Neural network:

it is a machine learning algorithm modeled by making something with the same structure as human brain, It consists of layers of interconnected neurons that process and transmit information. By adjusting the weights and biases of the connections between neurons, a neural network can learn to recognize patterns in data and make predictions or classifications.
Training a neural network involves providing it with labeled examples of data and adjusting its parameters until it achieves satisfactory performance.

The neural formula used depends on the specific type of neural network being used and the activation function used by each neuron, the basic formula used to calculate the output of a neuron involves multiplying the input values by a set of weights, summing the results, and then adding a bias term. The resulting sum is then passed through an activation function to produce the output of the neuron.

$$\mathbf{z} = \sum_{\mathbf{i}=1}^{n} x^i * w^i + b$$

where z is the weighted sum of the inputs x^i, with corresponding weights w^i and a bias term b. The activation function is then applied to z to produce the output of the neuron, which can be denoted as a(z).

Forward propagation:

it is the process of passing input data through a neural network to generate a prediction or output. Each neuron in the network calculates a weighted sum of its inputs, applies an activation function, and passes the result to the next layer of neurons.

z = Wx + b: calculates the weighted sum of inputs **x**, where **W** is the weight matrix and **b** is the bias vector
a = g(z): applies the activation function **g** to the weighted sum **z** to produce the output of the neuron
h = $a^{(L)}$ = g($z^{(L)}$): calculates the final output of the network for a given input **x**, where **L** is the index of the final layer

Backward propagation:

it is the process of calculating the gradient of the loss function with respect to the weights and biases of the network. This gradient is then used to update the network parameters through optimization algorithms such as stochastic gradient descent.
The goal of backward propagation is to adjust the network parameters to minimize the difference between the predicted output and the actual output, as measured by the loss function. This process of updating the parameters based on the error is repeated iteratively until the network achieves good performance.

$\boldsymbol{\delta^{(L)} = \nabla_a L \odot g'(z^{(L)})}$: calculates the error term for the final layer, where $\nabla_a L$ is the gradient of the loss function with respect to the final output $a^{(L)}$, and $\odot$ represents element-wise multiplication
$\boldsymbol{\delta^{(l)} = ((W^{(l+1)})^T \delta^{(l+1)}) \odot g'(z^{(l)})}$: calculates the error term for layer **l**, where $(W^{(l+1)})^T$ is the transpose of the weight matrix for the next layer, and **l** is the index of the current layer
$\frac{\partial L}{\partial W^{(l)}} = a^{(l-1)} (\delta^{(l)})^T$: calculates the gradient of the loss function with respect to the weights of layer **l**
$\frac{\partial L}{\partial b^{(l)}} = \delta^{(l)}$: calculates the gradient of the loss function with respect to the biases of layer **l**
$W^{(l)} = W^{(l)} - \alpha \left(\frac{\partial L}{\partial W^{(l)}}\right)$: updates the weights of layer **l** using the learning rate **α** and the gradient of the loss function with respect to the weights
$b^{(l)} = b^{(l)} - \alpha \left(\frac{\partial L}{\partial b^{(l)}}\right)$: updates the biases of layer **l** using the learning rate **α** and the gradient of the loss function with respect to the biases

Hyper-parameters:

they are adjustable parameters in a neural network that are set before training and can have a significant impact on the network's performance.

Like:

Activation function: the function used to introduce non-linearity into the output of each neuron. Common activation functions include **sigmoid**, **ReLU**, and **tanh**.

Learning rate: the step size at which the network adjusts its parameters during back-propagation. A high learning rate can cause the network to overshoot the optimal weights, while a low learning rate can make the training process too slow.

Batch size: the number of samples processed by the network in each iteration of training. A larger batch size can speed up training, but can also increase the risk of over-fitting and require more memory.

Number of hidden layers: the number of layers between the input and output layers of the network. Adding more layers can increase the network's capacity to learn complex features, but can also increase the risk of over-fitting.

Number of neurons per layer: the number of nodes in each hidden layer of the network. Increasing the number of neurons can also increase the network's capacity to learn, but can make the training process slower and increase the risk of over-fitting.

Regularization: techniques used to prevent over-fitting, such as **L1** or **L2 regularization**, **dropout**, or **early stopping**.

Optimizer: the algorithm used to update the network parameters during back-propagation, such as **stochastic gradient descent** or **Adam**.

	Classification	**Regression**
Definition	Classification is a machine learning task where the goal is to assign input data to one of several predetermined categories or classes.	Regression is a machine learning task where the goal is to predict a continuous output variable based on input data.
Loss function	Cross-entropy loss	Mean squared error, mean absolute error, etc.
Input layer	Accepts input data with a fixed number of features	Accepts input data with a fixed number of features
Output activation function	• Softmax (for multi-class classification) • Sigmoid (for binary classification) • Linear (for simple regression)	No activation function
Goal	Assign input to one of several classes	Predict a continuous output variable