NNW-Übung 2

Bemerkungen zu MATLAB

Denken Sie daran, dass MATLAB ein Interpreter ist. Sie können Befehle also auch auf der Kommandozeile von MATLAB testen. Hilfe zu einzelnen Funktionen können Sie durch den Befehl `doc <Funktionsname>` abrufen oder einfach durch Positionieren des Cursors auf dem Befehl und drücken von `F1`.

Weil MATLAB ein Interpreter ist, sind besonders Schleifenkonstruktionen langsam und sollten – soweit möglich – vermieden werden. Sie können auch eigene Funktionen definieren (siehe die Kurzeinführung, Abschnitt 5 "Funktionen").

Im Editor können Sie mit doppelten Kommentarzeilen (`%%`) Blöcke („cells") definieren, die einzeln abgearbeitet werden können (Start über gelb/weiße Icons). Markierte Zeilen können durch Drücken von `F9` ausgeführt werden.

1 Datenstruktur für einschichtiges Netz und Propagierung

Realisieren Sie alle im Folgenden beschriebenen Funktionen, die mit `Sln` beginnen, als eigene `.m` Dateien mit dem Namen der Funktion als Dateinamen. Damit können Sie die Funktionen in mehreren MATLAB-Programmen verwenden.

a) Erzeugen Sie die Trainingsdaten für das UND-Gatter. Erzeugen Sie eine Matrix X mit den Eingabe-Vektoren $\vec{x}^{(n)}$ als Spalten und eine Matrix T mit den erwarteten Ausgabe-Werten $\vec{t}^{(n)}$ als Spalten. Überlegen Sie genau, welche Dimensionen die beiden Matrizen haben müssen.

b) Als Datenstruktur für das neuronale Netz bietet es sich an, eine Struktur zu verwenden. Die Elemente einer Struktur werden (wie in Java) mit `struktur.element` angesprochen. Sie brauchen aber *nicht* deklariert zu werden, sondern werden bei der ersten Verwendung automatisch erzeugt.

   Schreiben Sie eine Funktion `sln=SlnInit(dIn,cOut)`, die eine Struktur für ein Single-Layer Network initialisiert und zurückgibt. `dIn` steht dabei für die Anzahl der Eingabe-Werte (in der Vorlesung mit d bezeichnet), `cOut` für die Anzahl der Ausgabe-Werte, die bei einem Single-Layer Network der Anzahl Neuronen entspricht (in der Vorlesung mit c bezeichnet).

   Die Struktur soll folgende Elemente besitzen, die entsprechend zu initialisieren sind:

   - `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)
   - `W1`: Matrix der Gewichte (ohne Bias!)
   - `b1`: Bias-Vektor

   Initialisieren Sie `W1` und `b1` mit zufälligen, normalverteilten Werten mit Hilfe der MATLAB-Funktion `randn`. Initialisieren Sie den Zufallsgenerator durch den Aufruf `randn('state', 17)`, um reproduzierbare Ergebnisse zu erhalten. Teilen Sie die Werte durch die Wurzel aus `dIn+1`.

c) Schreiben Sie Funktionen, die das neuronale Netz auf die Daten `X` anwenden, also zunächst die (Forward-)Propagierung (Berechnung von *net*) durchführen, und dann eine Aktivierungsfunktion anwenden: Eine Funktion `Y=SlnFwdLinear(sln, X)` für die lineare Aktivierungsfunktion und eine Funktion `Y=SlnFwdThreshold(sln, X)` für die McCulloch-Pitts-Funktion mit der Stufe/Schwelle (threshold) bei 0 (die Verschiebung der Schwelle übernimmt der Bias), die praktischerweise `SlnFwdLinear` aufrufen kann.

Als Parameter `X` soll dabei das X aus der ersten Teilaufgabe eingesetzt werden können und die Ausgabe `Y` wird später mit T aus der ersten Teilaufgabe verglichen.

Überprüfen Sie, ob Ihre Funktionen auch funktionieren (keine Fehlermeldung produzieren), wenn Sie zwei (oder mehr) Neuronen verwenden.

Eine nützliche MATLAB-Funktion für diese Teilaufgabe ist `repmat`.

d) Schreiben Sie eine Funktion `SlnErrorRate(Y, T)`, die die Fehlklassifikationsrate ausrechnet. Die Fehlklassifikationsrate ist die Anzahl der inkorrekt klassifizierten Trainingsmuster dividiert durch die Gesamtanzahl der Trainingsmuster.