# Artificial Intelligence

Dr. Mubashir Ahmad (Ph.D.)

# Artificial Neural Network

- The term "Artificial neural network" refers to a biologically inspired sub-field of artificial intelligence modeled after the brain.
- An Artificial neural network is usually a computational network based on biological neural networks that construct the structure of the human brain.
- Similar to a human brain has neurons interconnected to each other, artificial neural networks also have neurons that are linked to each other in various layers of the networks.
- These neurons are known as nodes. having neurons interconnected to each other, artificial neural networks also have neurons

# Artificial Neural Network

- The term "Artificial Neural Network" is derived from Biological neural networks that develop the structure of a human brain.
- Similar to the human brain which has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks.
- These neurons are known as nodes.

**Biological Neural Network.**

**The typical Artificial Neural Network**

# Artificial Neural Network

- Dendrites from Biological Neural Networks represent inputs in Artificial Neural Networks, cell nucleus represents Nodes, synapse represents Weights, and Axon represents Output.
- Relationship between Biological neural network and artificial neural network:

| Biological Neural Network | Artificial Neural Network |
|---|---|
| Dendrites | Inputs |
| Cell nucleus | Nodes |
| Synapse | Weights |
| Axon | Output |

# Artificial Neural Network

- An **Artificial Neural Network** in the field of **Artificial intelligence** it attempts to mimic the network of neurons that make up a human brain so that computers will have an option to understand things and make decisions in a human-like manner. The artificial neural network is designed by programming computers to behave simply like interconnected brain cells.
- There are around 1000 billion neurons in the human brain. Each neuron has an association point somewhere in the range of 1,000 and 100,000. In the human brain, data is stored in such a manner as to be distributed, and we can extract more than one piece of this data when necessary from our memory parallelly. We can say that the human brain is made up of incredibly amazing parallel processors.
- We can understand the artificial neural network with an example, consider an example of a digital logic gate that takes an input and gives an output. "OR" gate, which takes two inputs. If one or both the inputs are "On," then we get "On" in the output. If both the inputs are “Off,” then we get “Off” in output. Here the output depends upon the input. Our brain does not perform the same task. The outputs-input relationship keeps changing because of the neurons in our brain, which are "learning."

# Artificial Neural Network

- While mapping functions play an important role, there are many disadvantages, as mapping involves more computations and learning costs.
- Also, the disadvantages of transformations are that there is no generalized thumb rule available describing what transformations should be applied and if the data is large, mapping process takes huge amount of time.
- In real applications, there might be many features in the data and applying transformations that involve many polynomial combinations of these features will lead to extremely high and impractical computational costs.

# The Architecture of Artificial Neural Network

- To understand the concept of the architecture of an artificial neural network, we have to understand what a neural network consists of. In order to define a neural network that consists of a large number of artificial neurons, which are termed units arranged in a sequence of layers. Lets us look at various types of layers available in an artificial neural network.

# Neuron Activation

- Now we are coming to the topic of neuron activation. If the value of the neuron is greater than the threshold value so neuron is active, if the value is less then the neuron is not active.

# Types of Neuron Activation Function

1. Binary Step Function
2. Linear Activation Function
3. Non-Linear Activation Functions

# Binary Activation Function

- Binary step function depends on a threshold value that decides whether a neuron should be activated or not.
- The weighted sum is fed to the activation function and is compared to a certain threshold; if the weighted sum is greater than it, then the neuron is activated, else it is deactivated, meaning that its output is not passed on to the next hidden layer.

*Binary step*

$$f(x) = \begin{cases} 0 & for\ x < 0 \\ 1 & for\ x \geqslant 0 \end{cases}$$

# Binary Activation Function

- Here are some of the limitations of binary step function:
- It cannot provide multi-value outputs—for example, it cannot be used for multi-class classification problems.
- The gradient of the step function is zero, which causes a hindrance in the backpropagation process.

# Linear Activation Function

- The linear activation function, also known as "no activation," or "identity function", is where the activation is proportional to the input.
- The function doesn't do anything to the weighted sum of the input, it simply spits out the value it was given.

$$f(x) = x$$

# Linear Activation Function Disadvantages

- However, a linear activation function has two major problems :
- It's not possible to use backpropagation as the derivative of the function is a constant and has no relation to the input x.
- All layers of the neural network will collapse into one if a linear activation function is used.
- No matter the number of layers in the neural network, the last layer will still be a linear function of the first layer.
- So, essentially, a linear activation function turns the neural network into just one layer.

# Linear Activation Function Disadvantages

- The linear activation function shown above is simply a linear regression model.
- Because of its limited power, this does not allow the model to create complex mappings between the network's inputs and outputs.

# Non-Linear Activation Function

- Non-linear activation functions solve the following limitations of linear activation functions:
- They allow backpropagation because now the derivative function would be related to the input, and it's possible to go back and understand which weights in the input neurons can provide a better prediction.

# Non-Linear Activation Functions

10 Non-Linear Neural Networks Activation Functions

- Sigmoid
- Tanh
- ReLU
- Leaky ReLU
- Parametric ReLU
- ELU
- Softmax
- Swish
- GELU
- SELU

# Sigmoid/Logistic Activation Function

- This function takes any real value as input and outputs values in the range of 0 to 1.
- The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0.

*Sigmoid / Logistic*

$$f(x) = \frac{1}{1 + e^{-x}}$$

# Sigmoid/Logistic Activation Function

- Here's why sigmoid/logistic activation function is one of the most widely used functions:
- It is commonly used for models where we have to predict the probability as an output.
- Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice because of its range.
- The function is differentiable and provides a smooth gradient, i.e., preventing jumps in output values.

# Tanh/Hy-parabolic Tangent Activation Function

- Tanh function is very similar to the sigmoid/logistic activation function, and even has the same S-shape with the difference in output range of -1 to 1.
- In Tanh, the larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0.

x is weighted sum of neurons

$$Tanh \quad f(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$$

# ReLU Activation Function

- ReLU stands for Rectified Linear Unit.
- Although it gives an impression of a linear function, ReLU has a derivative function and allows for backpropagation while simultaneously making it computationally efficient.
- The main catch here is that the ReLU function does not activate all the neurons at the same time.
- The neurons will only be deactivated if the output of the linear transformation is less than 0.

$$ReLU$$

$$f(x) = max\ (0, x)$$

# ReLU Activation Function

- The advantages of using ReLU as an activation function are as follows:
- Since only a certain number of neurons are activated, the ReLU function is far more computationally efficient when compared to the sigmoid and tanh functions.

# ReLU Activation Function

- The limitations faced by this function are:
- ReLu suffers from **Dying ReLU problem.**
- The negative side of the graph makes the gradient value zero.
- Due to this reason, during the backpropagation process, the weights and biases for some neurons are not updated.
- This can create dead neurons which never get activated.

# Leaky ReLU Activation Function

- Leaky ReLU is an improved version of ReLU function to solve the Dying ReLU problem as it has a small positive slope in the negative area.

$$Leaky\ ReLU$$

$$f(x) = max\ (0.1x, x)$$

# Leaky ReLU Activation Function

Solve Dying ReLU Problem

- Derivative of the Leaky ReLU function.

$$f'(x) = g(x) = 1, \; x>=0$$
$$= 0.01, \; x<0$$

Constant value

Small number so we can update Biases and weights

**Leaky ReLu (derivative)**

# Leaky ReLU Activation Function

- The limitations that this function faces include:
- The predictions may not be consistent for negative input values.
- The gradient for negative values is a small value that makes the learning of model parameters time-consuming.

# Parametric ReLU Activation Function

- Parametric ReLU is another variant of ReLU that aims to solve the problem of gradient's becoming zero for the left half of the axis.
- This function provides the slope of the negative part of the function as an argument a.
- By performing backpropagation, the most appropriate value of **a** is learnt.

*Parametric ReLU*

$$f(x) = max\ (ax, x)$$

**Where "*a*" is the slope parameter for negative values.**

value of a could be change

# Parametric ReLU Activation Function

- The parameterized ReLU function is used when the leaky ReLU function still fails at solving the problem of dead neurons, and the relevant information is not successfully passed to the next layer.
- This function's limitation is that it may perform differently for different problems depending upon the value of slope parameter **a**.

- The parameterized ReLU function is used when the leaky ReLU function still fails at solving the problem of dead neurons, and the relevant information is not successfully passed to the next layer.

# SoftMax Activation Function

- Before exploring the ins and outs of the Softmax activation function, we should focus on its building block—the sigmoid/logistic activation function that works on calculating probability values.

*Sigmoid / Logistic*

$$f(x) = \frac{1}{1 + e^{-x}}$$

# SoftMax Activation Function

- The output of the sigmoid function was in the range of 0 to 1, which can be thought of as probability.
- But—
- This function faces certain problems.
- Let's suppose we have five output values of 0.8, 0.9, 0.7, 0.8, and 0.6, respectively. How can we move forward with it?
- The answer is: We can't.
- The above values don't make sense as the sum of all the classes/output probabilities should be equal to 1.

# SoftMax Activation Function

- You see, the Softmax function is described as a combination of multiple sigmoids.
- It calculates the relative probabilities.
- Similar to the sigmoid/logistic activation function, the SoftMax function returns the probability of each class.
- It is most commonly used as an activation function for the last layer of the neural network in the case of multi-class classification.

**Softmax**

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

# SoftMax Activation Function Example

- Let's go over a simple example together.
- Asume that you have three classes, meaning that there would be three neurons in the output layer. Now, suppose that your output from the neurons is [1.8, 0.9, 0.68]
- Applying the softmax function over these values to give a probabilistic view will result in the following outcome: [0.53, 0.27, 0.20]

$= 1$

**Softmax**

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

# SoftMax Activation Function

- You see, the Softmax function is described as a combination of multiple sigmoids.
- It calculates the relative probabilities.
- Similar to the sigmoid/logistic activation function, the SoftMax function returns the probability of each class.
- It is most commonly used as an activation function for the last layer of the neural network in the case of multi-class classification.

**Softmax**

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

# SoftMax Activation Function

- The function returns 1 for the largest probability index while it returns 0 for the other two array indexes.
- **outcome** [0.53, 0.27, 0.20]

**Softmax**

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$