Non-linearly separable problems,

CT-466 | Week 5 - Lecture 10

Instructor: Mehar Fatima Shaikh

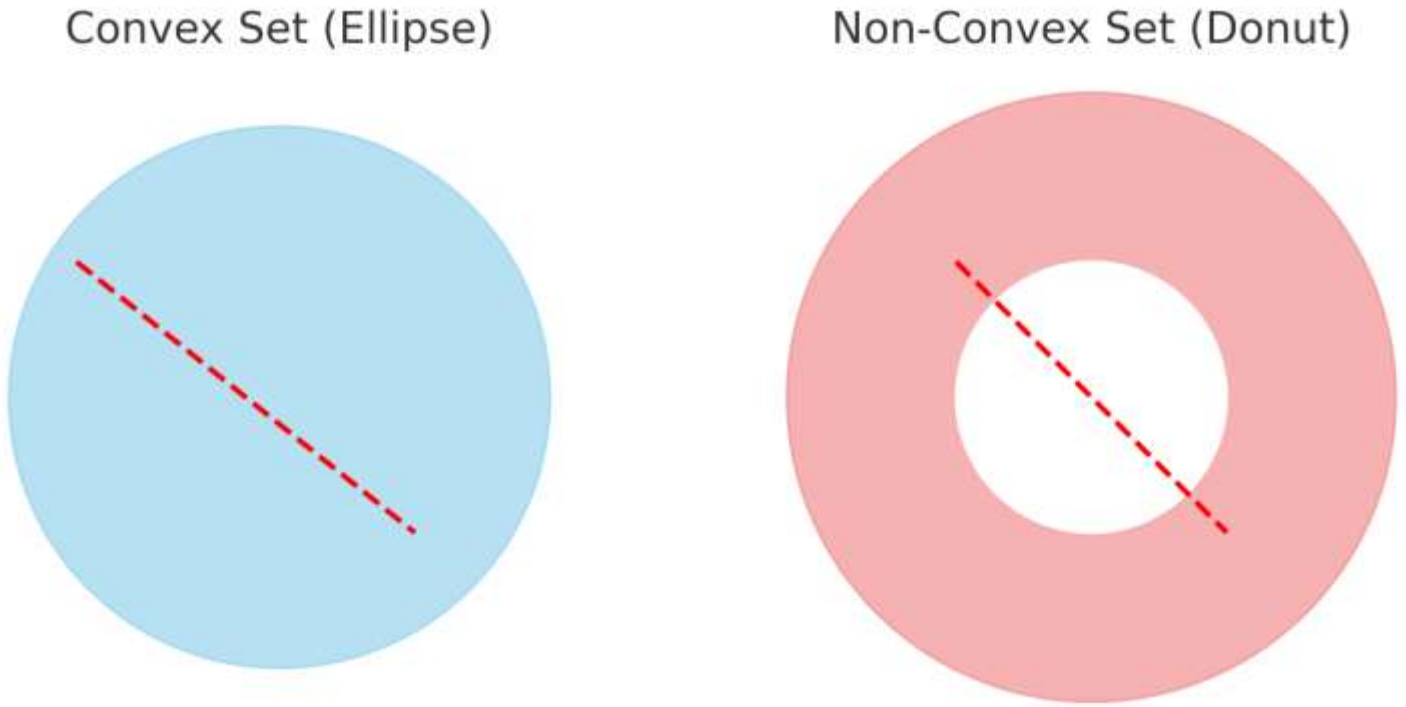
Convex Set

- A set S in a vector space is called convex if, for any two points $x1, x2 \in S$, the line segment joining them also lies entirely within the set.

- Mathematically:
  $\lambda x1 + (1 - \lambda)x2 \in S$ for all $x1, x2 \in S$ and $\lambda \in [0,1]$.

  This means the set has no 'dents' or 'holes'; it's 'bulged outwards.'

Convex Set



Convex Sets in ANN Context

In neural networks, convexity appears in several places:

1. Feasible Region of Parameters (Weights/Biases)
   - During training, weights and biases form a parameter space.
   - If we impose linear constraints (e.g., non-negative weights), the feasible solutions form a convex set.
   - Example: If $w1, w2 \geq 0$, all valid weight pairs lie in the first quadrant, which is convex.
2. Loss Functions and Optimization
   - For simple models like linear regression, the Mean Squared Error (MSE) loss is convex.
   - Convexity of the loss function means no local minima → gradient descent will always find the global minimum.
   - But in deep neural networks, due to multiple layers and nonlinear activation functions, the loss surface is generally non-convex (many local minima, saddle points).
3. Convex Sets in Regularization
   - In L1 (Lasso) and L2 (Ridge) regularization, the constraint regions are convex:
     - L1 → diamond-shaped convex set
     - L2 → circular/elliptical convex set
   - Convexity ensures optimization is tractable.

Why Convexity Matters in ANN

- Optimization simplicity: Convex sets + convex loss → guarantees convergence to a unique solution.
- Deep networks: Loss surface is non-convex, but still contains large convex regions where gradient descent works well.
- Regularization: Convex constraint sets ensure stability and good generalization.

Convex Hull

- The **convex hull** of a set of points is the *smallest convex set* that contains all those points.
- Geometrically, imagine stretching a rubber band around the outermost points in a scatter plot → when released, it forms the convex hull.
- Mathematically:
  - $Convex\ Hull(X) = \{ \sum_{i=1}^{n} \lambda_i\, x_i \mid x_i \in X, \lambda_i \geq 0, \sum \lambda_i = 1 \}$

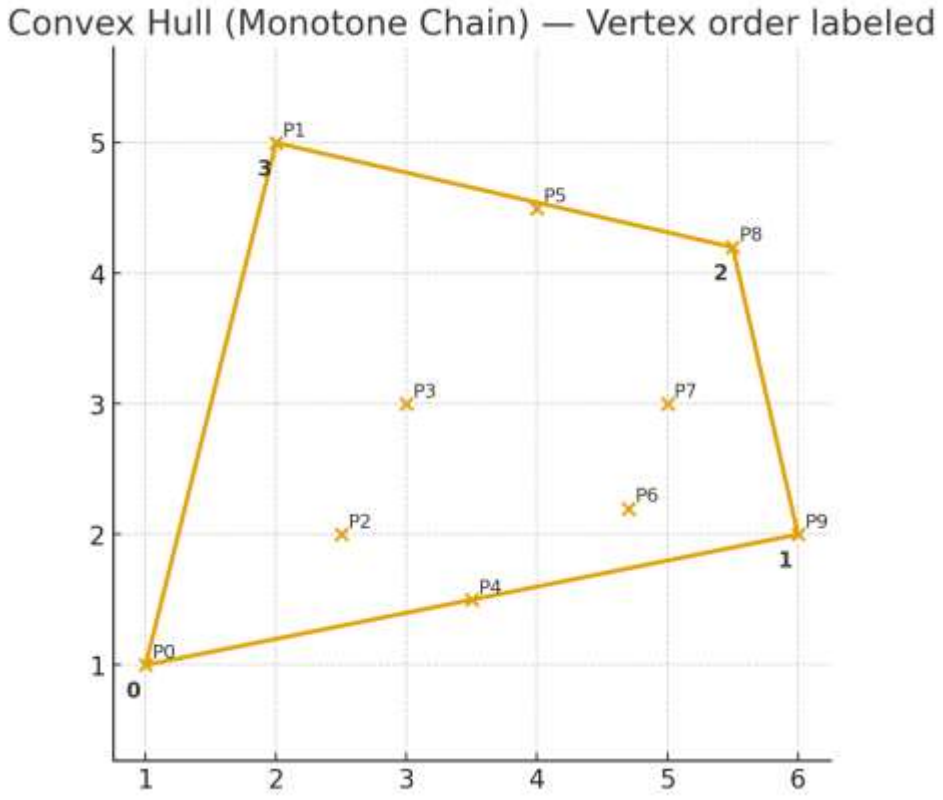
1) Pencil-and-paper (Gift Wrapping / Jarvis March)

- Goal: Walk around the “outer fence” of the points.
- 1. Plot points on graph paper (or a coordinate plane).
- 2. Start point: Pick the left-most point (if tie, pick the lowest y). Call it p0.
- 3. Pick direction: Imagine a horizontal ray to the left from p0 (or any fixed reference direction).
- 4. Find next vertex: For every other point q, compute the orientation of triangle (current, candidate, q).
  Use cross product sign:
  cross(a, b, c) = (b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x)
  Positive → left turn (more counter-clockwise). Choose the most counter-clockwise point as the next hull vertex.
- 5. Draw edge to that point and set it as current.
- 6. Repeat step 4 until you return to the starting point p0.
- 7. The edges you traced form the convex hull.
- Tip: If two points are collinear in the chosen direction, keep the farther one for the hull.

2) Algorithmic (Andrew's Monotone Chain)

- **Idea:** Sort points, then build **lower** and **upper** hulls using a left-turn test.
- **Steps**
- **Sort** points by x, then y.
- **Lower hull:**
  - Scan left→right.
  - While the last two points in the hull plus the new point make a **non-left turn** (cross ≤ 0), **pop** the last point.
  - Append the new point.
- **Upper hull:**
  - Scan right→left; do the same procedure.
- **Concatenate** lower[:-1] + upper[:-1] to avoid duplicating endpoints.
- The result is the convex hull in order (counter-clockwise).
- **Orientation test (same as above):**
- $\text{cross}(o, a, b) = (a_x - o_x)(b_y - o_y) - (a_y - o_y)(b_x - o_x)$
- Keep points that make **left turns** (cross > 0).

Andrew’s Monotone Chain)



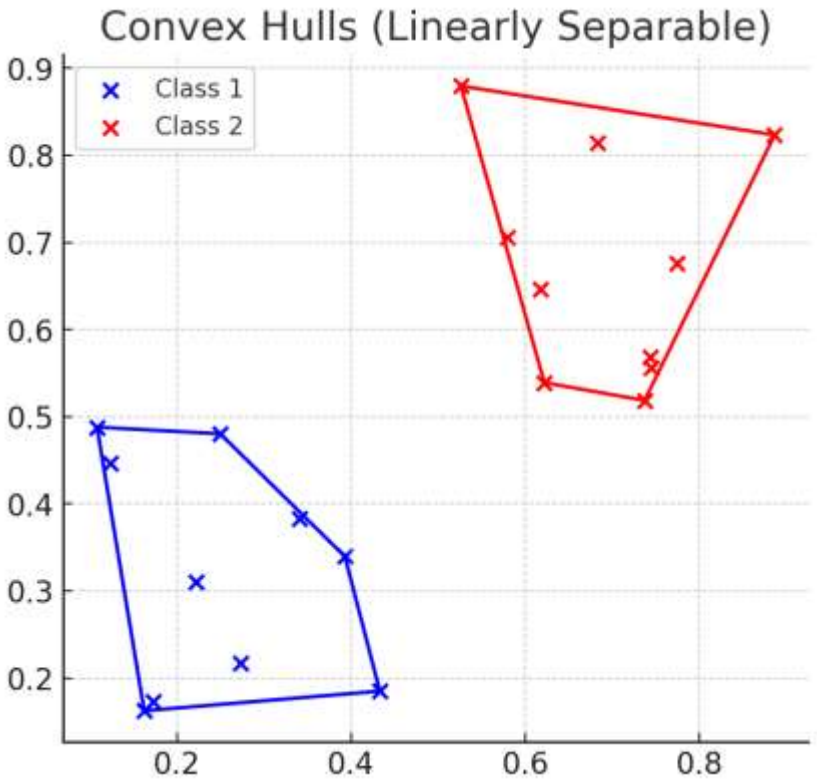
Convex Hull in ANN Context

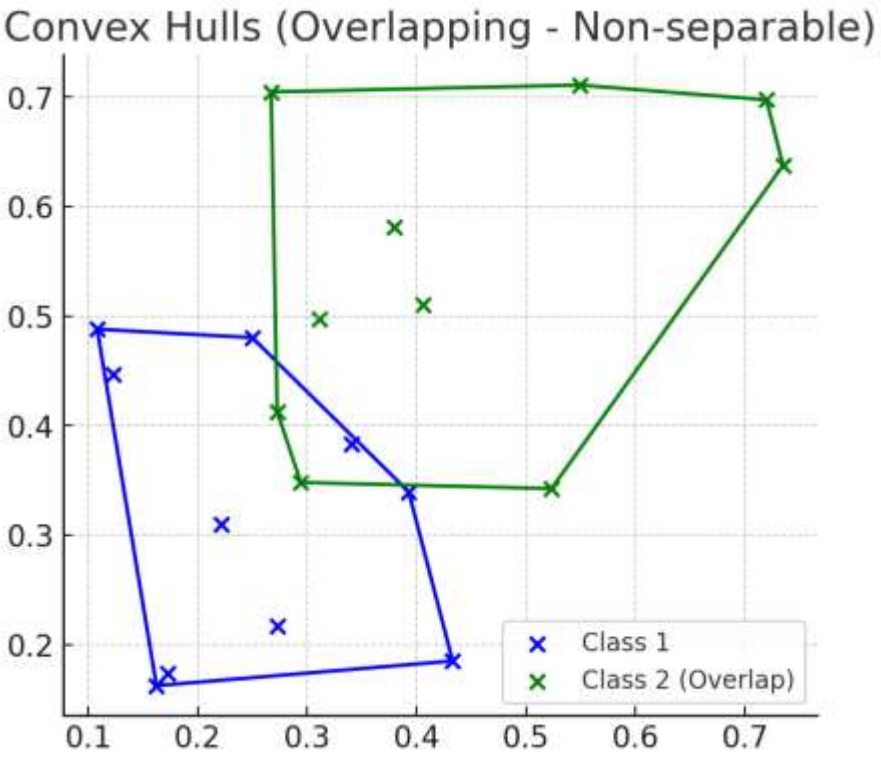
- **Classification & Decision Boundaries**
  - In simple perceptron learning, if two classes' convex hulls **do not overlap**, the data is *linearly separable*.
  - That's why perceptrons can solve AND/OR problems (convex hulls separated), but **fail on XOR** (convex hulls overlap).
- **Geometry of Feature Space**
  - Convex hulls help visualize how training data is distributed in high-dimensional feature spaces.
  - ANN decision boundaries often approximate or cut through convex hulls.
- **Support Vector Machines (SVMs) relation**
  - SVM margin is effectively defined by support vectors lying on the convex hulls of different classes.

Why Convex Hull Matters in Neural Networks

- **Linearly separable data**: Perceptron convergence theorem says a perceptron will find a separating hyperplane *if convex hulls of the classes do not intersect*.
- **Non-linearly separable data** (like XOR): Convex hulls overlap → need Multi-Layer Perceptron's (MLPs) or kernel tricks.

Convex Hull


Convex Hulls (Linearly Separable)
Class 1
Class 2
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.2
0.4
0.6
0.8


Convex Hulls (Overlapping - Non-separable)
Class 1
Class 2 (Overlap)
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.2
0.3
0.4
0.5
0.6
0.7

Linearly separable problems

- A problem is **linearly separable** if there exists a straight line (in 2D), a plane (in 3D), or a hyperplane (in higher dimensions) that can separate the data points of different classes **without error**.
- Mathematically:
- Dataset with inputs $x \in \mathbb{R}^n$and labels $y \in \{-, 1 + 1\}$
- Linearly separable if ∃ weight vector $w$and bias $b$such that:
- $y_i(w \cdot x_i + b) > 0 \; \forall i$

Examples

- **Linearly Separable (can be solved by Perceptron):**
  - **AND Gate**
    Inputs: {(0,0), (0,1), (1,0), (1,1)}
    Output: Only (1,1) → 1, others → 0
    ✓ Can be separated by a line.
  - **OR Gate**
    Output is 0 only when (0,0).
    ✓ Can be separated by a line.
- **Not Linearly Separable (cannot be solved by single perceptron):**
- **XOR Gate**
  (0,0) and (1,1) belong to one class, (0,1) and (1,0) belong to another.
  No single straight line can separate these two groups.
  Requires **Multi-Layer Perceptron (MLP)**.

Perceptron Convergence Theorem (PCT)

- Theorem Statement
- The Perceptron Convergence Theorem states:
- If a dataset is linearly separable, then the perceptron learning algorithm is guaranteed to find a separating hyperplane in a finite number of steps.
- Given a dataset with feature vectors $x_i \in \mathbb{R}^n$and labels $y_i \in \{-, 1 + 1\}$:
- If ∃ weight vector $w^*$and bias b^*such that:
- $y_i(w^* \cdot x_i + b^*) > 0 \; \forall i$
- →The perceptron learning algorithm will converge to a solution vector $w$after a finite number of weight updates.

Weight Update Rule

- $w^{(t+1)} = w^{(t)} + \eta\, y_i\, x_i$
- where:
- η= learning rate
- y_i =true label
- x_i =input vector
- This happens **only when the perceptron misclassifies** the input.

Implications

- **Guarantee of Convergence**
  - If the data is linearly separable → perceptron will always converge.
  - If not linearly separable → algorithm never converges (keeps updating weights).
- **No Guarantee of Optimal Margin**
  - The perceptron finds *a* separating hyperplane, but not necessarily the one with **maximum margin** (unlike SVMs).
- Each update "nudges" the hyperplane closer to correct classification.
- Since the dataset is linearly separable, repeated nudging eventually aligns the hyperplane to separate the classes.

Example

- **AND Gate** (linearly separable): PCT ensures perceptron will converge.
- **XOR Gate** (not linearly separable): PCT does **not apply**, perceptron won't converge → need MLP.
- The **Perceptron Convergence Theorem** gives a mathematical guarantee: *for linearly separable problems, a perceptron will always converge in finite steps*. It is one of the earliest proofs that machine learning models can *learn from data* successfully.

Non-linearly separable problems

- A problem is **non-linearly separable** if no single straight line (in 2D), plane (in 3D), or hyperplane (in higher dimensions) can perfectly separate the data points of different classes.
- Mathematically, if there does **not** exist a weight vector $w$and bias $b$such that:
- $y_i(w \cdot x_i + b) > 0 \; \forall i$
- then the problem is non-linearly separable.

Examples

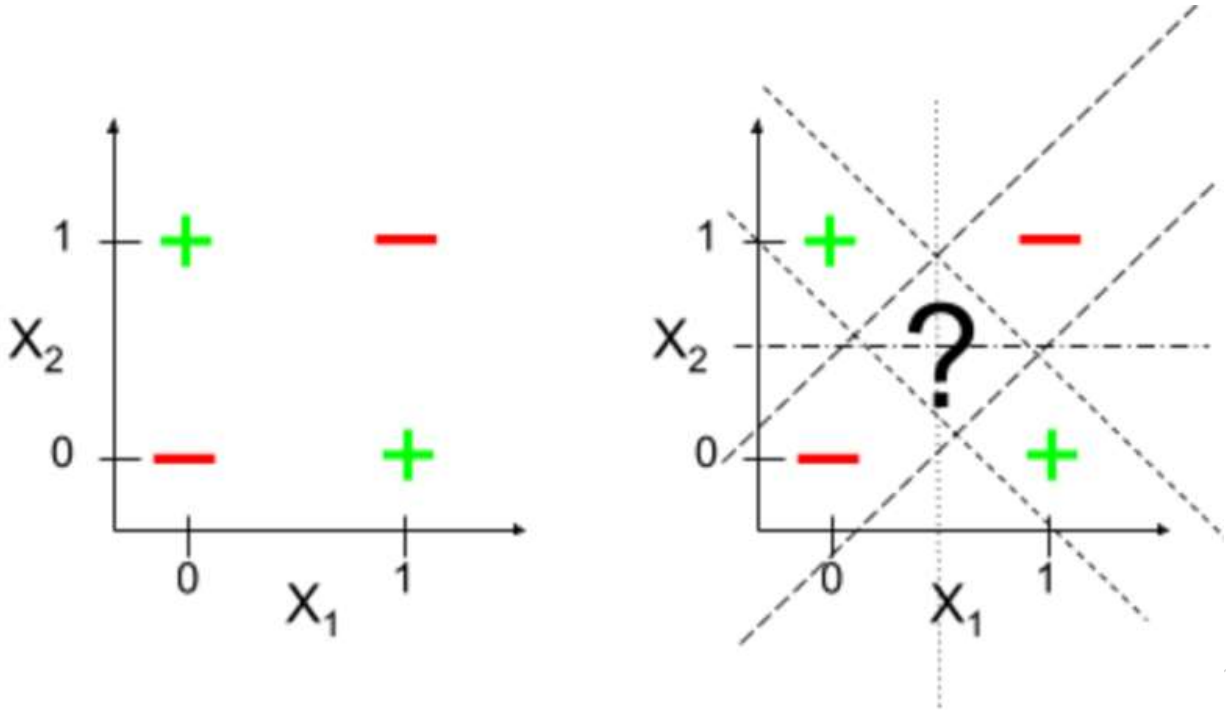
- **XOR Gate**
  - Inputs: (0,0), (0,1), (1,0), (1,1)
  - Outputs: 0, 1, 1, 0
    No straight line can separate the two classes.
    Needs **Multi-Layer Perceptron (MLP)**.
- **Circular Decision Boundary**
  - Example: Class A = points inside a circle, Class B = points outside.
    Cannot be separated by a line; requires a curved boundary.
- **Spirals or Moons Dataset**
  - Popular toy problems where data is distributed in non-linear shapes.
    Require nonlinear models.

In ANN Context

- **Single Perceptron Limitation**:
  A perceptron can only form a **linear decision boundary** → works only for linearly separable data.
- **Multi-Layer Perceptron (MLP)**:
  By stacking hidden layers + nonlinear activation functions (ReLU, sigmoid, tanh), MLPs can model **non-linear decision boundaries** and solve non-linearly separable problems.
- **Universal Approximation Theorem**:
  An MLP with at least one hidden layer and nonlinear activations can approximate any continuous function → hence can handle non-linear problems.

XOR Problem

- A single perceptron draws one straight line (a linear decision boundary) in the (x_1, x_2) plane. For XOR, the 1's are at (0,1) and (1,0), while the 0's are at (0,0) and (1,1). No single line can separate those two classes hence non-linear separability.
(Geometric intuition: the positive points sit on opposite corners; any straight line that groups them together also includes at least one negative point.).



Why Single-Layer Perceptron Fails

- **Linear Decision Boundary**
  - A perceptron computes:
- $y = f(w_1 x_1 + w_2 x_2 + b)$
  - This is always a **linear separator** (straight line in 2D).
- **XOR is Non-Linearly Separable**
  - The two classes in XOR are **diagonal** to each other.
  - Their convex hulls overlap → no linear hyperplane exists that separates them.
  - Hence, perceptron keeps updating weights forever and never converges.

Limitations of Single-Layer Perceptron

- Can only solve linearly separable problems (like AND, OR, NOT).
- Cannot solve non-linear problems (like XOR, circle-in-circle, spirals).
- Provides only linear decision boundaries.
- Limited computational power → not a universal function approximator.

Solution: Multi-Layer Perceptron (MLP)

- Introduce hidden layers + nonlinear activation functions.
- An MLP can combine linear boundaries into nonlinear ones, making it capable of solving XOR and other complex problems.
- This was the historical reason why backpropagation and deep networks became essential in AI.