COSC 3337 : Data Science I

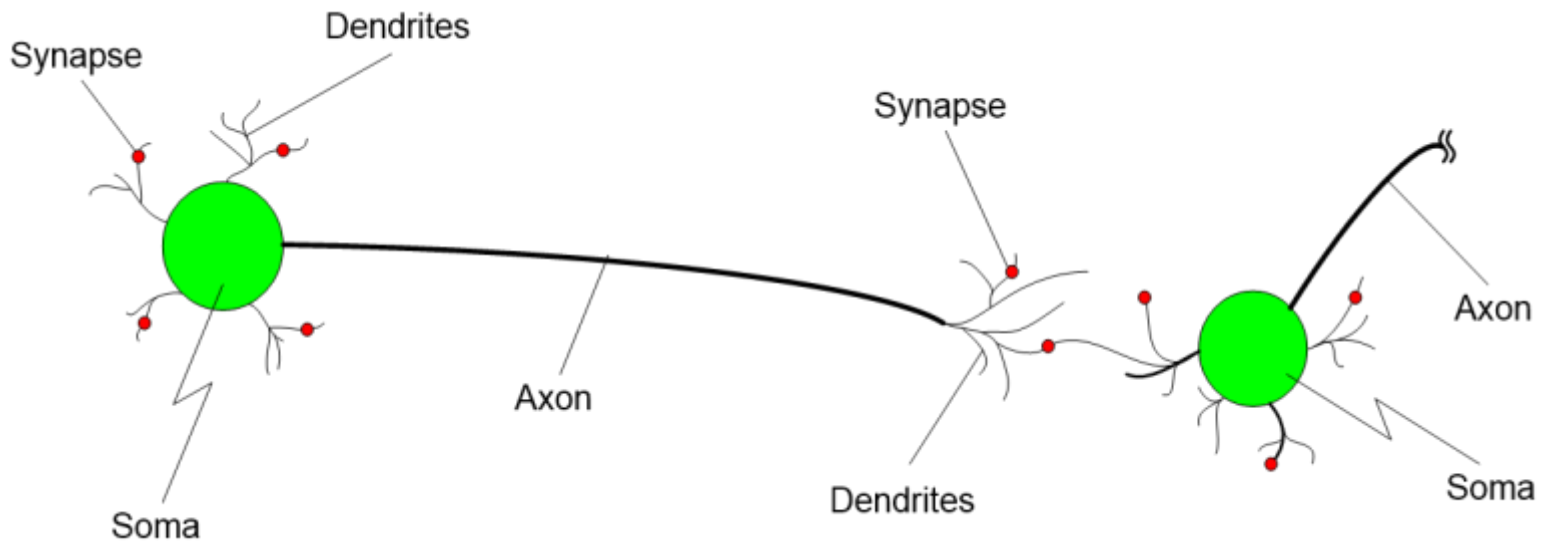
N. Rizk

College of Natural and Applied Sciences

Department of Computer Science

University of Houston

Artificial Neural Networks for Data Mining



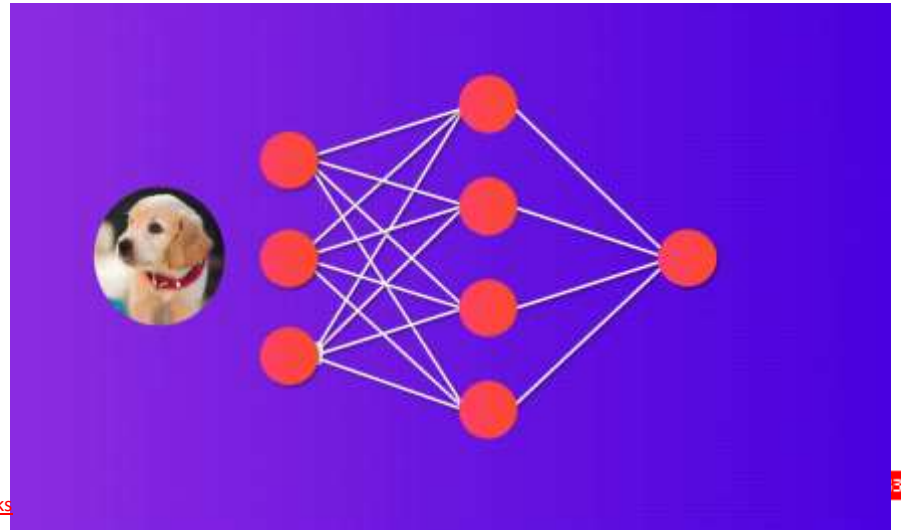
Two interconnected brain cells (neurons)

Biological Neural Networks

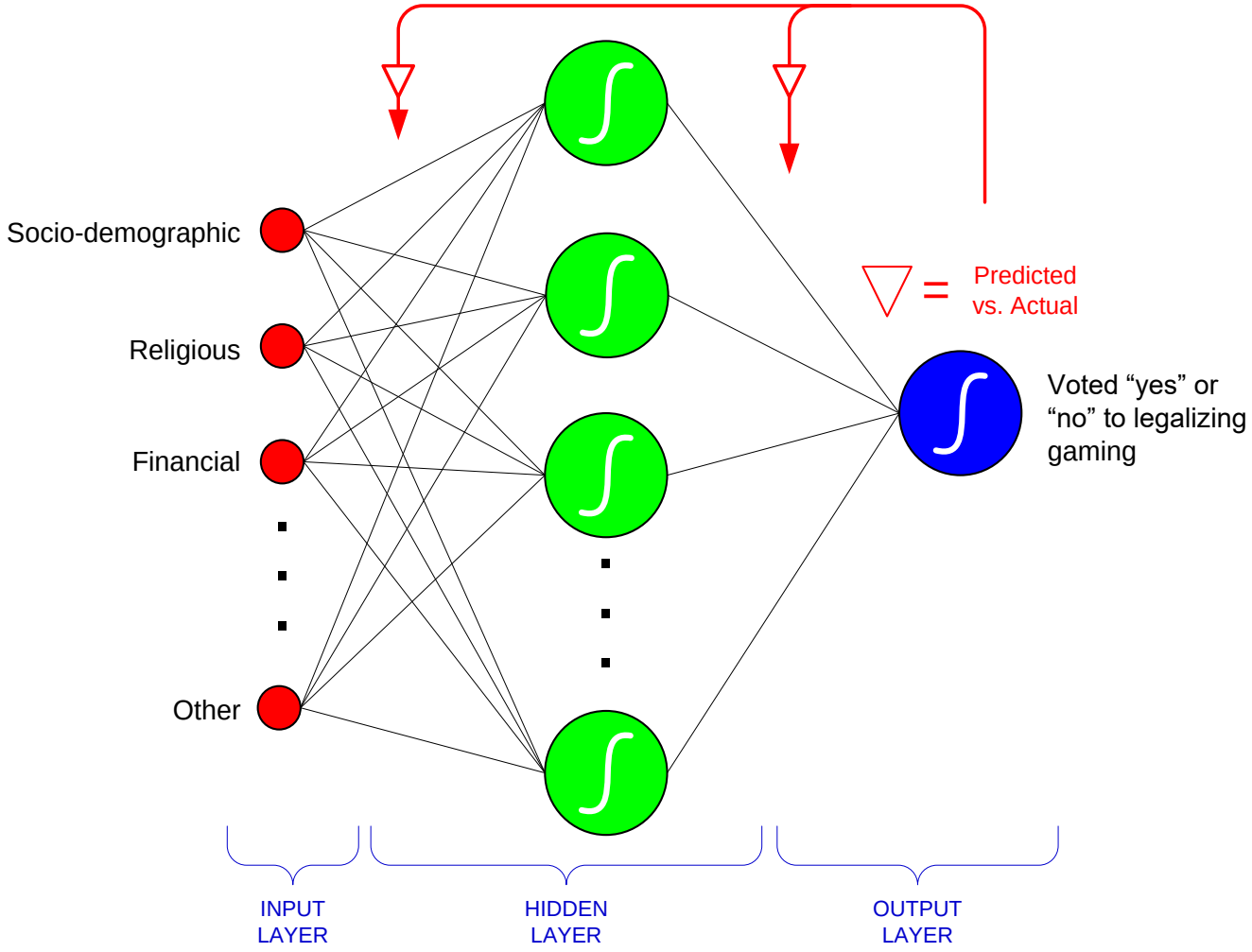
Opening Vignette:

"Predicting Gambling Referenda with Neural Networks"

- Decision situation
- Proposed solution
- Results
- Answer and discuss the case questions



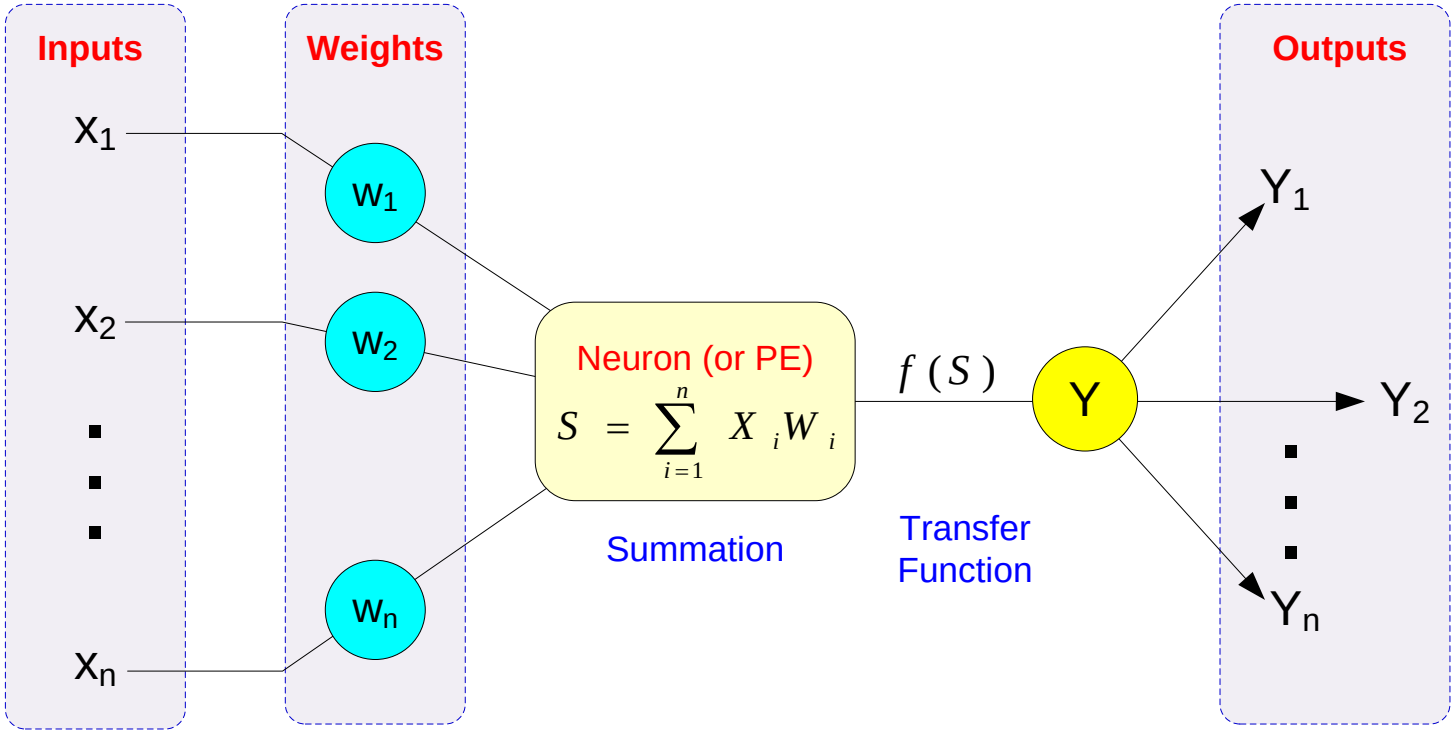
Opening Vignette: Predicting Gambling Referenda…



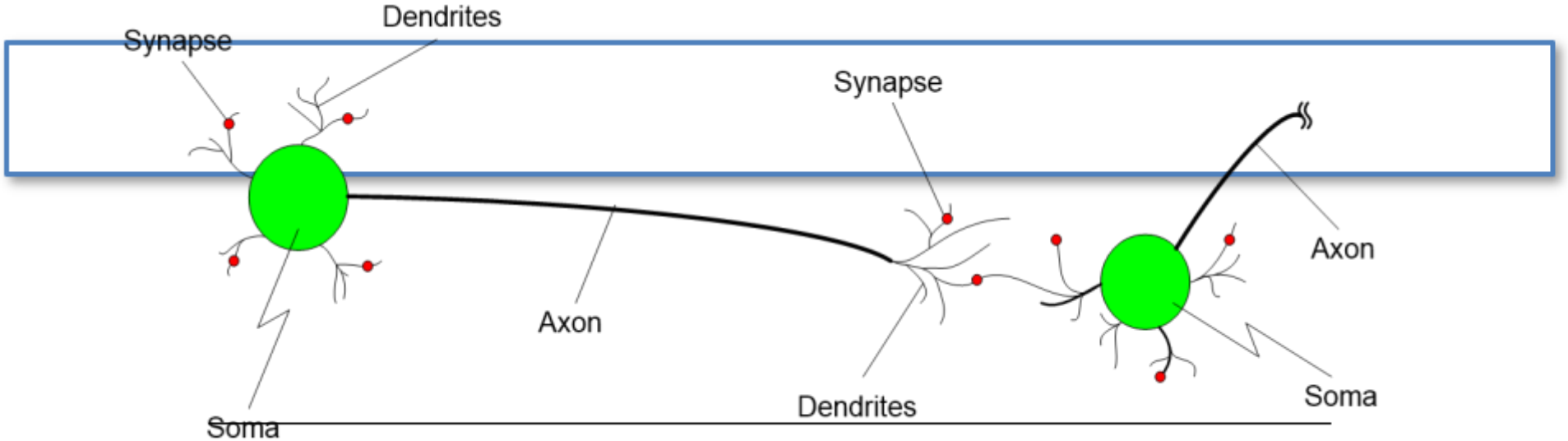
Neural Network Concepts

- Neural networks (NN): a brain metaphor for information processing
- Neural computing
- Artificial neural network (ANN)
- Many uses for ANN for
  - pattern recognition, forecasting, prediction, and classification
- Many application areas
  - finance, marketing, manufacturing, operations, information systems, and so on

Processing Information in ANN



- A single neuron (processing element – PE) with inputs and outputs



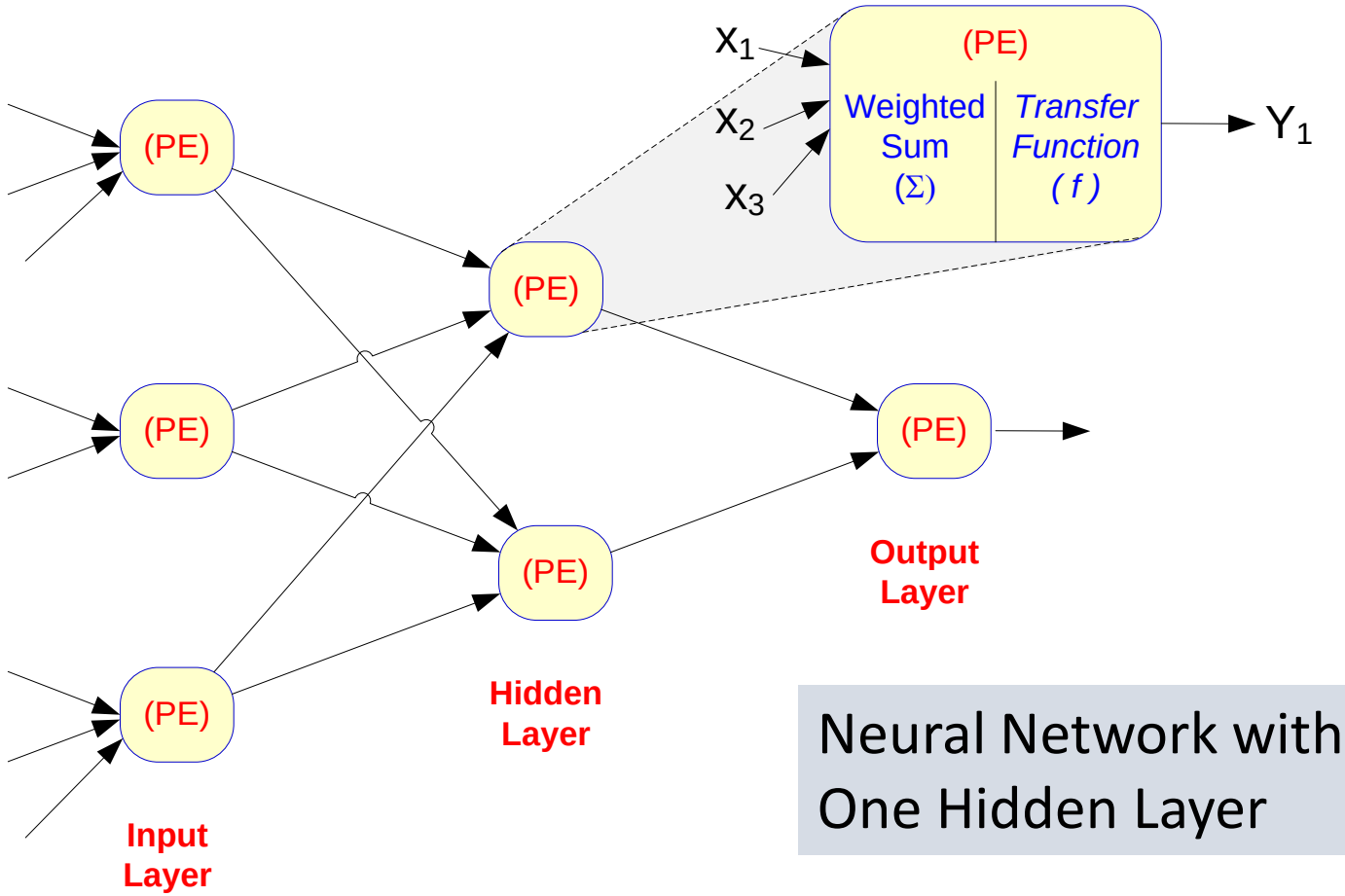
Biological versus Artificial NNs

Soma	Node
Dendrites	Input
Axon	Output
Synapse	Weight
Slow	Fast
Many neurons (10^9)	Few neurons (~100s)

Elements of ANN

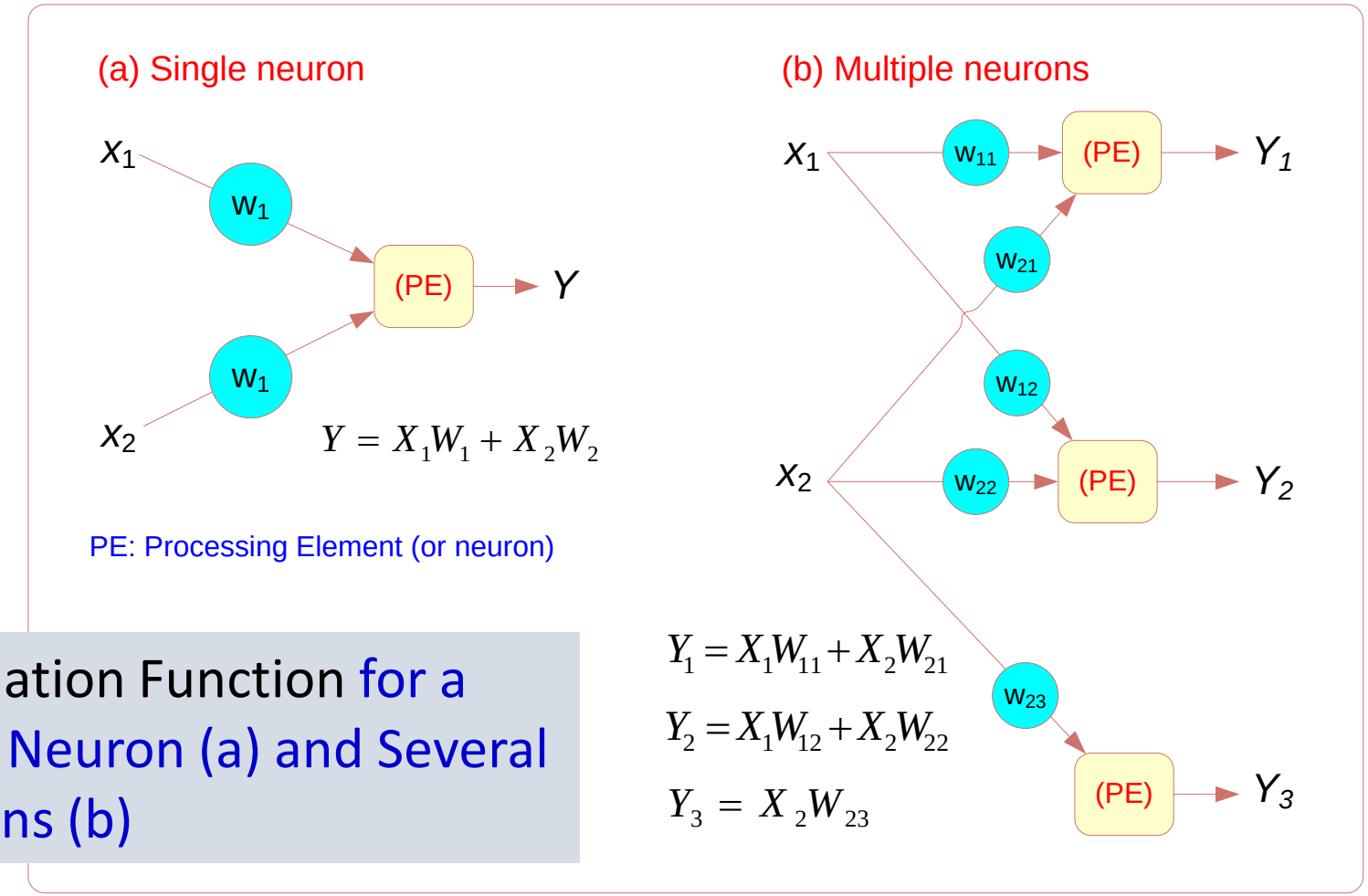
- Processing element (PE)
- Network architecture
  - Hidden layers
  - Parallel processing
- Network information processing
  - Inputs
  - Outputs
  - Connection weights
  - Summation function

Elements of ANN



Neural Network with One Hidden Layer

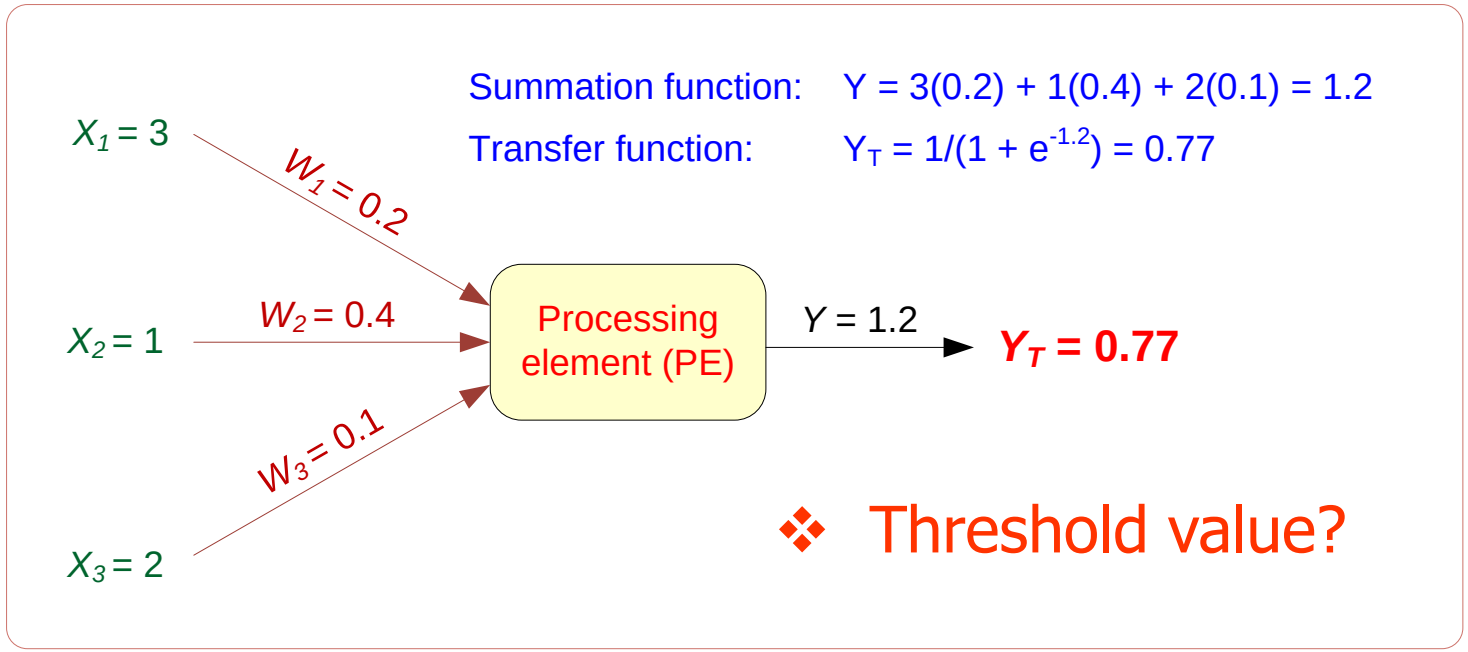
Elements of ANN



Summation Function for a Single Neuron (a) and Several Neurons (b)

Elements of ANN

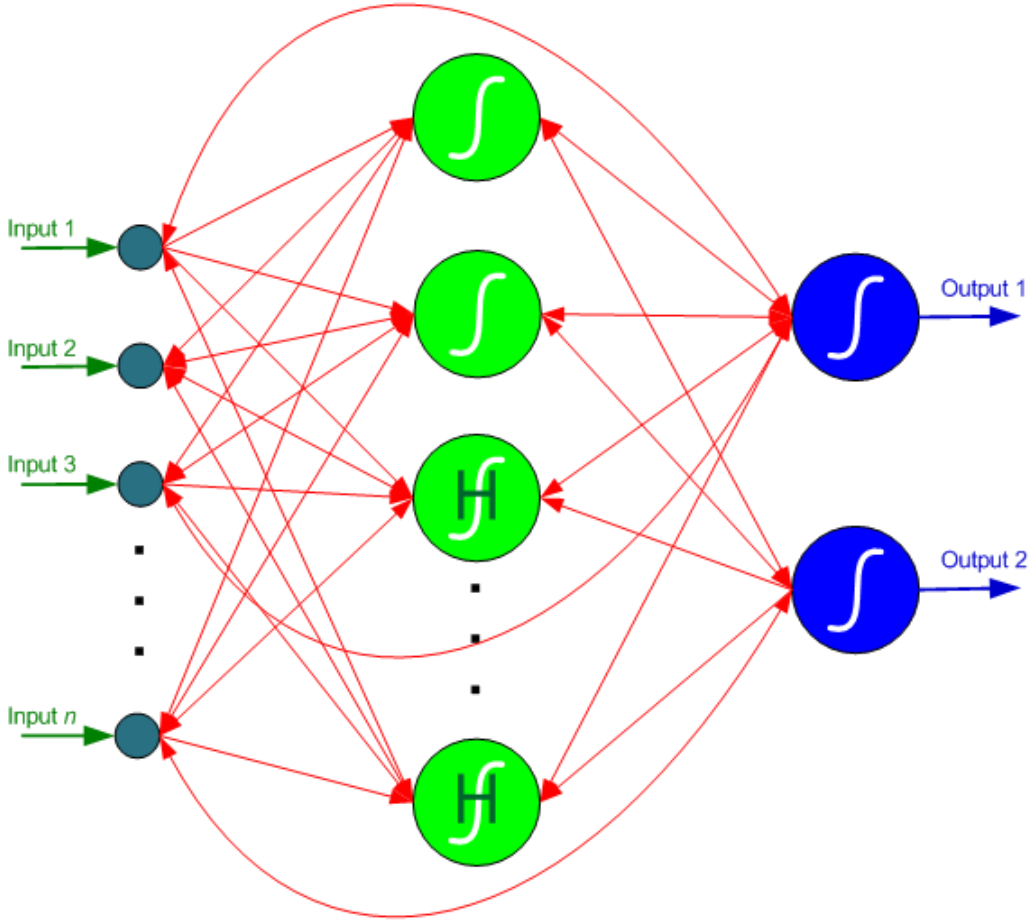
- Transformation (Transfer) Function
  - Linear function
  - Sigmoid (logical activation) function [0 1]
  - Tangent Hyperbolic function [-1 1]



Neural Network Architectures

- Several ANN architectures exist
  - Feedforward
  - Recurrent
  - Associative memory
  - Probabilistic
  - Self-organizing feature maps
  - Hopfield networks
  - … many more …

Neural Network Architectures
Recurrent Neural Networks



*H: indicates a "hidden" neuron without a target output

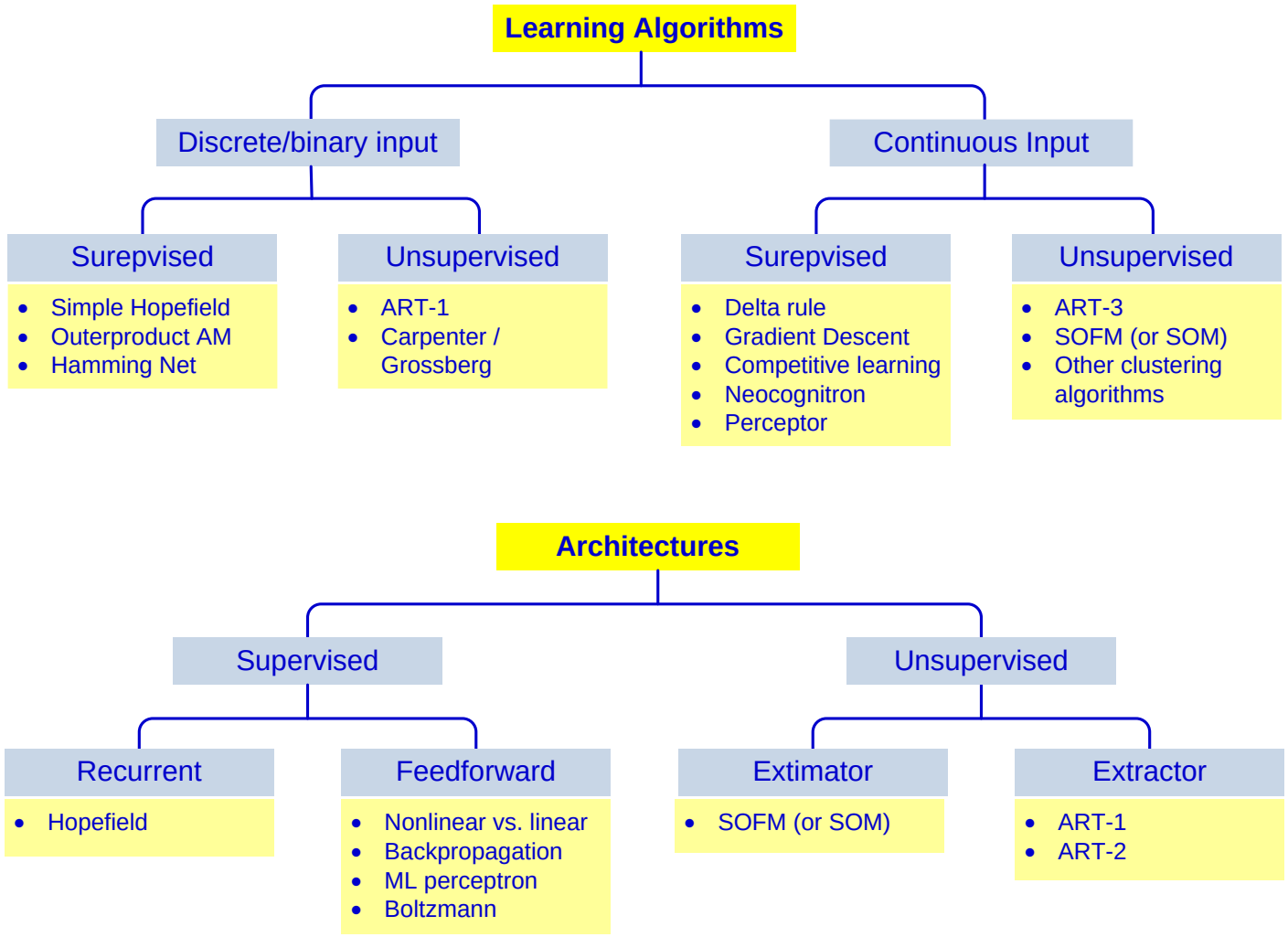
Neural Network Architectures

- Architecture of a neural network is driven by the task it is intended to address
  - Classification, regression, clustering, general optimization, association, ….
- Most popular architecture: Feedforward, multi-layered perceptron with backpropagation learning algorithm
  - Used for both classification and regression type problems

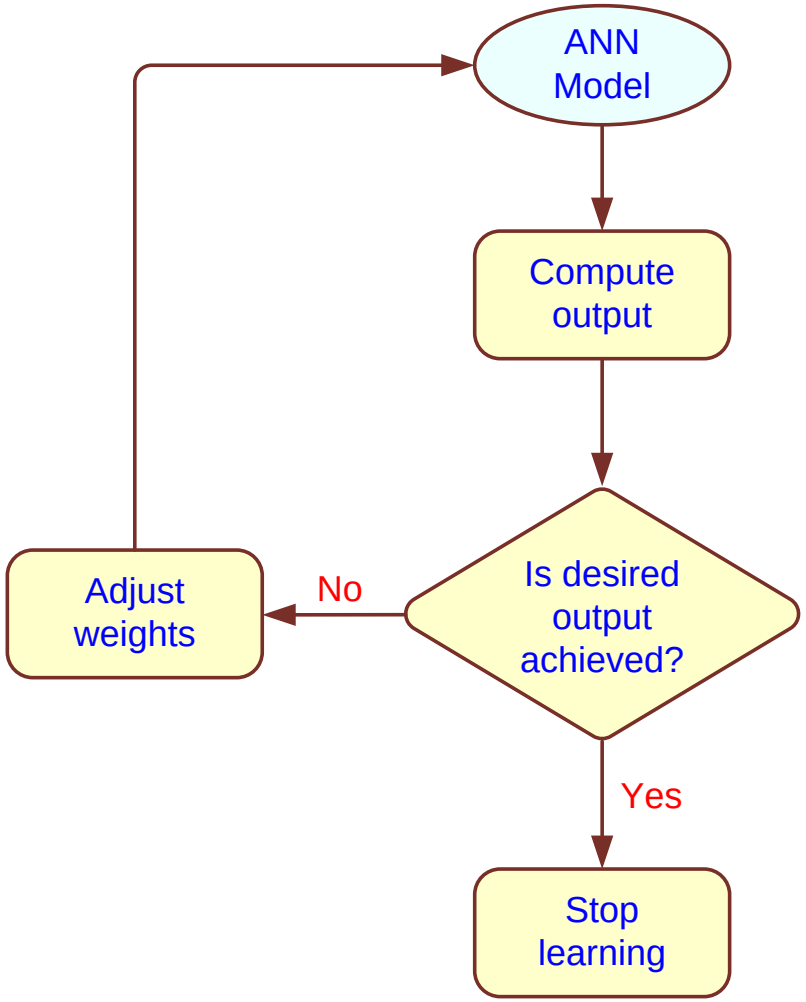
Learning in ANN

- A process by which a neural network learns the underlying relationship between input and outputs, or just among the inputs
- Supervised learning
  - For prediction type problems
  - E.g., backpropagation
- Unsupervised learning
  - For clustering type problems
  - Self-organizing
  - E.g., adaptive resonance theory

A Taxonomy of ANN Learning Algorithms



A Supervised Learning Process



Three-step process:

1. Compute temporary outputs
2. Compare outputs with desired targets
3. Adjust the weights and repeat the process

How a Network Learns

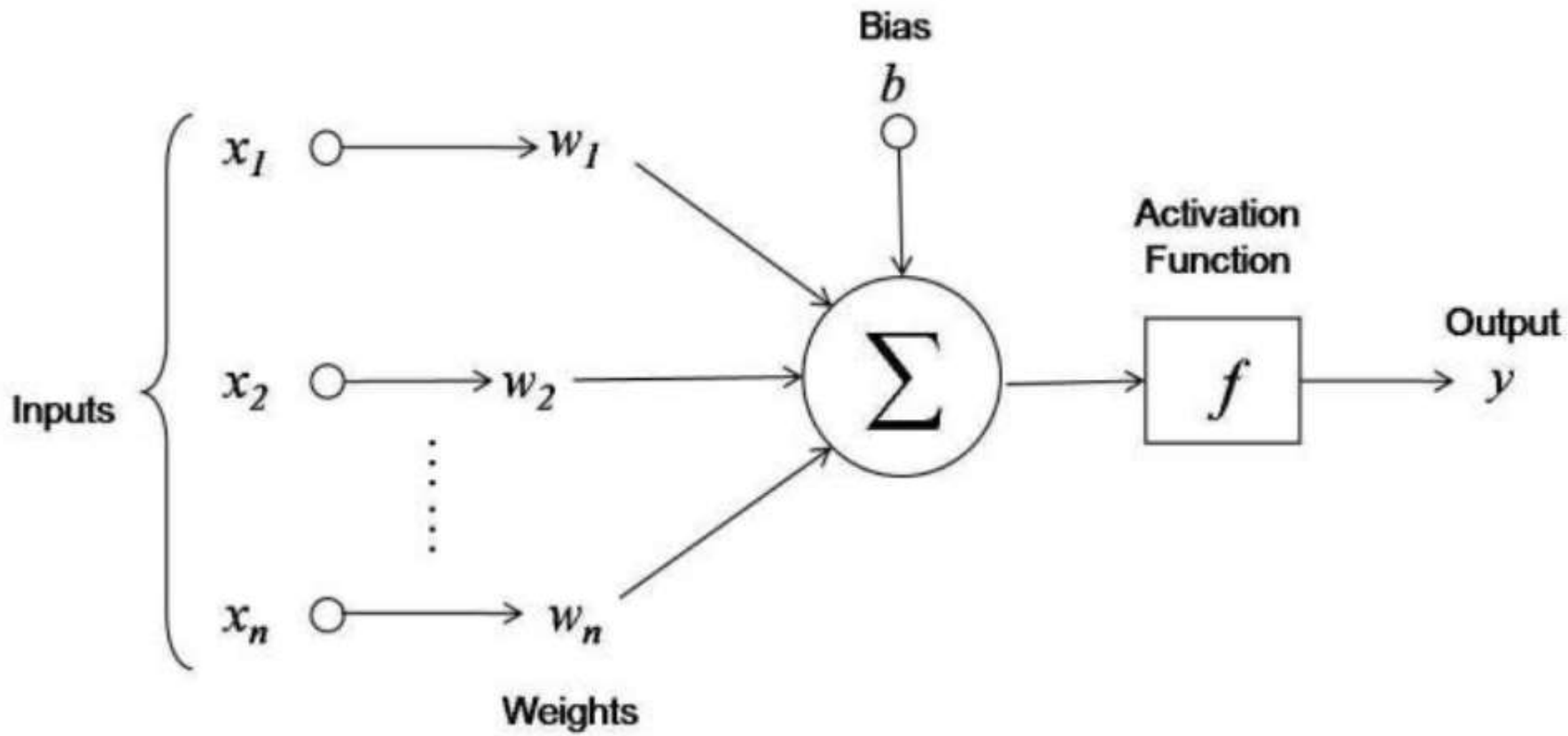
- Example: single neuron that learns the inclusive OR operation

	Inputs		
Case	X_1	X_2	Desired Results
1	0	0	0
2	0	1	1 (positive)
3	1	0	1 (positive)
4	1	1	1 (positive)

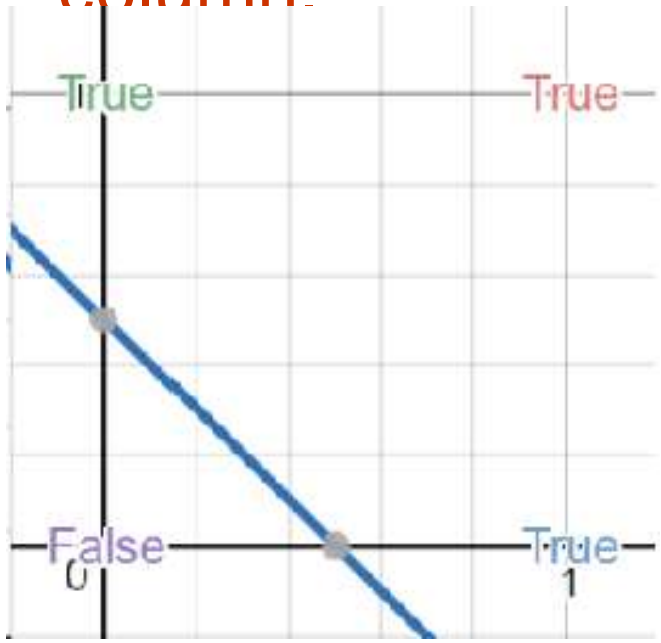
Learning parameters:

- Learning rate
- Momentum

classifier to distinguish 2 groups of output 0 or 1


Bias
b
Activation
Function
Output
y
Inputs
x_1
x_2
x_n
w_1
w_2
w_n
Weights

Perceptron is the first neural network ever created. It consists on 2 neurons in the inputs column and 1 neuron in the output column.



- if A is true and B is true, then A or B is true.
- if A is true and B is false, then A or B is true.
- if A is false and B is true, then A or B is true.
- if A is false and B is false, then A or B is false.

We are able to separate using a line (UNLIKE XOR)

In [1]:
```
import numpy, random, os
lr = 1 #learning rate
bias = 1 #value of bias
weights = [random.random(),random.random(),random.random()] #weights generated in a list (3 weights in total for 2 neurons and t
he bias)
```

In [4]:
```
#function which defines the work of the output neuron
def Perceptron(input1, input2, output) :
   outputP = input1*weights[0]+input2*weights[1]+bias*weights[2]
   if outputP > 0 : #activation function (here Heaviside)
      outputP = 1
   else :
      outputP = 0
   error = output - outputP
   weights[0] += error * input1 * lr
   weights[1] += error * input2 * lr
   weights[2] += error * bias * lr
```

In [5]:
```
# repeat in this case 50 times (keep low to avoid overfitting)
for i in range(50) :
   Perceptron(1,1,1) #True or true
   Perceptron(1,0,1) #True or false
   Perceptron(0,1,1) #False or true
   Perceptron(0,0,0) #False or false
```

In [6]:
```
x = int(input())
y = int(input())
outputP = x*weights[0] + y*weights[1] + bias*weights[2]
if outputP > 0 : #activation function
   outputP = 1
else :
   outputP = 0
print(x, "or", y, "is : ", outputP)
```

```
1
0
1 or 0 is :  1
```

In [7]:
```
x = int(input())
y = int(input())
outputP = 1/(1+numpy.exp(-outputP)) #sigmoid function
print(x, "or", y, "is : ", outputP)
```

```
1
0
1 or 0 is :  0.7310585786300049
```

In [ ]:
```
#Multi-Layer Perceptron model
import pandas as pd
wine = pd.read_csv('wine_data.csv', names = ["Cultivator", "Alchol", "Malic_Acid", "Ash", "Alcalinity_of_Ash", "Ma
```

In [ ]:
```
wine.head()
```

In [ ]:
```
wine.describe().transpose()
```

In [ ]:
```
wine.shape
```

In [ ]:
```
#set you label
X = wine.drop('Cultivator',axis=1)
y = wine['Cultivator']
```

In [ ]:
```
from sklearn.model_selection import train_test_split
```

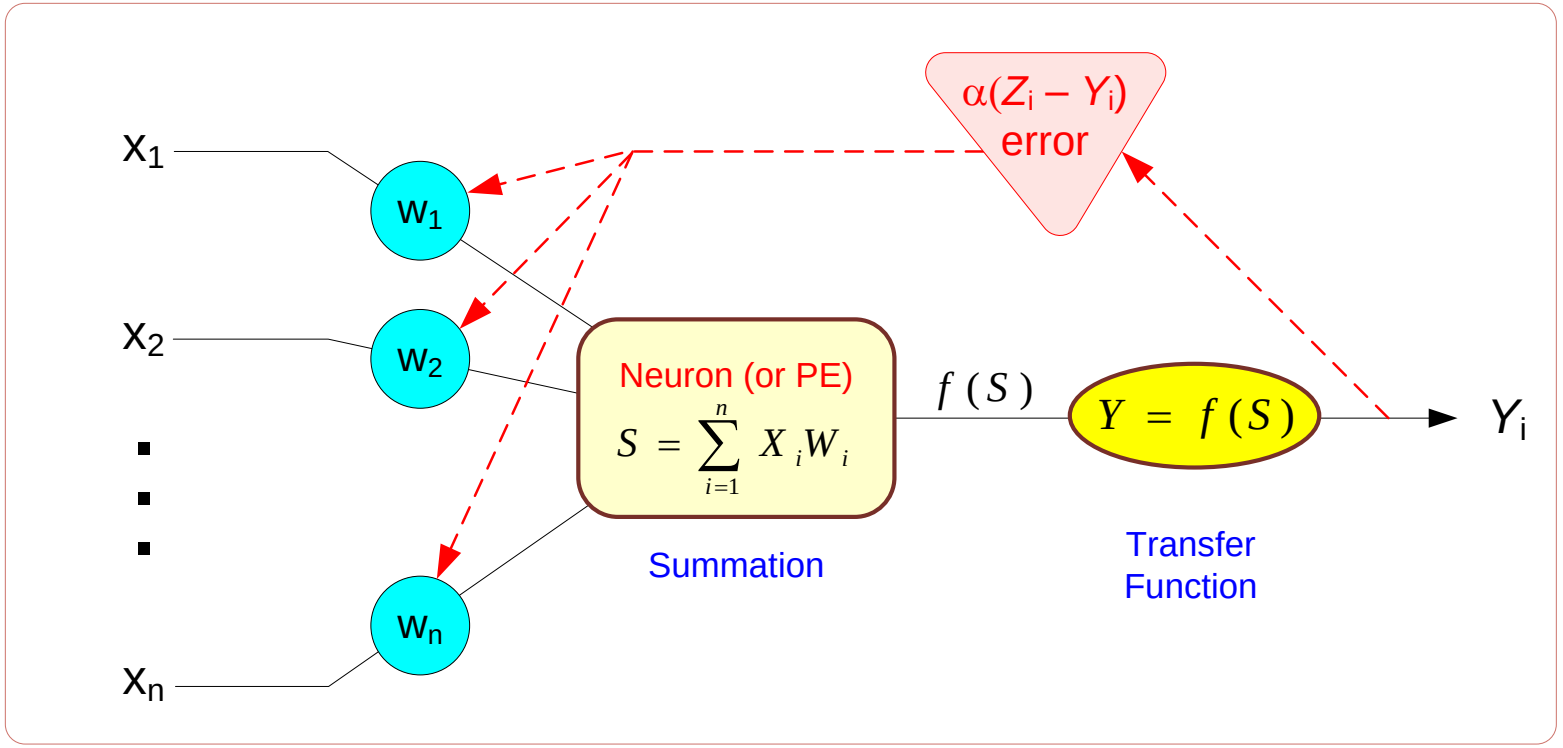
In [ ]:
```
#X_train, X_test, y_train, y_test = train_test_split(X, y)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

In [ ]:
```
#preprocessing
from sklearn.preprocessing import StandardScalerc
```

In [ ]:
```
 from sklearn import preprocessing
scaler = preprocessing.StandardScaler().fit(X_train)
```

```
In [ ]: mlp = MLPClassifier(hidden_layer_sizes=(13,13,13),max_iter=500)

In [ ]: mlp.fit(X_train,y_train)

In [ ]: MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(13, 13, 13), learning_rate='constant',
              learning_rate_init=0.001, max_iter=500, momentum=0.9,
              nesterovs_momentum=True, power_t=0.5, random_state=None,
              shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
              verbose=False, warm_start=False)

In [ ]: #predictions
        predictions = mlp.predict(X_test)

In [ ]: from sklearn.metrics import classification_report,confusion_matrix

In [ ]: print(confusion_matrix(y_test,predictions))

In [ ]: print(classification_report(y_test,predictions))
```

Backpropagation Learning

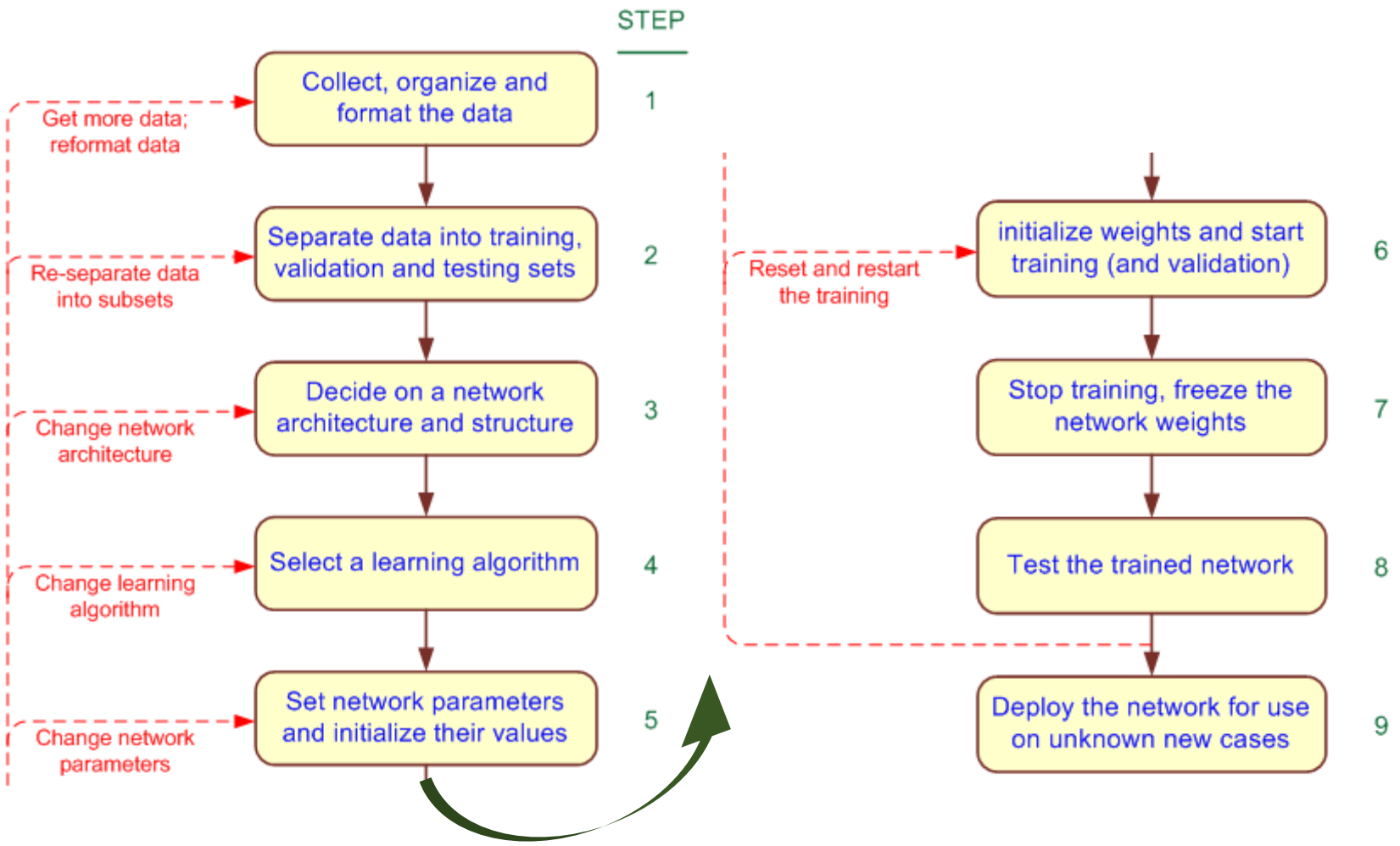


- Backpropagation of Error for a Single Neuron

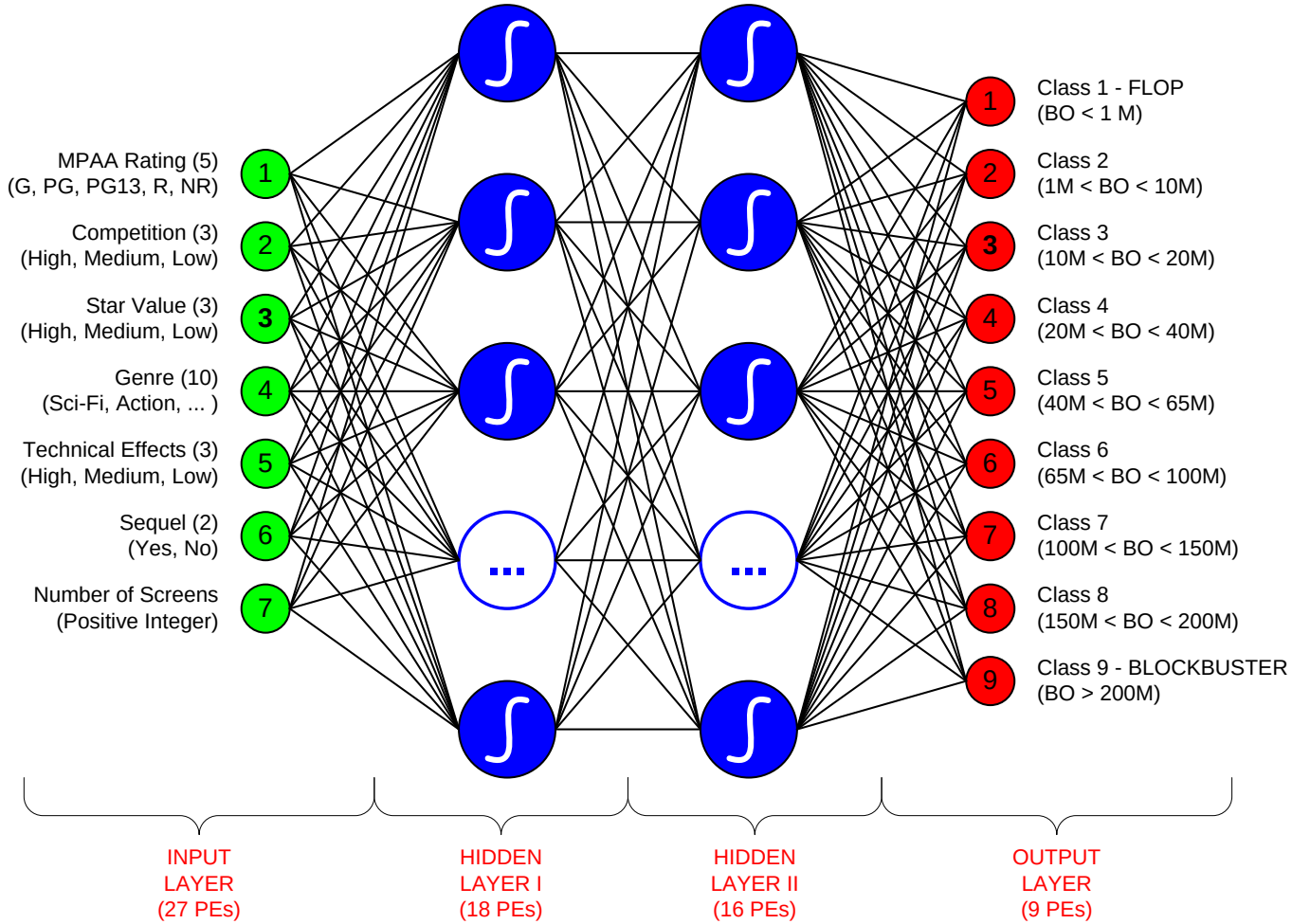
Backpropagation Learning

- The learning algorithm procedure:
  1. Initialize weights with random values and set other network parameters
  2. Read in the inputs and the desired outputs
  3. Compute the actual output (by working forward through the layers)
  4. Compute the error (difference between the actual and desired output)
  5. Change the weights by working backward through the hidden layers
  6. Repeat steps 2-5 until weights stabilize

Development Process of an ANN



An MLP ANN Structure for the Box-Office Prediction Problem



Testing a Trained ANN Model

- Data is split into three parts
  - Training (~60%)
  - Validation (~20%)
  - Testing (~20%)

- *k*-fold cross validation
  - Less bias
  - Time consuming

In [ ]:
```
 import pandas as pd
wine = pd.read_csv(r'wine_data.csv', names = ["Cultivator", "Alchol", "Malic_Acid", "Ash", "Alcalinity_of_Ash", "M
X = wine.drop('Cultivator',axis=1) #input
y = wine['Cultivator']
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, random_state=1)
```

In [ ]:
```
from sklearn.model_selection import KFold
kf = KFold(n_splits = 5, shuffle = True, random_state = 2)

for train_index, test_index in kf.split(X):
    X_tr_va, X_test = X.iloc[train_index], X.iloc[test_index]
    y_tr_va, y_test = y[train_index], y[test_index]
    X_train, X_val, y_train, y_val = train_test_split(X_tr_va, y_tr_va, test_size=0.25, random_state=1)
    print("TRAIN:", list(X_train.index), "VALIDATION:", list(X_val.index), "TEST:", test_index)
```

Applications Types of ANN

- Classification
  - Feedforward networks (MLP), radial basis function, and probabilistic NN
- Regression
  - Feedforward networks (MLP), radial basis function
- Clustering
  - Adaptive Resonance Theory (ART) and SOM
- Association
  - Hopfield networks

Advantages of ANN

- Able to deal with (identify/model) highly nonlinear relationships
- Not prone to restricting normality and/or independence assumptions
- Can handle variety of problem types
- Usually provides better results (prediction and/or clustering) compared to its statistical counterparts
- Handles both numerical and categorical variables (transformation needed!)

Disadvantages of ANN

- They are deemed to be black-box solutions, lacking expandability
- It is hard to find optimal values for large number of network parameters
  - Optimal design is still an art: requires expertise and extensive experimentation
- It is hard to handle large number of variables (especially the rich nominal attributes)
- Training may take a long time for large datasets; which may require case sampling

ANN Software

- Standalone ANN software tool
  - NeuroSolutions
  - BrainMaker
  - NeuralWare
  - NeuroShell, … for more (see pcai.com) …
- Part of a data mining software suit
  - PASW (formerly SPSS Clementine)
  - SAS Enterprise Miner
  - Statistica Data Miner, … many more …

In order to create a neural network:

1- specify the number of layers within the model (e.g two layers)

2-Select the type of activation for each layer (e.g a sigmoid activation for the first layers)

3- Number of nodes in the final layer (e.g having 10 nodes)➔set to return 10 probability scores

4- To compile the model, a loss function must be defined (how the model evaluates its own performance)

5- An optimizer must be determined, which is how the information from the cost function is used to change the weights and the bias of each node.

```
model = keras.Sequential([keras.layers.Flatten(input_shape (28,28)),
keras.layers.Dense(128,activation = tf.nn.sigmoid),
keras.layers.Dense(10,activation = tf.nn.softmax)])
model.compile(optimizer =
'adam',loss='sparse_categorical_crossentropy',metrics =['accuracy'])
```

Train the model , then evaluate it based on the testing data. ➔the need number of epochs.

Epochs determine how many iterations through the data shall be done.

```
model.fit(x_train, y_train,epochs = 5)
```

```
Epoch 1/5
60000/60000 [==============================] - 6s 93us/sample - loss: 0.8991 - accuracy: 0.6946
Epoch 2/5
60000/60000 [==============================] - 5s 84us/sample - loss: 0.7550 - accuracy: 0.7120
Epoch 3/5
60000/60000 [==============================] - 5s 90us/sample - loss: 0.7294 - accuracy: 0.7199
Epoch 4/5
60000/60000 [==============================] - 6s 92us/sample - loss: 0.7073 - accuracy: 0.7318
Epoch 5/5
60000/60000 [==============================] - 5s 90us/sample - loss: 0.6969 - accuracy: 0.7340
```

class NeuralNetwork:

```
def __init__(self, x, y):
self.input = x
self.weights1 =
np.random.rand(self.input.shape[1],4)
self.weights2 = np.random.rand(4,1)
self.y = y
self.output = np.zeros(y.shape)
```

Neural Networks consist of the following components

- An input layer, *x*
- An arbitrary amount of hidden layers
- An output layer, $\hat{y}$
- A set of weights and biases between each layer, *W and b*
- A choice of activation function for each hidden layer, σ. Such as a Sigmoid activation function.

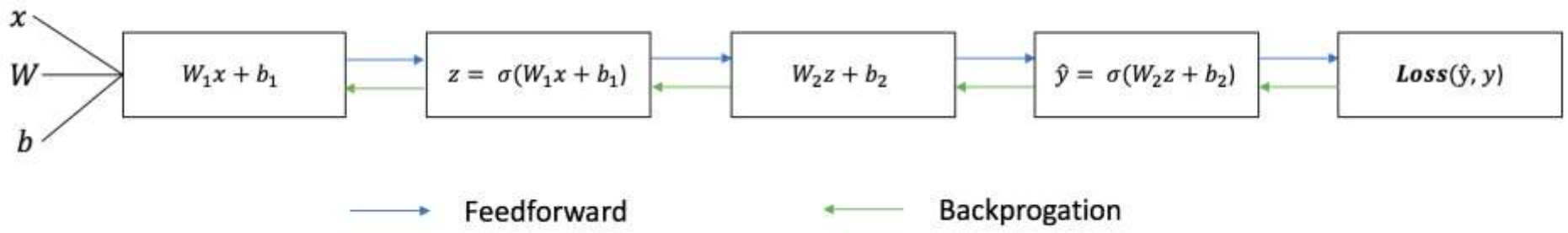
Training the Neural Network

The process of fine-tuning the weights and biases from the input data is known as training the Neural Network.

The output $\hat{y}$ of a simple 2-layer Neural Network is:

$$\hat{y} = \sigma(W_2 \sigma(W_1 x + b_1) + b_2)$$

Each iteration of the training process consists of the following steps:



- Calculating the predicted output $\hat{y}$, known as feedforward
- Updating the weights and biases, known as backpropagation

```
def feedforward(self):

    self.layer1 = sigmoid(np.dot(self.input,
    self.weights1))
    self.output = sigmoid(np.dot(self.layer1,
    self.weights2))
```

Evaluate the "goodness" of the predictions ➔loss function

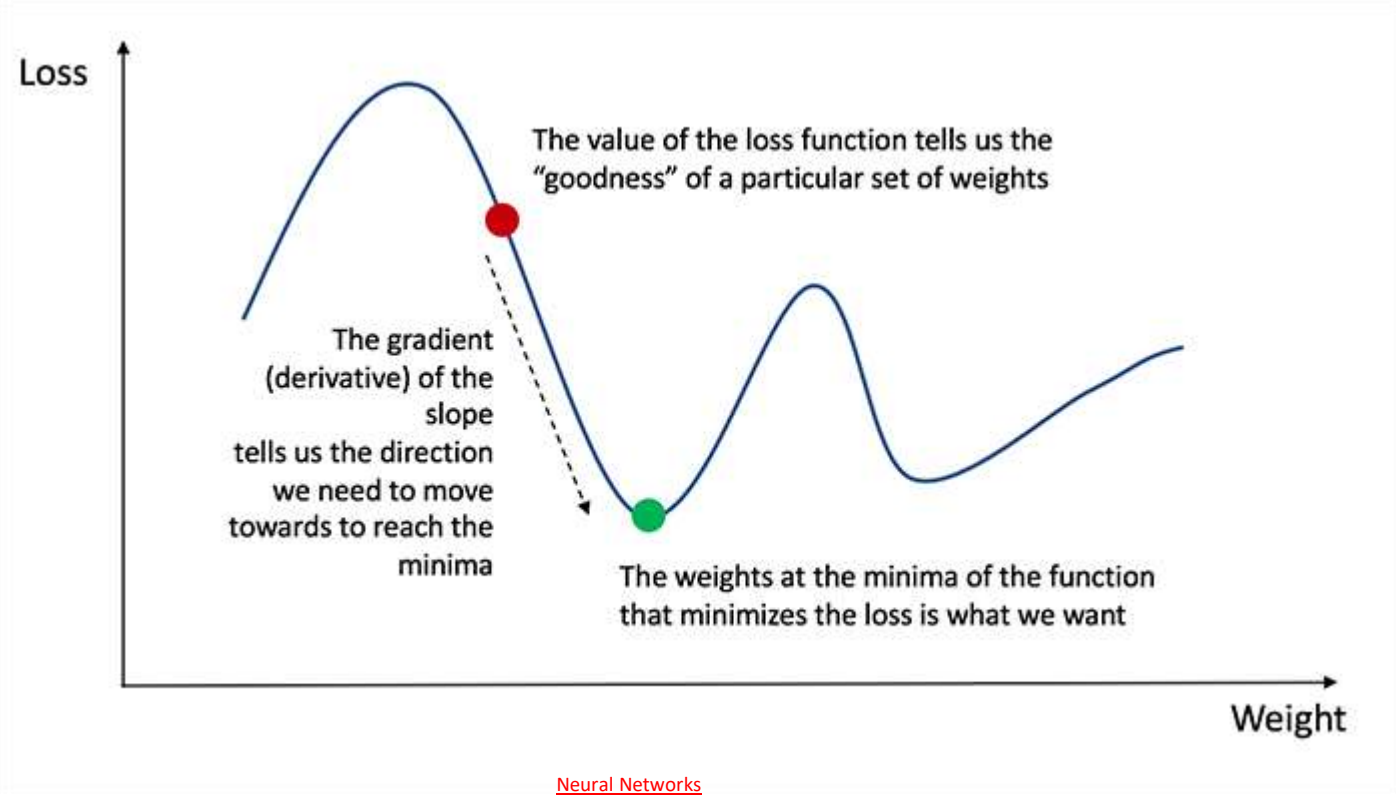
$$Sum - of - Squares\ Error = \sum_{i=1}^{n} (y - \hat{y})^2$$

Training the model is to find the best set of weights and biases that minimizes the loss function.

Backpropagation

How to propagate the error back, and to update weights and biases?

By finding derivative of the loss function with respect to the weights and biases (the slope)



The derivative of the loss function cannot be calculated with respect to the weights and biases because the equation of the loss function does not contain the weights and biases. ➔the we need the chain rule

$$Loss(y, \hat{y}) = \sum_{i=1}^{n} (y - \hat{y})^2$$

$$\frac{\partial\, Loss(y,\hat{y})}{\partial W} = \frac{\partial Loss(y,\hat{y})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \frac{\partial z}{\partial W} \qquad \text{where } z = Wx + b$$

$$= 2(y - \hat{y}) * \text{derivative of sigmoid function} * x$$

$$= 2(y - \hat{y}) * z(1\text{-}z) * x$$

```
def backprop(self):
    # application of the chain rule to find
    derivative of the loss function with respect
    to weights2 and weights1
    d_weights2 = np.dot(self.layer1.T,
    (2*(self.y - self.output) *
    sigmoid_derivative(self.output)))
    d_weights1 = np.dot(self.input.T,
    (np.dot(2*(self.y - self.output) *
    sigmoid_derivative(self.output),
    self.weights2.T) *
    sigmoid_derivative(self.layer1)))

    # update the weights with the derivative
    (slope) of the loss function
    self.weights1 += d_weights1
    self.weights2 += d_weights2
```

Example

X1	X2	X3	Y
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	0

Predictions after 1500 training iterations

Prediction	Y (Actual)
0.023	0
0.979	1
0.975	1
0.025	0

feedforward and backpropagation algorithm trained the Neural Network successfully and the predictions converged on the true values.