statinfer

Artificial Neural Networks(ANN)

Venkat Reddy

Artificial Neural Networks(ANN)

Contents

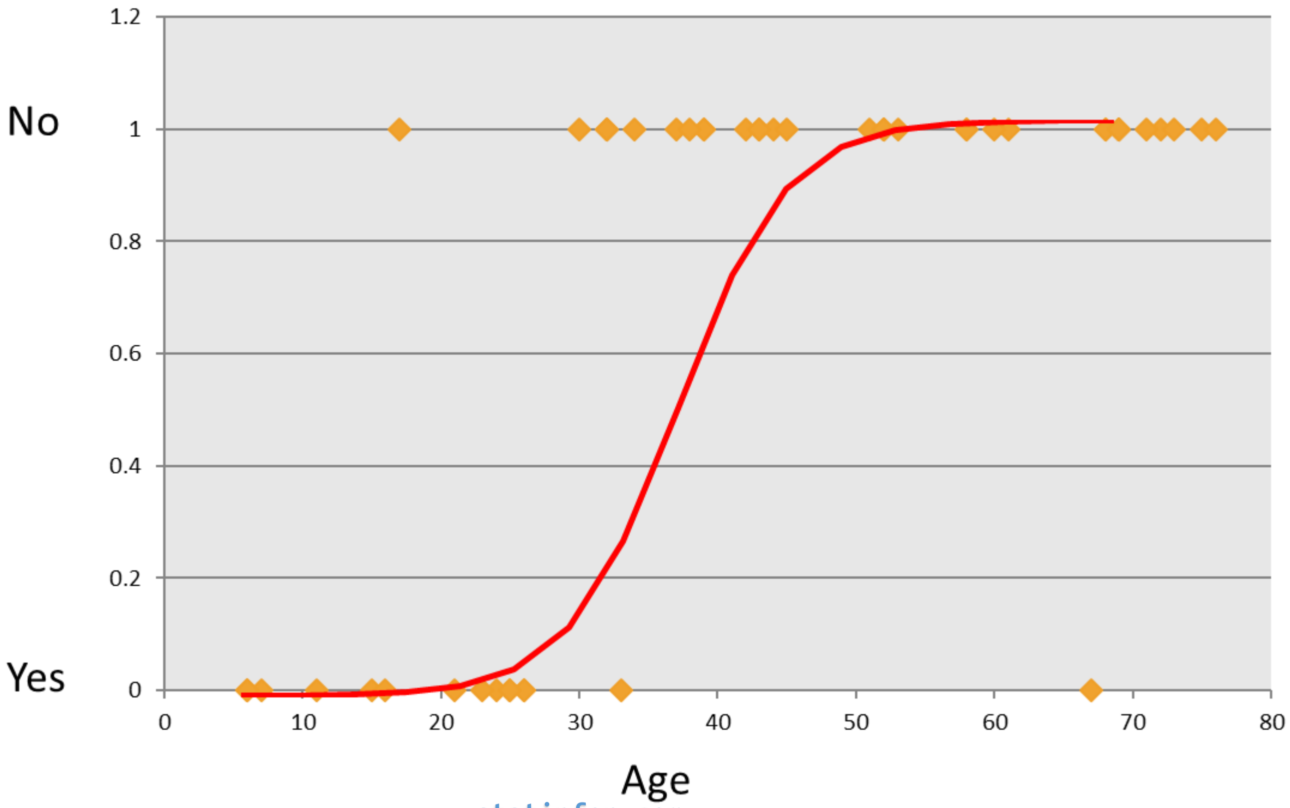
Contents

- 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

Recap of Logistic Regression

Recap of Logistic Regression

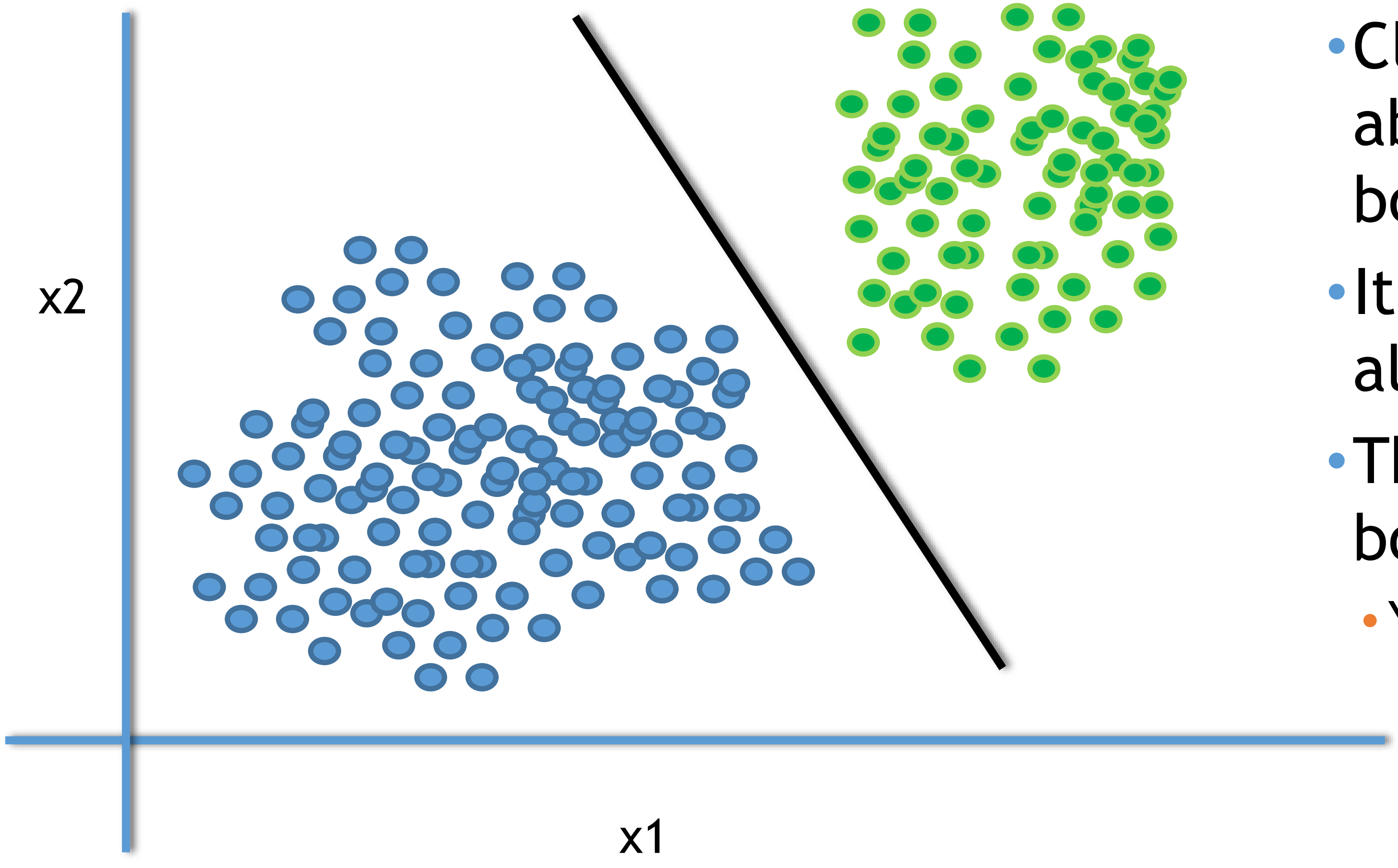
- Categorical output YES/NO type
- Using the predictor variables to predict the categorical output



statinfer

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 $w^T x+w_0>0$; 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
    - $y=e^{(b1x1+b2x2)}/(1+e^{(b1x1+b2x2)})$
    - $\log(y/1-y)=b_0+b_1x_1+b_2x_2$
    - $\text{Log}(0.5/0.5)=b_0+b_1x_1+b_2x_2$
    - $0=b_0+b_1x_1+b_2x_2$
    - $b_0+b_1x_1+b_2x_2=0$ is the line

Decision Boundary – Logistic Regression

- Rewriting it in y=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)

LAB: Decision Boundary

LAB: Logistic Regression

- Dataset: Emp_Purchase/Emp_Purchase.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 purchases using age and experience
- Create the confusion matrix
- Calculate the accuracy and error rates

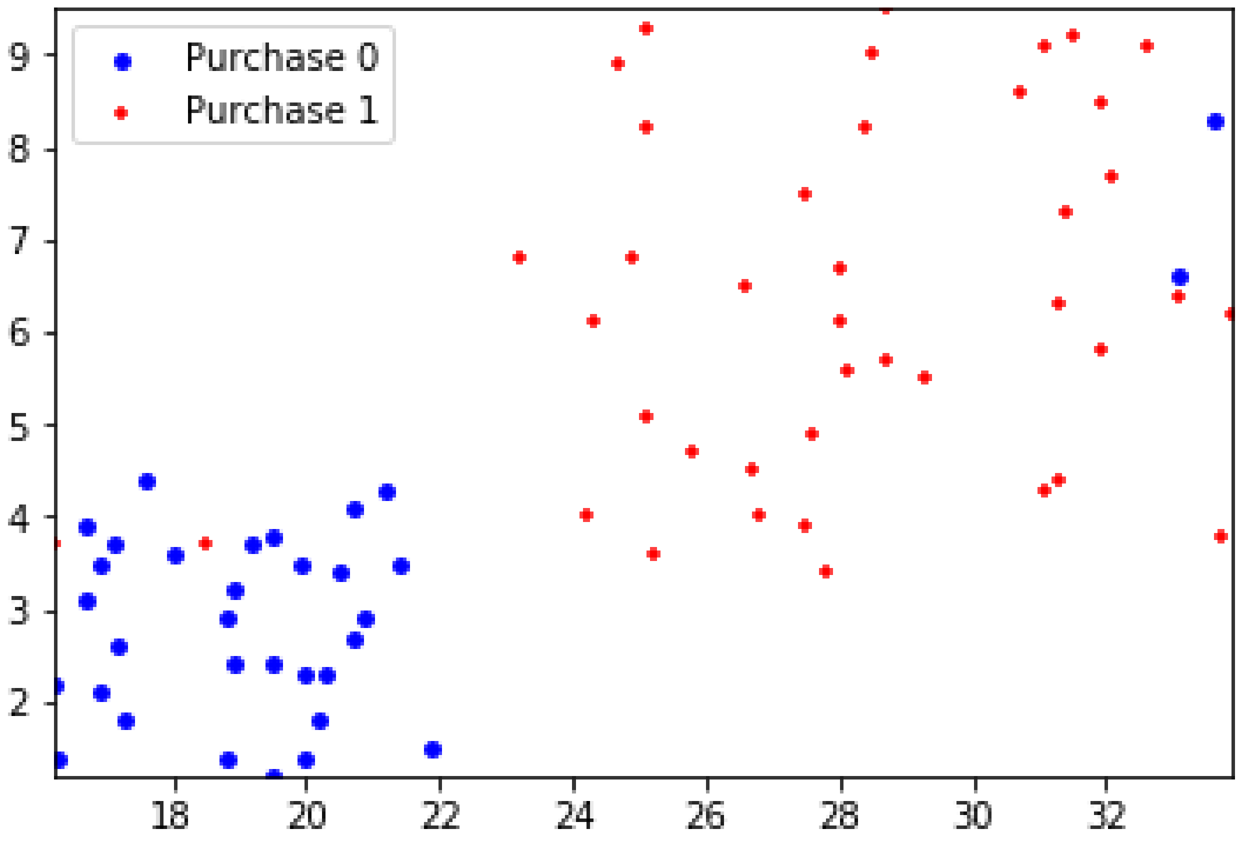
LAB: Decision Boundary

- 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 purchases using age and experience
- Finally draw the decision boundary for this logistic regression model

Code: Logistic Regression

```
import pandas as pd
Emp_Purchase_raw = pd.read_csv("D:\\Google Drive\\Training\\Datasets\\Emp_Purchase\\Emp_Purchase.csv")
Emp_Purchase_raw.shape
Emp_Purchase_raw.columns.values
Emp_Purchase_raw.head(10)

####Filter the data and take a subset from above dataset . Filter condition is Sample_Set<3
Emp_Purchase1=Emp_Purchase_raw[Emp_Purchase_raw.Sample_Set<3]
Emp_Purchase1.shape
Emp_Purchase1.columns.values
Emp_Purchase1.head(10)

#frequency table of Purchase variable
Emp_Purchase1.Purchase.value_counts()

####The clasification graph
#Draw a scatter plot that shows Age on X axis and Experience on Y-axis. Try to distinguish the two classes with colors or sho
import matplotlib.pyplot as plt

fig = plt.figure()
ax1 = fig.add_subplot(111)

ax1.scatter(Emp_Purchase1.Age[Emp_Purchase1.Purchase==0],Emp_Purchase1.Experience[Emp_Purchase1.Purchase==0], s=15, c='b', ma
ax1.scatter(Emp_Purchase1.Age[Emp_Purchase1.Purchase==1],Emp_Purchase1.Experience[Emp_Purchase1.Purchase==1], s=15, c='r', ma

plt.xlim(min(Emp_Purchase1.Age), max(Emp_Purchase1.Age))
plt.ylim(min(Emp_Purchase1.Experience), max(Emp_Purchase1.Experience))
plt.legend(loc='upper left');

plt.show()
```

Code: Logistic Regression



Code: Logistic Regression

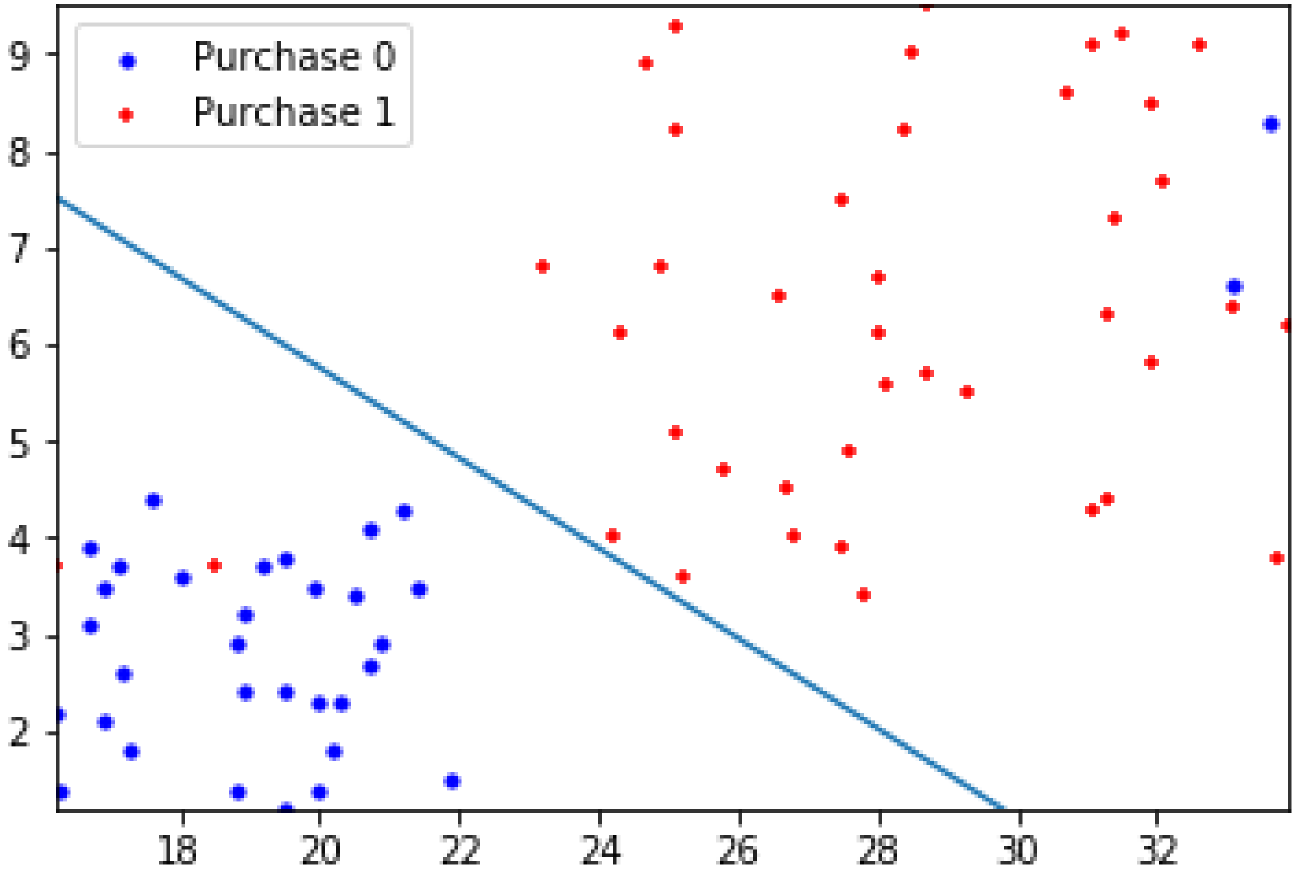
```
import statsmodels.formula.api as sm
model1 = sm.logit(formula='Purchase ~ Age+Experience', data=Emp_Purchase1)
fitted1 = model1.fit()
fitted1.summary2()

#######Accuracy and error of the model1
#Create the confusion matrix
predicted_values=fitted1.predict(Emp_Purchase1[["Age"]+["Experience"]])
predicted_values[1:10]
threshold=0.5

import numpy as np
predicted_class=np.zeros(predicted_values.shape)
predicted_class[predicted_values>threshold]=1

predicted_class

from sklearn.metrics import confusion_matrix as cm
ConfusionMatrix = cm(Emp_Purchase1[['Purchase']],predicted_class)
print(ConfusionMatrix)
accuracy=(ConfusionMatrix[0,0]+ConfusionMatrix[1,1])/sum(sum(ConfusionMatrix))
print('Accuracy : ',accuracy)
error=1-accuracy
print('Error: ',error)
```

```
[[31  2]
 [ 2 39]]
Accuracy :  0.945945945946
Error:  0.0540540540541
```

Code: Logistic Regression

```
slope1=fitted1.params[1]/(-fitted1.params[2])
intercept1=fitted1.params[0]/(-fitted1.params[2])


#Finally draw the decision boundary for this logistic regression model

import matplotlib.pyplot as plt

fig = plt.figure()
ax1 = fig.add_subplot(111)

ax1.scatter(Emp_Purchase1.Age[Emp_Purchase1.Purchase==0],Emp_Purchase1.Experience[Emp_Purchase1.Purchase==0], s=10,
arker="o", label='Purchase 0')
ax1.scatter(Emp_Purchase1.Age[Emp_Purchase1.Purchase==1],Emp_Purchase1.Experience[Emp_Purchase1.Purchase==1], s=10,
arker="+", label='Purchase 1')

plt.xlim(min(Emp_Purchase1.Age), max(Emp_Purchase1.Age))
plt.ylim(min(Emp_Purchase1.Experience), max(Emp_Purchase1.Experience))
plt.legend(loc='upper left');

x_min, x_max = ax1.get_xlim()
ax1.plot([0, x_max], [intercept1, x_max*slope1+intercept1])
plt.show()
```

Code: Logistic Regression



4 Sub Theories

1. Every logistic regression line gives us a decision boundary. It looks like a straight line between class-0 and class-1

statinfer

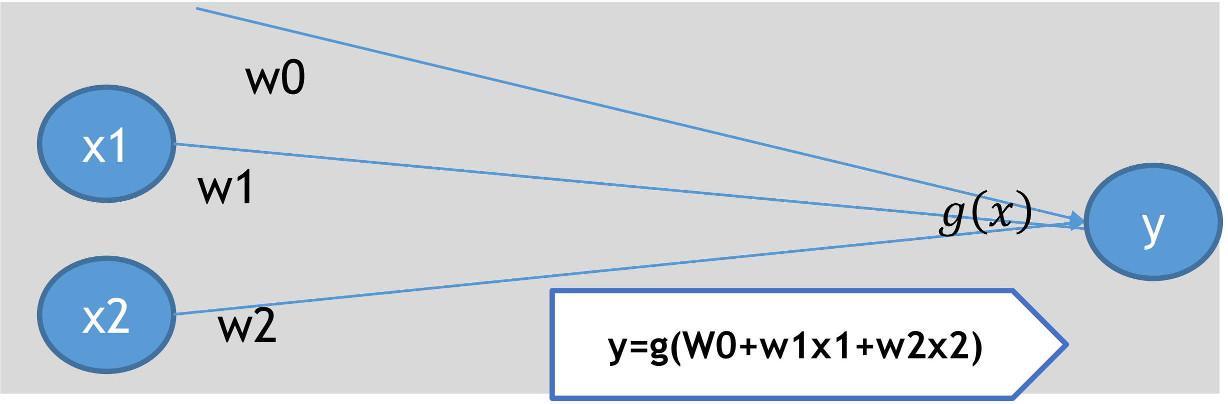
New representation for logistic regression

New representation for logistic regression

$$y = \frac{e^{(\beta_0+\beta_1 x_1+\beta_2 x_2)}}{1+e^{(\beta_0+\beta_1 x_1+\beta_2 x_2)}}$$

$$y = \frac{e^{(w_0+w_1 x_1+w_2 x_2)}}{1+e^{(w_0+w_1 x_1+w_2 x_2)}}$$

$$y = g(w_0 + w_1 x_1 + w_2 x_2) \quad Where\ g(x) = \frac{e^x}{1+e^x}$$



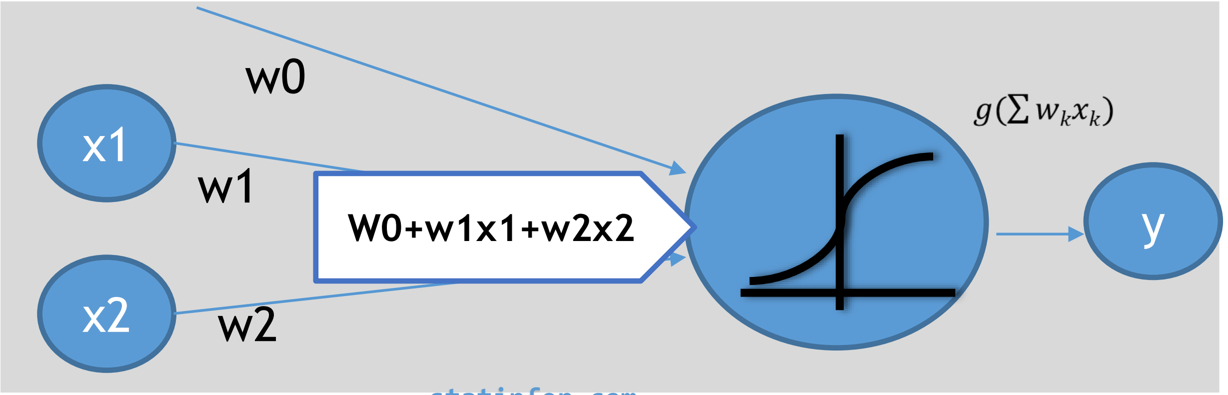
New representation for logistic regression

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

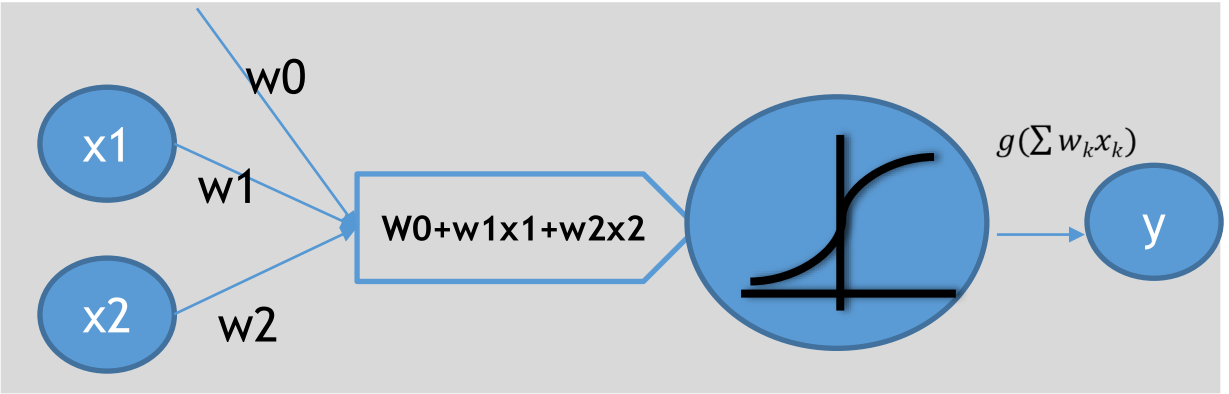
$$y = \frac{1}{1+e^{-(\beta 0+\beta 1x1+\beta 2x2)}}$$

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

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



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

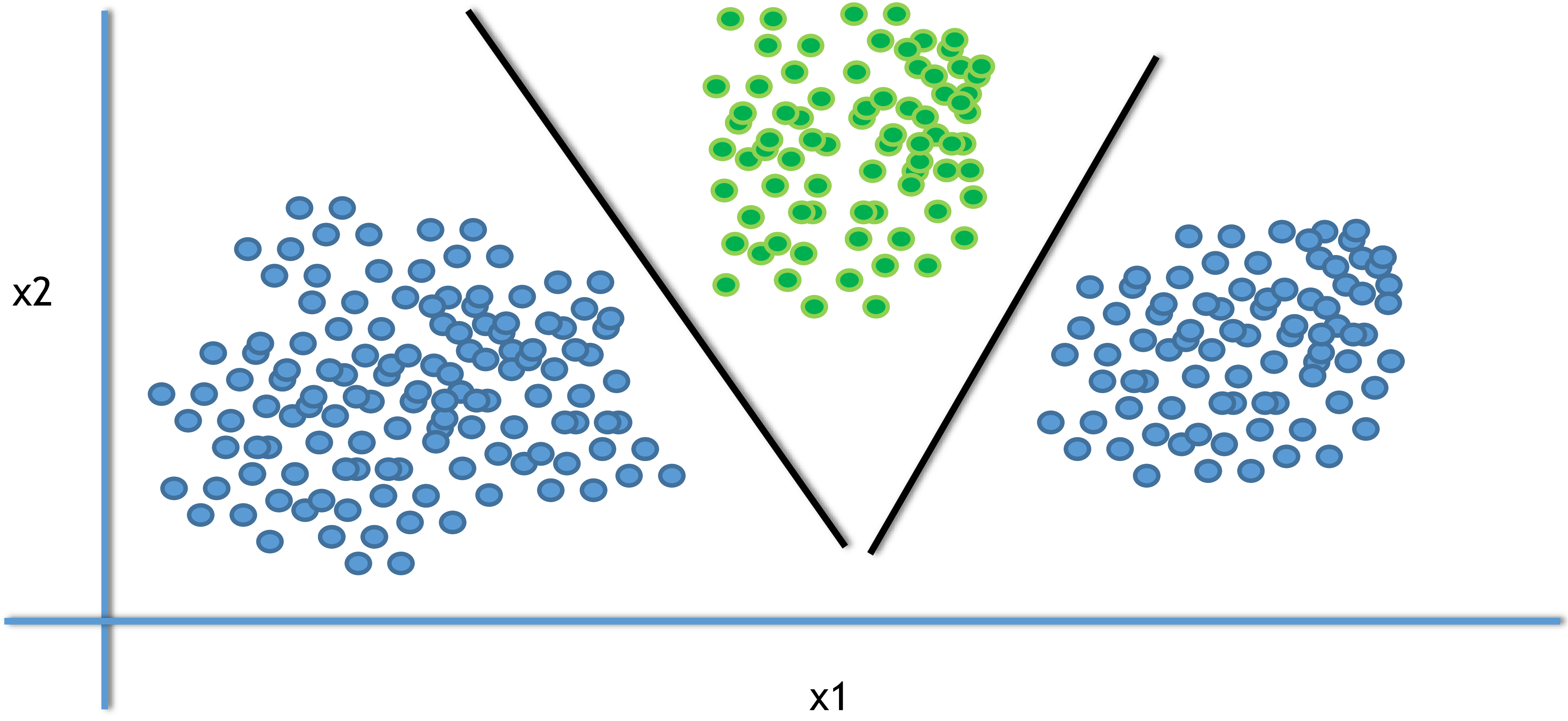
$$We\ find\ \ w\ to\ minimize\ \sum_{i=1}^{n}[y_i - g(\sum w_k x_k)]^2$$

4 Sub Theories

1. Every logistic regression line gives us a decision boundary. It looks like a straight line between class-0 and class-1
2. We are trying to find w’s by minimizing error. While building any model

Multiple / Non-Linear Decision Boundaries-Issue

Multiple /Non-Linear Decision Boundaries



Multiple /Non-Linear Decision Boundaries-issues

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

LAB: Non-Linear Decision Boundaries

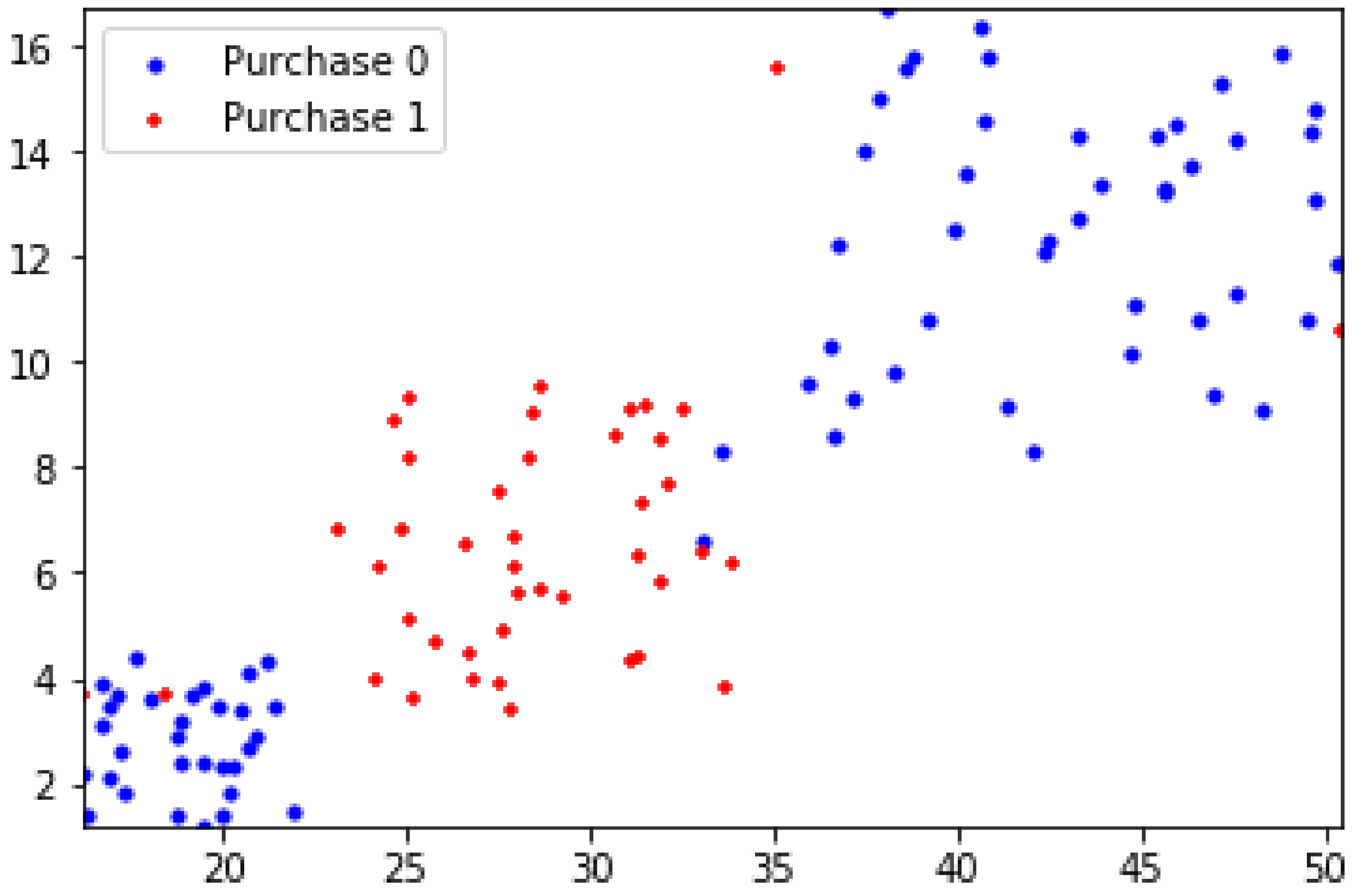
LAB: Non-Linear Decision Boundaries

- Dataset: “Emp_Purchase/ Emp_Purchase.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 Purchases using age and experience
- Finally draw the decision boundary for this logistic regression model
- Create the confusion matrix
- Calculate the accuracy and error rates

LAB: Non-Linear Decision Boundaries

```
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111)

ax.scatter(Emp_Purchase_raw.Age[Emp_Purchase_raw.Purchase==0],Emp_Purchase_raw.Experience[Emp_Purcha
se_raw.Purchase==0], s=10, c='b', marker="o", label='Purchase 0')
ax.scatter(Emp_Purchase_raw.Age[Emp_Purchase_raw.Purchase==1],Emp_Purchase_raw.Experience[Emp_Purcha
se_raw.Purchase==1], s=10, c='r', marker="+", label='Purchase 1')

plt.xlim(min(Emp_Purchase_raw.Age), max(Emp_Purchase_raw.Age))
plt.ylim(min(Emp_Purchase_raw.Experience), max(Emp_Purchase_raw.Experience))
plt.legend(loc='upper left');
plt.show()
```

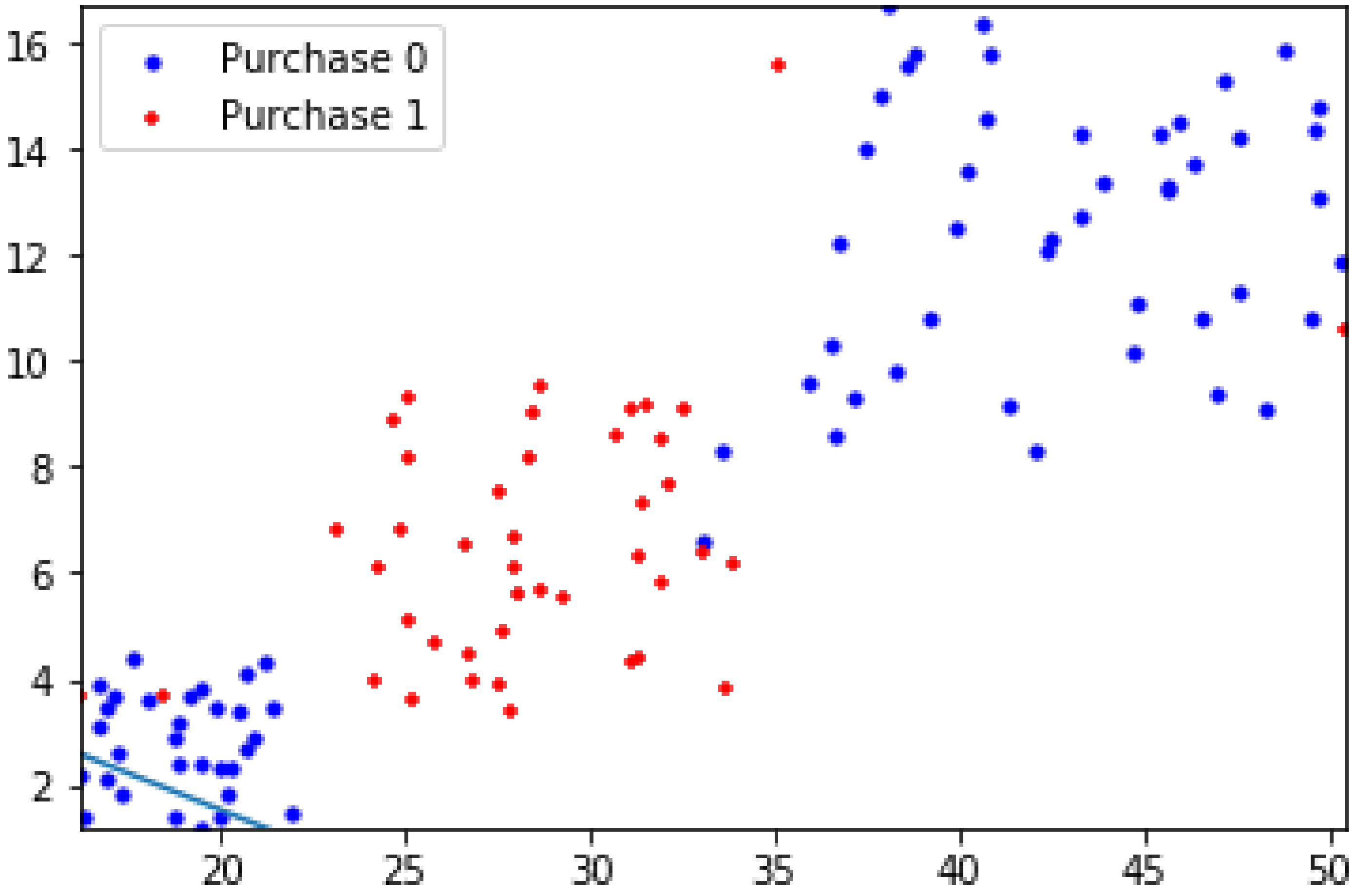
LAB: Non-Linear Decision Boundaries



Code: Non-Linear Decision Boundaries

```
###Logistic Regression model1
import statsmodels.formula.api as sm
model = sm.logit(formula='Purchase ~ Age+Experience', data=Emp_Purchase_raw)
fitted = model.fit()
fitted.summary2()

# getting slope and intercept of the line
slope=fitted.params[1]/(-fitted.params[2])
intercept=fitted.params[0]/(-fitted.params[2])

#Finally draw the decision boundary for this logistic regression model
fig = plt.figure()
ax = fig.add_subplot(111)

ax.scatter(Emp_Purchase_raw.Age[Emp_Purchase_raw.Purchase==0],Emp_Purchase_raw.Experience[Emp_Purchase_raw.Purchase==0], s=10, 
c='b', marker="o", label='Purchase 0')
ax.scatter(Emp_Purchase_raw.Age[Emp_Purchase_raw.Purchase==1],Emp_Purchase_raw.Experience[Emp_Purchase_raw.Purchase==1], s=10, 
c='r', marker="+", label='Purchase 1')
plt.xlim(min(Emp_Purchase_raw.Age), max(Emp_Purchase_raw.Age))
plt.ylim(min(Emp_Purchase_raw.Experience), max(Emp_Purchase_raw.Experience))
plt.legend(loc='upper left');

x_min, x_max = ax.get_xlim()
ax.plot([0, x_max], [intercept, x_max*slope+intercept])
plt.show()
```

Code: Non-Linear Decision Boundaries



Code: Non-Linear Decision Boundaries

```
#######Accuracy and error of the model1
#Create the confusion matrix
#predicting values
predicted_values=fitted.predict(Emp_Purchase_raw[["Age"]+["Experience"]])
predicted_values[1:10]

#Lets convert them to classes using a threshold
threshold=0.5
threshold

import numpy as np
predicted_class=np.zeros(predicted_values.shape)
predicted_class[predicted_values>threshold]=1

#Predcited Classes
predicted_class[1:10]

from sklearn.metrics import confusion_matrix as cm
ConfusionMatrix = cm(Emp_Purchase_raw[['Purchase']],predicted_class)
print(ConfusionMatrix)
accuracy=(ConfusionMatrix[0,0]+ConfusionMatrix[1,1])/sum(sum(ConfusionMatrix))
print(accuracy)
error=1-accuracy
```

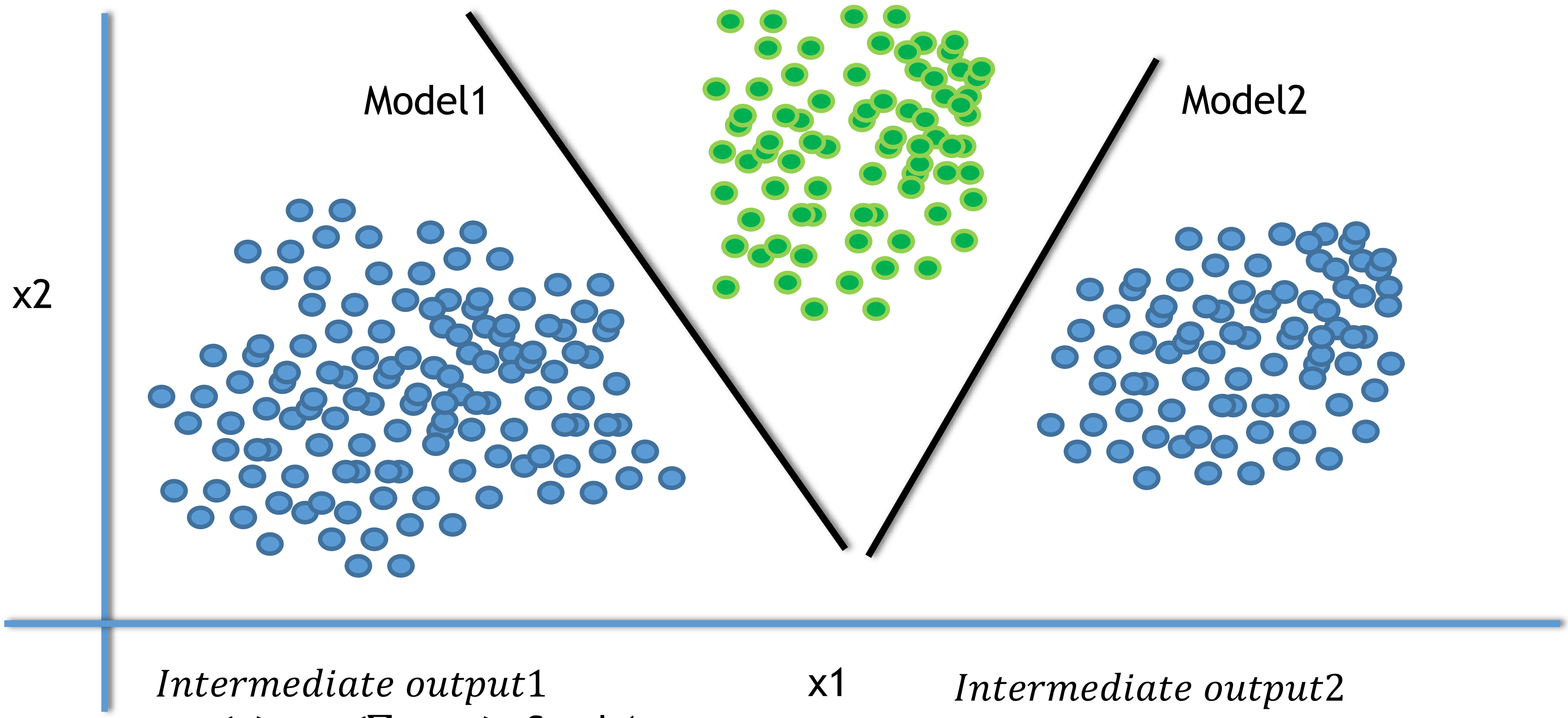
```
[[69  7]
 [43  0]]
0.579831932773
```

4 Sub Theories

1. Every logistic regression line gives us a decision boundary. It looks like a straight line between class-0 and class-1
2. We are trying to find w's by minimizing error. While building any model
3. Logistic Regression Line fails in case of multiple/ non linear decision boundaries

Non-Linear Decision Boundaries-Solution

Intermediate outputs

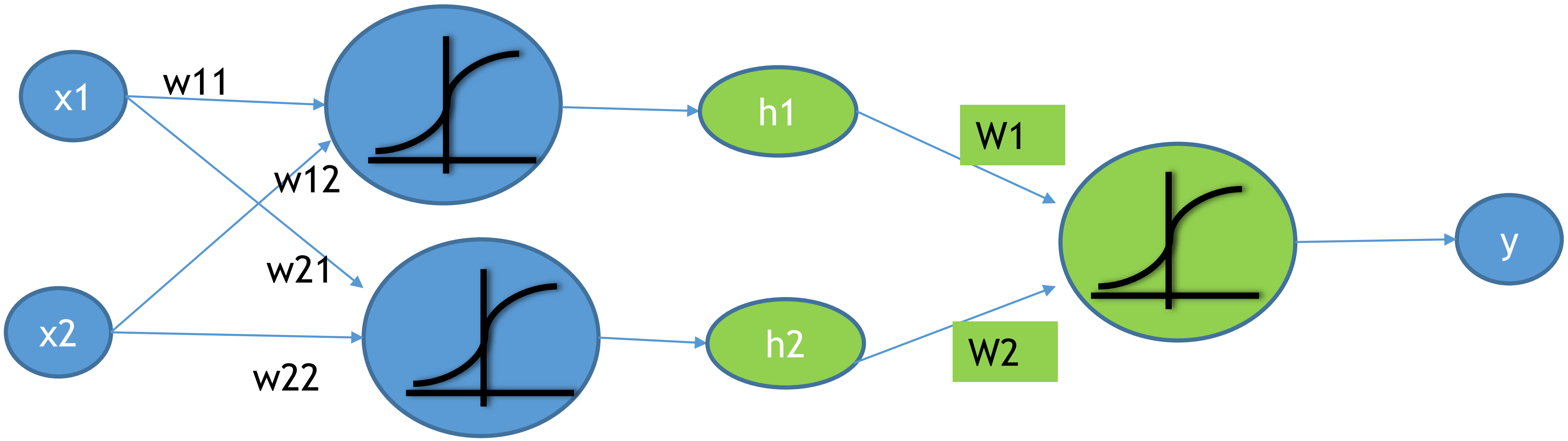


$Intermediate\ output1$
$out(x) = g(\sum w_k x_k)$,Say h1

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

The Intermediate output

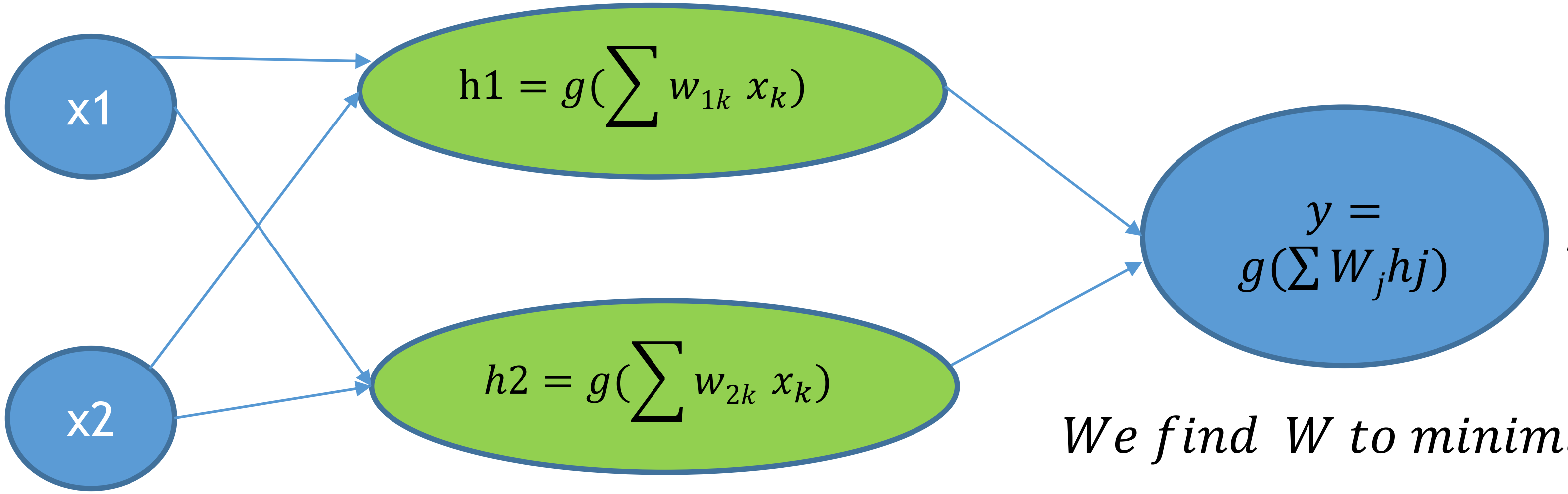
- Using the x’s Directly predicting y is challenging.
- We can predict h, the intermediate output, which will indeed predict Y



Finding the weights for intermediate outputs

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

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



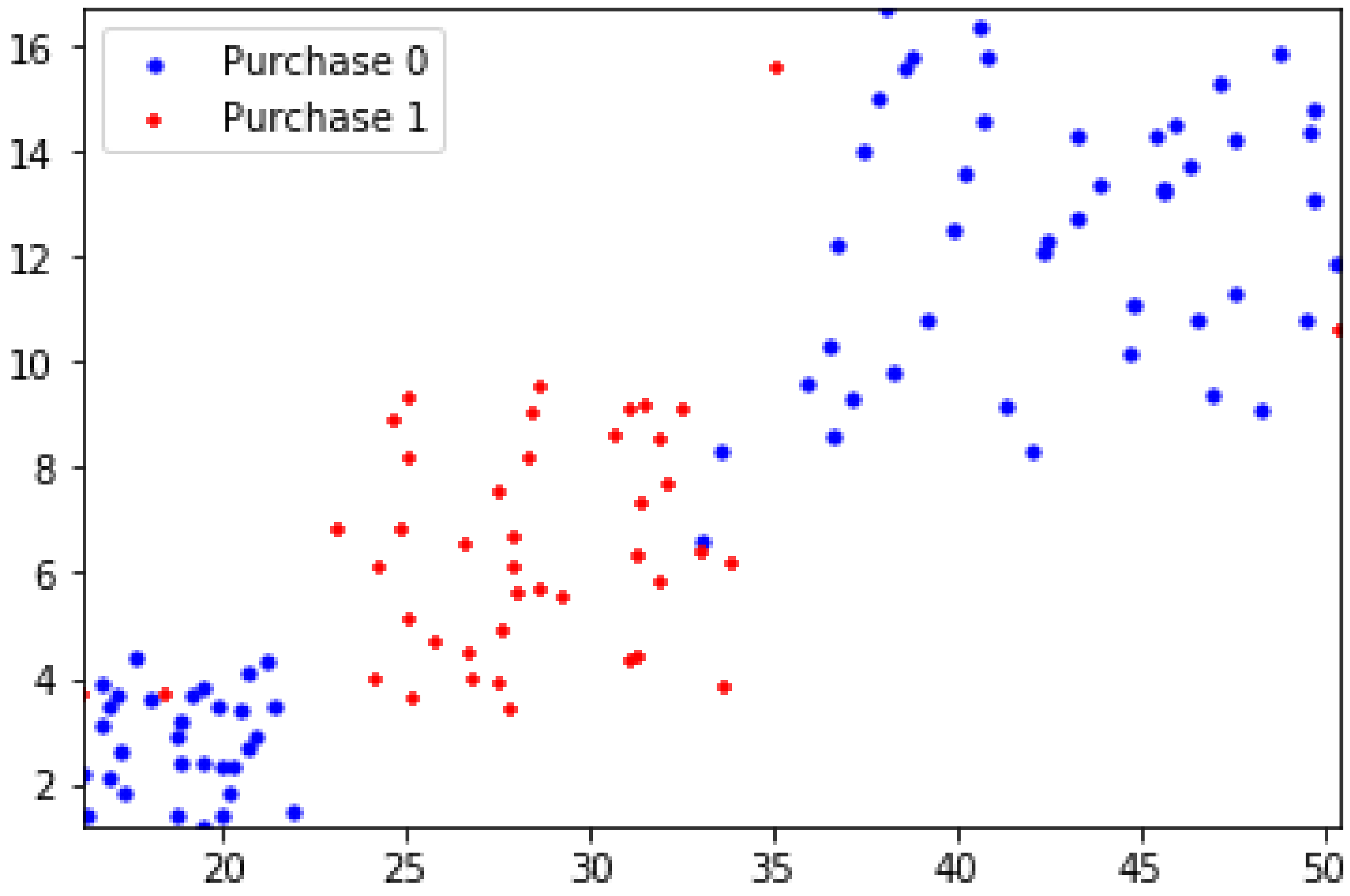
Final output
$y = out(h) = g(\sum W_j hj)$

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

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

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

Non-Linear Decision Boundaries



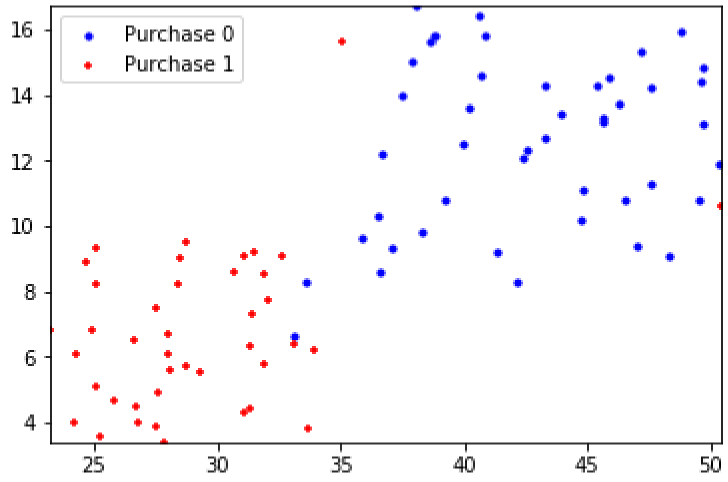
LAB: Intermediate output

LAB: Intermediate output

- Dataset: Emp_Purchase/ Emp_Purchase.csv
- Filter the data and take first 74 observations from above dataset .
- Build a logistic regression model to predict purchases 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 purchases using age and experience
- Calculate the prediction probabilities for all the inputs. Store the probabilities in inter2 variable
- Build a consolidated model to predict purchases 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

```
Emp_Purchase2=Emp_Purchase_raw[Emp_Purchase_raw.Sample_Set>1]
Emp_Purchase2.shape
Emp_Purchase2.columns.values
Emp_Purchase2.head(10)

#frequency table of Purchase variable
Emp_Purchase2.Purchase.value_counts()

####The clasification graph
#Draw a scatter plot that shows Age on X axis and Experience on Y-axis. Tr
with colors or shapes.
import matplotlib.pyplot as plt

fig = plt.figure()
ax2 = fig.add_subplot(111)

ax2.scatter(Emp_Purchase2.Age[Emp_Purchase2.Purchase==0],Emp_Purchase2.Experience[Emp_Purchase2.Purchase==0],
s=10, c='b', marker="o", label='Purchase 0')
ax2.scatter(Emp_Purchase2.Age[Emp_Purchase2.Purchase==1],Emp_Purchase2.Experience[Emp_Purchase2.Purchase==1],
s=10, c='r', marker="+", label='Purchase 1')
plt.xlim(min(Emp_Purchase2.Age), max(Emp_Purchase2.Age))
plt.ylim(min(Emp_Purchase2.Experience), max(Emp_Purchase2.Experience))
plt.legend(loc='upper left');
plt.show()
```



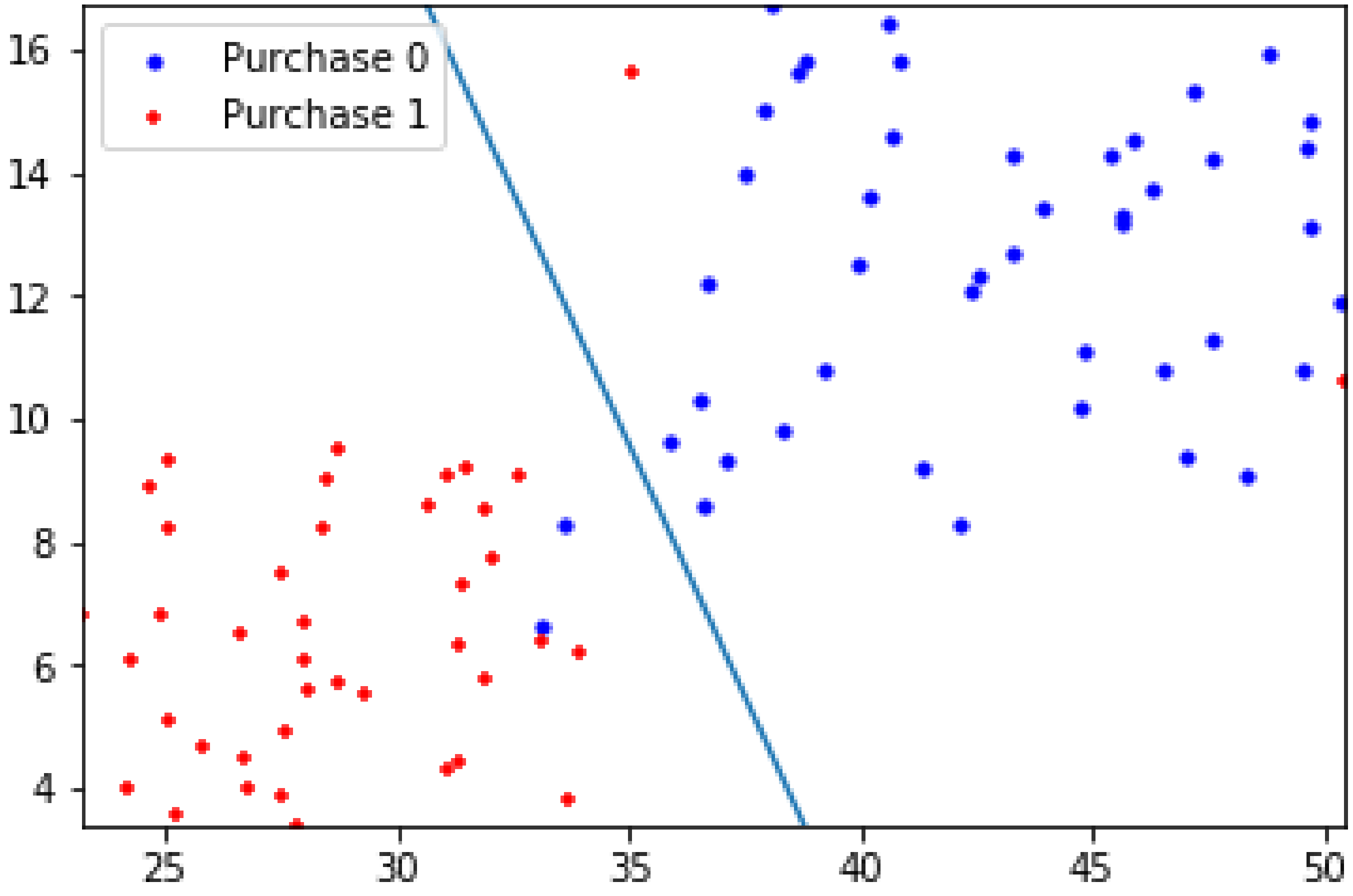
Code: Intermediate output

```
import statsmodels.formula.api as sm
model2 = sm.logit(formula='Purchase ~ Age+Experience', data=Emp_Purchase2)
fitted2 = model2.fit(method="bfgs")
fitted2.summary2()
```

Code: Intermediate output

```
# getting slope and intercept of the line
# getting slope and intercept of the line
slope2=fitted2.params[1]/(-fitted2.params[2])
intercept2=fitted2.params[0]/(-fitted2.params[2])

import matplotlib.pyplot as plt

fig = plt.figure()
ax2 = fig.add_subplot(111)

ax2.scatter(Emp_Purchase2.Age[Emp_Purchase2.Purchase==0],Emp_Purchase2.Experience[Emp_Purchase2.Purchase==0],
s=10, c='b', marker="o", label='Purchase 0')
ax2.scatter(Emp_Purchase2.Age[Emp_Purchase2.Purchase==1],Emp_Purchase2.Experience[Emp_Purchase2.Purchase==1],
s=10, c='r', marker="+", label='Purchase 1')
plt.xlim(min(Emp_Purchase2.Age), max(Emp_Purchase2.Age))
plt.ylim(min(Emp_Purchase2.Experience), max(Emp_Purchase2.Experience))
plt.legend(loc='upper left');

x_min, x_max = ax2.get_xlim()
y_min,y_max=ax2.get_ylim()
ax2.plot([x_min, x_max], [x_min*slope2+intercept2, x_max*slope2+intercept2])
plt.show()
```

Code: Intermediate output



Code: Intermediate output

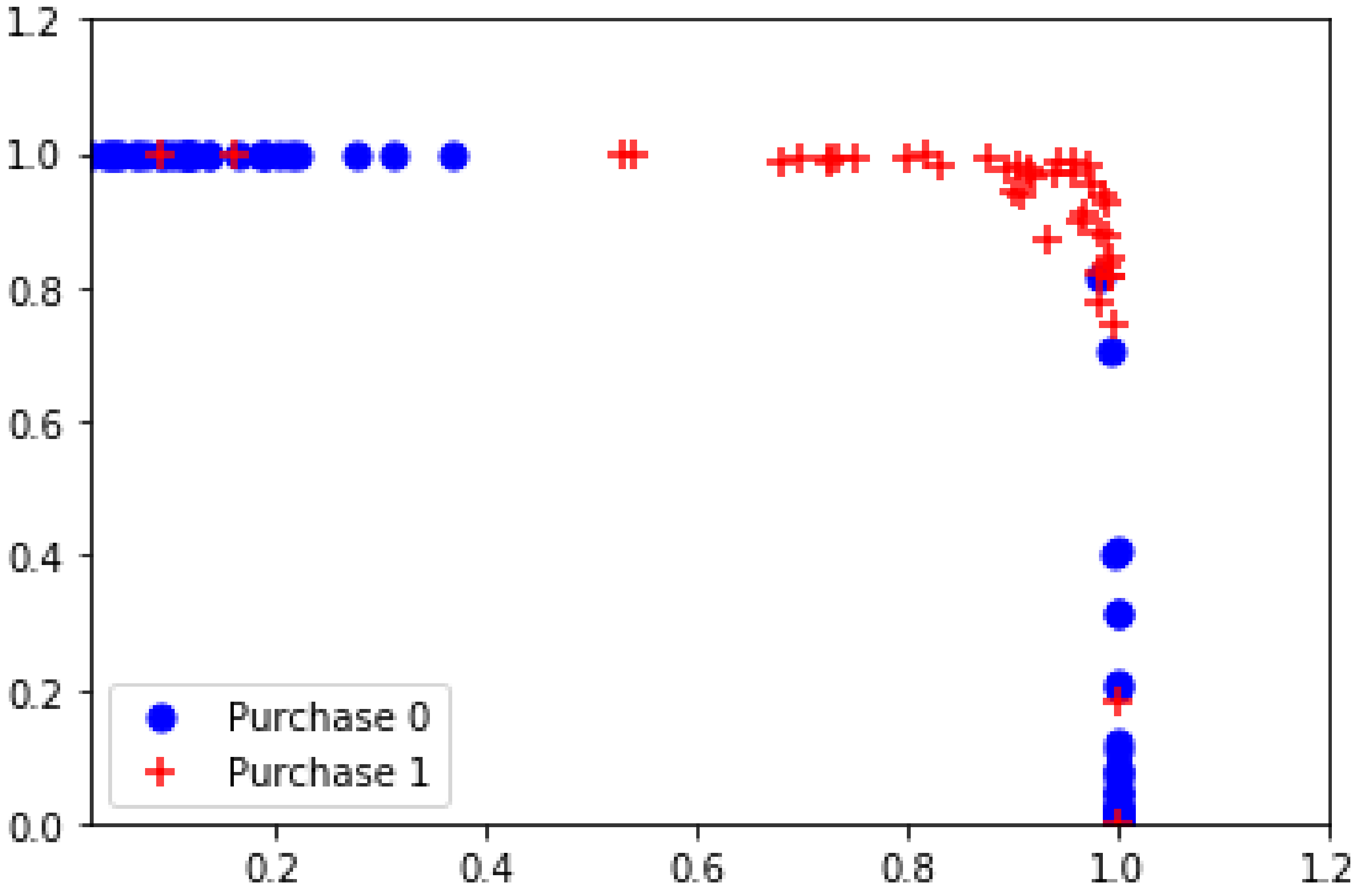
```
#######Accuracy and error of the model1
#Create the confusion matrix
#Predciting Values
predicted_values=fitted2.predict(Emp_Purchase2[["Age"]+["Experience"]])
predicted_values[1:10]

#Lets convert them to classes using a threshold
threshold=0.5
threshold

import numpy as np
predicted_class=np.zeros(predicted_values.shape)
predicted_class[predicted_values>threshold]=1

#Predcited Classes
predicted_class[1:10]

from sklearn.metrics import confusion_matrix as cm
ConfusionMatrix = cm(Emp_Purchase2[['Purchase']],predicted_class)
print(ConfusionMatrix)
accuracy=(ConfusionMatrix[0,0]+ConfusionMatrix[1,1])/sum(sum(ConfusionMatrix))
print(accuracy)

error=1-accuracy
error
```

```
[[43  2]
 [ 2 39]]
0.953488372093
```

Code: Intermediate output

```
fitted1.summary2()
fitted2.summary2()

#The two new coloumns
Emp_Purchase_raw['inter1']=fitted1.predict(Emp_Purchase_raw[["Age"]+["Experience"]])
Emp_Purchase_raw['inter2']=fitted2.predict(Emp_Purchase_raw[["Age"]+["Experience"]])

#plotting the new columns
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111)

ax.scatter(Emp_Purchase_raw.inter1[Emp_Purchase_raw.Purchase==0],Emp_Purchase_raw.inter2[Emp_Purchase_raw.Purchase==0], s=50, c='b', marker="o", label='Purchase 0')
ax.scatter(Emp_Purchase_raw.inter1[Emp_Purchase_raw.Purchase==1],Emp_Purchase_raw.inter2[Emp_Purchase_raw.Purchase==1], s=50, c='r', marker="+", label='Purchase 1')

plt.xlim(min(Emp_Purchase_raw.inter1), max(Emp_Purchase_raw.inter1)+0.2)
plt.ylim(min(Emp_Purchase_raw.inter2), max(Emp_Purchase_raw.inter2)+0.2)

plt.legend(loc='lower left');
plt.show()
```

Code: Intermediate output



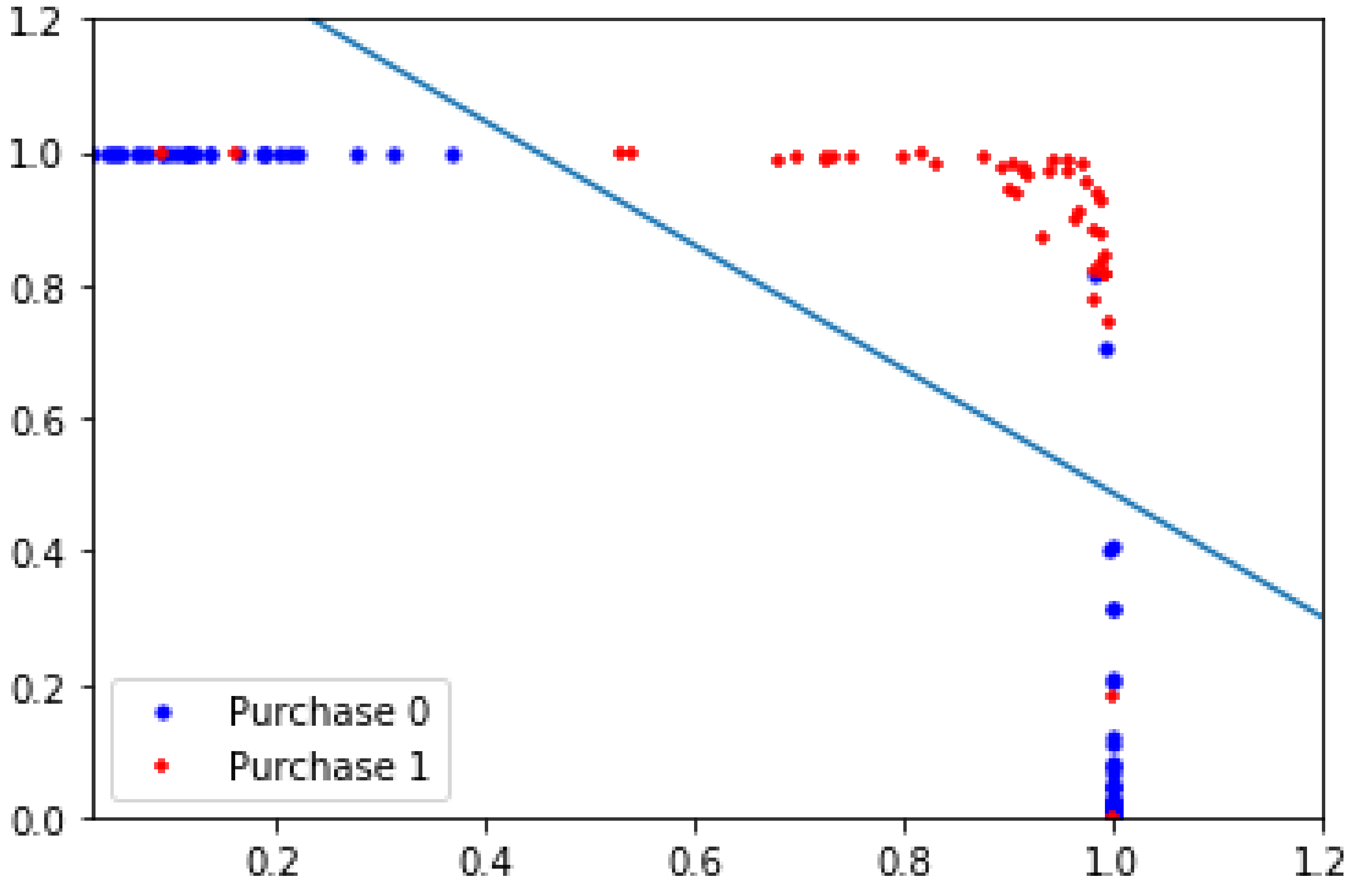
Code: Intermediate output

```
import statsmodels.formula.api as sm

model_combined = sm.logit(formula='Purchase ~ inter1+inter2', data=Emp_Purchase_raw)
fitted_combined = model_combined.fit(method="bfgs")
fitted_combined.summary()

# getting slope and intercept of the line
slope_combined=fitted_combined.params[1]/(-fitted_combined.params[2])
intercept_combined=fitted_combined.params[0]/(-fitted_combined.params[2])
```

Code: Intermediate output

```
#Finally draw the decision boundary for this logistic regression model
import matplotlib.pyplot as plt

fig = plt.figure()
ax2 = fig.add_subplot(111)

ax2.scatter(Emp_Purchase_raw.inter1[Emp_Purchase_raw.Purchase==0],Emp_Purchase_raw.inter2[Emp_Purchase_raw.Purchase==0], s=10, c='b', marker="o", label='Purchase 0')
ax2.scatter(Emp_Purchase_raw.inter1[Emp_Purchase_raw.Purchase==1],Emp_Purchase_raw.inter2[Emp_Purchase_raw.Purchase==1], s=10, c='r', marker="+", label='Purchase 1')

plt.xlim(min(Emp_Purchase_raw.inter1), max(Emp_Purchase_raw.inter1)+0.2)
plt.ylim(min(Emp_Purchase_raw.inter2), max(Emp_Purchase_raw.inter2)+0.2)

plt.legend(loc='lower left');

x_min, x_max = ax2.get_xlim()
y_min,y_max=ax2.get_ylim()
ax2.plot([x_min, x_max], [x_min*slope_combined+intercept_combined,
x_max*slope_combined+intercept_combined])
plt.show()
```

Code: Intermediate output



Code: Intermediate output

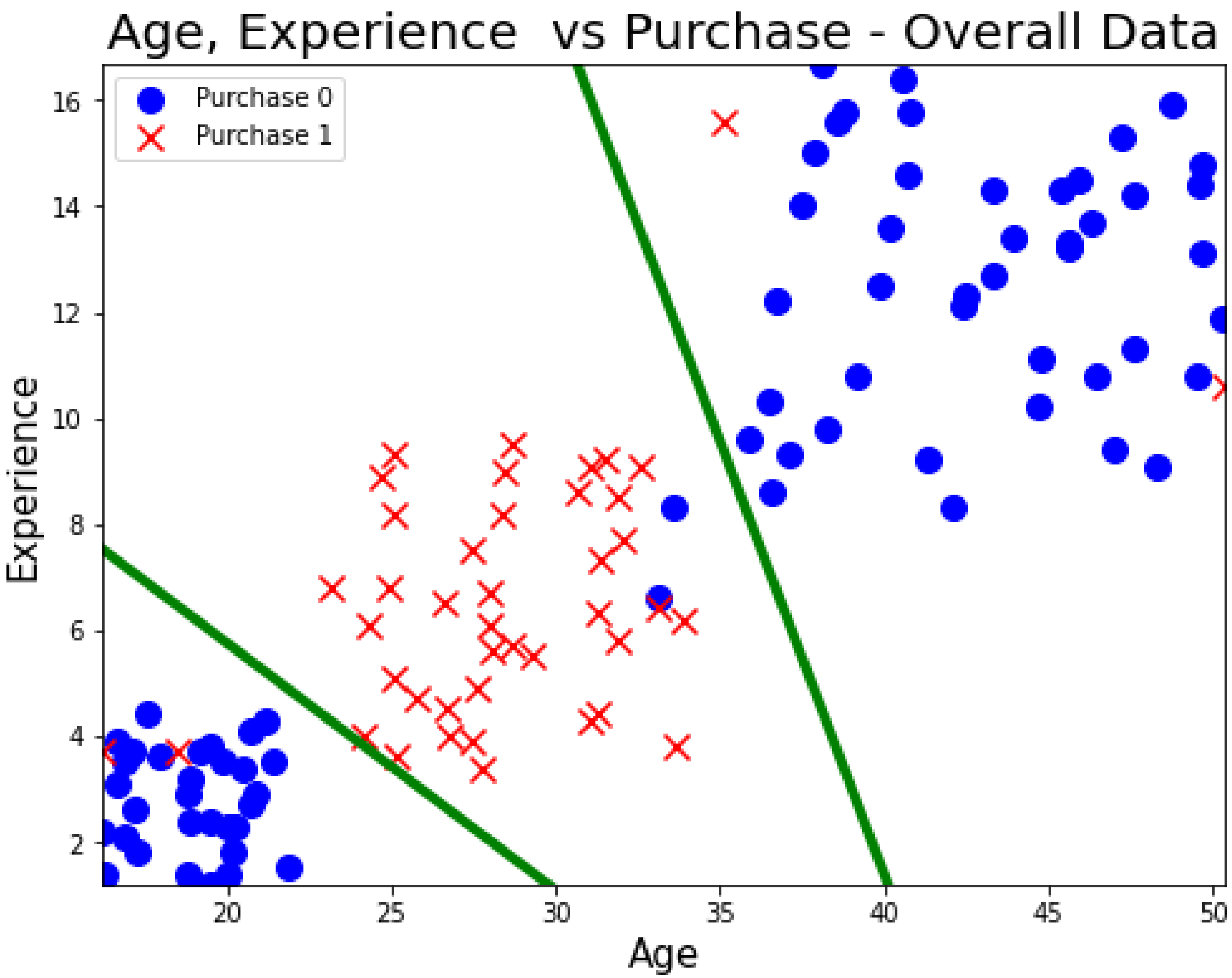
```
#######Accuracy and error of the model1
#Create the confusion matrix
#Predciting Values
predicted_values=fitted_combined.predict(Emp_Purchase_raw[["inter1"]+["inter2"]])
predicted_values[1:10]

#Lets convert them to classes using a threshold
threshold=0.5
threshold

import numpy as np
predicted_class=np.zeros(predicted_values.shape)
predicted_class[predicted_values>threshold]=1

#ConfusionMatrix
from sklearn.metrics import confusion_matrix as cm
ConfusionMatrix = cm(Emp_Purchase_raw[['Purchase']],predicted_class)
print(ConfusionMatrix)
accuracy=(ConfusionMatrix[0,0]+ConfusionMatrix[1,1])/sum(sum(ConfusionMatrix))
print(accuracy)
```

```
[[31  2]
 [ 2 39]]
Accuracy :  0.94594
```

```
[[43  2]
 [ 2 39]]
0.953488372093
```

```
[[74  2]
 [ 4 39]]
0.949579831933
```

Result from Two models



4 Sub Theories

1. Every logistic regression line gives us a decision boundary. It looks like a straight line between class-0 and class-1
2. We are trying to find w’s by minimizing error. While building any model
3. Logistic Regression Line fails in case of non linear decision boundaries
4. We used intermediate outputs to solve the problem of non linear decision boundaries

statinfer

Neural Network intuition

Neural Network intuition

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

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

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

- 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

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

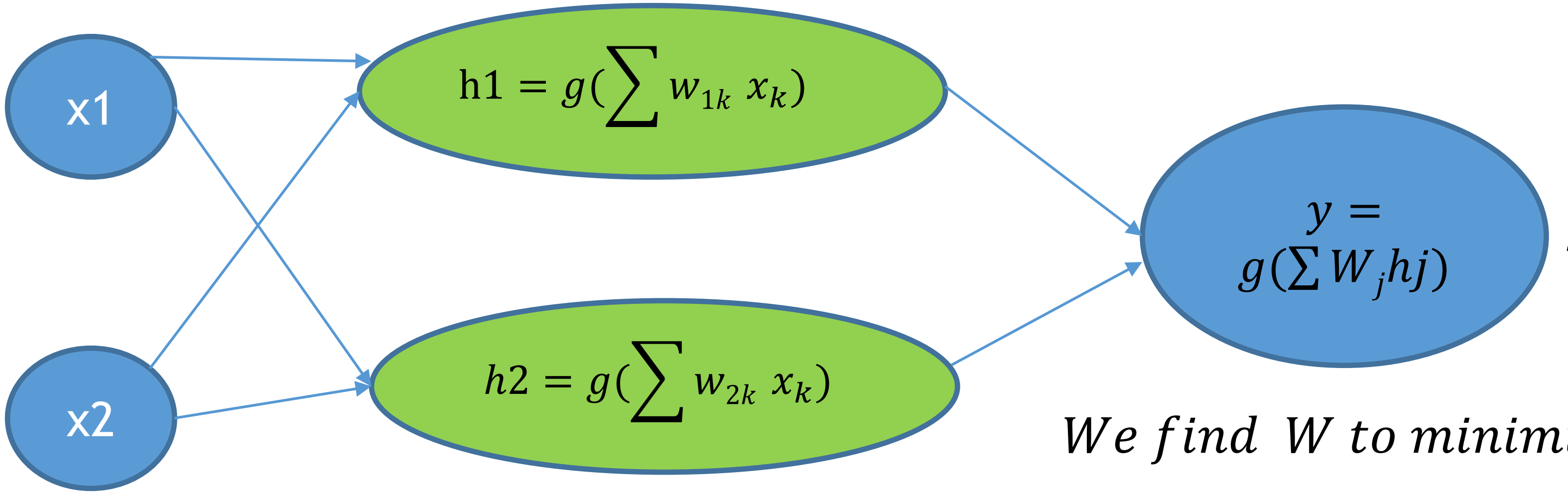
$$We\ find\ \{W_j\}\ \&\ \{w_{jk}\}\ to\ minimize\ \sum_{i=1}^{n}[y_i - g(\sum W_j g(\sum w_{jk} x_k))]^2$$

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

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

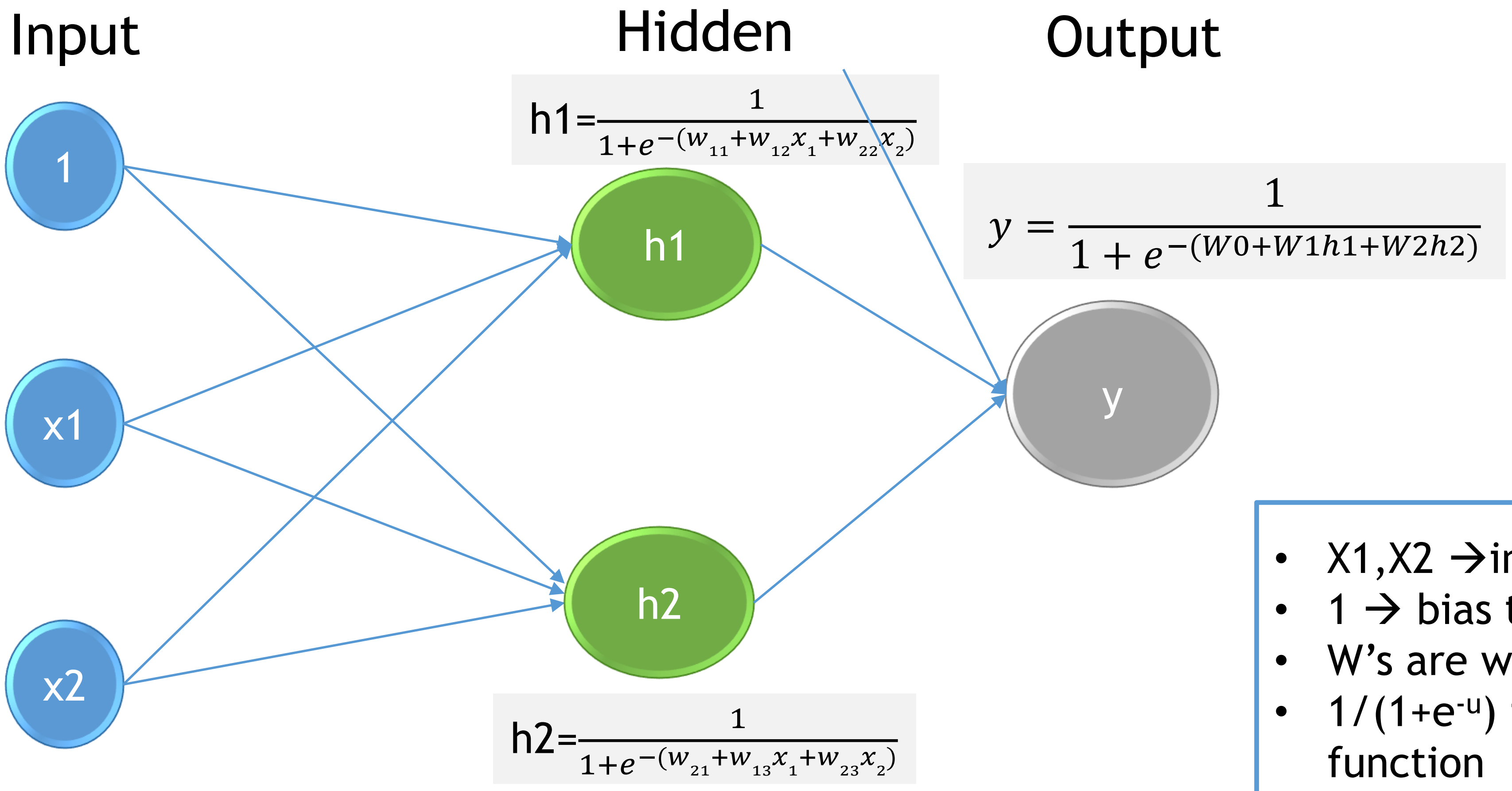
Intermediate output2
$h2 = out(x) = g(\sum w_{2k}\ x_k)$

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

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

Neural network and vocabulary



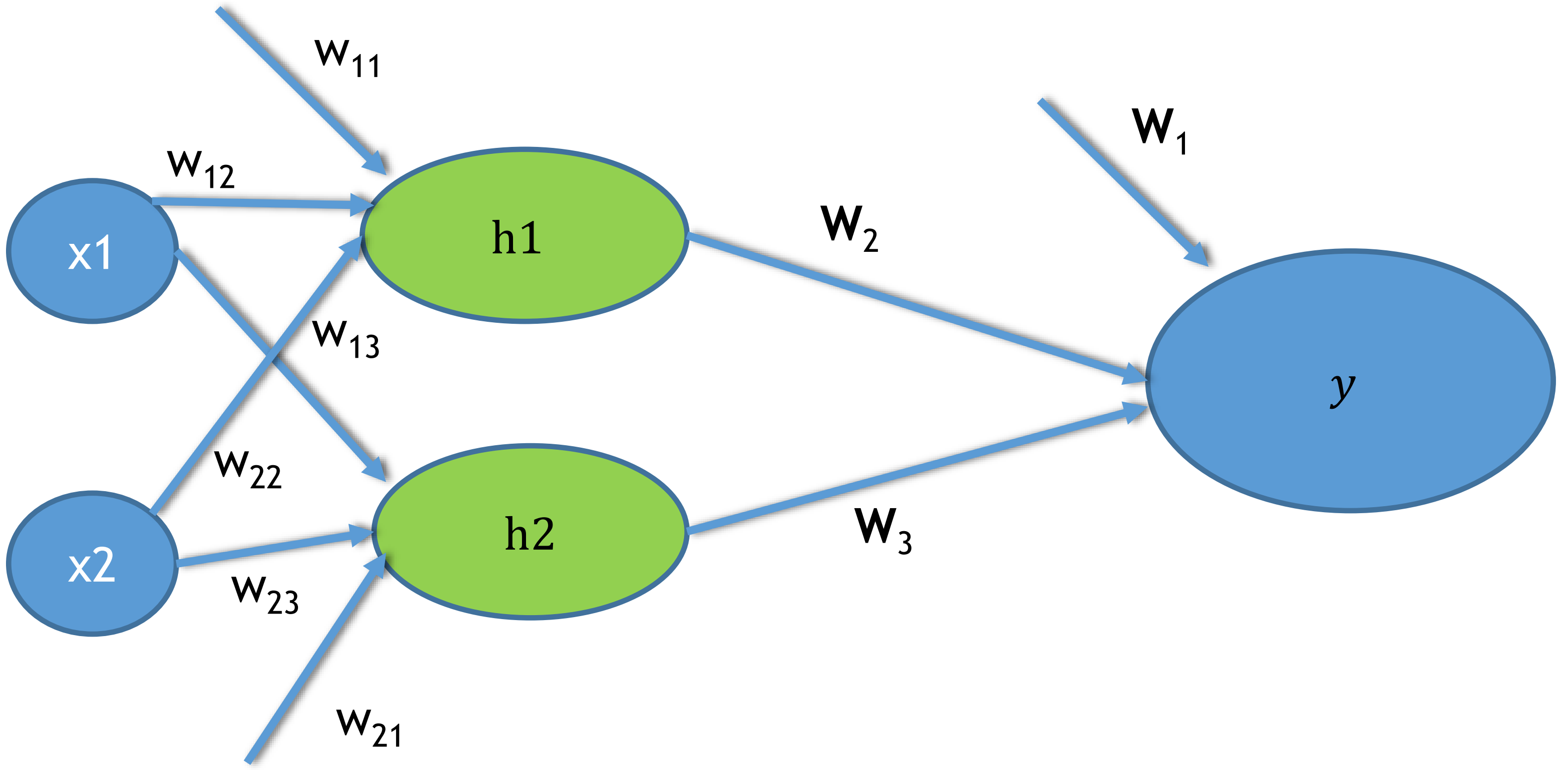
- X1,X2 → inputs
- 1 → bias term
- W's are weights
- 1/(1+e^{-u}) is the sigmoid function
- Y is output

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.

The Neural network Algorithm

Algorithm for Finding weights



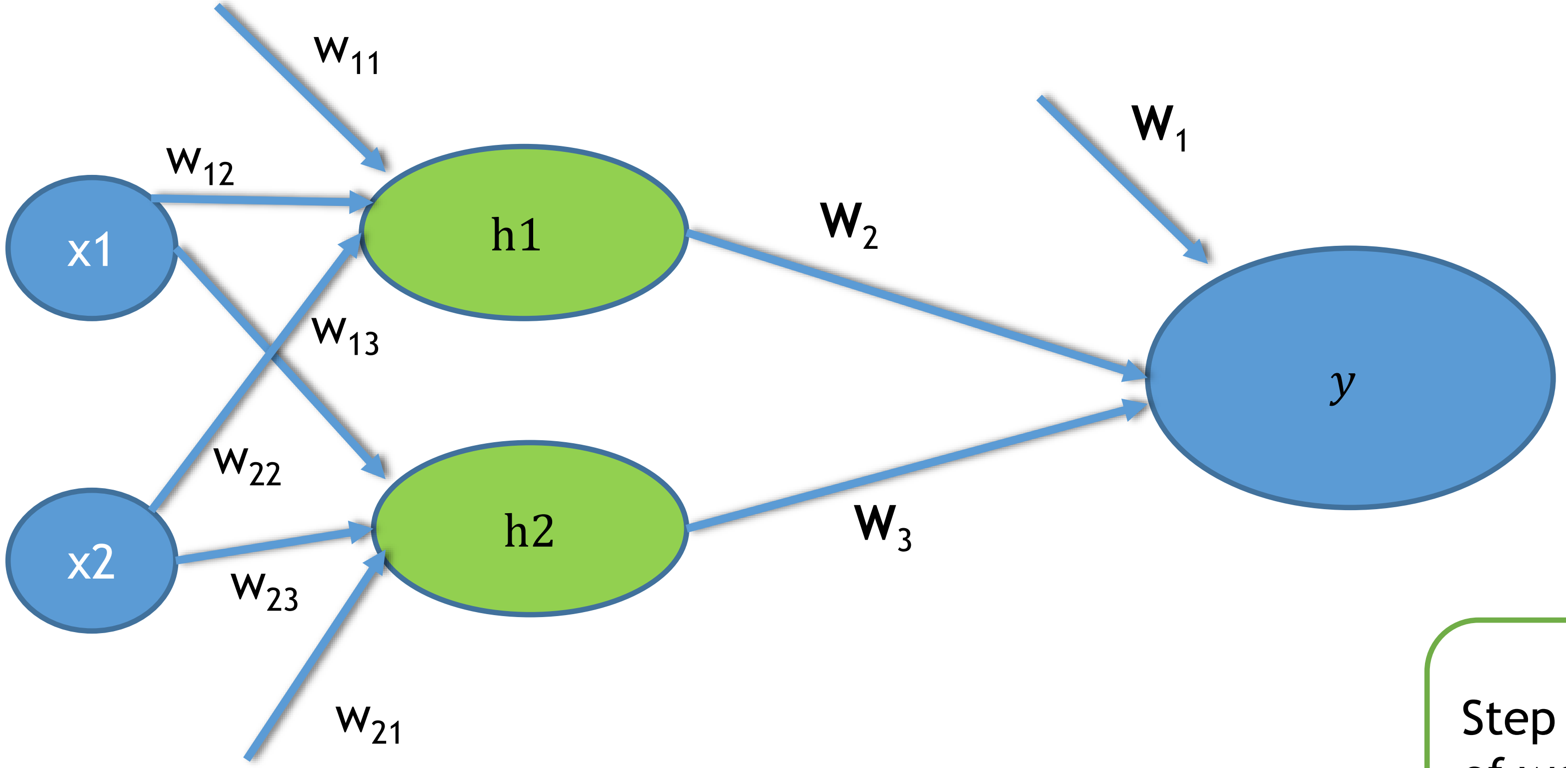
- Algorithm is all about finding the weights/coefficients
- We randomly initialize some weights; Calculate the output by supplying training input; If there is an error the weights are adjusted to reduce this error.

statinfer

The Neural Network Algorithm

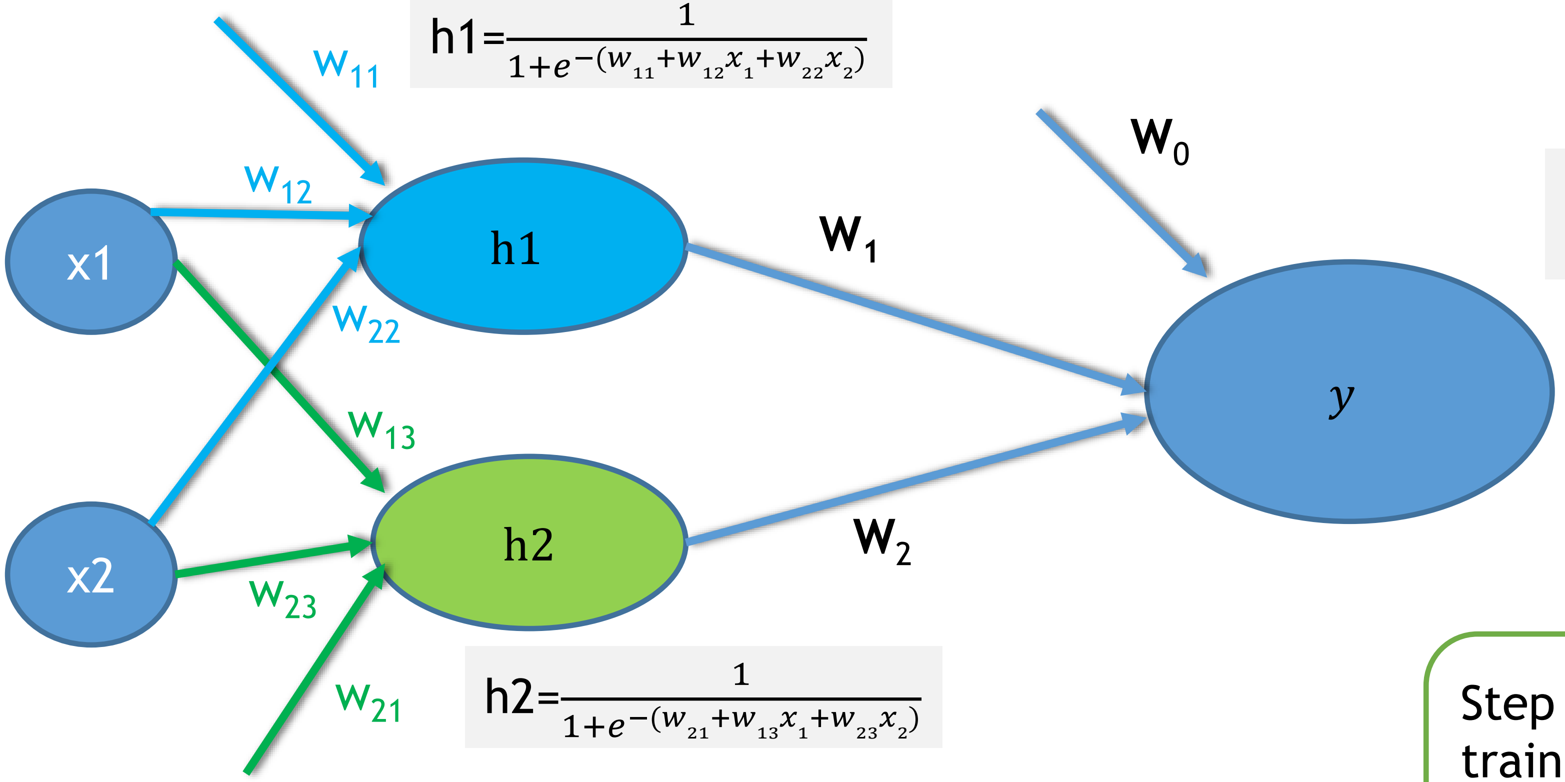
- **Step 1: Initialization of weights:** Randomly select some weights
- **Step 2 : Training & Activation:** Input the training values and perform the calculations forward.
- **Step 3 : Error(cost) Calculation and back propagation:** Calculate the error at the outputs. Use the output error to calculate 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



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

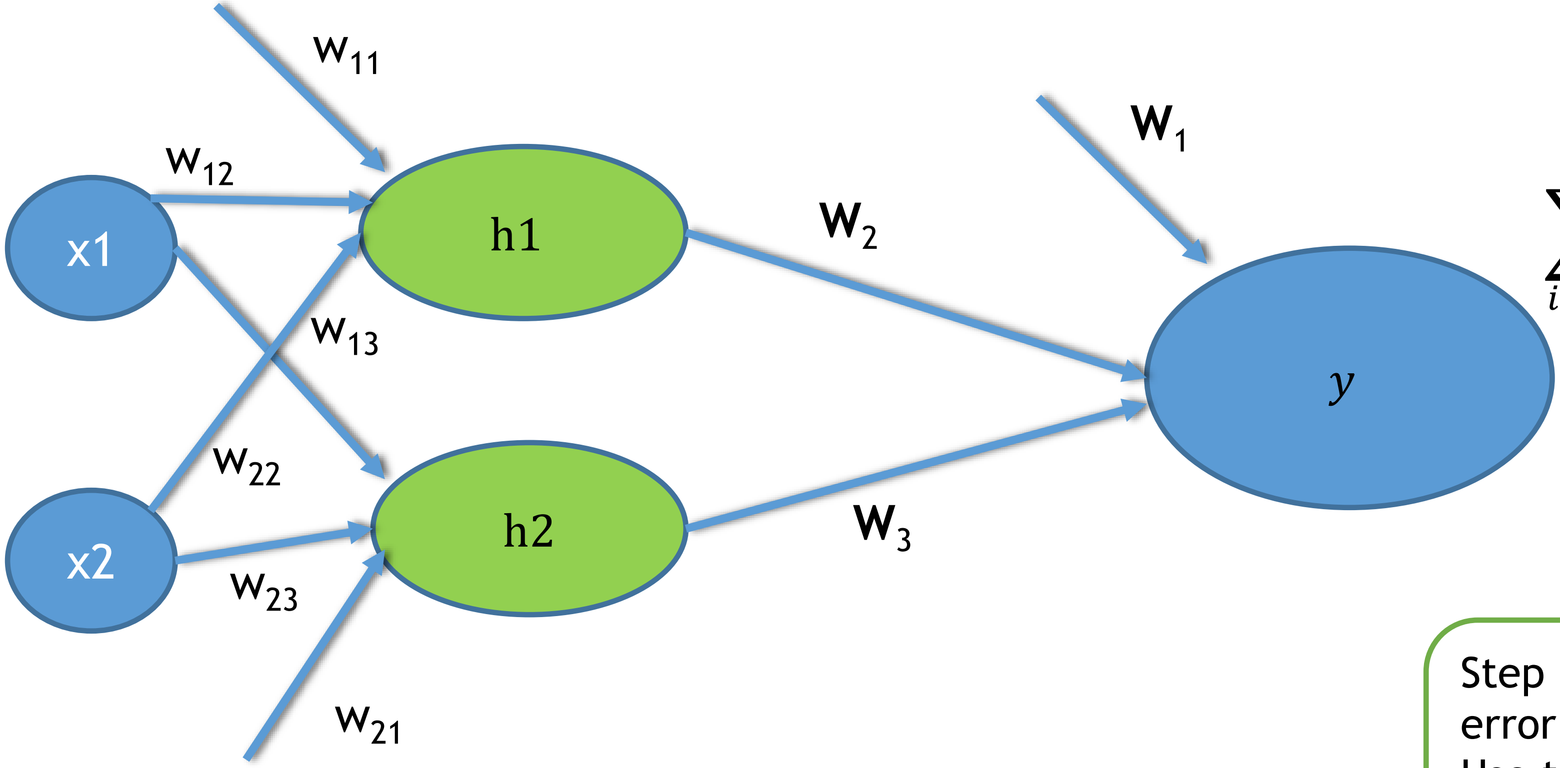
$$h2=\frac{1}{1+e^{-(w_{21}+w_{13}x_1+w_{23}x_2)}}$$

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

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

Training input & calculations – **Feed Forward**

Error Calculation at Output



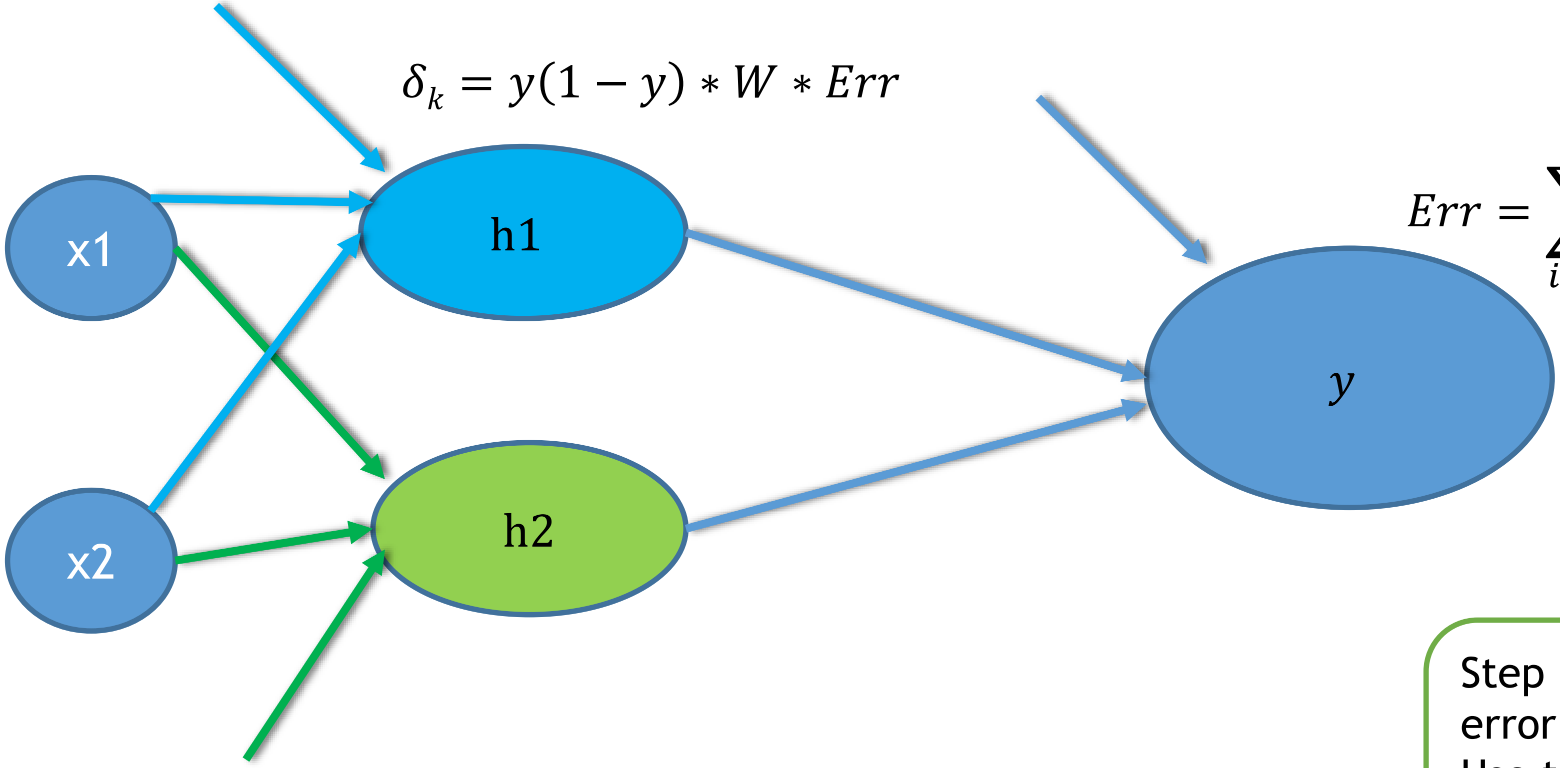
$$\sum_{i=1}^{n}\left[y_i - g\left(\sum_{k=1}^{m} w_k h_{ki}\right)\right]^2$$

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

Error Calculation and backpropagation

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

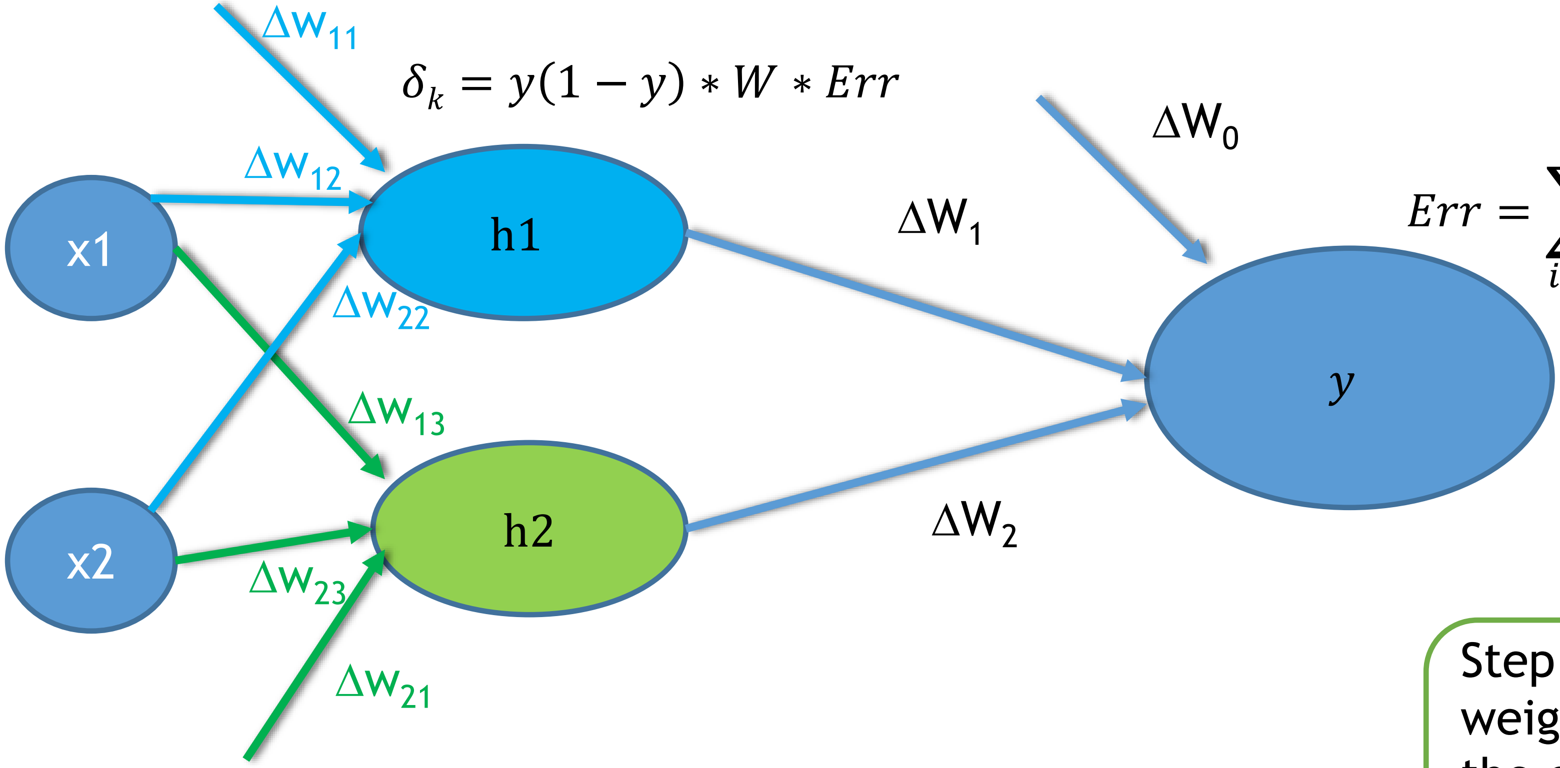
$$Err = \sum_{i=1}^{n}\left[y_i - g\left(\sum_{k=1}^{m} w_k h_{ki}\right)\right]^2$$



Back Propagation - Calculate Errors signals backwards

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

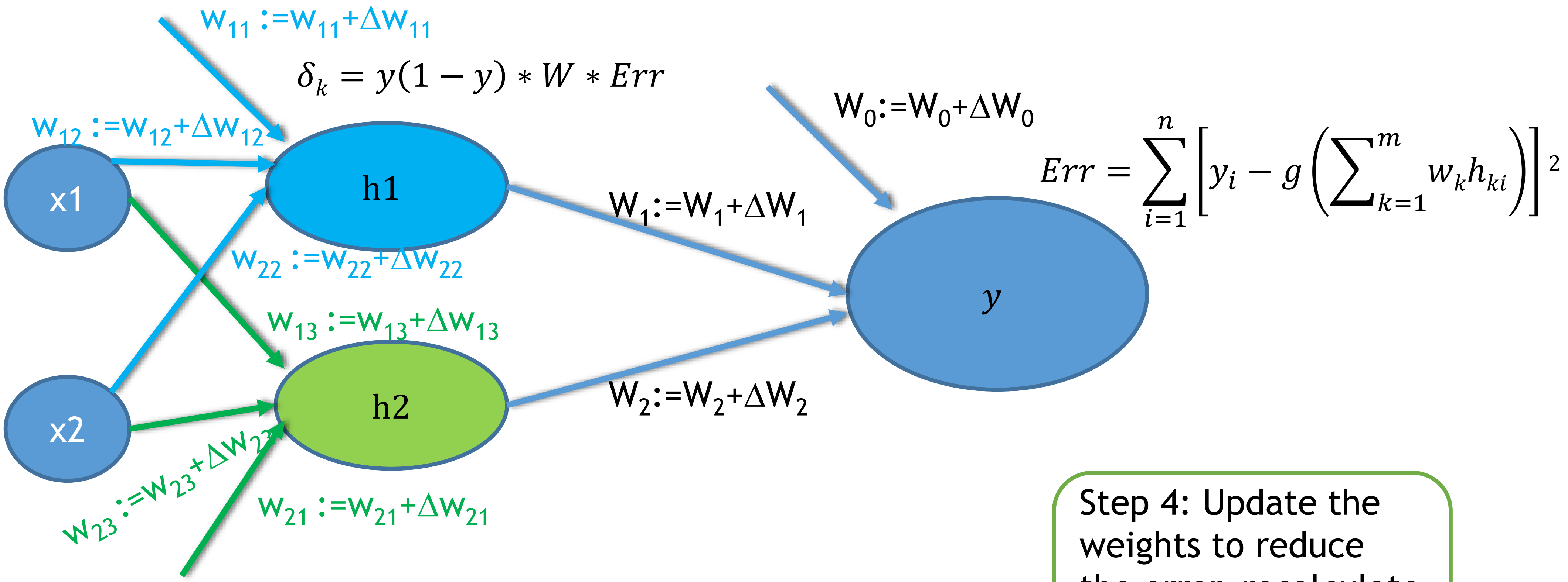
Calculate weight corrections



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

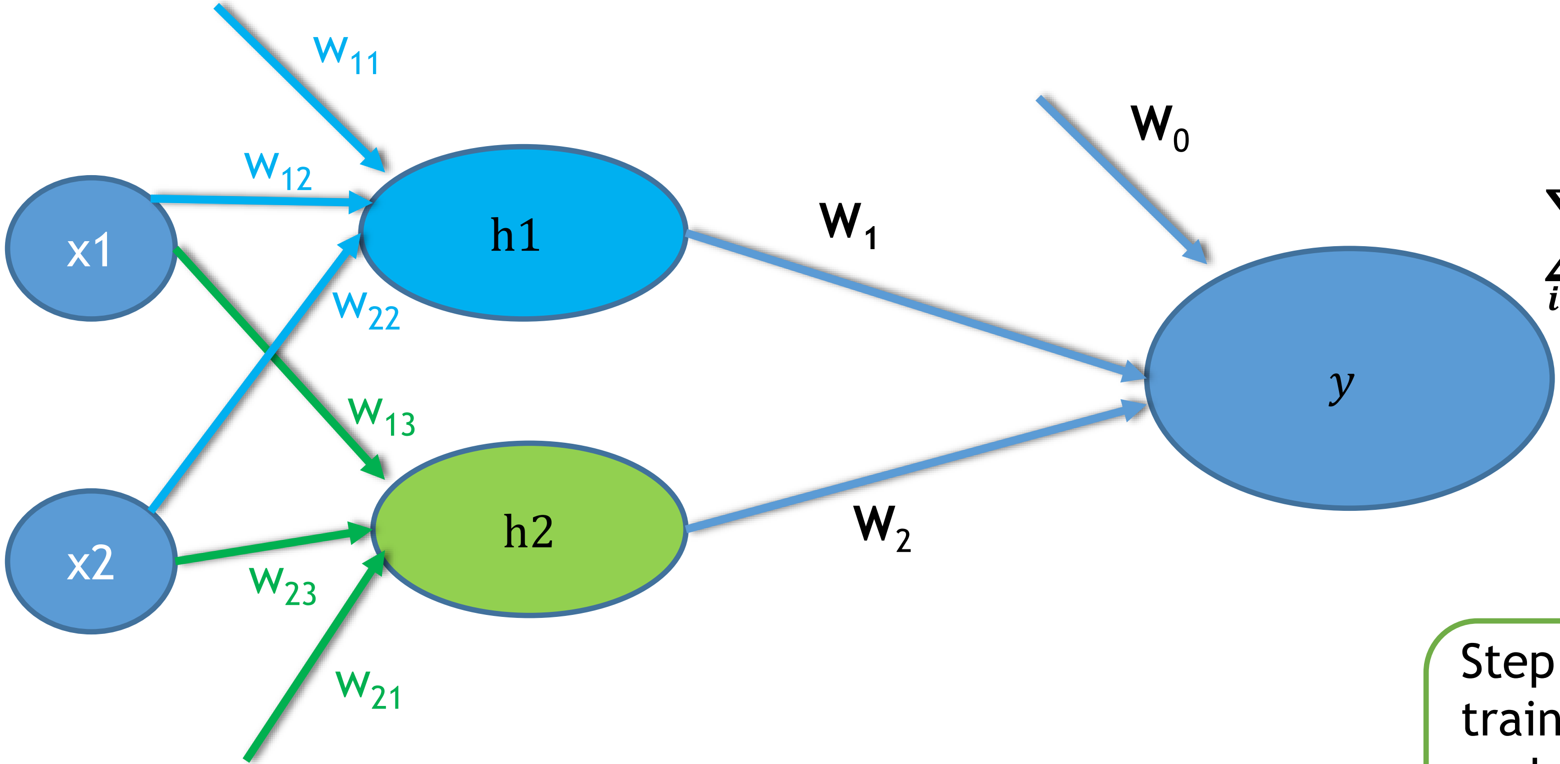
Step 4: Update the weights to reduce the error, recalculate and repeat the process

Update Weights



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

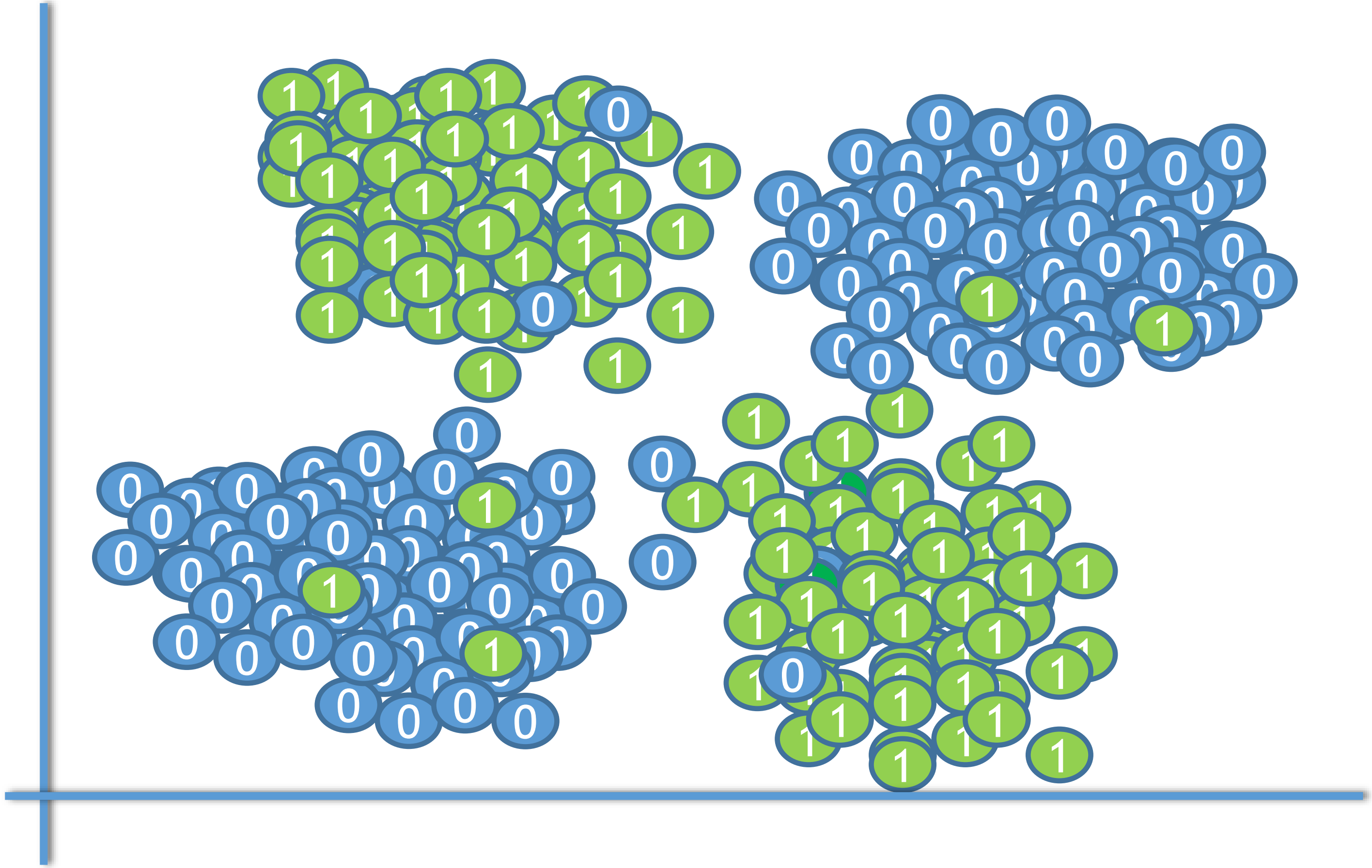
Once Again ….Neural network Algorithm

- 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

statinfer

Neural network Algorithm-Demo

Neural network Algorithm-Demo

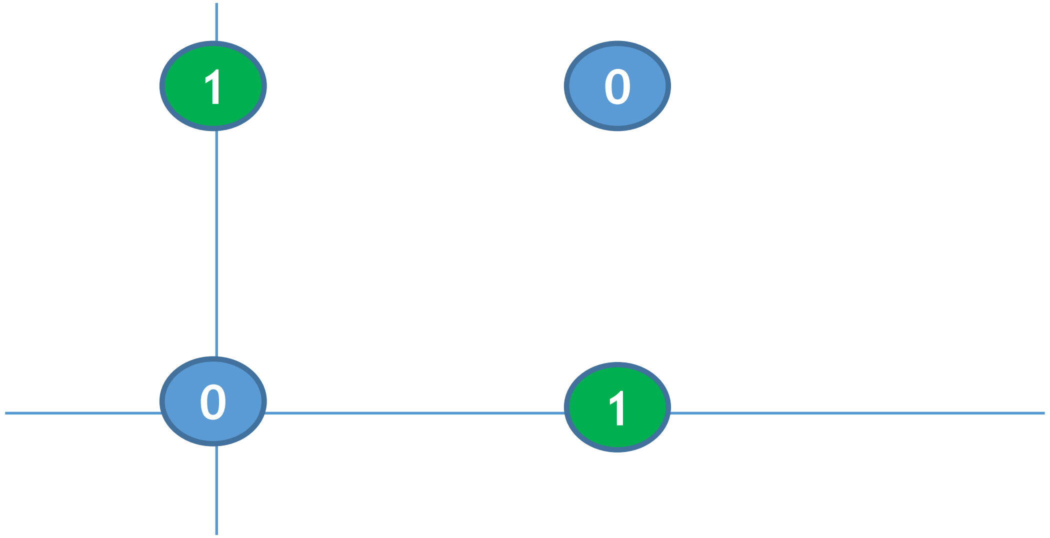


Looks like a dataset that can’t be separated by using single linear decision boundary/perceptron

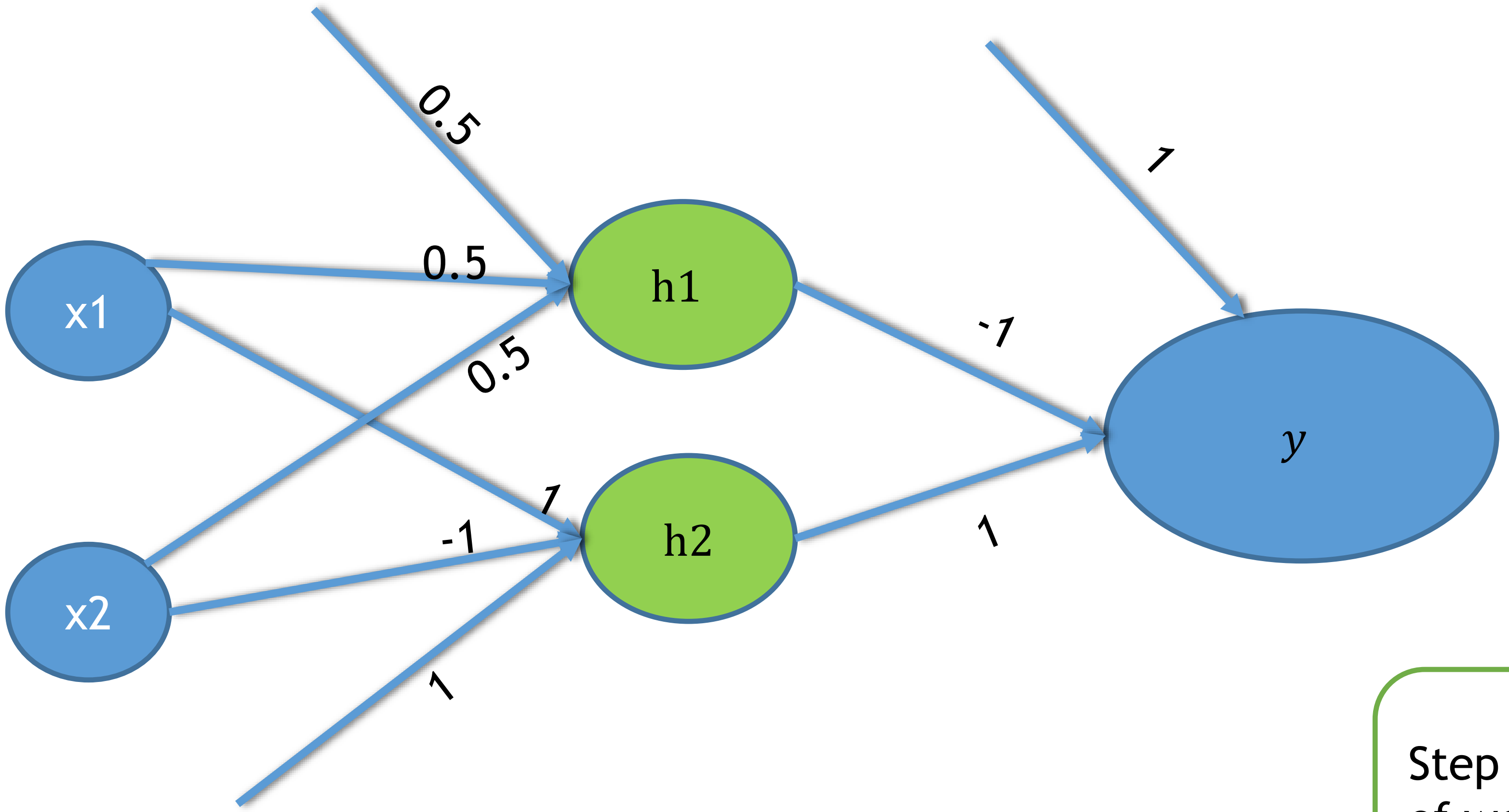
Neural network Algorithm-Demo

- 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



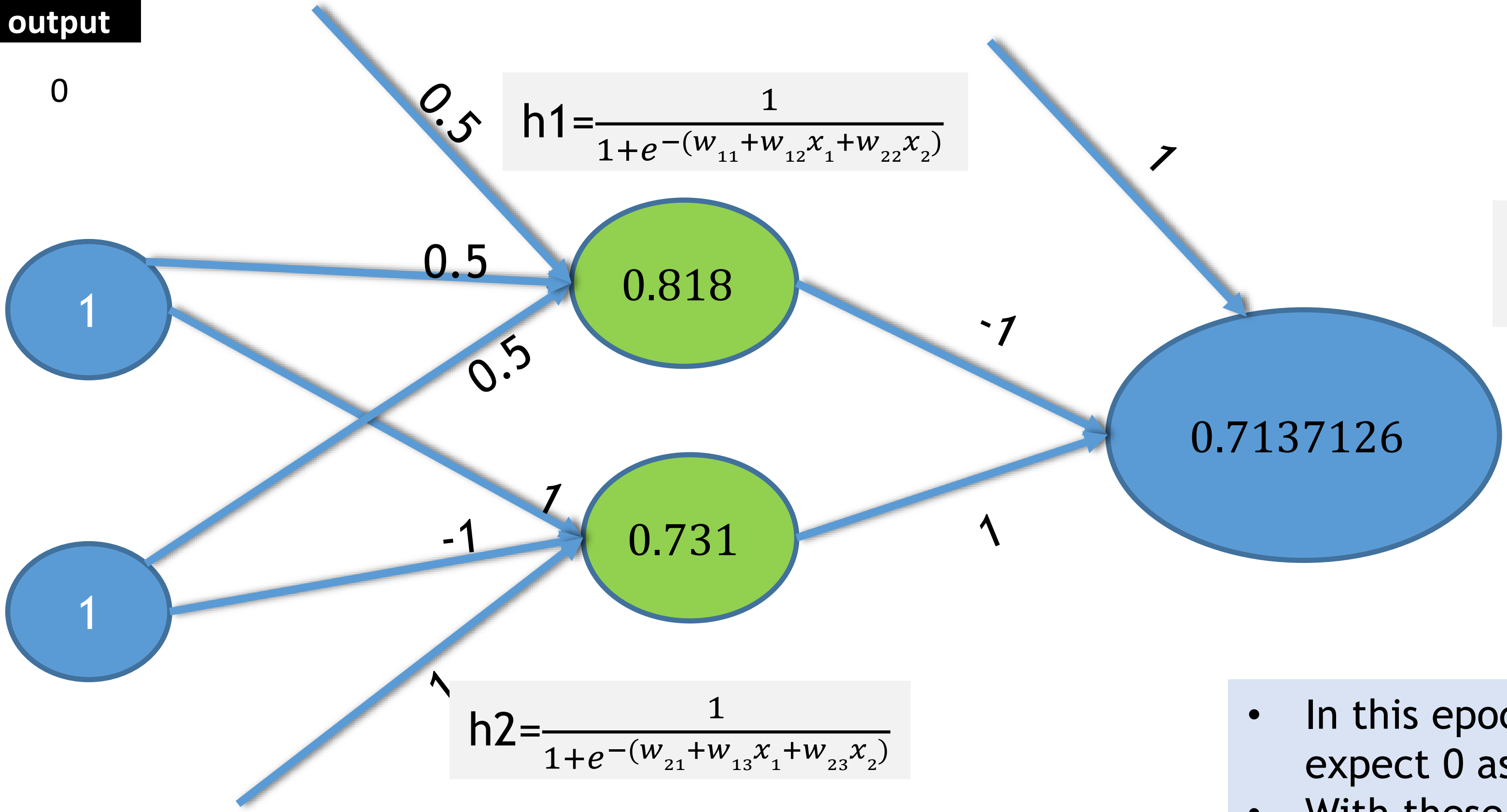
Randomly initialize weights



Step 1: Initialization of weights: Randomly select some weights

Activation

input1	input2	output
1	1	0



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

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

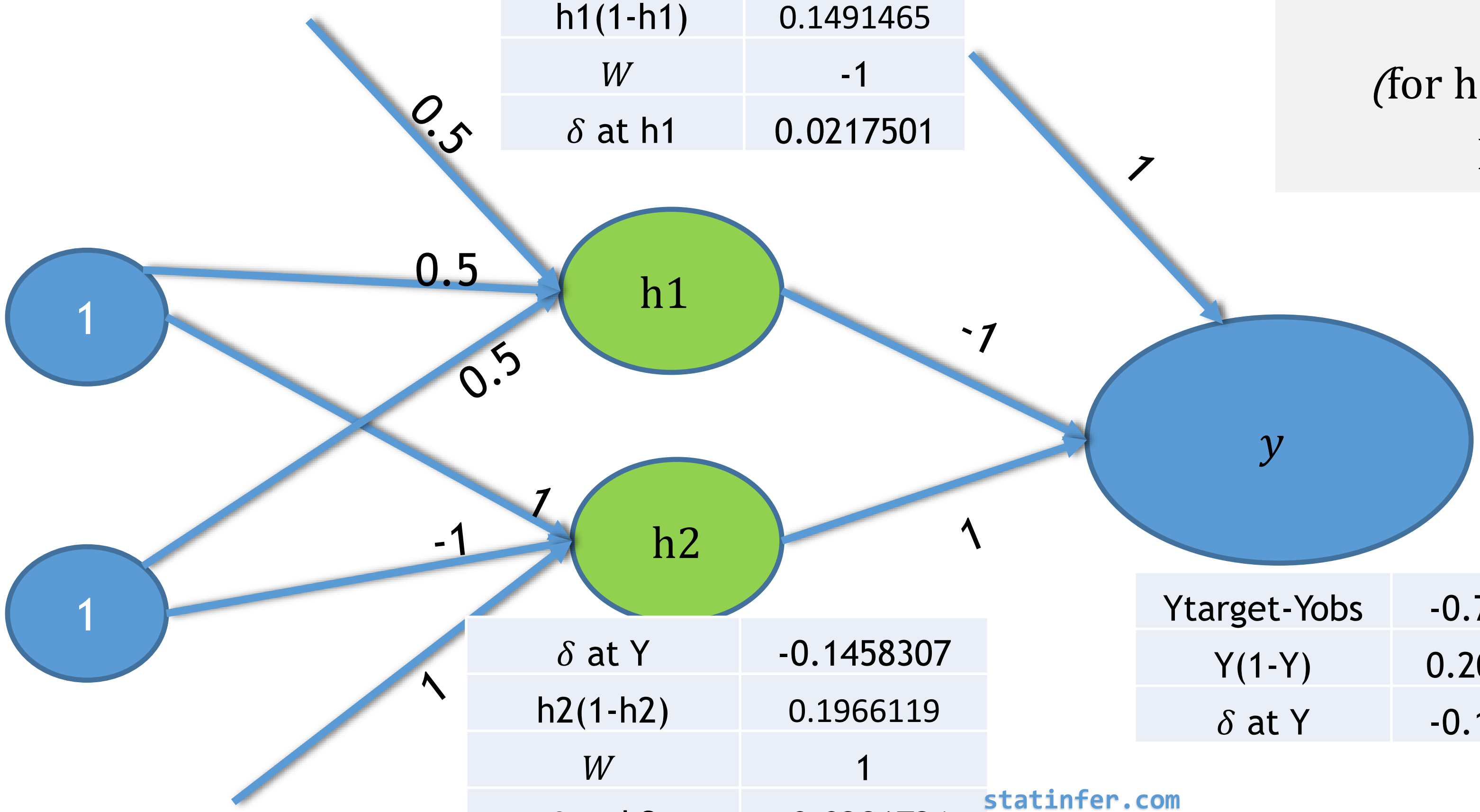
$$y=\frac{1}{1+e^{-(W0+W1h1+W2h2)}}$$

- In this epoch, 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

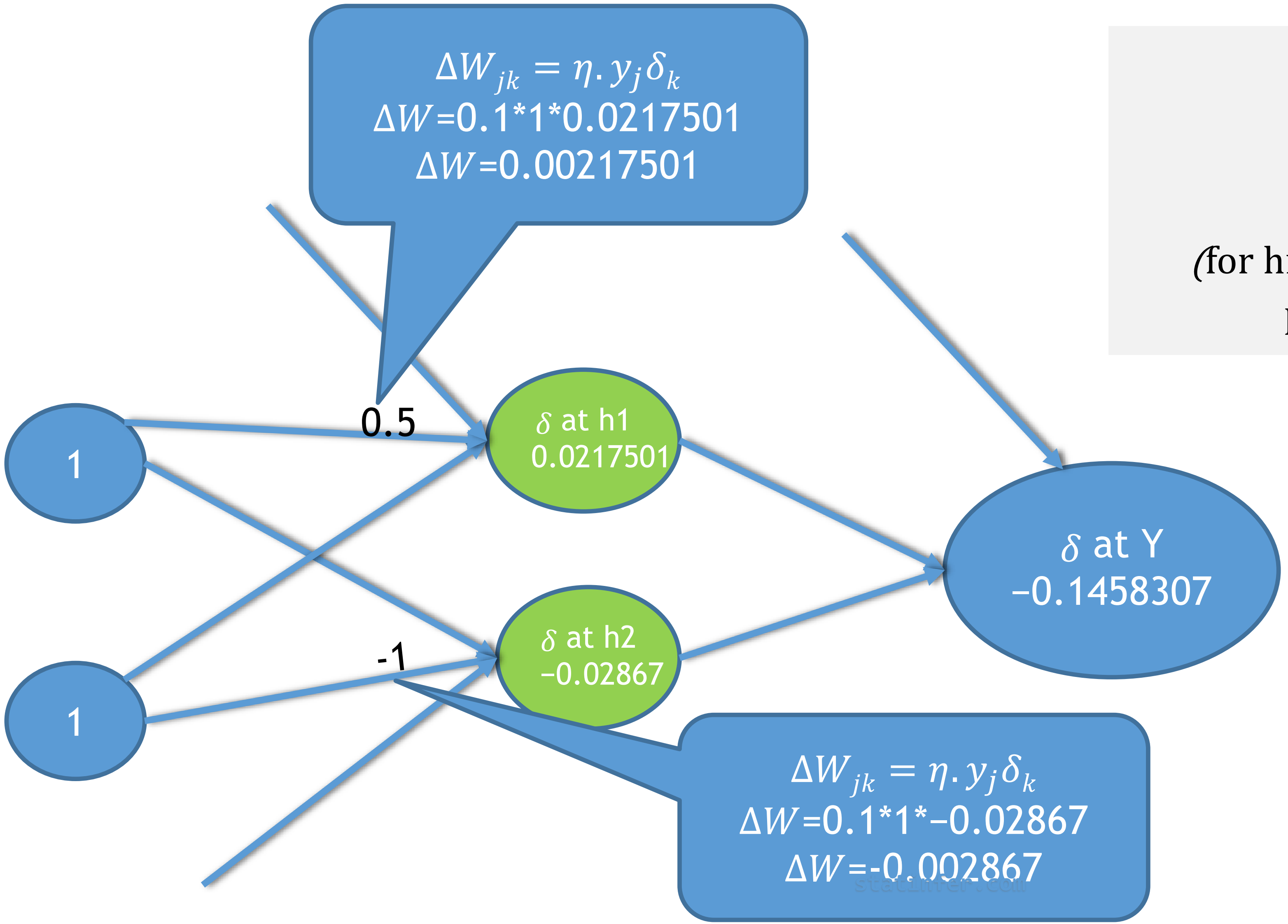
δ at Y	-0.1458307
h1(1-h1)	0.1491465
W	-1
δ at h1	0.0217501



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

$$\delta_k = y_k(1 - y_k) * Err$$

$$(\text{for hidden layers}\delta_k = y_k(1 - y_k) * w_j * Err)$$

Err=Expected output-Actual output

Updated Weights

$$W_{jk} := W_{jk} + \Delta W_{jk}$$

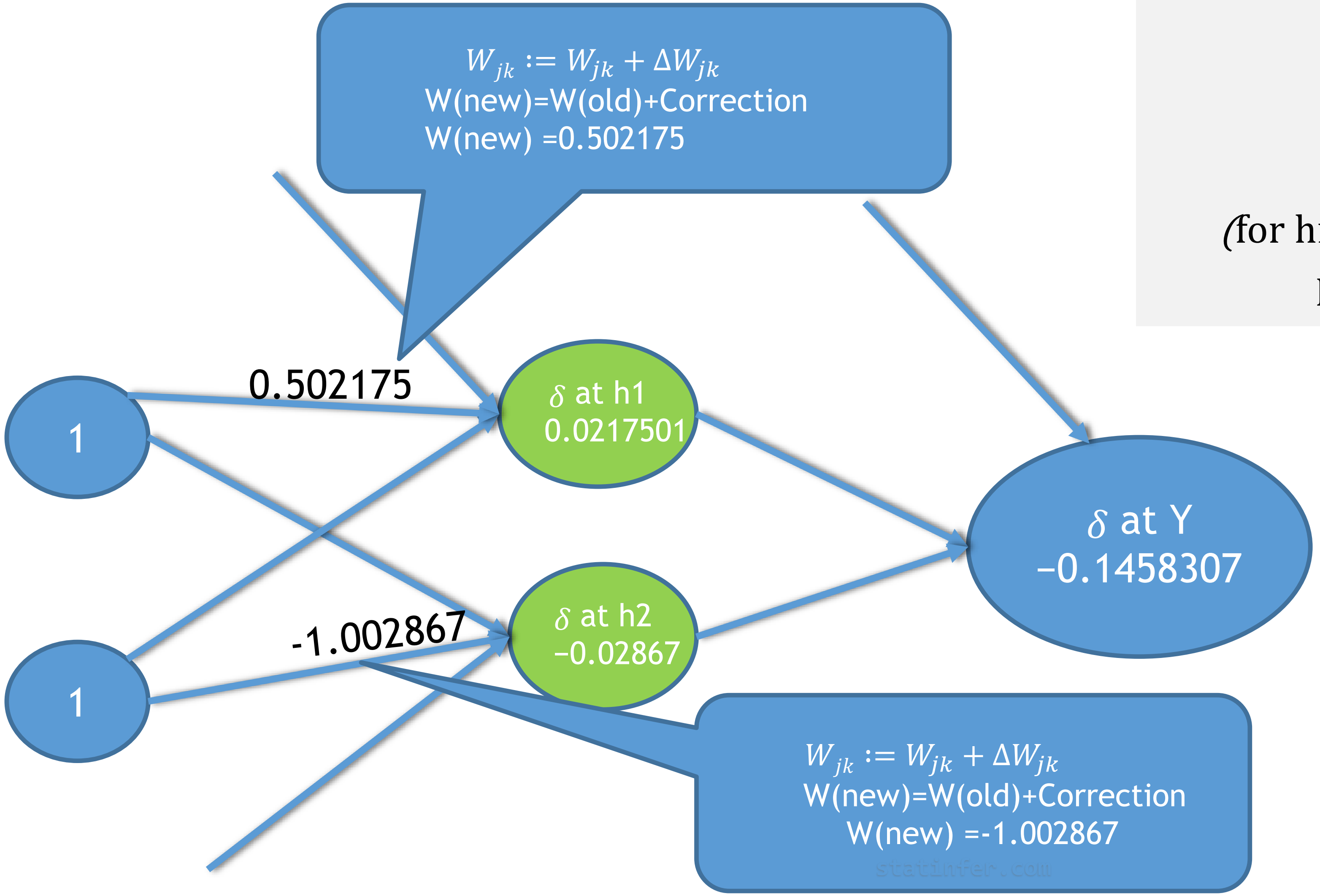
$$where\ \Delta W_{jk} = \eta . y_j \delta_k$$

η is the learning parameter

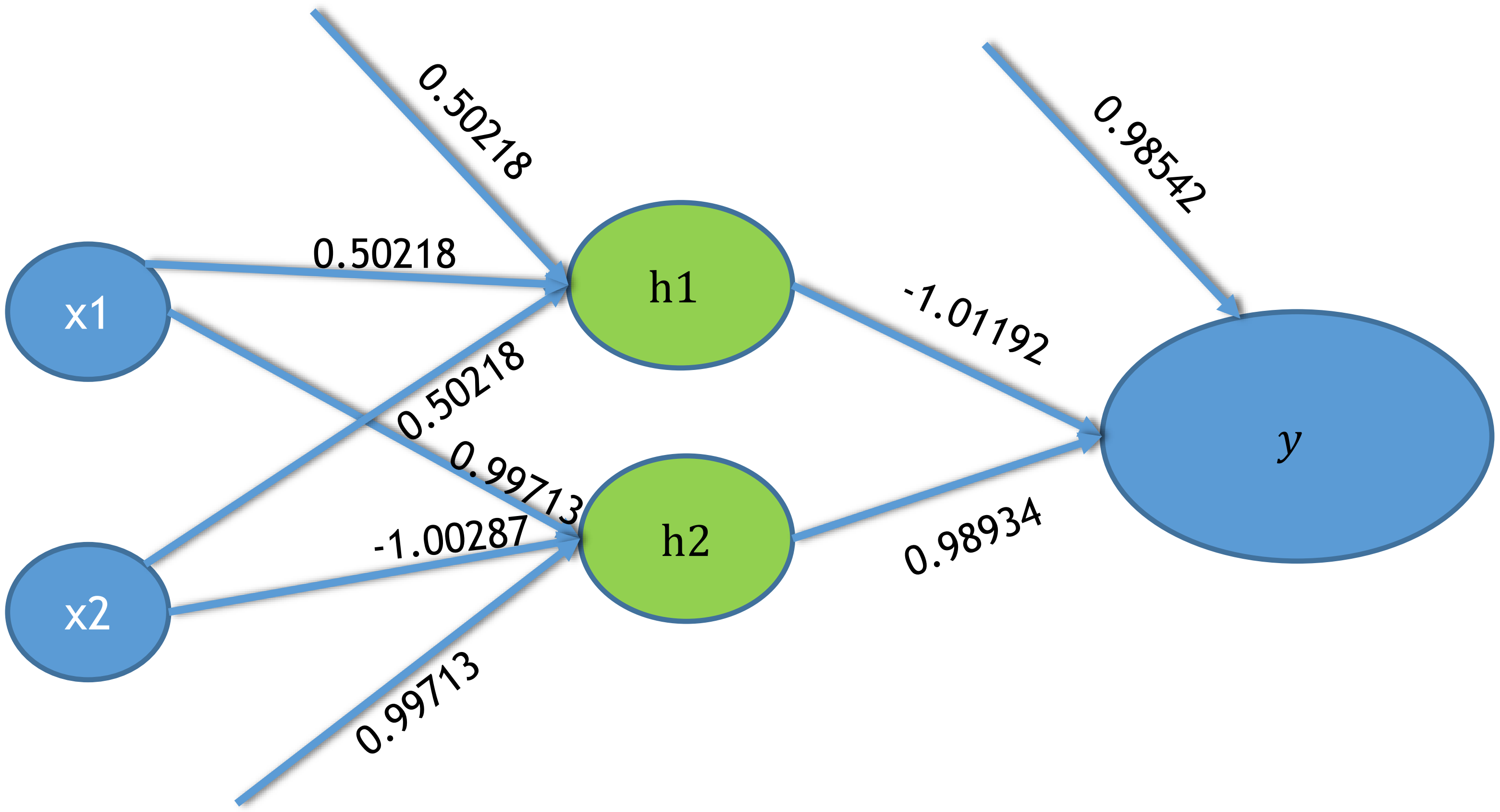
$$\delta_k = y_k(1 - y_k) * Err$$

$$(\text{for hidden layers} \delta_k = y_k(1 - y_k) * w_j * Err)$$

Err=Expected output-Actual output



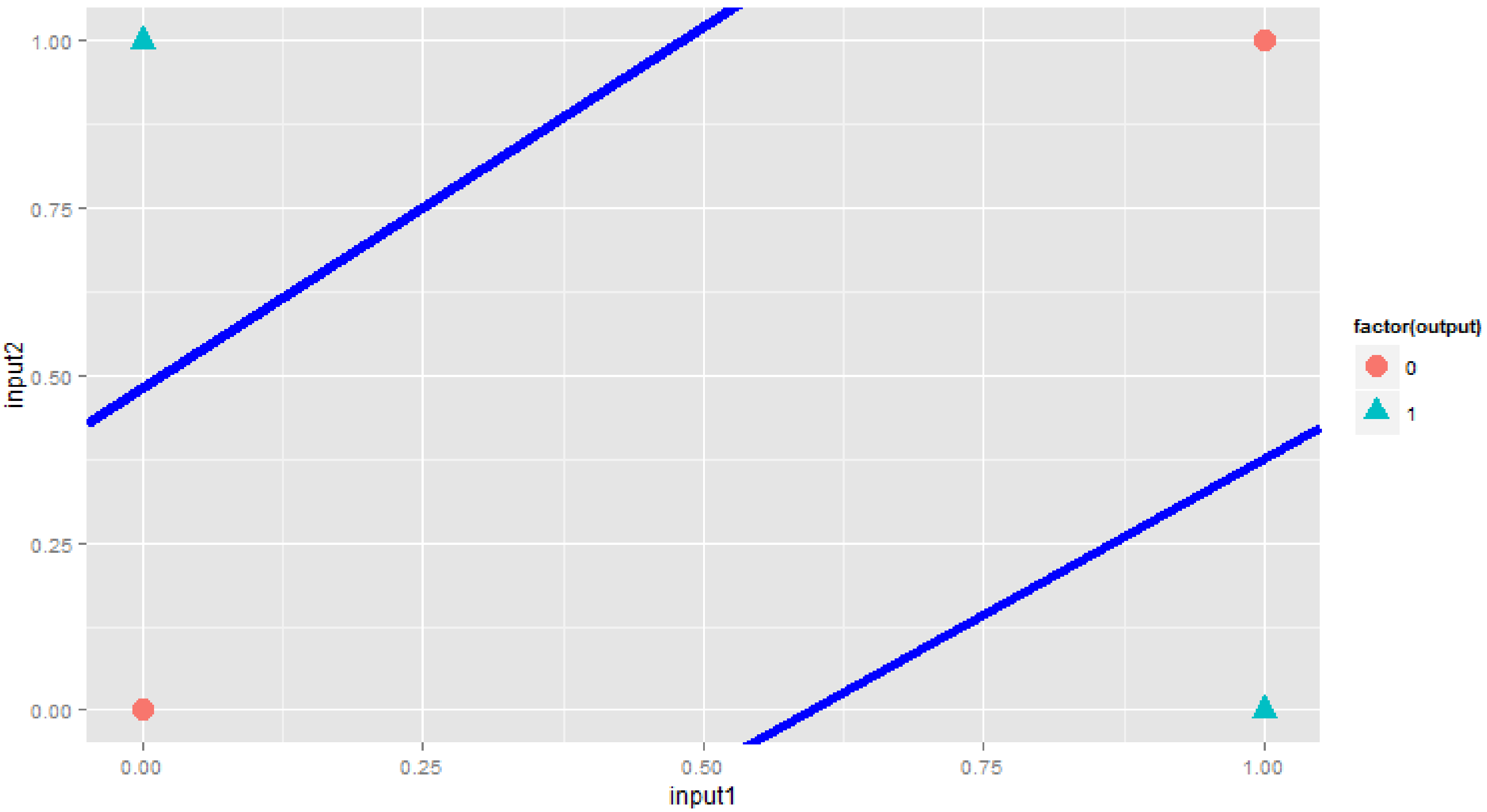
Updated Weights..contd



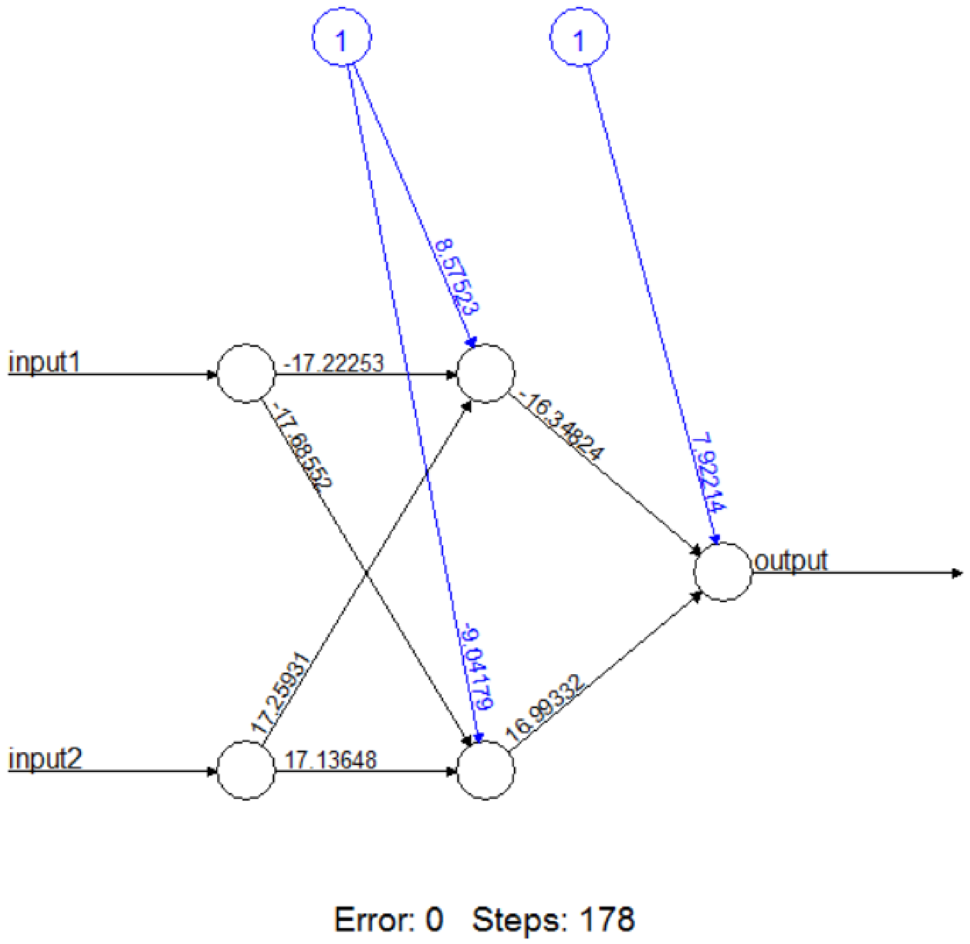
Iterations and Stopping Criteria

- 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



XOR Gate final NN Model



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
- 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 Python

- 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

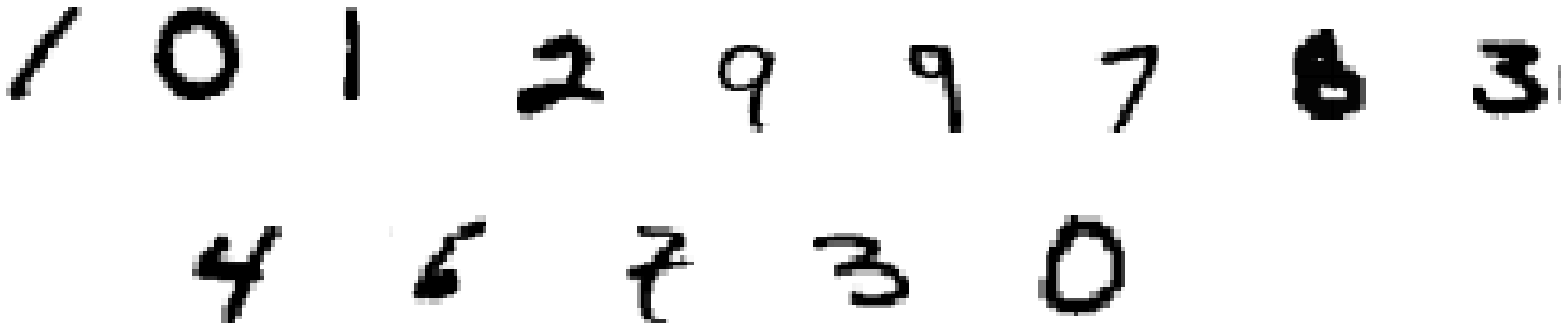
statinfer

LAB: Digit Recognizer

Number Plate Recognition

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?



How computer sees an image

-1	-1	-1	-1	-1	-1	-1	-1	0.9	-0	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	0.3	1	0.3	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-0	1	1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	0.8	1	0.6	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	0.5	1	0.8	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	0.1	1	0.9	-0	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-0	1	1	-0	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	0.9	1	0.3	-1	-1	-1	-1	0.5	1	0.9	0.1	-1	-1
-1	-1	0.3	1	0.9	-1	-1	-1	0.1	1	1	1	1	1	-1	-1
-1	-1	0.8	1	0.3	-1	-1	0.4	1	0.7	-0	-0	1	1	-1	-1
-1	-1	1	1	0.1	-1	0.1	1	0.3	-1	-1	-0	1	0.6	-1	-1
-1	-1	1	1	0.8	0.3	1	0.7	-1	-1	-1	0.5	1	0	-1	-1
-1	-1	0.8	1	1	1	1	0.5	0.2	0.8	0.8	1	0.9	-1	-1	-1
-1	-1	-0	0.8	1	1	1	1	1	1	1	1	0.1	-1	-1	-1
-1	-1	-1	-0	0.8	1	1	1	1	1	1	0.2	-1	-1	-1	-1
-1	-1	-1	-1	-1	-0	0.3	0.8	1	0.5	-0	-1	-1	-1	-1	-1

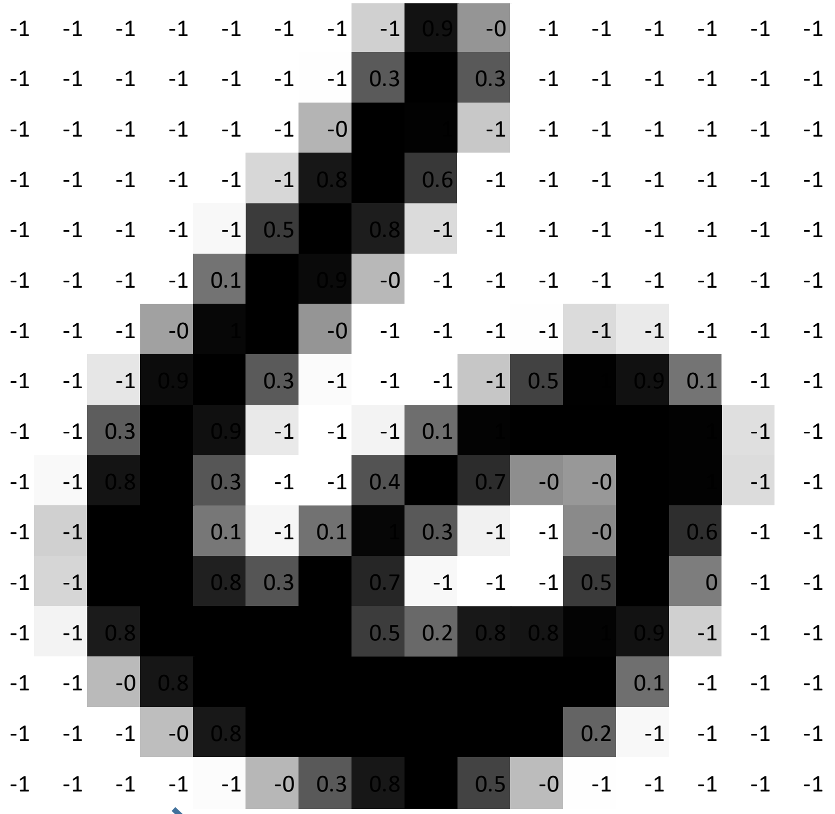
Humans see this

Computer sees this

How computer sees an image

-1	-1	-1	-1	-1	-1	-1	-1	0.9	-0	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	0.3	1	0.3	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-0	1	1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	0.8	1	0.6	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	0.5	1	0.8	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	0.1	1	0.9	-0	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-0	1	1	-0	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	0.9	1	0.3	-1	-1	-1	-1	0.5	1	0.9	0.1	-1	-1
-1	-1	0.3	1	0.9	-1	-1	-1	0.1	1	1	1	1	1	-1	-1
-1	-1	0.8	1	0.3	-1	-1	0.4	1	0.7	-0	-0	1	1	-1	-1
-1	-1	1	1	0.1	-1	0.1	1	0.3	-1	-1	-0	1	0.6	-1	-1
-1	-1	1	1	0.8	0.3	1	0.7	-1	-1	-1	0.5	1	0	-1	-1
-1	-1	0.8	1	1	1	1	0.5	0.2	0.8	0.8	1	0.9	-1	-1	-1
-1	-1	-0	0.8	1	1	1	1	1	1	1	1	0.1	-1	-1	-1
-1	-1	-1	-0	0.8	1	1	1	1	1	1	0.2	-1	-1	-1	-1
-1	-1	-1	-1	-1	-0	0.3	0.8	1	0.5	-0	-1	-1	-1	-1	-1

Computer sees this



Same matrix, highlight the cells based on cell value

How computer sees an image

Human Vision

-1	-1	-1	-1	-1	-1	-1	-1	0.9	-0	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	0.3	1	0.3	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-0	1	1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	0.8	1	0.6	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	0.5	1	0.8	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	0.1	1	0.9	-0	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-0	1	1	-0	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	0.9	1	0.3	-1	-1	-1	-1	0.5	1	0.9	0.1	-1	-1
-1	-1	0.3	1	0.9	-1	-1	-1	0.1	1	1	1	1	1	-1	-1
-1	-1	0.8	1	0.3	-1	-1	0.4	1	0.7	-0	-0	1	1	-1	-1
-1	-1	1	1	0.1	-1	0.1	1	0.3	-1	-1	-0	1	0.6	-1	-1
-1	-1	1	1	0.8	0.3	1	0.7	-1	-1	-1	0.5	1	0	-1	-1
-1	-1	0.8	1	1	1	1	0.5	0.2	0.8	0.8	1	0.9	-1	-1	-1
-1	-1	-0	0.8	1	1	1	1	1	1	1	1	0.1	-1	-1	-1
-1	-1	-1	-0	0.8	1	1	1	1	1	1	0.2	-1	-1	-1	-1
-1	-1	-1	-1	-1	-0	0.3	0.8	1	0.5	-0	-1	-1	-1	-1	-1

Computer Vision

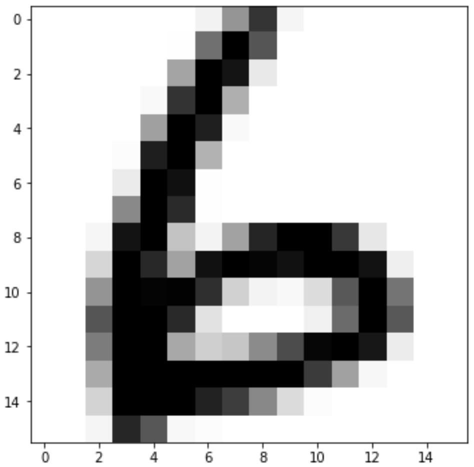
Converting image into numbers

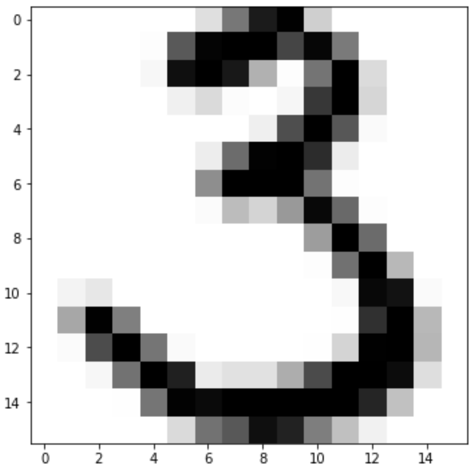
```
import matplotlib.pyplot as plt

x=plt.imread(r'D:\Datasets\cat.jpeg')
plt.imshow(x)

print('Shape of the image',x.shape)
print(x)
```

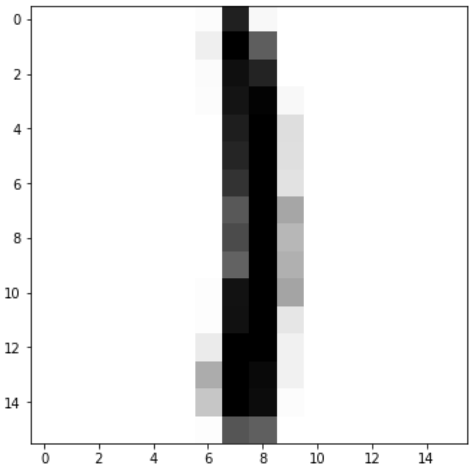
Code: Digit Recognizer

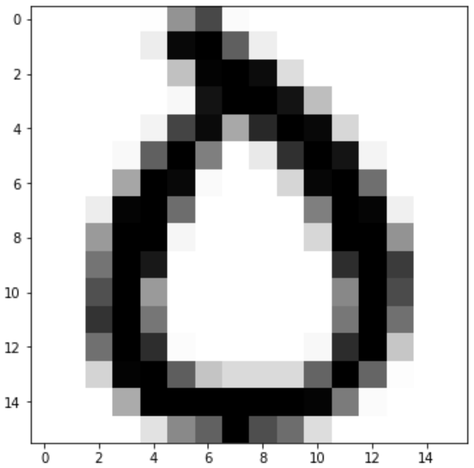
```
#Importing test and training data
import numpy as np
digits_train = np.loadtxt("D:\\Google Drive\\Training\\Datasets\\Digit Recognizer\\USPS\\zip.train.txt")

#digits_train is numpy array. we convert it into dataframe for better handling
train_data=pd.DataFrame(digits_train)
train_data.shape

digits_test = np.loadtxt("D:\\Google Drive\\Training\\Datasets\\Digit Recognizer\\USPS\\zip.test.txt")
#digits_test is numpy array. we convert it into dataframe for better handling
test_data=pd.DataFrame(digits_test)
test_data.shape

train_data[0].value_counts()     #To get labels of the images

import matplotlib.pyplot as plt
```

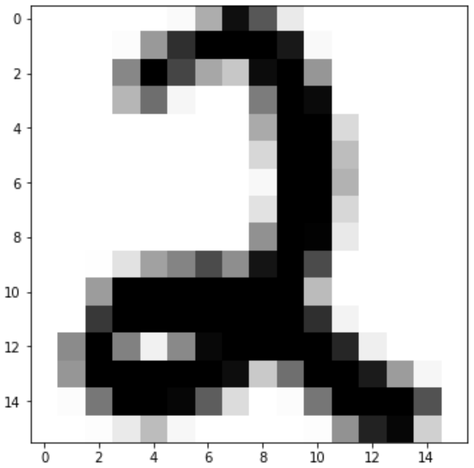
Code: Digit Recognizer

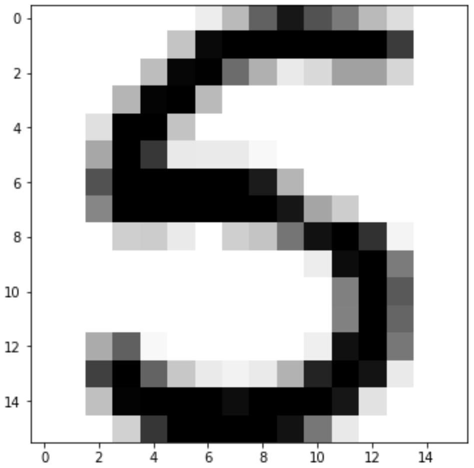













x1
x2
x256
y0
y1
y2
y9

Code: Digit Recognizer

```
#getting minimum and maximum of each column of x_train into a list
min_max_all_cols=[[X_train[i][0:].min(), X_train[i][0:].max()] for i in range(1,X_t
rain.shape[1]+1)]
print(len(min_max_all_cols))
print(min_max_all_cols)
```

```
#Creating multiple binary columns for multiple outputs
#####We need these variables while building the model
digit_labels=pd.DataFrame()

#Convert target into onehot encoding
digit_labels = pd.get_dummies(y_train)

#see our newly created labels data
digit_labels.head(10)
```

Building NN– Configure neural net

Min and max of all cols

Number of output classes

```
net = nl.net.newff(minmax=min_max_all_cols,size=[20,10],transf=[nl.trans.LogSig()]*2)
```

Number of hidden nodes

Sigmoid function

```
net.trainf = nl.train.train_rprop
```

Train algorithms based gradients algorithms - Resilient Backpropagation

Building NN– Train neural net

Y_train

```
net.train(X_train, digit_labels, show=0, epochs=300)
```

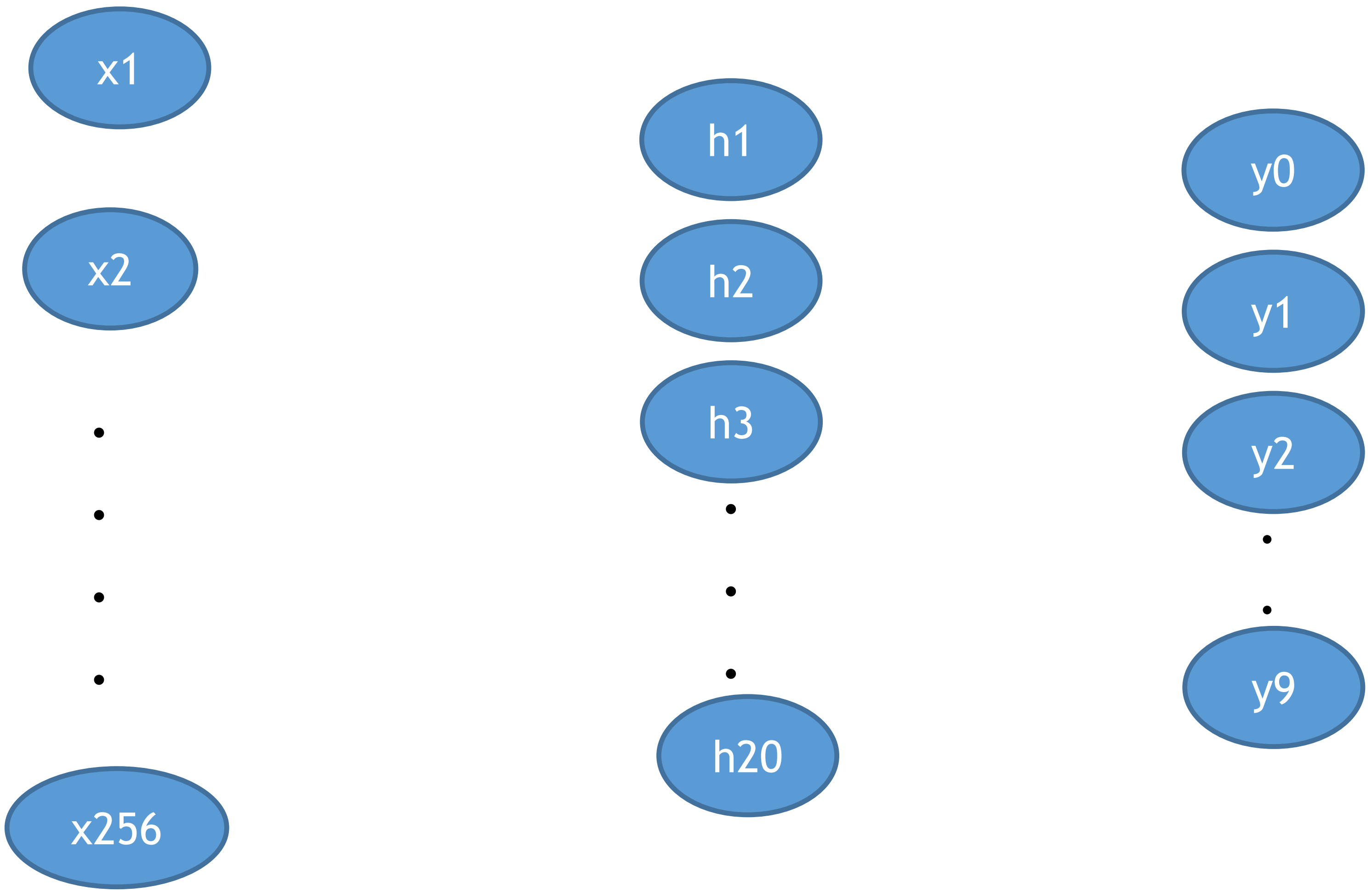
X_tran

Number of epochs in training data

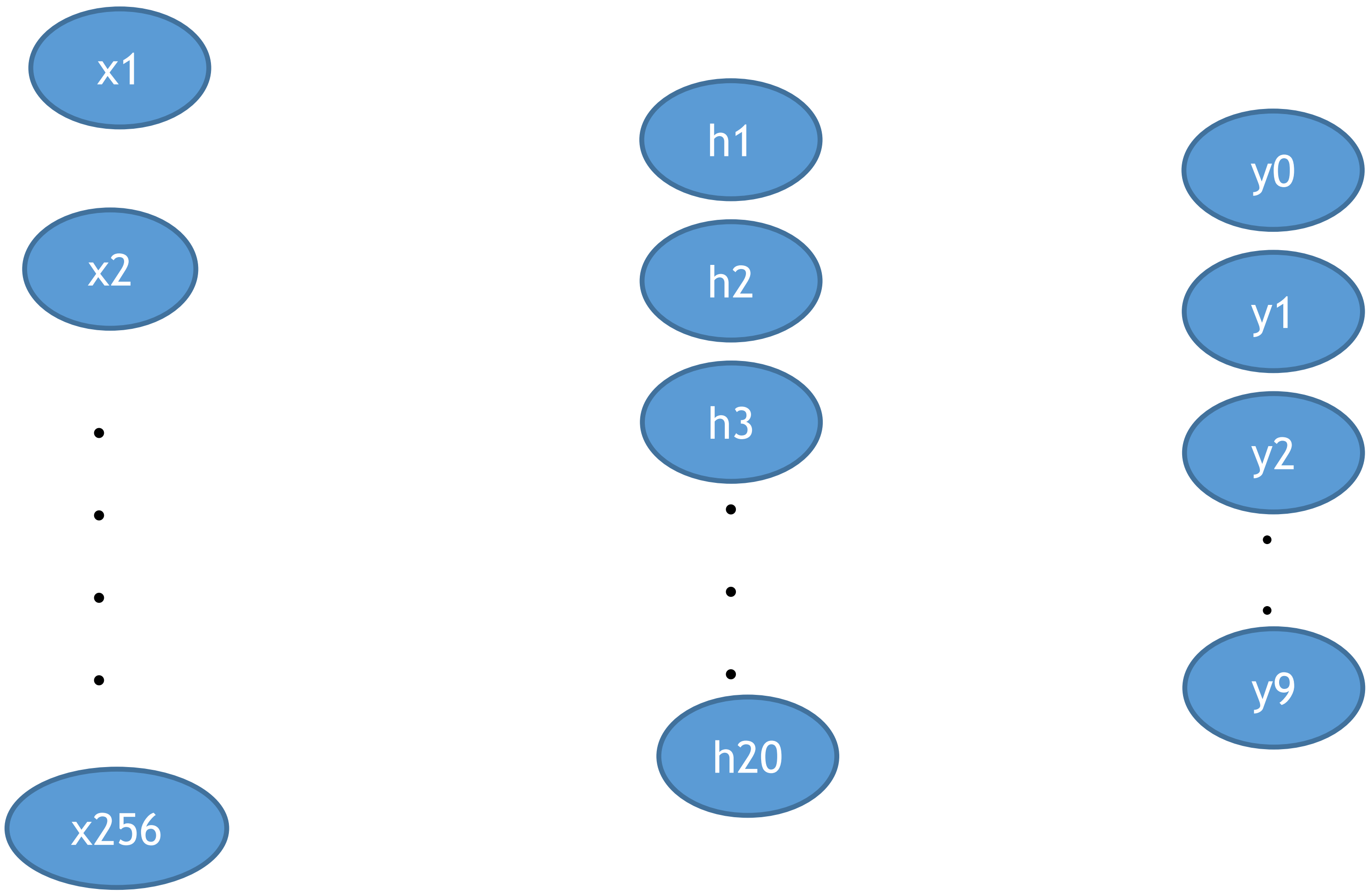
Code: Digit Recognizer

```
##Configure the network
net = nl.net.newff(minmax=min_max_all_cols,size=[20,10],transf=[nl.trans.LogSig()]*2)

#Training method is Resilient Backpropagation method
net.trainf = nl.train.train_rprop

##Train the network
import time
start_time = time.time()
net.train(X_train, digit_labels, show=0, epochs=300)
print("--- %s seconds ---" % (time.time() - start_time))
```

Final Network



Calculate the weights in the network



Code: Digit Recognizer

```
# Prediction testing data
x_test=test_data.drop(test_data.columns[[0]], axis=1)
y_test=test_data[0:][0]

predicted_values = net.sim(x_test.as_matrix())
predict=pd.DataFrame(predicted_values)

index=predict.idxmax(axis=1)

#confusion matrix
from sklearn.metrics import confusion_matrix as cm
ConfusionMatrix = cm(y_test,index)
print(ConfusionMatrix)

#accuracy
accuracy=np.trace(ConfusionMatrix)/sum(sum(ConfusionMatrix))
print(accuracy)

error=1-accuracy
print(error)
```

```
....
[[344   0   1   1   5   2   4   0   1   1]
 [  1 252   1   1   2   1   5   0   0   1]
 [  2   0 175   3   7   1   1   2   7   0]
 [  5   0   7 137   2  10   0   0   3   2]
 [  1   2   3   0 182   1   2   2   1   6]
 [  5   1   3   9   5 130   2   1   2   2]
 [  3   0   2   0   4   4 156   0   1   0]
 [  0   0   0   4   8   0   0 131   0   4]
 [  4   0   3   4   7   4   1   2 138   3]
 [  0   1   0   1   6   1   0   3   4 161]]
0.899850523169
0.100149476831
```

Output: Digit Recognizer

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

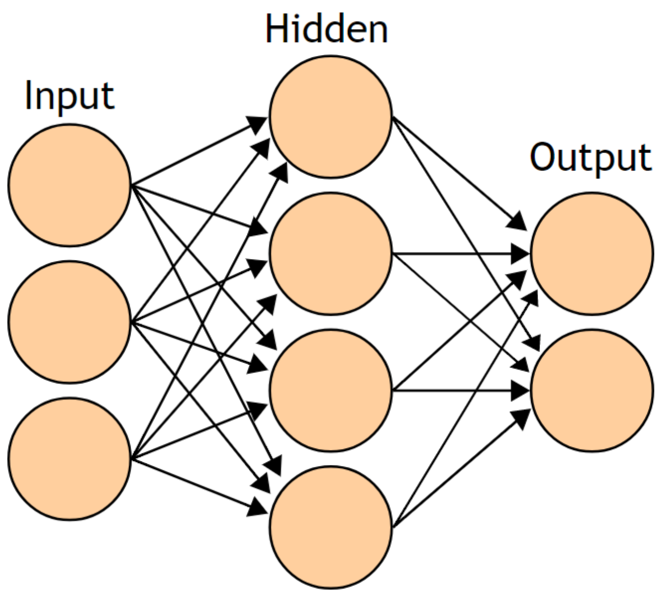
Drawbacks of Neural Networks

- 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

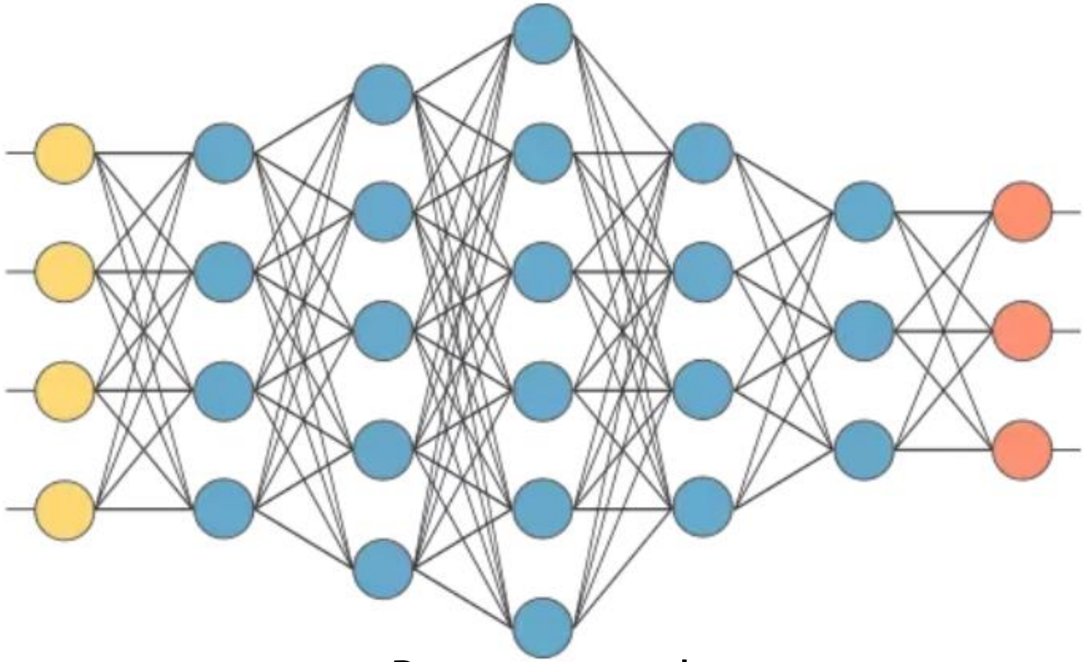
Deep vs Shallow networks

A neural network with single hidden layer is called a shallow network

A neural network with more than one hidden layer is called deep neural network



shallow network



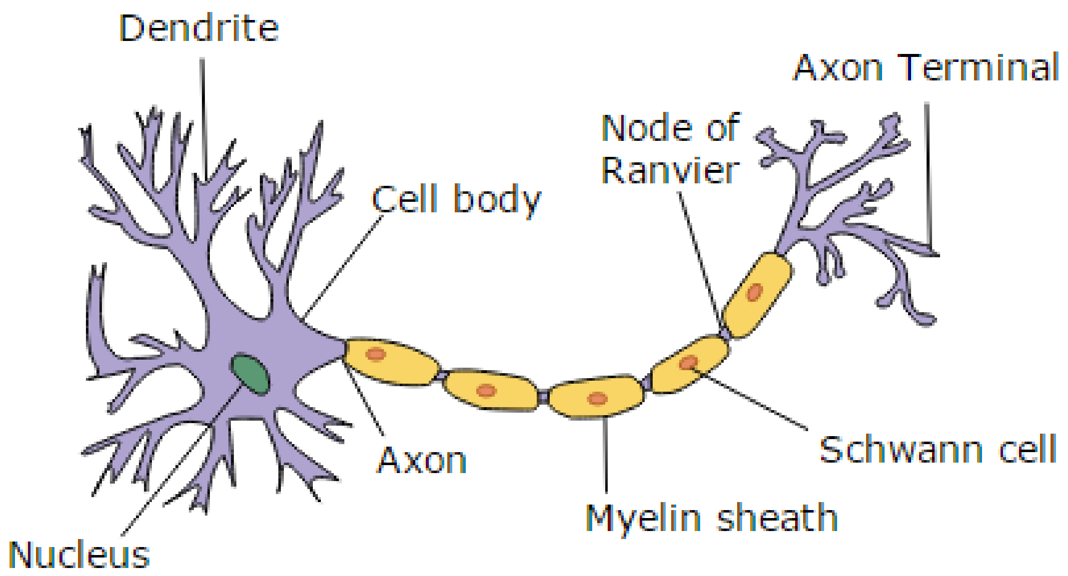
Deep network

Why the name neural network?

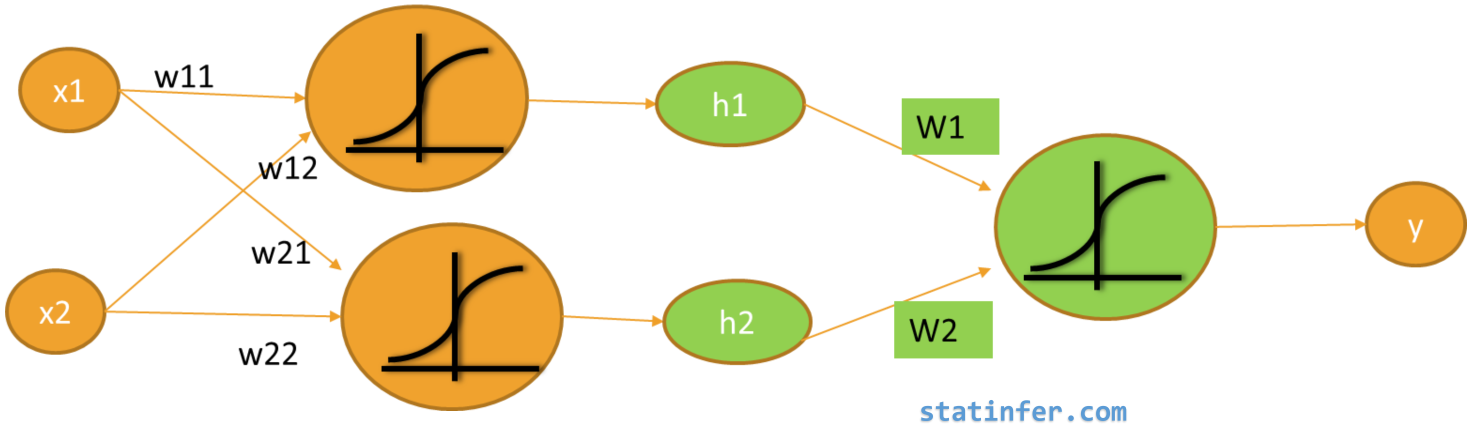
Why the name neural network?

- 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

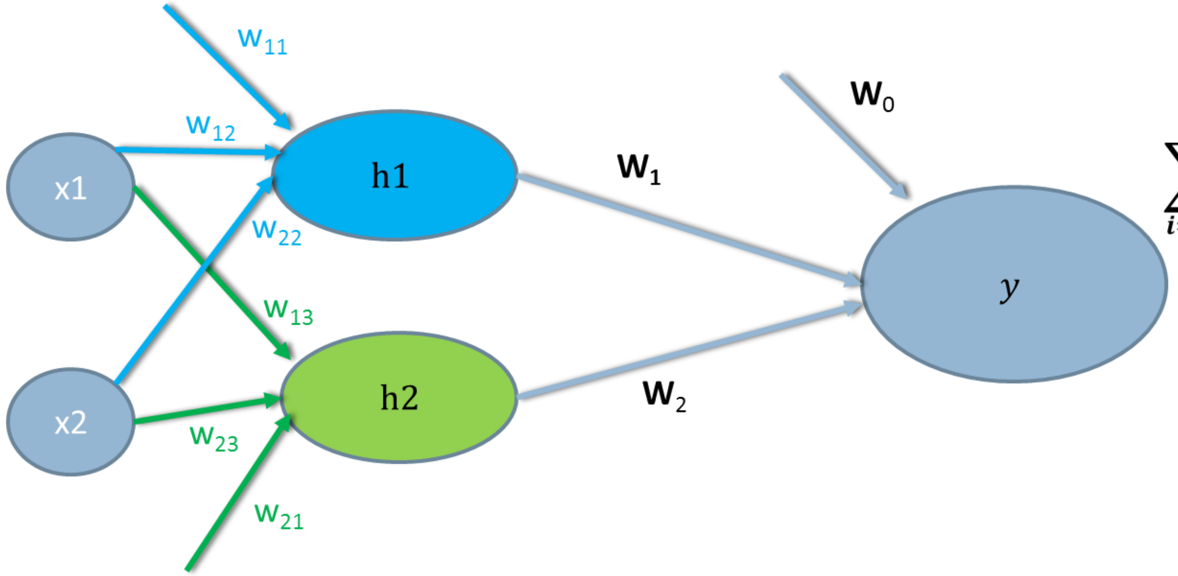
Conclusion

- 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 network algorithms are very calculation intensive. They require highly efficient computing machines. Large datasets take significant amount of runtime. 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

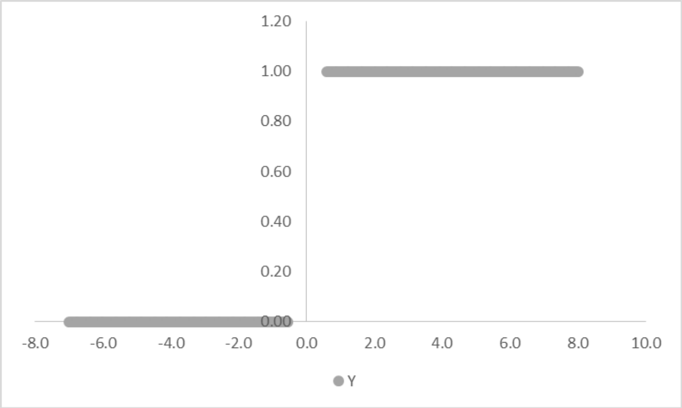
Math- How to update the weights?

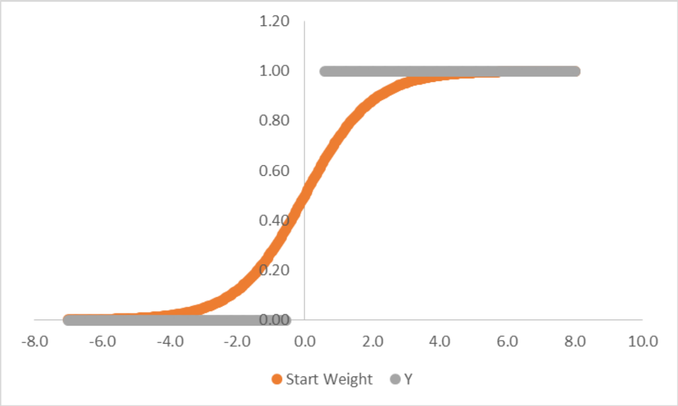
- $W_{jk} := W_{jk} + \Delta W_{jk}$
  - $where\ \Delta W_{jk} = \eta . y_j \delta_k$
  - η is the learning parameter
  - $\delta_k = y_k(1 - y_k) * Err$ (for hidden layers $\delta_k = y_k(1 - y_k) * w_j * Err$)
  - Err=Expected output-Actual output
- The weight corrections is calculated based on the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

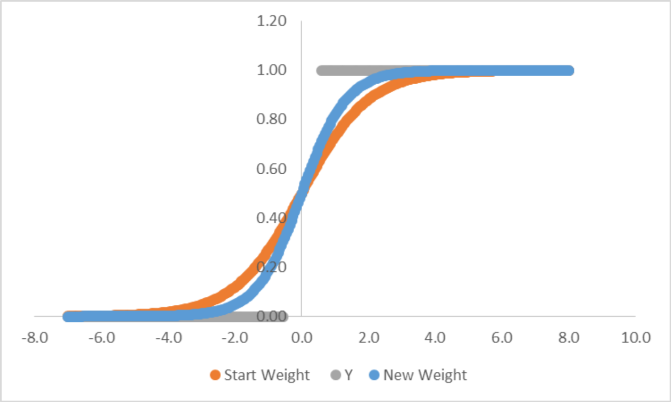
Math-How does the delta rule work?

How does the delta rule work?

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





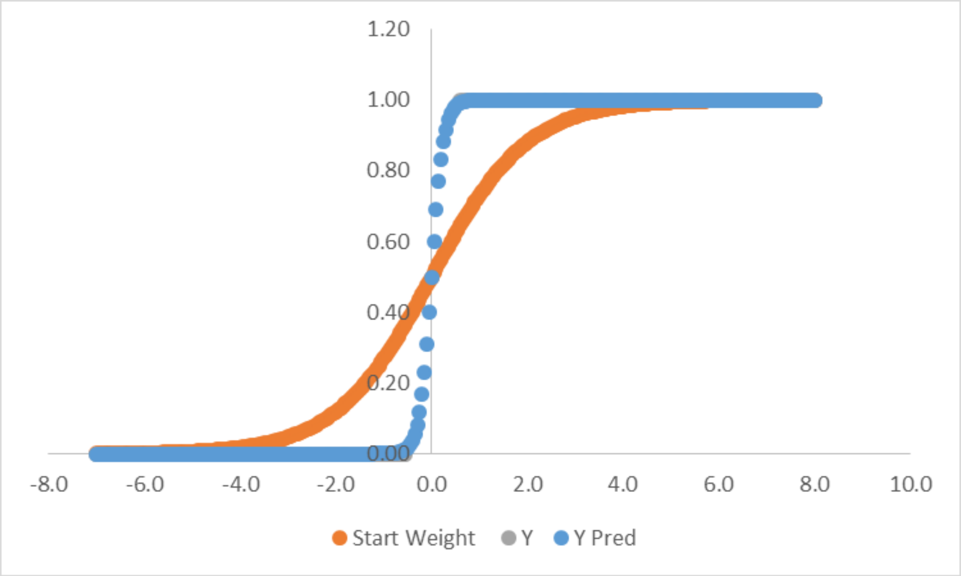


- If we building a simple logistic regression line. We would like to find the weights using weight update rule
- $Y=1/(1+e^{-wx})$ is the equation
- We are searching for the optimal w for our data

- 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?



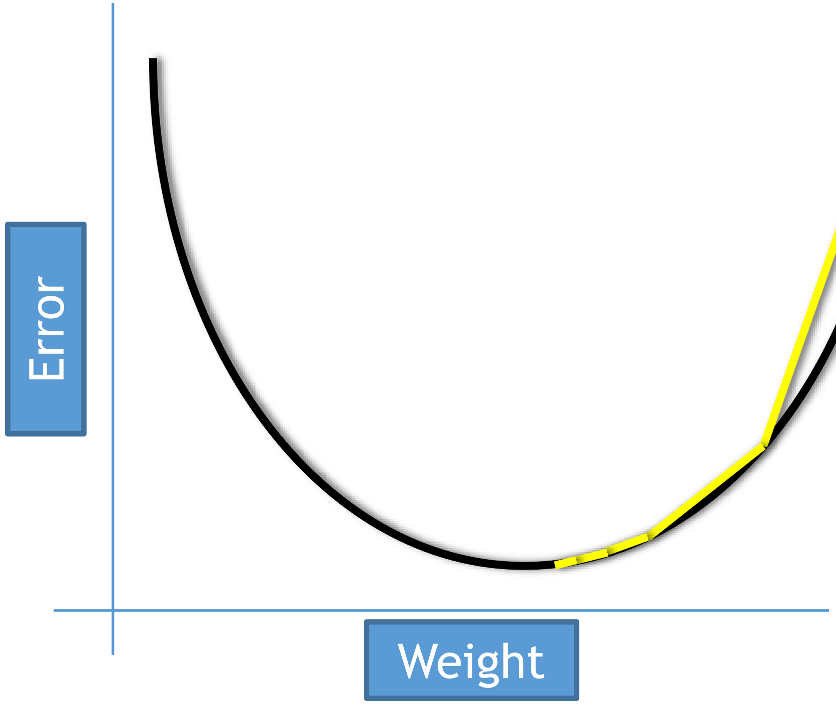
- If we repeat the same process of adding delta and updating weights, we can finally end up with minimum error
- The weight at that final step is the optimal weight
- In this example the weight is 8, and the error is 0
- $Y=1/(1+e^{-8x})$ is the final equation

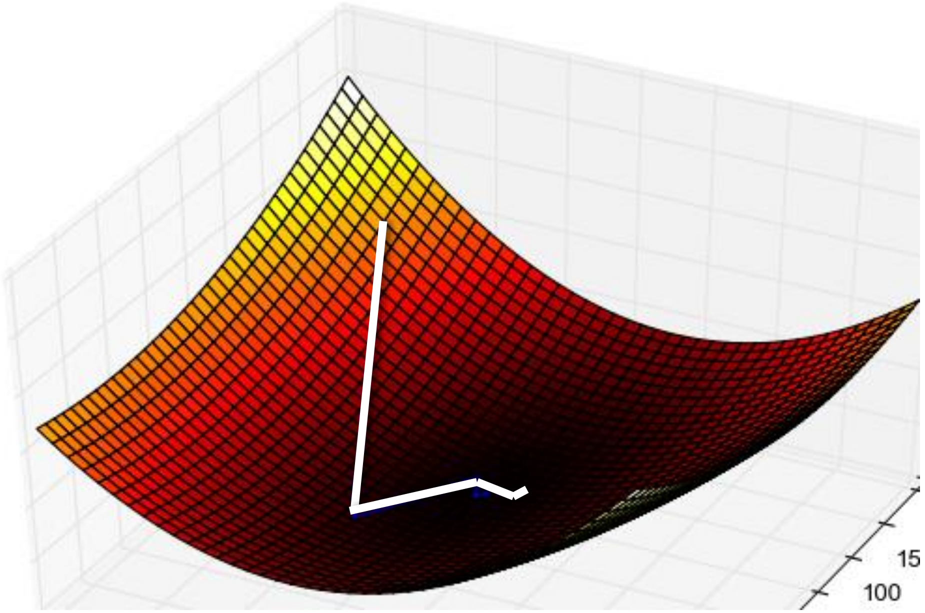
- In this example, we manually changed the weights to reduce the error. This is just for intuition, manual updating is not feasible for complex optimization problems.
- In gradient descent is a scientific optimization method. We update the weights by calculating gradient of the function.

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).

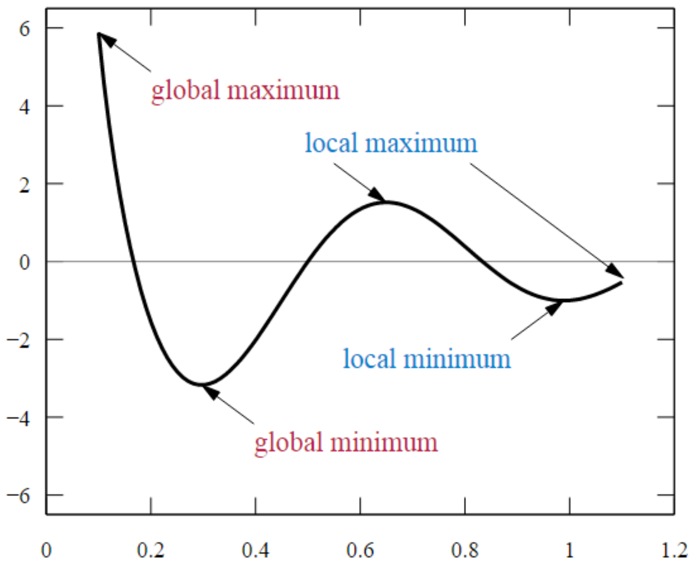


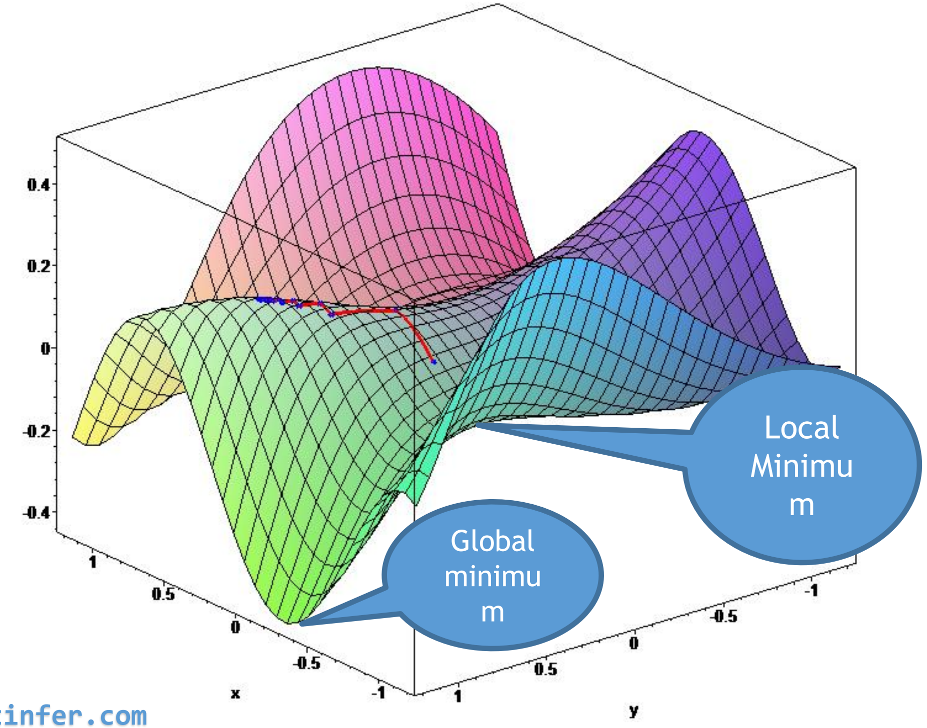


Local vs. Global Minimum

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.

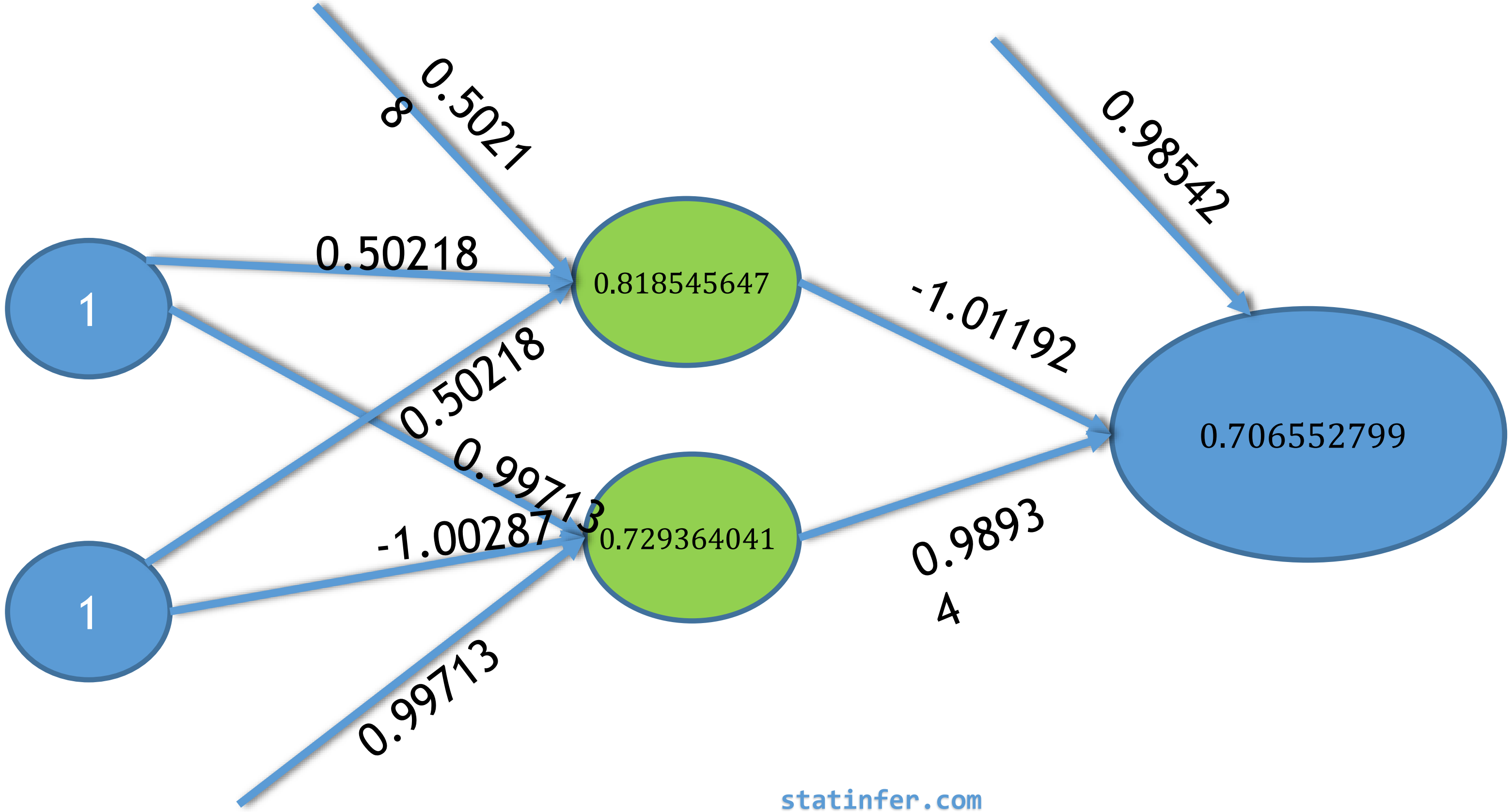




Demo-How does gradient descent work?

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

References & Image Sources

- "ROC curve" by Masato8686819 - Own work. Licensed under CC BY-SA 3.0 via Wikimedia Commons - https://commons.wikimedia.org/wiki/File:ROC_curve.svg#/media/File:ROC_curve.svg
- “Curvas”作者UPO649 1112 prodgom - 自己的作品。来自维基共享资源 - https://commons.wikimedia.org/wiki/File:Curvas.png#/media/File:Curvas.png根据CC BY-SA 3.0授权
- http://www.autonlab.org/tutorials/neural.html
- "Gradient ascent (surface)". Licensed under Public Domain via Commons - https://commons.wikimedia.org/wiki/File:Gradient_ascent_(surface).png#/media/File:Gradient_ascent_(surface).png
- "Gradient descent method" by Роман Сузи - Сгенерирован программой, повернут вручную. Licensed under CC BY-SA 3.0 via Wikimedia Commons - https://commons.wikimedia.org/wiki/File:Gradient_descent_method.png#/media/File:Gradient_descent_method.png
- Lecture 7 :Artificial neural networks: Supervised learning: Negnevitsky, Person Education 2005
- Gradient descent can find the local minimum instead of the global minimum By I, KSmrq
- "Neuron". Licensed under CC BY-SA 3.0 via Wikimedia Commons - https://commons.wikimedia.org/wiki/File:Neuron.svg#/media/File:Neuron.svg
- "Neural signaling-human brain" by 7mike5000 - Gif created from Inside the Brain: Unraveling the Mystery of Alzheimer's Disease, an educational film by the National Institute on Aging.. Licensed under CC BY-SA 3.0 via Wikimedia Commons - https://commons.wikimedia.org/wiki/File:Neural_signaling-human_brain.gif#/media/File:Neural_signaling-human_brain.gif