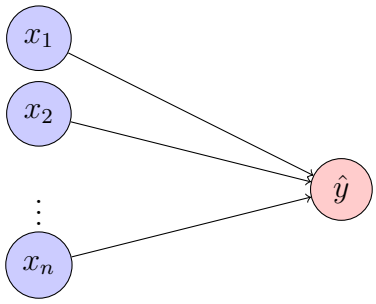
Mathematical Formulation of Perceptron, Logistic Regression, and Multilayer Perceptron

1. Perceptron

Model Architecture

The perceptron is a simple neural network with a single linear layer and a threshold activation function. Below is its architecture:



Vector Representation of Data

The input data is represented as a vector:

$$\mathbf{x} = [x_1, x_2, \ldots, x_n]^\top,$$

where x_i represents individual input features. The weights of the model are:

$$\mathbf{w} = [w_1, w_2, \ldots, w_n]^\top,$$

and b is the bias term. The output $\hat{y}$ is a binary value, $\hat{y} \in \{-1, 1\}$.

Mathematical Formulations

Linear Combination:

$$z = \mathbf{w}^\top \mathbf{x} + b$$

Here, z is the weighted sum of the inputs, with the bias b added to shift the result.

Activation Function:

$$\hat{y} = \text{sign}(z)$$

The sign function outputs 1 if $z > 0$ and -1 otherwise.

Loss Function

The perceptron uses a hinge loss function to penalize misclassified samples:

$$L(\mathbf{w}, b) = -\sum_i \delta(y_i(\mathbf{w}^\top \mathbf{x}_i + b) \leq 0),$$

where δ is an indicator function that equals 1 if the condition is true and 0 otherwise. This ensures only misclassified samples contribute to the loss.

Gradient Descent and Weight Updates

Gradient descent is used to optimize the model parameters. The weight update rule is:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$$

and the bias update rule is:

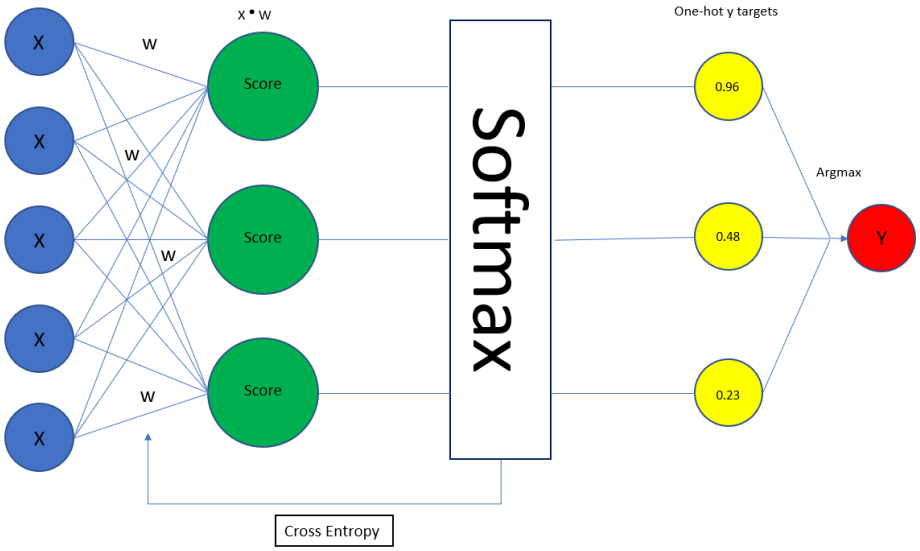
$$b \leftarrow b + \eta y_i$$

Here, η is the learning rate, a hyperparameter controlling the step size during optimization.

2. Logistic Regression

Model Architecture

Logistic regression is a single-layer model used for binary classification. Unlike the perceptron, it uses a sigmoid activation function to output probabilities.



Vector Representation of Data

The input vector $\mathbf{x}$ and weights $\mathbf{w}$ are the same as in the perceptron. However, the output $\hat{y} \in [0, 1]$ represents a probability.

Mathematical Formulations

Linear Combination:

$$z = \mathbf{w}^\top \mathbf{x} + b$$

Activation Function (Sigmoid):

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

The sigmoid function maps the linear combination z into a probability.

Loss Function

The loss function is the binary cross-entropy loss, which measures the difference between predicted probabilities and true labels:

$$L(\mathbf{w}, b) = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

Gradient Descent and Weight Updates

Gradients:

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i) \mathbf{x}_i$$

$$\frac{\partial L}{\partial b} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)$$

Update Rules:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial L}{\partial \mathbf{w}}, \quad b \leftarrow b - \eta \frac{\partial L}{\partial b}$$

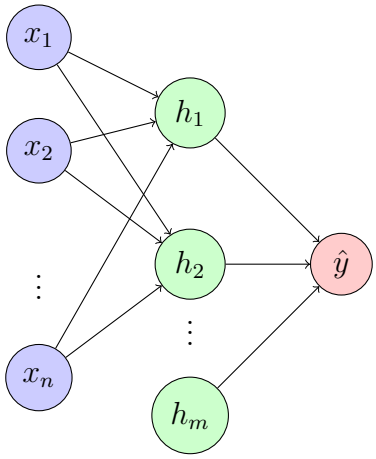
Explanation of Gradient Descent

Gradient descent adjusts the weights and biases in the direction of the negative gradient to minimize the loss. The learning rate η controls how much the weights change with each iteration.

3. Multilayer Perceptron (MLP)

Model Architecture

An MLP consists of multiple layers. Below is a representation with one hidden layer:



Mathematical Formulations

Hidden Layer:

$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

Here, σ is an activation function (e.g., ReLU or sigmoid), $\mathbf{W}^{(1)}$ are the weights, and $\mathbf{b}^{(1)}$ are the biases for the hidden layer.

Output Layer:

$$\hat{y} = \sigma(\mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)})$$

Loss Function: Cross-entropy is commonly used:

$$L = -\sum_i [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

Backpropagation: Gradients are computed for each layer starting from the output and propagating backward.