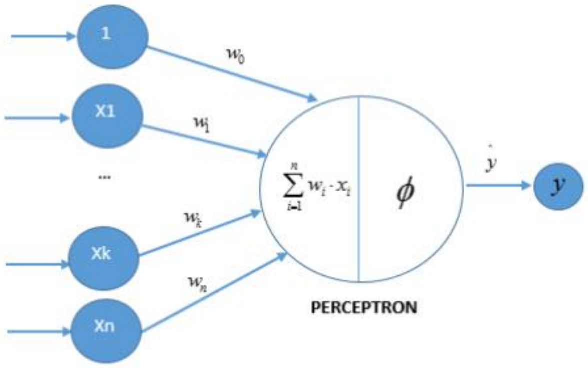
Ha Hoang Giang – 5130203/20102

Report on the Perceptron Algorithm

Model Architecture : The Perceptron is the simplest type of neural network. It consists of:

- **Input Layer:** Takes input features.
- **Linear Combination:** Computes a weighted sum of the inputs and adds a bias term.
- **Activation Function:** A step function determines the binary output.



Vector Representation of Data

- **Inputs:** x=[x{1}, x{2} ,...,x{n}]
- **Weights:** w=[w{1}, w{2} ,...,w{n}]
- **Bias:** b
- **Output:** y∈{0,1}

Mathematical Formulation

1. **Linear Combination:** $$z = \mathbf{w}^T\mathbf{x} + b = \sum_{i=1}^{n} w_i x_i + b$$
2. **Activation Function (Step Function):** $$f(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$
3. **Loss Function:** To measure prediction error, we use: $$L(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$$

Prediction Calculation

The Perceptron predicts output (y^\hat{y}y^) based on the following steps:

1. Compute z, the weighted sum of inputs plus bias: $$L(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$$
2. Apply the activation function f(z)f(z)f(z): $$\frac{\partial L}{\partial w_i} = -(y - \hat{y})x_i$$ $$\frac{\partial L}{\partial b} = -(y - \hat{y})$$

3. Update Rules:

- Update weights: $$w_i = w_i - \eta \frac{\partial L}{\partial w_i}$$
- Update bias: $$b = b - \eta \frac{\partial L}{\partial b}$$

Here, η is the learning rate.

Summary

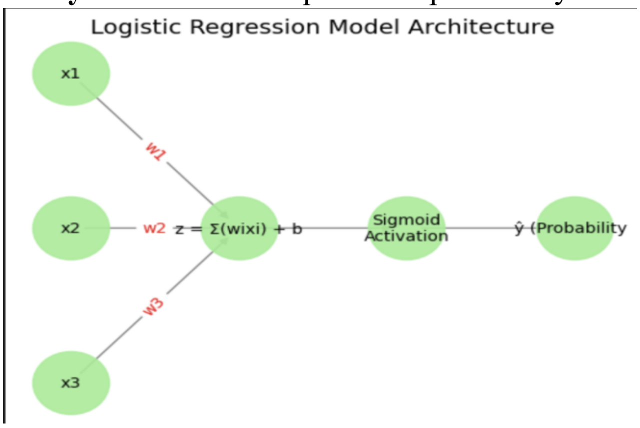
- **Perceptron** is a linear classifier that calculates a weighted sum of inputs, applies a step function, and updates its weights using gradient descent.
- It learns a decision boundary to separate two classes in the feature space.

Report on Logistic Regression

Model Architecture: Logistic Regression is a binary classification algorithm that applies the sigmoid function to a linear combination of inputs. The structure is as follows:

- **Input Layer:** Takes input features.
- **Linear Combination:** Computes a weighted sum of the inputs and adds a bias term.
- **Activation Function (Sigmoid):** Converts the linear combination into a probability value.
- **Output Layer:** Produces the predicted probability of the positive class.

Diagram:



Vector Representation of Data

- **Inputs:** x=[x1,x2,…,xn]
- **Weights:** w=[w1,w2,…,wn]
- **Bias:** b
- **Output (Probability):** y^∈[0,1]

Mathematical Formulation

1. **Linear Combination:**

The linear combination of inputs is computed as: $$z = \mathbf{w}^T\mathbf{x} + b = \sum_{i=1}^{n} w_i x_i + b$$

2. **Activation Function (Sigmoid):**

The sigmoid function transforms z into a probability value: $f(z) = \frac{1}{1+e^{-z}}$

3. **Loss Function (Binary Cross-Entropy):** To quantify the difference between predicted (y^) and true values (y), we use the binary cross-entropy loss:

$$L(y, \hat{y}) = -\left[y \log(\hat{y}) + (1-y)\log(1-\hat{y})\right]$$

Prediction Calculation

Logistic Regression predicts the output (y^{y}) in the following steps:

1. Compute the linear combination: $z = \mathbf{w}^T\mathbf{x} + b$
2. Apply the sigmoid function to zzz to get the probability: $\hat{y} = f(z) = \frac{1}{1+e^{-z}}$

Gradient Descent Algorithm

The objective of the gradient descent algorithm is to minimize the binary cross-entropy loss L(y,y^) by iteratively updating the weights and bias.

1. **Objective:** Minimize: $J(\mathbf{w}, b) = \frac{1}{m}\sum_{i=1}^{m} L(y^{(i)}, \hat{y}^{(i)})$

where m is the number of training samples.

2. **Gradient Formulas:** The gradients of the loss with respect to the weights and bias are:
   - For weight wi: $\frac{\partial J}{\partial w_i} = \frac{1}{m}\sum_{j=1}^{m}\left(\hat{y}^{(j)} - y^{(j)}\right)x_i^{(j)}$
   - For bias b: $\frac{\partial J}{\partial b} = \frac{1}{m}\sum_{j=1}^{m}\left(\hat{y}^{(j)} - y^{(j)}\right)$
3. **Weight and Bias Updates:** Using the gradients, the weights and bias are updated as follows:
   - Weight update: $w_i = w_i - \eta\frac{\partial J}{\partial w_i}$

- Bias update: $b = b - \eta \frac{\partial J}{\partial b}$

Here, η is the learning rate.

Summary

- Logistic Regression is a fundamental binary classifier that transforms a linear combination of inputs into a probability using the sigmoid function.
- The model minimizes the binary cross-entropy loss via gradient descent to iteratively update weights and bias.
- Its ability to output probabilities makes it widely applicable to binary classification tasks.

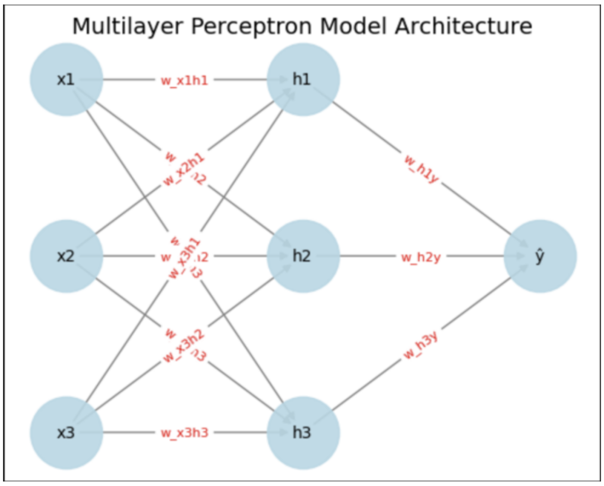
Report on Multilayer Perceptron (MLP)

Model Architecture: The Multilayer Perceptron (MLP) is a feedforward neural network consisting of one or more hidden layers between the input and output layers. Each neuron performs a linear combination of its inputs, applies an activation function, and passes the output to the next layer.

Architecture:

- **Input Layer:** Accepts the input features.
- **Hidden Layers:** Each layer consists of neurons with weights, biases, and nonlinear activation functions.
- **Output Layer:** Computes predictions (probabilities or class scores).

Diagram:



Vector Representation of Data

- **Inputs:** x=[x1,x2,…,xn]
- **Weights (Layer** L **):** W[l] (matrix of weights for layer L)
- **Bias (Layer L):** b[l] (vector of biases for layer L)
- **Outputs (Activations for Layer L):** a[l]
- **Final Output:** y^ (predicted value)

Mathematical Formulation

1. **Linear Combination (Layer L):** For each neuron in a layer:

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

where a^[L−1] is the activation output of the previous layer (or input data if L=1).

2. **Activation Function:** Apply a nonlinear function to z^{L} to introduce nonlinearity. Common activation functions:
   - **ReLU:** $f(z) = \max(0, z)$
   - **Sigmoid:** $f(z) = \frac{1}{1+e^{-z}}$
   - **Tanh:** $f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

Activations: $\mathbf{a}^{[l]} = f(z^{[l]})$

3. **Loss Function:** The loss function quantifies the error between predicted values y^ and true values y. For binary classification:

$$L(y, \hat{y}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1 - y^{(i)})\log(1 - \hat{y}^{(i)})\right]$$

For regression, a common choice is the Mean Squared Error: $L(y, \hat{y}) = \frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$

Prediction Calculation

1. **Forward Propagation:** Predictions are calculated by propagating inputs through all layers:
   - Compute z[l]: $z^{[l]} = \mathbf{W}^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}$
   - Compute activations: $\mathbf{a}^{[l]} = f(z^{[l]})$
2. **Final Output:** At the output layer (L): $\hat{y} = \mathbf{a}^{[L]}$

Gradient Descent Algorithm

Gradient Descent minimizes the loss L(y,y^) by updating weights W^[l] and biases b^[l] using the gradients of the loss.

1. **Backward Propagation:** Gradients are computed for each layer starting from the output layer (using the chain rule).

- Output layer gradient: $\delta^{[L]} = \frac{\partial L}{\partial z^{[L]}} = \hat{y} - y$
- Hidden layer gradients (for l < L): $\delta^{[l]} = \left(\mathbf{W}^{[l+1]T}\delta^{[l+1]}\right) \odot f'(z^{[l]})$

where f'(z^[l]) is the derivative of the activation function.

2. **Gradient Formulas:**
   - Gradients of weights: $\frac{\partial L}{\partial \mathbf{W}^{[l]}} = \delta^{[l]}\mathbf{a}^{[l-1]T}$
   - Gradients of biases: $\frac{\partial L}{\partial \mathbf{b}^{[l]}} = \delta^{[l]}$
3. **Update Rules:**
   - Update weights $\mathbf{W}^{[l]} = \mathbf{W}^{[l]} - \eta\frac{\partial L}{\partial \mathbf{W}^{[l]}}$
   - Update biases: $\mathbf{b}^{[l]} = \mathbf{b}^{[l]} - \eta\frac{\partial L}{\partial \mathbf{b}^{[l]}}$

Here, η is the learning rate.

Summary

- **Multilayer Perceptron (MLP):** A neural network with one or more hidden layers that can model complex, nonlinear relationships.
- **Forward Propagation:** Calculates the predicted values (y^) layer by layer.
- **Backward Propagation:** Updates weights and biases by propagating gradients backward through the network.
- **Optimization:** Uses gradient descent to minimize the loss function.