K2 Analytics
Building Skills, Building Individuals

Artificial Neural Network
(…or simply Neural Network)

- Rajesh Jakhotia

26-Oct-2015

Earning is in Learning
- Rajesh Jakhotia

About K2 Analytics

At K2 Analytics, we believe that skill development is very important for the growth of an individual, which in turn leads to the growth of Society & Industry and ultimately the Nation as a whole. For this it is important that access to knowledge and skill development trainings should be made available easily and economically to every individual.

Our Vision: *"To be the preferred partner for training and skill development"*

Our Mission: *"To provide training and skill development training to individuals, make them skilled & industry ready and create a pool of skilled resources readily available for the industry"*

*We have chosen Business Intelligence and Analytics as our focus area. With this endeavour we make this presentation on "**Neural Network**" accessible to all those who wish to learn Analytics. We hope it is of help to you. For any feedback / suggestion or if you are looking for job in analytics then feel free to write back to us at ar.jakhotia@k2analytics.co.in*

Welcome to Artificial Intelligence – Neural Network!!!

K2 Analytics
Building Skills, Building Individuals

Agenda

Introduction
Application of analytics in business
Skills required for Business Analytics
Data Mining in a nut shell
Basic number skills
Classification Tree

Artificial Neural Network

Artificial Neural Network (ANN)

are

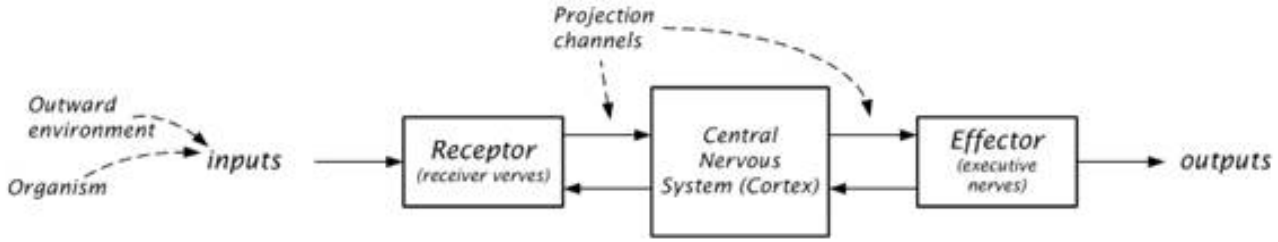
Family of Models

inspired by

Biological Neural Networks

Artificial Neural Networks, commonly referred to as *neural networks* has been motivated right from its inception by the recognition that the brain computes in an entirely different way from the conventional digital computer

Biological Neural Network

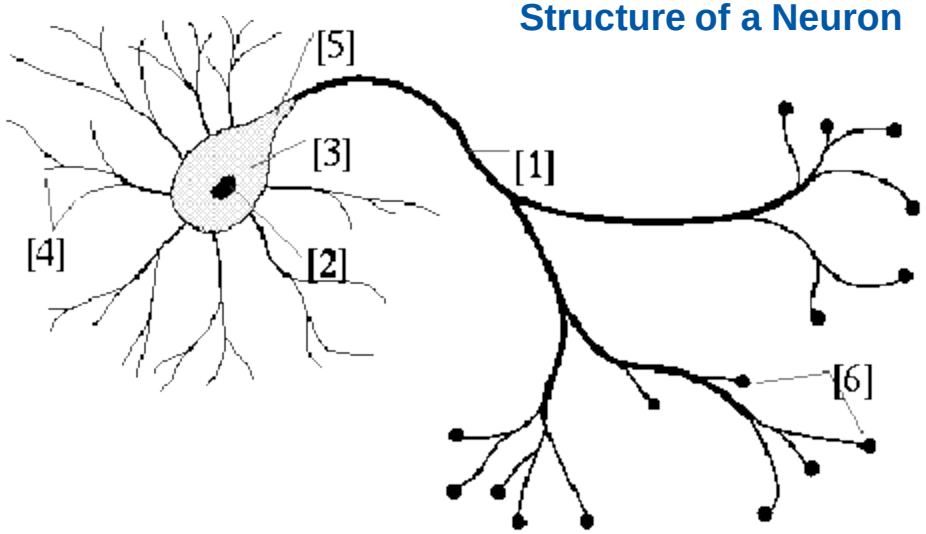


- The **human nervous system** intermediates the relationships between the *outward environment* and *the organism itself* as well as among its parts to ensure the corresponding response to external stimuli and internal states of the organism, respectively
- This process proceeds by transmitting impulses from particular sensors, so-called **receptors** which enable to receive mechanical, thermal, chemical, and luminous stimuli, to other nervous cells that process these signals and send them to corresponding executive organs, so-called **effectors**

Biological Neurons

The human brain is made of about 100 billions of neurons.

Structure of a Neuron



1.Axon 2. Nucleus 3.Soma (Body) 4. Dendrite 5. Axon Hillock 6. Terminals (Synapses)

- A neuron, (also known as a neurone or nerve cell) is an electrically excitable cell that processes and transmits information through electrical and chemical signals.

- These signals between neurons occur via synapses, specialized connections with other cells.
- Neurons can connect to each other to form neural networks
  - each neuron can be connected with about 5,000 other neurons

Some characteristics of Biological Neural Network

ANN models tries to mimic these properties of BNN

- Massive connectivity and Parallel
  - each neuron can be connected with about 5,000 other neurons
- Robust and Fault Tolerant
  - tolerance against damage to individual neurons
  - in biological systems, tolerance to loss of neurons is a high priority since a graceful degradation of performance is very important to the survival of the organism
- Capability to adapt to surroundings
  - With change in our surrounding / work conditions, our brain helps us gradually adapt to the new environment
- Ability to learn and generalize from known examples
  - Suppose you meet a friend after a long time and assume the person has changed his / her different hair style. Most of the time you will notice that the brain is will be able to recollect and map out the person, and this is kind of generalization
- Collective behavior is more important than individual behaviour
  - The biological neural network is so complex that it is meaningless to understand every single neuron and its interaction with the entire system

http://delta.cs.cinvestav.mx/~debrup/neural.ppt

Artificial NN vs Biological NN

- Artificial Neural Network aim is to be able to mimic how the human brain functions (Biological Neural Network)… however we have still a long way to go
- Some comparisons between ANN & BNN

	ANN	BNN
No. of Neurons	'000	10^9 (10 billion)
No. of Synapses	'000	$60 * 10^{12}$ (60 trillion)
Processing Speed per operation of single neuron	10^{-9} (silicon logic gates processing speed per o is in nanoseconds)	10^{-3} (milliseconds for human neuron)
Energetic Efficiency	10^{-6} joules per operation	10^{-16} joules per operation

The brain is a highly *complex*, *nonlinear*, and *parallel* information-processing system
As such the processing of brain is many times faster than the fastest digital computer

Applications of Neural Network

- Neural Network finds its applications in Artificial Intelligence and solving various business problems like:
  - Classification:
    - Pattern Recognition – Character Recognition, Face Recognition
    - Feature Extraction – Fingerprint processing
    - Image Processing – Signature matching in banks
  - Prediction: Extrapolation based on historical data
  - Noise Reduction: Recognize patterns in inputs and produce noiseless output

Simple Example of ANN

Training Data	
Input	Output
1.0	1.0
4.0	2.0
9.0	3.0
90.25	9.5
57.74	7.6
14.44	3.8
23.04	4.8
5.29	2.3
3.61	1.9
5.76	2.4
13.69	3.7

- For given Training Data we need to build a Neural Network Model
- From data the relation between Input and Output is simple Square Root Relationship
- Let us build Neural Network Model for this data… the R code for same is in:

  NeuralNet_RSqrt_Sample_Example.R

Sample Neural Net Example

Source : http://gekkoquant.com/2012/05/26/neural-networks-with-r-simple-example/

```
## un-comment below install statement if 'neuralnet' package is not installed on your machine
##install.packages('neuralnet')
library("neuralnet")

#Going to create a neural network to perform sqare rooting
#Type ?neuralnet for more information on the neuralnet library
library("neuralnet")

#Generate 50 random numbers uniformly distributed between 0 and 100
#And store them as a dataframe
traininginput <-  as.data.frame(runif(50, min=0, max=100))
trainingoutput <- sqrt(traininginput)

#Column bind the data into one variable
trainingdata <- cbind(traininginput,trainingoutput)
colnames(trainingdata) <- c("Input","Output")
head(trainingdata)
```

```
> head(trainingdata)
        Input      Output
1 14.210854541 3.769728709
2 47.938588145 6.923769793
3 90.959487134 9.537268327
4 98.237913102 9.911504079
5 90.809771162 9.529416098
6  3.029931104 1.740669729
```

Example… contd

#Train the neural network

#**formula** a symbolic description of the model to be fitted.

#**data** a data frame containing the variables specified in formula.

#**hidden** a vector of integers specifying the number of hidden neurons (vertices) in each layer.

#**threshold** is a numeric value specifying the threshold for the partial

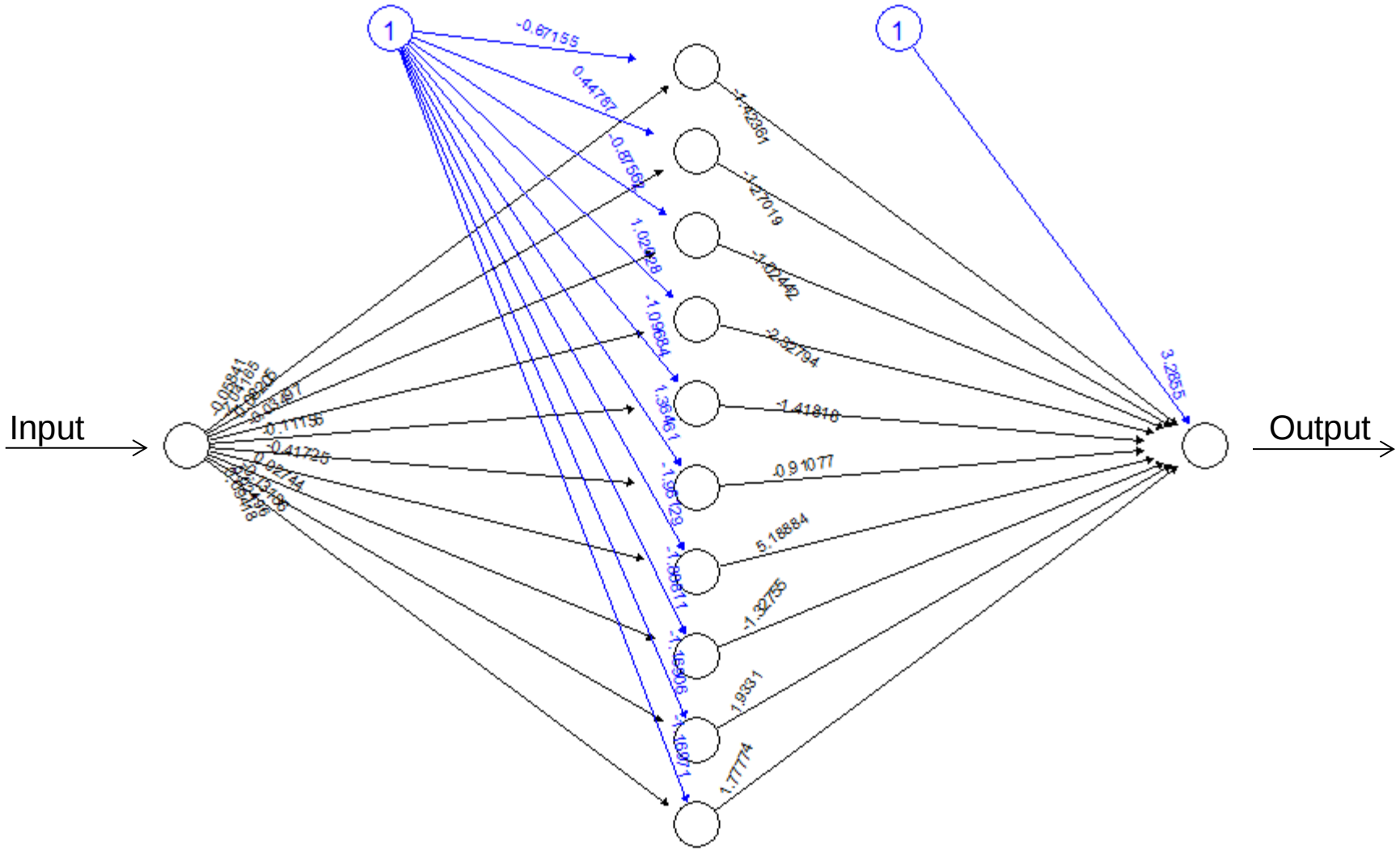
derivatives of the error function as stopping criteria.

```
net.sqrt <- neuralnet( formula = Output~Input, data = trainingdata, hidden=10, threshold=0.01 )
print(net.sqrt)
```

#Plot the neural network

```
plot(net.sqrt)
```

Plot Output for the Sqrt Example



Example… contd

#To test the neural network on some training data let us generate some squared numbers

```
testdata <- as.data.frame((1:10)^2)
```

#Run them through the neural network

##compute function is used to compute the output for a given input using the neuralnet model object

```
?compute
net.results <- compute(net.sqrt, testdata)
```

#Lets see what properties net.sqrt has and print the results

```
ls(net.results)
print(net.results$net.result)
```

#Lets display a better version of the results

```
cleanoutput <- cbind ( testdata,  sqrt(testdata),  as.data.frame(net.results$net.result) )
colnames(cleanoutput) <- c("Input", "Expected Output", "Neural Net Output" )
print(cleanoutput)
```

Why use Neural Network

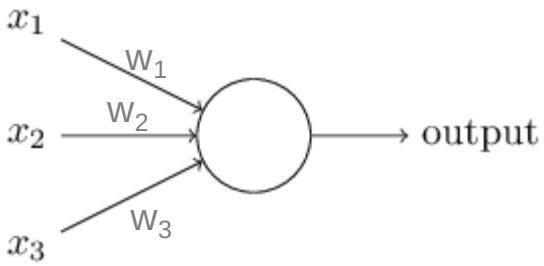
- Ability to learn
  - Neural Networks figure out how to perform their function on their own
  - Determine their function based only upon sample inputs
- Ability to generalize
  - Produce outputs for inputs it has not been taught how to deal with
- Fault tolerance
- Adaptivity – can be easily retrained to changing environmental conditions

K2 Analytics
Building Skills, Building Individuals

Perceptrons
Single / Multi Layer Neural Network
Neurons and Activation Functions
Cost Function
Backpropogation
Gradient Descent & Delta Rule
Learning Rate & Epoch

Perceptrons

- Perceptrons are a type of artificial neurons developed in 1950's – 60's
- A perceptron takes several inputs and produces a single binary output
- The perceptron output is determined by whether the weighted sum is greater than or less than a threshold value



$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

$w \cdot x \equiv \sum_j w_j x_j$ We can represent the summation as dot product, where *w* and *x* are vector of weights and inputs respectively

We can move the threshold on the left hands side of the equation and introduce a bias (b = -threshold) term and rewrite the output equation as:

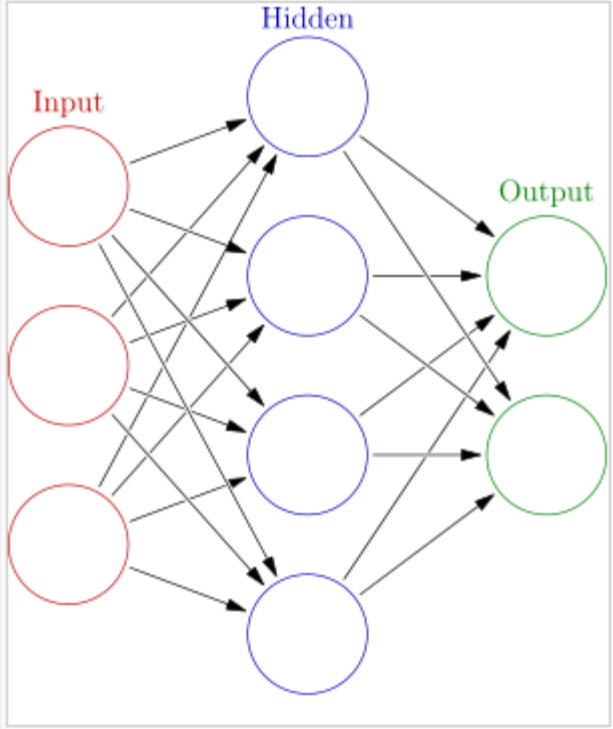
$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

http://neuralnetworksanddeeplearning.com/chap1.html

Challenges with initial neural networks

- **Initial Challenges**
  - Single layer neural network (Perceptrons) were incapable of processing Exclusive-OR circuit
  - Output of a perceptron is 0 or 1. As such they are not conducive for learning with small changes in weight (or bias) leading to small change in output. With perceptron the output may flip from 0 to 1 or vice-versa
  - Computers then (1960's - 1970's) were not powerful enough to handle the long run time required to train the neural networks
- **Advancements that led development of Neural Networks**
  - Backpropogation
  - Usage of activation functions like Sigmoid function which can take any value between 0 and 1
  - Significant increase in computer processing power

Neural Network Architecture

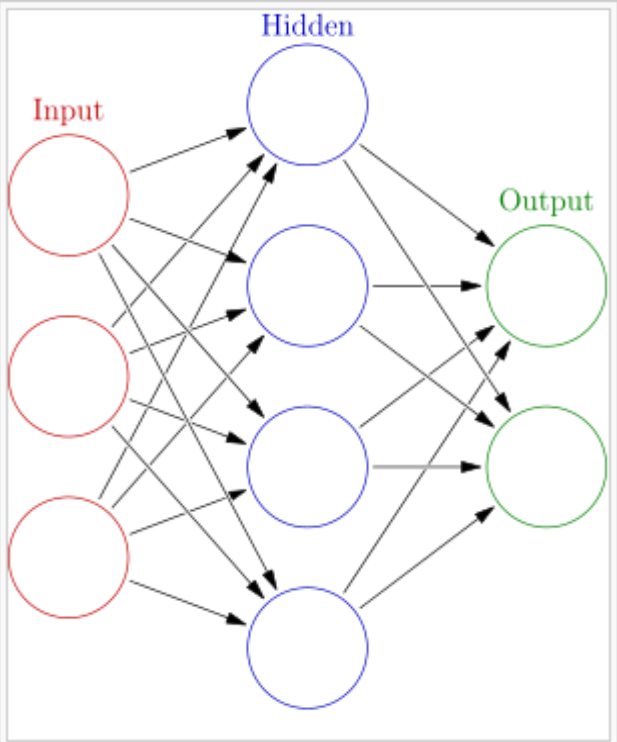


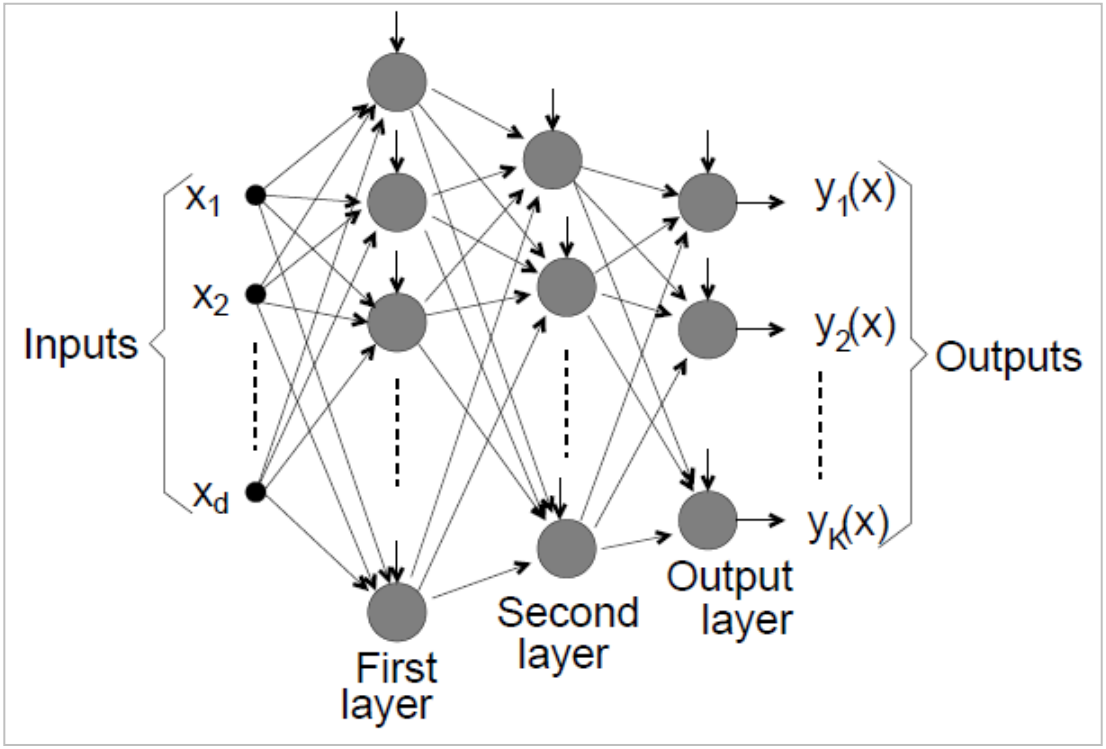
An artificial neural network is an interconnected group of nodes, akin to the vast network of neurons in a brain. Here, each circular node represents an artificial neuron and an arrow represents a connection from the output of one neuron to the input of another.

- Neural Network is made of layers with many interconnected nodes (neurons)
- There are three main layers, specifically
  - Input Layer
  - Hidden Layer
  - Output Layer
- Hidden Layer can be one or more and accordingly it is called
  - Single Layer Neural Network
  - Multi Layer Neural Network

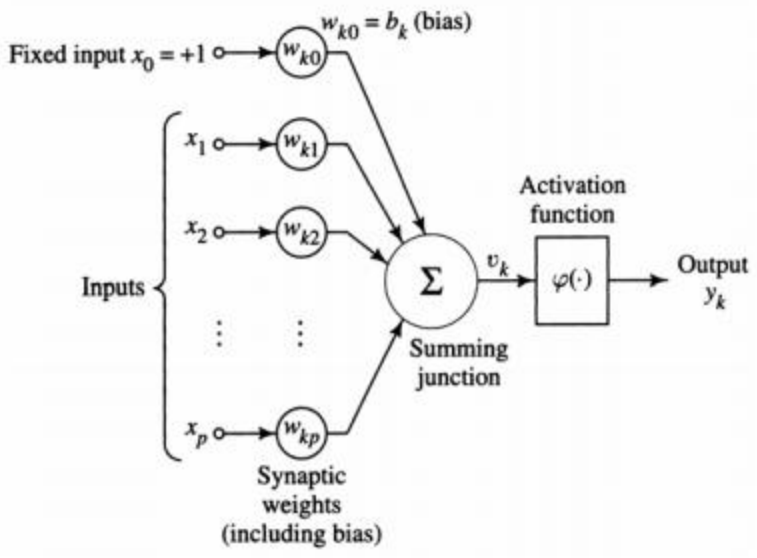
https://en.wikipedia.org/wiki/Artificial_neural_network

Single Layer & Multi Layer Network





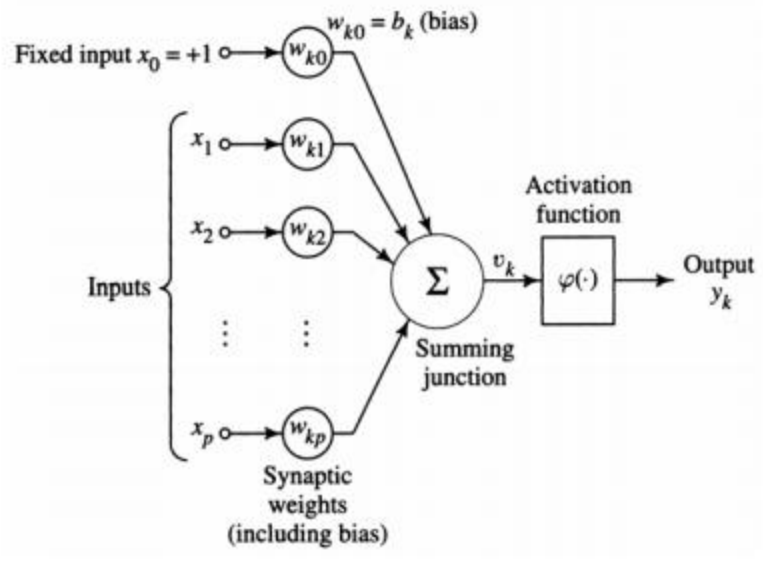
ANN - Neurons



- A neuron is an information-processing unit that is fundamental to the operation of a neural network.
- Three basic elements of the neuron model:
  - Synaptic weights
  - Combination (Addition) function
  - Activation function
- External input bias to increase or lower the net input to the Activation function

http://www.cs.ukzn.ac.za/notes/NeuralNetworks2005.pdf

How does a Neuron Work



- Neuron Output

  = fn ($\sum_{i=1}^{n} wi * xi + bias$)

- fn is some Activation Function
- Activation Function is sometime also called Transfer Function
- The output of the Activation Function is passed on to the neurons in the next layer and so on till the final output layer

Activation Function

- Activation Function is applied to the weighted sum of the inputs of a neuron to product the output
- Different types of Activation Function
  - Heaviside
  - Sigmoid (Logistic)
  - Softmax (Generalized Logistic)
  - Linear
  - Hyperbolic Tangent
- There is no theoretical reason for selecting a particular Activation Function
- The exact nature of the function has little effect on the abilities of the neural network

Different Activation Functions

Softmax (Generalized Logistic Function)

$$y_j = \frac{\exp(z_j)}{\sum_{i=1}^{n} \exp(z_i)}$$

Output is between 0 and 1 and sum of all output values is 1

Heaviside Function

$$y_j = \begin{cases} 0, & z_j < 0 \\ 1, & z_j \geq 0 \end{cases}$$

Output is 0 or 1

Hyperbolic (Tanh) Function

$$y_j = \frac{\exp(z_j) - \exp(-z_j)}{\exp(z_j) + \exp(-z_j)}$$

The output is continuous between -1 & 1

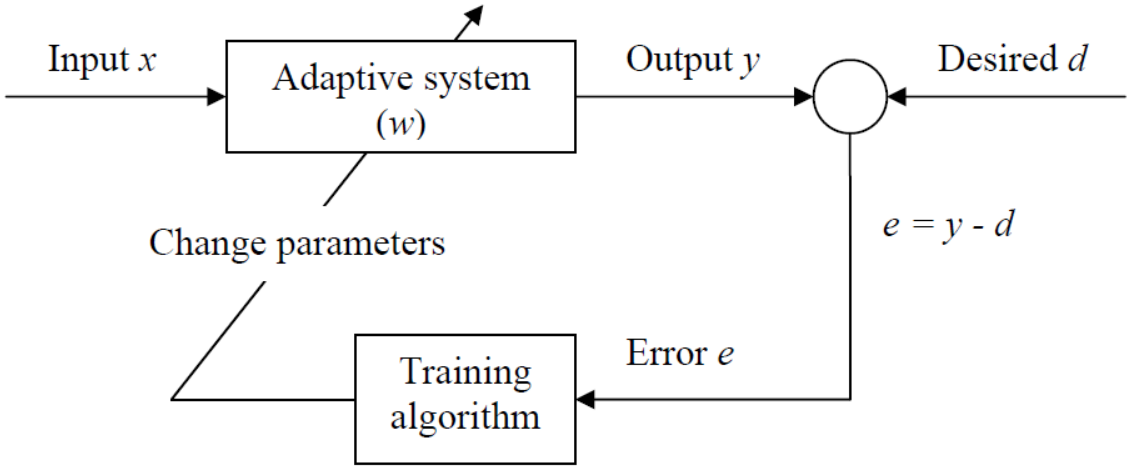
Sigmoid Function (Logistic function)

$$y_j = \frac{1}{1 + \exp(-z_j)}$$

The output is continuous between 0 & 1

How is the synaptic weights of neurons determined?

- The synaptic weights of neurons are determined based on the neural net learning process (Learning Algorithm)



- Most common measure of the Error (cost function) is mean square error E = (y – d)^2
- Iterations of the above process of providing the network with an input and updating the network's weight to reduce error is used to train the network

Cost Function

- Cost Function is a *loss function*, a function to be minimized
  - In ANN, the cost function is used to return a number to indicate how well the neural network performed to map training examples to correct output
- Desirable properties of Cost Function for ANN
  - Non-negativity
  - Cost function should tend to zero as actual output is close to the desired output
  - Globally continuous and differentiable
- Cost Functions e.g.
  - Quadratic Cost Function = $\frac{1}{2n}\sum_{i=1}^{n}(yi\ - di)$^2
  - Cross-Entropy Cost Function = $-\frac{1}{n}\sum_{i=1}^{n}(diln(yi) - (1 - di)\ ln(1 - yi))$

Backpropogation

- Backpropagation, an abbreviation for "backward propagation of errors", is a common method of training artificial neural networks used in conjunction with an optimization method such as gradient descent.
- The method calculates the gradient of a loss function with respect to all the weights in the network.
- The gradient is fed to the optimization method which in turn uses it to update the weights, in an attempt to minimize the loss function.
- Backpropagation requires a known, desired output for each input value in order to calculate the loss function gradient. It is therefore usually considered to be a supervised learning method, although it is also used in some unsupervised networks such as autoencoders.

A Step by Step Backpropagation Example

http://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

file:///D:/K2Analytics/Neural_Network/A%20Step%20by%20Step%20Backpropagation%20Example%20%E2%80%93%20Matt%20Mazur.html

K2 Analytics
Building Skills, Building Individuals

Break…

Let us resummarize
too many concepts

5 minutes settle down
Derive what you learnt in last few slides

DERIVATIVES

What is Derivatives?

- Derivative of a function *f(x)* of a variable *x* is a measure of the rate at which the value of the function changes with respect to the change of the variable
- Some Derivative Notations

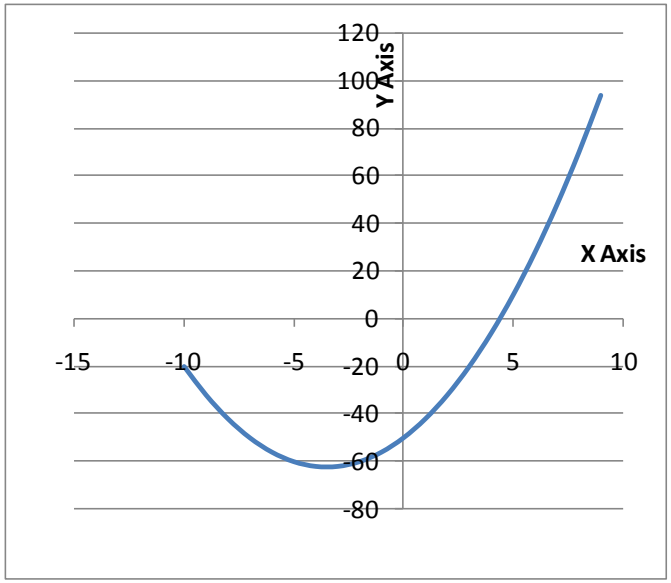
$$\frac{dy}{dx} \qquad f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}. \qquad \frac{\Delta y}{\Delta x}$$

http://www.globalspec.com/reference/62245/203279/chapter-3-calculus-and-differential-equations
https://en.wikipedia.org/wiki/Derivative

Simple e.g to Explain Neural Network Backpropogation and Gradient Descent

- Let us assume there is a cost function $f(x) = x^2 + 7x - 50$
- Below you can see the table of this cost function and the plot

X	Y
-10	50
-9	31
-8	14
-7	-1
-6	-14
-5	-25
-4	-34
-3	-41
-2	-46
-1	-49
0	-50
1	-49
2	-46
3	-41
4	-34
5	-25
6	-14
7	-1
8	14
9	31



Neural Network Gradient Descent

Let us assume randomly chosen a value
X = -8

What is the derivative of our function
f'(x) = 2x + 7

At *X = -8, f'(x) = 2 * (-8) + 7 = -16 + 7 = -11*

Assume Decay Factor = 0.1
Decay Factor is a factor by which the value of f'(x) should be multiplied to adjust X to find Minima

*New X = X – f'(x) * Decay = -8 – (-11) * 0.1*
New X = -8 + 1.1 = -7.1

Iterate above steps

For iteration and simulation with different values of Adjust see the Excel

Gradient Descent

- Gradient : means ‘a slope’ in English language
  - in physics, “an increase or decrease in the magnitude of a property (e.g. temperature, pressure, or concentration) observed in passing from one point or moment to another”.

- Descent : an act of moving downwards, dropping, or falling

- Gradient Descent: a first-order optimization algorithm.
  - To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point.
  - If instead one takes steps proportional to the positive of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent.

Delta Rule

- Delta rule is a gradient descent learning rule for updating the weights of the inputs to artificial neurons
- Delta rule states that to reduce error E by gradient descent, move / incremental weights in the negative direction to the weight

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5}$$

Reference:
A Step by Step Backpropagation Example
http://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

$$\Delta w = w - w_{old} = -\eta \frac{\partial E}{\partial w}$$

Learning Rate

- In order for the Gradient Descent to work we must set the learning rate to an appropriate value
  - Too small, we will need too many iterations for convergence
  - Too large, we may skip the optimal solution

- Adaptive Learning Rate : start with high learning rate and gradually reduce the learning rate with each iteration
  - Similar age dependent learning rates are found to exist in human beings

$$\eta(t) = \frac{\eta(1)}{t}$$

- Trial & Error – Use a range of learning rate (1, 0.1, 0.001, 0.0001) and use the results as a guide

Epoch

- An **epoch** is a measure of the number of times all of the training vectors are used once to update the weights

- Batch training: For batch training all of the training samples pass through the learning algorithm simultaneously in one epoch before weights are updated

- Sequential training - For sequential training all of the weights are updated after each training vector is sequentially passed through the training algorithm.

Importing data for building NN Model

Let us first set the working directory path

```
setwd ("D:/K2Analytics/Neural_Network/")
getwd()
```

Ideally for any modeling you should have Training & Testing dataset

Typically you would use sampling strategy

However for the Neural Net training I am supplying the Training & Testing data separately

```
nn.dev <- read.table("datafile/DEV_SAMPLE.csv", sep = ",", header = T)
nn.holdout <- read.table("datafile/HOLDOUT_SAMPLE.csv", sep = ",", header = T)
c(nrow(nn.dev), nrow(nn.holdout))
str(nn.dev)
```

Let's build the NN Model

Installing the Neural Net package; If already installed do not run the below step

```
## install.packages("neuralnet")

library(neuralnet)


nn1 <- neuralnet (

        formula = Target ~  Age + Balance  + SCR +  No_OF_CR_TXNS + Holding_Period ,
        data = nn.dev,        hidden = 2,        err.fct = "sse",      linear.output = FALSE,
        lifesign = "full",      lifesign.step = 10,                    threshold = 0.1,
        stepmax = 2000

        )
```

There is not much gradation in predicted probabilities

The distribution of the estimated results

```
quantile( nn1$net.result[[1]],  c(0,1,5,10,25,50,75,90,95,99,100) / 100 )
```

```
          0%           1%           5%          10%          25%          50%
0.08696514658 0.08696514658 0.08696514658 0.08696514658 0.08696514658 0.08696514658
         75%          90%          95%          99%         100%
0.08696514658 0.08696514658 0.08696514658 0.15966895460 0.15966895460
```

Scaling Variables

- Let us scale the variable and see if it has an impact on neural net model output
- **Theoretically** scaling should not make a difference, it just means that the optimal weights will be scaled
- **In practice** it could make a difference,
  - because floating point representations are not exact, i. e. when you have huge input, your weights will be very small and a little change maybe could not be represented
  - On the other hand, you usually have sigmoid activation functions, which tend to saturate for large inputs and then will only adjust slowly during training
  - That means, scaling your data often accelerates training.

No	Ln(No)	Rel. Diff
1	0	
2	0.693147	0.693147
3	1.098612	0.405465
4	1.386294	0.287682
5	1.609438	0.223144
6	1.791759	0.182322
7	1.94591	0.154151
8	2.079442	0.133531
9	2.197225	0.117783
10	2.302585	0.105361
11	2.397895	0.09531
12	2.484907	0.087011
13	2.564949	0.080043
14	2.639057	0.074108
15	2.70805	0.068993
16	2.772589	0.064539
17	2.833213	0.060625
18	2.890372	0.057158
19	2.944439	0.054067
20	2.995732	0.051293
21	3.044522	0.04879

…NN with scaling

build the neural net model by scaling the variables

```
x <- subset(nn.dev, select = c("Age","Balance", "SCR", "No_OF_CR_TXNS", "Holding_Period"))

nn.devscaled <- scale(x)

nn.devscaled <- cbind(nn.dev[2], nn.devscaled)

View(nn.devscaled)

nn2 <- neuralnet (

            formula = Target ~  Age + Balance  + SCR +  No_OF_CR_TXNS + Holding_Period ,
            data = nn.devscaled,    hidden = 3,        err.fct = "sse",       linear.output = FALSE,
            lifesign = "full",       lifesign.step = 10,     threshold = 0.1,       stepmax = 2000

             )

nn2

plot(nn2)

quantile(nn2$net.result[[1]], c(0,1,5,10,25,50,75,90,95,99,100)/100)
```

```
             0%              1%              5%             10%             25%             50%
0.008873901954  0.010797206952  0.013215366438  0.015978694495  0.027729559618  0.059384864542
            75%             90%             95%             99%            100%
0.119498149281  0.206988830393  0.265576097885  0.361954272588  0.448515452933
```

Model Performance | Classification Accuracy

```
misClassTable = data.frame(Target = nn.devscaled$Target,
                           Predict.score = nn2$net.result[[1]] )
```

SSE Error Computation **sum((misClassTable$Target - misClassTable$Predict.score)^2)/2**

```
misClassTable$Predict.class = ifelse(misClassTable$Predict.score>0.21,1,0)
with(misClassTable, table(Target, Predict.class))
```

Target	Predict.class 0	1
0	11758	1007
1	835	400

Mis Classification – 1821
Mis Classifcaiton Error = 1821 / 14000 = 13%
Classification Accuracy = 87%

Theoretically speaking we would have been better off had we classified all cases as 0. The Mis-Classification would be only 1235 cases (8.8%)

What will be the proportional by chance Classification Error (Classification Accuracy)?
(i.e. Accuracy if records are classified in proportion based on random selection)

Model Performance | Rank Ordering

```
## deciling function
decile <- function(x){
  deciles <- vector(length=10)
  for (i in seq(0.1,1,.1)){
    deciles[i*10] <- quantile(x, i, na.rm=T)
  }
  return (
  ifelse(x<deciles[1], 1,
  ifelse(x<deciles[2], 2,
  ifelse(x<deciles[3], 3,
  ifelse(x<deciles[4], 4,
  ifelse(x<deciles[5], 5,
  ifelse(x<deciles[6], 6,
  ifelse(x<deciles[7], 7,
  ifelse(x<deciles[8], 8,
  ifelse(x<deciles[9], 9, 10
  ))))))))))
}

## deciling
misClassTable$deciles <-
decile(misClassTable$Predict.score)
```

```
## Ranking code
library(data.table)
tmp_DT = data.table(misClassTable)
rank <- tmp_DT[, list(
  cnt = length(Target),
  cnt_resp = sum(Target),
  cnt_non_resp = sum(Target == 0)) ,
  by=deciles][order(-deciles)]
rank$rrate <- round (rank$cnt_resp * 100 / rank$cnt,2);
rank$cum_resp <- cumsum(rank$cnt_resp)
rank$cum_non_resp <- cumsum(rank$cnt_non_resp)
rank$cum_rel_resp <- round(rank$cum_resp /
sum(rank$cnt_resp),2);
rank$cum_rel_non_resp <- round(rank$cum_non_resp /
sum(rank$cnt_non_resp),2);
rank$ks <- abs(rank$cum_rel_resp -
rank$cum_rel_non_resp);
```

Model Performance | Rank Order Table

View(rank)

deciles	cnt	cnt_resp	cnt_non_resp	rrate	cum_resp	cum_non_resp	cum_rel_resp	cum_rel_non_resp	ks
10	1400	400	1000	28.57	400	1000	0.32	0.08	0.24
9	1400	218	1182	15.57	618	2182	0.50	0.17	0.33
8	1400	163	1237	11.64	781	3419	0.63	0.27	0.36
7	1400	123	1277	8.79	904	4696	0.73	0.37	0.36
6	1400	90	1310	6.43	994	6006	0.80	0.47	0.33
5	1400	94	1306	6.71	1088	7312	0.88	0.57	0.31
4	1400	63	1337	4.50	1151	8649	0.93	0.68	0.25
3	1400	47	1353	3.36	1198	10002	0.97	0.78	0.19
2	1400	26	1374	1.86	1224	11376	0.99	0.89	0.10
1	1400	11	1389	0.79	1235	12765	1.00	1.00	0.00

Scoring

Scoring another dataset using the Neural Net Model Object

To score we will use the compute function

```
?compute
x <- subset( nn.holdout,
             select = c("Age", "Balance", "SCR", "No_OF_CR_TXNS", "Holding_Period")
             )
x.scaled <- scale(x)
compute.output = compute(nn2, x.scaled)
nn.holdout$Predict.score = compute.output$net.result
View(nn.holdout)

quantile(nn.holdout$Predict.score, c(0,1,5,10,25,50,75,90,95,99,100)/100)
```

Overfitting

- Overfitting – Neural Network Models are susceptible to overfitting

- Main causes being
  - Large number of weights and biases
  - Excessive learning (Epocs) on training data

- Ways to avoid Overfitting
  - Increase sample size
  - Early stopping
  - Reduce Network Size
  - Regularization

Regularization

- Regularization in Machine Learning refers to a process of introducing additional information to prevent the problem of overfitting
- The mechanics of Regularization to prevent overfitting is by way of penalizing models with extreme weights

- Types of Regularization
  - L1 : LASSO -> Cost function change is $C = C_0 + \lambda |w|$
  - L2 : Ridge -> Cost function change is $C = C_0 + \frac{\lambda}{2n} \sum_w w^2$

  Note:

  The cost function is changed such that it will penalize large weights

  In Machine Learning techniques we will use Ridge norm as it is differentiable

https://www.quora.com/What-is-regularization-in-machine-learning

Regularization…contd

- λ in the previous equation is called Decay Factor
- In Neural Network, the L2 regularization may also be called "weight decay"

- Optimal value of λ is determined through cross-validation
  - Build model on training set
  - Validate mode on testing set
  - Iterate for different values of λ (Decay Factor)
  - Finally select the decay factor where you get optimum

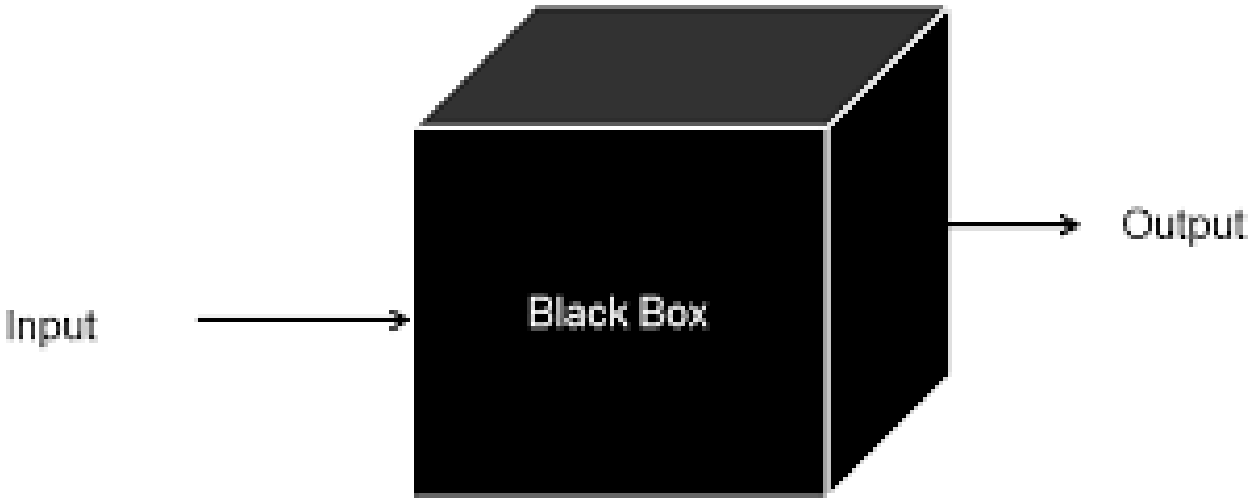
Some of the options like Decay Factor, No. of Epoch cannot be directly set in "**neuralnet**" package in R… or probably I am unware of it at time of making this presentation

Art in Neural Net Model building

- Deciding the number of hidden layers
- Deciding the number of neurons in hidden layers
  - For single layer neural network, you may choose to set the number of neurons as square root of number of variables
- Deciding the decay factor
- Deciding the Number of Epoch
- Deciding the Learning Rate
- Deciding the Activation Function
- Ensuring that the model is not overfitting

In practice, it is advisable to use variable reduction techniques before starting with Neural Network Model

Drawback (Strength) of Neural Network Model



- It does not provide you with Business Insights

The drawback of Neural Network Model is also its greatest strengths
… it does away with the process of trend fitting
… it can be easily refreshed & retrained
… easy process automation
thereby enabling deployed in Decision Logic Unit / Recommendation Engine

K2 Analytics
Building Skills, Building Individuals

Thank you

Email Id : ar.jakhotia@k2analytics.co.in

Earning is in Learning
- Rajesh Jakhotia