Introduction to Neural Network

As the name Artificial Neural Networks (ANNs) suggests, the design of ANNs is inspired by the human brain. Although not as powerful as the brain (yet), artificial neural networks are the most powerful learning models in the field of machine learning.

Common Interview Questions

1. What is an Artificial Neural Network and how does it work?
2. Describe the structure of a typical feedforward neural network.
3. What is the purpose of an activation function in an ANN, and what are some commonly used activation functions?
4. Explain back propagation algorithm and its role in training ANN.
5. What is the meaning of the term "vanishing gradient" and how can it affect the training process of ANNs?
6. How does regularization help prevent overfitting in neural networks?
7. What is the difference between supervised and unsupervised learning in the context of ANNs?
8. Can you explain the concept of dropout in neural networks and how it helps improve generalization?
9. Describe the concept of batch normalization and its benefits in training ANNs.
10. How do you determine the optimal number of hidden layers and neurons in an ANN architecture?

Applications of ANNs:

1. Speech recognition
2. Fraud detection
3. Image classification
4. Sentiment analysis
5. Credit scoring

Why NN?

- Neural networks are capable of capturing and representing highly complex nonlinear relationships in data. They can learn intricate patterns and make accurate predictions, which makes them suitable for tasks where linear models or traditional algorithms may fall short.
- Neural networks facilitate end-to-end learning, where they can directly learn from raw input data to produce desired outputs without relying on manual feature engineering

Basics of ANN

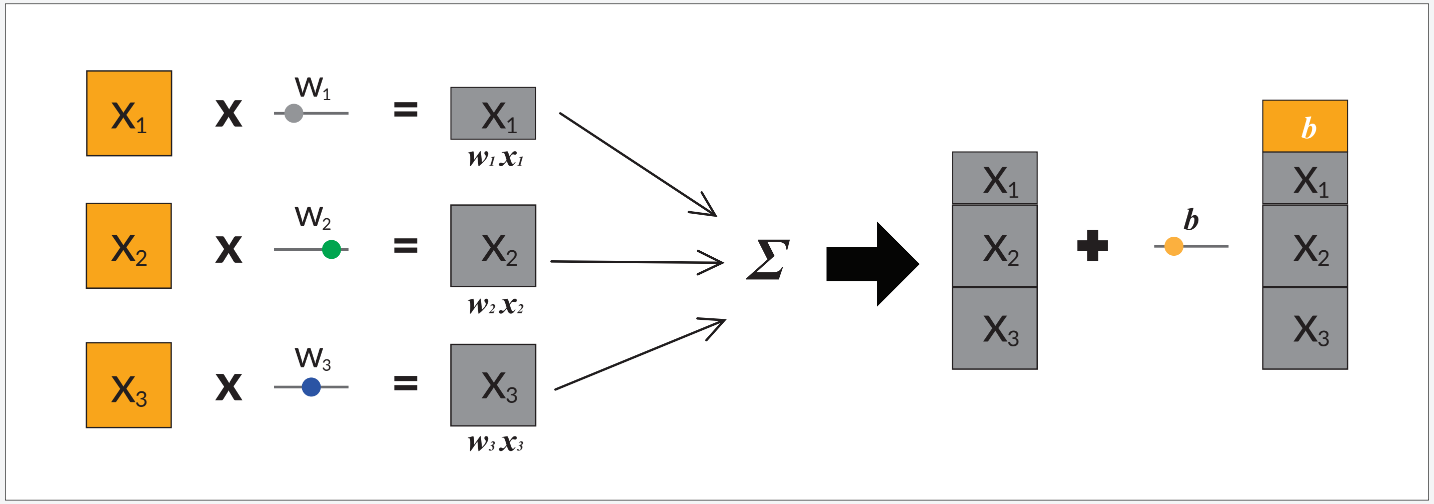
Perceptron: A perceptron is a single-layer neural network that takes a set of input features, applies weights to them, and produces an output. It can be used for binary classification tasks where the goal is to separate two classes of data.

The output of the perceptron (Output) is calculated using the following steps:

1. Multiply each input by its corresponding weight.
2. Sum up the weighted inputs.
3. Apply an activation function to the sum (commonly a step function, such as the Heaviside function or a sigmoid function).
4. The activation function produces the final output, which is typically a binary value representing the predicted class label (e.g., 0 or 1).

Working of Single Neuron:

The weights are applied to the inputs respectively, and along with the bias, the cumulative input is fed into the neuron. An activation function is then applied on the cumulative input to obtain the output of the neuron. We have seen some of the activation functions such as softmax and sigmoid in the previous segment. We will explore other types of activation functions in the next segment. These functions apply non-linearity to the cumulative input to enable the neural network to identify complex non-linear patterns present in the data set.



$$z = W_2 x_2 + W_2 X_2 + W_3 x_3 + b$$

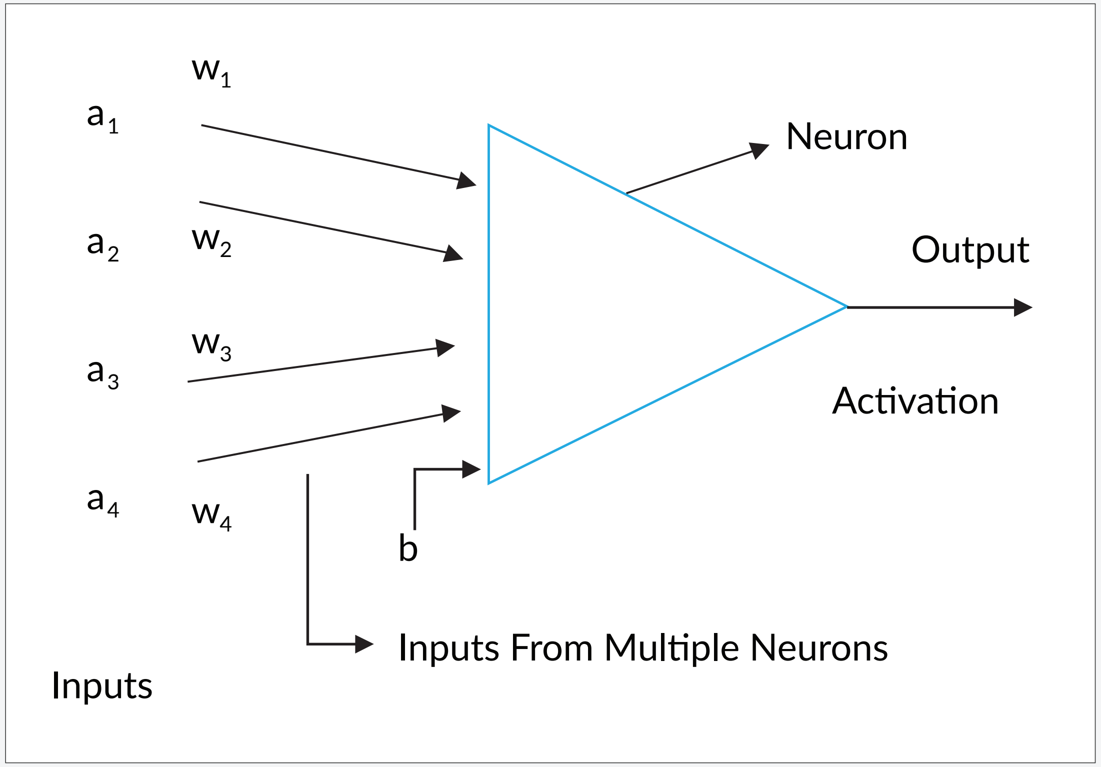
Architecture of Neuron and Neural Networks

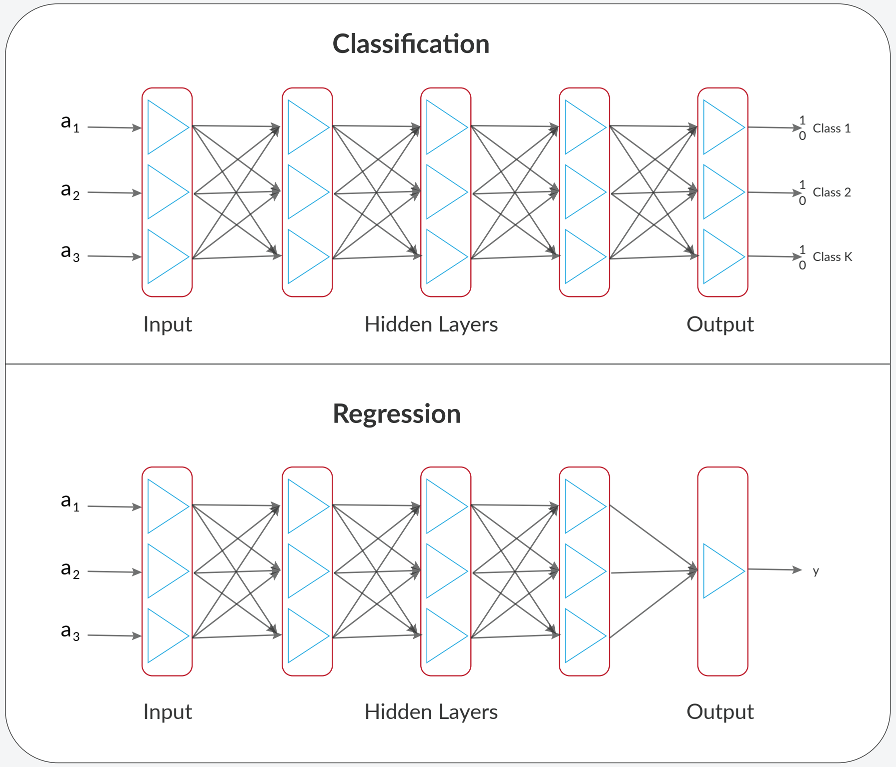
The first layer is known as the input layer, and the last layer is called the output layer. The layers in between these two are the hidden layers. The number of neurons in the input layer is equal to the number of attributes in the data set, and the number of neurons in the output layer is determined by the number of classes of the target variable (for a classification problem).

For a regression problem, the number of neurons in the output layer would be 1 (a numeric variable). Take a look at the image given below to understand the topology of neural networks in the case of classification and regression problems.

There are six main elements that must be specified for any neural network.

1. Input layer
2. Output layer
3. Hidden layers
4. Network topology or structure
5. Weights and biases
6. Activation functions



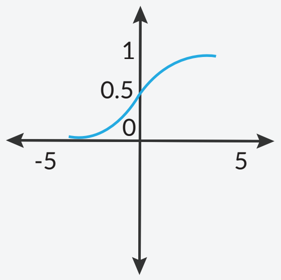


Activation Functions

1. Sigmoid: When this type of function is applied, the output from the activation function is bound between 0 and 1 and is not centred around zero. A sigmoid activation function is usually used when we want to regularise the magnitude of the outputs we get from a neural network and ensure that this magnitude does not blow up.
2. Tanh (Hyperbolic Tangent): When this type of function is applied, the output is centred around 0 and bound between -1 and 1, unlike a sigmoid function in which case, it is centred around 0.5 and will give only positive outputs. Hence, the output is centred around zero for tanh.
3. ReLU (Rectified Linear Unit): The output of this activation function is linear in nature when the input is positive and the output is zero when the input is negative. This activation function allows the network to converge very quickly, and hence, its usage is computationally efficient. However, its use in neural networks does not help the network to learn when the values are negative.
4. Leaky ReLU (Leaky Rectified Linear Unit): This activation function is similar to ReLU. However, it enables the neural network to learn even when the values are negative. When the input to the function is negative, it dampens the magnitude, i.e., the input is multiplied with an epsilon factor that is usually a number less than one. On the other hand, when the input is positive, the function is linear and gives the input value as the output. We can control the parameter to allow how much 'learning emphasis' should be given to the negative value.

Sigmoid

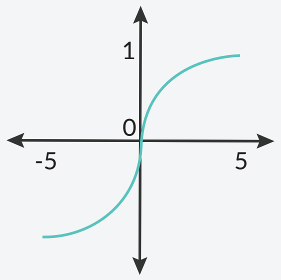
$$g(z) = \frac{1}{1 + e^{-z}}$$



Output values bound between O and 1.
Outputs not zero-centered.

Tanh

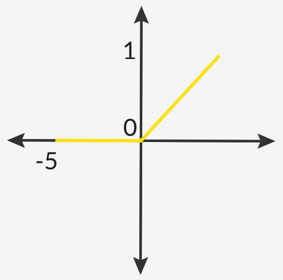
$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



Output values bound between -1 and 1.
Outputs are zero-centered

ReLU

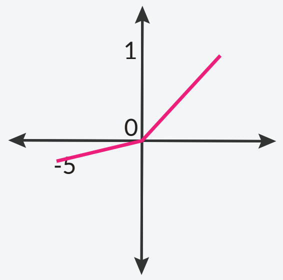
$$g(z) = max(0,z)$$



Computationally efficient: allows the network to converge very quickly.

Leaky ReLU

$$g(z) = max(\epsilon z, z); \epsilon << 1$$



Allows learning even when the value is negative.

Time Series Forecast Models

During training, the neural network learning algorithm fits various models to the training data and selects the best prediction model. The learning algorithm is trained with a fixed set of hyperparameters associated with the network structure. Some of the important hyperparameters to consider to decide the network structure are given below:

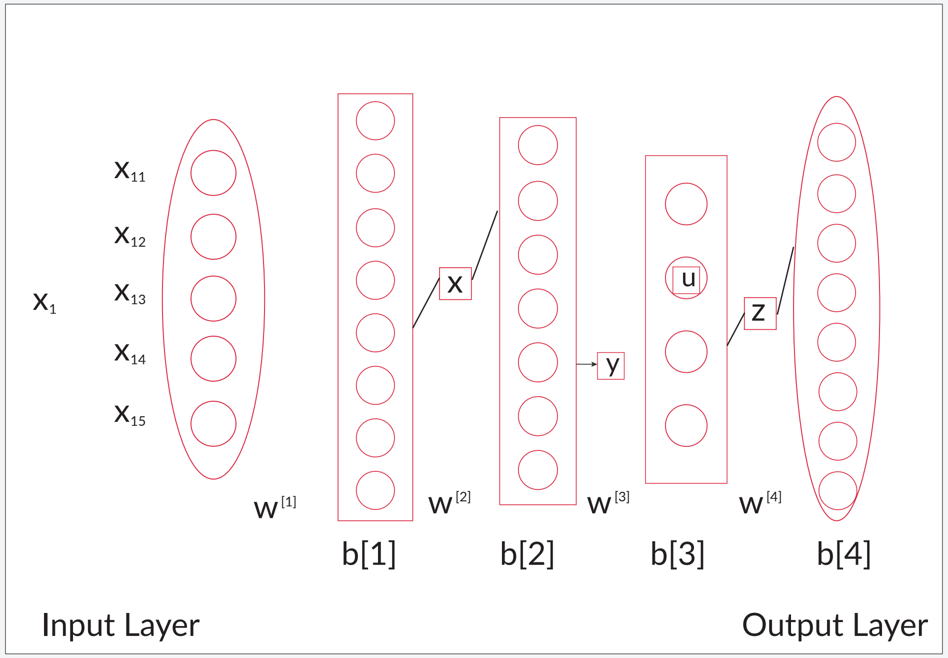
- Number of layers
- Number of neurons in the input, hidden and output layers
- Learning rate (the step size taken each time we update the weights and biases of an ANN)
- Number of epochs (the number of times the entire training data set passes through the neural network)

The purpose of training the learning algorithm is to obtain optimum weights and biases that form the parameters of the network.

Notations used in neural networks:

The notations that you will come across going forward are as follows:

1. W represents the weight matrix, b stands for bias, x represents the input, y represents the ground truth label.
2. p represents the probability vector of the predicted output for the classification problem. hL represents the predicted output for the regression problem (where L represents the number of layers).
3. h also represents the output of the hidden layers with appropriate superscript. The output of the second neuron in the nth hidden layer is denoted by hn2.
4. z represents the accumulated input to a layer. The accumulated input to the third neuron of the nth hidden layer is zn3.
5. The bias of the first neuron of the third layer is represented as b31.
6. The superscript represents the layer number. The weight matrix connecting the first hidden layer to the second hidden layer is denoted by W2.
7. The subscript represents the index of the individual neuron in a given layer. The weight connecting the first neuron of the first hidden layer to the third neuron of the second hidden layer is denoted by w231.



Assumptions of Neural Network

Commonly used neural network architectures make the following simplifying assumptions:

1. The neurons in an ANN are arranged in layers, and these layers are arranged sequentially.
2. The neurons within the same layer do not interact with each other.
3. The inputs are fed into the network through the input layer, and the outputs are sent out from the output layer.
4. Neurons in consecutive layers are densely connected, i.e., all neurons in layer l are connected to all neurons in layer l+1.
5. Every neuron in the neural network has a bias value associated with it, and each interconnection has a weight associated with it.
6. All neurons in a particular hidden layer use the same activation function. Different hidden layers can use different activation functions, but in a hidden layer, all neurons use the same activation function.

Loss Function:

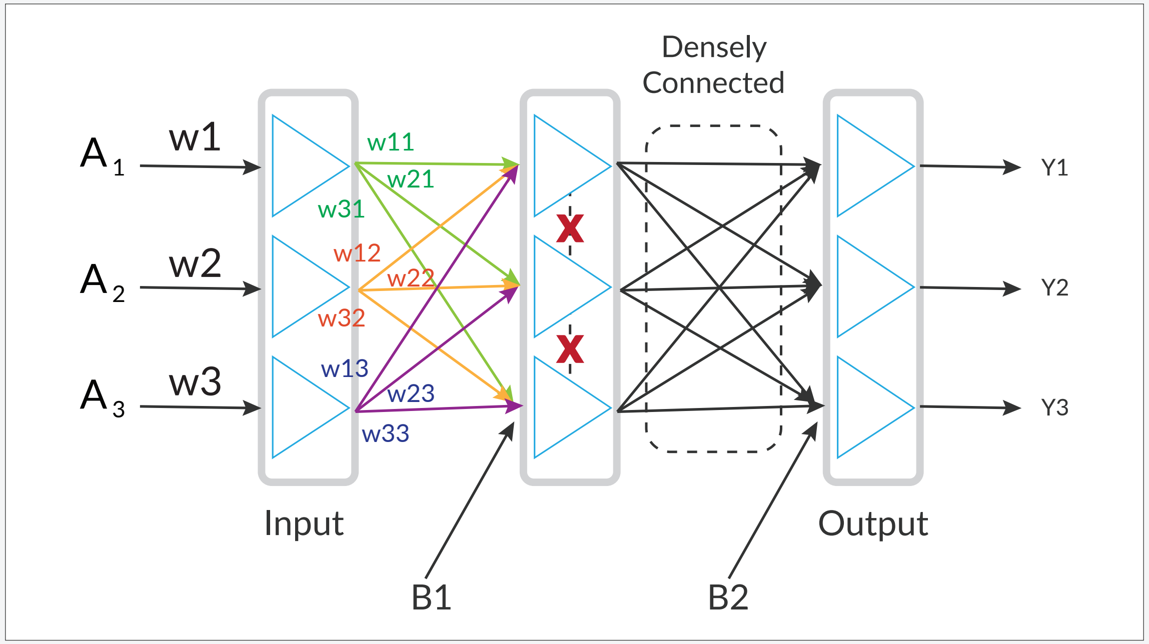
A loss function or cost function is a function that maps an event or values of one or more variables onto a real number intuitively, representing some ‘cost’ associated with the ‘event’, as shown below:

L(y,^y)=f:(y,^y)→R

Neural networks minimise the error in the prediction by optimising the loss function with respect to the parameters in the network. In other words, this optimisation is done by adjusting the weights and biases. We will see how this adjustment is done in subsequent sessions. For now, we will concentrate on how to compute the loss.

In the case of regression, the most commonly used loss function is MSE/RSS.

In the case of classification, the most commonly used loss function is Cross Entropy/Log Loss.



Feed Forward & Back-propagation Algorithm

The pseudocode/pseudo-algorithm is given as follows:

1: Initialise with the input

Forward Propagation

2: For each layer, compute the cumulative input and apply the non-linear activation function on the cumulative input of each neuron of each layer to get the output.

3: For classification, get the probabilities of the observation belonging to a class, and for regression, compute the numeric output.

4: Assess the performance of the neural network through a loss function, for example, a cross-entropy loss function for classification and RMSE fo regression.

Backpropagation

5: From the last layer to the first layer, for each layer, compute the gradient of the loss function with respect to the weights at each layer and all the ntermediate gradients.

6: Once all the gradients of the loss with respect to the weights (and biases) are obtained, use an optimisation technique like gradient descent to update the values of the weights and biases.

Repeat this process until the model gives acceptable predictions:

7: Repeat the process for a specified number of iterations or until the predictions made by the model are acceptable.

Pseudocode for feed forward pass:

The pseudocode for a feedforward pass is given below:

1. We initialise the variable h^0 as the input: $h_0 = x_i$
2. We loop through each of the layers computing the corresponding output for each layer, i.e., h^l.
   For I in [1,2,......,L]: $h^l = \sigma(W^l.h^{l-1} + b^l)$
3. We compute the prediction p by applying an activation function to the output from the previous layer, i.e., we apply a function to h^L, as shown below. $p = f(h^L)$

Pseudocode for feed forward pass for Classification:

```
1. $h^0 = X_i$
2. For 1 in [1,2,....,L]: $h^l = \sigma(W^l. h^{l-1} + b^l)$
3. $P_i = e^{W^0}.h^L$
4. $P_i = normalise\ (p_i)$
```

Pseudocode for Backward Propagation:

```
Initialize Weights;
While not Stop-Criterion do
    For all i, j do
        $w_{ij} = w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$
End For
End While
```

Feed Forward & Back-propagation Algorithm

Tensor: A tensor is the fundamental data structure used in TensorFlow. It is a multidimensional array with a uniform data type. The data type for an entire tensor is the same. In the case of data frames, all the raw data, such as integers, strings and floats, can be loaded into a single data frame. So, you could load raw data into a data frame and then process the data to convert it into a numerical form for ML. In the case of tensors, data would need to be loaded into another data structure and processed first.

1. Creating Tensors

```
import tensorflow as tf

# Create a tensor with random values
X = tf.random.normal(shape=(3, 3))

# Create a tensor from a list
y = tf.constant([1, 2, 3, 4, 5])

# Create a tensor of zeros
z = tf.zeros(shape=(2, 2))
```

2. Performing Operations on Tensors

```
import tensorflow as tf

#Element-wise multiplication
x = tf.constant([1, 2, 3])
y= tf.constant([[4, 5, 6])
c= tf.multiply(a, b)

#Matrix multiplication
x = tf.constant([[1, 2], [3, 4]])
y= tf.constant([[5, 6], [7, 8]])
z = tf.matmul(x, y)

#Reshaping a tensor
original tensor = tf.constant([[1, 2], [3, 4]])
reshaped tensor = tf.reshape(original tensor, shape=(1, 4))
```

3. Using Tensors in Neural Networks

```
import tensorflow as tf

# Define a simple neural network
model = tf.keras. Sequential([
  tf.keras.layers. Dense (32, activation='relu', input_shape=(784,)),
  tf.keras.layers. Dense (10, activation='softmax')
])

# Create a tensor from numpy array
import numpy as np
data = np.random.rand(10, 784)
tensor= tf.convert_to_tensor(data)

#Forward pass through the neural network
output model(tensor)
```