M2 Evidence

Karla Stefania Cruz Muñiz A01661547

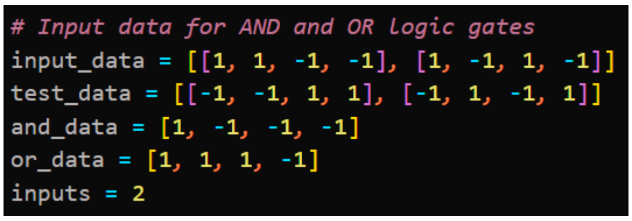
Selected algorithm: Hebb

This algorithm is based on the idea that the weight of a connection between two neurons should increase if both neurons are activated at the same time. It's a single layer neural network where there's an input layer and an output layer. "Hebbian rule works by updating the weights between neurons in the neural network for each training sample." (Jain)

Code explanation

Declaration of input data

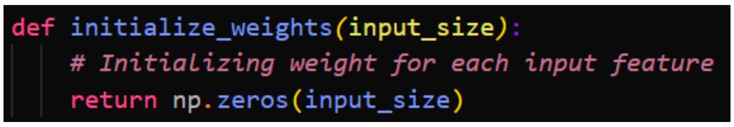
Input data is the data for logic gates AND and OR. Each sublist represents the inputs from x1 and x2. And data is the y expected results for AND logic gate, same as or data but for OR logic gate, this is used for guided training. Lastly, the inputs variable saves the quantity of inputs the algorithm is going to receive, in this case the number of inputs is two, x1 and the x2.



```
# Input data for AND and OR logic gates
input_data = [[1, 1, -1, -1], [1, -1, 1, -1]]
test_data = [[-1, -1, 1, 1], [-1, 1, -1, 1]]
and_data = [1, -1, -1, -1]
or_data = [1, 1, 1, -1]
inputs = 2
```

Initializing weights function

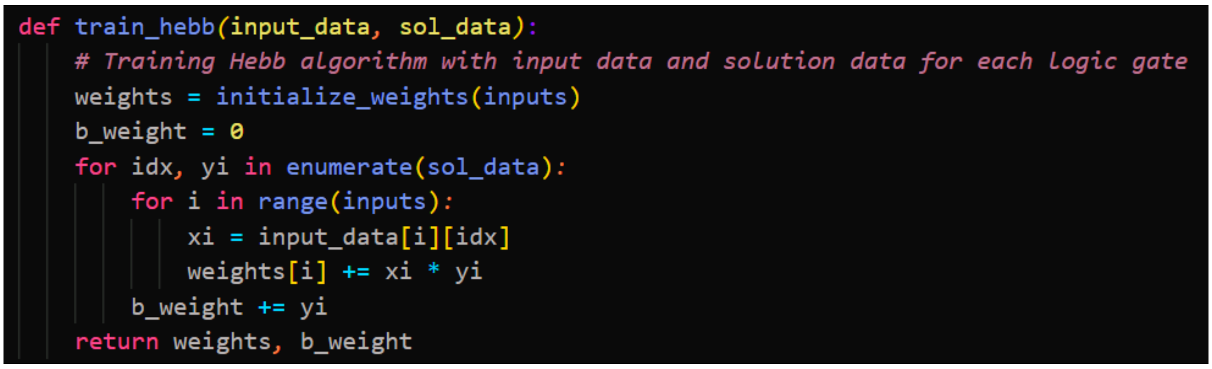
For the model we are initializing weights of the connections between neurons with zeros, for having the guarantee weights are not previously influenced by some kind of non-neutral number.



```
def initialize_weights(input_size):
    # Initializing weight for each input feature
    return np.zeros(input_size)
```

Training function

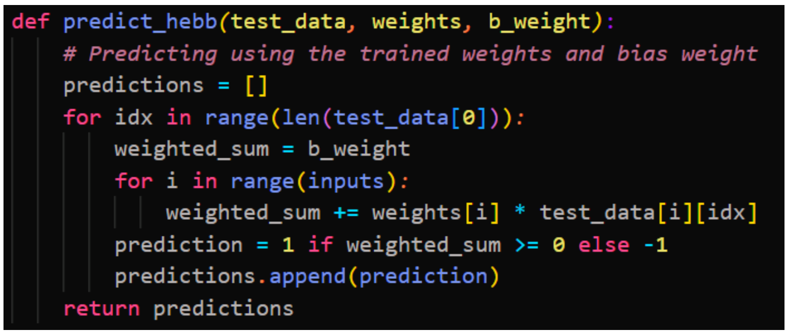
This function works with a given solution data, so it can be used either for AND or OR logic gates. This works by updating the weights of the paths with the function $w_n(new) = w_n(old) + x_n y$ and also updating the weight of the bias path with the formula $b(new) = b(old) + y$. The first for iterates through each output value in the solution data set and its index, then, the second for iterates through input data so it can only take two values, 0 or 1, where 0 correspond to the x1 data and 1 to the x2 data, updating its weights at the weights array. So we are accessing data in order with input_data[i][idx], which means the first index is for the list we want to visit and the second for the index in that list. This function returns the final weights after the iteration, from x1, x2 and the bias.



```python
def train_hebb(input_data, sol_data):
    # Training Hebb algorithm with input data and solution data for each logic gate
    weights = initialize_weights(inputs)
    b_weight = 0
    for idx, yi in enumerate(sol_data):
        for i in range(inputs):
            xi = input_data[i][idx]
            weights[i] += xi * yi
        b_weight += yi
    return weights, b_weight
```

Prediction function

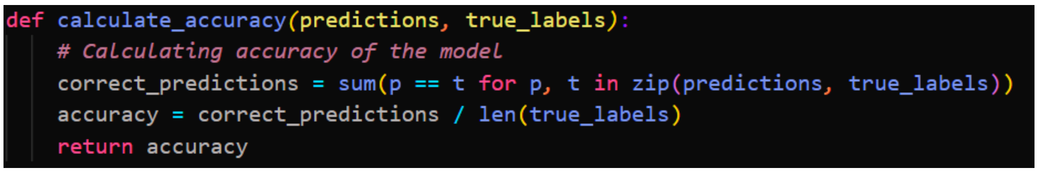
This is the function where we predict using the resulting weights from training. The for iterates over the test dataset length giving the number of test cases; since this algorithm is tested for the results of a logic gate there are only four possible combinations considering there are only two inputs, so there are 4 test cases, then, weighted_sum is the variable which initializes the weighted sum with the bias. The second for iterates over each input feature and for each input data set, calculate the weighted sum using the learned weights and the bias weight. Use a threshold of 0 to determine if the output is 1 or -1. If the weighted sum is greater than or equal to 0, the prediction is 1; otherwise, it is -1; at the end, it returns a list of predictions for all inputs.



```python
def predict_hebb(test_data, weights, b_weight):
    # Predicting using the trained weights and bias weight
    predictions = []
    for idx in range(len(test_data[0])):
        weighted_sum = b_weight
        for i in range(inputs):
            weighted_sum += weights[i] * test_data[i][idx]
        prediction = 1 if weighted_sum >= 0 else -1
        predictions.append(prediction)
    return predictions
```

Accuracy calculation function

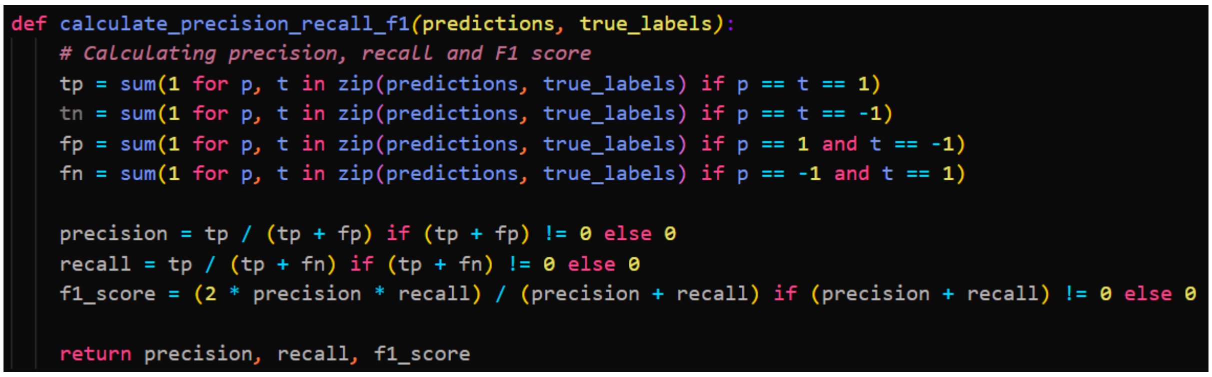
This function computes the accuracy of the model's predictions by comparing the predicted outputs with the actual expected outputs. It calculates the ratio of the number of correct predictions to the total number of predictions, giving a measure of how well the model performs with the test dataset.



```python
def calculate_accuracy(predictions, true_labels):
    # Calculating accuracy of the model
    correct_predictions = sum(p == t for p, t in zip(predictions, true_labels))
    accuracy = correct_predictions / len(true_labels)
    return accuracy
```

Precision, recall and F1 score calculation function

This function calculates key evaluation metrics for the model's performance. Precision measures the accuracy of positive predictions, recall the ability of the model to find all positive solutions, and the F1 score is the mean of precision and recall.

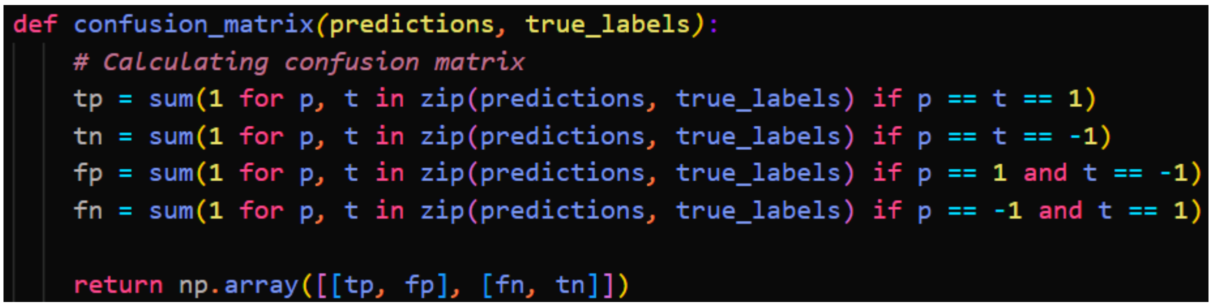


```python
def calculate_precision_recall_f1(predictions, true_labels):
    # Calculating precision, recall and F1 score
    tp = sum(1 for p, t in zip(predictions, true_labels) if p == t == 1)
    tn = sum(1 for p, t in zip(predictions, true_labels) if p == t == -1)
    fp = sum(1 for p, t in zip(predictions, true_labels) if p == 1 and t == -1)
    fn = sum(1 for p, t in zip(predictions, true_labels) if p == -1 and t == 1)

    precision = tp / (tp + fp) if (tp + fp) != 0 else 0
    recall = tp / (tp + fn) if (tp + fn) != 0 else 0
    f1_score = (2 * precision * recall) / (precision + recall) if (precision + recall) != 0 else 0

    return precision, recall, f1_score
```

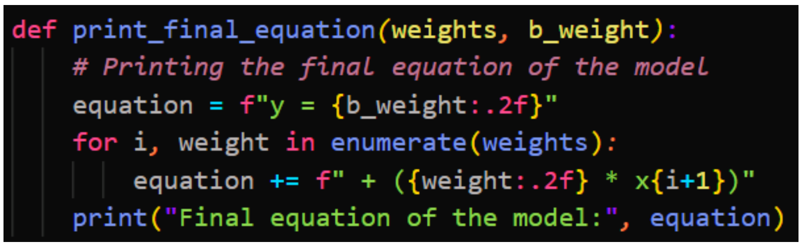
Confusion matrix printing function
This function generates and prints the confusion matrix counting the values for true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) comparing the model's predictions with the true labels.



```python
def confusion_matrix(predictions, true_labels):
    # Calculating confusion matrix
    tp = sum(1 for p, t in zip(predictions, true_labels) if p == t == 1)
    tn = sum(1 for p, t in zip(predictions, true_labels) if p == t == -1)
    fp = sum(1 for p, t in zip(predictions, true_labels) if p == 1 and t == -1)
    fn = sum(1 for p, t in zip(predictions, true_labels) if p == -1 and t == 1)

    return np.array([[tp, fp], [fn, tn]])
```

Equation printing function
This function prints the final equations that represent the decision weights learned by the model after training.

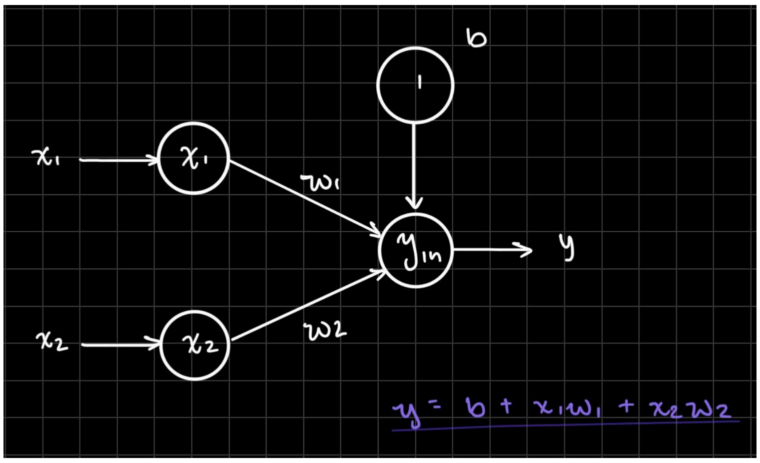


```python
def print_final_equation(weights, b_weight):
    # Printing the final equation of the model
    equation = f"y = {b_weight:.2f}"
    for i, weight in enumerate(weights):
        equation += f" + ({weight:.2f} * x{i+1})"
    print("Final equation of the model:", equation)
```

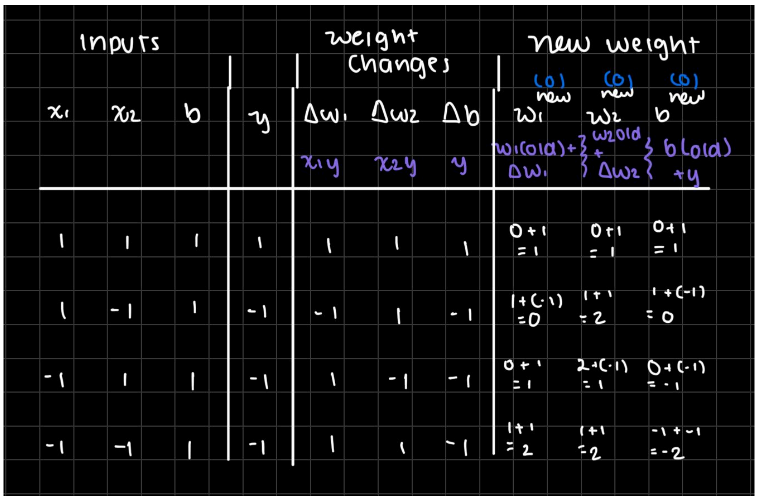
Main function
Last but not least, the main function provides the user with the option to use the system with either an AND logic gate or an OR logic gate.

Example

This is a manually calculated example of how the algorithm works. The first picture shows how the neural network started, the second one is the process followed to obtain the desired results and the third is how the neural network looks after calculating the weights. This example is using an AND logic gate.

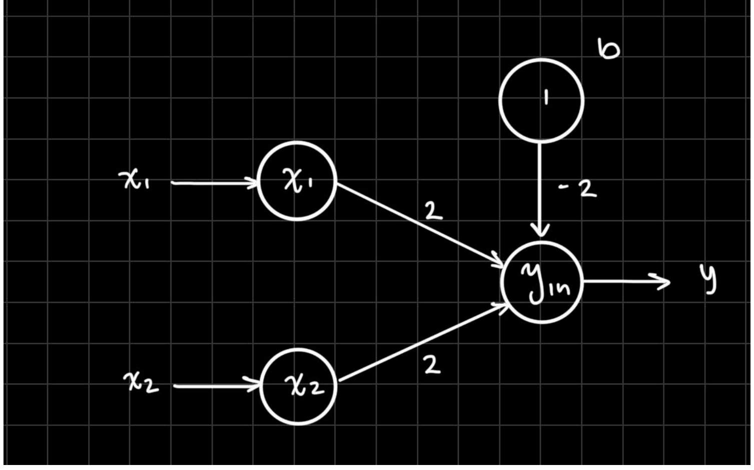


$y = b + x_1 w_1 + x_2 w_2$

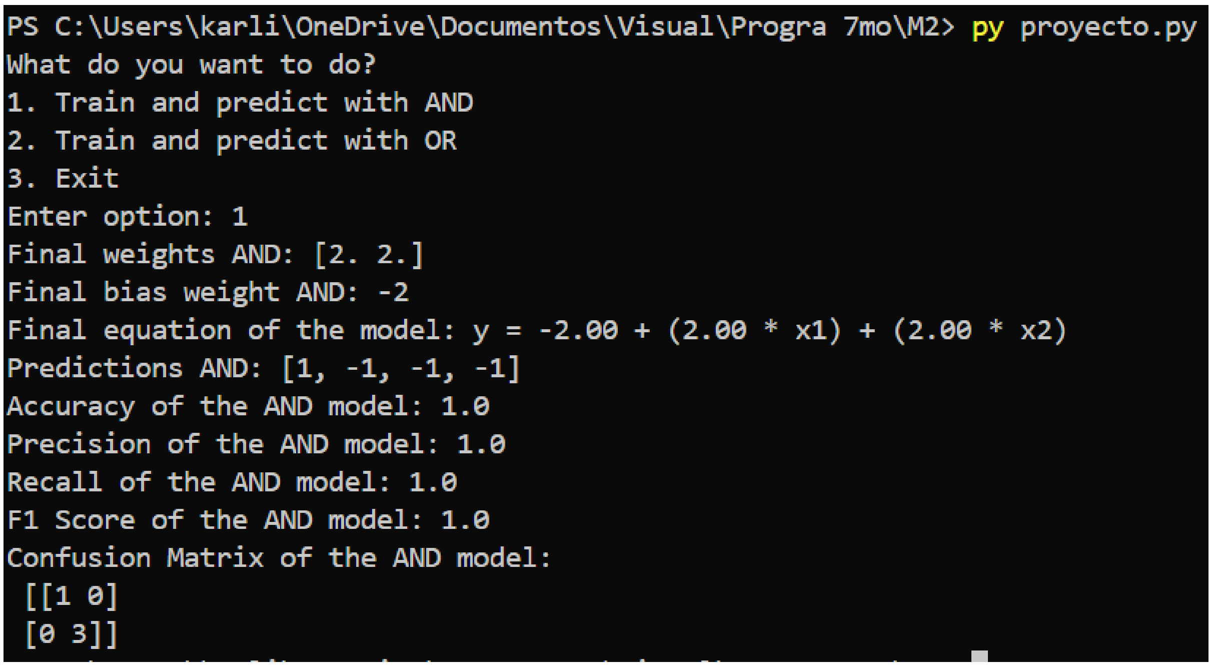


Inputs				Weight Changes			New weight		
x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 (0) new	w_2 (0) new	b (0) new
				$x_1 y$	$x_2 y$	y	$w_1(\text{old}) + \Delta w_1$	$w_2\text{old} + \Delta w_2$	$b(\text{old}) + y$
1	1	1	1	1	1	1	0+1 =1	0+1 =1	0+1 =1
1	-1	1	-1	-1	1	-1	1+(-1) =0	1+1 =2	1+(-1) =0
-1	1	1	-1	1	-1	-1	0+1 =1	2+(-1) =1	0+(-1) =-1
-1	-1	1	-1	1	1	-1	1+1 =2	1+1 =2	-1+-1 =-2

Final weights are: $w_1 = 2, w_2 = 2, b = -2$



Here are the results of the coded algorithm selecting the option of the AND logic gate.



```
PS C:\Users\karli\OneDrive\Documentos\Visual\Progra 7mo\M2> py proyecto.py
What do you want to do?
1. Train and predict with AND
2. Train and predict with OR
3. Exit
Enter option: 1
Final weights AND: [2. 2.]
Final bias weight AND: -2
Final equation of the model: y = -2.00 + (2.00 * x1) + (2.00 * x2)
Predictions AND: [1, -1, -1, -1]
Accuracy of the AND model: 1.0
Precision of the AND model: 1.0
Recall of the AND model: 1.0
F1 Score of the AND model: 1.0
Confusion Matrix of the AND model:
 [[1 0]
 [0 3]]
```

References
Jain, Sandeep. “Hebbian Learning Rule with Implementation of AND Gate.” GeeksforGeeks, 26 November 2020, https://www.geeksforgeeks.org/hebbian-learning-rule-with-implementation-of-and-gate/. Accessed 1 September 2024.