Fakultät Informatik/Mathematik

Wissenschaftliches Seminar

im Wintersemester 2017/2018

Eine Einführung in TensorFlow

---

bearbeitet von:	Bierschneider Christian	3118760
	Maximilian Poeschl	3121342
	Benjamin Maiwald	3097528
	Studiengang: Informatik	
	Schwerpunkt: Software Engineering	
Betreuer:	Prof. Dr. Jan Dünnweber	
	OTH Regensburg	

Regensburg, 18. November 2017

Abstract

Abkürzungsverzeichnis

KI	Künstliche Intelligenz.
ML	Machine Learning, bzw. Maschinelles Lernen.
Neuronales Netz	Neuronales Netz, bzw. Neural Network.

Inhaltsverzeichnis

Abkürzungsverzeichnis **iii**

1 Eine Einführung in Maschinelles Lernen **1**
1.1 Meilensteine des Maschinellen Lernens 1

2 Neuronale Netze in Tensorflow **4**
2.1 Feedforwardnetzwerk 4
2.2 Der Input Tensor 5
2.3 Gewichte 6
2.4 Aktivierungsfunktionen 7
2.4.1 Rectified linear unit function 7
2.4.2 Sigmoidfunktion 8
2.4.3 Tangenshyperbolicus 8
2.5 Kostenfunktionen 9
2.5.1 MSE 9
2.5.2 cross entropy 10
2.6 Lernprozess 10

3 Das Framework TensorFlow **11**
3.1 Eine Einführung zu TensorFlow 11
3.2 Der Weg des Frameworks zur Open Source Software 11
3.3 Angesprochene Zielgruppe 11
3.4 Hard- und Software Anforderungen 11
3.4.1 Hardware Anforderungen 11
3.4.2 Software Anforderungen 11
3.5 Softwarearchitektur von TensorFlow 11

4 Der Allgemeine Workflow in TensorFlow **12**
4.1 Die Tooling Pipeline 12
4.2 Vorgehensweise beim Trainingsprozess 12
4.3 Die Graphelemente im Datenfluss 12

4.4 Die Visualisierungsmöglichkeiten mit TensorBoard 12

5 Ausblick **13**

Abbildungsverzeichnis **14**

Tabellenverzeichnis **15**

Anhang **16**

Literaturverzeichnis **17**

Kapitel 1

1 Eine Einführung in Maschinelles Lernen

Da Vorwissen in den Bereichen Künstliche Intelligenz (KI) und Machine Learning (ML) selbst bei Studierenden der Informatik nicht generell vorausgesetzt werden kann, gibt dieses Kapitel eine kurze Einführung in das Thema. Es wird über die Grundlagen im Bereich des Maschinellen Lernens mit dem Schwerpunkt auf Neuronale Netze (NL) informiert, die zum Verständnis der Arbeit benötigt werden. Sollte sich der Leser bereits mit dem Thema auseinander gesetzt haben und Begriffe wie „Label“, „Schichten (Layers)“ und „Gewichte“ schon bekannt sein, kann dieses Kapitel auch übersprungen werden.

Für große Teile des geschichtlichen Abrisses diente das Buch „Künstliche Intelligenz, ein moderner Ansatz“ von Stuart Russell und Peter Norvig als Quelle, welches wohl eines der bekanntesten Werke zum Thema KI sein dürfte [1]. Generell kann ich dieses Buch allen Interessierten empfehlen, gleichwohl aufgrund des doch sehr großen Umfangs je nach Situation meist nur einzelne Kapitel hilfreich sein werden.

1.1 Meilensteine des Maschinellen Lernens

In den letzten Jahren (seit ca. 2010) hat sich das Thema KI zu einem regelrechten Hype-Thema entwickelt — und das nicht ganz zu unrecht. Denn gerade durch Anwendungen in den Bereichen Bildverarbeitung, Spracherkennung, sogenannter „Recommender Systems“ oder auch des automatisierten Fahrens, gab es enorme Fortschritte. Diese machten KI für die breite Masse salonfähig und ermöglichten die Entwicklung von Produkten wie Siri, den Skype Translator, Filmempfehlungen auf Netflix oder die Fahrerassistenzsysteme im Tesla Model S, die heute von Millionen von Menschen täglich genutzt werden.

Allen voraus liegt das Hauptaugenmerk vieler Informatiker und Forscher gerade auf den sogenannten „Künstlichen Neuronalen Netzen“ (Artificial Neural Networks). Manche dieser Neuronalen Netze wurden sogar zu echten Superstars in der Szene, wie zum Beispiel „AlphaGo“, das von Google entwickelt wurde und 2016 den damaligen Vize-Weltmeister Lee Sedol in vier von fünf Runden im Brettspiel „Go“ besiegte. Aufgrund der unglaublichen

Komplexität[1] des Spiels galt der Sieg einer Maschine über einen realen Meister lange Zeit als unmöglich.

Dabei sind die meisten Grundlagen auf diesem Gebiet bereits Jahrzehnte alt. Schon in den 1940er Jahren und somit unmittelbar nach der Erfindung des modernen Computers, begannen die ersten Forscher damit, ein Modell für Künstliche Neuronale Netze zu entwickeln und behaupteten sogar bereits, dass entsprechend definierte Netze auch lernfähig seien. In demselben Artikel, in dem Alan Turing 1950 die Idee des weltbekannten Turing-Tests — in [2] „The Imitation Game“ genannt — veröffentlichte, schrieb er außerdem zum ersten Mal über „lernende Maschinen“ und philosophierte darüber, wie man einer Maschine beibringen könne, im Imitation Game zu bestehen. In dieser Niederschrift formulierte Turing auch die Grundideen zum heute als „Reinforcement Learning“ bezeichneten Lernen durch Bestrafung und Belohnung.

1956 war schließlich offiziell das Geburtsjahr der Künstlichen Intelligenz. Am Dartmouth College (Hanover, New Hampshire) veranstalteten McCarthy, Minsky, Shannon und Rochester (allesamt Größen in der Entwicklung der KI) ein „Summer Research Project on Artificial Intelligence“ zusammen mit weiteren Forschern aus ganz Amerika. Hier wurde der Begriff „Artificial Intelligence“ zum ersten Mal überhaupt benutzt. Ziel des Workshops war es, in zwei Monaten einen signifikanten Fortschritt bei der Entwicklung einer intelligenten Maschine zu erreichen. Dieses Ziel konnte zwar nicht erfüllt werden, jedoch sorgte das Treffen dafür, dass sich die wichtigsten Personen kennenlernten, die in den darauffolgenden 20 Jahren die größten Neuerungen auf diesem Gebiet entwickelten.

Wie auch heute gab es schon einmal in den 1980er Jahren einen großen Boom in der KI-Industrie. Die Investitionen stiegen von einigen Millionen Dollar im Jahr 1980 auf mehrere Milliarden Dollar im Jahr 1988. Viele der KI-Firmen konnten ihre Versprechen jedoch nicht halten, weshalb der Markt in den 90er Jahren zusammenbrach. In Folge dessen ging auch die Forschung auf dem Gebiet zurück und es kam zu keinen nennenswerten Erkenntnisgewinnen in den 90er Jahren. Deshalb wird dieses Jahrzehnt auch als „KI-Winter“ bezeichnet.

Wenn alle diese Entwicklungen im Bereich der KI aber bereits so lange zurück liegen, warum hat es dann bis heute gedauert, dass es die ersten Anwendungen zum Endkunden schaffen?

Wir leben heute in einer spannenden Zeit, denn einige wichtige Faktoren, die für den Erfolg der KI wichtig sind, wurden nahezu zeitgleich verfügbar.

[1] Ein Go-Brett besteht aus einem Raster von 19 x 19 Plätzen zum Setzen. Für den ersten Zug existieren also 361 Möglichkeiten, für den zweiten 360 und so weiter. Dies ergibt bereits für die ersten drei Züge mehr als 46 Millionen mögliche Spielabläufe.

Zum einen stieg die Leistung von Computern seit deren Erfindung stetig an. Für die meisten Berechnungen im Bereich des Maschinellen Lernens wird eine sehr hohe Rechenleistung benötigt. Vor allem die Verwendung von GPUs zur parallelen Berechnung von allgemeinen Aufgaben beschleunigt das Trainieren von neuronalen Netzen um ein Vielfaches. Rechenvorgänge, die noch vor zehn Jahren große Rechencluster für viele Monate beanspruchten, können heute in Stunden, maximal aber wenigen Tagen abgeschlossen werden – und das auf kompakten Rechnern, die sogar für Privatpersonen erschwinglich sind. Dies gibt den Forschern und Softwareingenieuren die Möglichkeit, schon nach kurzer Zeit ein Feedback zu erhalten, ob die von Ihnen gewählten Ansätze richtig sind und falls nicht, Anpassungen vorzunehmen.

Zum anderen leben wir im Zeitalter von „Big Data“. Für ML ist es extrem wichtig, dass große Datenmengen zur Verfügung stehen, die für den Trainingsprozess verwendet werden können. Firmen wie Google, Facebook, Apple oder Microsoft (und natürlich vielen anderen) steht ein schier unerschöpflicher Pool an Informationen zur Verfügung. Diese maschinell generierten Daten eignen sich aufgrund ihres großen Umfangs perfekt dazu, neuronale Netze zu trainieren und das wird von diesen Firmen natürlich auch genutzt, um neue Geschäftsideen zu entwickeln und die angebotenen Dienste für ihre Kunden kontinuierlich zu verbessern.

Noch hinzu kam der dritte große Punkt: Die Entwicklung von „hacks“, welche die bereits erforschten Algorithmen leicht abwandelten, um enorme Einsparungen in der Rechenzeit zu erreichen. So fand man zum Beispiel heraus, dass es in den meisten Fällen ausreicht, nur einen bestimmten Prozentsatz eines neuronalen Netzes zu trainieren, um trotzdem nahezu das gleiche Ergebnis zu erzielen [**?**].

Natürlich birgt die Entwicklung der Künstlichen Intelligenz auch Gefahren. Diese können ganz real sein, wie der Wegfall tausender Arbeitsplätze [3] oder aber spekulativ und in der Zukunft liegend, wie die mögliche Gefährdung der Menschheit durch eine künstliche Superintelligenz [4]. In jedem Fall wird die Weiterentwicklung und Forschung auf dem Gebiet auch die nächsten Jahre ein extrem vielfältiges und spannendes Thema bleiben.

Kapitel 2

2 Neuronale Netze in Tensorflow

Neuronale Netze sind das am häufigsten benutzte Werkzeug in Tensorflow. Im Folgenden werden die Grundlagen für Neuronale Netze vorgestellt, sowie deren Umsetzung in Tensorflow.

2.1 Feedforwardnetzwerk

Die häufigste in Tensorflow implementierte Version Neuronaler Netze ist das Feed Forward Netz. Diese werden auch am häufigsten für Deep Learning verwendet.[1] Ein Feed Forward Netz besteht aus einer Eingabeschicht, einer Ausgabeschicht und n versteckten Schichten dazwischen. Hierbei ist jedes Neuron mit jedem Neuron der nachfolgenden Schicht durch

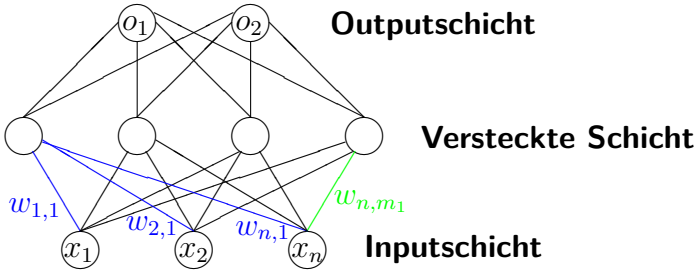

Abbildung 2.1: Feed Forward Netz

Gewichte verbunden.
Die Abbildung zeigt beispielhaft ein Feed Forward Netzwerk mit 3 Input-Neuronen in der Eingabeschicht, einer versteckten Schicht mit 4 versteckten Neuronen und einer Ausgabeschicht mit 2 Ausgabe-Neuronen.[2]
Ein Feed Forward Netzwerk kann beliebig viele versteckten Schichten enthalten, aber nur eine Eingabe und eine Ausgabeschicht. Ebenso kann die Anzahl der Neuronen in den versteckten Schichten frei gewählt werden. Eine sinnvolle Anzahl von Neuronen in den versteckten Schichten lässt sich nur experimentell bestimmen.[3] Man sollte darauf achten

[1] [5] S.168
[2] [6]s.117
[3] [7] S.385

nicht zu wenige Neuronen zu verwenden da sonst die Lernkapazität möglicherweiße zu eingeschränkt ist. Auch zu viele Neuronen können problematisch sein, da es sehr lang dauern kann jede der vielen Neuronen zu trainieren und die Effizienz des Netzwerks darunter leidet.[4]
Man kann bereits mit einer versteckten Schicht und einer ausreichend großen Anzahl Neuronen jedes Problem simulieren, tendenziell ist es aber besser die Anzahl der Schichten zu erhöhen anstatt die Anzahl der Neuronen pro Schicht.[5]

2.2 Der Input Tensor

Vektoren und Matrizen werden in Tensorflow als Tensors bezeichnet. Der Input eines Neuronalen Netzes ist ein n-dimensionaler Vektor der für jedes Input-Neuron einen Wert enthält. Die Anzahl der Input-Neuronen richtet sich nach dem untersuchten Problem. Das Einführungsbeispiel für Tensorflow das in etwa dem "Hello World" für Programmiersprachen entspricht, behandelt ein Klassifikationsproblem.[6] In diesem Problem sollen mithilfe der MNIST Datenbank die tausende von handgeschriebenen 28×28 Bilder der Zahlen von 1-9 enthält, das Neuronale Netz lernen handgeschriebene Ziffern zu unterscheiden. Für jedes dieser Bilder hat man entsprechend eine 28×28 Matrix mit Grauwerten. Mit dem Befehl reshape$(-1,..)$ [7] lässt es sich in einen eindimensionalen Vektor mit $28 * 28 = 784$ Input Neuronen verwandeln.
Input-Tensore werden in Tensorflow mit Platzhaltern angelegt, da während des Lernprozesses der Inputtensor für jedes zu lernende Beispiel aktualisiert wird. Um den Platzhalter anzulegen verwendet man den Befehl

```
input= tf.placeholder("float",[None,Input_Neuronen])
```

[8] Mit float wird der Datentyp des inputs für die einzelnen Neuronen gewählt. Die Variable Input_Neuronen gibt die Anzahl der Input Neuronen an. Das None steht für die Anzahl der Trainingsdaten, die später noch dynamisch eingefügt wird. So muss man sich zunächst nicht festlegen wie viele Trainingsdaten man benutzen will.[9]

[4] [8] S.341
[5] [7] S.384
[6] [7] S.124
[7] [7] S.99
[8] [9] S.218
[9] [7] S.377

2.3 Gewichte

Man kann alle Gewichte zwischen Inputschicht und der ersten versteckten Schicht als Gewichtsmatrix

$$W^{(1)} := \begin{pmatrix} {w_{1,1}}^{(1)} & {w_{1,2}}^{(1)} & \cdots & {w_{1,m_1}}^{(1)} \\ {w_{2,1}}^{(1)} & {w_{2,2}}^{(1)} & \cdots & . \\ {w_{3,1}}^{(1)} & {w_{3,2}}^{(1)} & \cdots & . \\ \vdots & \vdots & \vdots & \vdots \\ {w_{n,1}}^{(1)} & {w_{n,2}}^{(1)} & \cdots & {w_{n,m_1}}^{(1)} \end{pmatrix} \tag{2.1}$$

auffassen.

Die Zeilen von $W^{(1)}$ entsprechen allen ausgehenden Verbindungen für ein Neuron aus der Input-Schicht. Die Spalten entsprechen den eingehenden Verbindungen für ein Neuron aus der versteckten Schicht.

Der Tensorflow Befehl um die Gewichte zwischen zwei Schicht zu Initialisieren lautet:

```
init = tf.truncated_normal(n_inputs, n_neurons)
W =tf.Variable(init,name="weights")
```

[10] Damit werden die Gewichte mit zufälligen Gewichten belegt und es wird eine Matrix bzw Tensor der Dimension (input_neuronen×versteckte_neuronen) erzeugt.
Um den Output des Neuronalen Netzes zu berechnen führt man den Befehl tf.matmul(**input**,W) eine Matrix Multiplikation zwischen Input-Tensor und Gewichtsmatrix durch.[11]
Auf diese Weise bekommt man einen weiteren Tensor der für jedes versteckte Neuron einen Wert hat. Auf diese Werte wendet man nun eine Aktivierungsfunktion an, das Ergebnis nennt man **Aktivität**[12] der versteckten Schicht. Dieses Verfahren kann man nun für beliebig viele versteckte Schichten wiederholen, dabei nutzt man die Aktivität und die Gewichtsmatrix für die zweite versteckte Schicht um eine Matrix Multiplikation durchzuführen und die Aktivität der nächsten Schicht zu bekommen.

[10] [7] S.377
[11] [7] S.377
[12] [10] S.247

2.4 Aktivierungsfunktionen

Aktivierungsfunktionen werden verwendet um den Wertebereich den die Neuronen annehmen können einzugrenzen. So liegen manche Aktivierungsfunktionen nur zwischen den Werten -1 und 1 oder bilden alle negative Werte auf Null ab. Exemplarisch werden 3 der beliebtesten Aktivierungsfunktionen beschrieben.

2.4.1 Rectified linear unit function

Eine beliebte Möglichkeit ist die Rectified linear unit function, kurz relu.
Für relu gilt in der parametrisierten Form[13]

$$\sigma(x) = \begin{cases} 0 & \text{falls} \quad x <= 0 \\ ax & \text{sonst} \end{cases}$$

a ist dabei frei wählbar und kann an das jeweilige Beispiel angepasst werden. Obwohl die Relufunktion eine sehr einfache fast lineare Funktion ist, erweist sie sich als sehr Leistungsfähig. Sie dient als Standardaktivierungsfunktion, die für die meisten FeedForward Netzwerke empfohlen wird.[14] In Tensorflow ist die Relu Funktion auf verschiedene Arten implementiert, die oben beschriebene Standard Relu Funktion allerdings nur für den Parameterwert a=1. Sie wird mit

```
tf.nn.relu(features, name=None)
```

[15] eingebunden. Eine andere in tensorflow verwendete Versionen der relu Funktion ist

```
tf.nn.relu6(features, name=None)
```

[9] Für diese gilt:

$$\sigma(x) = \begin{cases} 0 & \text{falls} \quad x <= 0 \\ x & \text{falls,} \quad 0 < x < 6 \\ 6 & \text{sonst} \end{cases}$$

Sie kann schneller berechet werden und hat den Vorteil das weder Werte nahe der Null verschwinden, noch die Werte zu groß werden können.[9]

[13] [5]S.174
[14] [5]S.175
[15] [9]S.58f.

2.4.2 Sigmoidfunktion

Eine andere der Standardaktivierungsfunktionen ist die Sigmoid Funktion. Sie eignet sich gut für das lernen mittels Backpropagation, welches ein sehr häufig eingesetztes Lernverfahren ist.[16]
Die Sigmoidfunktion hat folgende Form:[11]

$$\sigma_c(x) = \frac{1}{1 + e^{-cx}} \tag{2.2}$$

In Tensorflow ist sie nur mit dem Parameterwert $c = 1$ enthalten.

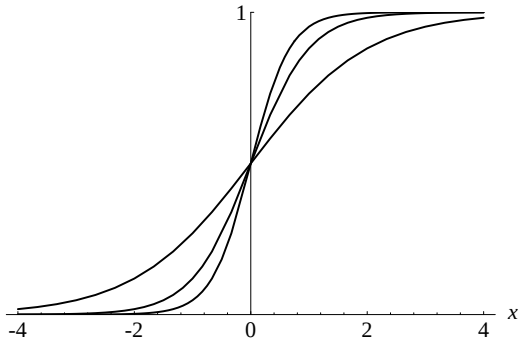

Abbildung 2.2: Sigmoidfunktion für c=1 [11]

Der Wertebereich der Sigmoidfunktion liegt zwischen 0 und 1, was sich sehr gut eignet falls die Ausgabewerte des Netzwerks ebenfalls in diesem Bereich liegen. In Tensorflow ist sie mit dem Befehl tf.sigmoid(x, name=None)[17] definiert.

2.4.3 Tangenshyperbolicus

Der Tangenshyperbolicus ist definiert als[18]

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.3}$$

Die Ableitungen der Sigmoidfunktion und des Tangenshyperbolicus sind leicht zu berechnen, weshalb sie sich gut zur Berechnung des Gradienten der Fehlerfunktion eignen.
Wie in der Abbildung ersichtlich wird, befindet sich der Wertebereich des Tangenshyperbolicus im Intervall von [-1,1]. Es wird mit f.tanh(x, name=None)[19] aufgerufen und eignet

[16] [11] S.151f.
[17] [12]S.175
[18] [6]S.127
[19] [12]S.175

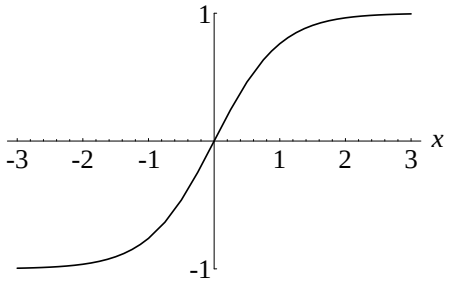

Abbildung 2.3: Graph des Tangenshyperbolicus [11]

sich besonders gut, wenn man in den Lernbeispielen auch negative Zahlen berücksichtigen will.

2.5 Kostenfunktionen

Kosten- oder Fehlerfunktionen sind ein Maß dafür wie gut ein Neuronales Netz lernt. Es stellt eine berechenbare Formel bereit, die den durch das Netz berechneten Output für ein Trainingsbeispiel mit dem zu lernenden Output vergleicht. Außerdem werden sie für das lernen des Neuronales Netz benötigt, da aus den Ableitungen der jeweiligen Kostenfunktion für das zugehörige Gewicht die neuen Gewichte gebildet werden.[20]

2.5.1 MSE

Eine mögliche Kostenfunktion ist der Mean Squared Error, kurz MSE.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{o}_i - o_i)^2. \tag{2.4}$$

[21] $\hat{o}$ steht für den gewünschten Output und o für den vom Netz für den zugehörigen Input Vektor generierten Output.
Der MSE wird mit Kosten=tf. losses .mean_squared_error(labels, predictions)[22] eingebunden und summiert die Quadrate der Abweichungen auf, d.h. je geringer der MSE wird, desto genauer hat das Netz die Trainingsdaten gelernt.

[20] [5]S.177f.
[21] [11]S.156
[22] [9]S.94f.

2.5.2 cross entropy

Eine andere Möglichkeit ist die sogenannte Cross Entropy Funktion.[23]

$$CE = -\frac{1}{N}\sum_{i=1}^{N} \hat{o}_i \ln o_i + (1-\hat{o}_i)\ln(1-o_i) \tag{2.5}$$

Der Vorteil der cross entropy Funktion besteht daran, dass sie je schneller lernt je größer der anfängliche Fehler ist. Fängt man also bei sehr ungünstig gewählten zufälligen Gewichten mit dem Lernprozess an, wird cross entropy schneller bessere Ergebnisse liefern als der MSE.[24] Welche man letztendlich verwendet hängt vom zugrunde liegenden Problem ab.
Cross entropy wird in Tensorflow mit dem Befehl

```
Kosten=tf.nn.softmax_cross_entropy_with_logits()
```

verwendet.[25]

2.6 Lernprozess

Damit das Netz lernt müssen Stück für Stück die Gewichte angepasst werden. Zu diesem Zweck berechnet man die Ableitung der Kostenfunktion und benutzt sie um die Gewichte abzuändern. Dieses Verfahren wird "Backpropagation of Error" mittels "Gradient Descent" - dem Gradientabstiegsverfahren genannt.
Das Update der Gewichte funktioniert nach folgender Regel:[26]

$$w_{i,j} = w_{i,j} - \frac{\partial Kosten}{\partial w_{i,j}}\eta, \tag{2.6}$$

wobei η die Lernrate darstellt. Diese ist frei wählbar, üblicherweise liegt sie im Bereich von 0.01 und 0.5.
Die optimale Lernrate für das gegebene Problem muss experimentell bestimmt werden, da man dazu im Vorfeld keine genauen Vorhersagen machen kann. Trainiert wird das Netzwerk letztendlich mit dem Befehl `tf.train.GradientDescentOptimizer(Lernrate).minimize(Kosten)`[27]

[23] [13]
[24] [13]
[25] [9]S.88
[26] [11]S.157
[27] [12]S.156

Kapitel 3

3 Das Framework TensorFlow

3.1 Eine Einführung zu TensorFlow

3.2 Der Weg des Frameworks zur Open Source Software

3.3 Angesprochene Zielgruppe

3.4 Hard- und Software Anforderungen

3.4.1 Hardware Anforderungen

3.4.2 Software Anforderungen

3.5 Softwarearchitektur von TensorFlow

Kapitel 4

4 Der Allgemeine Workflow in TensorFlow

4.1 Die Tooling Pipeline

4.2 Vorgehensweise beim Trainingsprozess

4.3 Die Graphelemente im Datenfluss

4.4 Die Visualisierungsmöglichkeiten mit TensorBoard

Kapitel 5

5 Ausblick

Abbildungsverzeichnis

2.1 Feed Forward Netz 4
2.2 Sigmoidfunktion für c=1 [11] 8
2.3 Graph des Tangenshyperbolicus [11] 9

Tabellenverzeichnis

Anhang

Literaturverzeichnis

[1] Russell, Stuart ; Norvig, Peter ; Kirchner, Frank: *Künstliche Intelligenz: Ein moderner Ansatz*. 3., aktualisierte Aufl. München : Pearson Higher Education, 2012 (Always learning). – ISBN 978–3–86894–098–5

[2] TURING, A. M.: I.—COMPUTING MACHINERY AND INTELLIGENCE. In: *Mind* LIX (1950), Nr. 236, S. 433–460. http://dx.doi.org/10.1093/mind/LIX.236.433. – DOI 10.1093/mind/LIX.236.433. – ISSN 0026–4423

[3] Russell, Stuart ; Dewey, Daniel ; Tegmark, Max: Research Priorities for Robust and Beneficial Artificial Intelligence. (2015), jan. http://futureoflife.org/data/documents/research_priorities.pdf

[4] Barrat, James: *Our Final Invention: Artificial Intelligence and the End of the Human Era*. THOMAS DUNNE BOOKS, 2013 http://www.ebook.de/de/product/20253628/james_barrat_our_final_invention_artificial_intelligence_and_the_end_of_the_human_era.html. – ISBN 0312622376

[5] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. MIT Press, 2016. – http://www.deeplearningbook.org

[6] Bishop, Christopher M.: *Neural Networks for Pattern Recognition*. Oxford : Clarendon Press, 1995. – ISBN 978–0–198–53864–6

[7] Géron, A.: *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017 https://books.google.de/books?id=bRpYDgAAQBAJ. – ISBN 9781491962244

[8] Rashid, Tariq: *Make Your Own Neural Network*. 1st. USA : CreateSpace Independent Publishing Platform, 2016. – ISBN 1530826608, 9781530826605

[9] McClure, N.: *TensorFlow Machine Learning Cookbook*. Packt Publishing, 2017 https://books.google.de/books?id=LVQoDwAAQBAJ. – ISBN 9781786466303

[10] Ertel, Wolfgang: *Grundkurs Künstliche Intelligenz - Eine praxisorientierte Einführung*. Berlin Heidelberg New York : Springer-Verlag, 2013. – ISBN 978–3–834–82157–7

[11] Rojas, Raul ; Varga, Peter: *Neural Networks - A Systematic Introduction*. Berlin Heidelberg : Springer Science, Business Media, 1996. – ISBN 978–3–540–60505–8

[12] Bonnin, R.: *Building Machine Learning Projects with TensorFlow*. Packt Publishing, 2016 https://books.google.de/books?id=pZ3cDgAAQBAJ. – ISBN 9781786466822

[13] Michael, Nielsen: Improving the way neural networks learn. (2017), Aug. http://neuralnetworksanddeeplearning.com/chap3.html

[14] Abadi, Martín ; Barham, Paul ; Chen, Jianmin ; Chen, Zhifeng ; Davis, Andy ; Dean, Jeffrey ; Devin, Matthieu ; Ghemawat, Sanjay ; Irving, Geoffrey ; Isard, Michael ; Kudlur, Manjunath ; Levenberg, Josh ; Monga, Rajat ; Moore, Sherry ; Murray, Derek G. ; Steiner, Benoit ; Tucker, Paul ; Vasudevan, Vijay ; Warden, Pete ; Wicke, Martin ; Yu, Yuan ; Zheng, Xiaoqiang: *TensorFlow: A system for large-scale machine learning*. https://www.tensorflow.org/. Version: 2016. – Software available from tensorflow.org

[15] Abadi, Martín ; Agarwal, Ashish ; Barham, Paul ; Brevdo, Eugene ; Chen, Zhifeng ; Citro, Craig ; Corrado, Greg S. ; Davis, Andy ; Dean, Jeffrey ; Devin, Matthieu ; Ghemawat, Sanjay ; Goodfellow, Ian ; Harp, Andrew ; Irving, Geoffrey ; Isard, Michael ; Jia, Yangqing ; Jozefowicz, Rafal ; Kaiser, Lukasz ; Kudlur, Manjunath ; Levenberg, Josh ; Mané, Dan ; Monga, Rajat ; Moore, Sherry ; Murray, Derek ; Olah, Chris ; Schuster, Mike ; Shlens, Jonathon ; Steiner, Benoit ; Sutskever, Ilya ; Talwar, Kunal ; Tucker, Paul ; Vanhoucke, Vincent ; Vasudevan, Vijay ; Viégas, Fernanda ; Vinyals, Oriol ; Warden, Pete ; Wattenberg, Martin ; Wicke, Martin ; Yu, Yuan ; Zheng, Xiaoqiang: *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. https://www.tensorflow.org/. Version: 2015. – Software available from tensorflow.org