## Artificial Neural Network

Neural networks are machine learning models that mimic the complex functions of the human brain. These models consist of interconnected nodes or neurons that process data, learn patterns, and enable tasks such as pattern recognition and decision-making.

Neural networks are capable of learning and identifying patterns directly from data without pre-defined rules. These networks are built from several key components:

* **Neurons**: The basic units that receive inputs, each neuron is governed by a threshold and an activation function.
* **Connections**: Links between neurons that carry information, regulated by weights and biases.
* **Weights and Biases**: These parameters determine the strength and influence of connections.
* **Propagation Functions**: Mechanisms that help process and transfer data across layers of neurons.
* **Learning Rule**: The method that adjusts weights and biases over time to improve accuracy.

Learning in neural networks follows a structured, three-stage process:

* **Input Computation**: Data is fed into the network.
* **Output Generation**: Based on the current parameters, the network generates an output.
* **Iterative Refinement**: The network refines its output by adjusting weights and biases, gradually improving its performance on diverse tasks.

Layers in Neural Network Architecture

* **Input Layer:** This is where the network receives its input data. Each input neuron in the layer corresponds to a feature in the input data.
* **Hidden Layers:** These layers perform most of the computational heavy lifting. A neural network can have one or multiple hidden layers. Each layer consists of units (neurons) that transform the inputs into something that the output layer can use.
* **Output Layer:** The final layer produces the output of the model. The format of these outputs varies depending on the specific task (e.g., classification, regression). A diagram of a network

  AI-generated content may be incorrect.

## Working of Neural Networks

**Forward Propagation**

When data is input into the network, it passes through the network in the forward direction, from the input layer through the hidden layers to the output layer. This process is known as forward propagation. Here’s what happens during this phase:

1. **Linear Transformation:** Each neuron in a layer receives inputs, which are multiplied by the weights associated with the connections. These products are summed together, and a bias is added to the sum. This can be represented mathematically as:  where  represents the weights, x represents the inputs, and *b* is the bias.
2. **Activation:** The result of the linear transformation (denoted as z) is then passed through an activation function. The activation function is crucial because it introduces non-linearity into the system, enabling the network to learn more complex patterns. Popular activation functions include ReLU, sigmoid, and tanh.

**Backpropagation**

After forward propagation, the network evaluates its performance using a loss function, which measures the difference between the actual output and the predicted output. The goal of training is to minimize this loss. This is where backpropagation comes into play:

1. **Loss Calculation:** The network calculates the loss, which provides a measure of error in the predictions. The loss function could vary; common choices are mean squared error for regression tasks or cross-entropy loss for classification.
2. **Gradient Calculation:** The network computes the gradients of the loss function with respect to each weight and bias in the network. This involves applying the chain rule of calculus to find out how much each part of the output error can be attributed to each weight and bias.
3. **Weight Update:** Once the gradients are calculated, the weights and biases are updated using an optimization algorithm like stochastic gradient descent (SGD). The weights are adjusted in the opposite direction of the gradient to minimize the loss. The size of the step taken in each update is determined by the learning rate.

**Mathematical Explanation**

Forward propagation is the process of passing input data through the network to compute the output.

* Let be the input vector.
* Each neuron in the first hidden layer computes a weighted sum of the inputs:

**A black and white math symbol

AI-generated content may be incorrect.**

where:

*  is the weight connecting input ​ to neuron *j*.
* ​ is the bias term for neuron *j*.
* is the pre-activation value.

The weighted sum  is then passed through an activation function *f* to introduce non-linearity:



Common activation functions include:

* **Sigmoid**: 
* **ReLU (Rectified Linear Unit)**: 
* **Tanh**: 
* **Softmax**: A black text with a white background

  AI-generated content may be incorrect.

This process repeats for each layer until the output layer produces the final result. Then using the cost function, we will measure how well the network's output matches the true target values. For example,

A black text on a white background

AI-generated content may be incorrect.

A black text on a white background

AI-generated content may be incorrect.

A math equations and formulas

AI-generated content may be incorrect.

Backpropagation computes the gradients of the cost function with respect to the weights and biases. Let’s derive the gradients for each layer.

A math equations on a white background

AI-generated content may be incorrect. A math equations and formulas

AI-generated content may be incorrect.

where is the learning rate.

## Gradient Descend

**Batch Gradient Descent**

Batch Gradient Descent involves calculations over the full training set at each step as a result of which it is very slow on very large training data. Thus, it becomes very computationally expensive to do Batch GD. However, this is great for convex or relatively smooth error manifolds. Also, Batch GD scales well with the number of features. Batch gradient descent takes longer to converge since it computes the gradient using the entire training dataset in each iteration. Batch gradient descent is more accurate since it computes the gradient using the entire training dataset. Batch gradient descent requires more computation and memory since it needs to process the entire training dataset in each iteration.

**Stochastic Gradient Descent**

Stochastic Gradient Descent (SGD) is a variant of the [Gradient Descent](https://www.geeksforgeeks.org/gradient-descent-algorithm-and-its-variants/)algorithm that is used for optimizing [machine learning](https://www.geeksforgeeks.org/machine-learning-algorithms/) models. It addresses the computational inefficiency of traditional Gradient Descent methods when dealing with large datasets in machine learning projects.

In SGD, instead of using the entire dataset for each iteration, only a single random training example (or a small batch) is selected to calculate the gradient and update the model parameters. This random selection introduces randomness into the optimization process, hence the term “stochastic” in stochastic Gradient Descent.

The advantage of using SGD is its computational efficiency, especially when dealing with large datasets. By using a single example or a small batch, the computational cost per iteration is significantly reduced compared to traditional Gradient Descent methods that require processing the entire dataset.

One thing to be noted is that, as SGD is generally noisier than typical Gradient Descent, it usually took a higher number of iterations to reach the minima, because of the randomness in its descent. Even though it requires a higher number of iterations to reach the minima than typical Gradient Descent, it is still computationally much less expensive than typical Gradient Descent. Hence, in most scenarios, SGD is preferred over Batch Gradient Descent for optimizing a learning algorithm.

**Mini-Batch Gradient Descent**

Mini-Batch Gradient Descent is a middle ground between Batch Gradient Descent and Stochastic Gradient Descent (SGD). Instead of updating parameters using all data (Batch GD) or one example (SGD), it updates parameters using a small batch of examples (e.g., 32, 64, or 128 samples per batch) at each step. Typical mini-batch sizes in practice

* Small datasets → 16,32
* Medium datasets → 64,128
* Large datasets → 256,512,1024

A screenshot of a computer

AI-generated content may be incorrect.

**Difference between Stochastic Gradient Descent & batch Gradient Descent**

A screenshot of a white grid

AI-generated content may be incorrect.

A screenshot of a white box

AI-generated content may be incorrect.

## Regularization

Regularization is a technique used to **prevent overfitting** in neural networks by adding constraints or penalties to the model during training. It helps the model generalize better to unseen data. Types of Regularization in Neural Networks:

**1.L1 and L2 Regularization (Weight Decay)**

These add a penalty term to the loss function to discourage large weight values.

* L1 Regularization (Lasso): Adds the sum of absolute values of weights to the loss function. Encourages sparsity (some weights become exactly zero). Useful for feature selection.



* L2 Regularization (Ridge): Adds the sum of squared weights to the loss function. Prevents overly large weight values. Helps with weight decay, leading to smoother models.



* **Elastic Net (L1 + L2 Combined):** Uses **both** L1 and L2 penalties. Helps when **L1 alone is too aggressive** in removing features.



* L2 (Ridge) is more commonly used in deep learning than L1. The lambda (λ) parameter controls the strength of regularization in L1 (Lasso) and L2 (Ridge) regularization. Choosing the right range for λ depends on the dataset and model complexity. General Range of λ
* Typical values: 
* Small λ → Minimal regularization (model may overfit).
* Medium λ → Balanced regularization.
* Large λ → Strong regularization (model may underfit).

**2. Dropout**

Randomly drops (sets to zero) a fraction of neurons in each training step. Forces the network to not rely too much on specific neurons, improving generalization.

* Prevents co-adaptation of neurons.
* Reduces overfitting.
* Should be turned off during inference (testing).
* Typical values: Dropout rate = 0.2 to 0.5 (20-50% of neurons dropped).

Formula: If neuron activation is  dropout applies:

A black and white image of a mathematical equation

AI-generated content may be incorrect.where:

* M = dropout mask (randomly 0 or 1)
* p = keep probability (e.g., 0.8 means 80% neurons kept)

**3.Data Augmentation**

* Artificially increases dataset size by applying transformations (flipping, rotation, cropping, etc.).
* Helps prevent the network from memorizing training examples.
* Common in image-based tasks (used in CNNs).
* Makes model more robust
* No change in network architecture.
* Only applicable for specific data types (e.g., images, text, audio).

**Which Regularization Should You Use?**

* For fully connected networks (MLPs) → Use L2 + Dropout.
* For CNNs → Use BatchNorm + Dropout + Data Augmentation.
* For RNNs/LSTMs → Use Dropout + L2 Regularization.
* For small datasets → Use Data Augmentation + L2.
* For large datasets → Use BatchNorm + L2 Regularization.