Einführung in die Neuroinformatik SoSe 2019

Institut für Neuroinformatik
PD Dr. F. Schwenker

4. Aufgabenblatt (Abgabe bis 4. Juni 2019 zur Vorlesung)

Aufgabe 1 (4 Punkte): Lernschritt im Perzeptron-Lernalgorithmus [Pen and Paper]

Wenn wir bereits wissen, dass unsere Daten linear separierbar sind, dann können wir mit Hilfe des Perzeptron-Lernalgorithmus eine Trennlinie finden, welche die Daten anhand der Klassengrenzen separiert. Wir werden den vollständigen Algorithmus in der nächsten Aufgabe implementieren. Zuerst wollen wir uns jedoch mit dem Lernschritt (Folien 86–87) vertraut machen. Dazu arbeiten wir mit einem einfachen Beispiel im $\mathbb{R}^2$ und zwei Klassen ω_1 und ω_{-1}. Es gibt einen Gewichtsvektor $\boldsymbol{w} \in \mathbb{R}^2$, einen dazugehörigen Bias $w_0 \in \mathbb{R}$ sowie einen einzigen Datenpunkt $\boldsymbol{p} \in \mathbb{R}^2$. Diese sind definiert als

$$\boldsymbol{w} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, w_0 = -1 \quad \text{und} \quad \boldsymbol{p} = \begin{pmatrix} 2 \\ 2 \end{pmatrix} \in \omega_{-1}.$$

Abbildung 1 zeigt den Gewichtsvektor $\boldsymbol{w}$ sowie den Datenpunkt $\boldsymbol{p}$.

1. Wir wollen zuerst wissen, wie die Separierungslinie aktuell aussieht. Zeichne dazu die Linie
$$\boldsymbol{w}^T \boldsymbol{x} + w_0 = 0 \tag{1}$$
in Abbildung 1 ein ($\boldsymbol{x} = (x_1, x_2)$).

2. Um den Lernschritt durchzuführen, ist es einfacher, wenn wir den Bias w_0 in den erweiterten Gewichtsvektor $\boldsymbol{w}^*$ mit aufnehmen. Konsequenterweise müssen wir dann auch den erweiterten Datenpunkt $\boldsymbol{p}^*$ verwenden. Stelle beide erweiterten Vektoren gemäß des Perzeptron-Lernalgorithmus auf.

3. Die Separierungslinie teilt unseren Feature-Raum in zwei Bereiche auf. Die Region von Klasse ω_1 ist definiert als $(\boldsymbol{w}^*)^T \boldsymbol{x}^* > 0$ und die Region von Klasse ω_{-1} als $(\boldsymbol{w}^*)^T \boldsymbol{x}^* < 0$. Markiere beide Regionen in Abbildung 1.

4. Führe einen Lernschritt aus. Prüfe dazu zuerst, ob $\boldsymbol{p}^*$ bereits korrekt klassifiziert ist. Falls dem nicht so ist, berechne den neuen Gewichtsvektor $\tilde{\boldsymbol{w}}$ und den dazugehörigen Bias $\tilde{w}_0$ (verwende eine Lernrate von $\eta = 1$).

5. Zeichne den neuen Gewichtsvektor $\tilde{\boldsymbol{w}}$ und die neue Separierungslinie in ein neues Koordinatensystem (z. B. in Abbildung 2). Markiere auch hier wieder die beiden Regionen, welche zu den zwei Klassen gehören.

6. Wenn wir noch einen weiteren Lernschritt durchführen und dabei wieder den Datenpunkt $\boldsymbol{p}$ verwenden würden, bekämen wir dann einen neuen Gewichtsvektor (einen, der sich von $\tilde{\boldsymbol{w}}$ unterscheidet)?

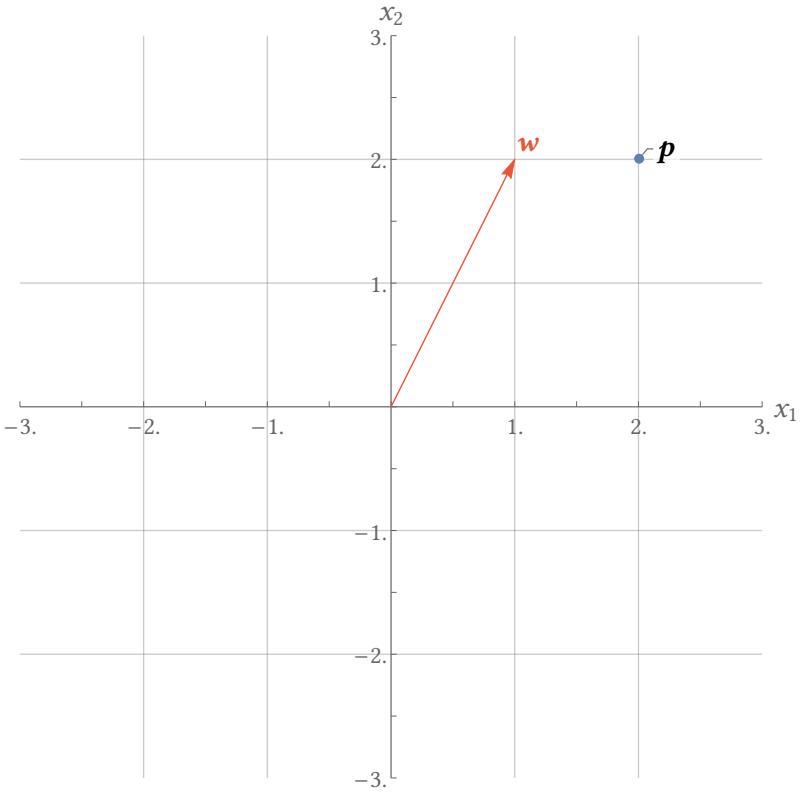

Abbildung 1: Ausgangssituation für Aufgabe 1 mit dem Gewichtsvektor $\boldsymbol{w}$ sowie dem Datenpunkt $\boldsymbol{p}$.

Aufgabe 2 (6 Punkte): Perzeptron-Lernalgorithmus [Python]

In dieser Aufgabe wollen wir uns noch etwas weiter mit dem Perzeptron-Lernalgorithmus beschäftigen. Nachdem wir uns mit dem Lernschritt in der vorherigen Aufgabe befasst haben, wollen wir nun den vollständigen Algorithmus in Python programmieren.

Wir wenden den Algorithmus auf ein Zweiklassenproblem mit Punkten aus dem $\mathbb{R}^2$ (Abbildung 3) an. Es ist leicht zu sehen, dass die Punkte tatsächlich linear separierbar sind. Verwende die Vorlage `PerzeptronLernalgorithmus.ipynb` als Basis, in welcher bereits die Datenpunkte definiert und die Initialisierungsphase des Algorithmus implementiert wurde. Die Punkte sind dabei in einer 10×3 Matrix abgespeichert, wobei die ersten beiden Spalten die (x_1, x_2) Koordinaten bezeichnen und die letzte Spalte das jeweilige Label speichert. Jeder Datenpunkt ist entweder Teil der negativen Klasse ω_{-1} oder Teil der positiven Klasse ω_1.

Allgemein ausgedrückt besteht die Aufgabe darin, den Perzeptron-Lernalgorithmus (Algorithmus 1) zu implementieren. Zur besseren Visualisierung möchten wir die Vorgehensweise des Algorithmus schrittweise darstellen, d. h. den Bearbeitungsstand *nach* jedem Lernschritt anzeigen.

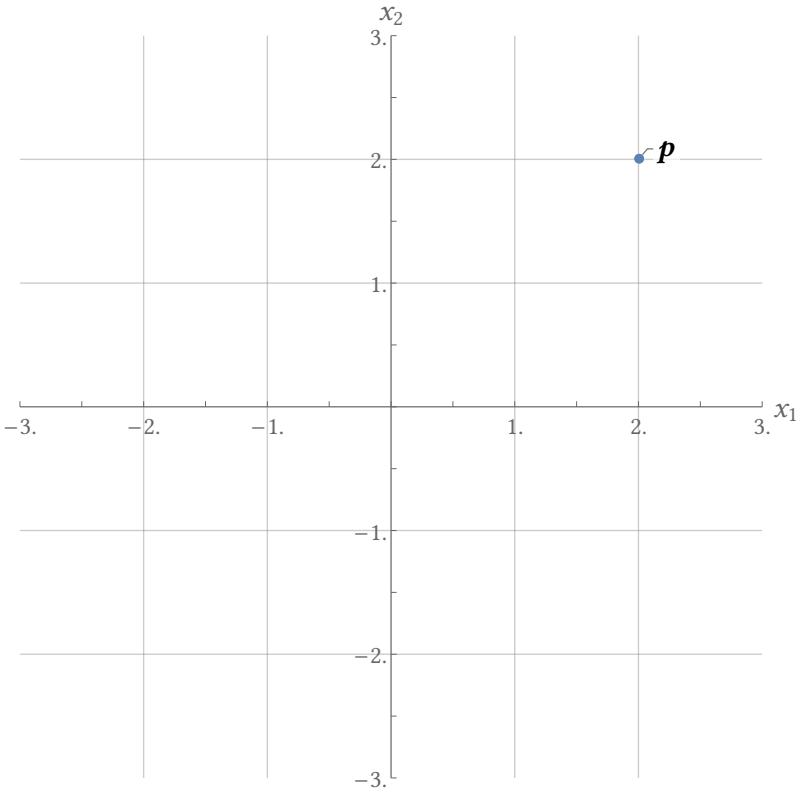

Abbildung 2: Vorlage für die Situation nach dem Lernschritt (Aufgabe 1).

1. Implementiere Algorithmus 1 und speichere nach jeder Bearbeitung eines Datenpunktes eine *Kopie* des aktuellen Gewichtsvektor $\boldsymbol{w}$ in `weight_vectors` ab. Der erste Eintrag in `weight_vectors` enthält dann z. B. den Gewichtsvektor, nachdem der Punkt $(-3, 1)$ bearbeitet wurde. Für das Testskript bitte auch Tabelle 1 beachten.

2. Ziel ist es nun, eine kleine Animation zu erstellen, die uns erlaubt, den Algorithmus schrittweise nachzuvollziehen. Ein Beispiel für eine Iteration ist in Abbildung 4 gezeigt. Euer Plot sollte dabei mindestens die folgenden Bestandteile aufweisen: Punkte der beiden Klassen, der letzte bearbeitete Punkt, die Separierungslinie, den aktuellen Gewichtsvektor $\boldsymbol{w}$, der in Richtung der positiven Fläche $(\boldsymbol{w})^T \boldsymbol{x} \geq 0$ (welche zu ω_1 gehört) zeigt, und selbstverständlich eine saubere Beschriftung (Achsenbeschriftung, Legende, Titel).

   Im Folgenden noch eine generelle Liste mit Anmerkungen für die Implementierung. Ihr dürft diese jedoch gerne noch nach eurem Belieben anpassen.

   - Für die Interaktivität bieten sich die *Jupyter Widgets* an. So kann beispielsweise mit einem `IntSlider` zwischen den einzelnen Iterationsschritten gewechselt werden.
   - Algorithmus 1 arbeitet mit den erweiterten Versionen der Vektoren. Es kann aber trotzdem nützlich sein, die einzelnen Komponenten (Gewichtsvektor (w_1, w_2) und

Bias w_0) zu extrahieren, um sie z. B. im Titel der Abbildung anzuzeigen oder die Separierungsline darzustellen.

- Ihr könnt die Linie $x_2(x_1)$ als eine Funktion der unabhängigen Variablen x_1 darstellen. Vergesst dann jedoch nicht, vertikale Linien gesondert zu behandeln.
- Zuletzt noch ein paar nützliche Python-Befehle zur Visualisierung mit *Matplotlib*: `scatter` (Anzeigen der Punkte), `quiver` (Darstellung des Gewichtsvektors) und `axvspan` sowie `fill_between` (Markierung der Flächen).

Algorithm 1: Perceptron Learning Algorithm

Input: data set $X_1 \cup X_{-1} = X \subset \mathbb{R}^{N \times d+1}$ of N samples; X_1 denotes the data of class ω_1 and X_{-1} the data of class ω_{-1}; data points are extended $\boldsymbol{p} = (1, p_1, p_2, \dots, p_d) \in \mathbb{R}^{d+1}$.
Input: learning rate $\eta > 0$.
Output: extended weight vector $\boldsymbol{w} = (w_0, w_1, \dots, w_d)$.

```
1  w = (0, 0, ..., 0)      // Initialize the extended weight vector
2  change = true            // Stop criterion
3  while change do
4      change = false
5      foreach p ∈ X do
6          if p ∈ ω_1 ∧ ⟨w, p⟩ < 0 then
7              w = w + η · p
8              change = true
9          else if p ∈ ω_{-1} ∧ ⟨w, p⟩ ≥ 0 then
10             w = w − η · p
11             change = true
12         end
13     end
14 end
```

Tabelle 1: Variablennamen für das Jupyter Notebook `PerzeptronLernalgorithmus.ipynb` aus Aufgabe 2. Die Variablen müssen exakt so benannt werden, damit das Testskript deren Inhalt überprüfen kann.

Variablenname	Typ	Beschreibung
`weight_vectors`	Liste	Speichert den Verlauf des erweiterten Gewichtsvektors $\boldsymbol{w}$ über die Iterationen hinweg. Der i-te Eintag in der Liste gibt dabei den Stand des Gewichtsvektors $\boldsymbol{w}$ nach Bearbeitung des j-ten Datenpunktes ($j = i \mod 10$) an.

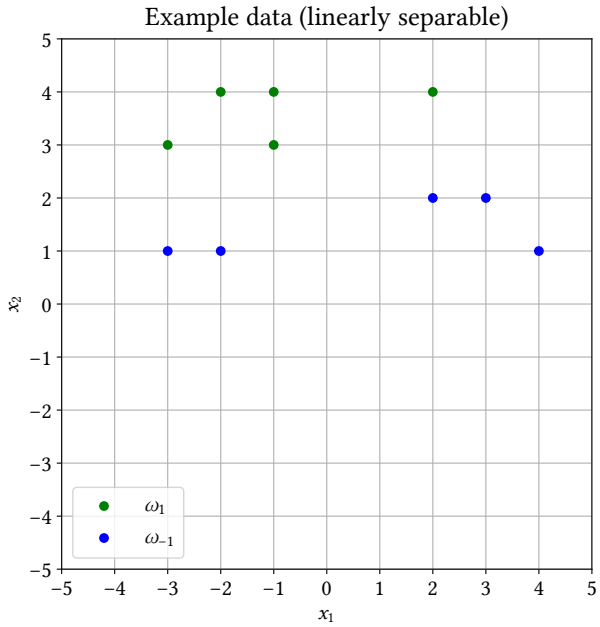

Abbildung 3: Linear separierbare Daten, welche für den Perzeptron-Lernalgorithmus verwendet werden (Aufgabe 2).

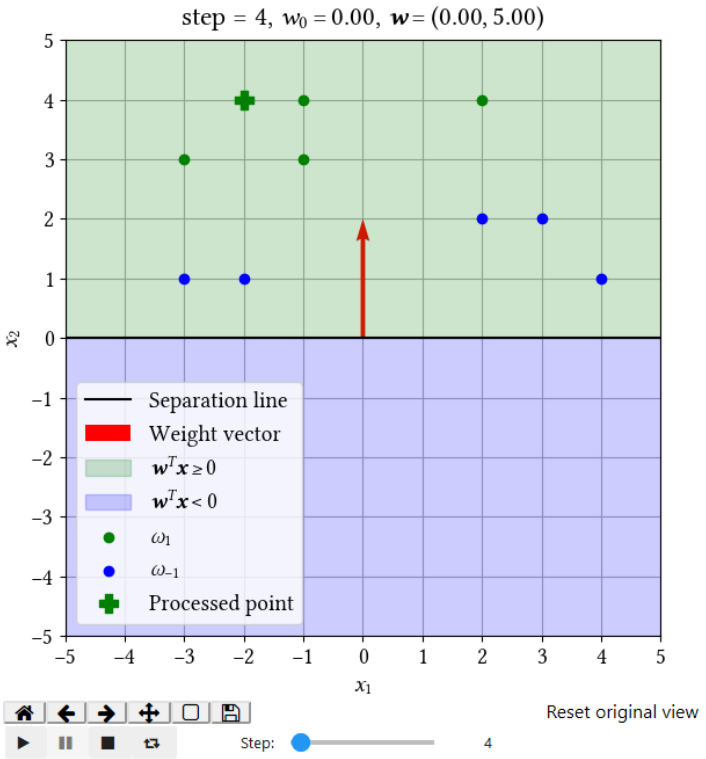

Abbildung 4: Beispielschritt des Perzeptron-Lernalgorithmus. In der Grafik ist die Situation dargestellt, nachdem der Punkt $(x_1, x_2) = (-2, 4) \in \omega_{-1}$ bearbeitet wurde.