Artificial neural networks (ANNs) are computational models inspired by the structure and function of the human brain. They are composed of interconnected "neurons" that can process and transmit information, and they are able to learn and adapt based on the data they are exposed to. ANNs are widely used in a variety of applications, including image and speech recognition, natural language processing, and decision-making.

There are many different types of ANN algorithms, each with its own specific characteristics and capabilities. Some of the most common types of ANNs include:

- **Feedforward neural networks:** These are the simplest type of ANNs, in which the neurons are arranged in layers, with the inputs being passed through the layers and processed by the neurons to produce an output.
- **Convolutional neural networks (CNNs):** These are specialized ANNs that are designed to process and analyze images and other spatial data. They are composed of multiple layers of neurons, each of which is responsible for analyzing a different part of the input data.
- **Recurrent neural networks (RNNs):** These ANNs are designed to process sequential data, such as text or time series data. They are able to "remember" past inputs and use that information to process the current input.
- **Self-organizing maps (SOMs):** These ANNs are used for clustering and visualization tasks. They are composed of neurons arranged in a two-dimensional grid, with the neurons "competing" to represent different parts of the input data.
- **Autoencoders:** These are a type of ANN that is used for dimensionality reduction and feature learning. They are composed of an encoder layer that processes the input data and a decoder layer that reconstructs the original data from the encoded representation.

Each of these ANN algorithms has its own specific characteristics and capabilities, and they can be used in a variety of different applications. For example, feedforward neural networks are often used for classification tasks, while CNNs are well-suited for image recognition tasks. RNNs are frequently used for natural language processing tasks, and SOMs are often used for visualization and clustering tasks. Overall, ANNs are a powerful and widely used tool in the field of artificial intelligence and machine learning. They are able to process and analyze large amounts of data and learn from it, making them well-suited for a wide range of applications.

Here are a few examples of applications that use artificial neural networks (ANNs):

- **Image recognition:** ANNs are widely used for tasks such as object recognition in images and videos. For example, a neural network might be trained to recognize specific objects or patterns in an image, such as a person's face or a traffic sign.
- **Speech recognition:** ANNs are also used for tasks such as speech-to-text conversion, which involves transcribing spoken words into written text. Neural networks are able to learn the patterns and structures of spoken language and are able to accurately transcribe spoken words in a variety of languages and accents.

- **Natural language processing:** ANNs are frequently used for tasks such as language translation, text classification, and sentiment analysis. They are able to process and analyze large amounts of text data and are able to learn the patterns and structures of language.
- **Decision-making:** ANNs are sometimes used to support decision-making processes, such as in the field of finance. For example, a neural network might be trained to analyze financial data and make predictions about the performance of different investments.
- **Autonomous systems:** ANNs are also used in a variety of autonomous systems, such as self-driving cars and drones. They are able to process and analyze sensor data and make decisions based on that information, allowing the system to navigate and perform tasks without human intervention.

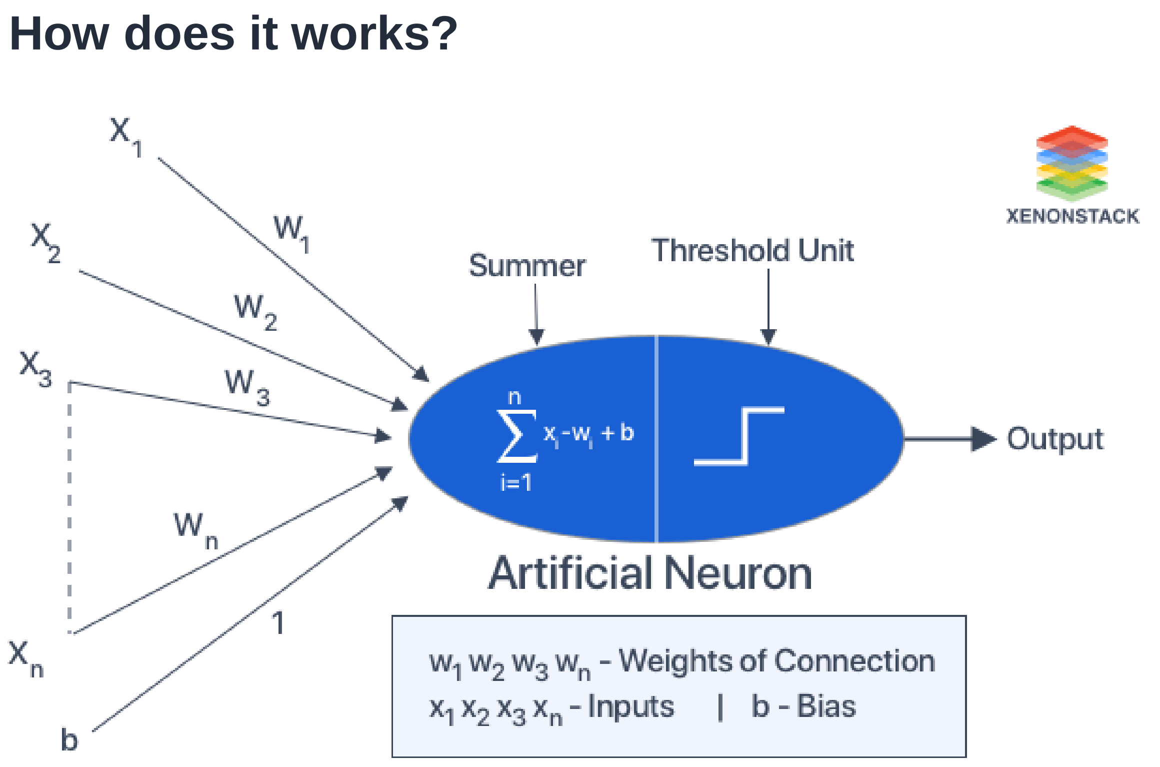
Some Facts related to ANN are:

1. ANNs can be trained to perform a wide range of tasks, including classification, regression, and clustering.
2. Training an ANN involves feeding it a large dataset and adjusting the weights and biases of the network to minimize the error between the predicted output and the actual output.
3. ANNs can be sensitive to the scale of the input data, so it is often a good idea to normalize the data before training.
4. One of the challenges of training ANNs is finding the right balance between under fitting (not capturing the underlying patterns in the data) and overfitting (memorizing the training data and not generalizing well to new data).
5. ANNs are highly flexible and can be used in a wide variety of applications, such as image recognition, natural language processing, and predictive modelling.
6. ANNs are prone to the "black box" problem, where it can be difficult to understand why the network made a particular decision or prediction. This can be a drawback in some applications where transparency is important.

Google has used artificial neural networks in a variety of its products and services. For example, ANNs have been used in Google's image search functionality to enable the system to recognize and classify different objects and scenes in images. ANNs have also been used in Google Translate to enable the system to accurately translate text from one language to another.

In terms of the mathematical algorithm used to train an ANN, there are many different approaches that can be taken. One common algorithm is called "backpropagation," which involves adjusting the weights of the connections between the neurons in the network based on the error between the predicted output and the actual output. This process is often repeated for many iterations, using a set of labeled training data, in order to optimize the performance of the ANN. Here is a more detailed description of the backpropagation algorithm:

- Initialize the weights of the connections between the neurons in the network to small random values.
- Feed a sample of the training data through the network, using the input data to activate the neurons in the input layer and propagating the signals through the hidden layers to the output layer.
- Calculate the error between the predicted output and the actual output for the sample.
- Propagate the error back through the network, adjusting the weights of the connections between the neurons based on the error. This step is repeated for each sample in the training data.
- Adjust the learning rate and repeat the process for multiple epochs until the error is minimized and the performance of the network is optimized.


How does it works?
XENONSTACK
X_1
X_2
X_3
X_n
b
W_1
W_2
W_3
W_n
1
Summer
Threshold Unit
$\sum_{i=1}^{n} x_i \cdot w_i + b$
Output
Artificial Neuron
$w_1\ w_2\ w_3\ w_n$ - Weights of Connection
$x_1\ x_2\ x_3\ x_n$ - Inputs | b - Bias

- It can be viewed as weighted directed graphs in which artificial neurons are nodes, and directed edges with weights are connections between neuron outputs and neuron inputs.
- The Artificial Neural Network receives information from the external world in pattern and image in vector form. These inputs are designated by the notation x(n) for n number of inputs.

- Each input is multiplied by its corresponding weights. Weights are the information used by the neural network to solve a problem. Typically weight represents the strength of the interconnection between neurons inside the Neural Network.
- The weighted inputs are all summed up inside the computing unit (artificial neuron). In case the weighted sum is zero, bias is added to make the output not- zero or to scale up the system response. Bias has the weight and input always equal to '1'.
- The sum corresponds to any numerical value ranging from 0 to infinity. To limit the response to arrive at the desired value, the threshold value is set up. For this, the sum is forward through an activation function.
- The activation function is set to the transfer function to get the desired output. There are linear as well as the nonlinear activation function.

Think of each individual node as its own linear regression model, composed of input data, weights, a bias (or threshold), and an output. The formula would look something like this:

$\sum wixi + bias = w1x1 + w2x2 + w3x3 + bias$

Output = $f(x) = 1$ if $\sum w1x1 + b >= 0$; 0 if $\sum w1x1 + b < 0$

Once an input layer is determined, weights are assigned. These weights help determine the importance of any given variable, with larger ones contributing more significantly to the output compared to other inputs. All inputs are then multiplied by their respective weights and then summed. Afterward, the output is passed through an activation function, which determines the output. If that output exceeds a given threshold, it "fires" (or activates) the node, passing data to the next layer in the network. This results in the output of one node becoming in the input of the next node. This process of passing data from one layer to the next layer defines this neural network as a feedforward network.

Let's break down what one single node might look like using binary values. We can apply this concept to a more tangible example, like whether you should go surfing (Yes: 1, No: 0). The decision to go or not to go is our predicted outcome, or y-hat. Let's assume that there are three factors influencing your decision-making:

1. Are the waves good? (Yes: 1, No: 0)
2. Is the line-up empty? (Yes: 1, No: 0)
3. Has there been a recent shark attack? (Yes: 0, No: 1)

Then, let's assume the following, giving us the following inputs:

- X1 = 1, since the waves are pumping
- X2 = 0, since the crowds are out
- X3 = 1, since there hasn't been a recent shark attack

Now, we need to assign some weights to determine importance. Larger weights signify that particular variables are of greater importance to the decision or outcome.

- W1 = 5, since large swells don’t come around often
- W2 = 2, since you’re used to the crowds
- W3 = 4, since you have a fear of sharks

Finally, we’ll also assume a threshold value of 3, which would translate to a bias value of –3. With all the various inputs, we can start to plug in values into the formula to get the desired output.

Y-hat = (1*5) + (0*2) + (1*4) – 3 = 6

If we use the activation function from the beginning of this section, we can determine that the output of this node would be 1, since 6 is greater than 0. In this instance, you would go surfing; but if we adjust the weights or the threshold, we can achieve different outcomes from the model. When we observe one decision, like in the above example, we can see how a neural network could make increasingly complex decisions depending on the output of previous decisions or layers.

In the example above, we used perceptrons to illustrate some of the mathematics at play here, but neural networks leverage sigmoid neurons, which are distinguished by having values between 0 and 1. Since neural networks behave similarly to decision trees, cascading data from one node to another, having x values between 0 and 1 will reduce the impact of any given change of a single variable on the output of any given node, and subsequently, the output of the neural network.

As we start to think about more practical use cases for neural networks, like image recognition or classification, we’ll leverage supervised learning, or labeled datasets, to train the algorithm. As we train the model, we’ll want to evaluate its accuracy using a cost (or loss) function. This is also commonly referred to as the mean squared error (MSE). In the equation below,

- i represents the index of the sample,
- y-hat is the predicted outcome,
- y is the actual value, and
- m is the number of samples.

$$Cost\ Function = MSE = 1/2m \sum 129_{(i=1)}^{m} ▒ (\hat{y}^{((i))} - y^{((i))})^2$$

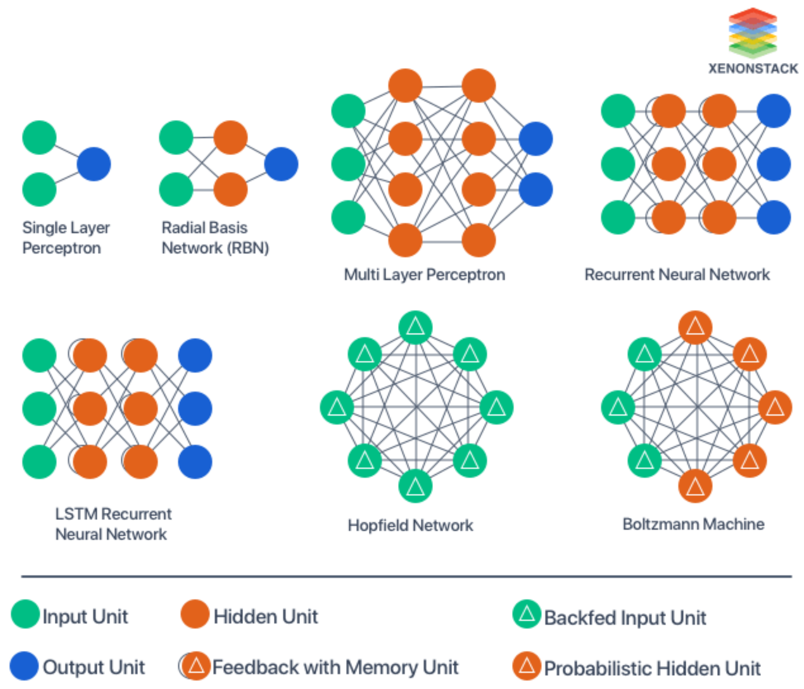
Ultimately, the goal is to minimize our cost function to ensure correctness of fit for any given observation. As the model adjusts its weights and bias, it uses the cost function and reinforcement learning to reach the point of convergence, or the local minimum. The process in which the algorithm adjusts its weights is through gradient descent, allowing the model to determine the direction to take to reduce errors (or minimize the cost function). With each training example, the parameters of the model adjust to gradually converge at the minimum.

Neural Network Architecture Types

- Neural Network Architecture Types

Artificial Neural Networks

- Perceptron Model in Neural Networks
- Radial Basis Function Neural Network
- Multilayer Perceptron Neural Network
- Recurrent Neural Network
- Long Short-Term Memory Neural Network (LSTM)
- Hopfield Network
- Boltzmann Machine Neural Network
- Convolutional Neural Network
- Modular Neural Network
- Physical Neural Network


XENONSTACK
Single Layer Perceptron
Radial Basis Network (RBN)
Multi Layer Perceptron
Recurrent Neural Network
LSTM Recurrent Neural Network
Hopfield Network
Boltzmann Machine
Input Unit
Hidden Unit
Backfed Input Unit
Output Unit
Feedback with Memory Unit
Probabilistic Hidden Unit