Simple Perceptron
Binary Linear Classifier

Zbigniew Dendzik

March 2026

Lecture Plan

- Introduction to neural networks
- Binary linear classifier concept
- Perceptron algorithm
- Point2DLabel class – 2D point with binary label
- TrainingSet2D class – training data storage
- Perceptron2D implementation
- Training and convergence
- Decision line geometry
- Non-linearly separable data (XOR problem)

Introduction to Neural Networks

Goal: Learn to classify data based on labeled examples.

Historical context:

- 1943: McCulloch-Pitts neuron model
- 1958: Rosenblatt's perceptron algorithm
- Foundation of modern neural networks

Perceptron:

- Simplest neural network model
- Single artificial neuron
- Binary classification (two classes)
- Linear decision boundary

Binary Linear Classifier

Binary classification: Assign input to one of two classes.

Examples:

- Email: spam / not spam
- Medical test: positive / negative
- Image: cat / dog
- Sentiment: positive / negative

Linear classifier:

- Decision boundary = straight line (2D) or hyperplane (n-D)
- Separates feature space into two regions
- Simple but powerful for linearly separable data

Perceptron Model

Input: Feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$

Parameters:

- Weight vector $\mathbf{w} = (w_1, w_2, \ldots, w_n)$
- Bias b (threshold)

Computation:

1. Weighted sum: $s = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$
2. Activation function: $y = \text{sign}(s) = \begin{cases} +1 & \text{if } s \geq 0 \\ -1 & \text{if } s < 0 \end{cases}$

Output: Class label $+1$ or -1

Geometric Interpretation (2D)

Decision boundary: $w_1 x_1 + w_2 x_2 + b = 0$

In 2D, this is a straight line dividing the plane into two half-spaces:

- Region 1: $w_1 x_1 + w_2 x_2 + b \geq 0$ (class $+1$)
- Region 2: $w_1 x_1 + w_2 x_2 + b < 0$ (class -1)

Example: $x_1 + x_2 - 1 = 0$

- Line: $x_2 = -x_1 + 1$
- Above line: class $+1$
- Below line: class -1

Perceptron Learning Algorithm

Training process: Iteratively adjust weights to reduce errors.

Update rule: For each misclassified example (x, y):

$$w_i \leftarrow w_i + \eta \cdot \text{error} \cdot x_i$$

$$b \leftarrow b + \eta \cdot \text{error}$$

where:

- η = learning rate (e.g. 0.1)
- $\text{error} = y_{\text{true}} - y_{\text{pred}}$

Key idea:

- If error $= 0$: no update (correct prediction)
- If error $\neq 0$: adjust weights in direction of correction

Perceptron Convergence Theorem

Theorem: If training data is linearly separable, the perceptron algorithm converges in finite steps.

Linearly separable:

- There exists a line (2D) or hyperplane (n-D) that perfectly separates the two classes
- No overlapping regions

Convergence:

- After finite iterations, all training examples correctly classified
- Number of errors drops to zero
- Weights stabilize

Non-linearly separable

- Perceptron does NOT converge
- Weights oscillate indefinitely
- Need more complex models (multi-layer networks)

System Architecture

System components:

1. **Data representation**
   - `Point2DLabel` – 2D point with binary label ($+1$ or -1)
   - `TrainingSet2D` – collection of training points

2. **Perceptron**
   - `Perceptron2D` – binary linear classifier
   - Fields: `w1`, `w2`, `bias`, `learningRate`
   - Training: iterative weight updates
   - Prediction: sign of weighted sum

Point2DLabel Class

```
1  class Point2DLabel {
2      private double x;
3      private double y;
4      private int label; // -1 or +1
5
6      public Point2DLabel(double x, double y, int label)
       {
7          this.x = x;
8          this.y = y;
9          this.label = label;
10     }
11
12     public double getX() { return x; }
13     public double getY() { return y; }
14     public int getLabel() { return label; }
15
16     public String toString() {
17         return "Point2DLabel(" + x + ", " + y + ", " +
       label + ")";
18     }
19 }
```

Point2DLabel Class – Explanation

Private fields:

- `x` – x-coordinate
- `y` – y-coordinate
- `label` – binary class label ($+1$ or -1)

Public methods:

- `getX()` – returns x-coordinate
- `getY()` – returns y-coordinate
- `getLabel()` – returns class label
- `toString()` – string representation

Application:

- Storing training examples with labels
- Representing test points
- Basic building block of training set

TrainingSet2D Class

```
1  import java.util.*;
2
3  class TrainingSet2D {
4      private List<Point2DLabel> points;
5
6      public TrainingSet2D() {
7          points = new ArrayList<Point2DLabel>();
8      }
9
10     public void add(Point2DLabel p) {
11         points.add(p);
12     }
13
14     public List<Point2DLabel> getPoints() {
15         return points;
16     }
17
18     public int size() {
19         return points.size();
20     }
21 }
```

TrainingSet2D Class – Explanation

Purpose: Store collection of labeled training points.

Fields:

- `List<Point2DLabel> points` – list of training examples

Methods:

- `add(Point2DLabel p)` – add training point
- `getPoints()` – retrieve all points
- `size()` – number of training examples

Application:

- Organizing training data
- Iterating through examples during training
- Managing labeled datasets

TestTrainingSet2D – Example Program

```
1  public class TestTrainingSet2D {
2      public static void main(String[] args) {
3          TrainingSet2D set = new TrainingSet2D();
4
5          // Class -1: below line y = x
6          set.add(new Point2DLabel(1, 0, -1));
7          set.add(new Point2DLabel(2, 0, -1));
8          set.add(new Point2DLabel(3, 1, -1));
9
10         // Class +1: above line y = x
11         set.add(new Point2DLabel(0, 1, 1));
12         set.add(new Point2DLabel(0, 2, 1));
13         set.add(new Point2DLabel(2, 3, 1));
14
15         for (Point2DLabel p : set.getPoints()) {
16             System.out.println(p);
17         }
18     }
19 }
```

Perceptron2D Class – Fields and Constructor

```
1  class Perceptron2D {
2      private double w1;
3      private double w2;
4      private double bias;
5      private double learningRate;
6
7      public Perceptron2D(double learningRate) {
8          this.w1 = 0.0;
9          this.w2 = 0.0;
10         this.bias = 0.0;
11         this.learningRate = learningRate;
12     }
13
14     public double getW1() { return w1; }
15     public double getW2() { return w2; }
16     public double getBias() { return bias; }
17 }
```

Perceptron2D Class – Explanation

Fields:

- `w1, w2` – weights for x and y coordinates
- `bias` – bias term (threshold b)
- `learningRate` – step size for weight updates (η)

Initialization:

- All weights and bias initialized to 0.0
- Learning rate set by user (typically 0.01 – 0.1)

Decision boundary:

$$w_1 x + w_2 y + b = 0$$

Perceptron2D – sum Method

```
1 public double sum(Point2DLabel p) {
2     return w1 * p.getX() + w2 * p.getY() + bias;
3 }
```

Explanation:

- Computes weighted sum: $s = w_1 x + w_2 y + b$
- Also called *activation* or *net input*
- Used for classification decision

Application:

- If $s \geq 0$: predict class $+1$
- If $s < 0$: predict class -1

Perceptron2D – predict Method

```
1 public int predict(Point2DLabel p) {
2     double s = sum(p);
3     if (s >= 0)
4         return 1;
5     else
6         return -1;
7 }
```

Explanation:

- Activation function: `sign(s)`
- Threshold at zero
- Returns binary label: $+1$ or -1

Classification:

- Determines which side of decision boundary point lies on

Perceptron2D – trainOnExample Method (1)

```
1  public void trainOnExample(Point2DLabel p) {
2      int y = p.getLabel();
3      int yPred = predict(p);
4      int error = y - yPred;
5
6      if (error != 0) {
7          w1 = w1 + learningRate * error * p.getX();
8          w2 = w2 + learningRate * error * p.getY();
9          bias = bias + learningRate * error * 1.0;
10     }
11 }
```

Step 1: Compute prediction and error

- `error = y_true - y_pred`
- If error $= 0$: no update needed (correct prediction)

Perceptron2D – trainOnExample Method (2)

Step 2: Update weights if misclassified

Update rule:

$$w_1 \leftarrow w_1 + \eta \cdot \text{error} \cdot x$$
$$w_2 \leftarrow w_2 + \eta \cdot \text{error} \cdot y$$
$$b \leftarrow b + \eta \cdot \text{error}$$

Error cases:

- $y = +1$, $y_{\text{pred}} = -1$: error $= +2$ (move boundary up)
- $y = -1$, $y_{\text{pred}} = +1$: error $= -2$ (move boundary down)

Effect:

- Adjusts decision boundary to reduce misclassification

TestPerceptron2D – Example Program (1)

```
1  public class TestPerceptron2D {
2      public static void main(String[] args) {
3          TrainingSet2D set = new TrainingSet2D();
4          set.add(new Point2DLabel(1, 0, -1));
5          set.add(new Point2DLabel(2, 0, -1));
6          set.add(new Point2DLabel(0, 1, 1));
7          set.add(new Point2DLabel(0, 2, 1));
8          set.add(new Point2DLabel(2, 3, 1));
9          set.add(new Point2DLabel(3, 1, -1));
10
11         Perceptron2D p = new Perceptron2D(0.1);
12
13         // manual training loop...
14     }
15 }
```

TestPerceptron2D – Example Program (2)

```
int epochs = 10;
for (int e = 0; e < epochs; e++) {
    for (Point2DLabel pt : set.getPoints()) {
        p.trainOnExample(pt);
    }
    System.out.println("Epoch " + e + ": " + p);
}
```

Training process:

- Iterate through all training examples
- Update weights on each misclassified example
- Repeat for fixed number of epochs
- Print weights after each epoch

Observation:

- Weights change initially
- Stabilize when all examples correctly classified

Perceptron2D – countErrors Method

```java
public int countErrors(TrainingSet2D set) {
    int errors = 0;
    for (Point2DLabel p : set.getPoints()) {
        int y = p.getLabel();
        int yPred = predict(p);
        if (y != yPred)
            errors++;
    }
    return errors;
}
```

Explanation:

- Counts number of misclassified examples
- Used to monitor training progress
- Convergence achieved when errors $= 0$

Perceptron2D – train Method

```
1  public void train(TrainingSet2D set, int maxEpochs) {
2      for (int e = 0; e < maxEpochs; e++) {
3          for (Point2DLabel p : set.getPoints()) {
4              trainOnExample(p);
5          }
6          int errors = countErrors(set);
7          System.out.println("Epoch " + e
8              + " errors=" + errors + " " + this);
9          if (errors == 0)
10             break;
11     }
12 }
```

Early stopping:

- Stop when all examples correctly classified
- Saves computation time
- Indicates convergence

TestPerceptron2D – Using train Method

```
1  public class TestPerceptron2D {
2      public static void main(String[] args) {
3          TrainingSet2D set = new TrainingSet2D();
4          // ... add training points ...
5
6          Perceptron2D p = new Perceptron2D(0.1);
7          p.train(set, 50);
8
9          System.out.println("Test on training set:");
10         for (Point2DLabel pt : set.getPoints()) {
11             int yPred = p.predict(pt);
12             System.out.println(pt + " pred=" + yPred
13                 + " " + (yPred == pt.getLabel() ? "OK"
       : "ERROR"));
14         }
15     }
16 }
```

Training Progress Example

Initial weights: $w_1 = 0,\ w_2 = 0,\ b = 0$

Training set: 6 points, 2 classes

Epoch-by-epoch:

- Epoch 0: errors $= 6$ (all wrong initially)
- Epoch 1: errors $= 4$ (weights adjusting)
- Epoch 2: errors $= 2$ (converging)
- Epoch 3: errors $= 1$ (almost there)
- Epoch 4: errors $= 0$ (converged!)

Final weights: $w_1 = 0.4,\ w_2 = -0.2,\ b = 0.1$

Decision boundary: $0.4x - 0.2y + 0.1 = 0$

Decision Line Geometry

Decision boundary: $w_1 x + w_2 y + b = 0$

Standard line form: $y = ax + c$

If $w_2 \neq 0$, we can rewrite:

$$w_1 x + w_2 y + b = 0$$

$$w_2 y = -w_1 x - b$$

$$y = -\frac{w_1}{w_2} x - \frac{b}{w_2}$$

Slope and intercept:

- Slope: $a = -\frac{w_1}{w_2}$
- Intercept: $c = -\frac{b}{w_2}$

Perceptron2D – Decision Line Methods

```
1 public double getA() {
2     return -w1 / w2;
3 }
4
5 public double getC() {
6     return -bias / w2;
7 }
```

Application:

- Visualize decision boundary
- Understand learned model
- Predict for new points using line equation

Example output:

- "Decision line: y = 2.0x - 0.5"

Testing Decision Line

```
1  System.out.println("Decision line: y = "
2      + p.getA() + "x + " + p.getC());
3
4  // Test new points
5  Point2DLabel test1 = new Point2DLabel(5, 5, 0);
6  Point2DLabel test2 = new Point2DLabel(1, 1, 0);
7
8  System.out.println("Point (5,5) -> " + p.predict(test1
       ));
9  System.out.println("Point (1,1) -> " + p.predict(test2
       ));
10
11 // Point on decision line
12 double x = 2.0;
13 double y = p.getA() * x + p.getC();
14 Point2DLabel testLine = new Point2DLabel(x, y, 0);
15 System.out.println("Point on line (" + x + "," + y
16     + ") -> " + p.predict(testLine));
```

Points on Decision Line

Question: What happens for points exactly on the line?

Decision rule: $\text{sign}(s)$ where $s = w_1 x + w_2 y + b$

For points on line: $s = 0$

Implementation choice:

- `if (s >= 0) return 1;`
- Zero maps to class $+1$

Note:

- Points exactly on line are rare in practice
- Convention: $s \geq 0$ means class $+1$
- Alternative: $s > 0$ for $+1$, $s \leq 0$ for -1

Linearly Separable Data

Definition: Two classes are linearly separable if there exists a line (2D) or hyperplane (n-D) that perfectly separates them.

Examples of linearly separable:

- Points above/below line $y = x$
- Points inside/outside a half-plane
- Simple AND, OR logical functions

Perceptron behavior:

- Guaranteed to converge in finite steps
- Finds separating line
- Errors drop to zero

Non-Linearly Separable Data

Definition: No single line can separate the two classes.

Examples:

- XOR problem: $(0,0) \to +1$, $(1,1) \to +1$, $(0,1) \to -1$, $(1,0) \to -1$
- Concentric circles: inner circle $+1$, outer ring -1
- Interleaved spirals

Perceptron behavior:

- Does NOT converge
- Errors never reach zero
- Weights oscillate indefinitely

XOR Problem

XOR (exclusive OR):

x_1	x_2	XOR
0	0	0 $(+1)$
0	1	1 (-1)
1	0	1 (-1)
1	1	0 $(+1)$

Geometric view:

- Opposite corners belong to same class
- No single straight line can separate them
- Classic example of non-linear problem

XOR Training Set

```
1  TrainingSet2D xorSet = new TrainingSet2D();
2  xorSet.add(new Point2DLabel(0, 0, 1));
3  xorSet.add(new Point2DLabel(1, 1, 1));
4  xorSet.add(new Point2DLabel(0, 1, -1));
5  xorSet.add(new Point2DLabel(1, 0, -1));
6
7  Perceptron2D p = new Perceptron2D(0.1);
8  p.train(xorSet, 100);
9
10 System.out.println("Final errors: "
11     + p.countErrors(xorSet));
```

Expected result:

- Errors never reach 0
- Typically oscillate between 2 and 4
- Weights keep changing

Why Single Perceptron Fails on XOR

Fundamental limitation:

- Single perceptron = single linear boundary
- XOR requires at least two lines to separate

Geometric impossibility:

- Try drawing one straight line through $(0, 0)$ and $(1, 1)$ that separates them from $(0, 1)$ and $(1, 0)$
- Impossible!

Historical significance:

- 1969: Minsky & Papert proved perceptron limitations
- Led to "AI winter" in 1970s
- Solved by multi-layer perceptrons (MLP)

Multi-Layer Perceptrons (MLP)

Solution to XOR: Add hidden layer

MLP architecture:

- Input layer: 2 neurons (x_1, x_2)
- Hidden layer: 2+ neurons
- Output layer: 1 neuron (classification)

How it works:

- Hidden layer creates multiple linear boundaries
- Combines them non-linearly
- Can solve XOR and other non-linear problems

Training:

- Backpropagation algorithm
- Gradient descent
- More complex than perceptron learning rule

Learning Rate Effect

Learning rate η: Controls step size of weight updates

Small η (e.g. 0.01):

+ Stable convergence
+ Fine-grained adjustments
– Slow learning
– Many epochs needed

Large η (e.g. 1.0):

+ Fast learning
+ Fewer epochs
– May overshoot optimal weights
– Oscillation or divergence

Typical values: 0.01 – 0.1

Perceptron vs Modern Neural Networks

Aspect	**Perceptron**	**Deep NN**
Layers	Single	Multiple (10-100+)
Parameters	3-4	Millions-billions
Learning	Simple update	Backpropagation
Problems	Linear only	Any complexity
Training	Seconds	Hours-days
Data	Dozens	Thousands-millions
Applications	Simple tasks	Image, NLP, games

Historical importance:

- Foundation of modern deep learning
- Understanding perceptron essential for neural networks

When is Perceptron Useful?

Good applications:

1. Linearly separable binary classification
   - Simple decision boundaries
   - Well-separated classes
2. Educational purposes
   - Understanding neural network basics
   - Learning gradient-based optimization
3. Baseline model
   - Quick check if problem is linearly separable
   - Comparison for more complex models
4. Real-time embedded systems
   - Minimal computational requirements
   - Fast prediction

Limitations of Perceptron

Challenges:

1. Linear separability requirement
   - Cannot solve XOR
   - Fails on complex patterns
2. Binary classification only
   - Only two classes
   - Extension to multi-class needs multiple perceptrons
3. No probabilistic output
   - Only hard classification ($+1$ or -1)
   - No confidence measure
4. Sensitive to feature scaling
   - Large features dominate
   - Need normalization

Extensions of Perceptron

Multi-class perceptron:

- One-vs-all strategy
- Train N perceptrons for N classes
- Choose class with highest activation

Kernel perceptron:

- Apply kernel trick (like SVM)
- Implicit non-linear feature mapping
- Can solve non-linear problems

Averaged perceptron:

- Average weights across all updates
- More stable predictions
- Better generalization

Modern Alternatives

For binary linear classification:

- Logistic regression – probabilistic output
- Support Vector Machines (SVM) – maximum margin
- Linear discriminant analysis (LDA)

For non-linear classification:

- Multi-layer perceptrons (MLP)
- Deep neural networks
- Random forests
- Gradient boosting (XGBoost, LightGBM)

For modern applications:

- Convolutional neural networks (CNN) – images
- Recurrent neural networks (RNN) – sequences
- Transformers – language, vision

Summary

- **Perceptron** – simplest neural network model
- **Binary linear classifier** – separates two classes with line
- **Learning rule** – iterative weight updates on errors
- **Convergence theorem** – guarantees convergence for linearly separable data
- **Decision boundary** – geometric interpretation as line
- **XOR problem** – demonstrates perceptron limitations
- **Non-linear data** – requires multi-layer networks
- **Historical significance** – foundation of deep learning
- **Applications** – simple classification, education, baselines