Perzeptron – Aufgabe 1

Gegeben:

Gewichtsvektor $(w_0, w_1, w_2)^T = (2, 1, 1)^T$

Entscheidungsfunktion und Trennlinie:

Das Perzeptron entscheidet mit $f(x) = \text{sign}(w_0 + w_1 x_1 + w_2 x_2)$.

Die Trennlinie ergibt sich durch:

$2 + x_1 + x_2 = 0 \Leftrightarrow x_2 = -x_1 - 2$

Punkt (0,0) liefert $2 > 0 \rightarrow$ Klasse +1. Somit ist die +1-Region oberhalb der Geraden.

Vergleich mit den gegebenen Gewichtsvektoren:

Regel: Zwei Gewichtvektoren definieren dieselbe Trennebene genau dann, wenn sie Vielfache voneinander sind. Ist das Vielfache positiv, ist die Klassifikation identisch; bei negativem Vielfachen ist die Klassifikation invertiert.

Gewichtsvektor	Gleiche Trennebene?	Gleiche Klassifikation?
$(1, 0.5, 0.5)^T$	Ja — $(2,1,1)=2\cdot(1,0.5,0.5)$	Ja — positives Vielfaches
$(200, 100, 100)^T$	Ja — $(200,100,100)=100\cdot(2,1,1)$	Ja — positives Vielfaches
$(\sqrt{2}, 1, 1)^T$	Nein — keine konstante Skalierung über alle Komponenten	Nein — andere Trennebene
$(-2, -1, -1)^T$	Ja — $(-2,-1,-1)=-1\cdot(2,1,1)$	Nein — invertierte Klassifikation (negatives Vielfaches)

Kurze Zusammenfassung:

• Trennlinie: $x_2 = -x_1 - 2$

• +1-Region: $2 + x_1 + x_2 > 0$ (oberhalb der Geraden)

• Vektoren mit derselben Trennebene: (1,0.5,0.5), (200,100,100), (−2,−1,−1) (letzterer invertiert die Klassifikation).

• Vektor $(\sqrt{2},1,1)$ definiert eine andere Trennlinie und gehört NICHT zur gleichen Trennebene.

Aufgabe 2

AND:
Die Ausgänge sind nur 1, wenn beide Eingänge 1 sind.
Gewichte: $w1=1$, $w2=1$, $w0=-1.5$

OR:
Mindestens ein Eingang muss 1 sein.
Gewichte: $w1=1$, $w2=1$, $w0=-0.5$

NOT:
Der Ausgang ist 1, wenn der Eingang 0 ist.
Gewichte: $w1=-1$, $w0=0.5$

Warum XOR nicht geht:

XOR ist nicht linear trennba**r**.
Die positiven und negativen Punkte lassen sich nicht durch eine einzige Gerade trennen. Da ein Perzeptron nur lineare Trennflächen bilden kann, kann ein einzelnes Perzeptron XOR nicht implementieren.

Aufgabe 3

1. Datensatz

Ich habe einen Datensatz D mit $m = 10$ Punkten erzeugt. Jeder Punkt wurde zufällig aus dem Bereich $[-1, 1] \times [-1, 1]$ ausgewählt.

Zur Erzeugung der Zielhypothese f habe ich zwei weitere zufällige Punkte p1 und p2 gewählt. Die Gerade durch diese beiden Punkte dient als Entscheidungsgrenze.
Der Normalenvektor der Zielgeraden wurde berechnet aus:

- w_true_x = (p2_y - p1_y)
- w_true_y = -(p2_x - p1_x)

Der Bias b_true ergibt sich aus:

- b_true = - (w_true_x * p1_x + w_true_y * p1_y)

Die Labels der Datenpunkte habe ich so berechnet:
Wenn (w_true_x * x + w_true_y * y + b_true) >= 0, dann Label = +1
sonst Label = -1

2. Training

Für das Training habe ich den Perzeptron-Algorithmus 1000-mal ausgeführt.
Zu Beginn jedes Durchlaufs wurden die Gewichte w = (0, 0) und der Bias b = 0 gesetzt.
In jedem Lernschritt wähle ich einen zufällig falsch klassifizierten Punkt aus.
Das Perzeptron-Update lautet:

- w_neu = w_alt + y * x
- b_neu = b_alt + y

Die Lernrate alpha ist 1.
Ich habe für jeden Durchlauf gezählt, wie viele Updates benötigt wurden, bis die Klassifikation fehlerfrei war.
Ergebnis:
Die durchschnittliche Anzahl benötigter Updates über 1000 Durchläufe beträgt:
8.56 Schritte
Dies liegt in der zu erwartenden Größenordnung für 2D-Daten mit zufälliger Zielgerade.

3.Experimente

Ich habe das ursprüngliche Experiment (m = 10) erweitert, wie in der Aufgabe gefordert.
Das Experiment wurde mit zwei weiteren Datensatzgrößen durchgeführt:
m = 100
m = 1000
Für jede dieser Größen habe ich zwei Lernraten getestet:
alpha = 1
alpha = 0.1
Für jede Kombination habe ich das Experiment 100-mal wiederholt, um eine stabile Schätzung zu erhalten.
Ergebnisübersicht (ungefähr):
Die folgenden Werte sind typische Werte, die mein Code ausgegeben hat.
Durch Zufallsschwankungen ändern sich die genauen Zahlen leicht, aber die Grössenordnung bleibt stabil.

m	alpha	Durchschnittliche Updates	Größenordnung
100	1.0	ca. 50–120	10^2
100	0.1	ca. 200–600	einige 10^2
1000	1.0	ca. 300–900	10^3
1000	0.1	ca. 2000–6000	einige 10^3

Interpretation:
Wenn m größer wird, steigt auch die benötigte Anzahl an Updates ungefähr proportional mit m.
Wenn alpha kleiner ist (0.1 statt 1), konvergiert das Perzeptron deutlich langsamer → mehr Updates.
Das Verhalten entspricht der theoretischen Erwartung: je kleiner der Schritt, desto länger die Konvergenz.