NNW-Übung 2

Bemerkungen zu Spyder

Im Editor können Sie (in Analogie zu MATLAB) mit der Kommentarzeile `#%%` Blöcke („cells“) definieren, die einzeln abgearbeitet werden können: Start über die beiden Icons *neben* dem grünen Pfeil oder `Strg-Enter` (aktuellen Block ausführen) bzw. `Shift-Enter` (aktuellen Block ausführen und zum nächsten springen). Markierte Zeilen können durch Drücken von `F9` ausgeführt werden.

1 Arbeiten mit Python und NumPy – Teil 2

Probieren Sie den folgenden Python-Code zeilenweise auf der Console aus und vollziehen Sie ihn nach:

```
import numpy as np

x=np.array([1.,2,3]) # Der Punkt bewirkt Datentyp float.
W=np.array([[1.,2,3],[4,3,2]]) # Der Punkt bewirkt Datentyp float.

W   # auf der Console werden Variablen ohne Zuweisung ausgegeben
    # in Skripten müssen Sie print(W) für Ausgaben schreiben

x.shape
W.shape

W.T
x.T       # seltsam ...? Logik:  1D bleibt 1D. x ist *kein* Spalten-Vektor!
y=x[np.newaxis].T # erzeugt (a) 2D Matrix 1x3 (=Zeilenvektor!), (b) transponiert sie.
y
y.shape


# Zu Klassen siehe https://docs.python.org/3/tutorial/classes.html
# und insbesondere https://docs.python.org/3/tutorial/classes.html#tut-private
class myclass:
    def __init__(self,W): # Konstruktor, self entspricht this in Java/C++
        self._W = W       # Konvention: private Variablen beginnen mit _
                          # self muss beim Zugriff angegeben werden, sonst
                          # ist es eine lokale Variable
    def f(self): # self muss immer explizit übergeben werden, sonst "static"
        return self._W*2

m=myclass(np.array([-1,1]))
m._W  # Zugriff ist auch auf private Variablen erlaubt
m.f() # m wird automatisch als "self" übergeben
```

Aus dem History-Tab von Spyder (zweites Tab im Fenster ganz rechts unten) können Sie den Code nachträglich in eine Datei im Editor-Fenster kopieren und abspeichern.

2 Klasse für ein einschichtiges Netz

Erstellen Sie eine Klasse `SLN` (für single-layer network) für ein einschichtiges Netz.

a) Implementieren Sie einen Konstruktor, der die folgenden Parameter erhält:

- `dIn`: Anzahl der Eingabe-Werte (in der Vorlesung mit d (bzw. d_k) bezeichnet)
- `cOut`: Anzahl der Neuronen/Ausgabe-Werte (in der Vorlesung mit c bezeichnet)

und der außerdem die folgenden Instanz-Variablen initialisiert:

- `_W`: Matrix der Gewichte (ohne Bias!). Initialisieren Sie `_W` mit zufälligen, normalverteilten Werten initialisiert (mit `numpy.random.randn`). Initialisieren Sie den Zufallsgenerator vorher durch den Aufruf `numpy.random.seed(42)`, um reproduzierbare Ergebnisse zu erhalten. Teilen Sie die generierten Zufallszahlen durch die Wurzel aus `dIn+1`.
- `_b`: (Spalten-)Vektor mit den Bias-Gewichten. Initialisieren Sie `_b`=$\vec{0}$ (mit `numpy.zeros`). Erzeugen Sie `_b` als Spaltenvektor.
  Tipp: aus einem 1D-Vektor `v` erhalten Sie (siehe Aufgabe 1) einen Spaltenvektor mit `v=v[np.newaxis].T`.

b) Kopieren Sie Ihre Funktion `neuron(X)` von dem vorigen Aufgabenblatt in die Klasse. Ändern Sie die Funktion so ab, dass sie die Instanzvariable `_W` benutzt (und überhaupt benutzen kann), wobei Sie für diese Aufgabe weiter davon ausgehen können, dass nur ein Neuron (`cOut`=1) verwendet wird. Ergänzen Sie die Verwendung des Bias `_b`.

c) Testen Sie, ob Ihre neue Klasse bei den Experimenten des letzten Aufgabenblatts dieselben Ergebnisse liefert. Setzen Sie dazu (entgegen der Konvention) die eigentlich privaten Variablen `_W` und `_b` entsprechend.

d) Die im Folgenden definierte globale Funktion `ErrorRate(Y, T)` soll die Fehlklassifikationsrate ausrechnen. Die Fehlklassifikationsrate ist die Anzahl der inkorrekt klassifizierten Trainingsmuster dividiert durch die Gesamtanzahl der Trainingsmuster. Übernehmen Sie die Funktion in Ihren Code – Sie brauchen sie dann in der nächsten Teilaufgabe. Vollziehen Sie den „if-Teil“ der Funktion nach – der „else-Teil“ ist erst für die Unterscheidung von mehr als zwei Klassen relevant, was noch nicht Teil dieses Aufgabenblatts ist.

```
def ErrorRate(Y,T):
    if Y.ndim==1 or Y.shape[0]==1:
        errors=Y!=T
        return errors.sum()/Y.size
    else: # für mehrere Ausgaben in one-hot Kodierung:
        # Dies brauchen Sie jetzt noch nicht nachzuvollziehen.
        errors=Y.argmax(0)!=T.argmax(0)
        return errors.sum()/Y.shape[1]
```

3 Delta-Regel

a) Implementieren Sie die Delta-Regel als Methode `DeltaTrain` der Klasse `SLN` mit den Parametern `X` und `T` für die Trainingsbeispiele, `eta` für die Lernrate η, `maxIter` für die maximale Anzahl Iterationen und `maxErrorRate` für die maximale Fehlklassifikationsrate.

Der Bias `b` soll mit trainiert werden, also genauso behandelt werden wie die übrigen Gewichte. Implementieren Sie das Verfahren als Batch-Learning.

Der Lernvorgang soll abgebrochen werden, wenn entweder die maximale Anzahl `maxIter` Iterationen (Lernzyklen, Epochen) erreicht oder die maximale Fehlklassifikationsrate `maxErrorRate` unterschritten wurde.

b) Optional: Lassen Sie sich zudem die Trennlinie nach jedem Lernschritt (oder einigen Lernschritten) mit `nnwplot.plotTwoFeatures` ausgeben. Damit der Update des Plots funktioniert, einmalig (vor der Trainings-Schleife) mit `plt.ion()` den „interaktiven Modus“ aktivieren.

c) Optional: Speichern Sie während der Iteration die Gewichte, die zu dem geringsten Fehler geführt hat. Achtung: Zuweisungen von Objekten sind nur Verweise, keine Kopien. Numpy-Arrays haben die Methode `copy` oder das Modul `copy` importieren und daraus die Funktion `copy` (oder `deepcopy`). Setzen Sie die Gewichte am Ende auf diese Werte (also nicht (unbedingt) auf die Werte, die am Ende aller Iteration vorliegen).

d) Trainieren Sie ein „Netz“ für das UND-Gatter-Problem.

e) Trainieren ein Netz zur Klassifikation (*nur*) der Iris-Arten „Iris Setosa“ und „Iris Versicolour“ (Klassen 0 und 1).

ACHTUNG: Unbedingt kontrollieren, dass Ihr Target-Vektor `T` wirklich nur 0en und 1en enthält und *keine* 2en – sonst funktioniert das Training nicht.

Verwenden Sie zunächst nur die ersten beiden Merkmale/Attribute. Die maximale Fehlklassifikationsrate soll 5% sein. Wie hoch ist die tatsächlich erreichte Fehlklassifikationsrate?

f) Probieren Sie in einem zweiten Versuch das zweite und dritte Merkmal/Attribut zur Klassifikation aus.

g) Optional: Implementieren Sie die Hebb-Regel als Methode `HebbTrain`, indem Sie Ihre Methode `DeltaTrain` kopieren und entsprechend anpassen. Probieren Sie die Hebb-Regel auf den UND-Daten aus. Wenn das nicht besonders gut funktioniert, ist das nicht verwunderlich – die Hebb-Regel funktioniert ja nicht besonders gut.