Einführung in die Neuroinformatik

Tim Luchterhand, Paul Nykiel (Gruppe P)

13. Juni 2018

1 Learning Slowdown

1.1

(a)

$$\begin{aligned} w_2 &= 0.6 \\ b_2 &= 0.9 \\ T &= 0 \\ \Rightarrow \frac{\partial E}{\partial b_2}(w,b) &= -2\left(0 - \frac{1}{1+e^{-1.5}}\right)\frac{1}{1+e^{-1.5}}\left(1 - \frac{1}{1+e^{-1.5}}\right) \\ &\approx 0.24 \end{aligned}$$

(b)

$$\begin{aligned} w_2 &= 2 \\ b_2 &= 2 \\ T &= 0 \\ \Rightarrow \frac{\partial E}{\partial b_2}(w,b) &= -2\left(0 - \frac{1}{1+e^{-4}}\right)\frac{1}{1+e^{-4}}\left(1 - \frac{1}{1+e^{-4}}\right) \\ &\approx 0.035 \end{aligned}$$

(c) Der Gradient ist sehr klein, wodurch jede Epoche nur minimalen Lernfortschrittt bringt. Problematisch ist hierbei die Ableitung der Sigmoid-Funktion, da $f'(x) \in \left[0, \frac{1}{4}\right] \; \forall x \in \mathbb{R}$, und verschwindet für betragsmäßig große x.

1.2

(a)

$$\begin{aligned}
\frac{\partial E}{\partial b_1} &= 2(T - y_2) \cdot (-1) \cdot \frac{\partial y_2}{\partial b_1} \\
&= -2(T - y_2) \cdot \frac{\partial}{\partial b_1} f(w_2 \cdot f(w_1 \cdot x + b_1) + b_2) \\
&= -2(T - y_2) \cdot f'(w_2 \cdot f(w_1 \cdot x + b_1) + b_2) \cdot w_2 \cdot f'(w_1 \cdot x + b_1) \\
&= -2(T - y_2) \cdot f'(u_2) \cdot w_2 \cdot f'(u_1)
\end{aligned}$$

Pro weiterer Netzwerkschicht kommen weitere f'-Terme hinzu die wiederrum $\in \left[0, \frac{1}{4}\right]$ sind. Dadurch wird der Gradient tendentiell noch kleiner und das Lernen verlangsamt sich zusätzlich zu Neuronen näher zur Eingangsschicht.

(b) Wie in (a) bereits erläutert, verschlimmern weitere Schichten das Problem.

(c) In jeder Epoche werden die Gewichte und der Bias nur marginal angepasst wodurch der Gradientenabstieg sehr langsam erfolgt.

1.3

(a)

$$\begin{aligned}
x &= 1 \\
T &= 0 \\
w_1 &= 100 \\
b_1 &= -100 \\
w_2 &= 100 \\
b_2 &= -50 \\
u_1 &= 100 - 100 = 0 \\
y_1 &= f(u_1) = \frac{1}{2} \\
u_2 &= 100 \cdot y_1 - 50 = 0 \\
y_2 &= f(u_2) = \frac{1}{2} \\
\frac{\partial E}{\partial b_2} &= 2 \cdot y_2 \cdot f'(u_2)) = \frac{1}{4} \\
\frac{\partial E}{\partial b_1} &= \frac{\partial E}{\partial b_2} \cdot 25 = \frac{25}{4}
\end{aligned}$$

(b) Der Gradient der aktuellen Schicht setzt sich aus dem Gradienten der Nachfolgenden Schicht multipliziert mit einem Faktor $\gg 1$. Dadurch werden die Gradienten in den Schichten nah am Eingang extrem groß.

(c) Ein zu großer Gradient kann dazu führen, dass Minima übersprungen werden wodurch der Lernprozess oszilliert und sich das Lernen in die Länge zieht.

1.4

Bei der Cross-Corelation-Funktion tritt immer ein f'-Term weniger auf als bei der Sigmoid-Funktion. Das Problem der zu kleinen Gradienten fällt deswegen weniger stark ins Gewicht.

2 Flat vs. Deep Networks

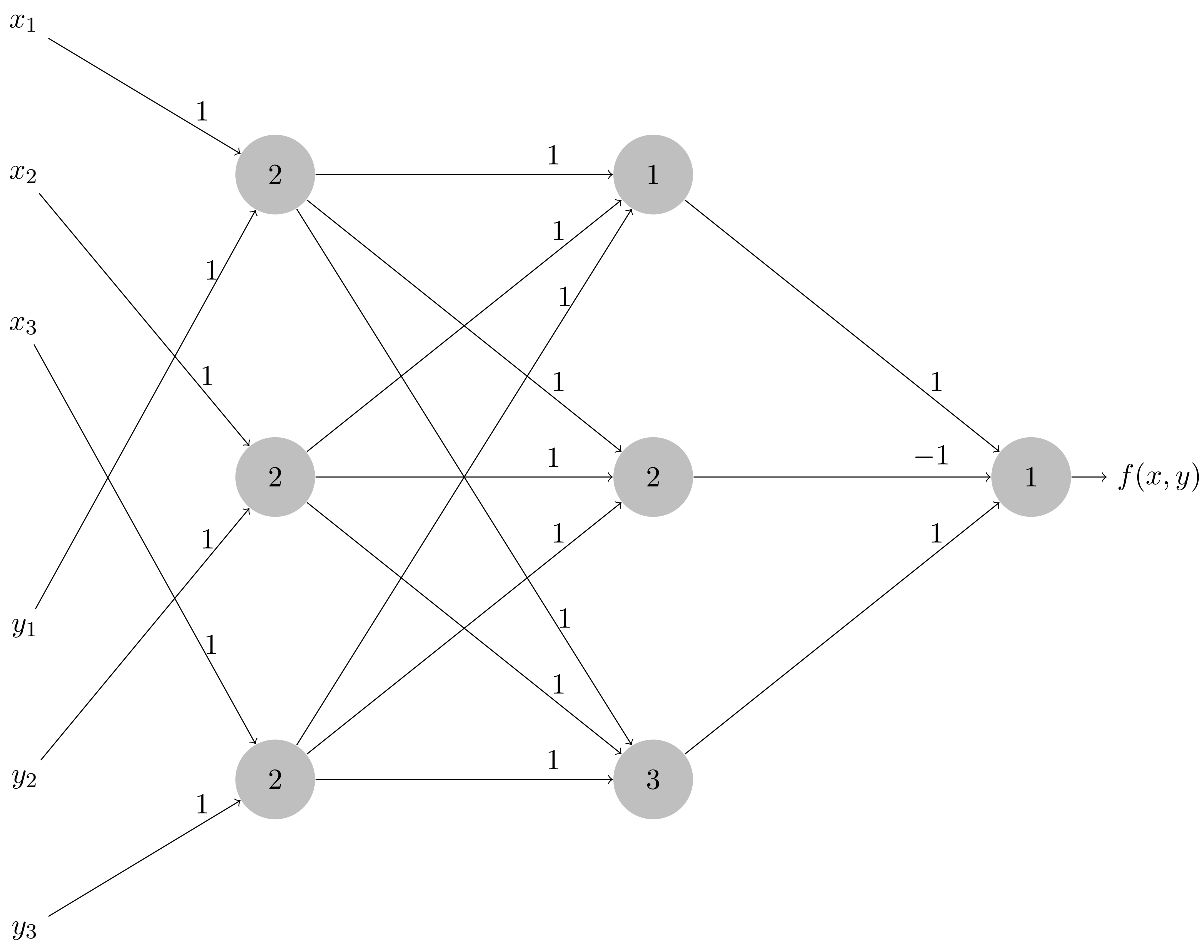
2.1

(a)

$$
\begin{aligned}
x &= (0,0) \\
y &= (1,1) \\
y_1^{(1)} &= 0 \\
y_2^{(1)} &= 0 \\
y_1^{(2)} &= 0 \\
y_2^{(2)} &= 0 \\
f(x,y) &= 0 \\
< x, y >_2 &= \begin{pmatrix} 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} \mod 2 = 0
\end{aligned}
$$

(b) Die Neuronen in der ersten Schicht fungieren als AND-Gatter und berechnen das Skalarprodukt. Die Neuronen in der zweiten Schicht bilden zusammen ein XOR-Gatter und führen die Modulo-Operation durch.

(c)



2.2

(a) Für das Gradientenabstiegsverfahren sind differenzierbare Funktionen f notwendig, die Sprunfunktion ist nicht differentierbar.

(b)

```
% Aufgabe b
% Generate input data
n = 4;
nInput = 2 * n;
combinations = de2bi(0:2^nInput-1)';

% Each column of both vectors represents an input sequence (this is contrary to the usual convention but Matlabs implementation of neural networks expects the input in this format)
```

```
X = combinations(1:n, :);
Y = combinations(n+1:end, :);

T = zeros(1,size(X,2));
for c = 1:size(X,2)
    x = X(:,c);
    y = Y(:,c);
    currentT = mod(transpose(x) * y, 2);
    T(:,c) = currentT;
end
```

(c)

```
function [bestError] = trainNetworks(combinations, targets, hidden)
%trainNetworks trains the network multiple times and returns the best result
%   Arguments:
%       - combinations: input data organized as features x observations (each column defines a data point). This is the required format for Matlabs train() function
%       - targets: label for each data point (as defined by the binary dot product function)
%       - hidden: number of hidden neurons to use per layer (as vector if multiple hidden layers should be used)
%
%   Return value:
%       - bestError: the value for the best error (MSE) over all trained networks
%
    assert(isrow(targets), 'targets should be a row vector');
    assert(isscalar(hidden) || isrow(hidden), 'hidden should either be a scalar value (number of hidden neurons in the flat hierarchy) or a vector defining the number of hidden neurons per layer');

    rng(1337, 'combRecursive');

    minError = 100000;
    for c = 1:20
        net = fitnet(hidden);
        net.divideParam.trainRatio = 1;
        net.divideParam.valRatio   = 0;
```

```
        net.divideParam.testRatio  = 0;
        net.trainParam.epochs = 300;
        net = train(net, combinations, targets);

        error = perform(net, net(combinations), targets);
        if error < minError
            minError = error;
        end
    end

    bestError = minError;
end
```

(d)

```

% Aufgabe c
fhidden = 1:2^n+5;
dhidden = 1:n+5;

ferrors = zeros(size(fhidden));
derrors = zeros(size(dhidden));

parfor h = fhidden
    ferrors(h) = trainNetworks(combinations, T, h);
end

parfor h = dhidden
   derrors(h) = trainNetworks(combinations, T, [h h]);
end

figure();
subplot(2,1,1);
plot(fhidden, ferrors);
title('Fehlerfunktion Flat-Network');
xlabel('Anzahl der Neuronen');
ylabel('Minimaler Fehler');

subplot(2,1,2);
plot(derrors);
title('Fehlerfunktion Deep-Network');
xlabel('Anzahl der Neuronen');
ylabel('Minimaler Fehler');

```

```
print('b07a02.eps', '-depsc');
```

(e)

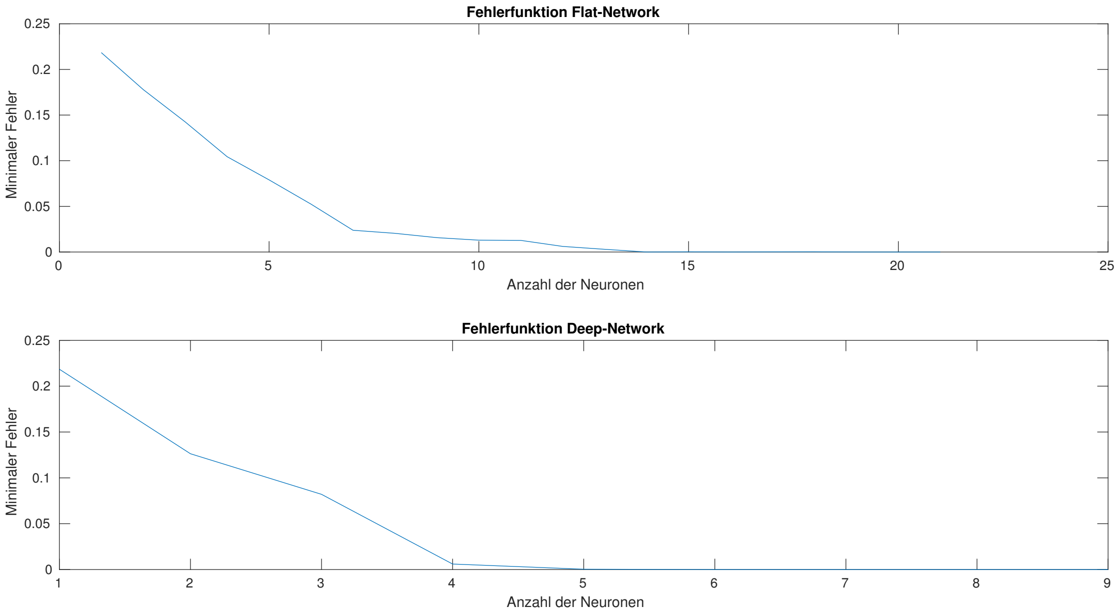


Abbildung 1: Verlauf der Fehlerfunktion