Hochschule **Rosenheim**
University of Applied Sciences

Technische Hochschule Rosenheim

Fakultät für Informatik

Seminararbeit

im Masterstudiengang Informatik - Schwerpunkt Software Engineering

Thema:	Optimisierungsmethoden neuronaler Netze
Autor:	Victor Wolf victorwolf@outlook.de MatNr. 845615
Version vom:	1. Dezember 2019
Betreuer:	Prof. Dr. Holaubek

Zusammenfassung

Diese Arbeit wird sich mit verschiedenen Optimierungsmethoden neuronaler Netze beschäftigen und Sie auf Basis von Beispiel Datensätzen evaluieren, wie dem Boston House Price Datensatz. Hierbei wird eine Metrik definiert um die Ergebnis der einzelnen Optimierungsmethode zu vergleichen.
Eine Optimierungsmethode ist eine Möglichkeit die Konvergenz der Fehlerfunktion $J(\theta)$ des neuronalen Netzes beim Lernen zu verbessern.
Hierbei wird auf den Lern Prozess des Neuronalen Netzes eingegangen. Besonderen Fokus wird der 'Gradient Descent', zu Deutsch Gradienten abstieg, einnehmen, da dies die Grundlage des Lernens darstellt. Dieser sucht im mehrdimensionalen Raum die Minima der nichtlinearen Fehlerfunktion und es gibt verschiedene Möglichkeiten diese Suche zu verbessern. Nach der theoretischen Aufarbeitung, werden wir ein paar Eigenschaften über diese Optimierungsmethoden annehmen und diese anhand der Test Daten überprüfen.

Abstract

This work will focus on explaning the different optimization methods of neural networks and evaluating them on example datasets like the boston house price data. We will define a metric to be able to evaluate the performance of the different optimizer.
An optimization method is a way to improve the performance of the error function $J(\theta)$ of the neural network.
Furthermore this work will give a detailed explanation of the learning process of Neural Networks especially focusing on the Gradient Descent, which is the foundation of learning in neural networks. This algorithm aims to find local minima in the Hyperplane of the non-linear error function and there are multiple ways to improve its search. After the Theory, we will assume some properties about those optimization methods and test those assumptions by evaluating the metrics of these neural networks.

Inhaltsverzeichnis

1 Einleitung

2 Theoretische Grundlagen

2.1 Neuronale Netze

Künstliche Neuronale Netze kurz KNNs sind der menschliche Versuch das biologische Nervensystem nachzuahmen. Sie basieren auf der Tatsache der Reizweitergabe. So wird ein Eingangsreiz von Rezeptoren aufgenommen und über verschiedene sogenannter Neuronen weitergegeben. Durch diese Weitergabe wird das Signal verändert, bis ein Ausgangssignal interpretiert werden kann. Diese Funktionsweise macht man sich bei künstlichen Neuronalen Netzen zu nutze. Der Eingangsreiz sind hier die sogenannten “features“, der Ausgangsreiz eine Klasse oder ein Wert der interpretiert werden kann. Wir wollen uns hier nun nur auf die “Feed Forward“ Netze fokussieren. Das bedeutet das Neuronen ihre Ausgabe nur in eine Richtung schicken dürfen.

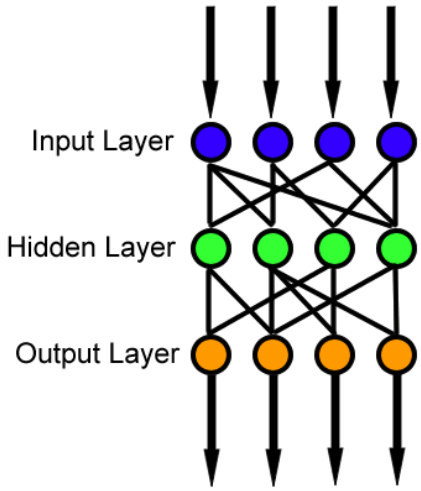

Abbildung 1: Beispiel eines Feed Forward KNNs

KNNs existieren in zwei Zuständen der Trainingsphase und der Arbeitsphase. Die Trainingsphase ist die interessantere und wird in dieser Arbeit beleuchtet. Hier werden durch Optimierung der Fehlerfunktion die Neuronen so ëingestellt", dass sie einen möglichst gute Vorhersage treffen.
Im Folgenden soll nun der Begriff des Neurons formalisiert werden, um die Verbesserungsmöglichkeiten des Gradienten Verfahrens in Abschnitt 2.3 nachvollziehen zu können.

Definition 2.1 *[Bur97, Kapitel 1.2] Ein* ***(formales) Neuron*** *ist eine Funktion* $\kappa : \mathbb{R}^n \to \mathbb{R}^m$ *definiert durch:*

- *eine Aktivierungsfunktion* $T : \mathbb{R} \to \mathbb{R}$

- *ein gewichteter Vektor* $\vec{w} = \{w_1, w_2, ..., w_n\}$
- *und eine Schwelle* $\Theta \in \mathbb{R}$.

Der Vektor $\vec{x} = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$ *wird auf den Vektor* $\vec{y} = (y, y, ..., y) \in \mathbb{R}^m$ *mit identischen Komponenten durch die folgende Rechenvorschrift abgebildet*

$$\kappa(\vec{x}) := (T(\sum_{i=1}^{n} w_i x_i - \Theta), ..., T(\sum_{i=1}^{n} w_i x_i - \Theta)) = \vec{y} \in \mathbb{R}^m \tag{1}$$

Hier seien ein paar Beispiele für Aktivierungsfunktionen angegeben

- Identität T_I

$$T(x) := x = T_I(x)$$

- Binary step

$$T(x) := \begin{cases} 0, \text{ for } x < 0 \\ 1, \text{ for } x \geq 0 \end{cases} =: T_1(x)$$

- Sigmoid

$$T(x) := \frac{1}{1 + e^{-x}} =: T_S(x)$$

- Tangens hyperbolicus

$$T(x) := \frac{1 + tanh(x)}{2} =: T_H(x)$$

Dies sind nur ein paar wenige Beispiele. Jede Funktion $T : \mathbb{R} \to \mathbb{R}$ die $\lim_{x \to -\infty} T(x) = 0$ and $\lim_{x \to \infty} T(x) = 1$ erfüllt, kann als Aktivierungsfunktion genutzt werden.

Definition 2.2 *[Mic, Kapitel 2] Die* ***Fehlerfunktion*** $J(\theta)$ *eines Neuronalen Netzes ist eine differenzierbare Funktion für die gilt:*

- $J(\theta) = \frac{1}{n} \sum_x J_x$ *wobei* x *ein Eingabe Datum beschreibt.*
- $J(\theta)$ *lässt sich aus der Summe der Elemente des Ausgabe Vektors darstellen.*

Die erste Eigenschaft bedeutet, dass die gesamte Fehlerfunktion sich auch durch die Fehlerfunkton der einzelnen Eingabe Daten darstellen lässt. Diese Fehlerfunktion wird im nächsten Abschnitt minimiert werden, um eine optimale Parameterbelegung der Gewichte $\vec{w}$ zu finden. Beispiele für eine solche Funktion wäre der mittlere quadratische Fehler.

2.2 Gradient Descent

Der Gradient Descent oder zu Deutsch Gradienten Abstiegsverfahren ist ein Weg eine Zielfunktion $J(\theta)$ parametrisiert durch $\theta \in \mathbb{R}^n$ zu minimieren. Man aktualisiert diese Parameter in Richtung des stärksten Abstiegs der Zielfunktion $\nabla_\theta J(\theta)$. Die Lern Rate μ bestimmt dabei die Größe der Aktualisierungsschritte. Das Verfahren folgt also der Richtung des Abstiegs der Oberfläche der Zielfunktion in ein Tal, welches ein lokales Minimum beschreibt. [Rud]
Im Fall eines neuronalen Netzes ist die Zielfunktion $J(\theta)$ die Fehlerfunktion des neuronalen Netzes. Wir benötigen also eine Möglichkeit die Richtung des stärksten Abstiegs der Fehlerfunktion zu bestimmen.

Definition 2.3 *[Kön02] Der* ***Gradient*** ∇ *der total differenzierbaren Funktion* $f : \mathbb{R}^n \to \mathbb{R}$ *im Punkt* $a \in \mathbb{R}$ *ist im Falle des Standard Skalar Produkts definiert durch:*

$$\nabla f := \frac{\partial f}{\partial x_1}\hat{e}_1 + \cdots + \frac{\partial f}{\partial x_n}\hat{e}_n \tag{2}$$

In einfachen Worte gefasst, ist der Gradient die Ableitung einer mehrdimensionalen Funktion, deren Funktionswerte man sich als Gebirge vorstellen kann. Hierbei ist der Gradient in einen Punkt ein Vektor der in die Richtung des stärksten Anstiegs.
Um die Richtung des stärksten Abstiegs zu erhalten, welche wir beim Gradienten Abstiegsverfahren benötigen, müssen wir nur den negativen Gradienten berechnen.

Mit diesem Wissen können wir nun den Standard gradient descent algorithmus definieren.

Definition 2.4 *[Rud, Kapitel 2.1] Der sogenannte* ***batch gradient descent****, berechnet den Gradienten der Kostenfunktion für den gesamten Datensatz. Jedes Update der Parameter ist definiert durch*

$$\theta = \theta - \mu \cdot \nabla_\theta J(\theta) \tag{3}$$

Als Algorithmus würde der batch gradient descent folgendermaßen aussehen.

```
for i in range(nb_epochs ):
    params_grad = evaluate_gradient(loss_function , data , params)
    params = params  - learning_rate * params_grad
```

wobei `nb_epochs` die Anzahl der Iterationen beschreibt. Diese Implementierung beschreibt die grundsätzliche Idee aber hat mehrere Nachteile. Da wir den Gradienten für den gesamten Datensatz berechnen ist diese Methode sehr langsam und nicht möglich für Datensätze, die nicht in den Arbeitsspeicher passen. Außerdem konvergiert dieser Algorithmus nur sehr langsam gegen ein lokales Minimum.

2.3 Optimierungsmethoden

Im vorherigen Abschnitt haben wir die Grundlagen des Gradienten Abstiegsverfahren kennengelernt und gesehen, dass dieses Probleme mit sich bringt. Die folgenden Optimierungsmethoden verbessern das grundsätzliche batch gradient descent Verfahren in die ein oder andere Richtung.

2.3.1 Stochastic Gradient Descent

2.3.2 Adagrad

2.3.3 Adam

3 Evaluation

4 Fazit

Literaturverzeichnis

[Bur97] Burkhard Lenze: *Einführung in die Mathematik neuronaler Netze*. Berlin : Logos Verlag, 1997

[Kön02] Königsberger: *Analysis 2*. Springer Berlin Heidelberg, 2002. – ISBN 3540435808

[Mic] Michael Nielsen: *Neural Networks and Deep Learning*. `http://neuralnetworksanddeeplearning.com/index.html`

[Rud] Ruder, Sebastian: *An overview of gradient descent optimization algorithms*. `http://arxiv.org/pdf/1609.04747v2`

Eidesstattliche Erklärung

Eidesstattliche Erklärung zur Seminararbeit

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

$Unterschrift$: $Ort, Datum$: