Einführung in die Neuroinformatik

Tim Luchterhand, Paul Nykiel (Gruppe P)

4. Juli 2018

1 Gewichtsinitialisierung

1.1

(a)

$$\begin{aligned} u_i^{(1)} &= \sum_{k=0}^{m} x_k \cdot w_{ki}^{(1)} + b_i^{(1)} \\ &= \sum_{k=1}^{\frac{m}{2}} w_{ki}^{(1)} \\ \Rightarrow N_u &= \sum_{k=1}^{\frac{m}{2}} N(0,1) \\ &= N\left(0, \frac{m}{2}\right) \end{aligned}$$

(b) Die Verteilungsdichtefunktion von u wird breiter, wodurch die Wahrscheinlickeit, dass u betragsmäßig große Werte annimmt, größer wird.

Bei großen dendritischen Potentialen befindet sich die Transferfunktion im Sättigungsbereich, was dazu führt, dass deren Ableitung sehr klein wird und sich der Lernprozess verlangsamt.

(c)

$$\begin{aligned} N_u &= \sum_{k=1}^{\frac{m}{2}} N\left(0, \frac{1}{\sqrt{n_{\text{in}}}}\right) \\ &= \sum_{k=1}^{\frac{m}{2}} N\left(0, \frac{1}{\sqrt{m}}\right) \\ &= N\left(0, \frac{\sqrt{m}}{2}\right) \end{aligned}$$

(d) Durch eine beschränkte Normalverteilung wird garantiert, dass die Gewichte in einem kompakten Intervall liegen. Dadurch wird es unwahrscheinlicher, dass das dendritische Potential im Sättigungsbereich der Transferfunktion liegt.

1.2

Die Ergebnisse aus A gehören zur Strategie 1. In A sieht man, dass das axonale Potential fasst ausschließlich bei -1 und 1 liegt (Sättigung der Transferfunktion). Passend dazu erkennt man, dass der Gradient fast immer 0 ist.

Mit Strategie 2 ist deutlich zu erkennen, dass das axionale Potential breit verteilt ist und dementsprechend der Gradient verschieden Werte annimmt.

Dieses Verhalten entspricht den Feststellungen aus Aufgabe 1.1.

1.3

In der vorherigen Aufgabe haben wir die Verteilung anhand der Anzahl der Eingangsneuronen gestaucht, um die Verteilung des dendritischen Potentials schmal zu halten. Wie aus der Gleichung für den Gradienten zu entnehmen, sorgen große Gewichte w für ein großen Gradienten. Wie in den vorherigen Blättern bereits diskutiert, kann dadurch das Training beeinträchtigt werden.

Um dem entgegenzuwirken, müssen die Anzahl der „Eingänge“ in der Rückrichtung beachtet bei der Verteilung der Gewichte beachtet werden, weshalb man diese wie in Gleichung 1 verteilt. Die Normalverteilung wird anhand des Durchschnitts der Anzahl der Eingangs- und Ausgangsneuronen gestaucht.

2 Regularisierung

2.1

(a)

$$\begin{aligned}
\nabla E(W) &= \nabla \left(E_0(W) + \frac{\lambda}{2} \sum_{l=1}^{L} \sum_i \sum_j \left(w_{ij}^{(l)} \right)^2 \right) \\
&= \nabla E_0(w_{ij}^{(l)}) + \lambda w_{ij}^{(l)} \\
\Rightarrow w(t+1) &= w(t) - \eta \cdot \nabla E(w(t)) \\
&= w(t) - \eta \cdot (\nabla E_0(w) + \lambda w(t)) \\
&= (1 - \eta \cdot \lambda) w(t) - \eta \nabla E_0(w(t))
\end{aligned}$$

(b)

$$\begin{aligned}
w(t+1) &= 0.6 \cdot w(t) \\
\Rightarrow w(t) &= 0.6^t \cdot w(0) \\
&= 2 \cdot 0.6^t \\
\Rightarrow w(10) &= 0.012
\end{aligned}$$

Das Gewicht nimmt exponentiell ab.

$$\lim_{t \to \infty} w(t) = \lim_{t \to \infty} 2 \cdot 0.6^t = 2 \cdot 0 = 0$$

Beweis mit z-Transformation und Endwertsatz (oBdA werden konstante Faktoren vernachlässigt):

$$\begin{aligned}
e(t) &:= \nabla E_0(w(t)) \not\equiv 0 \\
w(t+1) &= w(t) - e(t) \circ\!\!-\!\!\bullet\, W(z)z = W(z) - E(z) \\
\Rightarrow W(z) &= \frac{E(z)}{1-z} \\
\lim_{t \to \infty} w(t) &= \lim_{z \to 1} (z-1) \cdot W(z) \\
&= \lim_{z \to 1} (z-1) \frac{E(z)}{1-z} \\
&= \lim_{z \to 1} -E(z) \\
&= -E(1) \\
&= -\sum_{t=0}^{\infty} e(t) \neq 0
\end{aligned}$$

(c) Die Gewichte werden in jedem Lernschritt mit einem Faktor $0 < 1 - \eta \cdot \lambda < 1$ skaliert. Dadurch werden die Gewichte pro Lernschritt betragsmäßig kleiner. Folglich befinden sich die dendritischen Potentiale tendentiell näher um die 0 und der Gradient verschwindet nicht.

2.2

$$b(t+1) = b(t) + \eta \cdot (T^{\mu} - f(w \cdot x^{\mu} + b)) \cdot f'(w \cdot x^{\mu} + b)$$

Die Adaption des Bias wird nicht direkt vom Eingang beeinflusst, wodurch Rauschen weniger ein Problem ist.

2.3

Die Fehlerfunktion wird quadratisch verstärkt. Dadurch wird $E(W)$ für große W vergrößert und für kleine W kaum verändert. Das führt dazu, dass das Minimum der Fehlerfunktion in Richtung Ursprung verschoben wird und kleine Gewichte so bevorzugt werden. So wirkt man Overfitting entgegen.

2.4

(a)

```
function [netRegression] = networkRegression(input, target, lambda)
%networkRegression creates and trains a neural network for a regression task
%   Arguments:
%       - input: one-dimensional input values (first data dimension)
%       - target: one-dimensional target values (second data dimension)
%       - lambda: regularization parameter (set to 0 to disable regularization)
%
%   Returns:
%       - netRegression: the trained network object which can be used directly for prediction
%
    assert(isrow(input), 'The input values must be passed as a row-vector');
    assert(isrow(target), 'The target values must be passed as a row-vector');
```

```
13      assert(length(input) == length(target), 'Each input
            value must have an associated target value');
14      assert(isscalar(lambda), 'The regularization parameter
            must be a scalar value');
15
16      rng(1337, 'combRecursive');
17
18      netRegression = fitnet(5);
19      netRegression.divideParam.trainRatio = 1;
20      netRegression.divideParam.valRatio   = 0;
21      netRegression.divideParam.testRatio  = 0;
22      netRegression.trainParam.epochs = 50;
23
24      netRegression.performParam.regularization = lambda;
25
26      netRegression = train(netRegression, input, target);
27  end
```

(b)

```
1   % Initialization
2   data = [
3       1.505648148148148, 9.441774784298829;
4       4.4608333333333325, 7.2628223461156365;
5       2.871018518518518, 6.082556442099741;
6       2.889722222222222, 3.7825510906841497;
7       4.6291666666666655, 3.5707084925274506;
8       6.705277777777777, 4.629921483310946;
9       6.798796296296295, 6.566768095029339;
10      7.322499999999999, 2.9957071546735525;
11      1.187685185185185, 5.991766757175442;
12      2.197685185185185, 7.898350140585734;
13      4.142870370370369, 6.112819670407841
14  ];
15
16  % Train networks and predict values
17  netWithout = networkRegression(data(:,1)', data(:,2)', 0);
18  netWith = networkRegression(data(:,1)', data(:,2)', 0.5);
19
20  deltaX = 0.01;
21  x = 0:deltaX:10;
22
23  withoutOutput = netWithout(x);
24  withOutput = netWith(x);
```

(c)

```
26 figure();
27 scatter(data(:,1), data(:,2));
28 hold on;
29 plot(x,withoutOutput);
30 hold on;
31 plot(x,withOutput);
32
33 xlabel('x');
34 ylabel('y');
35 title('Vergleich mit vs. ohne Regularisierung');
36 legend('Datenpunkte', 'Ohne Regularisierung', 'Mit
       Regularisierung');
37 print("b08a02.eps", "-depsc");
```

(d)