Experiment-1

Perceptron

1.Aim

Implementation and Simulation of Perceptron.

2.Theory

Introduction to Perceptrons

The perceptron is the simplest form of artificial neural network, invented by Frank Rosenblatt in 1958. It is a binary classifier that learns to separate data points into two classes using a linear decision boundary. A perceptron computes a weighted sum of its inputs, adds a bias term, and applies a step activation function to produce a binary output.
The basic structure and working of a single-layer perceptron are illustrated in *Fig. 1*.

"The perceptron is a type of linear classifier, i.e., a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector."
- Rosenblatt, 1958

A single-layer perceptron consists of:

1. Inputs (x_1, x_2, ..., x_n)

The features or data values that the perceptron receives to make decisions. These are like the raw information fed into the model. These input nodes and their connections to the perceptron are shown in *Fig. 1.*

- Example: For the XOR problem, we have two binary inputs x_1 and x_2, each can be either 0 or 1

2. Weights (w_1, w_2, ..., w_n)

These are Importance scores that determine how much each input contributes to the final prediction. A larger weight (positive or negative) means that input has greater influence. The weighted connections between inputs and the summation unit are as seen in *Fig. 1.*

- Each input gets its own weight, acting as a multiplier for that feature
- Weights control the **angle or tilt** of the decision boundary line

3. Bias (b)

An adjustable constant added to the weighted sum that shifts the decision boundary's position.

- Bias allows the line to move freely in space to better separate the classes
- Bias controls the **position or shift** of the decision boundary line

Key Difference Between Weights and Bias:

- **Weights** determine the slope, how steep the line is and its direction
- **Bias** determines the position, where the line sits in the coordinate space

Together they define a complete line, similar to **y = mx + c**, where weights act like slope 'm' and bias acts like y-intercept 'c'. The bias term $w_o(t) = \theta$ is as seen in *Fig. 1.*

4. Net Input (Weighted Sum)

The perceptron combines all inputs, weights, and bias into a single numerical value:

$$z = w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n + b$$

This can be thought of as calculating a "score", multiply each input by its weight, sum them all together, then add the bias. This score determines the predicted class. The summation operation (Σ) is as seen in *Fig. 1*.

5. Activation Function (Step Function)

The activation function converts the numerical score into a binary decision:

$$\hat{y} = 1 \;\; if \;\; z \geq 0$$
$$\hat{y} = 0 \;\; if \;\; z < 0$$

- If the score z is positive or zero → predict Class 1
- If the score z is negative → predict Class 0
- This creates a threshold at z = 0 where the perceptron switches between classes
- The resulting decision boundary is always a straight line (linear), which is why perceptrons can only solve linearly separable problems

The step activation function and output of the perceptron are as seen in *Fig. 1*.

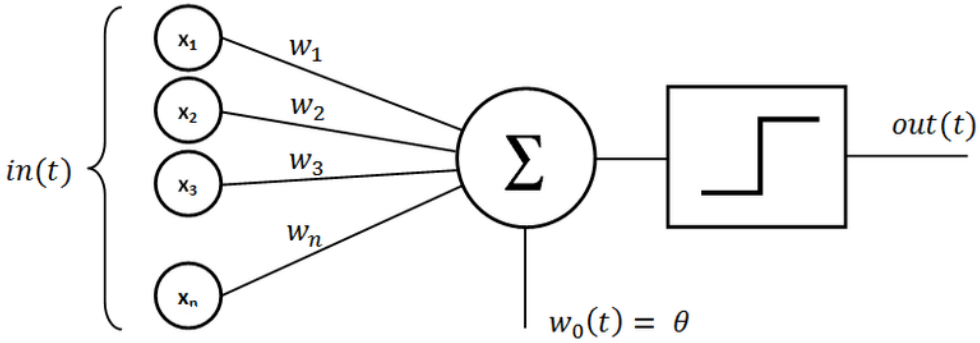

Fig. 1: Architecture of a Single-Layer Perceptron showing inputs, weights, summation unit, bias, and step activation function.

Perceptron Learning Algorithm

The perceptron learns through a simple trial-and-error process. It makes predictions, checks if they're correct, and adjusts its parameters when it makes mistakes. This is called **supervised learning** because we provide the correct answers during training.

The Learning Process:

1. Make a Prediction

The perceptron calculates its weighted sum and makes a prediction:
$\mathbf{z = w_1 \cdot x_1 + w_2 \cdot x_2 + ... + w_n \cdot x_n + b}$

$\mathbf{\hat{y} = 1 \text{ if } z \geq 0}$
$\mathbf{\hat{y} = 0 \text{ if } z < 0}$.

2. Check if the Prediction is Correct

Compare the predicted **output ($\hat{y}$)** with the actual **correct label (y**):
$\mathbf{error = y - \hat{y}}$

- If $error = 0$: The prediction was correct, no changes needed
- If $error = +1$: The perceptron predicted 0 but should have predicted 1 (underestimated)
- If $error = -1$: The perceptron predicted 1 but should have predicted 0 (overestimated)

This prediction process follows the perceptron structure as seen in *Fig. 1*.

3. Update the Weights (Learn from Mistakes)

When the perceptron makes an error, it adjusts each weight to reduce that error:

$$w_i = w_i + learning\ rate \times error \times x_i$$

where:

- **Learning rate**: Controls how big the adjustment steps are.
- **error**: Tells us the direction and magnitude to adjust
- $\mathbf{x_i}$: The input value that contributed to this weight

Example: If the perceptron predicted 0 but the answer was 1, the error is +1, so weights connected to active inputs ($x_i = 1$) increase, making the perceptron more likely to predict 1 next time.

4. Update the Bias (Shift the Boundary)

The bias is adjusted similarly, but without multiplying by any input:

$$b = b + learning\ rate \times error$$

This shifts the decision boundary to better separate the classes.

5. Repeat for All Training Examples

The perceptron goes through every data point in the training set, making predictions and adjusting weights. One complete pass through all data points is called an **epoch**.

6. Train for Multiple Epochs

The process repeats for several epochs. With each epoch:

- The decision boundary moves and rotates
- Classification accuracy typically improves
- The perceptron gets closer to the correct solution (if one exists)

7. Convergence or Stopping

Training stops when either:

- **Convergence**: All training examples are classified correctly (only possible for linearly separable data)
- **Maximum epochs reached**: We've trained for a fixed number of epochs

Important Note: For linearly separable problems like AND or OR gates, the perceptron will eventually find a perfect solution. However, for non-linearly separable problems like XOR, the perceptron will never converge and will continue making errors indefinitely, which is exactly what we observe in this experiment.

Linear Separability

A dataset is **linearly separable** if there exists a straight line (in 2D), plane (in 3D), or hyperplane (in higher dimensions) that can perfectly separate the two classes. Single-layer perceptrons can only learn linearly separable patterns.

Examples of linearly separable problems:

- AND gate: $(0,0) \rightarrow 0$, $(0,1) \rightarrow 0$, $(1,0) \rightarrow 0$, $(1,1) \rightarrow 1$
- OR gate: $(0,0) \rightarrow 0$, $(0,1) \rightarrow 1$, $(1,0) \rightarrow 1$, $(1,1) \rightarrow 1$

Examples of non-linearly separable problems:

- XOR gate

The XOR Problem

XOR (Exclusive OR) is a Boolean logic function that outputs 1 when the inputs are different and 0 when they are the same. The XOR truth table is:

x1	x2	Output
0	0	0
0	1	1

1	0	1
1	1	0

When plotted in 2D space, the XOR problem shows a diagonal pattern where:

- Points (0,0) and (1,1) belong to Class 0 (blue)
- Points (0,1) and (1,0) belong to Class 1 (red)

No matter how we adjust w1, w2, and b, we cannot draw a single straight line that achieves this separation. The perceptron will keep updating its weights indefinitely, oscillating between different incorrect solutions, never achieving 100% accuracy.

The solution came with the development of **multi-layer perceptrons (MLPs)** with hidden layers and non-linear activation functions. A two-layer neural network with at least 2 hidden neurons can solve XOR by creating multiple decision boundaries that, when combined, separate the classes correctly.

Merits of Perceptrons

- **Simplicity**: Easy to understand and implement
- **Computational Efficiency**: Fast training and prediction for linearly separable problems
- **Foundation**: Forms the basis for understanding more complex neural networks
- **Interpretability**: Decision boundary can be easily visualised and understood

Demerits of Perceptrons

- **Linear Limitation**: Cannot solve non-linearly separable problems like XOR
- **Binary Classification Only**: Limited to two-class problems in basic form
- **Sensitive to Feature Scaling**: Performance can be affected by input feature scales
- **No Convergence for Non-separable Data**: May oscillate indefinitely without reaching a solution

3. Pre-Test (MCQs)

1. What type of decision boundary does a single-layer perceptron create?

a) A circular boundary (Incorrect because circles are non-linear boundaries requiring complex functions.)

b) A linear hyperplane (Correct because perceptrons create linear decision boundaries defined by $w \cdot x + b = 0$.)

c) A polynomial curve (Incorrect because polynomial curves are non-linear and require multiple layers.)

d) Multiple disconnected regions (Incorrect because a single perceptron creates one continuous linear boundary.)

Answer: b

2. What does it mean for a dataset to be linearly separable?

a) The data can be separated by a straight line or hyperplane (Correct because linear separability means a linear boundary can perfectly divide the classes.)

b) The data points are arranged in a line (Incorrect because linear separability refers to class separation, not point arrangement.)

c) The dataset has only linear features (Incorrect because feature types don't determine separability.)

d) All data points have the same label (Incorrect because this would mean no classification problem exists.)

Answer: a

3. What is the XOR problem in the context of perceptrons?

a) A linearly separable problem that perceptrons can easily solve (Incorrect because XOR is the classic example of a non-linearly separable problem.)

b) A non-linearly separable problem that single-layer perceptrons cannot solve (Correct because XOR requires non-linear decision boundaries that single perceptrons cannot create.)

c) A problem with too much training data (Incorrect because XOR has only 4 data points, not an excess of data.)

d) A convergence issue due to wrong learning rate (Incorrect because no learning rate can make a perceptron solve XOR.)

Answer: b

4. In the perceptron learning rule, when are weights updated?

a) Only when the prediction is correct (Incorrect because weights don't change when predictions are already correct.)

b) After every epoch regardless of errors (Incorrect because updates happen based on individual prediction errors.)

c) Only when the perceptron makes an incorrect prediction (Correct because the error term $(y - \hat{y})$ is zero for correct predictions, causing no weight change.)

d) Before making any predictions (Incorrect because weights are updated after predictions and error calculation.)

Answer: c

5. What happens to the weights when the perceptron makes a correct prediction?

a) They are reset to zero (Incorrect because resetting would erase all learned information.)

b) They remain unchanged (Correct because the error term becomes zero $(y - \hat{y} = 0)$, resulting in no weight updates.)

c) They increase by the learning rate (Incorrect because weight changes depend on the error, not just learning rate.)

d) They are randomly reinitialised (Incorrect because random changes would disrupt the learning process.)

Answer: b

6. What role does the learning rate play in perceptron training?

a) It determines how many epochs to train (Incorrect because the number of epochs is set separately from learning rate.)

b) It controls the step size of weight updates (Correct because learning rate scales the magnitude of weight adjustments based on errors.)

c) It sets the initial weights (Incorrect because initial weights are typically set randomly or to zero.)

d) It defines the activation function (Incorrect because the activation function is a separate component.)

Answer: b

7. What does the bias term 'b' allow the perceptron to do?

a) Increase the number of input features (Incorrect because bias doesn't add features, it adjusts the decision boundary.)

b) Move the decision boundary away from the origin (Correct because bias shifts the line's position without needing all inputs to be zero.)

c) Make the decision boundary curved (Incorrect because perceptron boundaries are always linear regardless of bias.)

d) Classify multiple classes simultaneously (Incorrect because single perceptrons are binary classifiers.)

Answer: b

8. Which logic gate CAN be solved by a single-layer perceptron?

a) XOR gate (Incorrect because XOR is non-linearly separable.)

b) XNOR gate (Incorrect because XNOR is also non-linearly separable, being the inverse of XOR.)

c) AND gate (Correct because AND is linearly separable and can be solved by a single perceptron.)

d) None of the above (Incorrect because AND and OR gates are linearly separable.)

Answer: c

9. What mathematical equation defines the decision boundary of a perceptron in 2D?

a) $w1 \cdot x1 + w2 \cdot x2 + b = 0$ (Correct because this linear equation defines the line separating the two classes.)

b) $w1 \cdot x1^2 + w2 \cdot x2^2 + b = 0$ (Incorrect because this represents a quadratic curve, not achievable with single-layer perceptrons.)

c) x1 + x2 = 1 (Incorrect because the weights and bias are learnable parameters, not fixed to these values.)

d) sigmoid(w1·x1 + w2·x2 + b) = 0.5 (Incorrect because standard perceptrons use step functions, not sigmoid.)

Answer: a

10. What fundamental limitation did the XOR problem reveal about single-layer perceptrons?

a) They require too much training data (Incorrect because XOR uses only 4 data points; data amount isn't the issue.)

b) They can only create linear decision boundaries (Correct because the inability to solve XOR demonstrated that perceptrons cannot learn non-linear patterns.)

c) They cannot handle binary inputs (Incorrect because perceptrons work well with binary inputs for linearly separable problems.)

d) They are too slow to train (Incorrect because perceptrons train quickly; the issue is their representational capacity.)

Answer: b

4. Procedure

Step 1: Import Required Libraries

Import necessary Python libraries including pandas for data handling, NumPy for numerical computations, and Matplotlib for visualisation of the dataset and decision boundaries**.**

Step 2: Create XOR Dataset

Generate the XOR truth table with four data points:

- $(0, 0) \rightarrow Class\ 0$
- $(0, 1) \rightarrow Class\ 1$
- $(1, 0) \rightarrow Class\ 1$
- $(1, 1) \rightarrow Class\ 0$

Store this data in a pandas DataFrame for easy manipulation and visualisation.

Step 3: Visualise the XOR Dataset

Create a scatter plot showing the four XOR data points with:

- Blue dots representing Class 0: points (0,0) and (1,1)
- Red dots representing Class 1: points (0,1) and (1,0)

This visualisation clearly shows the diagonal pattern that makes XOR non-linearly separable.

Step 4: Initialise Perceptron Parameters

Set random seed for reproducibility and initialise:

- **w1, w2**: Random weights uniformly distributed between -1 and 1
- **b**: Random bias uniformly distributed between -1 and 1
- **learning_rate**: Set to 0.05 (or 0.1 in alternate version)

Print initial parameters to track how they change during training.

Step 5: Define Decision Boundary Plotting Function

Create a function `plot_decision_boundary(w1, w2, b)` that:

- Calculates the decision boundary line using equation: w1·x1 + w2·x2 + b = 0
- Rearranges to: x2 = -(w1·x1 + b) / w2
- Plots this line in green on the XOR scatter plot
- Handles the case when w2 = 0 (vertical line)

Step 6: Visualise Initial Random Decision Boundary

Plot the XOR dataset along with the initial random decision boundary to show the starting position before any training occurs. This demonstrates how randomly initialised weights create an arbitrary separation.

Step 7: Define Perceptron Training Function

Implement `perceptron_train()` function that:

For each epoch:

1. Initialise correct prediction counter
2. For each data point:
    i. Calculate linear output: **linear_output =** $w_1 \cdot x_1 + w_2 \cdot x_2 + b$
    ii. Apply step activation: $y_{pred} = 1$ if linear_output ≥ 0 else 0
    iii. Count if prediction matches actual label
    iv. Calculate error: $error = y - y_{pred}$
    v. Update weights: $w1 = w_1 + lr \times error \times x_1$
    vi. Update weights: $w2 = w_2 + lr \times error \times x_2$
    vii. Update bias: $b = b + lr \times error$
3. Calculate accuracy: $accuracy = \frac{correct}{total} \times 100$
4. Print epoch results showing accuracy and updated parameters
5. Visualise decision boundary after each epoch

Return final trained weights and bias.

Step 8: Run Perceptron Training

Execute the training function for 10 epochs on the XOR dataset.

Step 9: Visualise Training Progress

After each epoch, generate and display a plot showing:

- The current decision boundary (green line)
- XOR data points (blue and red dots)
- Epoch number in the title

- Current accuracy percentage

This creates a visual sequence showing how the perceptron tries but fails to find a correct solution.

Step 10: Make Final Predictions

Define `predict()` function and use it to make predictions on all four XOR points using the final trained weights. Display results showing:

- Input coordinates
- Predicted class
- Actual class
- Whether prediction is correct or incorrect

Calculate and display final accuracy on the complete XOR dataset.

5. Post-Test (MCQs)

1. When does a perceptron update its weights during training?

a) After every epoch, regardless of accuracy (Incorrect because weight updates happen during the epoch, not after it.)

b) Only when it makes an incorrect prediction (Correct because weights change only when error ≠ 0, which occurs during misclassification.)

c) Before making any predictions (Incorrect because predictions must be made first to calculate errors.)

d) Only in the first epoch (Incorrect because updates continue throughout all epochs when errors occur.)

Answer: b

2. What is the first step a perceptron takes when making a prediction?

a) Apply the activation function (Incorrect because the weighted sum must be calculated first.)

b) Calculate the weighted sum of inputs plus bias (Correct because $z = w_1 \cdot x_1 + w_2 \cdot x_2 + b$ is computed before applying the step function.)

c) Update the weights (Incorrect because weights are updated after prediction and error calculation.)

d) Generate random outputs (Incorrect because perceptrons make deterministic predictions based on learned parameters.)

Answer: b

3. What output does the perceptron's step function produce when z = 0?

a) 0.5 (Incorrect because the step function produces binary outputs, not probabilities.)

b) 1 (Correct because the standard convention is: output = 1 if $z \geq 0$.)

c) -1 (Incorrect because perceptrons output 0 or 1, not negative values.)

d) Undefined (Incorrect because $z = 0$ is a defined case where the output is 1.)

Answer: b

4. If you observe the decision boundary after each epoch, what would you notice?

a) It stays in the same position throughout training (Incorrect because weight updates cause the boundary to move.)

b) It rotates and shifts position trying to separate the classes (Correct because changing weights alter the slope and position of the linear boundary.)

c) It splits into multiple lines (Incorrect because a single perceptron creates only one decision boundary.)

d) It becomes a curve (Incorrect because perceptron boundaries are always linear.)

Answer: b

5. Which two XOR points are typically on opposite sides of the final decision boundary?

a) (0,0) and (0,1) (Incorrect because these are adjacent points, not diagonally opposite.)

b) (1,0) and (1,1) (Incorrect because these are also adjacent points.)

c) (0,0) and (1,1), or (0,1) and (1,0) (Correct because the linear boundary separates diagonally opposite points, though not correctly for XOR.)

d) All four points are on the same side (Incorrect because the boundary separates the space into two regions.)

Answer: c

6. What does the error value represent in the perceptron learning rule?

a) The learning rate (Incorrect because learning rate is a separate hyperparameter.)

b) The difference between actual and predicted labels (Correct because error = y_actual - y_predicted.)

c) The total number of training epochs (Incorrect because error is calculated per prediction, not related to epoch count.)

d) The weight value (Incorrect because error is used to update weights, not the weight itself.)

Answer: b

7. When does the perceptron learning algorithm update the bias term?

a) Only at the end of each epoch (Incorrect because bias updates happen for each misclassified point.)

b) When there is a prediction error, using: b = b + learning_rate × error (Correct because bias is updated the same way as weights, but without multiplying by input.)

c) The bias never changes during training (Incorrect because bias is a learnable parameter that gets updated.)

d) Only when accuracy drops below 50% (Incorrect because updates depend on individual prediction errors, not overall accuracy.)

Answer: b

8. What is the fundamental lesson learned from the XOR experiment?

a) Perceptrons are useless for all problems (Incorrect because perceptrons work well for linearly separable problems.)

b) More training epochs will eventually solve any problem (Incorrect because XOR cannot be solved regardless of training duration.)

c) Single layer perceptrons have a fundamental limitation in solving non-linearly separable problems (Correct because this is the key insight from the XOR problem and its historical significance.)

d) Learning rate is the most important hyperparameter (Incorrect because the limitation is architectural, not about hyperparameter tuning.)

Answer: c

9. Which components of a perceptron are adjusted during the training process?

a) Only the weights (Incorrect because bias is also updated during training.)

b) Both weights and bias (Correct because the learning algorithm updates both w and b based on errors.)

c) Only the input values (Incorrect because inputs are fixed data; they are not learned parameters.)

d) Only the activation function (Incorrect because the activation function remains constant during training.)

Answer: b

10. **What is an "epoch" in the context of perceptron training?**

a) A single prediction made by the perceptron (Incorrect because an epoch involves processing all training examples.)

b) One complete pass through the entire training dataset (Correct because an epoch means the perceptron has seen and learned from every training sample once.)

c) The final accuracy of the model (Incorrect because accuracy is a metric, not a training iteration.)

d) The time taken to train the model (Incorrect because epoch refers to iterations, not time duration.)

Answer: b

6. References

1. F. Rosenblatt, "The perceptron: A probabilistic model for information storage and Organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386-408, 1958.
2. M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, 1969.
3. Diagram - Wikipedia Commons (https://commons.wikimedia.org/wiki/File:Perceptron_moj.png)