cct | College Dublin
Computing • IT • Business

Machine Learning for Data Analysis
MSc in Data Analytics
CCT College Dublin

Artificial Neural Networks (ANN)
Week 7

Lecturer: Dr. Muhammad Iqbal*
Email: miqbal@cct.ie

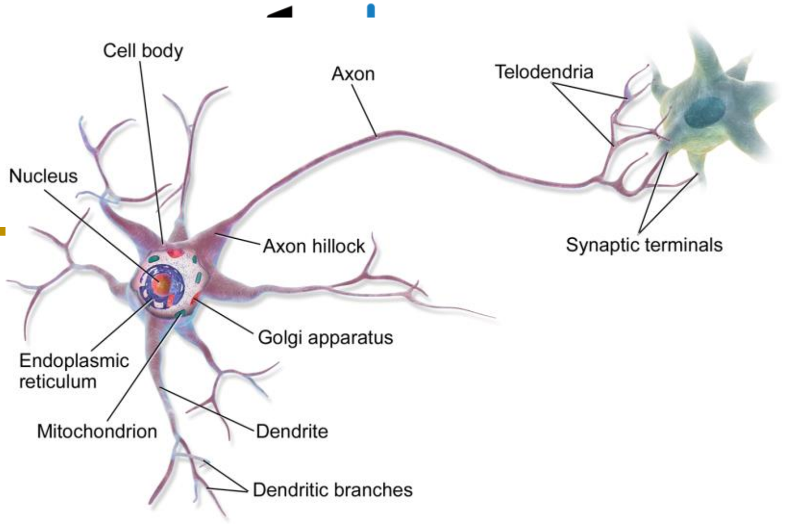
Agenda

- Introduction to Neural Networks
- Single Layer NN: Perceptron
- Why Neural Networks?
- Neural Network Architecture
- Types of Artificial Neural Networks
- Neural Network: Example
- Perceptron Learning Rule
- Gradient Descent Method
- Characteristics of ANN

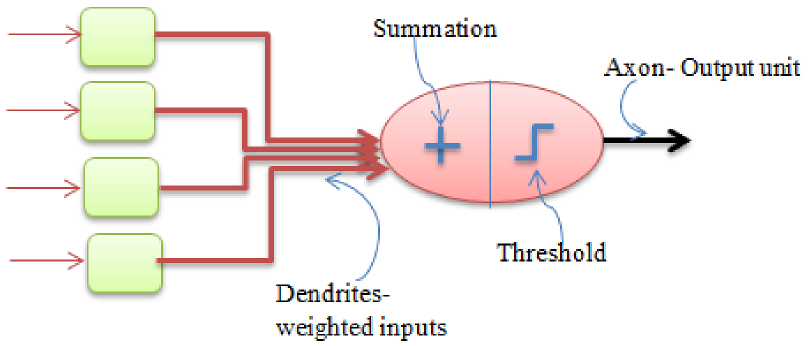
Introduction to Neural Networks

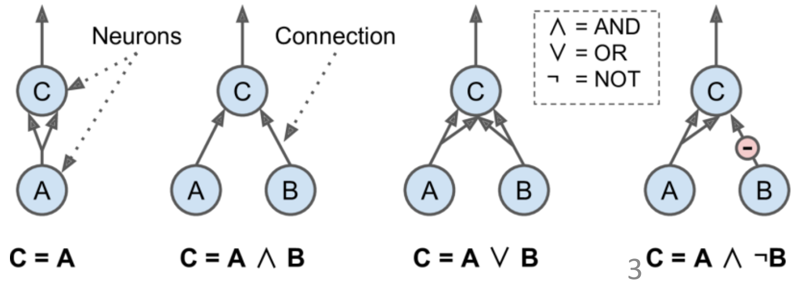
- **Neural Networks**
  - Complex learning systems recognized in animal brains
  - Single neuron has simple structure
  - Interconnected sets of neurons perform complex learning tasks
  - Human brain has approximately **10^{15} synaptic connections**
  - **Artificial Neural Networks** attempt to replicate non-linear learning found in nature
  - Dendrites gather inputs from other neurons and combine information
  - Further generate non-linear response when threshold reached
  - Signal sent to other neurons via axon
  - Artificial neuron model is similar
  - Data inputs (x_i) are collected from upstream neurons input to combination function (sigma)

Biological neuron



Structure of an Artificial Neuron

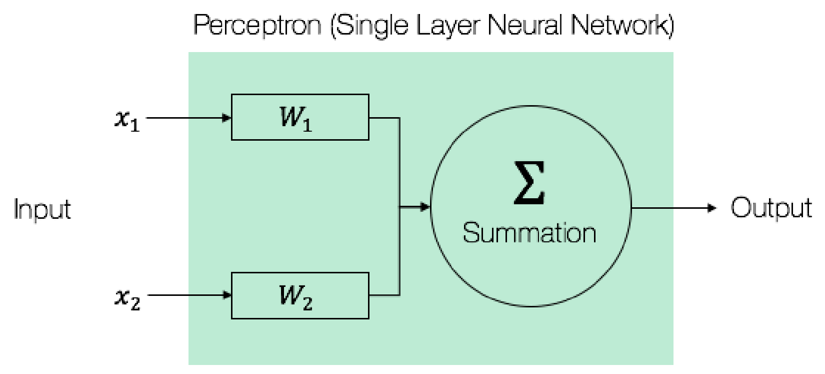




Introduction to Neural Networks

- Neural networks are a class of machine learning algorithms that are loosely inspired by neurons in the human brain.
- We describe neural networks as a mathematical function that maps a given input to the desired output.
- To understand this, let's take a look at a single layer neural network (known as a perceptron).
- **Perceptron is simply a mathematical function** that takes in a set of inputs, performs some mathematical computation, and outputs the result of the computation.
- ***w*$_i$** refers to the weights of the Perceptron. We explain what the weights in a neural network refers to in the next few slides.
- **Neural networks** are simply mathematical functions that map a given input to a desired output.

A Perceptron can be illustrated with the following diagram.



In this case, the mathematical function is simply as

$$y = \sum (w_i * x_i)$$

Why Neural Networks?

- Before creating our own neural network, it is worth understanding why neural networks have gained such an important foothold in machine learning, data analytics and other fields. The first reason is that **neural networks are universal function approximators**.

- Assuming that any problem in the world can be described by a mathematical function (no matter how complex), we can use neural networks to represent that function.

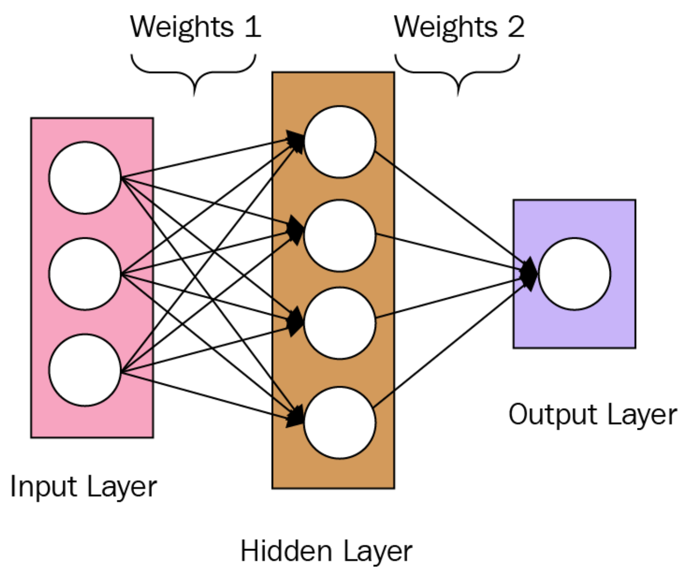
- The second reason is that the architecture of neural networks are highly scalable and flexible. We can increase or decrease the complexity of the neural network.

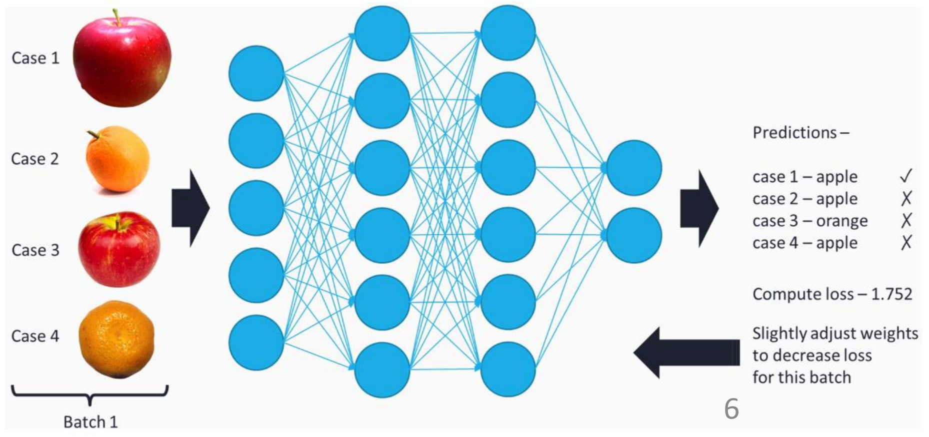
- Machine learning engineers have learned how to use neural networks to predict time series data (known as recurrent neural networks (RNNs)), which are used in the areas such as speech recognition.

- In recent years, scientists have also shown that by pitting two neural networks against each other in a contest (known as a **generative adversarial network (GAN)**), we can generate photorealistic images that are indistinguishable to the human eye.

Neural Network Architecture

- We look at the basic architecture of neural networks, the building blocks on which all complex neural networks are based. Neural networks consist of the following components
  - *An input layer (x)*
  - ***An arbitrary amount of hidden layers***
  - ***An output layer (ŷ)***
  - ***A set of weights and biases between each layer (W and b)***
  - ***A choice of activation function for each hidden layer (σ)***
- The diagram shows the architecture of a two-layer neural network (note that the input layer is typically excluded when counting the number of layers in a neural network).





Artificial Neural Networks

Types

- **FeedForward ANN**

- The information flow is unidirectional. A unit sends information to other unit from which it does not receive any information. There are no feedback loops. They are used in pattern generation/ recognition/ classification. They have fixed inputs and outputs.

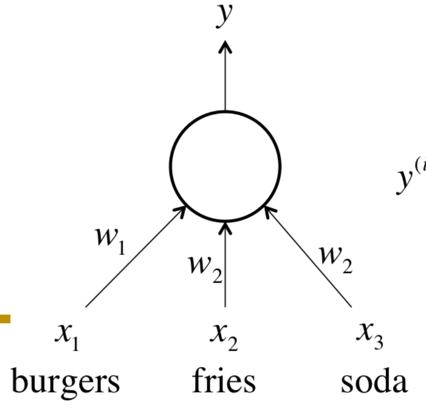
- **FeedBack ANN**

- Feedback loops are also allowed. They are used in content addressable memories.

- **Working of ANNs**

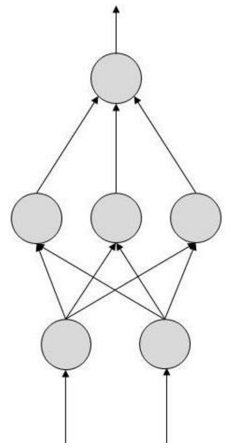
- In the topology diagrams shown, each arrow represents a connection between two neurons and indicates the pathway for the flow of information. Each connection has a weight, an integer number that controls the signal between the two neurons.

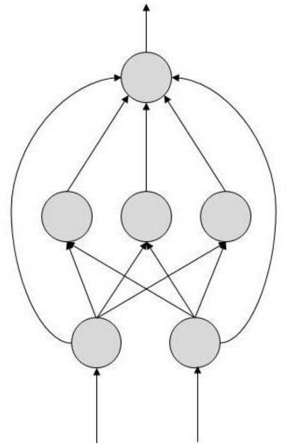
- If the network generates a "**good or desired**" output, there is no need to adjust the weights. However, if the network generates a "**poor or undesired**" output or an error, then the system <u>**alters the weights**</u> in order to improve subsequent results.

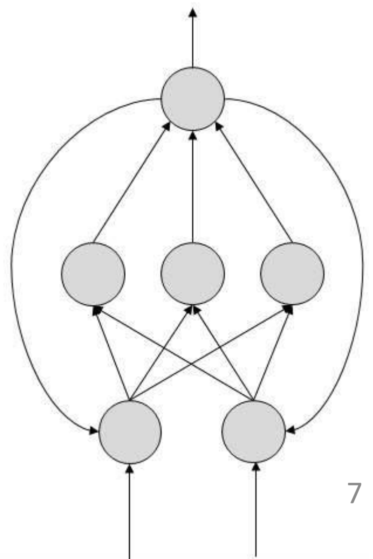


$$y^{(i)} = w_1 x_1^{(i)} + w_2 x_2^{(i)} + w_3 x_3^{(i)}$$

This is the neuron we want to train for the fast-food problem

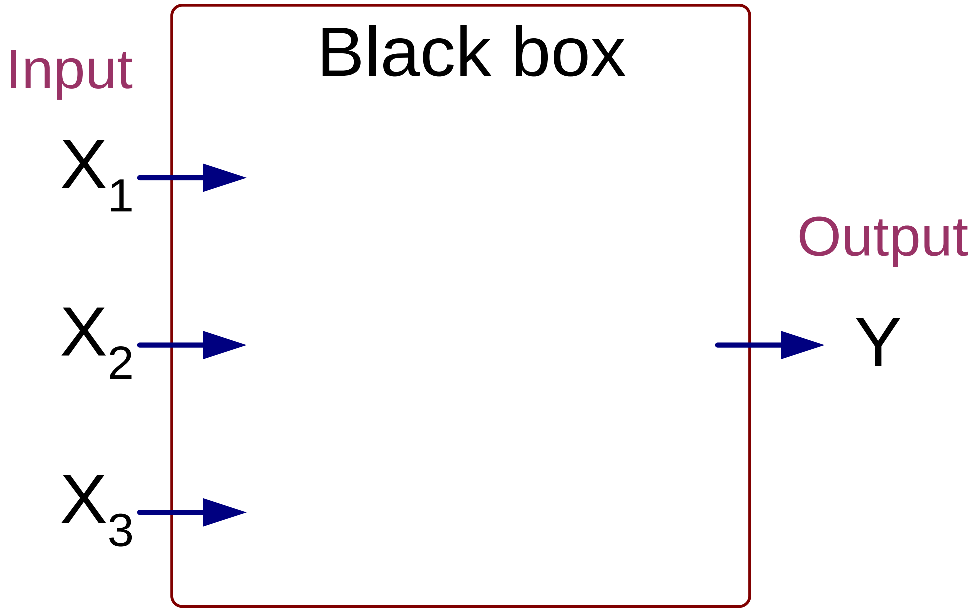






Artificial Neural Networks (ANN)

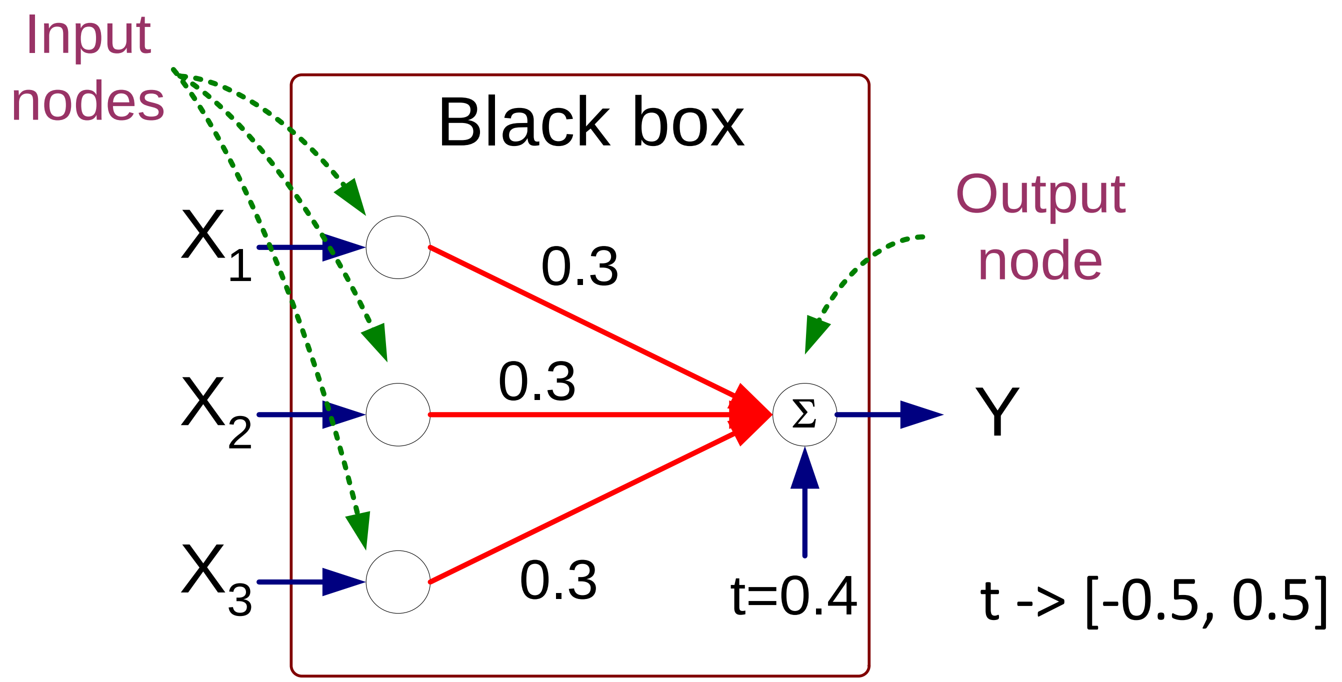
X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



Output Y is 1 if at least two of the three inputs are equal to 1.

Artificial Neural Networks (ANN)

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



$$Y = sign(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where} \quad sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Single Layer NN

Perceptron

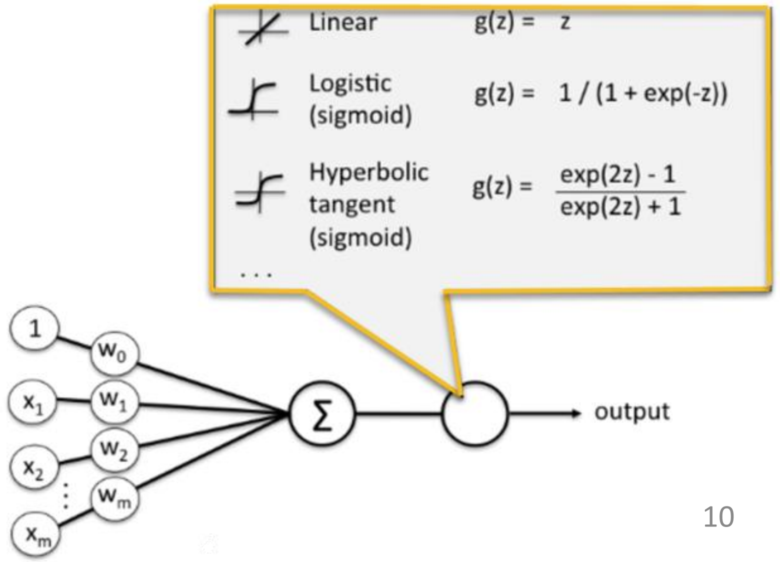
- **Single layer network**
  - Contains only input and output nodes
- Activation function: $f = sign(\mathrm{w.x})$
- Applying model is straight forward

$$Y = sign\,(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where} \quad sign\,(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

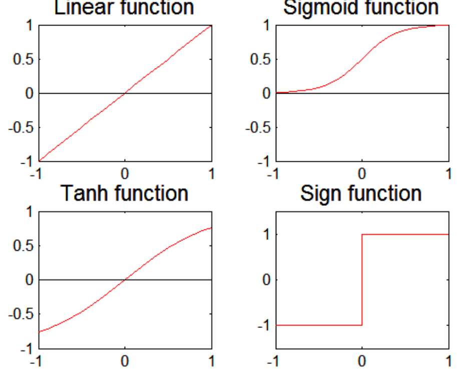
- $X_1 = 1$, $X_2 = 0$, $X_3 = 1$ => **y =** $\boldsymbol{sign(0.2) = 1}$

- ***Activation functions*** have an important role to play in neural networks.
- You can think of ***activation functions*** as transformers in neural networks; they take an input value, transform the input value, and pass the transformed value to the next layer.



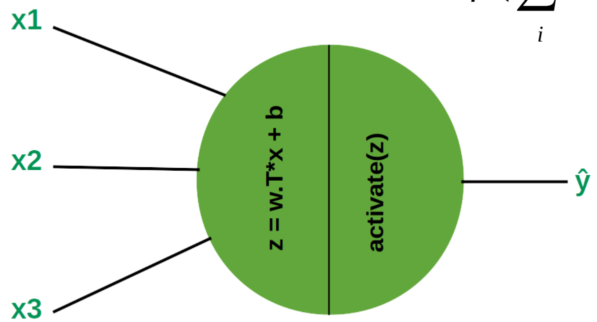
Neural Networks

Activation Function



- Activation function reads combined input and produces non-linear response (y)

$$Y = f\left(\sum_i w_i X_i\right)$$



- **What is an Activation function?**
  - The activation function is a mathematical "gate" in between the input feeding the current neuron and its output going to the next layer. They basically decide whether the neuron should be activated or not.
- **Why do we use an activation function?**
  - If we do not have the activation function, the weights and bias would simply do a linear transformation. A linear equation is simple to solve but is limited in its capacity to solve complex problems and have less power to learn complex functional mappings from data. **A neural network without an activation function is just a linear regression model.**

- Generally, neural networks use non-linear activation functions, which can help the network learn complex data, compute and learn and provide accurate predictions.
- **Types of Activation Function**
  - ***Linear or Identity*** Activation Function
  - ***Non-linear*** Activation Functions

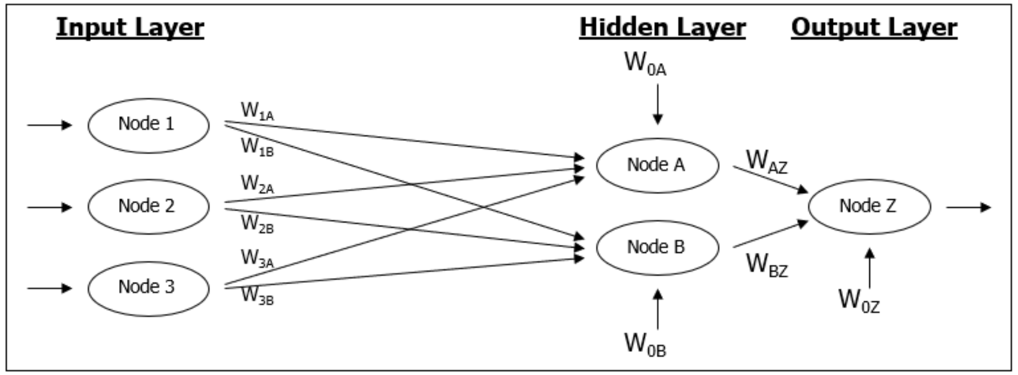
Perceptron Learning Rule

1. Initialize the weights ($\mathbf{w_0}$, $\mathbf{w_1}$, …, $\mathbf{w_d}$)
2. For each training example **(x_i, y_i)**
   - Compute ***f***(**w, x_i**)
   - ***Error (e)* = *y_i* – *f*(w, x_i)**
     - If ***y* = *f*(x, w)**, ***e* = 0 or tolerance: 0.0001** : no update needed and stop the process
     - Else If ***y* > *f*(x, w)**, ***e* = positive value:** weights (w's) must be increased, so that ***f*(x, w)** will increase
     - Else If ***y* < *f*(x, w)**, ***e* = negative value:** weights (w's) must be decreased, so that ***f*(x, w)** will decrease
     - Update the weights by using the following formula
     - $w^{(k+1)} = w^{(k)} + \alpha\,[y_i - f(w, x_i)]x_i$ where α is the learning rate.

- Repeat step 2 until stopping condition is met based on your tolerance.

Neural Network
Example

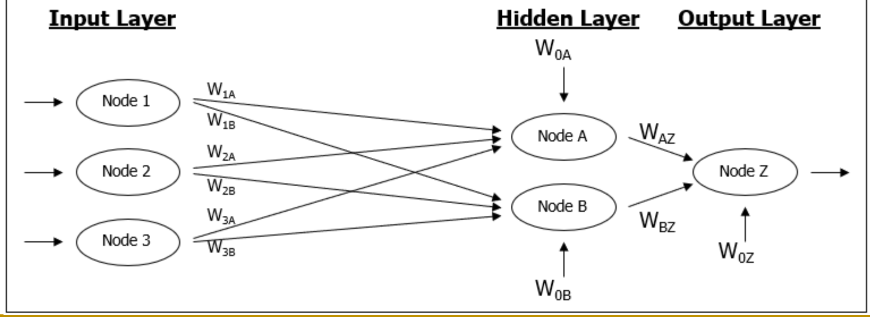
- Neural Network consists of **layered**, **feedforward**, **completely connected** network of nodes
- Feedforward restricts network flow to single direction
- Flow does not loop or cycle
- Network composed of two or more layers
- Most networks have **Input**, **Hidden**, **Output** layers
- Network may contain more than one hidden layer
- Network is completely connected



- Each node in given a layer, connected to every node in the next layer
- Every connection has weight (W_{ij}) associated with it
- Weight values randomly assigned from **0 to 1** by algorithm
- Number of input nodes dependent on number of predictors
- Number of hidden and output nodes configurable

Neural Network

Example



- **Hidden Layer**
  - How many nodes in hidden layer?
  - Large number of nodes increases complexity of model
  - Detailed patterns uncovered in data
  - Leads to overfitting, at the expense of generalizability
  - Reduce number of hidden nodes when overfitting occurs
  - Increase number of hidden nodes when training accuracy unacceptably low

- **Input Layer**
  - Input layer accepts values from input variables
  - Values passed to hidden layer nodes
  - Input layer nodes lack detailed structure compared to hidden and output layer nodes

- Combination function produces linear combination of node inputs and connection weights to single scalar value
- For a given node j: $\text{net}_j = \sum_i W_{ij} x_{ij} = W_{0j} x_{0j} + W_{1j} x_{1j} + \ldots + W_{Ij} x_{Ij}$

where

- x_{ij} is i*th* input to node j, W_{ij} is weight associated with i*th* input node and there are i + 1 inputs to node j
- $x_1, x_2, \ldots, x_i$ are inputs from upstream nodes and x_0 is constant input value = 1.0
- Each input node has extra input $W_{0j}x_{0j} = W_{0j}$

Neural Network

Example

- **Example – Using values from table 1**
  - The scalar value computed for hidden layer Node A equals

$$\text{net}_A = \sum_i W_{iA}\, x_{iA} = W_{0A}x_{0A} + W_{1A}x_{1A} + W_{2A}x_{2A} + W_{3A}x_{3A}$$
$$= 0.5(1.0) + 0.6(0.4) + 0.8(0.2) + 0.6(0.7) = 1.32$$

  - For **Node A**, net$_A$ = 1.32 is the input to activation function
  - This activation is analogous to how neurons "fire" nonlinearly in biological organisms

Table 1: Data inputs and initial values for neural network weights

$x_0 = 1.0$	$W_{0A} = 0.5$	$W_{0B} = 0.7$	$W_{0Z} = 0.5$
$x_1 = 0.4$	$W_{1A} = 0.6$	$W_{1B} = 0.9$	$W_{AZ} = 0.9$
$x_2 = 0.2$	$W_{2A} = 0.8$	$W_{2B} = 0.8$	$W_{BZ} = 0.9$
$x_3 = 0.7$	$W_{3A} = 0.6$	$W_{3B} = 0.4$	

- Firing response **not necessarily linearly related** to increase in input stimulation
- Artificial Neural Networks model behavior using non-linear activation function
- **Sigmoid function** most commonly used

$$y = \frac{1}{1 + e^{-x}}$$

- In **Node A**, sigmoid function takes net$_A$ = 1.32 as input and produces output

$$f(net_A) = y = \frac{1}{1 + e^{-1.32}} = 0.7892$$

Neural Network

Example

Table 1: Data inputs and initial values for neural network weights

$x_0 = 1.0$	$W_{0A} = 0.5$	$W_{0B} = 0.7$	$W_{0Z} = 0.5$
$x_1 = 0.4$	$W_{1A} = 0.6$	$W_{1B} = 0.9$	$W_{AZ} = 0.9$
$x_2 = 0.2$	$W_{2A} = 0.8$	$W_{2B} = 0.8$	$W_{BZ} = 0.9$
$x_3 = 0.7$	$W_{3A} = 0.6$	$W_{3B} = 0.4$	

- **Node A** outputs 0.7892 along connection to **Node Z**, and becomes component of **net$_Z$**
- Before **net$_Z$** is computed, the contribution from **Node B** is required

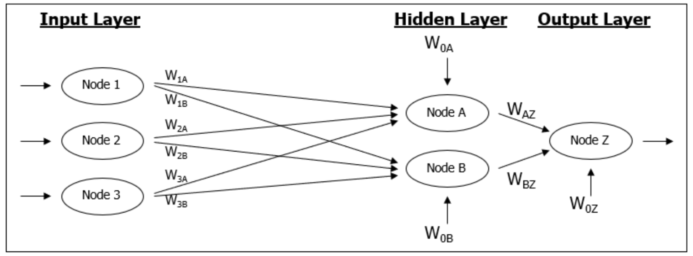
$$\text{net}_B = \sum_i W_{iB}\, x_{iB} = W_{0B}x_{0B} + W_{1B}x_{1B} + W_{2B}x_{2B} + W_{3B}x_{3B} = 0.7(1.0) + 0.9(0.4) + 0.8(0.2) + 0.4(0.7) = 1.5$$

- then

$$f(\text{net}_B) = \frac{1}{1 + e^{-1.5}} = 0.8176$$

- **Node Z** combines outputs from **Node A** and **Node B**, through **net$_Z$**, a weighted sum, using weight associated to the connections between nodes

- Inputs to **Node Z** not data attribute values
- Rather, outputs are from sigmoid function in upstream nodes



$$\text{net}_Z = \sum_i W_{iZ}\, x_{iZ} = W_{0Z}x_{0Z} + W_{AZ}x_{AZ} + W_{BZ}x_{BZ} = 0.5(1.0) + 0.9(0.7892) + 0.9(0.8176) = 1.9461$$

- then

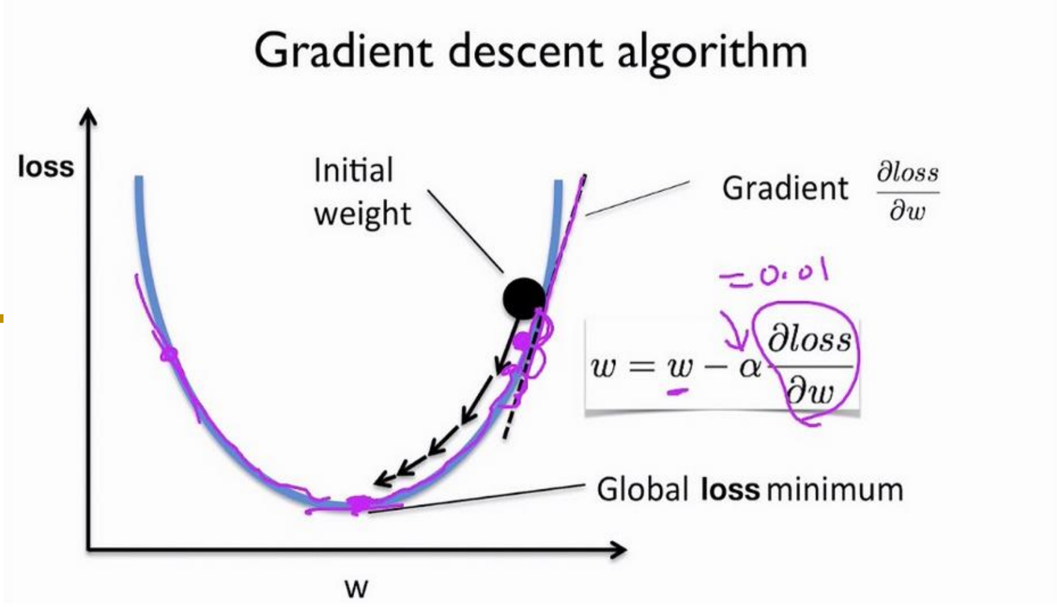
$$f(\text{net}_z) = \frac{1}{1 + e^{-1.9461}} = 0.8750$$

- Value 0.8750 output from Neural Network on first pass
- Represents predicted value for target variable, given first observation

Learning
Multi-layer Neural Network

- **Can we apply perceptron learning rule to each node, including hidden nodes?**
  - Perceptron learning rule computes error term ***e* = *y* - *f*(*w*, *x*)** and updates weights accordingly
    - **Problem:** How to determine the true value of ***y*** for hidden nodes?
  - Approximate error in hidden nodes by error in the output nodes
    - **Problem:**
      - Not clear how adjustment in the hidden nodes affect overall error
      - No guarantee of convergence to optimal solution

Gradient Descent Method


Gradient descent algorithm
loss
Initial weight
Gradient $\frac{\partial loss}{\partial w}$
$= 0.01$
$w = w - \alpha \frac{\partial loss}{\partial w}$
Global loss minimum
w

Formula: new weight = prev. weight — learning rate*gradient

- **Gradient descent** is an iterative approach for error correction in any learning model. For neural networks during backpropagation, the process of iterating the update of weights and biases with the error times derivative of the activation function is the gradient descent approach.
- Gradient is basically defined as the slope of the curve and is the derivative of the activation function.
- It's based on a convex function and tweaks its parameters iteratively to minimize a given function to its local minimum.
- **Gradient descent** is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function.
- To find a local minimum of a function using gradient descent, we take steps proportional to the negative of the gradient of the function at the current point.

Gradient Descent for Multilayer NN

- Weight update: $$w_j^{(k+1)} = w_j^{(k)} - \alpha \frac{\partial E}{\partial w_j}$$
- Error function: $$E = \frac{1}{2}\sum_{i=1}^{N}\left(t_i - f\left(\sum_j w_j x_{ij}\right)\right)$$
- Activation function ***f*** must be differentiable
- For sigmoid function:

$$w_j^{(k+1)} = w_j^{(k)} + \alpha \sum_i (t_i - o_i) o_i (1 - o_i) x_{ij}$$

- Stochastic gradient descent (update the weight immediately)

Termination Criteria

- Many passes through data set performed before termination criterion is met
- Constantly adjusting weights to reduce prediction error
- When to terminate?
- **Stopping criterion may be computational “clock” time?**
- Short training times likely result in poor model
- **Terminate when SSE reaches threshold level?**
- Neural Networks are prone to overfitting
- Memorizing patterns rather than generalizing

- **Cross-Validation Termination Procedure**
  - Retain portion of training set as “hold out” data set
  - Train network on remaining data
  - Apply weights learned from training set to validation set
  - Measure two sets of weights
  - “Current” weights for training set, “Best” weights with minimum SSE on validation set
  - Terminate algorithm when current weights has significantly greater SSE than best weights
  - However, Neural Networks not guaranteed to arrive at global minimum for SSE

Learning Rate and Dropout Rate

- Recall Learning Rate (Greek “eta”) is a constant

$$0 < \eta < 1, \text{ where}$$
$$\eta = \text{learning} \quad \text{rate}$$

- Helps adjust weights toward global minimum for SSE
- **Small Learning Rate**
  - With small learning rate, weight adjustments small
  - Network takes unacceptable time converging to solution
- **Large Learning Rate**
  - Suppose algorithm close to optimal solution
  - With large learning rate, network likely to “overshoot” optimal solution

- In order to keep neural networks away from overfitting
- Regularization techniques like the dropout rate are employed.
- A model is said to be overfited when it learns the training set too thoroughly, catching noise and unimportant patterns that do not transfer well to new data.
- Dropout effectively reduces the capacity or complexity of the network by randomly dropping out (setting to zero) a portion of the neurons in a layer during training.

```
model = Sequential([
    Dense(128, activation='relu', input_shape=(784,)),
    Dropout(0.1),  # Set the dropout rate to 0.1
    Dense(64, activation='relu'),
    Dropout(0.1),  # Set the dropout rate to 0.1
    Dense(10, activation='softmax')
])
```

Characteristics of ANN

- Multilayer ANN are universal approximators but could suffer from overfitting if the network is too large (Several layers)
- Gradient descent may converge to local minimum
- Model building can be very time consuming, but testing can be very fast
- Can handle redundant attributes because weights are automatically learnt
- Sensitive to noise in training data
- Difficult to handle missing attributes

Recent Noteworthy Developments in ANN

- Use in deep learning and unsupervised feature learning
  - Seek to automatically learn a good representation of the input from unlabeled data
- Google Brain project
  - Learned the concept of a ‘cat’ by looking at unlabeled pictures from YouTube
  - One billion connection network

Perceptron Learning

Example

College Dublin
Computing • IT • Business

$$w^{(k+1)} = w^{(k)} + \lambda\left[y_i - f(w^{(k)}, x_i)\right]x_i$$

$$Y = sign\left(\sum_{i=0}^{d} w_i X_i\right)$$

$$\lambda = 0.1$$

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

	w_0	w_1	w_2	w_3
0	0	0	0	0
1	-0.2	-0.2	0	0
2	0	0	0	0.2
3	0	0	0	0.2
4	0	0	0	0.2
5	-0.2	0	0	0
6	-0.2	0	0	0
7	0	0	0.2	0.2
8	-0.2	0	0.2	0.2

- 1 Epoch with 9 batches

Epoch	w_0	w_1	w_2	w_3
0	0	0	0	0
1	-0.2	0	0.2	0.2
2	-0.2	0	0.4	0.2
3	-0.4	0	0.4	0.2
4	-0.4	0.2	0.4	0.4
5	-0.6	0.2	0.4	0.2
6	-0.6	0.4	0.4	0.2

Resources/ References

- Introduction to Machine Learning with Python A Guide for Data Scientists, Andreas C. Müller and Sarah Guido, Copyright © 2017, O'Reilly.
- Discovering Knowledge In Data: An Introduction To Data Exploration, Second Edition, By Daniel Larose And Chantal Larose, John Wiley And Sons, Inc., 2014.
- Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning Paperback – 23 Mar. 2018. by. Chris Albon
- Thoughtful Machine Learning by Matthew Kirk Published by O'Reilly Media, Inc., 2014
- https://towardsdatascience.com/support-vector-machines-svm-c9ef22815589
- https://github.com/ApoorvRusia/Naive-Bayes-classification-on-Iris-dataset
- Some images are used from google search repository to enhance the level of learning.
- Sample datasets for Regression: https://www.kaggle.com/tags/regression