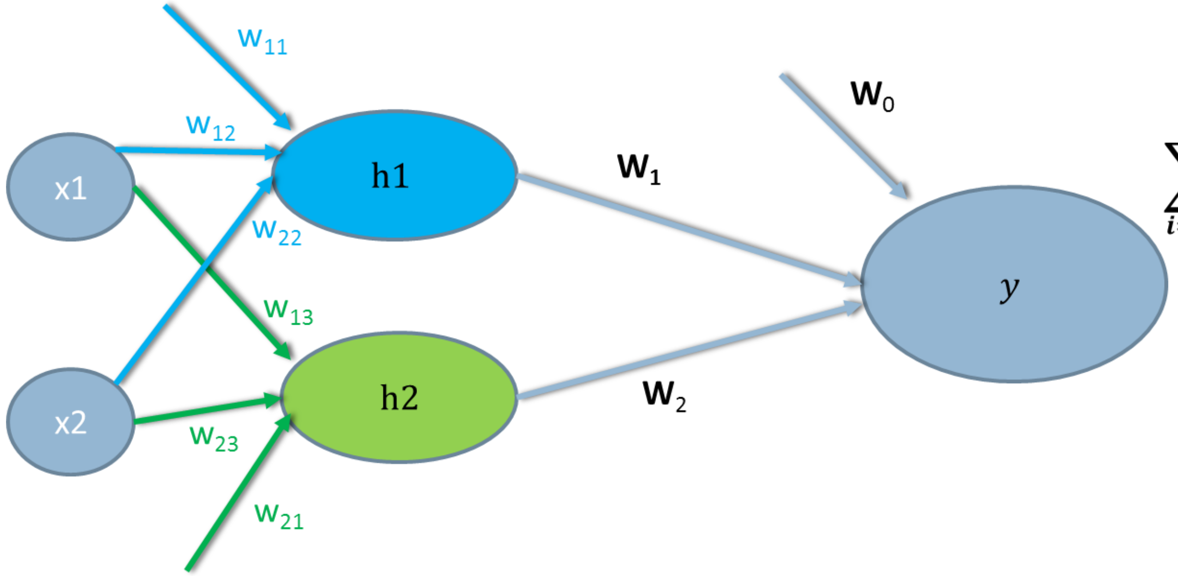
Conclusion

- Neural network is a vast subject. Many data scientists solely focus on only Neural network techniques
- In this session we practiced the introductory concepts only. Neural Networks has much more advanced techniques. There are many algorithms other than back propagation.
- Neural networks particularly work well on some particular class of problems like image recognition.
- The neural networks algorithms are very calculation intensive. They require highly efficient computing machines. Large datasets take significant amount of runtime on R. We need to try different types of options and packages.
- Currently there is a lot of exciting research is going on, around neural networks.
- After gaining sufficient knowledge in this basic session, you may want to explore reinforced learning, deep learning etc.,

Appendix

Math- How to update the weights?

Math- How to update the weights?



$$\sum_{i=1}^{n}\left[y_i - g\left(\sum_{k=1}^{m} w_k h_{ki}\right)\right]^2$$

- We update the weights backwards by iteratively calculating the error
- The formula for weights updating is done using gradient descent method or delta rule also known as Widrow-Hoff rule
- First we calculate the weight corrections for the output layer then we take care of hidden layers

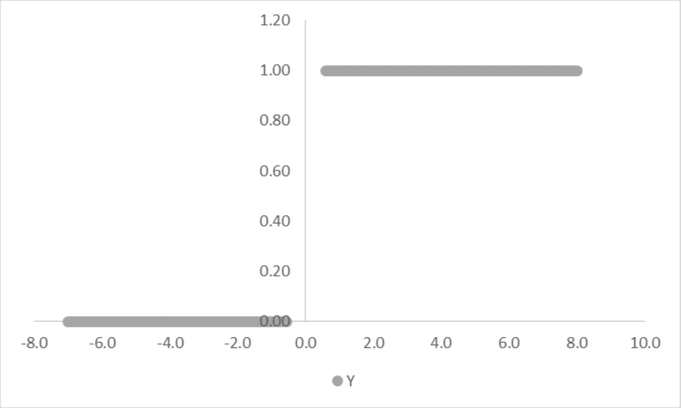
Math- How to update the weights?

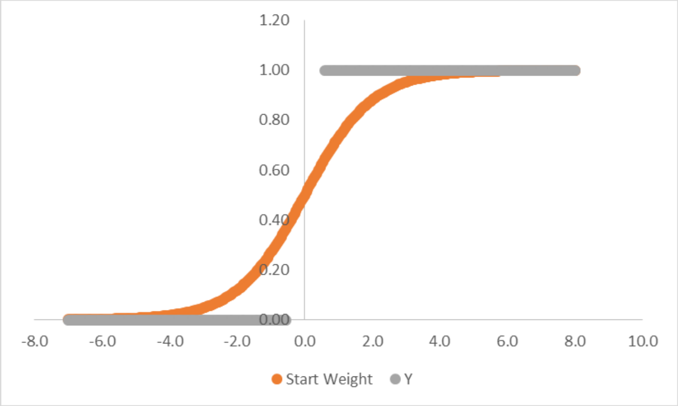
- $W_{jk} := W_{jk} + \Delta W_{jk}$
  - $where\ \Delta W_{jk} = \eta . y_j \delta_k$
  - η is the learning parameter
  - $\delta_k = y_k(1 - y_k) * Err$ (for hidden layers $\delta_k = y_k(1 - y_k) * w_j * Err$)
  - Err=Expected output-Actual output
- The weight corrections is calculated based on the error function
- The new weights are chosen in such way that the final error in that network is minimized

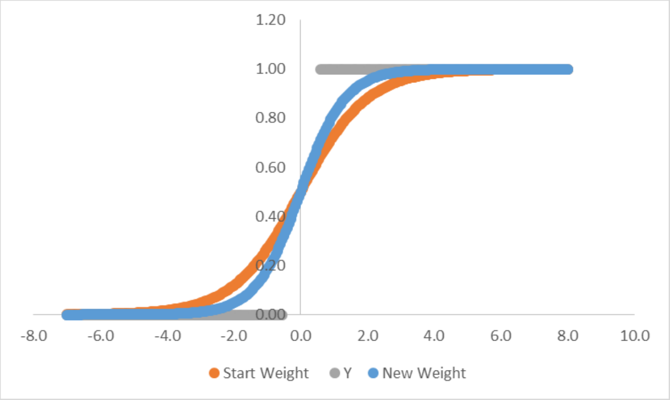
Math-How does the delta rule work?

How does the delta rule work?

- Lets consider a simple example to understand the weight updating using delta rule.





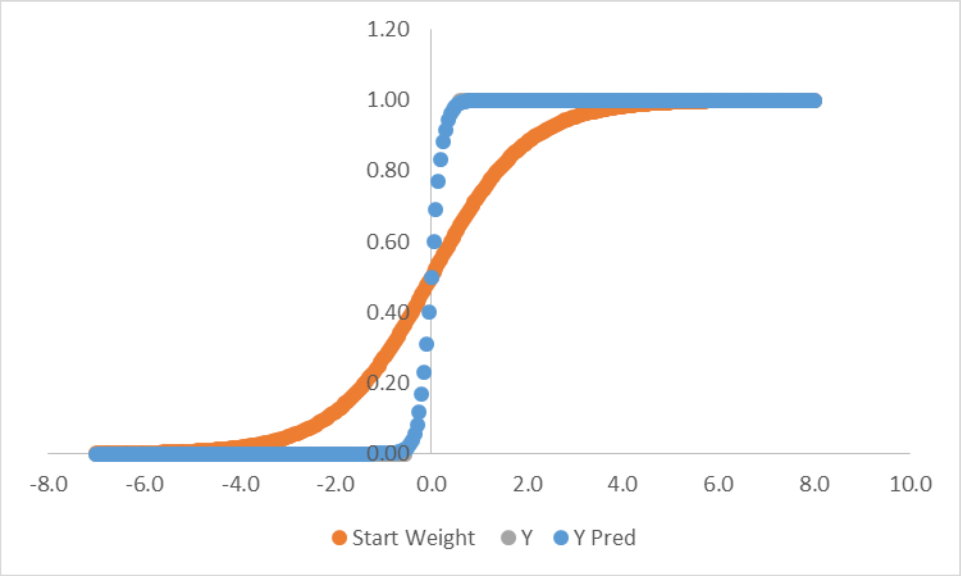


- If we building a simple logistic regression line. We would like to find the weights using weight update rule
- $Y=1/(1+e^{-wx})$ is the equation
- We are searching for the optimal w for our data

- Let w be 1
- $Y=1/(1+e^{-x})$ is the initial equation
- The error in our initial step is 3.59
- To reduce the error we will add a delta to w and make it 1.5

- Now w is 1.5 (blue line)
- $Y=1/(1+e^{-1.5x})$ the updated equation
- With the updated weight, the error is 1.57
- We can further reduce the error by increasing w by delta

How does the delta rule work?



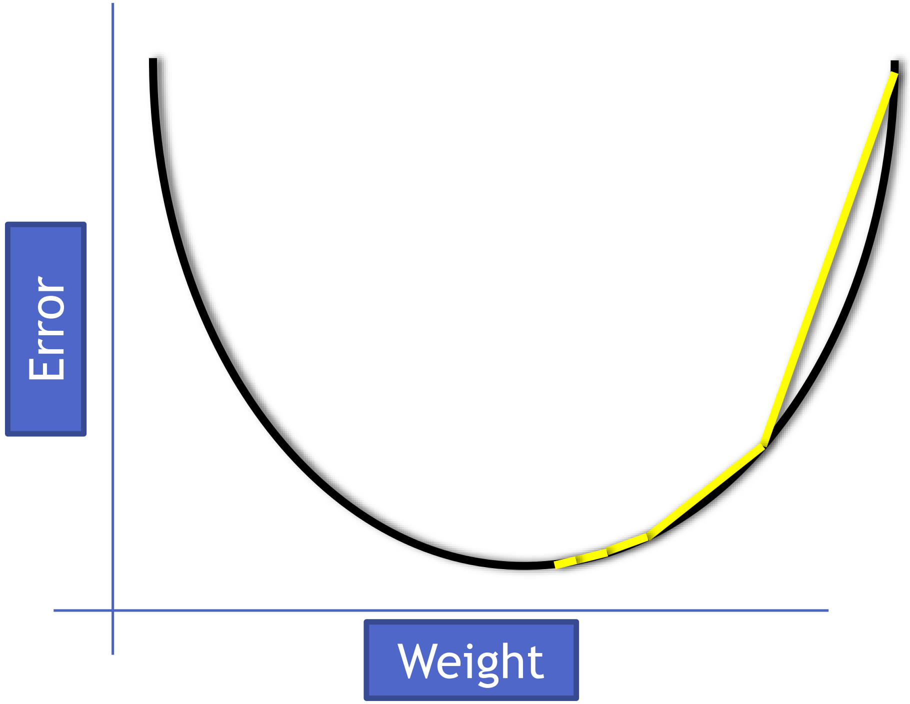
- If we repeat the same process of adding delta and updating weights, we can finally end up with minimum error
- The weight at that final step is the optimal weight
- In this example the weight is 8, and the error is 0
- $Y=1/(1+e^{-8x})$ is the final equation

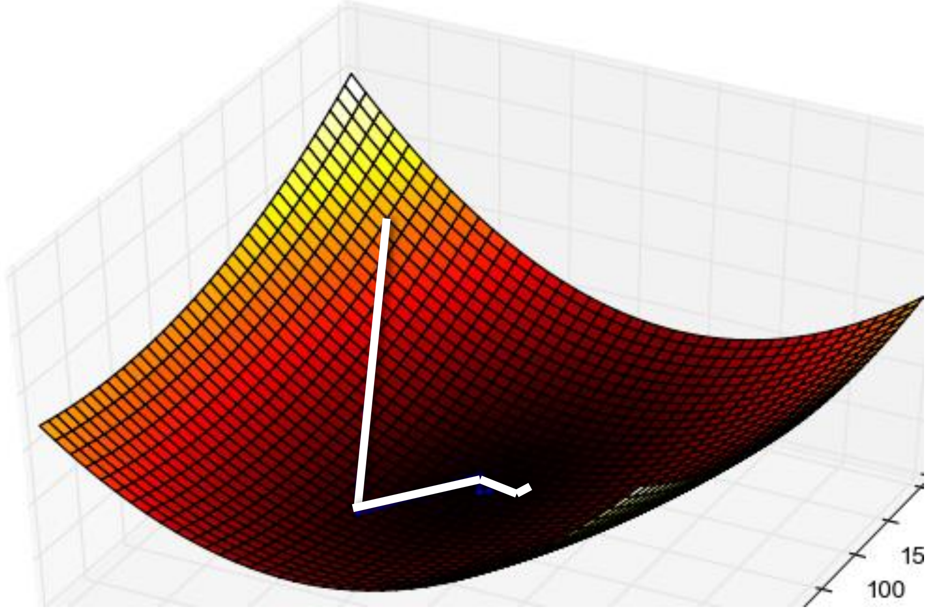
- In this example, we manually changed the weights to reduce the error. This is just for intuition, manual updating is not feasible for complex optimization problems.
- In gradient descent is a scientific optimization method. We update the weights by calculating gradient of the function.

Math-How does gradient descent work?

How does gradient descent work?

- Gradient descent is one of the famous ways to calculate the local minimum
- By Changing the weights we are moving towards the minimum value of the error function. The weights are changed by taking steps in the negative direction of the function gradient(derivative).

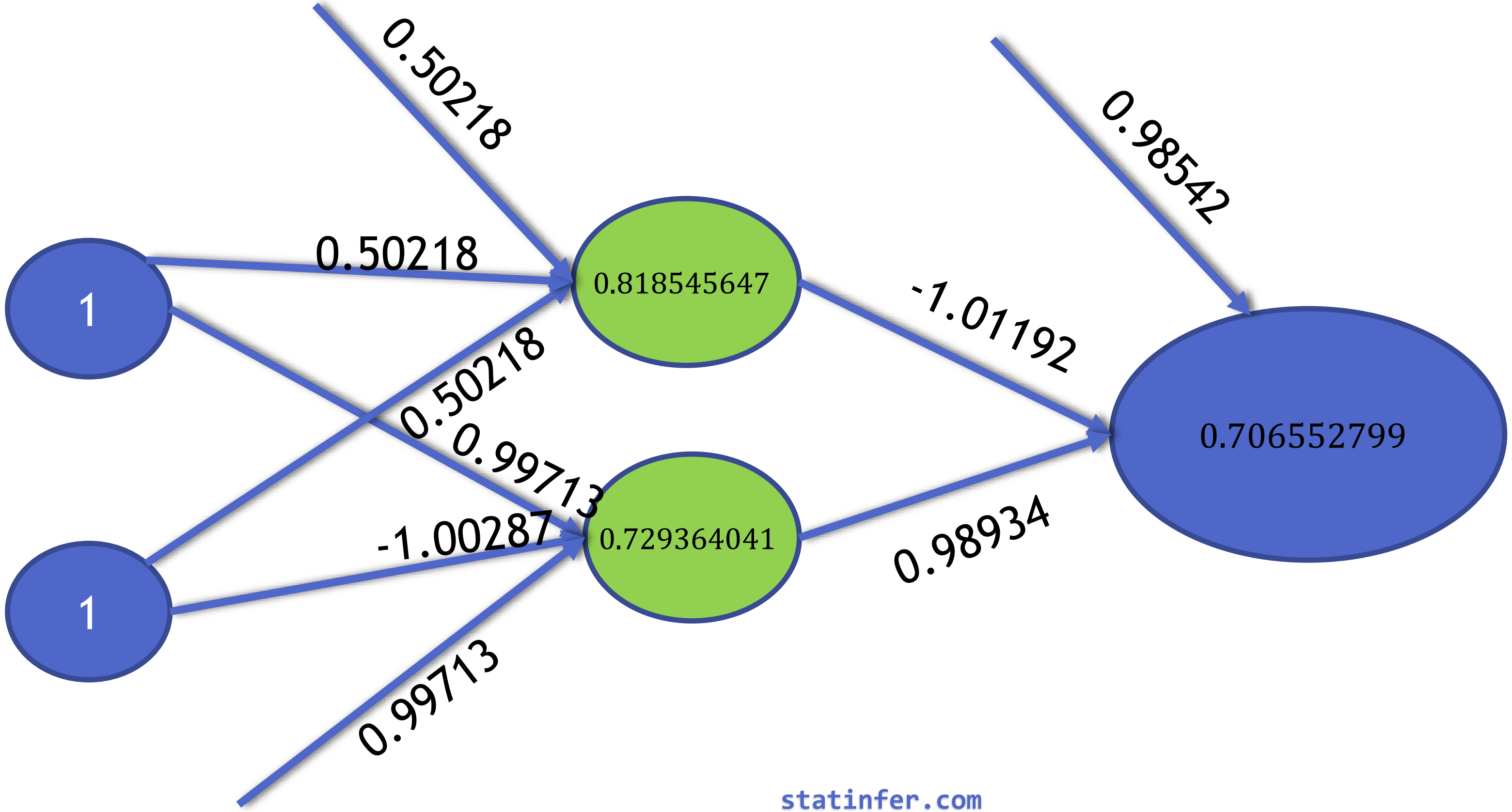




Demo-How does gradient descent work?

Does this method really work?

- We changed the weights did it reduce the overall error?
- Lets calculate the error with new weights and see the change



Gradient Descent method validation

- With our initial set of weights the overall error was 0.7137,Y Actual is 0, Y Predicted is 0.7137 error =0.7137
- The new weights give us a predicted value of 0.70655
- In one iteration, we reduced the error from 0.7137 to 0.70655
- The error is reduced by 1%. Repeat the same process with multiple epochs and training examples, we can reduce the error further.

	input1	input2	Output(Y-Actual)	Y Predicted	Error
Old Weights	1	1	0	0.71371259	0.71371259
Updated Weights	1	1	0	0.706552799	0.706552799

Thank you

Statinfer.com

Download the course videos and handouts from the below link
https://statinfer.com/course/machine-learning-with-r-2/curriculum/?c=b433a9be3189

Session 5 - Neural Network

Handout – Neural Networks		00:00:00
5.1 Introduction and LogReg Recap	FREE	00:00:00
5.2 Decision Boundary	FREE	00:00:00
5.3 Non Linear Decision Boundary NN		00:00:00
5.4 Non Linear Decision Boundary and Solution		00:00:00
5.5 Neural Network Intution		00:00:00
5.6 Neural Networks Algorithm		00:00:00
5.7 Neural Network Algorithm Demo		00:00:00
5.8 Building a Neural Network		00:00:00
5.9 Local vs Global Min		00:00:00