⍰ Perceptron Learning - Complete Notes

□ What is a Perceptron?

A **Perceptron** is one of the simplest types of **artificial neural networks**, mainly used for **binary classification**. It is considered the **building block of neural networks**.

A perceptron takes multiple inputs, applies corresponding weights, adds a bias, and passes the weighted sum through an **activation function** (usually a step function) to produce an output, typically **0 or 1**.

□ Mathematical Representation:

z=(w1x1+w2x2+...+wnxn)+bz = (w_1x_1 + w_2x_2 + ... + w_nx_n) + b Output={1if z>00otherwise\text{Output} = \begin{cases} 1 & \text{if} \ z > 0 \\ 0 & \text{otherwise} \end{cases}

□ Perceptron Trick (How It Learns)

The **Perceptron Trick** is an intuitive approach for adjusting the decision boundary.
Idea: If the perceptron misclassifies a point, adjust the weights and bias to push the decision boundary closer to correctly classifying that point.

□ Weight Update Rule:

wnew=wold+η·(ytrue−ypred)·xw_{\text{new}} = w_{\text{old}} + \eta \cdot (y_{\text{true}} - y_{\text{pred}}) \cdot x
bnew=bold+η·(ytrue−ypred)b_{\text{new}} = b_{\text{old}} + \eta \cdot (y_{\text{true}} - y_{\text{pred}})

Where:

- **w** = weights
- **b** = bias
- **x** = input feature vector
- **η (eta)** = learning rate (small positive value)
- **y_true** = true label (0 or 1)
- **y_pred** = predicted label (0 or 1)

□ Perceptron Training Algorithm (Step-by-Step)

□ Step 1: Initialize

- Set weights and bias to small random numbers or zeros.
- Choose a small learning rate η (example: 0.01).

□ Step 2: For Each Training Example

1. **Calculate Weighted Sum**

z=w·x+bz = w \cdot x + b

2. **Apply Activation Function (Step Function)**

Output=1 if z>0, else 0\text{Output} = 1 \text{ if } z > 0, \text{ else } 0

3. **Update Weights and Bias (if misclassified)**

w=w+η·(ytrue−ypred)·xw = w + \eta \cdot (y_{\text{true}} - y_{\text{pred}}) \cdot x b=b+η·(ytrue−ypred)b = b + \eta \cdot (y_{\text{true}} - y_{\text{pred}})

□ Step 3: Repeat

- Repeat this process for all data points multiple times (epochs) until:
  - Either convergence (no change in weights)
  - Or maximum epochs reached

□ OR Gate Problem Using Perceptron

□ OR Gate Truth Table

x1	x2	Output (OR)
0	0	0
0	1	1
1	0	1
1	1	1

□ How Transformation Happens? (Geometric Intuition)

1□ Inputs as Points on 2D Plane:

- (0, 0) → Output 0
- (0, 1), (1, 0), (1, 1) → Output 1

2□ Separating Boundary (Line Equation):

The perceptron tries to find a linear equation:

w1x1+w2x2+b=0w_1x_1 + w_2x_2 + b = 0

This line divides the plane into **two regions**:

- Above the line → Output 1
- Below the line → Output 0

3□ Perceptron Adjustment:

- Each misclassified point nudges the weights.
- Gradually, the line shifts to perfectly separate OR outputs.

□ Example of Final Weights for OR

Suppose after training:

w1=1,w2=1,b=−0.5w_1 = 1, \quad w_2 = 1, \quad b = -0.5

Final equation becomes:

x1+x2−0.5>0 ⟹ Output=1x_1 + x_2 - 0.5 > 0 \implies \text{Output} = 1 x1+x2−0.5≤0 ⟹ Output=0x_1 + x_2 - 0.5 \leq 0 \implies \text{Output} = 0

Visualization (OR Gate)

```
 x2
  |
1 |     (1,1)  1
  |  (0,1)  1
  |--------- x1
  |  (0,0)  0    (1,0)  1
```

A simple line (e.g., x1 + x2 = 0.5) easily separates 0 from 1.

□ Key Points (Summary)

✔ Perceptron is a **linear classifier**.
✔ **Perceptron Trick** modifies weights towards correct classification.
✔ **OR Gate** is linearly separable, hence perceptron solves it easily.
✔ Perceptron fails on problems like XOR (non-linear).

□ Advantages of Perceptron:

- Simple & easy to implement.
- Works well for **linearly separable** problems.

□ Limitations of Perceptron:

- Cannot solve **non-linear** problems (e.g., XOR).
- Only works for **binary classification**.

If this looks good, I'll create a **professional Word file** with:
□ Headings
□ Bold highlights
□ Diagrams
□ Clean formatting

Let me know:
□ **Ready for Word file?** Or any change required?