PART I : Perceptron Convergence Theorem Proof

Bounding the Increase in $\|\mathbf{w}\|$:

- When the Perceptron makes an update, we have:

$$\|\mathbf{w}_{t+1}\|^2 = \|\mathbf{w}_t + y_i\mathbf{x}_i\|^2 = \|\mathbf{w}_t\|^2 + 2y_i(\mathbf{w}_t \cdot \mathbf{x}_i) + \|y_i\mathbf{x}_i\|^2$$

- Since $\|y_i\mathbf{x}_i\|^2 = \|\mathbf{x}_i\|^2 \leq M^2$, we get:

$$\|\mathbf{w}_{t+1}\|^2 \leq \|\mathbf{w}_t\|^2 + R^2$$

Increase in Alignment with $\mathbf{w}^*$:

- The dot product with the target vector $\mathbf{w}^*$ increases by at least γ:

$$\mathbf{w}_{t+1} \cdot \mathbf{w}^* = (\mathbf{w}_t + y_i\mathbf{x}_i) \cdot \mathbf{w}^* = \mathbf{w}_t \cdot \mathbf{w}^* + y_i(\mathbf{x}_i \cdot \mathbf{w}^*) \geq \mathbf{w}_t \cdot \mathbf{w}^* + \gamma$$

Bounding the Number of Updates:

- Let N be the number of updates made. Then:

$$\|\mathbf{w}_N\|^2 \leq NM^2$$

- And:

$$\mathbf{w}_N \cdot \mathbf{w}^* \geq N\gamma$$

- Using the Cauchy-Schwarz inequality:

$$\mathbf{w}_N \cdot \mathbf{w}^* \leq \|\mathbf{w}_N\|\|\mathbf{w}^*\|$$

- Thus:

$$N\gamma \leq \|\mathbf{w}_N\|\|\mathbf{w}^*\|$$

- Substitute $\|\mathbf{w}_N\| \leq \sqrt{N}R$:

$$N\gamma \leq \sqrt{N}M\|\mathbf{w}^*\|$$

Solving for N:

- Dividing both sides by $\sqrt{N}$:

$$\sqrt{N}\gamma \leq M\|\mathbf{w}^*\|$$

- Squaring both sides:

$$N \leq \left(\frac{M\|\mathbf{w}^*\|}{\gamma}\right)^2$$

Conclusion:

- The number of updates N is finite and bounded by $\left(\frac{M\|\mathbf{w}^*\|}{\gamma}\right)^2$.
- Thus, the Perceptron algorithm will converge in a finite number of steps for linearly separable data.

PART II : Kernels

(1) To prove that $K(x, x') = (1 + x^T x')^2$ is a valid kernel:
Expand $K(x, x')$:

$$K(x, x') = (1 + x^T x')^2 = 1 + 2(x^T x') + (x^T x')^2.$$

Rewrite as a sum of terms: This expression can be rewritten as:

$$K(x, x') = 1 + 2\sum_{i=1}^{n} x_i x_i' + \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j x_i' x_j',$$

where $x = [x_1, x_2, \ldots, x_n]^T$ and similarly $x'[x_1', x_2', \ldots, x_n']^T$.
Finally, Construct $\phi(x)$ to include all terms that appear in the expansion:

$$\phi(x) = \left[1, \sqrt{2}x_1, \sqrt{2}x_2, \ldots, \sqrt{2}x_n, x_1^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_1x_n, x_2^2, \ldots, x_n^2\right]^T.$$

(2) Similarly, for $K(x, x') = (x^T x')^2 + 3(x^T x') + 2$ the representation for $\phi(x)$ is

$$\phi(x) = \left[\sqrt{2}, \sqrt{3}x_1, \sqrt{3}x_2, \ldots, \sqrt{3}x_n, x_1^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_1x_n, x_2^2, \ldots, x_n^2\right]^T.$$

Note: We will consider if you prove this for some specific choice of n other than $n = 1$.

PART III : Neural Networks

For, $f(x) = w_2(\sigma(w_1 x + b_1)) + b_2$, where $x = [x_1, x_2] \in \mathbb{R}^2$, $w_1 \in \mathbb{R}^{2\times 2}$, $w_2 \in \mathbb{R}^{1\times 2}$

(1) Implementing AND gate. (One possible configuration, infinitely many exist).
Parameters:

$$w_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad b_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix},$$
$$w_2 = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}, \quad b_2 = 0.$$

(2) Implementing XOR gate.
Parameters:

$$w_1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \quad b_1 = \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix},$$
$$w_2 = \begin{bmatrix} 2 & 2 \end{bmatrix}, \quad b_2 = 0.$$

(3) Implementing OR gate.
Parameters:

$$w_1 = \begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix},$$
$$w_2 = \begin{bmatrix} -1 & -1 \end{bmatrix}, \quad b_2 = 1.$$

(4) Implementing XNOR gate.
Parameters:

$$w_1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \quad b_1 = \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix},$$
$$w_2 = \begin{bmatrix} -2 & -2 \end{bmatrix}, \quad b_2 = 1.$$

PART IV : Two Sigmoids

Modelling Probabilities

The output probabilities of the first two classes, p_1 and p_2, must be constrained such that their sum does not exceed 1, i.e.:

$$p_1 + p_2 \leq 1$$

Constraints on Probabilities

The probabilities p_1, p_2, and p_3 must satisfy the following conditions:

$$p_1 + p_2 + p_3 = 1$$

Since $p_3 = 1 - p_1 - p_2$, the valid range for p_1 and p_2 is:

$$0 \leq p_1, p_2 \leq 1$$

and additionally, the sum constraint:

$$p_1 + p_2 \leq 1$$

This ensures that the sum of p_1 and p_2 does not exceed 1, thereby guaranteeing a valid probability distribution for all three classes.

Loss Function Formulation

The model uses binary cross-entropy (BCE) loss to compute the error for the first two classes. For each class, the BCE loss is calculated as:

$$L_1 = -\left[y_1 \log(p_1) + (1 - y_1) \log(1 - p_1)\right]$$

$$L_2 = -\left[y_2 \log(p_2) + (1 - y_2) \log(1 - p_2)\right]$$

where: - y_1 and y_2 are the ground truth labels for class C_1 and class C_2, respectively.
The total loss is a combination of the binary cross-entropy losses, the penalty for violating the constraint $p_1 + p_2 \leq 1$, and an optional regularization term. The penalty term ensures that $p_1 + p_2 \leq 1$, which can be written as:

$$\text{Penalty} = \max(0, p_1 + p_2 - 1)$$

This penalty term is added to the loss function to discourage the model from predicting invalid probabilities.

Total Loss Function

The total loss function combines the BCE loss terms, the penalty term, and an L2 regularization term to prevent overfitting (optional):

$$L_{\text{total}} = L_1 + L_2 + \lambda \cdot \max(0, p_1 + p_2 - 1) + \gamma \cdot \|w\|_2^2$$

where: - λ is a hyperparameter that controls the strength of the penalty term. - γ is a hyperparameter that controls the strength of the L2 regularization term. - w represents the model parameters (weights).
The L2 regularization term is given by:

$$\|w\|_2^2 = \sum_i w_i^2$$

which sums the squares of all model parameters.

PART V : Coding and PyTorch

The three errors in the code are as follows:

1. Input dimension in *line 46* should be **20**, not **32** as the input $x \in \mathbb{R}^{20}$ mentioned in *line 39*.
2. Train dataset is used for early stopping instead of the val dataset.
3. Learning rate is too high.