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

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

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

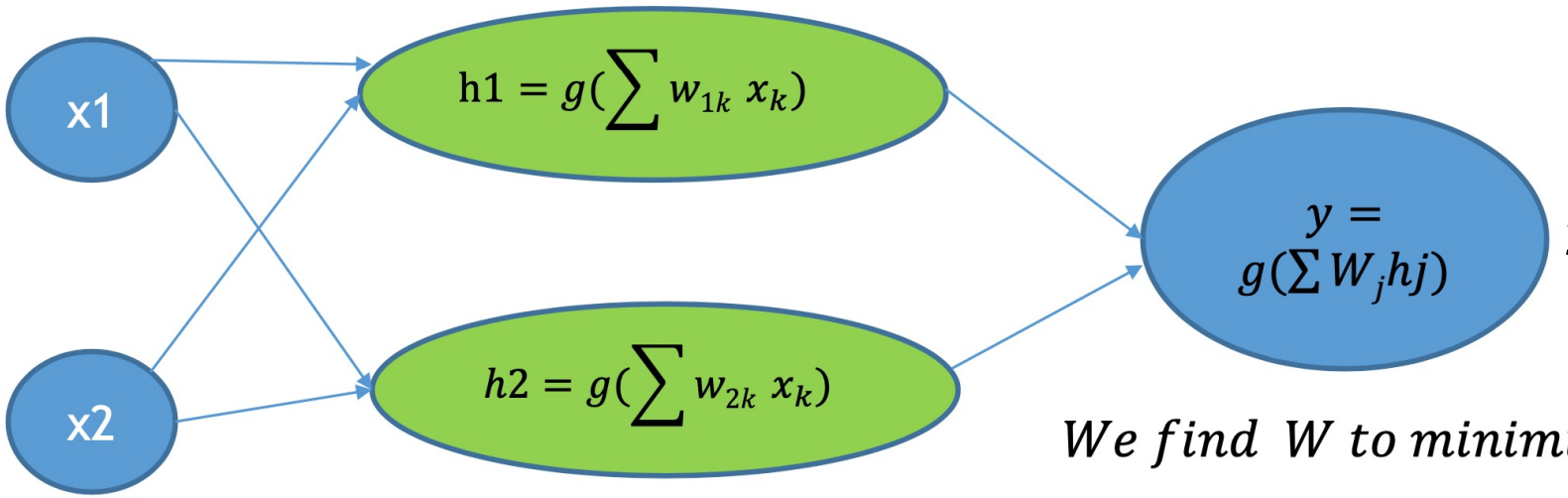
Neural networks is all about finding the sets of weights {Wj,} and $\{w_{jk}\}$ using **Gradient Descent Method**

Neural Network intuition

*Intermediate output*1

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

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



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 output*2

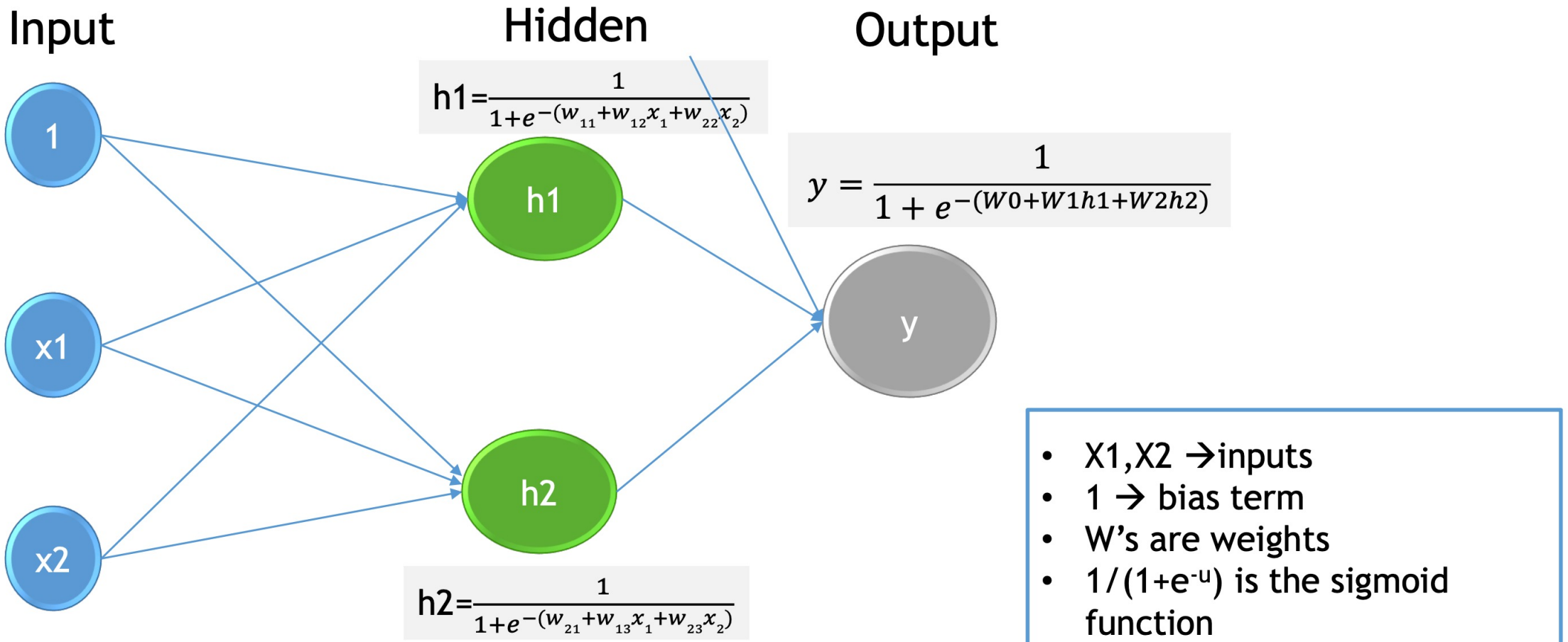
$h2 = out(x) = g(\sum w_{2k}\, x_k)$

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

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



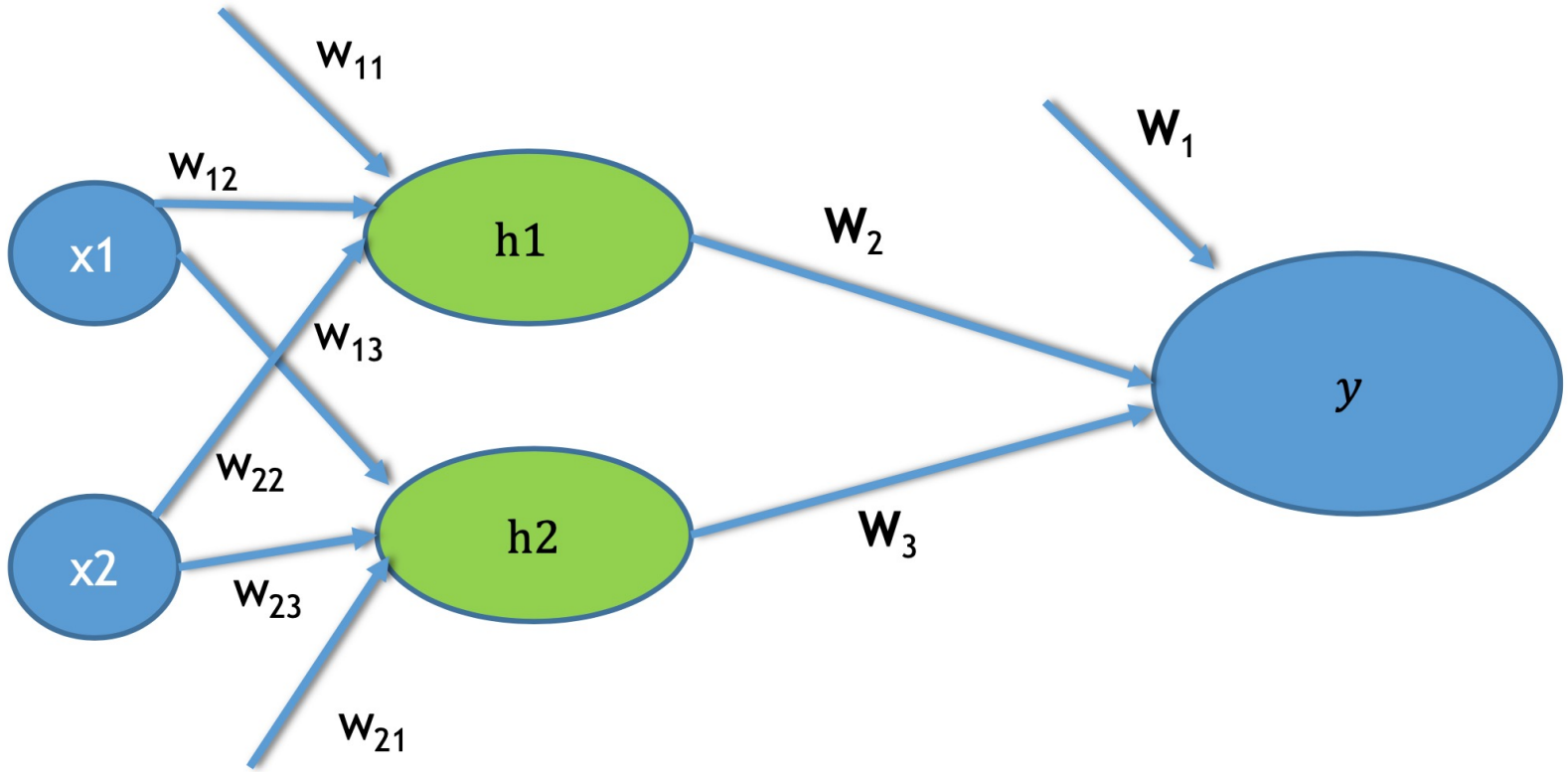
- X1,X2 →inputs
- 1 → bias term
- W's are weights
- $1/(1+e^{-u})$ is the sigmoid function
- Y is output

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

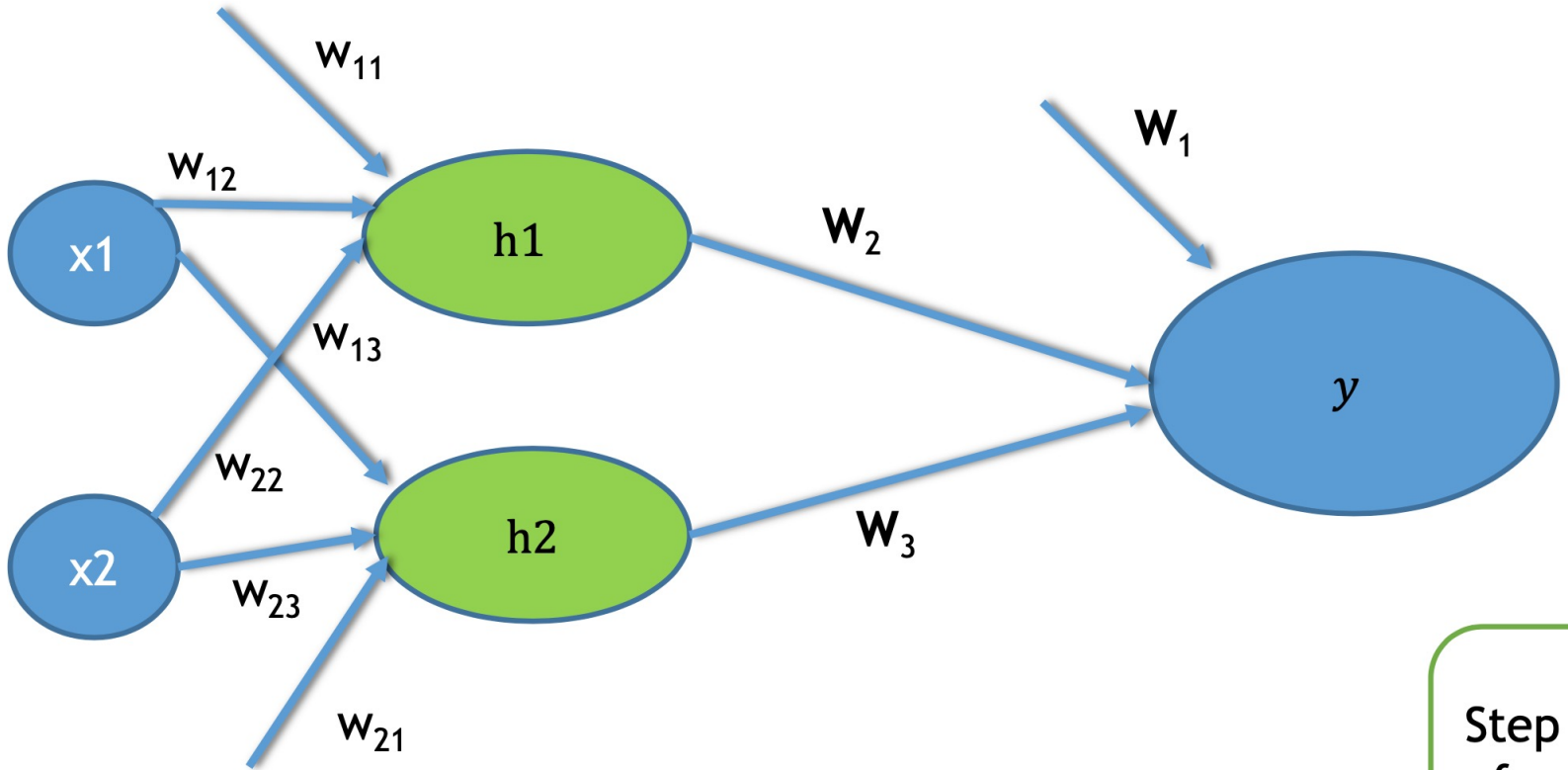


- Algorithm is all about finding the weights/coefficients
- We randomly initialize some weights; Calculate the output by supplying training input; If there is an error the weights are adjusted to reduce this error.

The Neural Network Algorithm

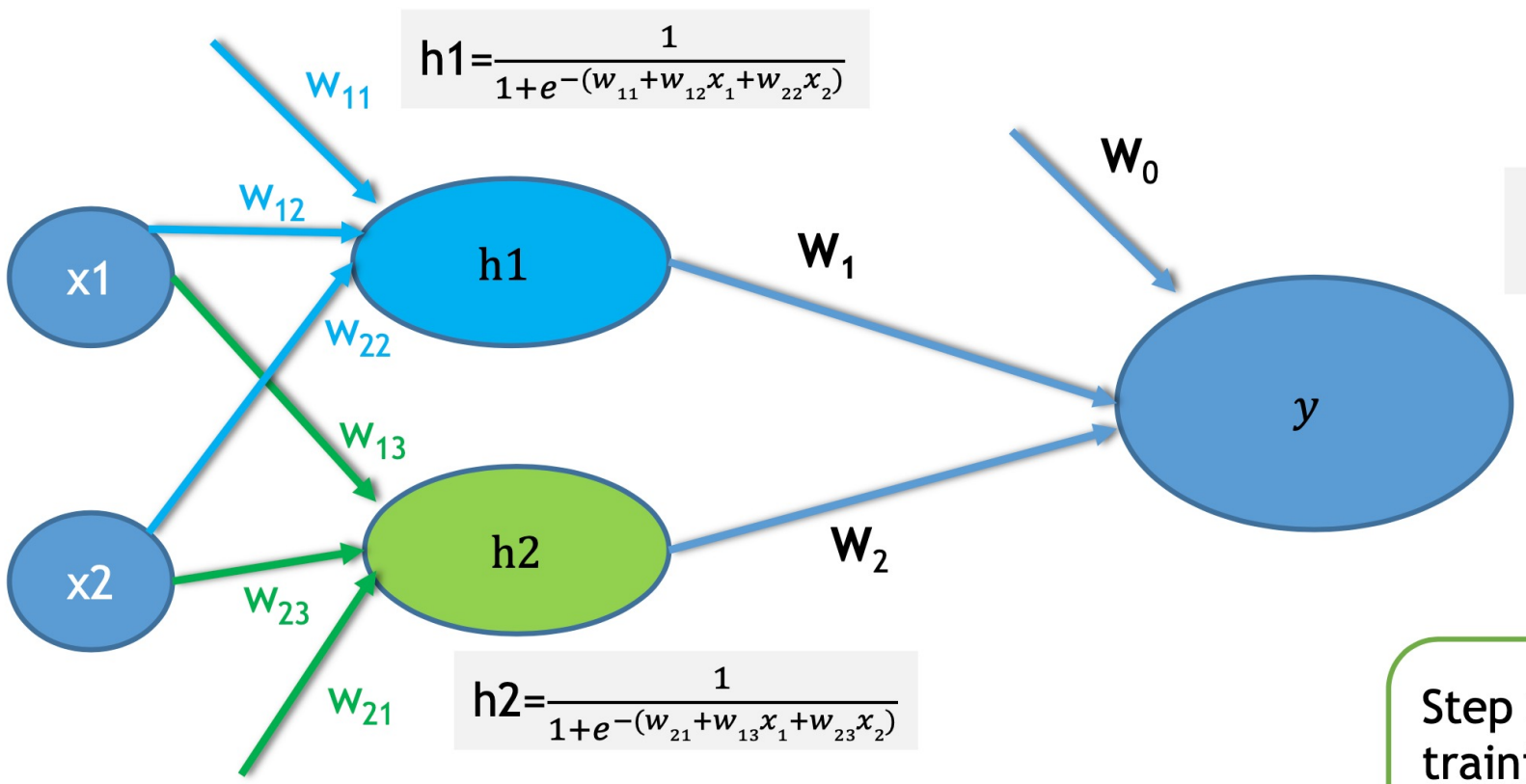
- **Step 1: Initialization of weights**: Randomly select some weights
- **Step 2 : Training & Activation**: Input the training values and perform the calculations forward.
- **Step 3 : Error(cost) Calculation and back propagation:** Calculate the error at the outputs. Use the output error to calculate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

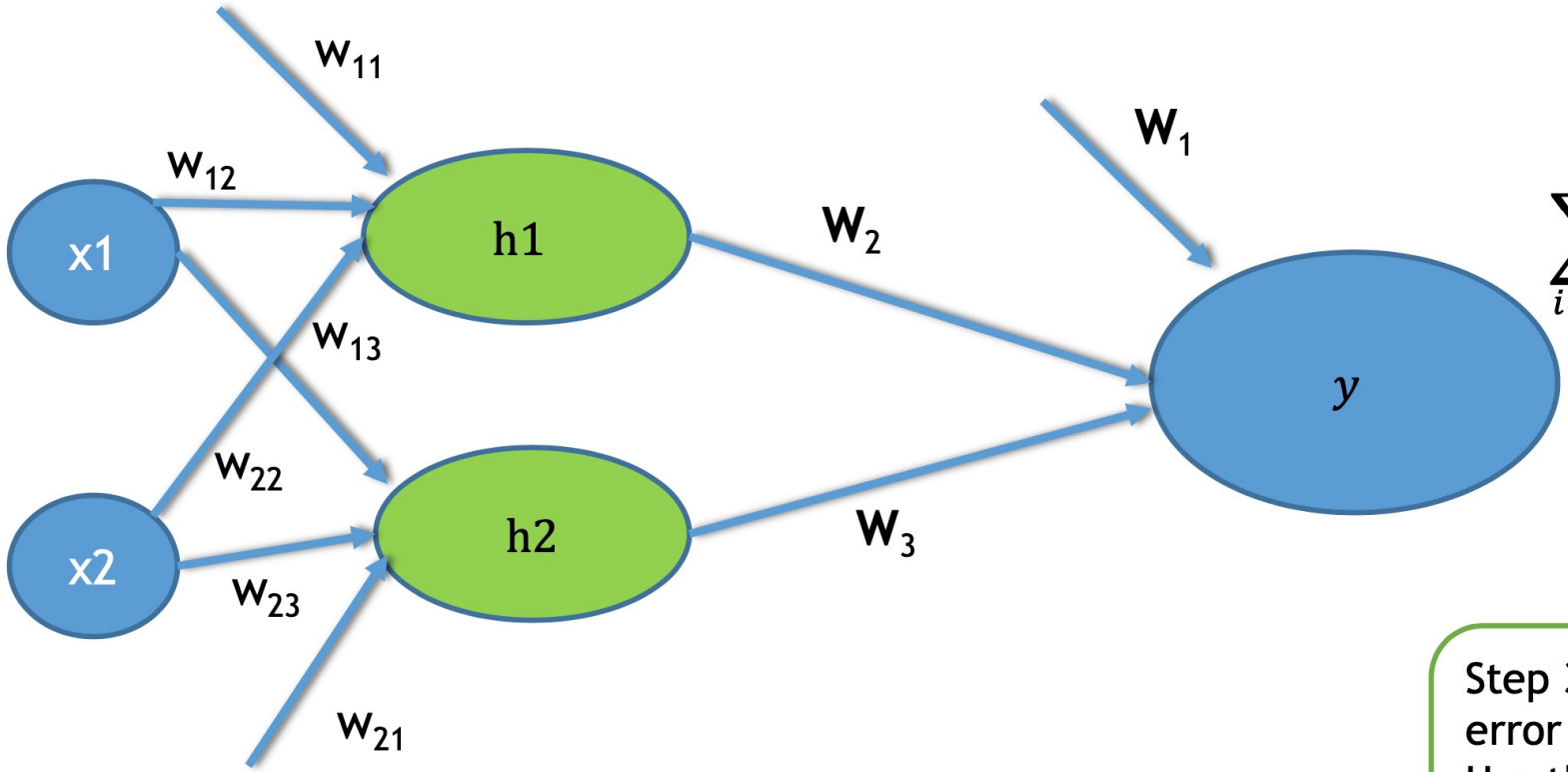


$$y = \frac{1}{1 + e^{-(W0+W1h1+W2h2)}}$$

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

Training input & calculations - **Feed Forward**

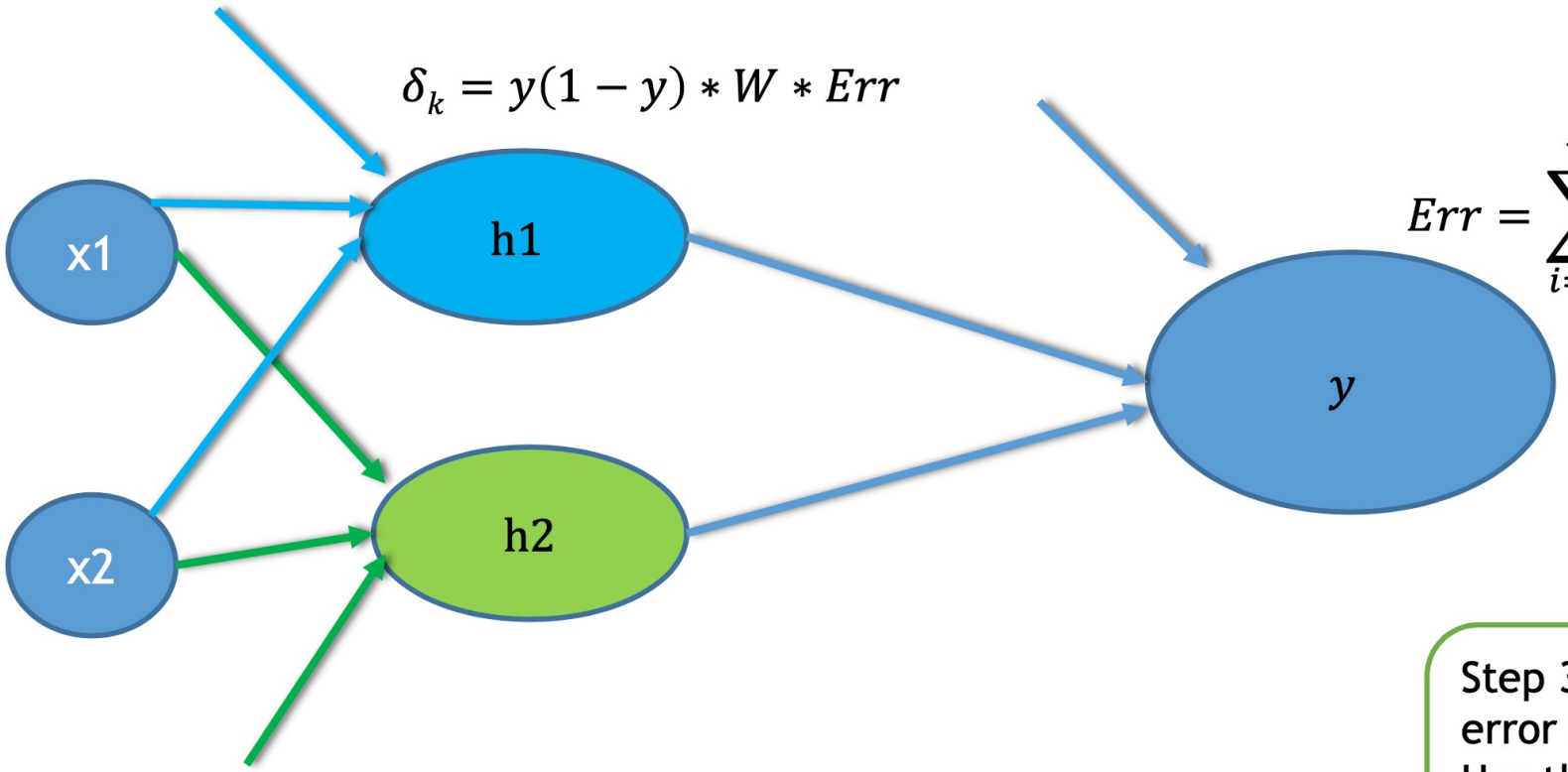
Error Calculation at Output



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

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

Error Calculation and backpropagation

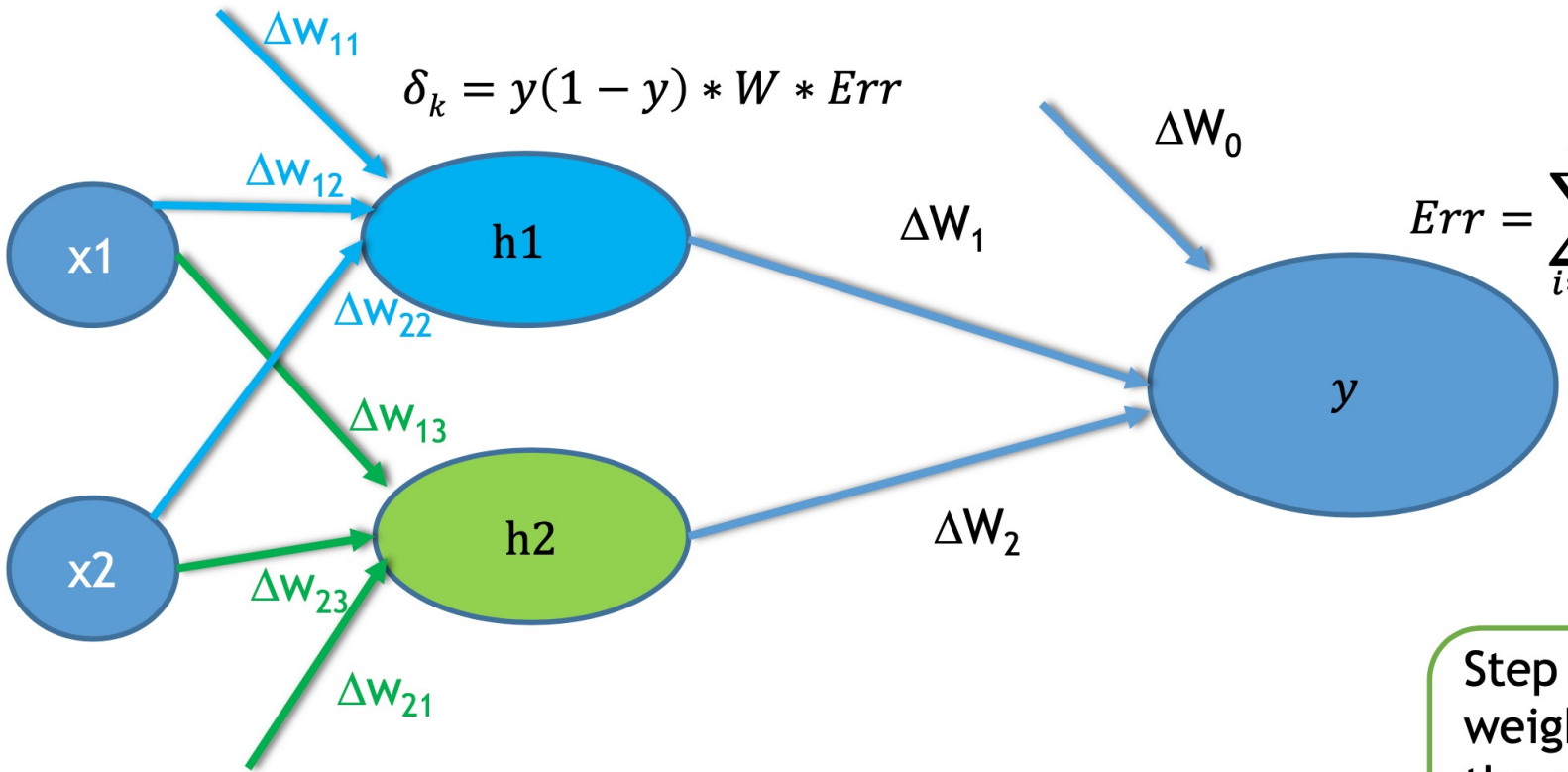


$$Err = \sum_{i=1}^{n} \left[y_i - g\left(\sum_{k=1}^{m} w_k h_{ki} \right) \right]^2$$

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

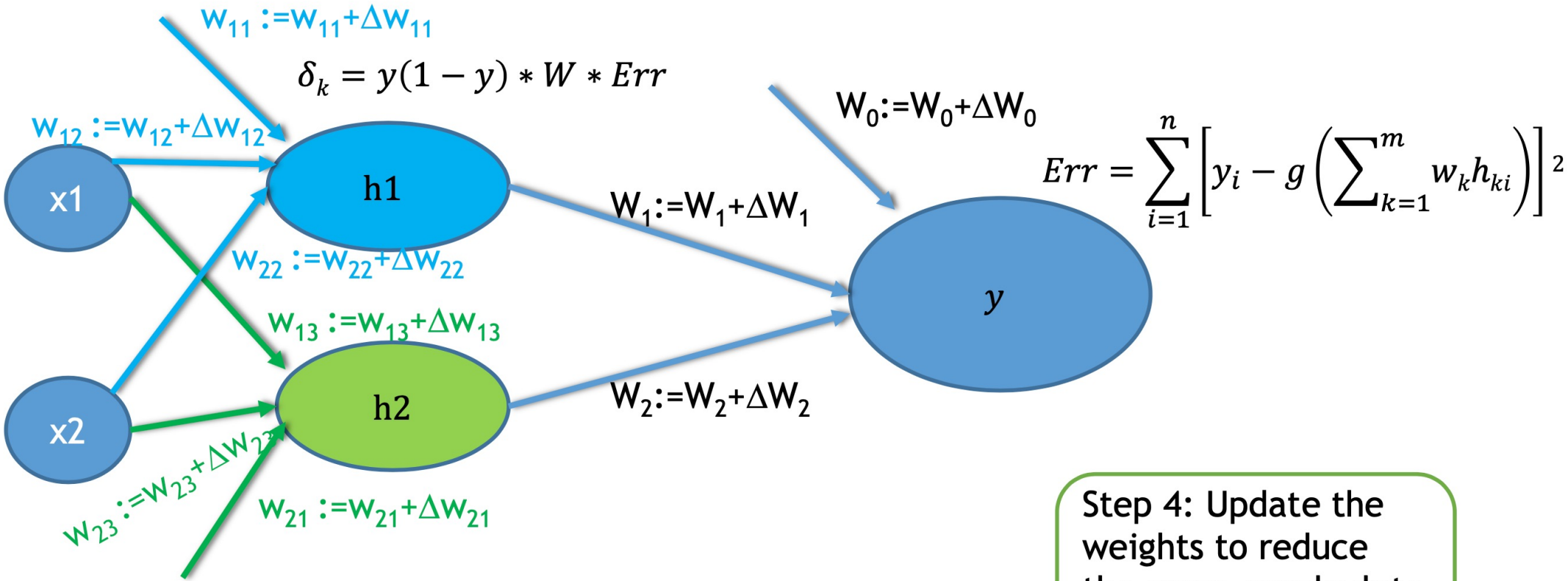


$$\delta_k = y(1-y) * W * Err$$

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

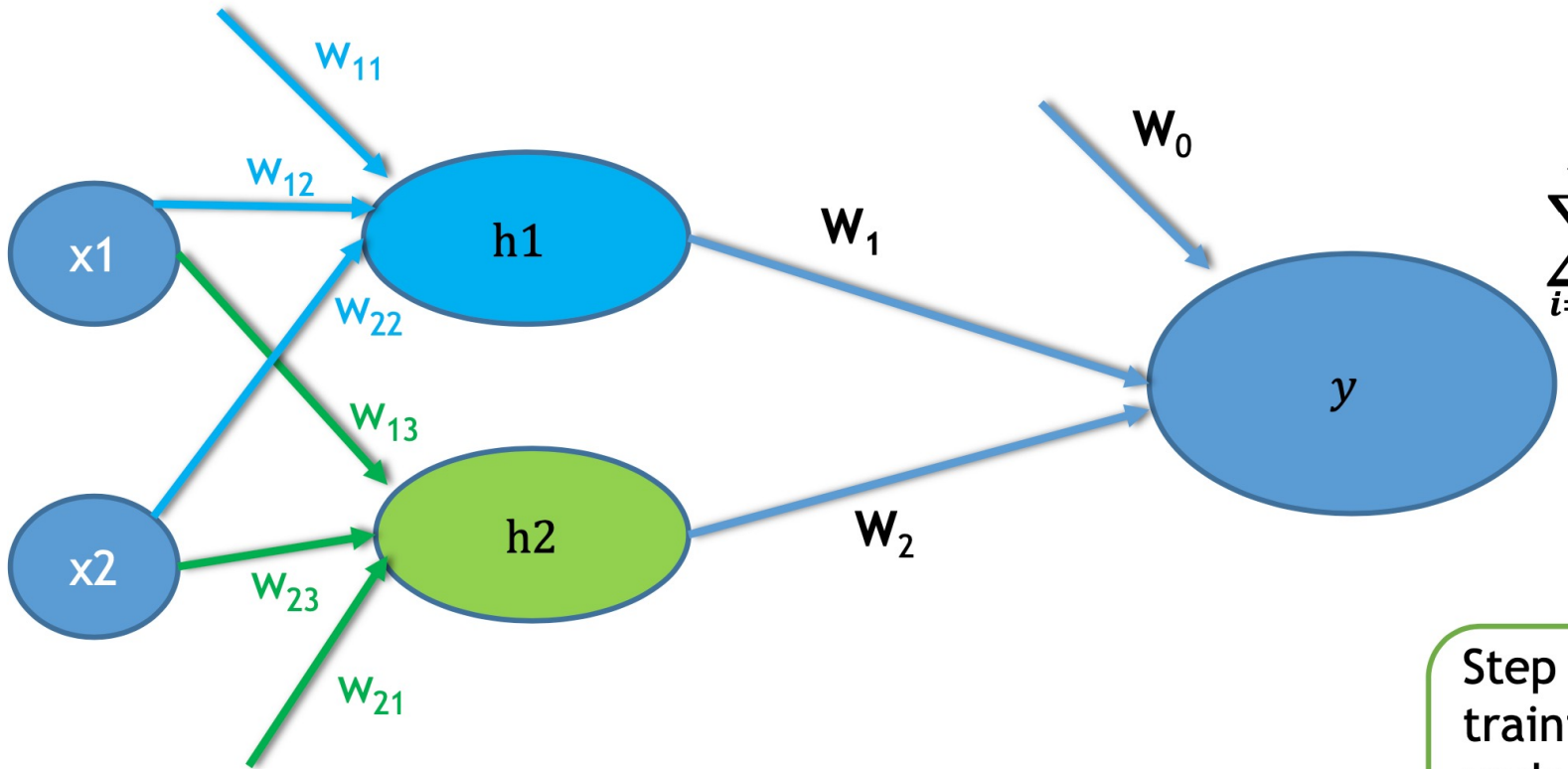
Step 4: Update the weights to reduce the error, recalculate and repeat the process

Update Weights



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

Stopping Criteria



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

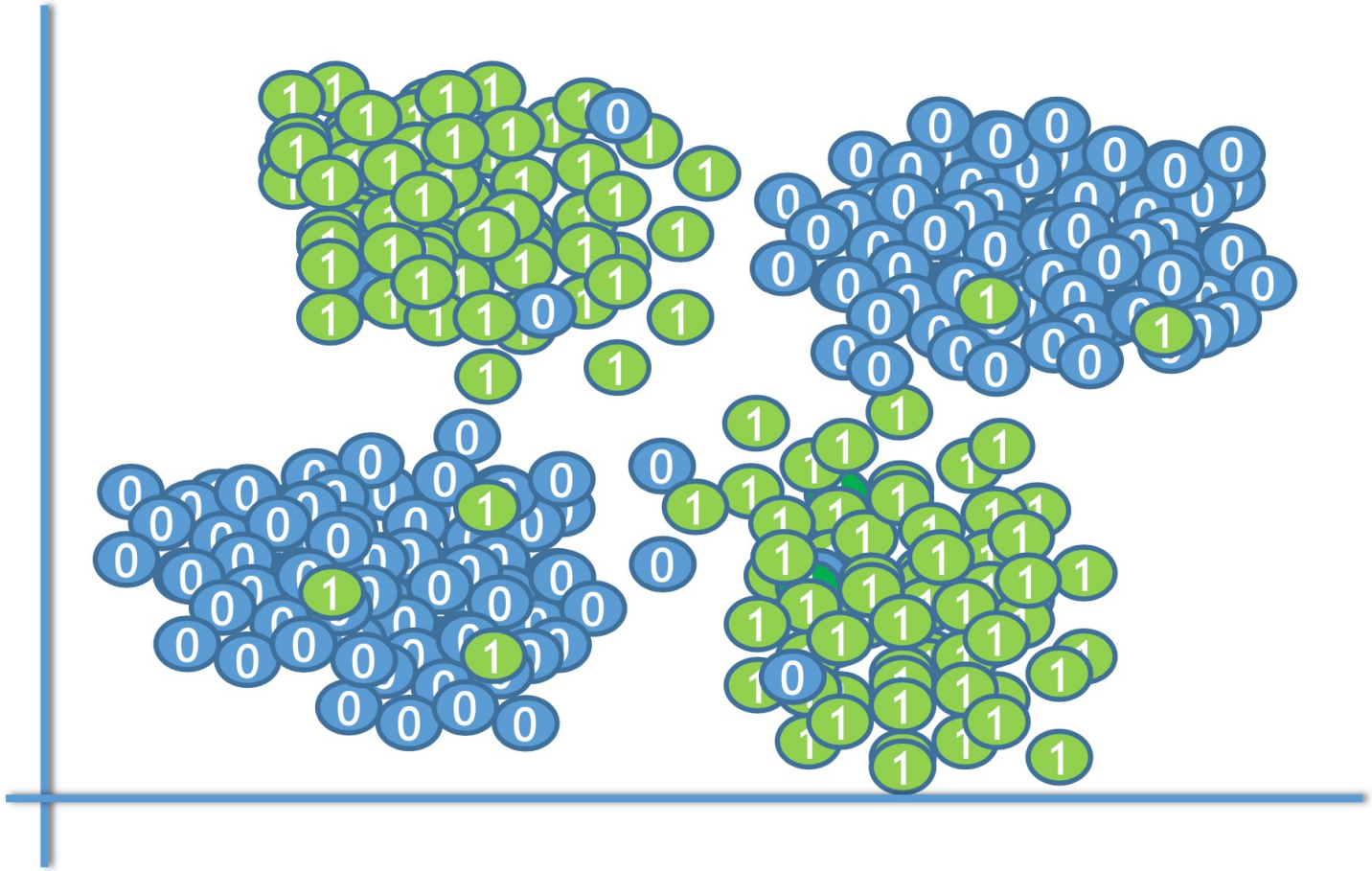
Step 5: Stop the training and weights updating process when the minimum error criteria is met

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

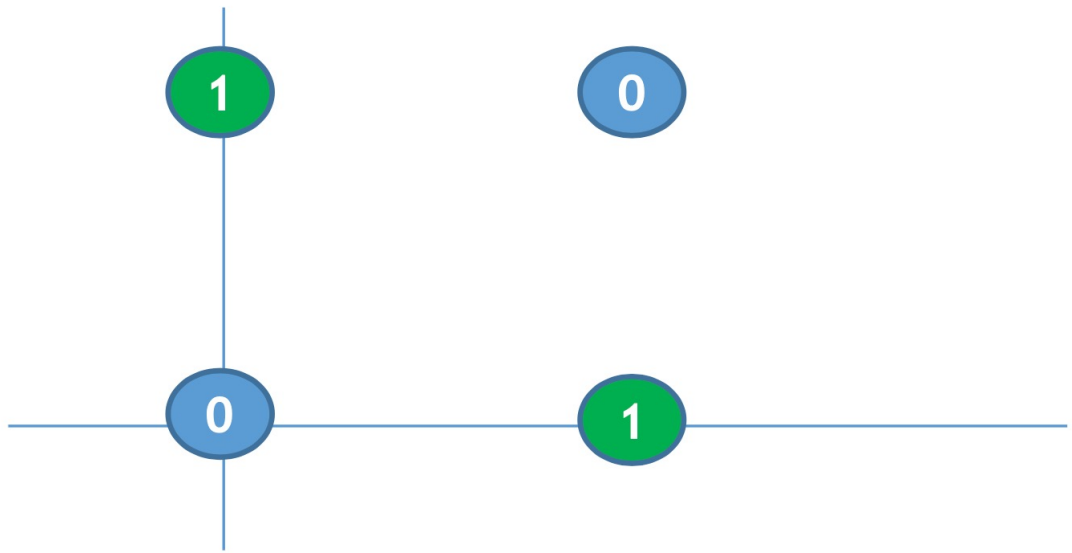


Looks like a dataset that can’t be separated by using single linear decision boundary/perceptron

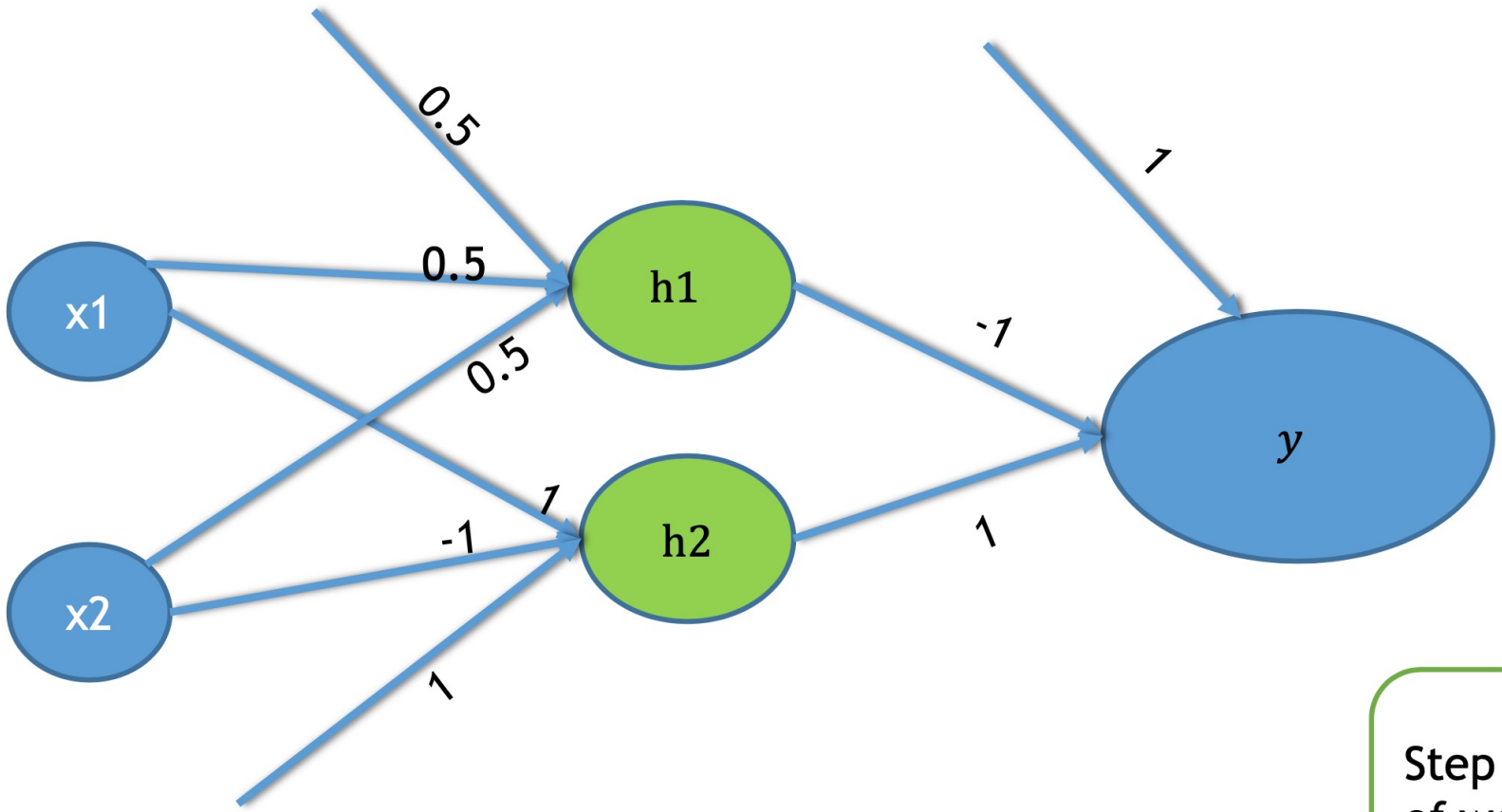
Neural network Algorithm-Demo

- 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



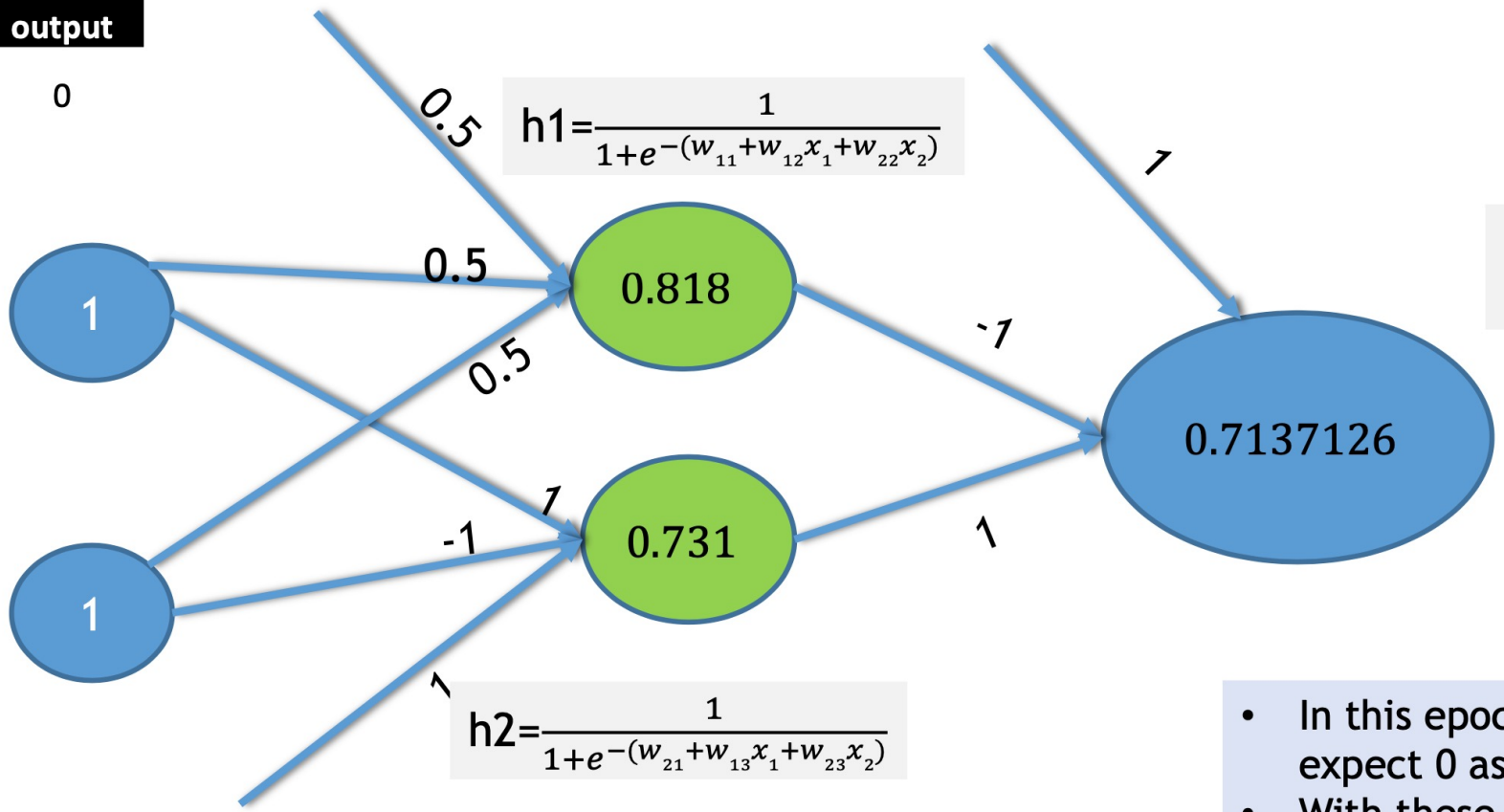
Randomly initialize weights



Step 1: Initialization of weights: Randomly select some weights

Activation

input1	input2	output
1	1	0

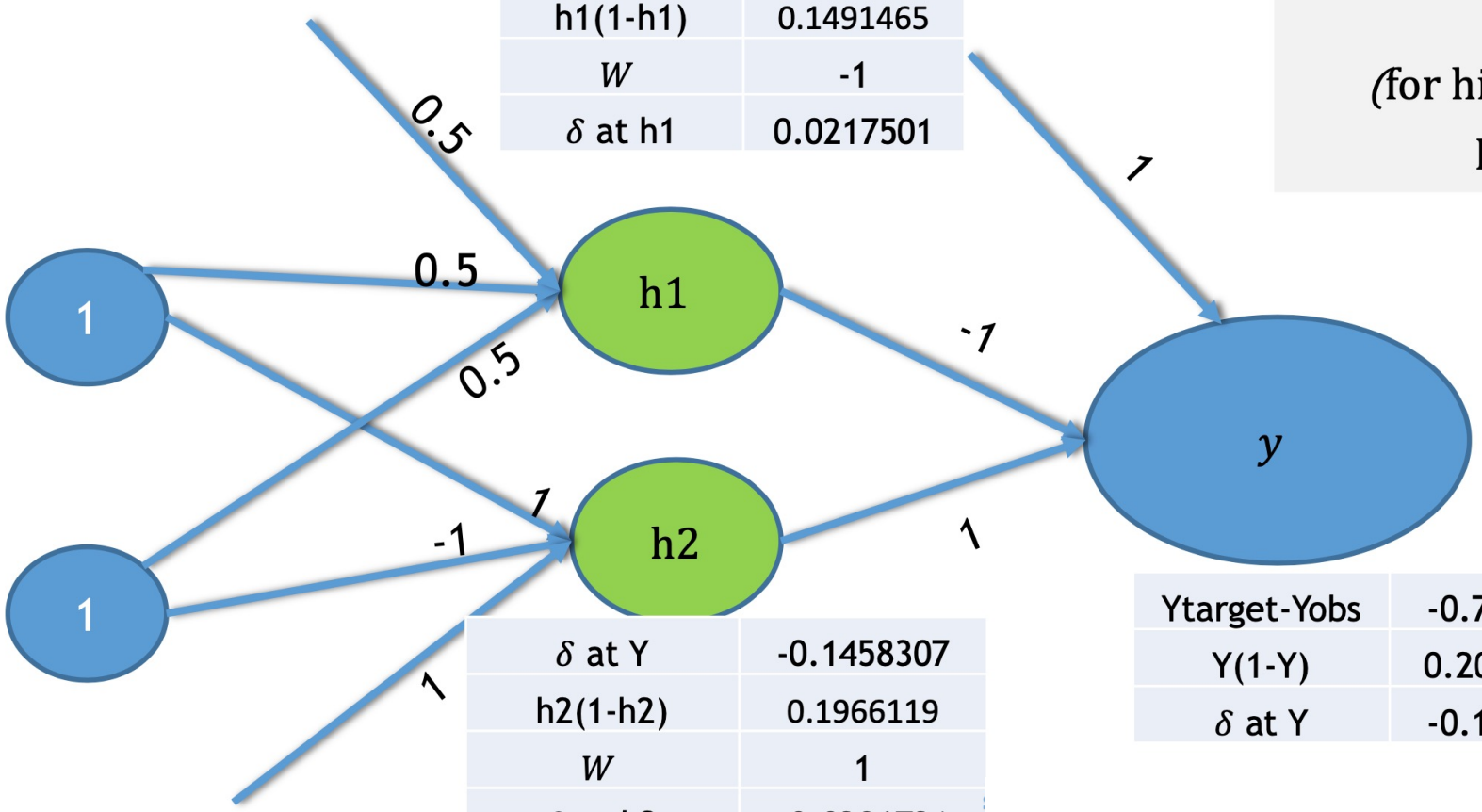


$$y = \frac{1}{1 + e^{-(W0+W1h1+W2h2)}}$$

- 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
$$\delta_k = y_k(1 - y_k) * Err$$
(for hidden layers $\delta_k = y_k(1 - y_k) * w_j * Err$)
Err=Expected output-Actual output

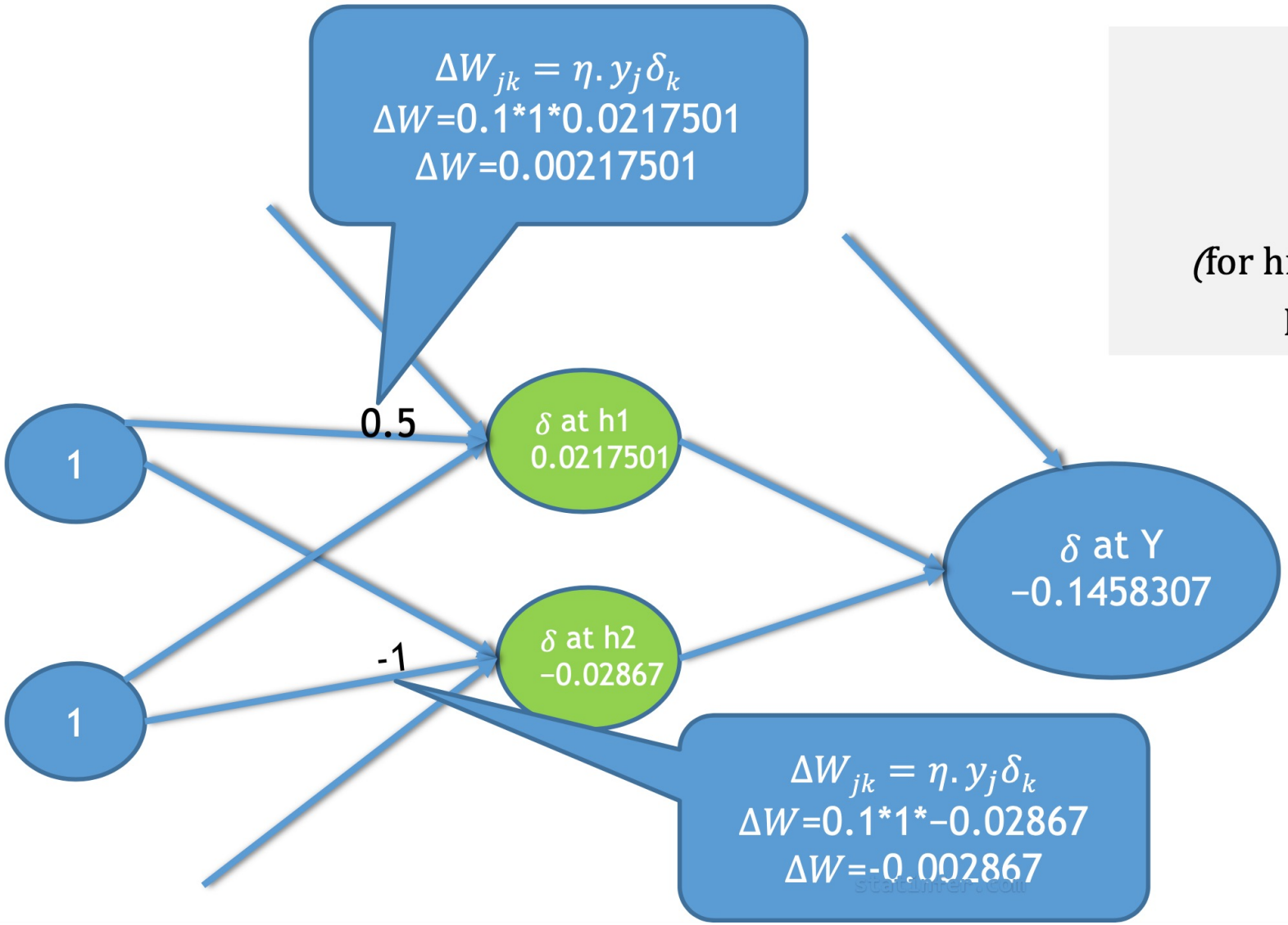


δ at Y	-0.1458307
h1(1-h1)	0.1491465
W	-1
δ at h1	0.0217501

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



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

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

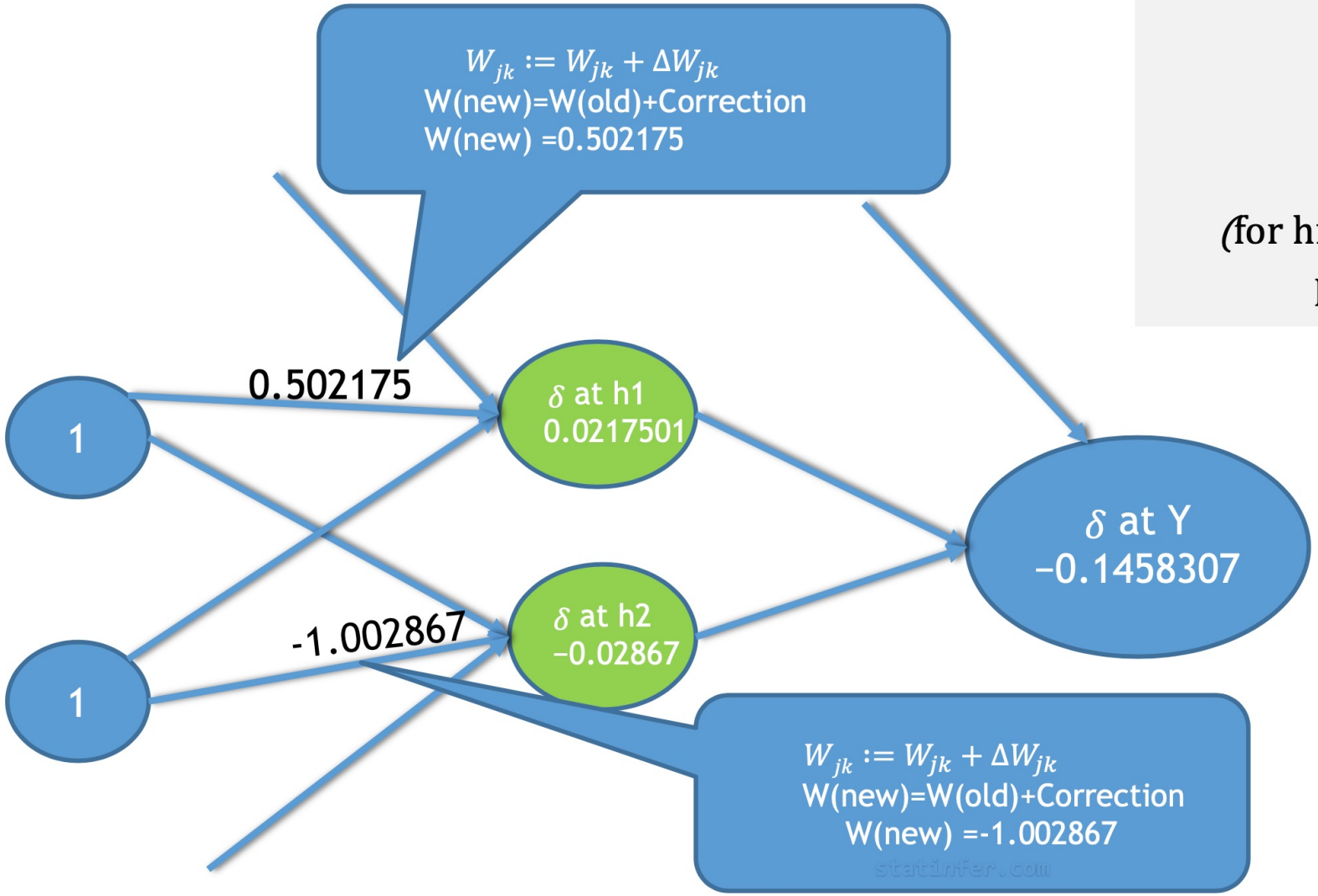
η is the learning parameter

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

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

Err=Expected output-Actual output

Updated Weights



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

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

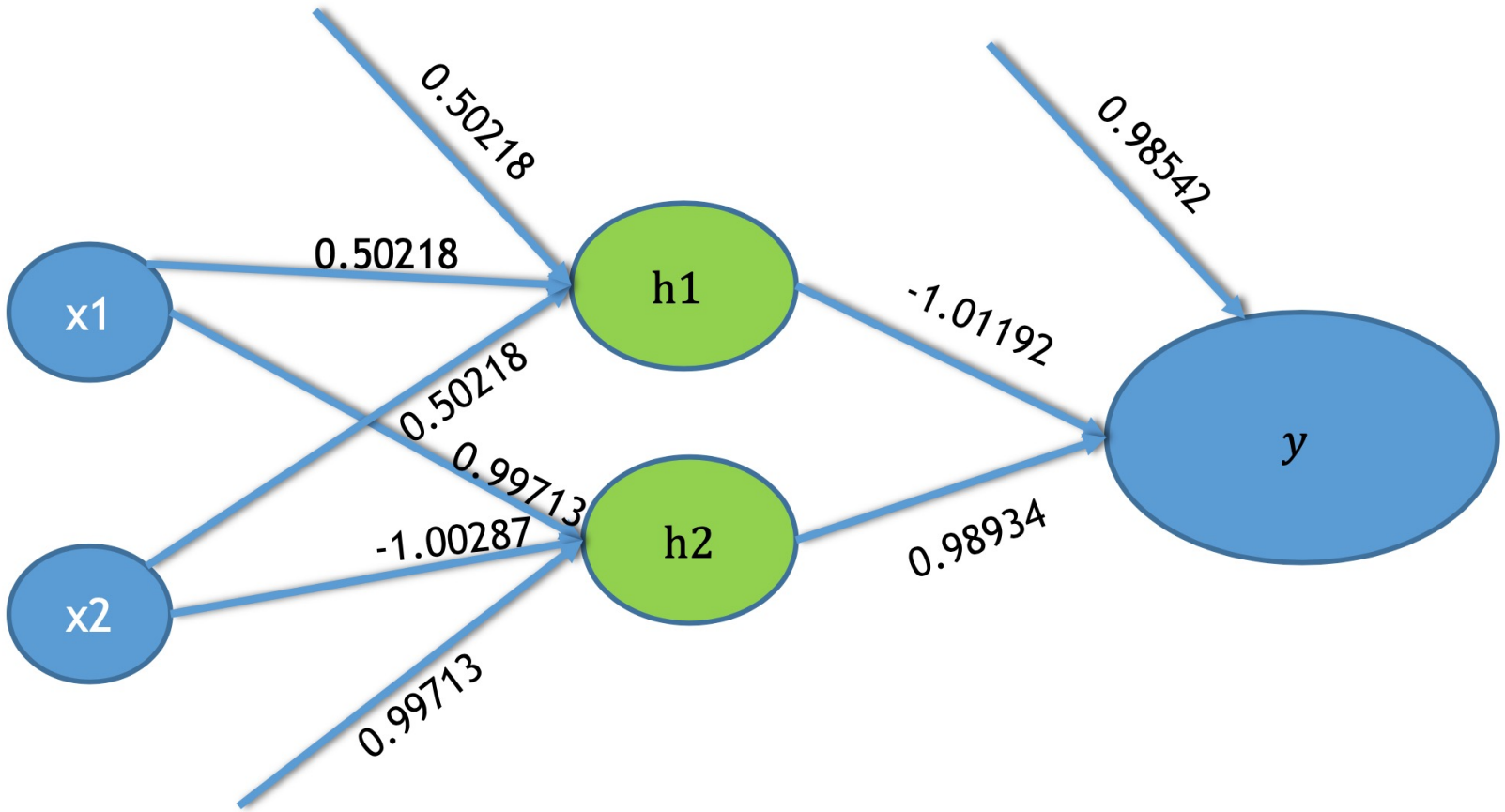
η is the learning parameter

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

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

Err=Expected output-Actual output

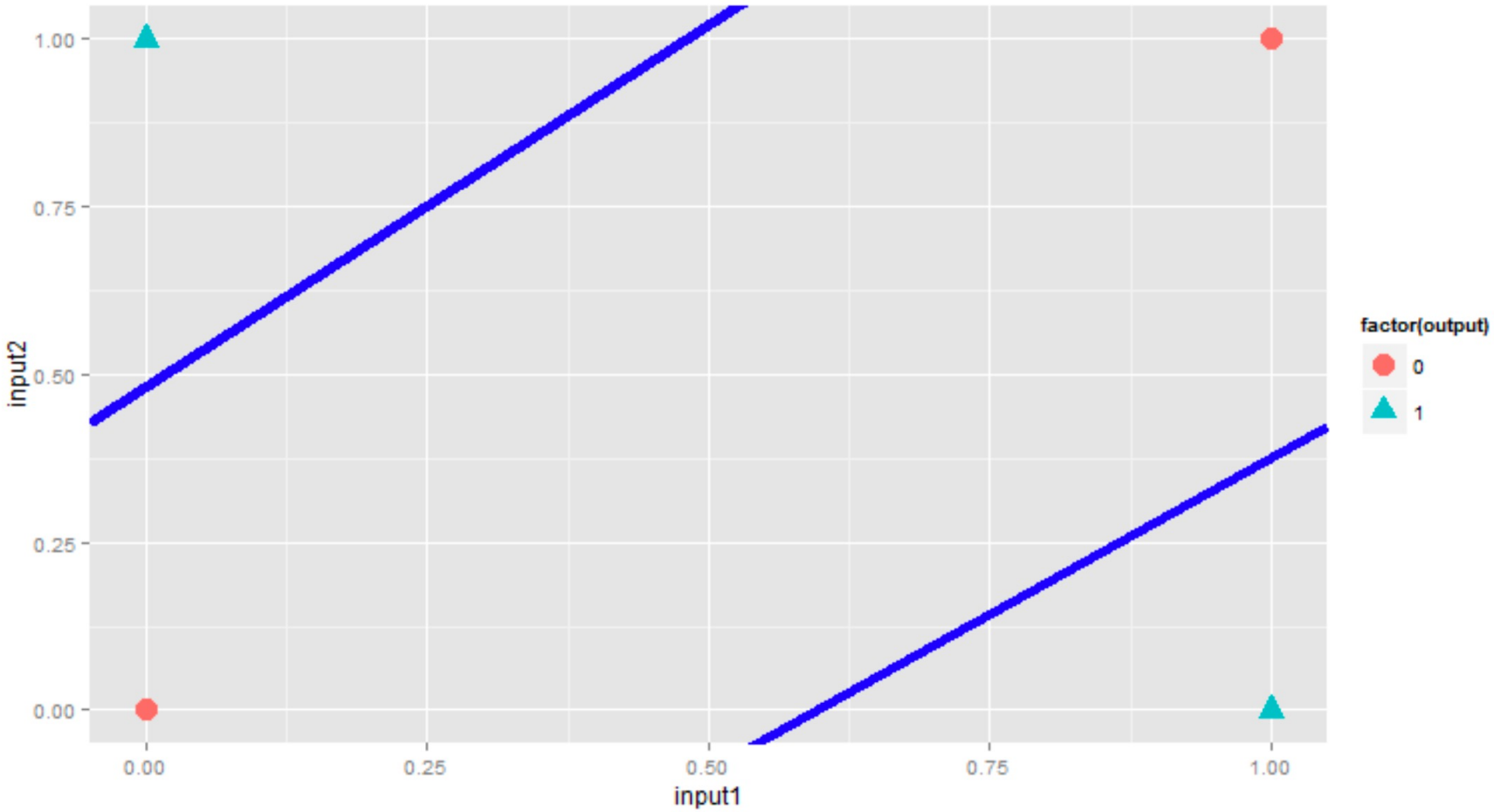
Updated Weights..contd



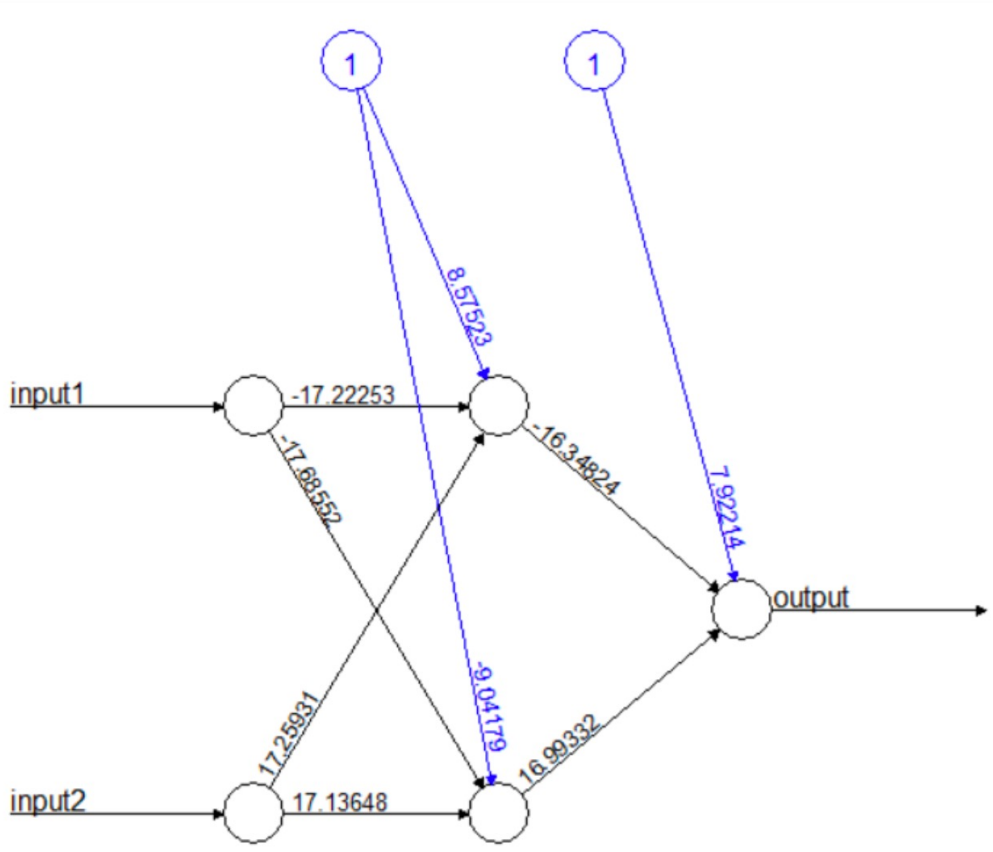
Iterations and Stopping Criteria

- 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



Error: 0 Steps: 178

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

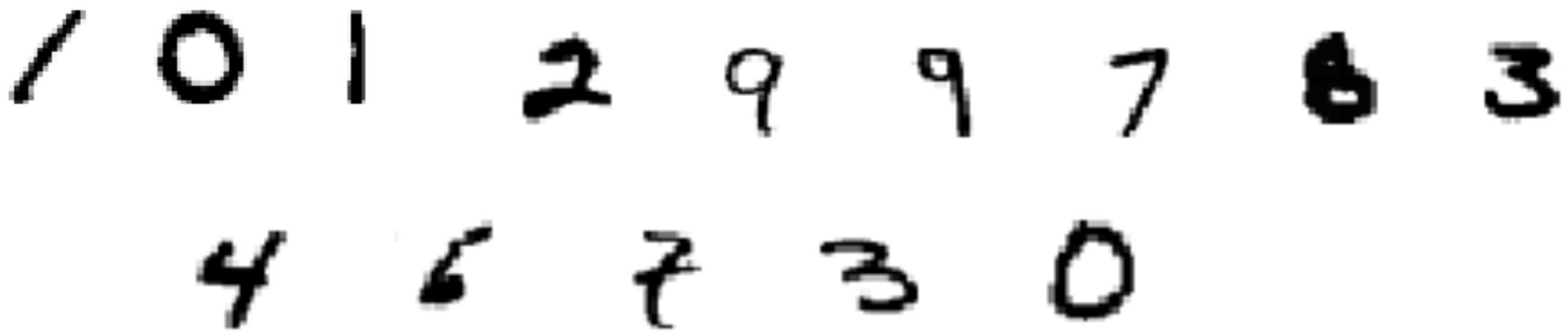
- We need to mention the dataset, input, output & number of hidden layers as input.
- Neural network calculations are very complex. The algorithm may take sometime to produce the results
- One need to be careful while setting the parameters. The runtime changed based on the input parameter values

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

Humans see this

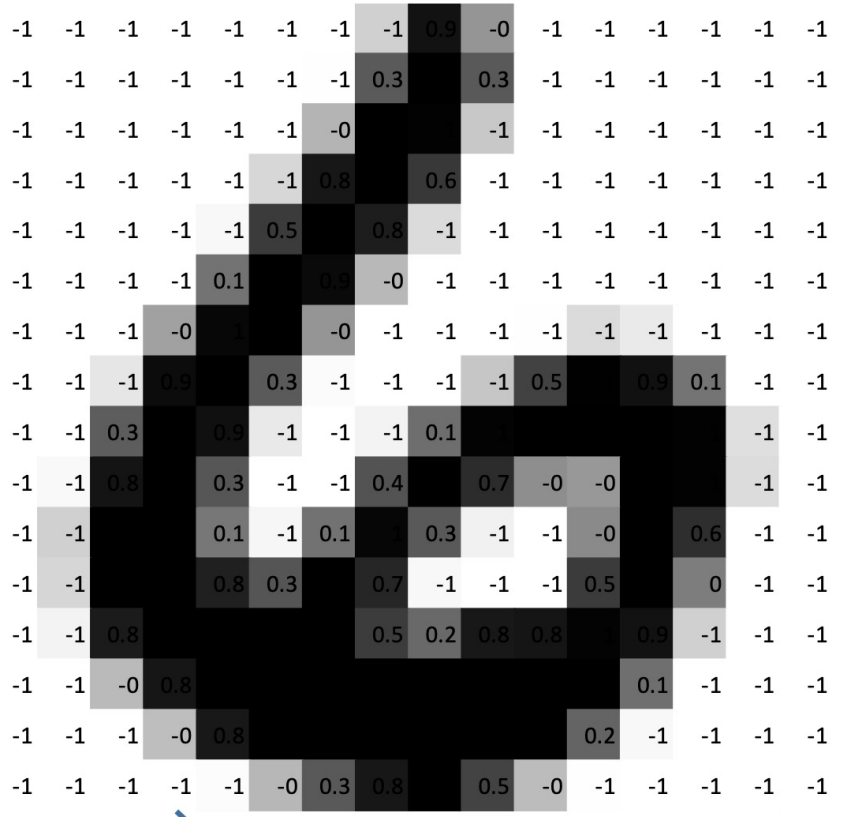
-1	-1	-1	-1	-1	-1	-1	-1	0.9	-0	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	0.3	1	0.3	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-0	1	1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	0.8	1	0.6	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	0.5	1	0.8	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	0.1	1	0.9	-0	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-0	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

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

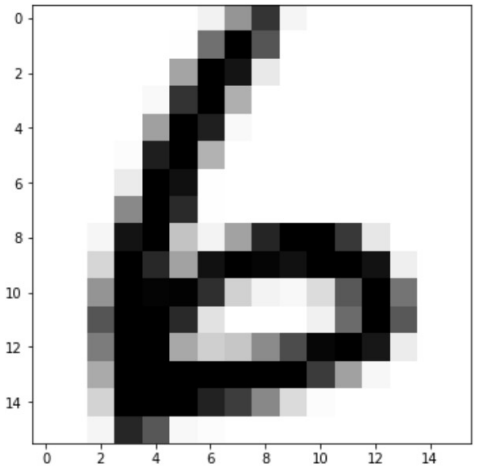
Converting image into numbers

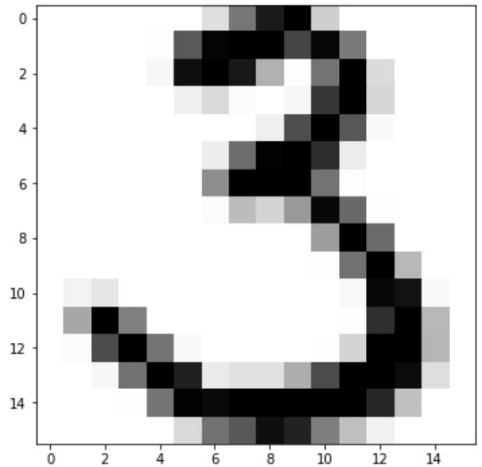
```
import matplotlib.pyplot as plt

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

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

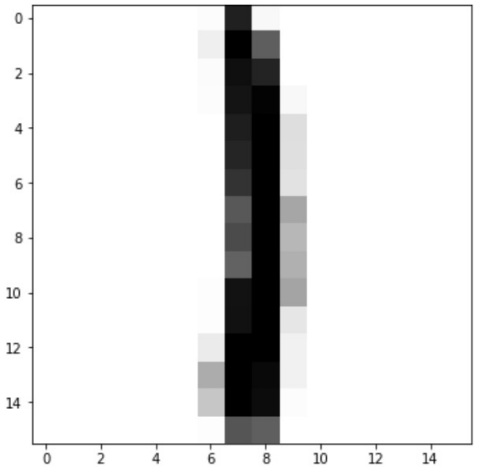
Code: Digit Recognizer

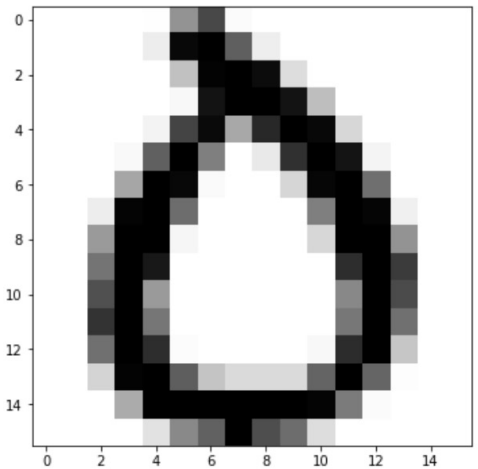
```
#Importing test and training data
import numpy as np
digits_train = np.loadtxt("zip.train.txt")


#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("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
```

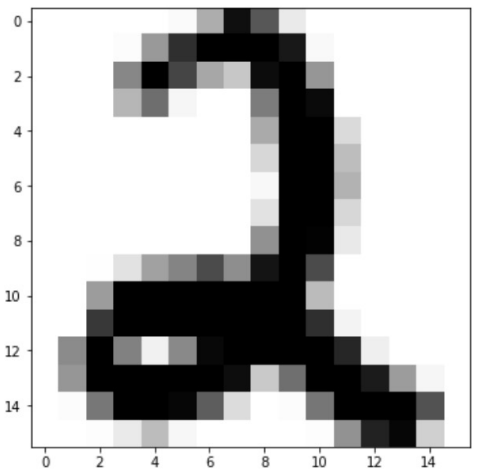
Code: Digit Recognizer

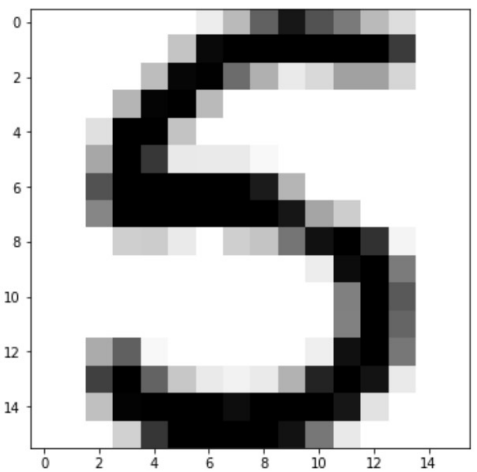












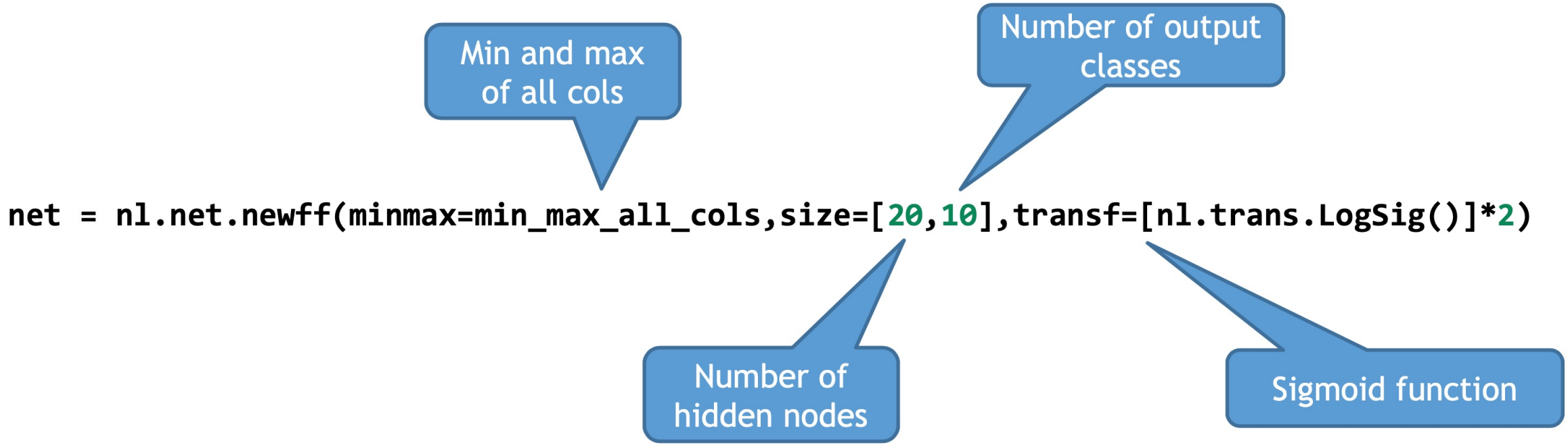

x1
x2
x256
y0
y1
y2
y9

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)
```

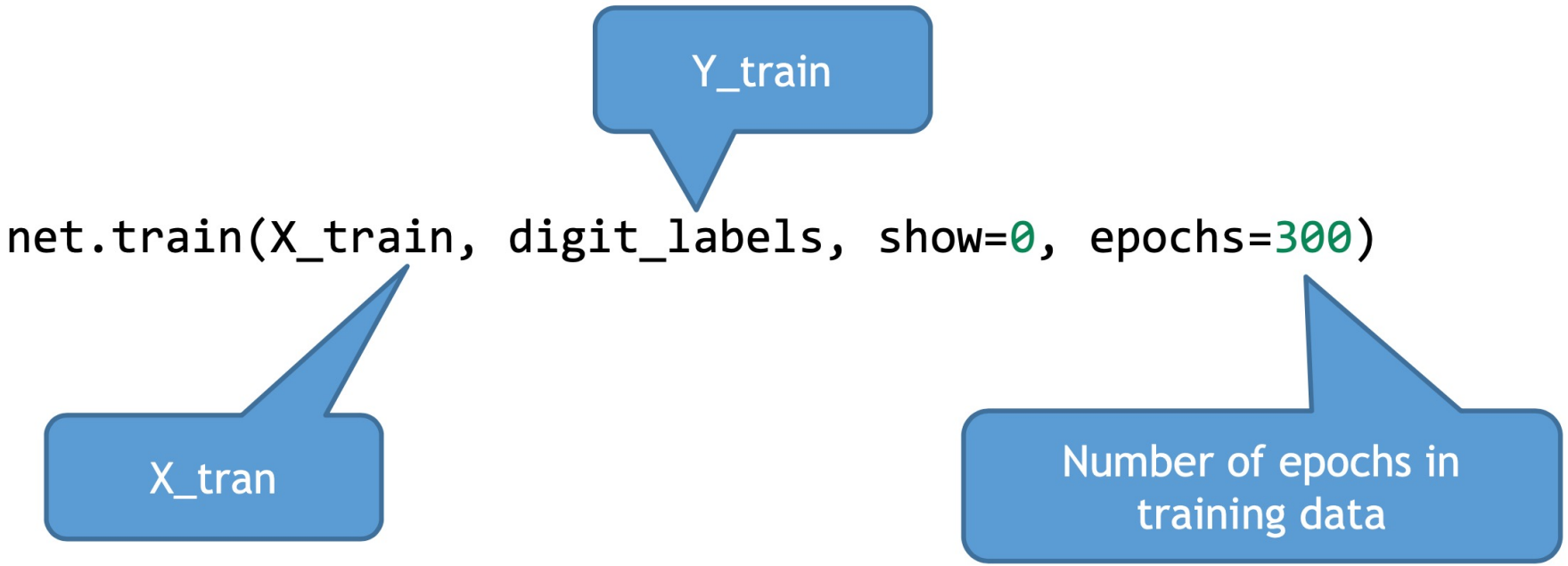
Building NN– Configure neural net




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

Train algorithms based gradients algorithms - Resilient Backpropagation

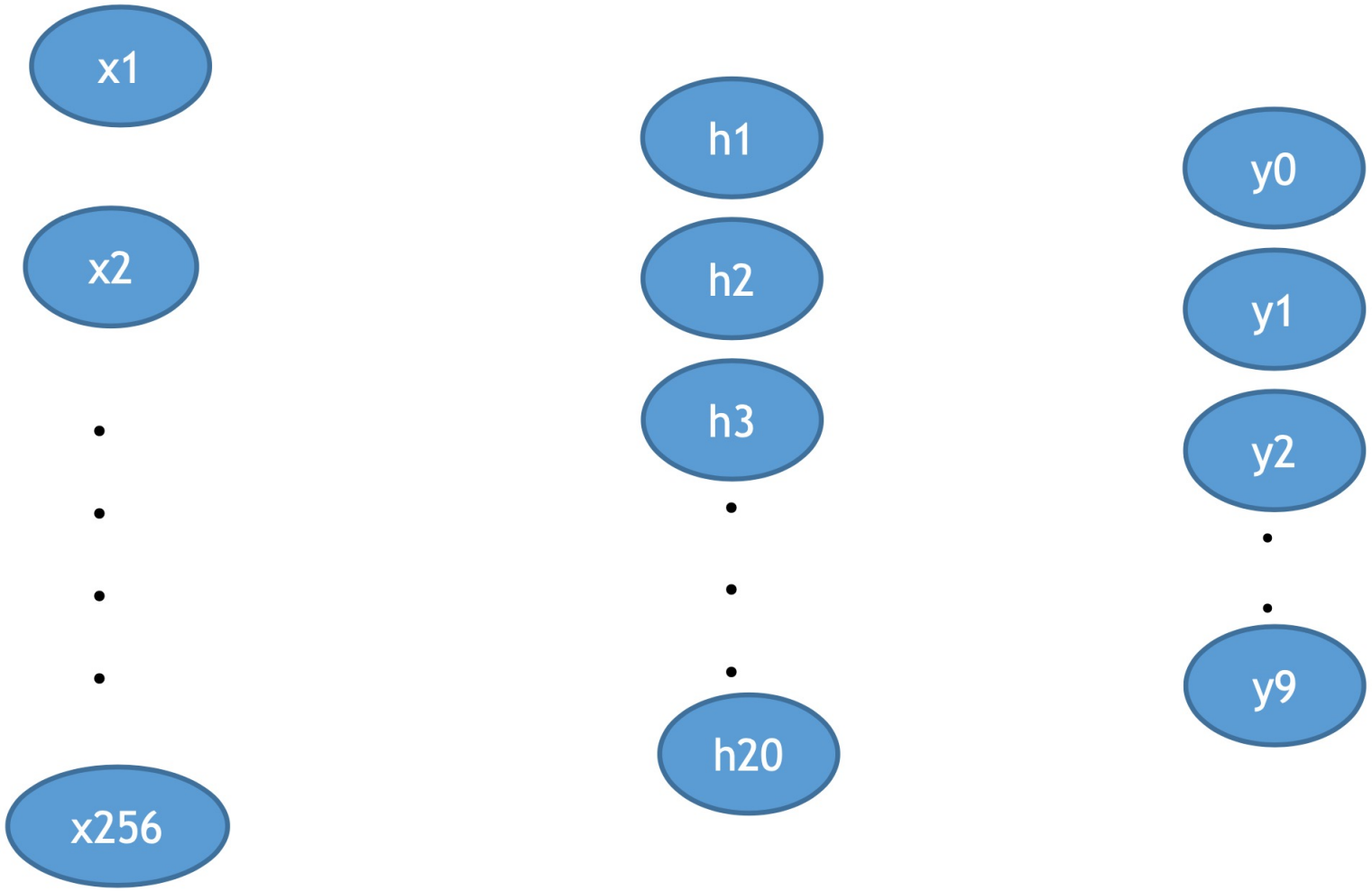
Building NN– Train neural net



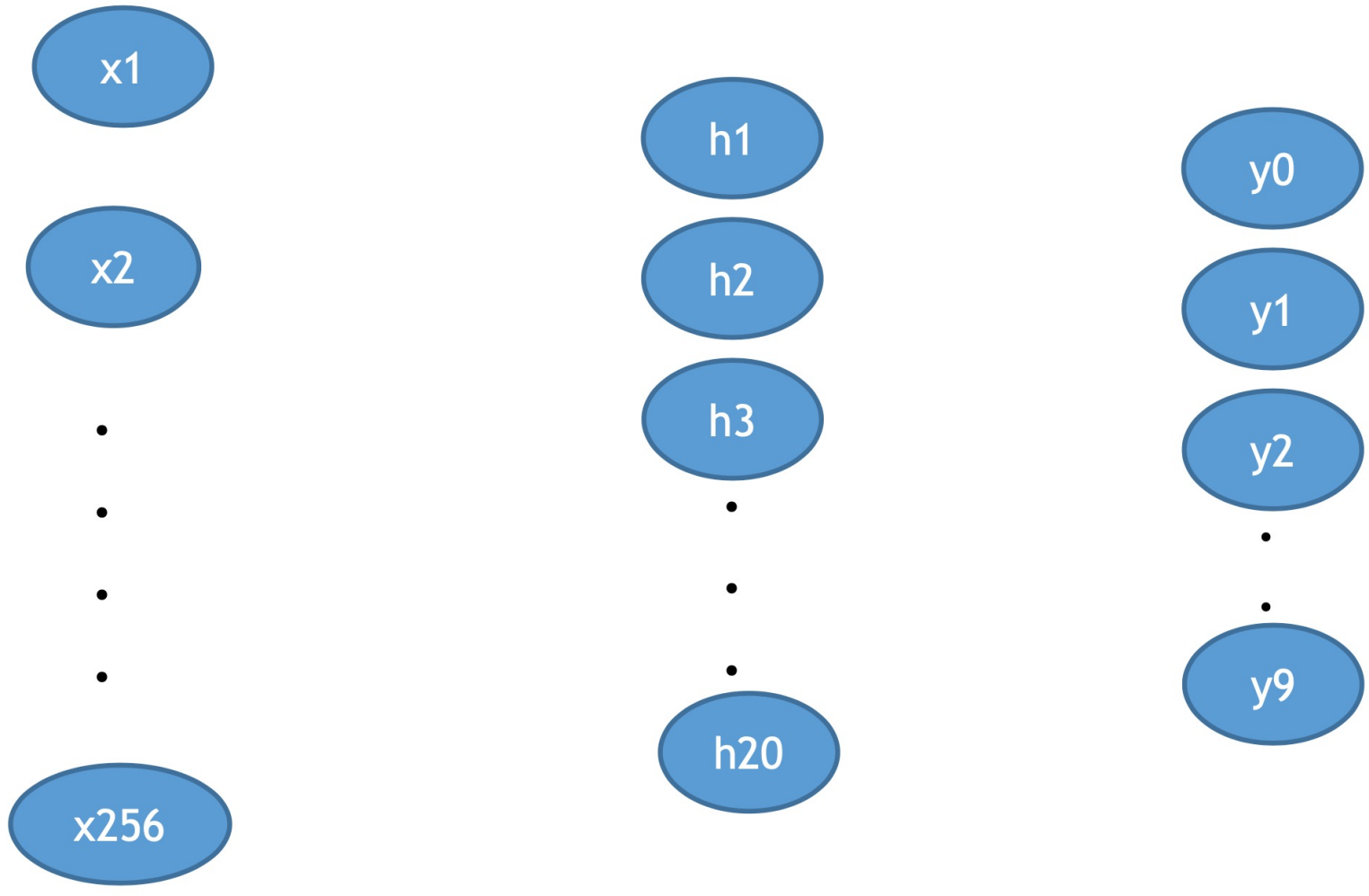
Code: Digit Recognizer

```
##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

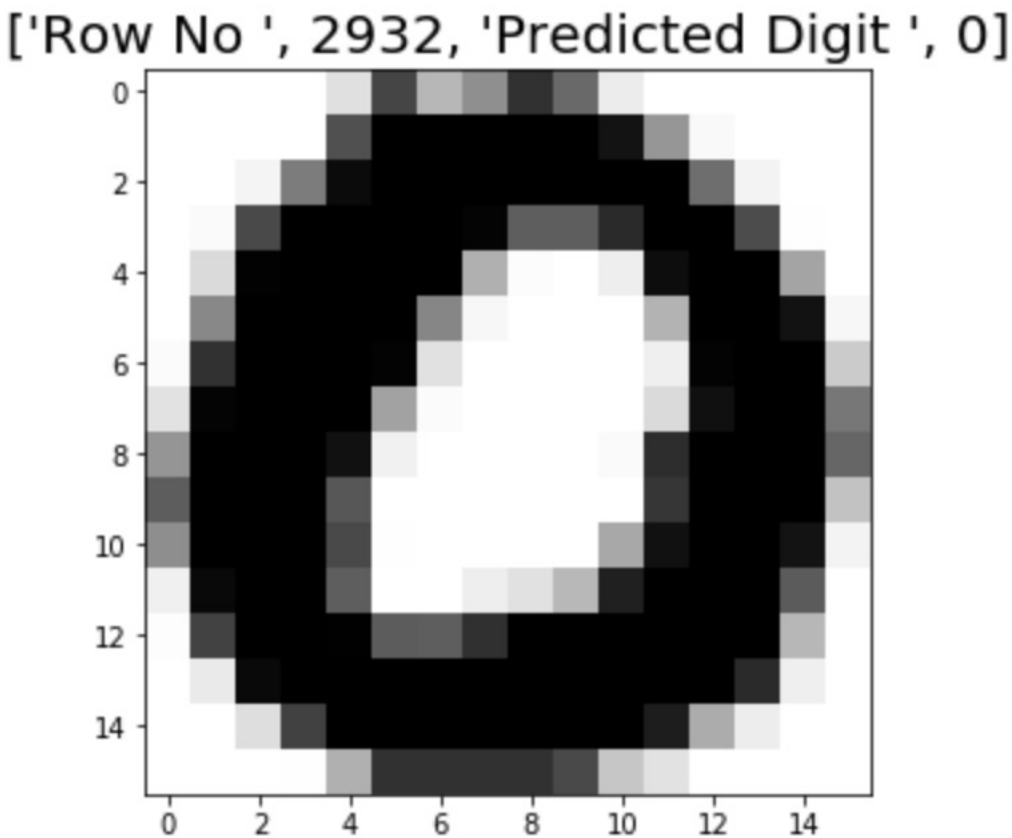


Code: Digit Recognizer

```
# 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 application

- Face recognition :

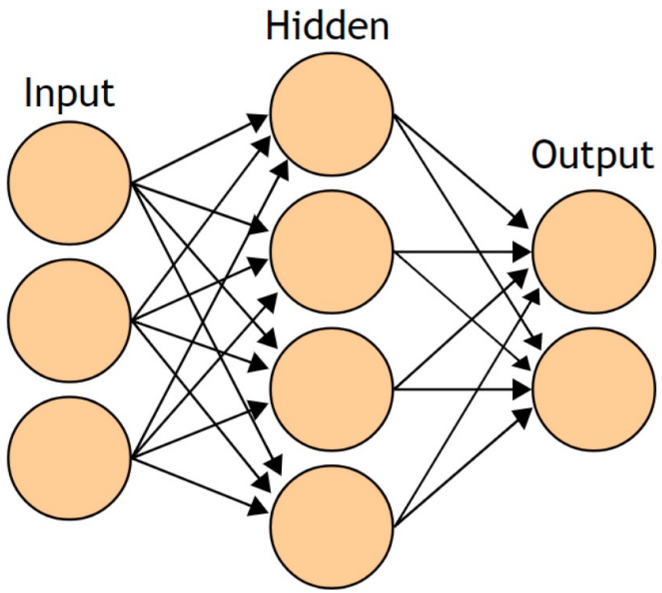
  https://www.youtube.com/watch?v=57VkfXqJ1LU

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

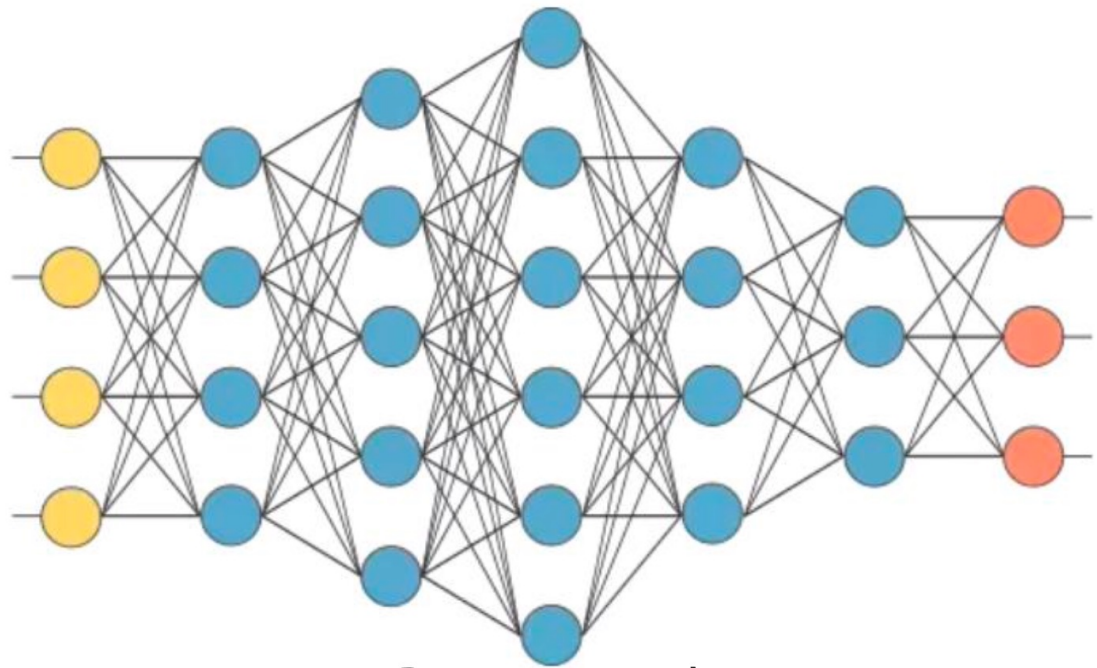
Deep vs Shallow networks

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



shallow network

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



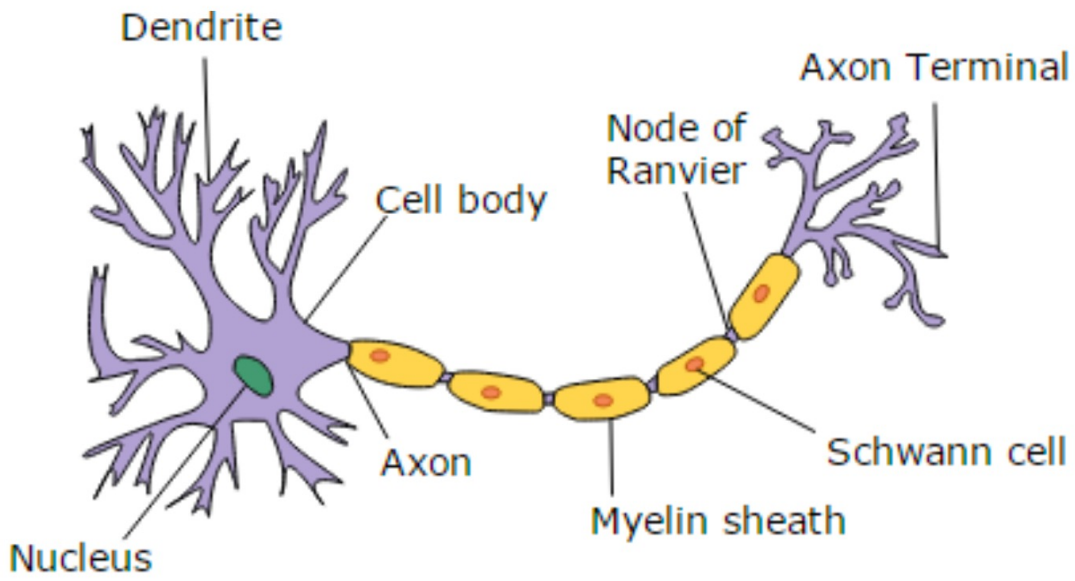
Deep network

Why the name neural network?

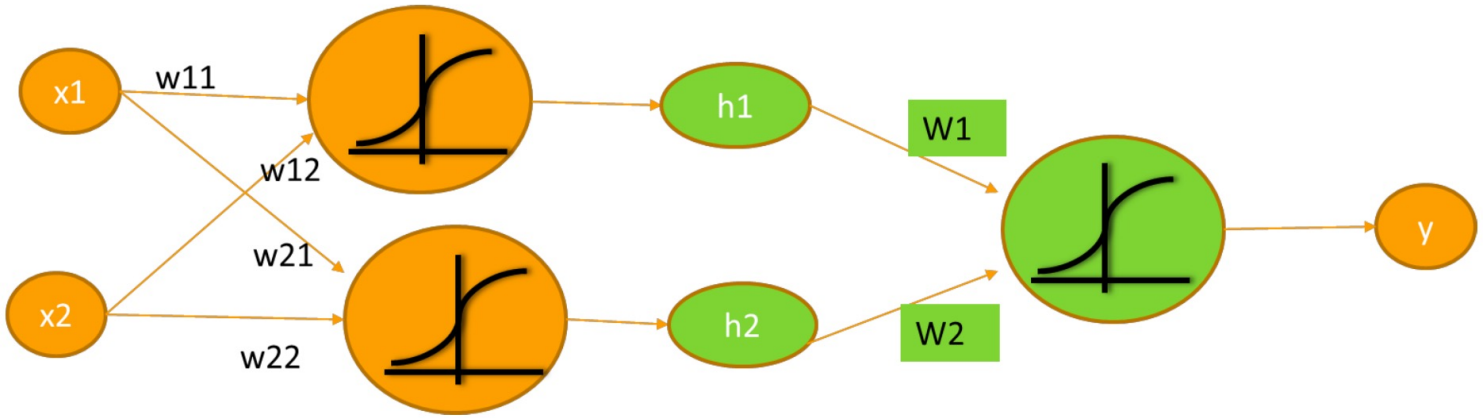
Why the name neural network?

- 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.,

Thank you