NNW-Übung 4

1 Verwendung von Matrizen

a) Machen Sie ein Backup Ihrer Klasse `SLN` (z.B. als Datei `SLN_Blatt2.py`).

b) Ändern Sie die Klasse `SLN` durch Verwendung von Matrix-Operationen so, dass `SLN.neuron` *keine* Schleife mehr notwendig braucht und `SLN.DeltaTrain` nur noch *eine* Schleife.

Tipps:

- `SLN.neuron` sollte auf alle Daten `X` (also die Matrix `X`) angewendet werden können und entsprechend viele Ergebnisse in einer Ergebnis-Matrix zurückliefern.
- Zu `SLN.DeltaTrain`: Die bei Batch-Learning in den Lernregeln auftretende Summe über alle Trainingsdaten $n = 1..N$ ergibt sich also auch durch eine Matrix-Multiplikation. Am einfachsten finden Sie diese Lösung, wenn Sie die Regel für die Multiplizierbarkeit von zwei Matrizen und die dabei auftretenden Dimensionen betrachten: $C = A \cdot B$, Dimensionen: $(c \times d) = (c \times n) \cdot (n \times d)$. Weiterer Tipp: Eine Matrix müssen Sie transponieren.

  Falls Sie einen Pseudo-Eingabevektor $\vec{x}_0$ (nützliche Funktion: `np.ones`) für die Verwendung zusammen mit den Bias-Gewichten anlegen, darauf achten, dass er überhaupt transponiert werden kann – die Dimension muss mindestens 2 sein.[1] Wenn die Dimension nur 1 ist, können Sie das mit `np.newaxis` oder `reshape` ändern.

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 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 (oder nur `False` und `True`) 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?

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

2 Mehrere Klassen

Ziel dieser Aufgabe ist, mehrere Klassen mittels eines Single-Layer Networks zu trennen. Als Beispieldaten sollen wieder die Iris-Daten dienen, wobei jetzt alle drei Iris Arten verwendet werden.

a) Für die Konvertierung der mit der Klassennummer kodierten Zieldaten `T` (also z.B. Klasse 2) in die one-hot-Codierung[2] für ein Netz (also im Beispiel $[0; 1; 0]$) können Sie die folgende Funktion zu Ihrer Klasse `SLN` hinzufügen:

[1] $\vec{x}_0$ ist nicht unbedingt nötig, es geht auch ohne.

[2] siehe 02_Aufgabenstellung, Folie 21 „one hot vectors“. Mit der Foliennummer ist die Nummer der Folie *oben links* gemeint, nicht die Seite im PDF!

```
def onehot(self,T):
    e=np.identity(self._W.shape[0])
    return e[:,T.astype(int)]
```

In der Funktion wird eine Einheitsmatrix (`np.identity(c)`) indiziert mit dem colon-Operator (`:`) und dem ursprünglichen Vektor der Zieldaten, der die Nummer der Klasse (Iris-Art) enthält.

Probieren Sie die Funktion zuerst für einen kleinen Zieldatenvektor `T=np.array([0,2,1,2])` aus und vollziehen Sie sie nach.

b) Wie in der Vorlesung besprochen, soll die Zuordnung zu der Klasse erfolgen, deren zugehöriges Output-Neuron den größten Wert geliefert hat (siehe 03_Grundkonzepte, Folie 28[3]).

Beispiel: `np.array([[0.2],[0.6],[0.5]])` wird der Klasse 1 (Zählung ab 0) zugeordnet. Nützliche NumPy-Methode hierfür: `argmax`. Probieren Sie sie aus. Ist das Ergebnis schon one-hot kodiert?

Ändern Sie nun die Klasse `SLN` wie folgt ab (wenn Ihr Code teilweise schon dasselbe tut, können Sie ihn natürlich auch beibehalten) und vollziehen Sie den Code nach:

```
def __init__(self,dIn,cOut): # Konstruktor
    np.random.seed(42)
    self._b=np.zeros((cOut,1))
    self._W=np.random.randn(cOut,dIn)/np.sqrt(dIn)
    if cOut==1:
        self.neuron=self.threshold
    else:
        self.neuron=self.thresholdMult
def netsum(self,X):
     return self._W.dot(X)+self._b
def threshold(self,X):
    return self.netsum(X)>=0
```

Was machen die Zeilen nach dem `if` und dem `else`? Wird hier eine Methode aufgerufen?

Implementieren Sie in der Klasse `SLN` eine neue Methode `thresholdMult(self,X)`, die die Netzwerksumme mit Hilfe von `netsum` berechnet und ein one-hot-kodiertes Ergebnis liefert.

c) Ihre Methode `SLN.DeltaTrain` sollte jetzt *ohne Änderungen* auch mit mehr als zwei Klassen funktionieren.

Trainieren Sie ein Single-Layer Network für die Unterscheidung aller drei Iris-Arten.

Verwenden Sie dazu die Merkmale/Attribute 3 und 4!

Übernehmen Sie den Plot in die Auswertung und geben Sie zudem an: den von Ihnen gewählten Wert für η, die erreichte Fehlerrate und nach wievielen Iterationsschritten diese Fehlerrate *erstmalig* erreicht wurde.

Anmerkung zur Interpretation des Ergebnisses: Die Daten sind *nicht* linear separabel!

[3] Mit der Foliennummer ist die Nummer der Folie *oben links* gemeint, nicht die Seite im PDF!