Machine Learning Übungsblatt 5

Ramon Leiser Tobias Hahn

12. Januar 2017

Fragen

0.1 POMDP

In einem MDP geht es darum, dass ein Agent aufgrund von Beobachtungen über die Welt sequentielle Entscheidungen trifft, und dabei entweder das Risiko minimiert oder den Ertrag maximiert. Dabei wird davon ausgegangen, dass die Welt direkt beobachtbar ist, dass also die Zustände vom Agenten eindeutig festgestellt werden können.

Beim POMDP hingegen wird davon ausgegangen, dass der Agent den Zustand der Welt nicht direkt feststellen kann, sondern nur aufgrund von den Beobachtungen die er macht die Wahrscheinlichkeit von mehreren möglichen Zuständen abschätzen kann. Nun sollen die getroffenen Entscheidungen nicht mehr einfach Risiko minimieren oder maximieren, sondern das jeweilige Ziel optimal erreichen für das mit der Wahrscheinlichkeit gewichtete Mittel über alle möglichen Zustände.

Dazu passt das Markov Hidden Model. Dies teilt den Grundgedanken, dass der Zustand der Welt nicht direkt sichtbar ist, sondern man nur Beobachtungen machen kann, und von denen aus auf eine gewisse Wahrscheinlichkeit des Zustands der Welt schließen kann. Wir haben das gemacht anhand eines Fußballspiels, welches man nicht direkt beobachten, sondern nur die hörbare Reaktion der Fans feststellen kann. Bei einem POMDP würde man darüber hinaus noch Aktionen haben, welche je nach Zustand andere Ergebnisse zeitigen, und müsste dann mit dem Wahrscheinlichkeitswissen über die versteckten Zustände den jeweils besten auswählen.

0.2 EM

Der EM-Algorithmus beschreibt eine Vorgehensweise, um Parameter in einem Modell so optimal auszuwählen, dass sie die vorliegenden Daten am besten beschreiben. Dabei wird im Expectation-Schritt eine Funktion gefunden, welche die Wahrscheinlichkeit der vorgefundenen Daten mit dem Modell unter den bisherigen Parametern beschreibt. Im Maximization-Schritt werden die neuen Parameter gefunden, indem die vorher aufgestellte Funktion mit Respekt zu den Parametern maximiert wird, also die Parameter gefunden werden welche das Auftauchen der Trainingsdaten maximiert.

Dabei kann der EM-Algorithmus nur dann angewendet werden, wenn bereits ein Modell bekannt ist, oder zumindest angenommen wird dass die Daten einem Modell folgen, sowie dieses Modell auf einigen wenigen unbekannten Parametern beruht, für welche dann maximiert werden kann. Geeignet sind also vor allem Wahrscheinlichkeitsmodelle aus der Statistik, wie Gaussche Verteilungen oder Poisson Verteilung.

1 Markov Decision Processes

2 Markov Decision Process

2.1 Utility Werte

Das Programm liefert nach 100 Iterationen ohne Discount folgende Utility Werte

```
Aufgabe 1: Utilities
0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
0.000000    0.553117    0.664085    0.801122    1.000000    0.000000
0.000000    0.454988    0.000000   -0.410327  -15.000000    0.000000
0.000000    0.374899    0.298694    0.146246   -0.899763    0.000000
0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
```

Um die eigentlichen Werte ist ein Ring aus Nullen, welche nicht betretbare Zustände sind. Die Terminalzustände behalten als Utility ihre Initialwerte.

2.2 Policy

Die Policy wird bei unserer Implementierung direkt von der Maschiene errechnet. In der Methode `determineMax()` wird bei der Bestimmung der Besten Aktion für das Update der Utility werte auch die Richtungsentscheidung direkt als Policy gespeichert:

```
//d is the currently tested Direction
if(sum>currentMax){
                currentMax=sum;
                policy[y][x]=d;
            }
```

Dadurch ist die Policy ohne Discount nach 100 Iterationen:

```
Aufgabe 2: Policies
null  null  null  null  null  null
null  RIGHT RIGHT RIGHT null  null
null  UP    null  UP    null  null
null  UP    LEFT  LEFT  DOWN  null
null  null  null  null  null  null
```

2.3 Discount

Das Programm findet atomatisch die erste Policy, welche von der Obigen abweiche. Es reduziert dazu den discount bei jeder Wiederholung om `0.01`

```
Aufgabe3
Policy changed at: 0.65000033  To:
null  null  null  null  null  null
null  RIGHT RIGHT RIGHT null  null
null  UP    null  UP    null  null
null  UP    LEFT  DOWN  DOWN  null
null  null  null  null  null  null
From:
null  null  null  null  null  null
null  RIGHT RIGHT RIGHT null  null
null  UP    null  UP    null  null
null  UP    LEFT  LEFT  DOWN  null
null  null  null  null  null  null
```

Es ändert sich lediglich die Zelle `(4,3)` Von `DOWN` zu `LEFT`, was die Policy etwas gieriger und mutiger macht, da so auch die möglichkeit besteht auf dem Feld neben den Bahngleisen bzw. der Weser zu landen.

2.4 Reward Weser

Ändert man die Belohnung für die Weser auf `-300` und setzt den Discount wieder auf `0`, so wird die Policy nach 100 Iterationen zu ängstlich und kann tatsächlich nie zum Ziel kommen. Durch zwei Unvorhergesehene Schritte könnte der komilitone zwar gemäß der Policy in die obere Linke Ecke kommen, von dort aus würde er sich jedoch niemals nach Rechts bewegen. Die Schritte dort hin werden bereits als zu gefährlich eingeschätzt.

```
Reward fuer Fluss auf -300
0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
0.000000    -0.013574   -0.045074   -0.487489   1.000000    0.000000
0.000000    -0.012425   0.000000    -18.211212  -300.000000     0.000000
0.000000    -0.021074   -0.170078   -2.386974   -18.177088  0.000000
0.000000    0.000000    0.000000    0.000000    0.000000    0.000000

null  null  null  null  null  null
null  LEFT  UP    RIGHT null  null
null  RIGHT null  LEFT  null  null
null  LEFT  LEFT  DOWN  DOWN  null
null  null  null  null  null  null
```

Ändert man hingegen die Belohnung für das Feld Weser auf `-3` so wird die Policy sehr mutig und würde sogar den Weg unterhalb des `X` nehmen um zum Ziel zu Kommen, direkt an der Weser vorbei.

```
 Reward fuer Fluss auf -2
0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
0.000000    0.605438    0.725594    0.874084    1.000000    0.000000
0.000000    0.499468    0.000000    0.579265    -2.000000   0.000000
0.000000    0.418631    0.415091    0.499967    0.249973    0.000000
```

```
0.000000     0.000000     0.000000     0.000000     0.000000     0.000000

null   null   null   null   null   null
null   RIGHT RIGHT RIGHT null   null
null   UP null   UP null   null
null   UP RIGHT UP LEFT   null
null   null   null   null   null   null
```

2.5 Delta als Terminierungskriterium

Sieht man sich die beiden Policys, welche mit den aus der Aufgabenstellung geforderten Parametern kreiert werden sieht man dass sie sich nicht ändern:

```
 Reward fuer Fluss auf -1000, Abbruch bei delta < 0.02
0.000000     0.000000     0.000000     0.000000     0.000000     0.000000
0.000000     -0.025381    -0.237923    -3.480986    1.000000     0.000000
0.000000     -0.014538    0.000000     -60.808903   -1000.000000    0.000000
0.000000     -0.042222    -0.518717    -7.934514    -60.565323   0.000000
0.000000     0.000000     0.000000     0.000000     0.000000     0.000000

null   null   null   null   null   null
null   LEFT   LEFT   RIGHT  null   null
null   RIGHT  null   LEFT   null   null
null   LEFT   LEFT   DOWN   DOWN   null
null   null   null   null   null   null

 Reward fuer Fluss auf -1000, Abbruch bei delta < 0.002
null   null   null   null   null   null
null   LEFT   LEFT   RIGHT  null   null
null   RIGHT  null   LEFT   null   null
null   LEFT   LEFT   DOWN   DOWN   null
null   null   null   null   null   null
```

Lässt man sich auf der Konsole die Deltawerte der Iterationen ausgeben so sieht man, dass beide Settings genau gleich viele Iterationen erzeugen:

```
60.009403228759766 7.7939229011535645 0.8440335392951965 0.5087775588035583
0.41358691453933716 0.329082101583480830 0.2648226022720337 0.2159448266029358
0.17371797561645508 0.13288389146327972 0.10694222152233124 0.07113255560398102
0.057297706604003906 0.025176461786031723 0.020351544022560120 0.00145682692527771
```

Beide male ist der Vorletzte Schritt über `0.02` und der nächste bereits unter `0.002`. Deshalb sind auch die Policies entsprechend gleich.

Value Iteration Optimisation

Es gibt eine Variation des Policy Iteration Algorithmus, welche die Policy für mehrere Schritte konstant hält und nur die Utility werte verändert.[1] (Zitiert nach [2]) Durch diese verbesserung wird die Policy gleichmäßiger der optimalen angenähert, jedoch wird das Optimum erst nach mehr Iterationen erreicht.

Weiterhin kann bei der Policy iteration die Berechnung der Utility Werte als Gleichungssystem gesehen werden. Es gibt eine Gleichung für jeden State, da für alle States ein Utility-Wert berechnet werden muss und in jeder Gleichung eventuell die möglichkeit in jeden anderen State zu kommen, weshalb in jeder Zeile all die anderen States als Variable vorkommen können.

Dieses Gleichungssystem kann mit dem Gauß-Jordan-Verfahren optimal gelößt werden, allerdings fürt das zu einer Komplexität von $O(n^3)$ (bei n Zuständen) was jedoch für große n schnell zu lange dauert. Deshalb kann man numerische Verfahren verwenden um die Lösung des Gleichungssystems zu approximieren. Dies Reduziert die Komplexität des Schrittes auf $O(n^2)$. Die reduzierte genauigkeit der Utility-Werte könnte jedoch dazu führen, dass etwas mehr Iterationen benötigt werden oder im Extremfall die optimale Policy nicht gefunden wird.[3]

[1] Martin L. Puterman and Moon Chirl Shin. Modified policy iteration algorithms for discounted Markov decision processes. Management Science, 24:1127-1137, 1978.

[2] `https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/node21.html`

[3] `http://www.mit.edu/\~dimitrib/Adaptive\_aggr.pdf`

3 Perzeptron

initial sind die Gewichte auf 0 gesetzt. Als Trainingsdaten für die Gewichtsanpassung per Gradientenabstieg werden die Belegungen der Variablen aus der Angabe benutzt. Da alle Daten angegeben werden sollen und die Trainingsdatenmenge äußerst gering ist, bietet es sich an die Berechnungen per Hand durchzuführen. Die zufällige Auswahl der Trainingsdaten wird per Hand gemacht.

3.1 Schritt 1

w_0	0
w_1	0
w_2	0

Zufällig ausgewähltes Beispiel (aus den vier Trainingsbeispielen) ist Beispiel 1. Input ist also 0 und 0, sowie Output 0. Zuerst muss die Aktivierung des Perzeptrons berechnet werden. Dazu brauchen wir den Wert y, der sich aus den Gewichten und den Eingabewerten ergibt, sowie den sich daraus ergebenden Wert der Aktivierungsfunktion.

$$y = w_0 * I_0 + w_1 * I_1 - w_2 = 0 * 0 + 0 * 0 - 0 = 0$$
$$\sigma(y) = \frac{1}{1 + e^{-y}} = 0.5$$

Nachdem nun die Aktivierung des Perzeptrons feststeht muss der Fehler berechnet werden. Hierzu verwenden wir den quadrierten Fehler, also das Quadrat des Unterschieds zwischen der erwarteten Ausgabe und der tatsächlichen.

$$Fehler = \frac{1}{2} * (o - O)^2 = 0.5 * (0.5 - 0)^2 = 0.125$$

Um nun die Gewichte anzupassen müssen wir uns zuerst den Gradienten ausrechnen. Der Gradient baut auf auf dem Fehler der Ausgabe sowie den Ableitungen der Aktivierungsfunktion. Außerdem baut er auf auf dem Input.

Um den Gradienten der Fehlerfunktion bezüglich der Gewichte zu berechnen müssen wir zuerst einmal die Fehlerfunktion bezüglich dieses Gewichts ableiten. Dies geschieht folgendermaßen (hier am Beispiel Gewicht 1 - o steht für die wahre Ausgabe, y für das Ergebnis der Linearen Gleichung):

$$\begin{aligned}
&\frac{\partial E}{\partial w_1} E(o - \sigma(y)) \\
&= \frac{\partial E}{\partial w_1} \frac{1}{2} (o - \sigma(y))^2 \\
&= (o - \sigma(y)) * (\frac{\partial E}{\partial w_1} o - \frac{\partial E}{\partial w_1} \sigma(y)) \\
&= (o - \sigma(y)) * -(\sigma(y) * (1 - \sigma(y)) * \frac{\partial E}{\partial w_1} (w_0 * I_0 + w_1 * I_1 - w_2) \\
&= (o - \sigma(y)) * -(\sigma(y) * (1 - \sigma(y))) * I_1 \\
&= Fehler * \sigma(y) * (1 - \sigma(y)) * I_1
\end{aligned}$$

Für die anderen Gewichte müsste man jeweils noch den Index des Signals ändern. Der Einfachheit halber wurde das Vorzeichen bei der Ableitung der Sigmoidfunktion weggelassen. Dafür muss beim Gradientenabstieg der Gradient einfach abgezogen werden, statt dazugezählt.

$$\frac{\delta E}{\delta w_0} = \sigma(y) * (1 - \sigma(y)) * Fehler * I_0 = 0.5 * 0.5 * 0.5 * 0 = 0$$
$$\frac{\delta E}{\delta w_1} = \sigma(y) * (1 - \sigma(y)) * Fehler * I_1 = 0.5 * 0.5 * 0.5 * 0 = 0$$
$$\frac{\delta E}{\delta w_2} = \sigma(y) * (1 - \sigma(y)) * Fehler * -1 = 0.5 * 0.5 * 0.5 * -1 = -0.125$$

Da der Gradient bei den ersten zwei Gewichten 0 ist nützt auch eine Anpassung nichts - nur beim letzten Gewicht, dem Biasgewicht, nehmen wir eine Anpassung vor. Hier lautet die Formel für das neue Gewicht:

$$w_2 = w_2 - \alpha * \frac{\delta E}{\delta w_2} = 0 - 10 * -0.125 = 0 + 1.25 = 1.25$$

3.2 Schritt 2

w_0	0
w_1	0
w_2	1.25

Zufällig ausgewähltes Beispiel (aus den vier Trainingsbeispielen) ist Beispiel 3. Input ist also 0 und 1, sowie Output 1. Zuerst muss die Aktivierung des Perzeptrons berechnet werden. Dazu brauchen wir den Wert y, der sich aus den Gewichten und den Eingabewerten ergibt, sowie den sich daraus ergebenden Wert der Aktivierungsfunktion.

$$y = w_0 * I_0 + w_1 * I_1 - w_2 = 0 * 0 + 0 * 1 - 1.25 = -1.25$$
$$\sigma(y) = \frac{1}{1 + e^{-y}} = 0.2227$$

Nachdem nun die Aktivierung des Perzeptrons feststeht muss der Fehler berechnet werden. Hierzu verwenden wir den quadrierten Fehler, also das Quadrat des Unterschieds zwischen der erwarteten Ausgabe und der tatsächlichen.

$$Fehler = \frac{1}{2} * (o - O)^2 = 0.5 * (0.2227 - 1)^2 = 0.5 * -0.7773^2 = 0.3$$

Um nun die Gewichte anzupassen müssen wir uns zuerst den Gradienten ausrechnen. Der Gradient baut auf auf dem Fehler der Ausgabe sowie den Ableitungen der Aktivierungsfunktion. Außerdem baut er auf auf dem Input. Die Formeln für den Gradienten sind:

$$\frac{\delta E}{\delta w_0} = \sigma(y) * (1 - \sigma(y)) * Fehler * I_0 = 0.2227 * 0.7773 * -0.7773 * 0 = 0$$
$$\frac{\delta E}{\delta w_1} = \sigma(y) * (1 - \sigma(y)) * Fehler * I_1 = 0.2227 * 0.7773 * -0.7773 * 1 = -0.1346$$
$$\frac{\delta E}{\delta w_2} = \sigma(y) * (1 - \sigma(y)) * Fehler * -1 = 0.2227 * 0.7773 * -0.7773 * -1 = 0.1346$$

Da der Gradient beim ersten Gewicht 0 ist nützt auch eine Anpassung nichts - nur bei den letzten zwei Gewichten nehmen wir eine Anpassung vor. Hier lautet die Formel für das neue Gewicht:

$$w_1 = w_1 - \alpha * \frac{\delta E}{\delta w_1} = 0 - 10 * -0.1346 = 0 + 1.346 = 1.346$$
$$w_2 = w_2 - \alpha * \frac{\delta E}{\delta w_2} = 1.25 - 10 * 0.1346 = 1.25 - 1.346 = -0.096$$

3.3 Schritt 3

w_0	0
w_1	1.346
w_2	-0.096

Zufällig ausgewähltes Beispiel (aus den vier Trainingsbeispielen) ist Beispiel 2. Input ist also 1 und 0, sowie Output 1. Zuerst muss die Aktivierung des Perzeptrons berechnet werden. Dazu brauchen wir den Wert y, der sich aus den Gewichten und den Eingabewerten ergibt, sowie den sich daraus ergebenden Wert der Aktivierungsfunktion.

$$y = w_0 * I_0 + w_1 * I_1 - w_2 = 1 * 0 + 0 * 1.346 + 0.096 = 0.096$$
$$\sigma(y) = \frac{1}{1 + e^{-y}} = 0.524$$

Nachdem nun die Aktivierung des Perzeptrons feststeht muss der Fehler berechnet werden. Hierzu verwenden wir den quadrierten Fehler, also das Quadrat des Unterschieds zwischen der erwarteten Ausgabe und der tatsächlichen.

$$Fehler = \frac{1}{2} * (o - O)^2 = 0.5 * (0.524 - 1)^2 = 0.5 * -0.476^2 = 0.11135$$

Um nun die Gewichte anzupassen müssen wir uns zuerst den Gradienten ausrechnen. Der Gradient baut auf auf dem Fehler der Ausgabe sowie den Ableitungen der Aktivierungsfunktion. Außerdem baut er auf auf dem Input. Die Formeln für den Gradienten sind:

$$\frac{\delta E}{\delta w_0} = \sigma(y) * (1 - \sigma(y)) * Fehler * I_0 = 0.227 * 0.476 * -0.476 * 1 = -0.05$$
$$\frac{\delta E}{\delta w_1} = \sigma(y) * (1 - \sigma(y)) * Fehler * I_1 = 0.227 * 0.476 * -0.476 * 0 = 0$$
$$\frac{\delta E}{\delta w_2} = \sigma(y) * (1 - \sigma(y)) * Fehler * -1 = 0.227 * 0.476 * -0.476 * -1 = 0.05$$

Da der Gradient beim zweiten Gewicht 0 ist nützt auch eine Anpassung nichts - nur bei den anderen zwei Gewichten nehmen wir eine Anpassung vor. Hier lautet die Formel für das neue Gewicht:

$$w_0 = w_0 - \alpha * \frac{\delta E}{\delta w_0} = 0 - 10 * -0.05 = 0 + 0.5 = 0.5$$
$$w_2 = w_2 - \alpha * \frac{\delta E}{\delta w_2} = -0.096 - 10 * 0.05 = -0.096 - 0.5 = -0.596$$

3.4 Schritt 3

w_0	0.5
w_1	1.346
w_2	-0.596

Zufällig ausgewähltes Beispiel (aus den vier Trainingsbeispielen) ist Beispiel 4. Input ist also 1 und 1, sowie Output 1. Zuerst muss die Aktivierung des Perzeptrons berechnet werden. Dazu brauchen wir den Wert y, der sich aus den Gewichten und den Eingabewerten ergibt, sowie den sich daraus ergebenden Wert der Aktivierungsfunktion.

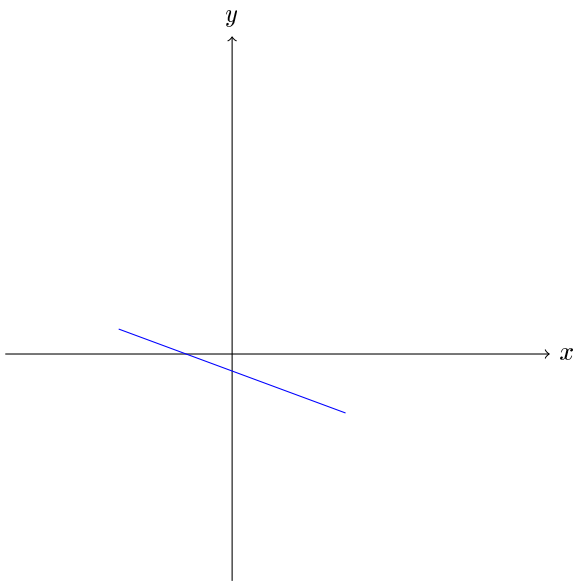
$$y = w_0 * I_0 + w_1 * I_1 - w_2 = 0.5 * 1 + 1.346 * 1 + 0.596 = 2.442$$
$$\sigma(y) = \frac{1}{1 + e^{-y}} = 0.92$$

Nachdem nun die Aktivierung des Perzeptrons feststeht muss der Fehler berechnet werden. Hierzu verwenden wir den quadrierten Fehler, also das Quadrat des Unterschieds zwischen der erwarteten Ausgabe und der tatsächlichen.

$$Fehler = \frac{1}{2} * (o - O)^2 = 0.5 * (0.92 - 1)^2 = 0.5 * -0.08^2 = 0.0032$$

Damit ist der quadrierte Fehler unter 0.1 gesunken, was in der Angabe als Abbruchbedingung eingeführt wurde. Damit sind die Gewichte, die wir herausgefunden haben, auch die tatsächlichen, und wir können mit den Berechnungen aufhören.

Plot Um zu veranschaulichen zu welchem Ergebnis wir gekommen sind, hier ein Plot der Linearen Funktion welche von den Gewichten beschrieben wird:



3.5 Beispiele

Laut Angabe sollen noch zwei Beispieleingaben durchgerechnet werden, und die Ergebnisse interpretiert. Dies geschieht hier.

3.5.1 0.5

Beide Eingaben sind 0.5. Damit ergibt sich folgende Berechnung:

$$y = w_0 * I_0 + w_1 * I_1 - w_2 = 0.5 * 0.5 + 1.346 * 0.5 + 0.596 = 1.519$$
$$\sigma(y) = \frac{1}{1 + e^{-y}} = 0.82$$

Hier entspricht die Berechnete Wahrscheinlichkeit ungefähr dem Wert der erwartet wird. Dies begründet sich so: Bei einer Wahrscheinlichkeit von 0.5 ist die Wahrscheinlichkeit für 1 oder 0 gleich wahrscheinlich, d.h. der Eingabewert entspricht dem Münzwurf einer perfekten Münze. Hier können nun vier verschiedene Ausgaben vorkommen, so wie in der Tabelle in der Angabe angegeben. Bei drei dieser Ausgaben bekommen wir einen Wert 1, bei einer 0, d.h. der Erwartungswert für die Ausgabe ist 0.75, was recht nahe an unserem Ergebnis ist.

3.5.2 0.1

Beide Eingaben sind 0.1. Damit ergibt sich folgende Berechnung:

$$y = w_0 * I_0 + w_1 * I_1 - w_2 = 0.5 * 0.1 + 1.346 * 0.1 + 0.596 = 0.78$$
$$\sigma(y) = \frac{1}{1 + e^{-y}} = 0.69$$

Diese Berechnung deckt sich nicht mit dem Wert den wir erwarten. Die Wahrscheinlichkeit dass einer der Eingaben 0 ist beträgt 0.9, d.h. dass beide Eingaben 0 sind 0.9 * 0.9 oder 0.81. In allen anderen Fällen ist die Ausgabe 1, d.h. unsere Ausgabe sollte 1 - 0.81 oder 0.19 sein. Davon ist unser Wert um 0.5 entfernt, was eine große Abweichung darstellt. Dies liegt daran, dass das Perzeptron nicht genug trainiert wurde, da immer nur ein Beispiel gewählt wurde, und es zu zufälligen sehr guten Ergebnissen bei einem Beispiel kommen kann. Besser wäre es gewesen alle Beispiele auf einmal zu lernen, dann hätte die Summe der Fehler etwas mehr ausgesagt, dan wäre aber mit Hand rechnen sehr kompliziert gewesen.

3.6 Lernrate

Die Lernrate hat einen äußerst großen Einfluss auf den Gradientenabstieg. Um diesen Einfluss zu verstehen muss man zuerst einmal verstehen was ein Gradient ist. Ein Gradient ist im Grunde nichts anderes als ein Vektor, der in eine Richtung zeigt. Diese Richtung ist die Richtung, in der man die jeweiligen Elemente des

Vektors verändern muss, um die Funktion für die der Gradient berechnet wurde zu maximieren (deswegen nimmt man bei der Gewichtsanpassung immer den negativen Gradienten). Die Lernrate gibt dann an, wie weit man in diese Richtung geht. Schon hier kann man erkennen dass die Lernrate eine große Rolle spielt, jedoch muss man sich noch verdeutlichen welche genauen Auswirkungen das hat.

Nehmen wir dafür einmal an wir haben eine zu große Lernrate gewählt. In diesem Fall machen wir einen so weiten Schritt in die Richtung des Minimums, dass wir das Minimum überschreiten und auf der anderen Seite den Funktionswert wiederum vergrößern statt verkleinern. Das kann die Auswirkung haben dass wir einen verlängerten Abstieg haben, aber immer noch absteigen - schreiten wir jedoch so weit über das Minimum hinaus, dass der Fehlerwert tatsächlich größer wird als beim Ausgangspunkt, so kann es passieren dass der Fehler sich immer weiter aufschaukelt (bei konkaven Funktionen ist dies sogar garantiert).

Dagegen ist die Lernrate zu klein gewählt: Hier wird sich zwar der Fehler nie automatisch vergrößern, jedoch machen wir einen so kleinen Schritt in die Richtung des Minimums dass unzählige Epochen notwendig sind um eine Verbesserung zu spüren. Dies führt dazu dass unser Algorithmus endlos lange rechnen muss bis er eine aktzeptable Lösung gefunden hat.