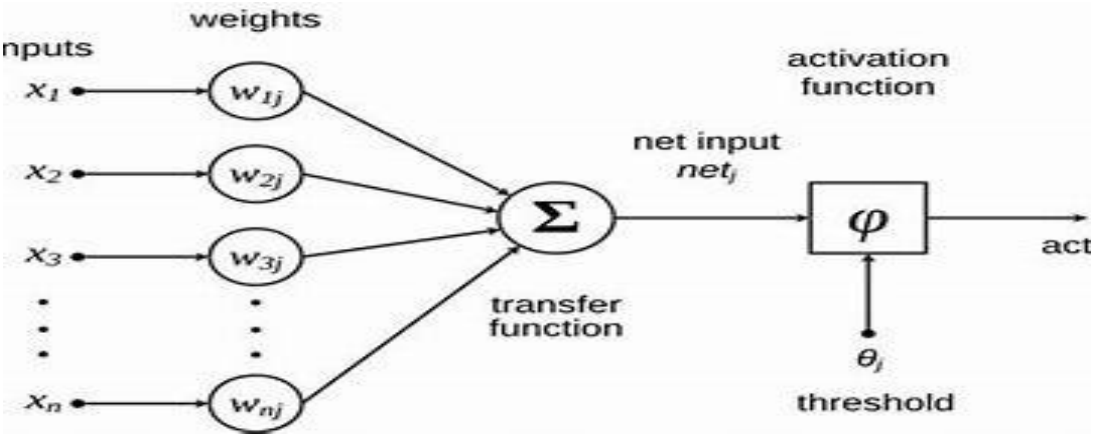
1. Artificial Neural Network is a group of connected artificial neurons which simulate the way neurons in the human brain are structured and function.

2. An appropriate problem for neural network learning is image classification whereby the machine is required to categorize objects detected in an image for example classifying trees from humans from cars etc.

   Its characteristics that make it suitable for neural network learning are:

   i. Its nonlinear meaning the output of classified objects cannot be predicted with accuracy by linear combination of the inputs
   ii. It is high-dimensional and complex meaning many features affect its output
   iii. It is large and diverse meaning there are many examples of the data which cover a wide range of possible scenarios
   iv. It accepts long training times
   v. It has fast evaluation of the learned target function

3. A perceptron is a single layer of perceptron which is comprised of five units which are the input values, the weights and bias, the net sum and then the activation function. It can be represented diagrammatically as follows:



4. The single perceptron learning starts by initializing the weights and bias .For each training example the output is calculated by computing the net sum and applying the activation function. Second step is to calculate the error by comparing the calculated output and its expected value. The weights and bias are updated using a learning rule that minimizes the error. The process is repeated until the error is small enough or a minimum number of iterations are reached.
5. A single perceptron can be used to represent Boolean functions by using appropriate weights and that satisfy the truth table of the function. Using the AND function as an example the following truth table may be used:

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

Which can be made by using weights and bias of w1=w2=1 and b=−1.5. This means that the net sum is:

net=x1+x2−1.5 and the activation function is a step function that returns 1 if the net sum is positive and 0 otherwise.

6.

i. Input layer: A and B

Output layer: C = A AND NOT B

w1 = 1 (weight for input A)

w2 = -1 (weight for input B)

b = -0.5 (bias for output C)

Input layer: A and B

Hidden layer: C = A AND B, D = A OR B

Output layer: E = NOT C AND D

7. It is false because perceptron B is more general than perceptron A and they are equivalent in terms of their decision boundaries.

8i. Perceptron Training Rule is an algorithm that learns the weights and bias of a single perceptron based on a set of training examples.

ii. Gradient Descent and Delta rule is optimization technique that is sued to find the minimum of a function by iteratively moving in the direction of steepest descent.

9. A Gradient Descent algorithm (pseudocode) for training a linear input is:

10. A gradient descent rule is an iterative optimization that is used for finding a local minimum of differentiable function which takes repeated steps in the opposite direction of the gradient which is the direction of the steepest descent.

11.

```
# Define the linear unit model
def linear_unit(x, w, b):
  return np.dot(x, w) + b

# Define the mean squared error loss function
def mse_loss(y_true, y_pred):
  return np.mean((y_true - y_pred) ** 2)

# Define the gradient of the loss function with
respect to w and b
def grad_loss(x, y_true, y_pred):
  grad_w = -2 * np.mean(x * (y_true - y_pred),
axis=0)
  grad_b = -2 * np.mean(y_true - y_pred)
  return grad_w, grad_b

# Define the Stochastic Gradient Descent algorithm
def sgd(x, y, epochs, lr):
  # Initialize random weights and bias
  w = np.random.randn(x.shape[1])
  b = np.random.randn()
  # Loop over the number of epochs
  for epoch in range(epochs):
    # Shuffle the data
    perm = np.random.permutation(x.shape[0])
    x = x[perm]
    y = y[perm]
    # Loop over each data point
    for i in range(x.shape[0]):
      # Make a prediction
      y_pred = linear_unit(x[i], w, b)
      # Compute the loss
      loss = mse_loss(y[i], y_pred)
      # Compute the gradient
      grad_w, grad_b = grad_loss(x[i], y[i], y_pred)
      # Update the weights and bias
      w = w - lr * grad_w
      b = b - lr * grad_b
    # Print the loss at the end of each epoch
    print(f"Epoch {epoch}, Loss: {loss}")
```

12. Differences between Gradient descent and stochastic gradient are:

Gradient descent	Stochastic gradient
i. uses all the training examples	uses a single or subset of training examples
ii. expensive to store all the training examples	relatively cheaper as it does not use all training
iii. most suitable for convex functions with single global minimum	most suitable for non-convex functions with multiple local minima
iv. slow but more accurate	fast but less accurate

13.

```
import numpy as np

# Define the sigmoid function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Define the derivative of the sigmoid function
def sigmoid_prime(x):
    return sigmoid(x) * (1 - sigmoid(x))

# Define the network parameters
n_input = 3 # Number of input units
n_hidden = 4 # Number of hidden units
n_output = 2 # Number of output units
eta = 0.1 # Learning rate

# Initialize the weights randomly
W1 = np.random.randn(n_hidden, n_input) # Weights
from input to hidden layer
W2 = np.random.randn(n_output, n_hidden) # Weights
from hidden to output layer

# Define the training data
X = np.array([[0, 0, 1], [0, 1, 1], [1, 0, 1], [1, 1,
1]]) # Input features
y = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]) #
Output labels

# Define the number of epochs
```

14. The back propagation rule is an algorithm that is used to calculate the gradients of the error function using the weights and biases of a neural network by applying the chain rule of derivatives to propagate the error from the output layer to the input layer.

15. Defining terms w.r.t Back Propagation algorithm:

i. Convergence and Local Minima

-convergence is a process for finding a solution that minimizes the error function while

-local minima are points in the error surface where the error is lower than all neighboring points but not lower than all possible points.

ii. Representational Power of Feedforward Networks

-defines their ability to approximate any continuous function on a compact domain

iii. Hypothesis Space Search and Inductive Bias

-hypothesis space search is defined as the set of all possible hypothesis that can be considered by a learning algorithm

-inductive bias is defined as the set of assumptions that guide the hypothesis space search and influence the choice of hypothesis

iv. Hidden Layer Representations

- Are defined as the values of the hidden units in a neural network after applying the activation function

v. Generalization, Overfitting, and Stopping Criterion

-generalization is defined as the ability of a neural network to perform well on new and unseen data that is not part of the training set.

-overfitting is defined as the process that happens when a neural network becomes too familiar with the training data and fails to generalize well to new data

-stopping criterion is defined as a condition that determines when to stop training the network to avoid overfitting or underfitting.