Human-Computer Interaction Group

Leibniz Universität Hannover

Vorlesung:
Priv.-Doz. Dr.-Ing. Matthias Becker
eMail: xmb@hci.uni-hannover.de

Übungsbetrieb:
Patric Plattner
eMail: patric.plattner@hci.uni-hannover.de

Aufgabenblatt vom 17.04.2019
Abgabe bis **30.04.2019, 23:59**
Korrektur ab **02.05.2019**

2. Hausübung

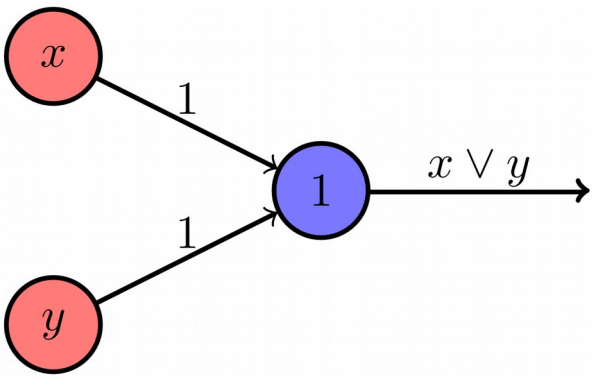
Algorithmen, Klassen, Re-Use, Packages

In diesem Aufgabenblatt sollen eigene Klassen implementiert und in Packages gegliedert werden. Ihr sollt ihr ein Perzeptron implementieren, das Basismodell der momentan so erfolgreichen Neuronalen Netze.

Perzeptron

Die Grundlagen für die teilweise bemerkenswerten Erfolge der Künstlichen Intelligenz der letzten Jahre wurden bereits in den 40/50er Jahre des 20ten Jahrhunderts gelegt. Der einfachste Fall eines ‚künstlichen Neurons' ist das Perzeptron. Dieselbe Struktur findet sich auch in den Deep Neural Networks, die heutzutage benutzt werden, allerdings in Millionenfacher Ausführung und Verknüpfung.

In Anlehnung an natürliche neuronale Netze hat das Perzeptron mehrere Eingänge und einen Ausgang. Die gewichteten Eingangssignale erzeugen zusammengenommen das Ausgangssignal. In der Natur und in mehrlagigen künstlichen Neuronalen Netzen wird dieses Ausgangssignal von weiteren Neuronen weiterverarbeitet.



Ein einfaches Perzeptron mit den Input-Neuronen x und y, welches eine ODER Funktion nachbildet.[1]

Das besondere an neuronalen Netzen ist, dass die gewünschte Funktionalität nicht genau spezifiziert oder entworfen wird, sondern dass das Netz anhand von **Beispielen** trainiert wird. Beispiele bestehen aus dem Inputvektor sowie dem gewünschten Ergebnis, also die Ausgabe. In der Lernphase werden fortwährend Trainingsbeispiele präsentiert, die Ausgabe berechnet und die Gewichte angepasst, falls die Ausgabe anders als im Trainingsbeispiel ist.

Als Ergebnis hat das Perzeptron dann den Zusammenhang zwischen Ein- und Ausgabe ‚gelernt'. Eine weitere Eigenschaft des gelernten Perzeptron ist die Fähigkeit zur

[1] Von MartinThoma - Eigenes Werk, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=28211506

Alle Abgaben sind bis zu dem in der Kopfzeile vermerkten Datum in unserem UploadTool abzugeben. Eine Anleitung zum Hochladen findet ihr hier. Solltet ihr es nicht schaffen, innerhalb des vorgegebenen Zeitraumes die Lösung korrekt abzugeben, wird dies zum nichtbestehen der Studienleistung führen. Bei technischen Problemen mit dem UploadTool bitte eine Mail an patric.plattner@hci.uni-hannover.de schicken.

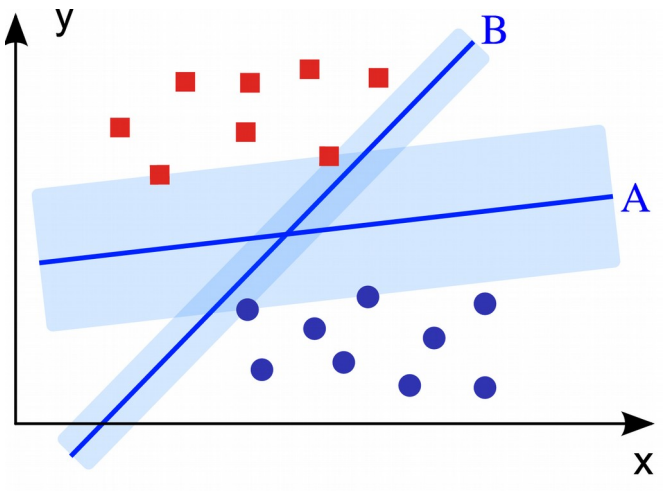
Vorlesung:
Priv.-Doz. Dr.-Ing. Matthias Becker
eMail: xmb@hci.uni-hannover.de

Übungsbetrieb:
Patric Plattner
eMail: patric.plattner@hci.uni-hannover.de

Aufgabenblatt vom 17.04.2019
Abgabe bis **30.04.2019, 23:59**
Korrektur ab **02.05.2019**

Verallgemeinerung: Auch Eingaben, die nicht explizit gelernt wurden in der Trainingsphase, erzeugen (hoffentlich) eine korrekte Ausgabe.

Abseits von der Anlehnung an die Natur ist ein Neuronales Netz nichts Anderes als ein linearer Klassifikator, der durch Geraden bzw. Hyperebenen die Trainingsbeispiele in Klassen entsprechend der Ausgabe separiert.



Zwei Trenngeraden mit verschiedenen Randgrößen[2]

Aufgabe 1

Implementiere ein Perzeptron, was für Eingabevektoren bestehend aus einer Temperatur und Niederschlagsmenge eine Wetterklassifikation in ‚gut' und 'schlecht' ausgibt.

Lege dazu eine Menge von Trainingsvektoren an, mit denen du das Netz trainierst und gib sie diese zusammen mit dem Quellcode ab.

Vorgehensweise:

- Erstelle ein Package de.uni_hannover.hci.<dein_name>. Arbeite von nun an in diesem Package.
- Erstelle Sie die Klasse `Main`. Über diese wird das Trainieren des Perzeptrons mit Übergabe der Trainingsmenge, Ausgabemenge, Lernrate und Anzahl der Epochen ausgeführt.
- Lege eine Trainingsmenge der Größe N an als Array Nx2 (`new double[N][2]`), wobei jeder der N Einträge des Arrays aus (Temperatur, Niederschlag) besteht. Lege eine Ausgabemenge der Größe N an, welches auf die Trainingsmenge bezogen die

[2]By Ennepetaler86 (Own work) [GFDL (http://www.gnu.org/copyleft/fdl.html), GFDL (http://www.gnu.org/copyleft/fdl.html) or CC BY 3.0 (http://creativecommons.org/licenses/by/3.0)], via Wikimedia Commons

Ausgabe in ‚gut' oder ‚schlecht' kodiert als 0 bzw. 1 angibt. Dies sind die zu erwarteten Werte. Du kannst eigene Daten generieren oder die Beispieldaten übernehmen, welche du am Ende des Übungsblattes findest.

- In der Trainingsmenge soll im Prinzip gelten: Temperatur hoch und Niederschlag niedrig soll gut sein. Temperatur niedrig und Niederschlag hoch soll schlecht sein.
- Implementiere die Klasse `Perzeptron` im Package `perzeptron`, welche ein Array für Gewichte besitzt und die Lernrate sowie die Anzahl der Epochen übergeben bekommt. Erstelle einen Konstruktor, der die Anzahl der Einträge im Gewichtearray, die Anzahl der Epochen und die Lernrate als Eingaben nimmt und ein passendes Perzeptron erstellt.
- Implementiere die Klasse `Data` und `DataReader` im Package `datareader`. Der `DataReader` ist für die Überführung der Daten aus den Trainingsvektoren und den zugehörigen Ausgaben in eine objektorientierte Datenstruktur `Data` verantwortlich. Die Klasse `Data` besitzt einen Eingabevektor sowie die zugehörige Ausgabe und einen passenden Konstruktor. Die Klasse `DataReader` besitzt eine statische Methode `read(traindata, trainoutput)`, welches eine Array von `Data` zurückgibt. In dieser befinden sich indexweise die Elemente aus `traindata` und `trainoutput` in den zugehörigen Attributen aus `Data`.
- Erstelle die Methoden `train(traindata)` und `output(input)` in der Klasse `Perzeptron`. Die Methode `train(traindata)` bekommt eine Array von Daten von `DataReader` übergeben. Die Methode `output(input)` berechnet das Skalarprodukt zwischen den Gewichten und des Eingabevektors: $\sum_{j=1}^{m} w_j \cdot i_j$. In diesem Falle: $w_1 i_1 + w_2 i_2$. Ist das Ergebnis größer gleich 0 so wird eine 1 zurückgegeben, andernfalls 0.
- Die Methode `train(…)` befüllt das Gewichtsarray zu Beginn mit zufälligen Werten zwischen 0.0 und 1.0[3]. Das Gewichtsarray besitzt dieselbe Länge wie die Eingabevektoren. Das Perzeptron trainiert solange bis die Anzahl der Epochen erreicht wird. Bei jedem Training wird über die gesamte Trainingsdatenmenge iteriert und dann für jedes einzelne Element (Temperatur, Niederschlag).
  Für den aktuellen Datensatzes wird das Gewicht wie folgt angepasst: `weights[k] += learningrate * error * traindata[j][k]`. `error` ist hierbei `erwarteter Wert – errechneter Wert` bezüglich des aktuellen Datensatzes.
- Wenn 95% der Trainingsbeispiele richtig erkannt werden, soll das Training beendet werden. Sammeln Sie hierfür während des Trainings die erforderlichen Daten in einer Array. Erstellen Sie eine weitere Methode, welche anhand der Werte die

[3] Sie können dafür die Klasse java.util.Random nehmen und die Methode Random.nextDouble nutzen https://docs.oracle.com/javase/8/docs/api/java/util/Random.html

Vorlesung:
Priv.-Doz. Dr.-Ing. Matthias Becker
eMail: xmb@hci.uni-hannover.de

Übungsbetrieb:
Patric Plattner
eMail: patric.plattner@hci.uni-hannover.de

Aufgabenblatt vom 17.04.2019
Abgabe bis **30.04.2019, 23:59**
Korrektur ab **02.05.2019**

Erkennungsrate errechnet.

- Trainiere das Perzeptron und testen Sie das Perzeptron sowohl mit Trainingsvektoren als auch mit anderen Eingabepaaren. Gib diese wie folgt aus: Temperatur: 11.2, Niederschlag: 2.1 -> Schlechtes Wetter
- Was passiert, wenn die Trainingsbeispiele nicht einfach linear separierbar sind? Also wenn gutes/schlechtes Wetter wie oben definiert ist, zusätzlich aber auch sehr kalt + Niederschlag (=schönes Winterwetter) als gut auftaucht? Erkläre dies deinem Tutor. Optionale Zusatzaufgabe: erweiter das Neuronale Netz mit einem Perzeptron um ein weiteres Ausgabeneuron, was aktiviert wird, wenn ‚schönes Winterwetter' auftaucht.

Lösungshinweise:

- Fuer einige der Mathematischen Operationen koennt ihr eure MatrixOperations Klasse aus dem ersten Uebungsblatt wiederverwenden.

```
TRAIN_DATA[][] =
            {{10.7f, 6.1f}, {4.9f, 3.4f}, {3.8f, 5.9f}, {1.7f, 4.7f},
            {8.9f, 2.6f}, {8.9f, 1.8f}, {8.8f, 1.9f}, {13.4f, 1.7f},
            {19.8f, 0.0f}, {18.9f, 0.1f}, {15.8f, 0.0f}, {12.1f, 1.4f},
            {8.6f, 3.8f}, {5.8f, 1.6f}, {2.3f, 2.9f}, {3.3f, 5.3f},
            {14.9f, 0.5f}, {18.8f, 0.3f}, {22.8f, 0.0f}, {27.3f, 0.3f},
            {30.7f, 0.0f}, {30.6f, 0.0f}, {27.6f, 0.0f}, {23.2f, 0.2f}};

 TRAIN_OUTPUT[] =
           {1,1,1,1,
            1,1,1,1,
            0,0,0,1,
            1,1,1,1,
            1,0,0,0,
            0,0,0,0};

TEST_DATA[][] =
{{11.2f, 2.1f}, {23.2f, 0.1f}, {16.0f, 0.0f}, {0.5f, 1.7f}, {14.9f, 1.7f}};
```

Aufgabe 2

Schreibe strukturierten und formatierten Code, der sich an eine gängige Code-Konvention hält. Nur so ist der Code von anderen lesbar. Nur lesbarer Code kann debugged werden bzw. von anderen weiterverwendet oder gewartet werden.

Kommentiere dein Programm mit sinnvollen, normalen Kommentaren sowie mit javadoc-Kommentaren.

Erzeuge die html Dateien für deine Implementierung und füge diese der Abgabe hinzu.

Human-Computer Interaction Group

Leibniz Universität Hannover

Vorlesung:
Priv.-Doz. Dr.-Ing. Matthias Becker
eMail: xmb@hci.uni-hannover.de

Übungsbetrieb:
Patric Plattner
eMail: patric.plattner@hci.uni-hannover.de

Aufgabenblatt vom 17.04.2019
Abgabe bis **30.04.2019, 23:59**
Korrektur ab **02.05.2019**

Javadoc ist Thema der Vorlesung am 24.4.

Alle Abgaben sind bis zu dem in der Kopfzeile vermerkten Datum in unserem UploadTool abzugeben. Eine Anleitung zum Hochladen findet ihr hier. Solltet ihr es nicht schaffen, innerhalb des vorgegebenen Zeitraumes die Lösung korrekt abzugeben, wird dies zum nichtbestehen der Studienleistung führen. Bei technischen Problemen mit dem UploadTool bitte eine Mail an patric.plattner@hci.uni-hannover.de schicken.