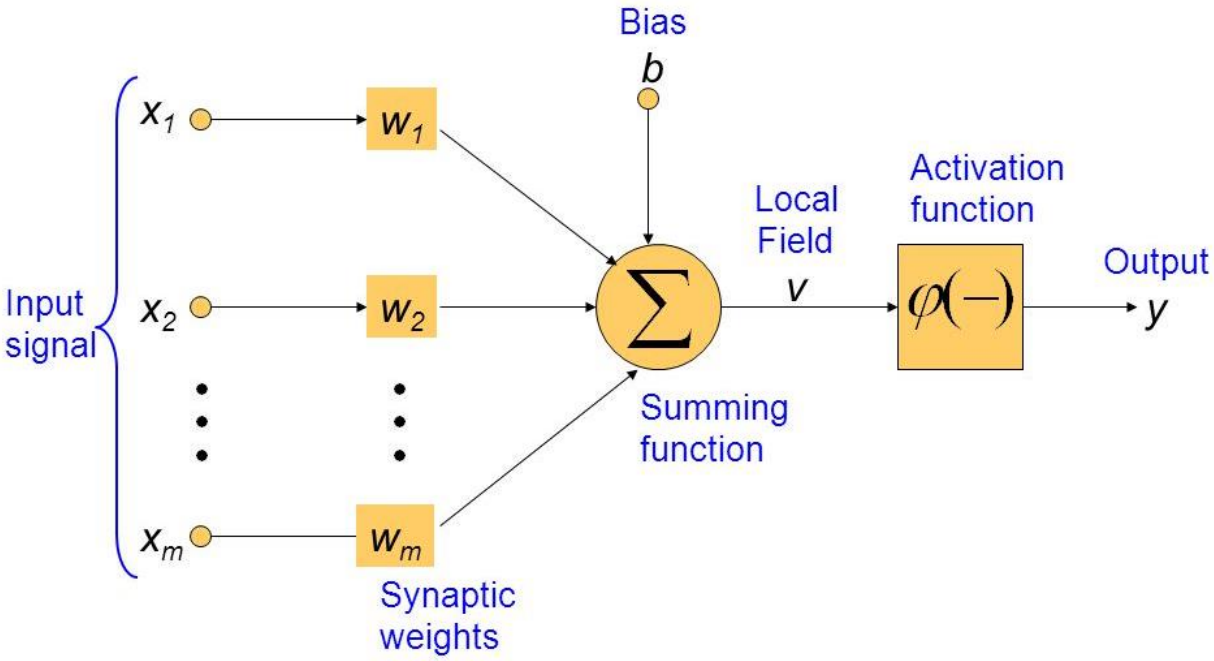
Perceptron

1. **Model Architecture**
   The perceptron is one of the simplest types of artificial neural networks. It consists of a single layer of nodes where each node represents a neuron. Each neuron receives a weighted sum of inputs and applies an activation function to produce an output.



2. **Vector Representation**
   In mathematical terms, the perceptron handles data as vectors. Inputs are represented $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, and each input is associated with a corresponding weight $\mathbf{w} = [w_1, w_2, \ldots, w_n]$. A bias term b ensures that the model can represent more complex decision boundaries by shifting the activation threshold.

3. **Mathematical Formulation**

   - **Linear Combination**:
     The perceptron computes the weighted sum of its inputs and adds a bias term:

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

   - This forms the basis for determining whether the output should be 0 or 1.

   - **Activation Function**:
     The step activation function converts z into a binary output:

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

     This function ensures the perceptron makes a hard decision for classification tasks.

4. **Prediction Calculation**
   The perceptron predicts the class label y by applying the step activation function to z.

This results in a binary classification, where the output is either y=1 or y=0. It is important to note that perceptrons can only handle linearly separable data, as the decision boundary is a hyperplane in the input space.

5. **Gradient Descent (Training)**
    - **Learning Rule**:
      Training a perceptron involves updating its weights and bias using the perceptron learning rule, which minimizes classification errors. For each training example, the weights are adjusted as follows:

      $$w_i := w_i + \eta \cdot (y - \hat{y}) \cdot x_i$$

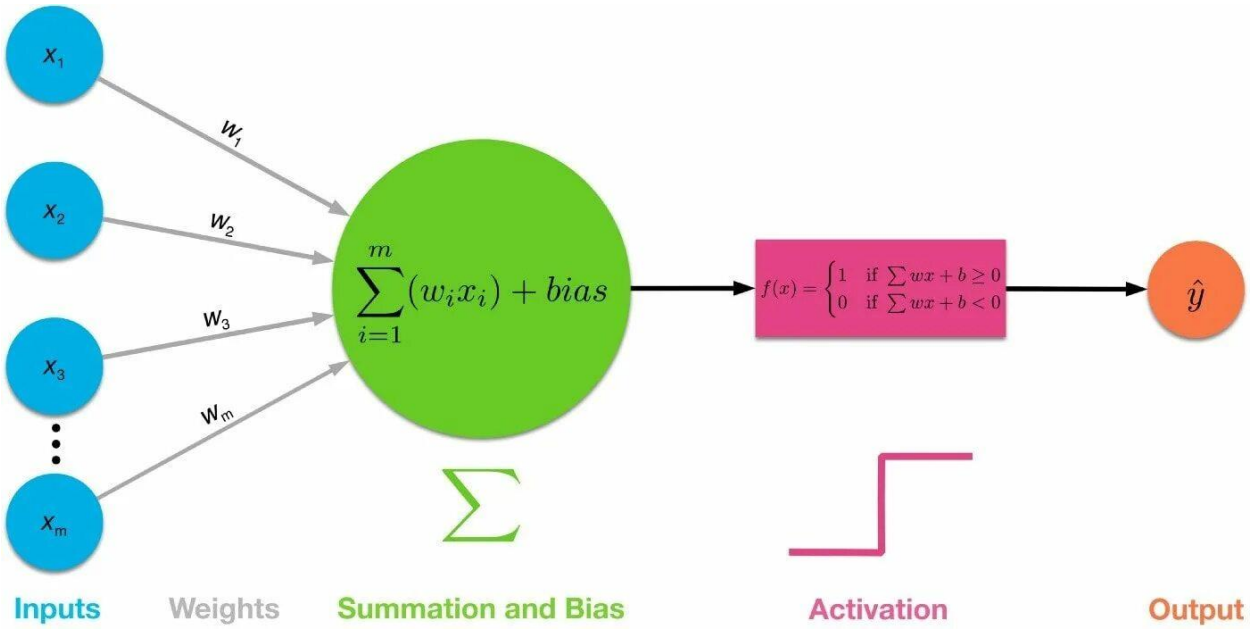
      Here, η is the learning rate, controlling the step size of updates.
    - **Bias Update**:
      Similarly, the bias is updated to reflect the error in predictions:

      $$b := b + \eta \cdot (y - \hat{y})$$

      This iterative process continues until the model achieves satisfactory accuracy or convergence.

Logistic Regression

1. **Model Architecture**
   Logistic regression is a fundamental algorithm used for binary classification. It extends the perceptron by introducing the sigmoid activation function, which outputs probabilities instead of hard binary decisions.



2. **Vector Representation**
   The inputs, weights, and bias are similar to the perceptron, with:

   $$\mathbf{x} = [x_1, x_2, \ldots, x_n], \quad \mathbf{w} = [w_1, w_2, \ldots, w_n], \quad b = \text{bias}$$

The output $\hat{y}$ represents the probability that the input belongs to a particular class, with $\hat{y} \in (0, 1)$

3. **Mathematical Formulation**
    - **Linear Combination**:
      Logistic regression starts by computing z: z=w·x+b
    - **Activation Function (Sigmoid)**:
      To map z into a probability, the sigmoid function is applied:

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

      The sigmoid function ensures the output lies in the range (0, 1), making it interpretable as a probability.
    - **Loss Function (Cross-Entropy)**:
      Logistic regression minimizes the cross-entropy loss to optimize its parameters:

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

4. **Prediction Calculation**
   Predictions are calculated as probabilities using $\hat{y} = \sigma(z)$. A threshold of 0.5 is commonly used to make binary decisions:

$$y = \begin{cases} 1 & \text{if } \hat{y} \geq 0.5 \\ 0 & \text{if } \hat{y} < 0.5 \end{cases}$$

   Gradient Descent (Training)
    - The weights and bias are updated using the gradients of the loss function:

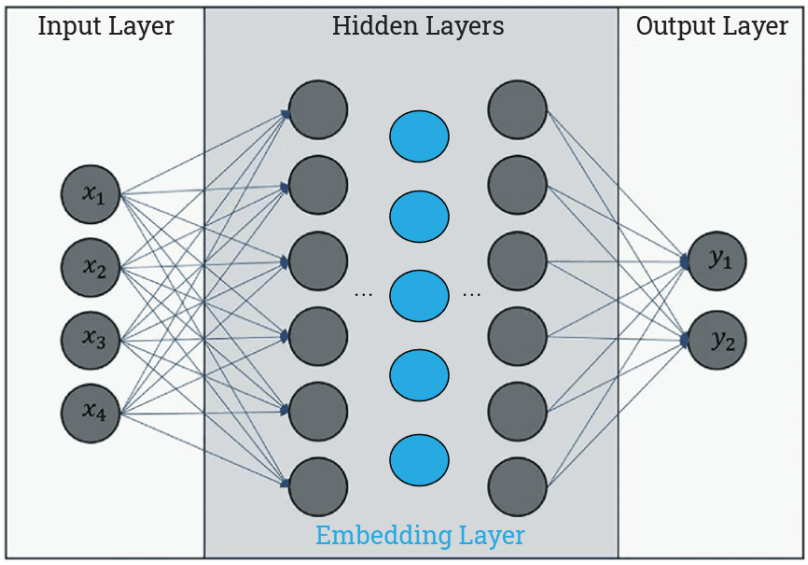
$$w_i := w_i - \eta \cdot (\hat{y} - y) \cdot x_i$$

    - The bias is similarly updated:

$$b := b - \eta \cdot (\hat{y} - y)$$

Multilayer Perceptron (MLP)

1. **Model Architecture**
   An MLP extends logistic regression by introducing one or more hidden layers, each containing multiple neurons. Each neuron applies an activation function, enabling the model to learn non-linear decision boundaries.



2. **Vector Representation**
   Inputs and weights are represented as:

$$\mathbf{x} = [x_1, x_2, \ldots, x_n], \quad \mathbf{W}^{[1]}, \mathbf{b}^{[1]}, \mathbf{W}^{[2]}, \mathbf{b}^{[2]}, \ldots$$

Outputs $\hat{y}$ represent class probabilities for multi-class classification tasks.

3. **Mathematical Formulation**
   - **Forward Propagation**:
     Each layer computes:

$$z^{[l]} = \mathbf{W}^{[l]} \cdot \mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}$$

$$a^{[l]} = \text{activation}(z^{[l]})$$

     Here, $a^{[l-1]}$ is the activation from the previous layer.

   - **Output Layer (Softmax)**:
     For multi-class classification:

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

4. **Prediction Calculation**
   Predictions are based on the maximum output probability:

$$\text{Class} = \arg\max_i \hat{y}_i$$

5. **Gradient Descent (Backpropagation)**
    - Gradients are computed for each layer to update weights and biases:

$$\frac{\partial L}{\partial \mathbf{W}^{[l]}} = \delta^{[l]} \cdot (\mathbf{a}^{[l-1]})^T$$

    - Backpropagation calculates $\delta^{[l]}$ for each layer recursively:

$$\delta^{[l]} = (\mathbf{W}^{[l+1]})^T \delta^{[l+1]} \odot \text{activation}'(z^{[l]})$$

Conclusion

In this report, we explored three foundational algorithms in machine learning: the Perceptron, Logistic Regression, and Multilayer Perceptron (MLP). Each of these models plays a crucial role in understanding the evolution of machine learning techniques, from basic linear classifiers to more complex neural network architectures.

The Perceptron laid the groundwork for machine learning by introducing the concept of iterative weight updates to classify linearly separable data. Logistic Regression expanded on this by introducing a probabilistic framework, allowing for outputs as probabilities and providing a more interpretable model. Finally, the MLP demonstrated the power of neural networks in modeling non-linear relationships, paving the way for modern deep learning applications.

Through mathematical formulations, vectorized representations, and gradient descent optimization, we have seen how these algorithms are designed to make accurate predictions while minimizing error. Visualizations of model architectures and the decision boundaries further illustrate their practical applications.

As machine learning continues to evolve, these algorithms remain relevant—not only as tools for solving specific problems but also as foundational building blocks for understanding more advanced models. The simplicity and power of these methods highlight the importance of core concepts in achieving breakthroughs in artificial intelligence.