HS Osnabrück
IuI – Biermann

Praktikum zu Wissensbasierten Methoden

Blatt 5

Vorgesehen für die Praktikumssitzungen am 31. Mai und am 03. Juni 2016

1. Aufgabe: Sie sollen ein vorwärts gerichtetes neuronales Netz vollständig implementieren. Dieses soll in Form einer oder mehrerer *Matlab*-Routinen erfolgen, wobei die Funktionen aus der *Neural Network toolbox* **nicht** verwendet werden dürfen. Zur Vereinfachung werden dabei folgende Festlegungen getroffen:

   - Das Netz besitzt genau eine innere Neuronenschicht.
   - Als Aktivierungsfunktion wird ausschließlich nur die Tangenshyperbolicus-Sigmoidfunktion (`tansig`) genommen[1].

   Unter dieser Voraussetzung kann das Netz durch zwei Matrizen dargestellt werden:

   (a) durch die Matrix W_1 der Übergangsgewichte der Verbindung von der 0-ten zur ersten Schicht sowie der Schwellwerte der ersten Schicht,

   (b) durch die Matrix W_2 der Übergangsgewichte der Verbindung von der ersten zur zweiten Schicht sowie der Schwellwerte der zweiten Schicht.

   Mit den Bezeichnungen der Vorlesung hat man (mit $m = 2$)

$$W_1 = \begin{pmatrix} w_{1,1,1} & \dots & w_{1,1,q_0} \\ \vdots & \ddots & \vdots \\ w_{1,q_1-1,1} & \dots & w_{1,q_1-1,q_0} \end{pmatrix} \qquad W_2 = \begin{pmatrix} w_{2,1,1} & \dots & w_{2,1,q_1} \\ \vdots & \ddots & \vdots \\ w_{2,q_2-1,1} & \dots & w_{2,q_2-1,q_1} \end{pmatrix} \tag{0.1}$$

   Die Neuronenzahlen der Schichten des Netzes werden damit durch die Dimensionierungen der Matrizen W_1 und W_2 festgelegt; der Aufruf der Auswertungsfunktion[2] des trainierten Neuronalen Netzes könnte damit folgendermaßen aussehen:

```
a=werteaus(W1,W2,e)
```

   Richten Sie sich bei der Auswertungsfunktion und bei der Funktion zur Berechnung des Gradienten der Fehlerfunktion nach den Formeln der Vorlesung. Bedienen Sie sich dabei insbesondere der Matrizen- und Vektorschreibweise. Möglicherweise könnten dabei auch zwei Besonderheiten *Matlabs* hilfreich sein:

   - Mit der Funktion `diag` können Diagonalmatrizen erzeugt werden.
   - Vorhanden sind die komponentenweisen Operatoren „`.*  ./  .^`“; werden diese auf zwei Vektoren oder Matrizen gleicher Dimensionierung angewandt, so werden die üblichen Operatoren (“`*  /  ^`“) komponentenweise ausgeführt, das Ergebnis ist dann wieder ein Vektor bzw. eine Matrix derselben Dimension wie die Operanden.

[1] auch wenn bei einem gegebenen Lernproblem eine andere Aktivierungsfunktion angezeigt wäre
[2] entspricht der *Matlab*-NNT-Funktion `sim`

- Alle Funktionen einer reellen Veränderlichen können auf Vektoren angewandt werden; die Auswertung der Funktion erfolgt dann komponentenweise. Für einen Vektor $\vec{x} = (x_1, \ldots, x_n)^t$ hat man etwa

$$\mathrm{f}(\vec{x}) \quad = \quad \begin{pmatrix} \mathrm{f}(x_1) \\ \vdots \\ \mathrm{f}(x_n) \end{pmatrix}$$

  Bei selbst erstellten Funktionen sollte man sicherstellen, daß auch diese die genannte Eigenschaft besitzen; dazu müssen im Code der Funktionen die eben genannten komponentenweisen Operatoren verwendet werden.

Mit dem berechneten Gradienten der Fehlerfunktion können Sie das (adaptive) Gradientenverfahren anwenden; passen Sie dazu entsprechend die Lösung einer früheren Aufgabe an, in der das Gradientenverfahren auf eine Funktion zweier Veränderlicher angewandt wurde. Stellen Sie den Gradienten in der Anordnung der beiden Matrizen (0.1) dar:

$$\begin{pmatrix} \frac{\partial \mathrm{F}}{\partial w_{1,1,1}} & \cdots & \frac{\partial \mathrm{F}}{\partial w_{1,1,q_0}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathrm{F}}{\partial w_{1,q_1-1,1}} & \cdots & \frac{\partial \mathrm{F}}{\partial w_{1,q_1-1,q_0}} \end{pmatrix} \qquad \begin{pmatrix} \frac{\partial \mathrm{F}}{\partial w_{2,1,1}} & \cdots & \frac{\partial \mathrm{F}}{\partial w_{2,1,q_1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathrm{F}}{\partial w_{2,q_2-1,1}} & \cdots & \frac{\partial \mathrm{F}}{\partial w_{2,q_2-1,q_1}} \end{pmatrix} \tag{0.2}$$

Erforderlich ist dann nur noch die zufällige Initialisierung der Gewichte.

Zum Test Ihrer Funktionen legen Sie vier Netze an und führen Sie damit folgende Lernaufgaben durch:

(a) zwei binäre Eingänge e_1 und e_2 und ein Ausgang mit $e_1 \oplus e_1$ (xor),

(b) drei binäre Eingänge e_1, e_2 und e_3 und ein Ausgang mit $e_1 \oplus e_1 \oplus e_3$ (zweifaches xor),

(c) zwei binäre Eingänge e_1 und e_2 und vier Ausgänge mit $e_1 \vee e_2$, $e_1 \wedge e_2$, $\overline{e_1}$ und $\overline{e_2}$ (Negationen),

(d) eine von Ihnen selbst gewählte Lernaufgabe mit mindestens vier Eingangswerten; denken Sie daran, daß möglicherweise eine Normalisierung erforderlich wird.

Empfehlung: Erstellen Sie zunächst die Funktion zur Auswertung des trainierten Netzes; testen Sie diese vorab, indem Sie die Gewichte des Netzen "per Hand" setzen.

2. Aufgabe: Ebenso wie in einer vorangegangenen Aufgabe soll ein vorwärts gerichtetes neuronales Netz vollständig implementiert werden. Auch dieses soll wieder in Form einer oder mehrerer *Matlab*-Routinen erfolgen, wobei die Funktionen aus der *Neural Network toolbox* **nicht** verwendet werden dürfen. Hier soll jedoch eine Netz **mit zwei inneren Schichten** erstellt werden. Dieses verläuft in völlig entsprechender Weise wie bei einem Netz mit nur einer inneren Schicht[3]:

   - Bei der Behandlung der Eingabeschicht und der ersten inneren Schicht ergibt sich keine Veränderung.
   - Ebenso bleibt bei der Ausgabeschicht alles beim Alten; diese Schicht erhält nur jetzt die Nummer $m = 3$; die Gewichtsmatrix mit den Gewichten der zur Ausgabeschicht führenden Verbindungen ist dann W_3.

[3] In der Vorlesung wurde zwar ebenfalls nur der Fall einer inneren Schicht behandelt. Dieses ist jedoch sofort verallgemeinerbar; ein entsprechender Hinweis befindet sich im Vorlesungsumdruck.

- Die zweite innere Schicht wird in gleicherweise wie die erste innere Schicht behandelt; zu den zu ihr führen Verbindungen gehört die Gewichtsmatrix W_2.

Dieses neuronale Netz wird nun durch die drei Gewichtsmatrizen W_1, W_2 und W_3, d. h. durch die Matrizen

$$\begin{pmatrix} w_{1,1,1} & \dots & w_{1,1,q_0} \\ \vdots & \ddots & \vdots \\ w_{1,q_1-1,1} & \dots & w_{1,q_1-1,q_0} \end{pmatrix}, \begin{pmatrix} w_{2,1,1} & \dots & w_{2,1,q_1} \\ \vdots & \ddots & \vdots \\ w_{2,q_2-1,1} & \dots & w_{2,q_2-1,q_1} \end{pmatrix}, \begin{pmatrix} w_{3,1,1} & \dots & w_{3,1,q_2} \\ \vdots & \ddots & \vdots \\ w_{3,q_3-1,1} & \dots & w_{3,q_3-1,q_1} \end{pmatrix}$$

dargestellt werden. Bei der Auswertungsfunktion[4] des trainierten neuronalen Netzes muß die zusätzliche Matrix berücksichtigt werden; ein Aufruf der Auswertungsfunktion könnte folgendermaßen aussehen:

```
a=werteaus(W1,W2,W3,e)
```

Zum Lernen soll wieder das adaptive Gradientenverfahren verwendet werden. Der Gradienten ist nun gemäß der drei Gewichtsmatrizen anzuordnen:

$$\begin{pmatrix} \frac{\partial F}{\partial w_{1,1,1}} & \cdots \frac{\partial F}{\partial w_{1,1,q_0}} \\ \vdots & \ddots \vdots \\ \frac{\partial F}{\partial w_{1,q_1-1,1}} & \cdots \frac{\partial F}{\partial w_{1,q_1-1,q_0}} \end{pmatrix}, \begin{pmatrix} \frac{\partial F}{\partial w_{2,1,1}} & \cdots \frac{\partial F}{\partial w_{2,1,q_1}} \\ \vdots & \ddots \vdots \\ \frac{\partial F}{\partial w_{2,q_2-1,1}} & \cdots \frac{\partial F}{\partial w_{2,q_2-1,q_1}} \end{pmatrix}, \begin{pmatrix} \frac{\partial F}{\partial w_{3,1,1}} & \cdots \frac{\partial F}{\partial w_{3,1,q_2}} \\ \vdots & \ddots \vdots \\ \frac{\partial F}{\partial w_{3,q_3-1,1}} & \cdots \frac{\partial F}{\partial w_{3,q_3-1,q_2}} \end{pmatrix}$$

Als Aktivierungsfunktion soll auch hier ausschließlich nur die Tangenshyperbolicus-Sigmoidfunktion (`tansig`) genommen werden.

Testen Sie Ihr Lernverfahren für vorwärts gerichtete Netze mit zwei inneren Schichten anhand der folgenden Lernaufgabe: Für Punkte $\vec{x}_i = (x_i, y_i)^t \in \mathbb{R}^2$ mit $-5 < x_i, y_i < 5$ soll die Nummer des zugehörigen Quadranten gelernt werden. Dieses Netz besitzt dann zwei Eingabe- und vier (binäre) Ausgabewerte. Für die Ausgabewerte $\vec{e} = (a_1, e_2, e_3, e_4)^t$ gilt dann:

$$e_j = 1 \quad \Leftrightarrow \quad \begin{array}{l}\text{Der betreffende Eingabewert lag} \\ \text{im } j\text{-ten Quadranten.}\end{array}$$

Erzeugen Sie die Lerndatensätze zufällig mit Hilfe der `rand`-Funktion; Sie können dazu den folgenden *Matlab*-Code verwenden:

```
n=100;
Mx=10;
E=Mx*rand([2,n])-Mx/2*ones(2,n);  % Mittelwert Null
k1=E(1,:)>=0;
k2=E(2,:)>=0;

A=[and(k1,k2);            ...    % 1  Quadrant
   and(not(k1),k2);       ...    % 2. Quadrant
   and(not(k1),not(k2)); ...    % 3. Quadrant
   and(k1,not(k2))];            % 4. Quadrant
```

[4] entspricht der *Matlab*-NNT-Funktion `sim`

```
plot(E(1,A(1,:)),E(2,A(1,:)),'ok'); hold on
plot(E(1,A(2,:)),E(2,A(2,:)),'*k');
plot(E(1,A(3,:)),E(2,A(3,:)),'sk');
plot(E(1,A(4,:)),E(2,A(4,:)),'pk');
```