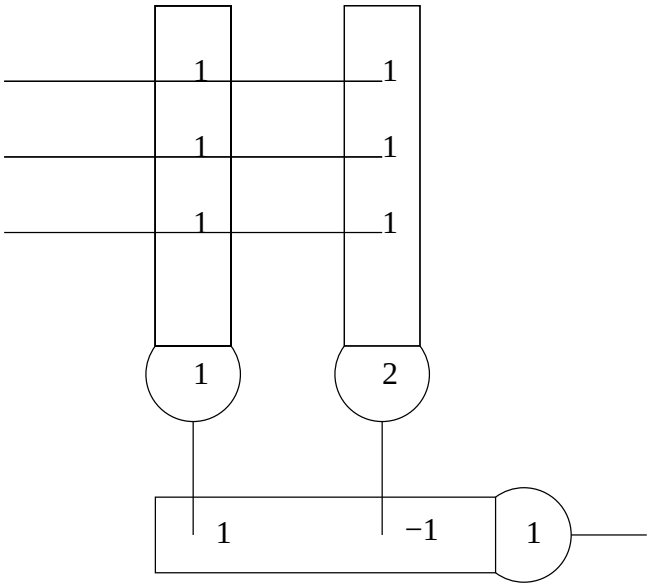
Einführung in die Neuroinformatik

Lösungen zum 5. Aufgabenblatt

7. Aufgabe : Summe $\in \{1\}$

Man sieht leicht ein, dass ein einzelnes Perzeptron mit Gewichten c_1, c_2, c_3 und Schwelle θ das Problem nicht lösen kann. Es gilt nämlich:
$c_1 \geq \theta$ und $c_2 \geq \theta$ und $c_3 \geq \theta$, denn für eine Eins soll das Schwellenneuron Ausgabe $y = 1$ liefern.
Hieraus folgt sofort - man addiere einfach die drei Ungleichungen
$c_1 + c_2 + c_3 \geq 3\theta$ und damit wäre auch die Eingabe mit 2 oder 3 Einsen überschwellig und die Ausgabe $y = 1$.
Wählt man nun eine 2-Schicht Architektur mir 2 Neuronen in der Zwischenschicht und einem Ausgabeneuron, sowie die Gewichte und Schwellwerte wie in folgender Abbildung, so ist hierdurch die Funktion F realisiert:



Das linke Neuron in der Zwischenschicht (mit Schwelle 1) ist aktiv (d.h. Ausgabe $y = 1$) für Eingaben mit mindestens einer Eins, das rechte Neuron der Zwischenschicht mit Schwelle 2 ist aktiv (d.h. Ausgabe $y = 1$) falls mindestens 2 Einsen in der Eingabe vorliegen. Das Ausgabeneuron ist aktiv genau dann wenn das erste Neuron aktiv ist, also genau 1 Eins in der Eingabe vorliegt.

8. Aufgabe : MLP und Error Backpropagation

Das Rahmenprogramm für das Backpropagation Lernverfahren

```
clear all; close all;

load -ascii einszwei.dat
Werte  = einszwei(:,1:3);
Lehrer = einszwei(:,4);

net = mlp(3, 2, 1, 'tanh','gtanh', 0.5);

y_before_training = mlpfwd(net, Werte)

steps = 10000;
[net_for, errors_for] = mlpbackprop(net, Werte, Lehrer, steps, 0.01);

y_after_training_for = mlpfwd(net_for, Werte)

plot(1:steps,errors_for,'go');
title(['Online Error Backpropagation']);
axis([0 steps 0 4]);
xlabel('Zahl der Lernepochen');
ylabel('Quadratischer Fehler');

print -depsc ../Figures/Aufgabe9-Fehler.eps
```

Das Hauptprogramm zur Anpassung der Netzwerkgewichte. Das matlab-Programm ist nicht optimiert, denn es enthält viele FOR-Schleifen.

```
%Backpropagation Lernregel in For-Schleifen
%Uebergebene Variablen:
%net ... Netzwerk
%input ... zu lernende Daten (Dimensionalitï¿½ muss Groesse der Inputschicht
%          entsprechen
%teacher ... Lehrersignale zu input
%steps ... Anzahl der Lernschritte
%lrate ... Lernrate
%
%Zurueckgegebene Variablen:
%net ... trainiertes Netzwerk
%error ... der Fehler zu allen Lernepochen
function [net,error] = mlpbackprop(net,input,teacher,steps,lrate)

for s=1:steps %Lernschritte
    error(s)=0;
    for m=1:size(input,1) %Jedes Muster

        % Forward step
        y_input = input(m,:);
        x_hidden = net.w1'*y_input'-net.b1';
        optstring = [net.afunction, '(x_hidden)'];
        y_hidden = eval(optstring);
        x_output = (net.w2'*y_hidden-net.b2')';
        optstring = [net.afunction, '(x_output)'];
        y_output = eval(optstring);

        % Deltas der Ausgabeschicht
        for j=1:net.noutput
            optstring = [net.gafunction, '(x_output(j))'];
            fg = eval(optstring);
            deltaj(j)= (teacher(m,j) - y_output(j))*fg;
        end
        % Deltas der Zwischenschicht
        for i=1:net.nhidden
            optstring = [net.gafunction, '(x_hidden(i))'];
            fg = eval(optstring);
            deltai(i)= 0;
            for j=1:net.noutput
                deltai(i) = deltai(i)+ deltaj(j)*net.w2(i,j)*fg;
            end;
        end

        % Update der Ausgabeschicht
        for j=1:net.noutput
            net.b2(1,j) = net.b2(1,j) - lrate * deltaj(j);
            for i=1:net.nhidden
                net.w2(i,j) = net.w2(i,j) + lrate* y_hidden(i) * deltaj(j);
            end
        end
        % Update der Zwischenschicht
```

```
        for i=1:net.nhidden
            net.b1(1,i) = net.b1(1,i) - lrate * deltai(i);
            for k=1:net.ninput
                net.w1(k,i) = net.w1(k,i) + lrate* y_input(k) * deltai(i);
            end
        end
    end
    error(s) = mlpfehler(net,input,teacher);
end
```

Definition eines MLP-Netzes:

```
%Erstellt ein neues Netz
%Uebergebene Variable:
%ninput ... Anzahl der Neuronen in der Eingabeschicht
%nhidden ... Anzahl der Neuronen in der Zwischenschicht
%noutput ... Anzahl der Neuronen in der Ausgabeschicht
%afunction ... Name der Transferfunktion (zb. tanh)
%gafunction ... Name der abgeleiteten Transferfunktion (zb. gtanh)
%a ... Parameter fr den Bereich in dem die Gewichte initialisiert werden
%
%Zurueckgegebene Variablen:
%net ... voll initialisiertes Netz
function net = mlp(ninput,nhidden,noutput,afunction,gafunction,a)
net.type = 'mlp';
net.ninput = ninput;
net.nhidden = nhidden;
net.noutput = noutput;
net.afunction = afunction;
net.gafunction = gafunction;

net.w1 = a*(2*rand(ninput,nhidden)-1);
net.b1 = a*(2*rand(1,nhidden)-1);

net.w2 = a*(2*rand(nhidden,noutput)-1);
net.b2 = a*(2*rand(1,noutput)-1);
```

Vorwärtsschritt für das MLP zur Outputberechnung:

```
%Berechnung des Outputs fuer ein MLP
function y = mlpfwd(net,input)
% Output for 1. layer
optstring = [net.afunction, '((net.w1''*input''-repmat(net.b1'',1,size(input,1))))'],
erg = eval(optstring);
% Output for 2. layer
optstring = [net.afunction, '((net.w2''*erg-repmat(net.b2'',1,size(input,1)))'')'];
y = eval(optstring);
```

Fehlerberechnung des Netzes:

```
%Berechnung des Fehlers fuer ein MLP
%Uebergebene Variablen:
%net ... Netzwerk
%input ... zu lernende Daten (Dimensionalitï¿½ muss Groesse der Inputschicht
%          entsprechen
%teacher ... Lehrersignale zu input
%
%Zurueckgegebene Variablen:
%error ... der Fehler zu allen Lernepochen
function error = mlpfehler(net,input,teacher)
% Output for 1. layer
optstring = [net.afunction, '((net.w1''*input''-repmat(net.b1'',1,size(input,1))))'],
erg = eval(optstring);
% Output for 2. layer
optstring = [net.afunction, '((net.w2''*erg-repmat(net.b2'',1,size(input,1)))'')'];
y = eval(optstring);
% Fehler
error = sum((teacher-y).^2);
```

Ein Fehlerverlauf (andere Verläufe sind möglich, auch mit Fehler > 1, ggf Neuronenzahl erhöhen):