# NNW-Übung 1

## Bemerkungen zu Python

Empfehlung für Python auf Ihrem eigenen Rechner: Python-Distribution *Anaconda*[1]. Anaconda enthält die meisten benötigten Pakete und zudem die Entwicklungsumgebungen *Spyder* (Desktop-Applikation) und *Jupyter* (läuft im Webbrowser). Python ist ein Interpreter und erlaubt eine *interaktive Befehlseingabe*, was ein schnelles Ausprobieren erleichtert, z.B. bei Spyder in der „IPython-Shell" rechts unten.

Auf der Moodle-Seite finden Sie Links zu Einführungen in Python. Ein auffälliger Unterschied zu Java/C ist, dass Blöcke *nicht* durch geschweifte Klammern, sondern durch *Einrückung* (indentation) gekennzeichnet werden. Empfohlen werden 4 Leerzeichen je Einrückungstiefe (Achtung: *nicht* mit Tabulatoren mischen). Semikolons werden normalerweise nicht gebraucht – nur bei mehr als einer Anweisung pro Zeile.

Hilfe zu Funktionen kann in Spyder durch Drücken von `Strg-I` aufgerufen werden oder in der „IPython-Shell" ein ? hinter die Anweisung eintippen, z.B. `print?`.

## 1 Arbeiten mit NumPy

`NumPy` ist eine Erweiterung von Python für das Rechnen mit Arrays (Matrizen).[2]

Probieren Sie die folgenden Befehle *nacheinander* (!) aus und vollziehen Sie die Ergebnisse nach:

```
import numpy as np # numpy heißt jetzt np ... spart Tipparbeit!

W=np.array([[1.,2,3],[3,4,5]])
W.shape   #Zeilen, Spalten
W.shape[0]
W.shape[1]

W.T
W.T.shape # shape ist keine Funktion, sondern eine Eigenschaft

type(W)
type(W.shape)

# Aufruf der NumPy-Funktion arange:
r=np.arange(10)
print(r)
r.shape

M=np.arange(12).reshape(3,4)
M
M.shape

M[2,0]
M[1,2]
```

[1] `https://www.anaconda.com/products/individual`

[2] Falls Sie Erfahrungen mit MATLAB haben: Einige Konstrukte sind ähnlich, es gibt aber auch einige deutliche Unterschiede. Auf der Moodle-Seite ist ein Link zu einer Seite mit einer Gegenüberstellung von MATLAB und NumPy.

```
M[1,:]
M[1]
M[:,1]   # ergibt Zeile statt Spalte
M[:,[1]] # ergibt eine Spalte
M[:,[3,0,1,1]]
M[:,2:4] # Spalte 2 bis exklusive Spalte 4
M[-2,:]  # 2te Zeile von hinten
M[-2:,:] # 2te Zeile von hinten bis Ende
M[:,2]=2 # ändern ist möglich

M>4
M[M>4]
M[M>4]=-17
M
```

Beachten Sie bei dem nun folgenden Code:

- Die Einrückung (indentation) der Zeilen zur Definition von Blöcken (siehe oben)!
- Benannte Parameter: Optionale Parameter *können* durch Angabe ihres Bezeichners gesetzt werden (statt über die Position des Parameters), im Beispiel bei `print` z.B. `sep=''`.

  Was diese Parameter bedeuten, verrät Spyder Ihnen, wenn Sie `Strg-I` drücken.

```
s=0
for i in range(M.shape[0]):
    for j in range(M.shape[1]):
        s+=M[i,j]
        print(s,',',sep='',end='') # Parameter sep und end über Namen setzen
print(s)

def func(x):
    print(x*x)
    return x

func(3)
```

## 2 Laden und Plotten der Iris-Daten

In der Vorlesung wird folgende Konvention verwendet:

- Die Merkmale $x_1, x_2, \ldots$ *eines* Objekts werden in *Spalten*vektoren $\vec{x}$ gespeichert. Die Merkmale stehen also in den *Zeilen*.
- Haben wir viele Objekte, z.B. $N$ Trainingsbeispiele, so speichern wir diese Daten in einer Matrix $X$: *je Trainingsbeispiel eine Spalte*! Das ist nichts anderes, als das Nebeneinanderschreiben der Merkmalsvektoren (Spalten) der einzelnen Objekte!
- Die Zielwerte bei supervised learning sollen entsprechend in einer Matrix $T$ (für target) stehen.

Auf der Moodle-Seite finden Sie die Iris-Daten als „comma-separated-values“ (`.csv`).

a) Schauen Sie sich den Inhalt der Datei an, zum Beispiel mit einem Text-Editor (wie `nano` (Unix) oder Notepad++ (Windows)). Gespeichert sind die 4 Merkmale und die Iris-Art (kodiert als Zahl von 0 bis 2).

   Wie viele Daten gibt es je Iris-Art?

b) Laden Sie die Daten in Python mit Hilfe der NumPy-Funktion `loadtxt`. Die Beschreibung der Parameter der Funktion und des Rückgabewerts finden Sie im Internet oder durch Drücken von `Strg-I` in Spyder.

Tipp: Sie brauchen (auch) den Parameter `delimiter`.

c) Zerlegen Sie die eingelesen Daten in die Matrizen $X$ und $T$.

d) Stehen die Daten je Blüte in den Zeilen oder Spalten? Passen Sie die Daten ggf. der Konvention (siehe oben) an.

e) Plotten Sie die ersten beiden Merkmale der Daten mit der Funktion `scatter` aus dem Modul `matplotlib.pyplot`, das Sie daher zuerst importieren müssen:

```
import matplotlib.pyplot as plt
```

f) Färben Sie die drei Iris Arten durch Übergabe der folgenden zusätzlichen Parameter von `scatter` ein: `c=T, cmap=plt.cm.prism` (unter der Annahme, dass `matplotlib.pyplot` als `plt` importiert wurde).

Alternativ können Sie auch selbst eine Color-Map definieren:
`cmap=colors.ListedColormap(['red', 'green', 'blue']`, wobei Sie vorher colors importieren müssen: `from matplotlib import colors`

## 3 Neuron mit voreingestellten Gewichten

Schreiben Sie eine Funktion `neuron(X)`, die die Berechnungen eines Neurons mit *zwei* festen Gewichten `W=[-0.3,1]` und fester Aktivierungsschwelle von $> 2$ für *alle* Daten (=vermessene Iris-Blüten) in $X$ durchführt, wobei als Daten wieder nur die ersten zwei Merkmale je Blüte verwendet werden sollen. Der Rückgabewert soll ein Array mit boolschen Werten (`True`=Neuron sendet Signal, `False`=Neuron sendet kein Signal) sein, das genauso viele XXX wie $X$ enthält, wobei XXX für „Spalten“ oder „Zeilen“ steht.

Tipps/Anmerkungen:

- Legen Sie in der Funktion zunächst ein mit Nullen gefülltes Array `net` an, das die Netzwerksummen für die einzelnen Blüten aufnehmen soll: `net=np.zeros(X.shape[XXX])` (`XXX` steht wieder für Spalten oder Zeilen).
- Definieren Sie `W=[-0.3,1]` einfach als lokale Variable der Funktion.
- Benutzen Sie `for`-Schleifen zur Berechnung der Netzwerksummen `net[n]`, wobei `n` für die Nummer der Blüte stehen soll.
- Die Anwendung der Aktivierungsschwelle ist dann nur noch eine Zeile, siehe Aufgabe 1.

Auf der Moodle-Seite finden Sie die Datei `nnwplot.py`. Importieren Sie diese mit `import nnwplot` (die Datei in dasselbe Verzeichnis kopieren wie Ihr Skript) und rufen Sie die darin enthaltenden Funktion `nnwplot.plotTwoFeatures(X,T,neuron)` auf, wobei `X` wie beschrieben nur zwei Merkmale je Blüte enthalten darf.

Mit `plt.figure()` können Sie neue Plot-Fenster öffnen, damit Sie die verschiedenen Plots zur Beantwortung der folgenden Fragen besser vergleichen können. Wenn dann zuviele Fenster offen sind, können Sie sie mit `plt.close('all')` alle schließen.

a) Experimentieren Sie mit anderen Aktivierungsschwellen als 2, also zum Beispiel 1 und 3. Was ändert sich?

b) Experimentieren Sie mit anderen Gewichten `W`, zum Beispiel `W=[-0.2,1]`,`W=[-0.1,1]`, `W=[0,1]`. Was ändert sich?