Mathematics Behind Artificial Neural Networks

1. Neurons and Layers

Neuron (Perceptron)

A single neuron receives inputs, applies weights, adds a bias, and passes the result through an activation function. Mathematically, this can be expressed as:

For a neuron j, with inputs x_i, weights w_{ij}, and bias b_j:

$$z_j = \sum_{i=1}^{n} w_{ij} x_i + b_j \tag{1}$$

Where:

- x_i are the inputs.
- w_{ij} are the weights.
- b_j is the bias.
- z_j is the linear combination of inputs and weights (also called the pre-activation value).

The output of the neuron a_j after applying the activation function σ is:

$$a_j = \sigma(z_j) \tag{2}$$

Activation Function

Common activation functions $\sigma(z)$ include:

- **Sigmoid**: $\sigma(z) = \frac{1}{1+e^{-z}}$
- **ReLU**: $\sigma(z) = \max(0, z)$
- **Tanh**: $\sigma(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
- **Softmax**: $\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$ (for multi-class classification)

2. Layers

Input Layer

The input layer consists of neurons that receive the initial data inputs.

Hidden Layers

Each hidden layer applies a transformation to the input received from the previous layer using neurons. The number of neurons in each hidden layer and the activation function can vary.

Output Layer

The output layer provides the final output of the network. The activation function used in the output layer depends on the type of problem (e.g., softmax for classification, linear for regression).

3. Forward Propagation

Forward propagation refers to the process of passing input data through the layers of the network to generate an output. This involves calculating the outputs of all neurons in each layer, starting from the input layer and moving forward to the output layer.

4. Loss Function

The loss function quantifies the difference between the predicted output and the actual target. The goal is to minimize this loss. Common loss functions include:

- **Mean Squared Error (MSE)**:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3}$$

  (for regression)

- **Cross-Entropy Loss**:

$$\text{CE} = -\sum_{i=1}^{n} y_i \log(\hat{y}_i) \tag{4}$$

  (for classification)

  Where:

- y_i are the true values.
- $\hat{y}_i$ are the predicted values.

5. Backward Propagation

Backward propagation is used to update the weights and biases of the network to minimize the loss function. This involves computing the gradient of the loss function with respect to each weight and bias using the chain rule of calculus. The gradients indicate the direction in which the weights and biases should be adjusted to decrease the loss.

Gradient Calculation

For a given weight w_{ij}, the gradient of the loss L with respect to w_{ij} is:

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_{ij}} \tag{5}$$

Where:

- $\hat{y}$ is the output of the neuron.
- z is the pre-activation value.

Weight Update

Weights are updated using an optimization algorithm, such as Stochastic Gradient Descent (SGD) or Adam. For SGD, the update rule is:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \eta \frac{\partial L}{\partial w_{ij}} \tag{6}$$

Where:

- $w_{ij}^{(t)}$ is the weight at time step t.
- η is the learning rate.

6. Optimization and Training

The training process involves iteratively performing forward and backward propagation over the dataset to minimize the loss function. This process continues until the model reaches satisfactory performance or a predefined number of iterations (epochs).

7. Evaluation

The trained model is evaluated on a separate test dataset to assess its performance. Common metrics include accuracy, precision, recall, F1-score (for classification), and Mean Absolute Error (MAE) (for regression).