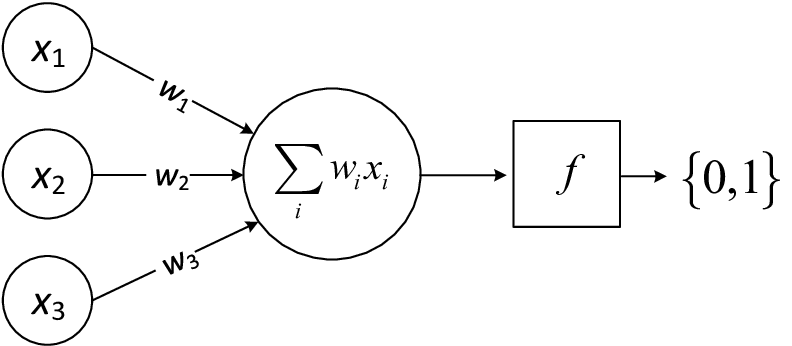
**EXPERIMENT 3**

**Aim: Design a single unit perceptron for classification of iris dataset without using predefined models**

**Description:**

****

A perceptron is a simple linear classifier that makes predictions based on a linear decision boundary. The goal of this experiment is to implement a **Single Unit Perceptron** from scratch to classify the **Iris dataset** without using predefined machine learning models. The perceptron will handle binary classification by distinguishing between two classes: Iris-setosa and Iris-versicolor.

The perceptron algorithm adjusts weights iteratively using the Perceptron Learning Rule until it converges or reaches a maximum number of iterations. The experiment will demonstrate:

1. **Loading and preprocessing the data.**
2. **Designing the perceptron algorithm.**
3. **Training the perceptron on the dataset.**
4. **Evaluating its performance on the test set.**

**Libraries used:**

Numpy (np):

* Description and Usage: Numpy supports large, multi-dimensional arrays and matrices, along with high-level mathematical functions. It's used for numerical operations like creating arrays, matrix multiplication, initializing weights, and calculating gradients and costs in the perceptron model.

Scikit-learn (sklearn.datasets):

* Description and Usage: Scikit-learn provides tools for data mining and analysis. It's used to load the Iris dataset for training and testing the perceptron model.

Matplotlib (matplotlib.pyplot as plt):

* Description and Usage: Matplotlib is a plotting library for embedding plots. It's used to plot the decision boundary and visualize data points of the Iris dataset, showing the perceptron's classification performance.

**Operations Performed:**

Loading the Iris Dataset:

* The Iris dataset is loaded using Scikit-learn's load\_iris function. This standard dataset is used for training and testing the perceptron model. Only the first 100 samples and the first two features (sepal length and sepal width) are selected for binary classification.

Data Preparation:

* The selected data (first 100 samples and first two features) is prepared for binary classification by creating feature and target arrays for training the perceptron model.

Defining Learning Parameters:

* Key parameters, including the learning rate (0.1) and the number of iterations (100), are defined to control the step size and the number of updates during training.

Initializing Weights:

* The weights of the perceptron are randomly initialized using Numpy. These weights are updated during training based on the calculated errors.

Defining the Activation Function:

* A simple threshold activation function is defined, classifying input based on whether it is greater than or equal to 0, returning 1 if true and 0 otherwise. This function determines the predicted class during training.

Training the Perceptron:

* The perceptron is trained iteratively over 100 iterations. In each iteration, the output for each sample is computed, errors are calculated, and weights are updated using gradient descent. The cost (error) for each iteration is calculated and printed to monitor progress.

Plotting the Decision Boundary:

* After training, the decision boundary is plotted using Matplotlib. The Iris dataset points are visualized to show how well the perceptron separates the two classes, illustrating the model's effectiveness.