HOCHSCHULE
DER MEDIEN

Deep Learning

Google TensorFlow

EDV-Nr: 253504A Aktuelle Themen

Dozent: Prof. Dr. Andreas Koch

Student:
Alexander Schübl
As333@hdm-stuttgart.de
Computer Science and Media
Sommersemester 2018

Inhalt

1 Einleitung

Die Künstliche Intelligenz umfasst ein riesiges und brandaktuelles Themengebiet mit nahezu unbegrenzten Möglichkeiten. Während der Vorlesung *„Aktuelle Themen: Künstliche Intelligenz und ihre Auswirkungen auf Ihre Zukunft*“ wurden unter anderem die aktuellsten Entwicklungen der Künstlichen Intelligenz (*kurz: KI*) aufgezeigt und diskutiert. In dieser Ausarbeitung wird darauf eingegangen, wie solch künstliche Intelligenzen überhaupt entwickelt werden können. Dazu wird das von Google entwickelte Framework *TensorFlow* untersucht und in der Theorie erläutert. Im ersten Schritt wird allerdings der Begriff *Deep Learning* näher erklärt und daraufhin der Übergang zu *TensorFlow* vorgenommen.

2 Deep Learning

Deep Learning (*kurz: DL*) ist lediglich ein kleines Teilgebiet der KI. Es findet sich im Gebiet des Machine Learnings und den damit verbundenen Neuronalen Netzen wieder. Obwohl DL ein Teilgebiet des Maschine Learnings ist, lässt es sich dennoch vom reinen maschinellen Lernen abgrenzen. Beim maschinellen Lernen greift der Mensch in die Analyse der Daten und auch in den Entscheidungsprozess mit ein. Beim DL muss der Mensch lediglich dafür sorgen, dass die Informationen zum Lernen und Trainieren eines Models bereitstehen. Der eigentliche Lernprozess wird von dem DL-Algorithmus übernommen, welcher sich auch stets selbstständig optimiert.

Die neuronalen Netze, welche für das maschinelle Lernen verwenden werden, teilen sich dabei in die einfachen künstlichen neuronalen Netze und die für das Deep Learning verwendeten tiefen neuronalen Netze (*kurz: DNNs)* auf. Diese neuronalen Netze versuchen das menschliche Gehirn in Form von Neuronen nachzubilden, um selbstständig anhand von Gewichten komplexe Sachverhalte erlernen zu können. Ein einfaches künstliches neuronales Netz verfügt hierbei in der Regel lediglich über eine verborgene Schicht, in welcher Gewichte angepasst werden, somit der Lernprozess optimiert und ein Merkmal extrahiert werden kann (*Abbildung 1*).

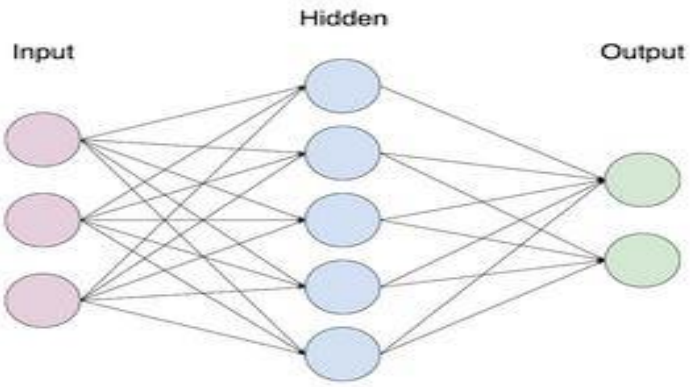


Abbildung 1: Einfaches neuronales Netz

Einfache neuronale Netze eignen sich demnach nicht um äußerst komplexe Sachverhalte erlernen zu können. Je komplexer das Problem, desto mehr verborgene Schichten müssen verwendet und somit auch viel mehr Gewichte trainiert werden. Hierbei kommen die tiefen neuronalen Netze, welche für das Deep Learning verwendet werden, ins Spiel.

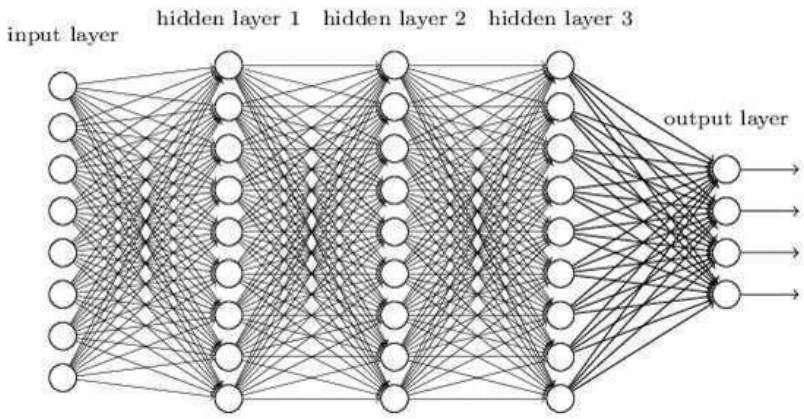


Abbildung 2: Tiefes neuronales Netz

Für das Deep Learning existieren eine Vielzahl an unterschiedlichen Netzwerkarchitekturen, beispielsweise RNNs, CNNs oder GANs. Diese werden in Kapitel 3.3 näher erläutert. In Abbildung 2 ist eine Beispielarchitektur eines tiefen neuronalen Netzes vereinfacht abgebildet. Dieses könnte beispielsweise zur Bilderkennung genutzt werden. In den Input Layer werden die einzelnen Pixel eines Bildes eingegeben. In der ersten verborgenen Schicht könnten demnach lediglich die Farbwerte und in der zweiten verborgenen Schicht die konkreten Kanten und Linien extrahiert werden. Von jeder Schicht werden dabei die Informationen der vorherigen Schichten abstrahiert, bis letztendlich in der Ausgabeschicht beispielsweise ein Gesicht erkannt werden kann.

3 TensorFlow

3.1 Intro

Bei TensorFlow handelt es sich um eine von Google entwickelte Open Source Programmbibliothek, bzw. Framework zum Trainieren und Erstellen von tiefen neuronalen Netzen. Eine der Besonderheiten von TensorFlow ist hierbei die Unterstützung von GPU-Computing auf NVIDIA-Grafikkarten. Da Grafikkarten generell sehr leistungsstark für Matrixberechnungen sind, bietet TensorFlow die Möglichkeit des Trainings von Modellen auf neuartigen NVIDIA-Grafikkarten GPUs an. Die Hochschule der Medien verfügt beispielsweise über zwei Deep Learning Rechner, welche jeweils mit vier NVIDIA Titan V Grafikkarten ausgestatten sind. Das Training von Modellen kann demnach über SSH auf diesen Deep Learning Rechnern durchgeführt werden. TensorFlow wird hauptsächlich mit der Programmiersprache *Python* genutzt. TensorFlow wird bereits in sämtlichen Branchen wie beispielsweise der Medizin, Industrie oder dem Finanzsektor angewendet. Außerdem setzen viele bekannte Unternehmen auf die Verwendung von TensorFlow. Einige bekannte Vertreter hierbei sind Dropbox, Airbnb, Uber, Ebay und Airbus.

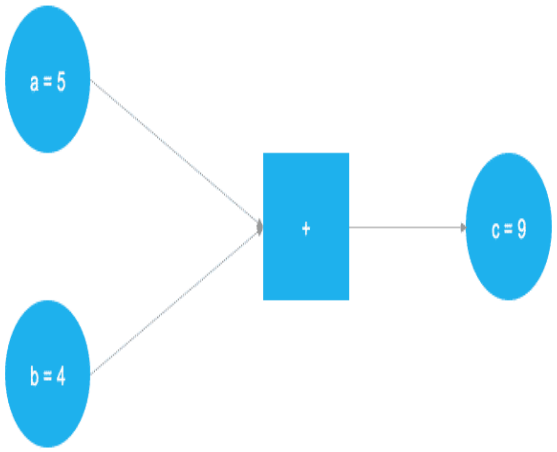
3.2 Funktionsweise

Der Name TensorFlow leitet sich von den Begriffen „*Tensor*“ und „*Flow*“ ab.
TensorFlow führt sämtliche Berechnungen auf Basis von Tensoren ab und baut dabei stets einen zyklenfreien gerichteten Graphen auf, welcher die einzelnen Berechnungen in einem Fluß, sprich „flow“, abarbeitet.
Bei einem Tensor handelt es sich um ein mehrdimensionales Datenarray.
Ein Tensor erster Ordnung stellt dabei einen simplen Vektor dar, ein Tensor zweiter Ordnung eine zweidimensionale Matrix und ein Tensor dritter Ordnung eine dreidimensionale Matrix und so weiter.
Die Tensoren können dabei über einen Input in den von TensorFlow aufgebauten Graphen eingespeist werden. Ein kleines Beispiel, welche Arten von Tensoren eingespeist werden können:

- 1-dim. Tensoren: Ein Sampling gesprochener Sprache, sprich die Klangwerte stellen den Eingabevektor dar.
- 2-dim. Tensoren: Schwarz-weiß-Bildausschnitte als Eingabepixelmatrix
- 3-dim. Tensoren: Farbbilder mit RGB-Werten als dreidimensionale Eingabepixelmatrix

Ein weiterer wichtiger Bestandteil von TensorFlow sind *Placeholder.* Diese dienen dem Definieren von Konstanten, welche TensorFlow übergeben werden. Platzhalter ermöglichen eine einheitliche Abarbeitung des Graphen, unabhängig vom Wert des Platzhalters beziehungsweise des Inputs, da dieser erst zur Laufzeit eingespeist wird und während des Trainings immer wieder mit neuen Werten belegt werden kann.
Darüber hinaus bietet TensorFlow eine Vielzahl von weiteren für das maschinelle Lernen notwendigen Methoden zum Training, wie zum Beispiel die Reduzierung von Kostenfunktionen durch Optimierer.

In Abbildung 3 ist ein sehr vereinfachter von TensorFlow aufgebauter Graph und der dazugehörende Programmcode ersichtlich.




```
# TensorFlow laden
import tensorflow as tf

# a und b als Platzhalter definieren
a = tf.placeholder(dtype=tf.int8)
b = tf.placeholder(dtype=tf.int8)

# Die Addition definieren
c = tf.add(a, b)

# Den Graphen initialisieren
graph = tf.Session()

# Den Graphen an der Stelle c ausführen
graph.run(c, feed_dict={a: 5, b: 4})
```

Abbildung 3: Beispielgraph und Programmbeispiel

Zuerst werden die beiden Konstanten *a* und *b* als Platzhalter definiert.
Durch den Befehl *tf.add(a, b)* wird eine Addition der beiden zuvor definierten Platzhalter durchgeführt und deren Ergebnis in der Variablen *c* abgespeichert.
Mit *tf.Session()* muss zunächst der Graph initialisiert werden, bevor mit *graph.run(c, feed_dict={a: 5, b: 4})* der Graph nun von Anfang bis zu der Variablen *c* abgearbeitet wird.
Der darin enthaltene Befehl *feed_dict={a: 5, b:4}* speist dabei die aktuellen Werte für *a* und *b* in den Graphen ein.
Bei einem zukünftigen Programmablauf könnten andere Werte für *a* und *b* eingespeist werden um einen anderen Wert für *c* zu erlangen.
Durch das Arbeiten mit Platzhaltern ändert sich der Aufbau des Graphen dabei niemals.

3.3 Anwendungsbeispiele

Mithilfe von TensorFlow lassen sich Modelle für sämtliche bekannte Machine-Learning-Algorithmen erstellen und Trainieren. Die Anwendungsgebiete hierfür sind demnach nahezu unbegrenzt und teilen sich unter anderem in die folgenden Gebiete ein:

- Objekterkennung (z.B. CNNs)
- Objekterzeugung (z.B. GANs)
- Vorhersagen (z.B. RNNs)

Hervorzuheben hierbei ist, dass verschiedene Netzwerkarchitekturen auch für verschiedene Anwendungszwecke genutzt werden können. So können RNNs zum Beispiel nicht nur für Vorhersagen, sondern unter anderem auch zur Objekterkennung genutzt werden.

Im Folgenden werden die Netzwerkarchitekturen CNN, GAN und RNN kurz erläutert.

3.3.1 Convolutional Neural Networks (CNN)

CNNs werden in erster Linie zur Bilderkennung genutzt. Dies ist aktuell ein sehr großes Thema, da hierauf auch beispielsweise beim Autonomen Fahren gesetzt wird, um zum Beispiel Hindernisse auf der Fahrbahn zu erkennen, die Spur zu halten oder Straßenschilder zu erkennen. Ein CNN besteht in der Regel aus einer Konstellation von hintereinander geschalteten Convolutional,- Pooling und Denselayern. Dabei folgt in der Regel auf jeden Convolutional-Layer ein Pooling-Layer und am Ende der Netzwerkarchitektur ein oder mehrere Dense-Layer. Bei CNNs wird nach dem Prinzip „Feature Extraction & Classification“ gearbeitet. Dies bedeutet, dass mit jeder Convolutional-Pooling-Layer Kombination ein spezielles Feature des Bildes extrahiert wird, die Informationen innerhalb des Netzwerkes weitergereicht werden und erst am Ende die eigentliche Klassifikation stattfindet.

Für ein besseres Verständnis werden im Folgenden die Schichten kurz erläutert.

Convolutional Layer

Beim Convolutional Layer wird über ein Inputbild, beziehungsweise dessen Pixelwerte, ein oder mehrere Filter gelegt und anhand der Filterkoeffizienten für jeden Bildbereich eine Summe gebildet. Diese Summe gibt im Endeffekt an, ob sich ein bestimmtes Feature innerhalb des Bildbereiches befindet, wie beispielsweise eine Ecke oder eine Kante. Für jedes einzigartige Feature wird hierbei ein eigener Filter über das Bild geschoben.

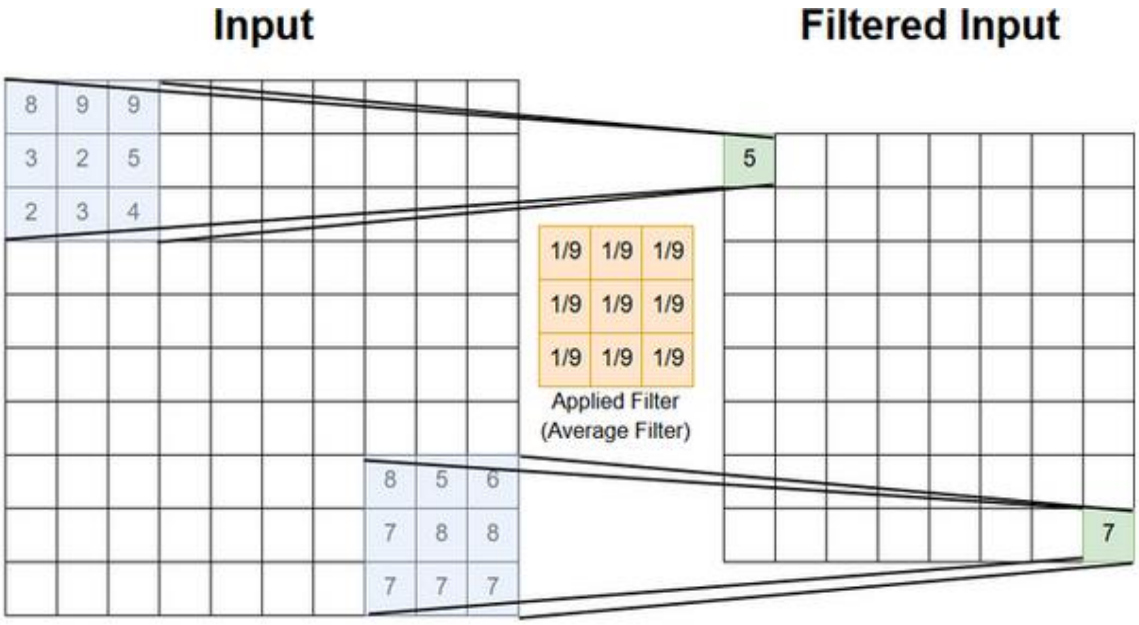


Abbildung 4: Convolutional Layer

Das Inputbild in Abbildung 4 ist beispielsweise ein 10x10 schwarzweiß oder RGB Bild.

Pooling Layer

Die Eingabe eines Pooling Layers ist stets die Ausgabe des vorhergegangen Convolutional Layers. Beim Pooling wird anhand von Min-, Max, -oder Meanpoolings die Komplexität des Netzwerkes reduziert, indem aus jedem Filterbereich jeweils nur ein bestimmter Wert übernommen wird. Dabei gehen zwar Informationen über die Bildposition des jeweiligen Features geringfügig verloren, aber in der Regel ist bei CNNs lediglich relevant, ob sich das jeweilige Feature innerhalb des Bildbereichs befindet und nicht exakt wo.

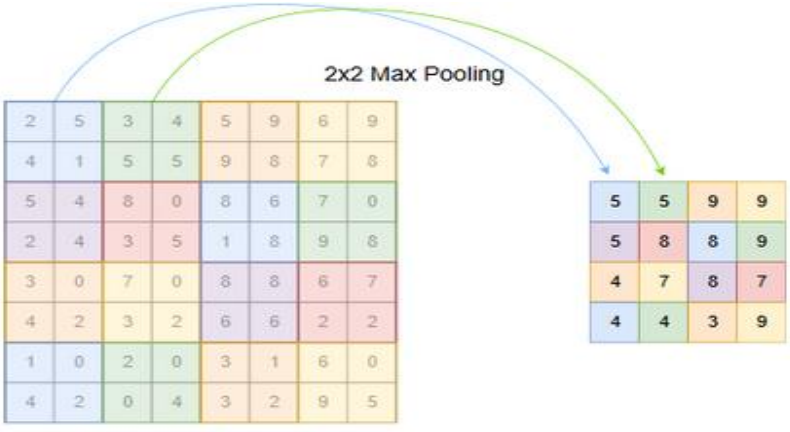


Abbildung 5: Pooling-Layer

In Abbildung 5 wird das standartgemäße Max-Pooling angewandt um die Komplexität zu reduzieren. Dabei wird aus jedem Filterbereich jeweils das Maximum extrahiert und anschließend dem nächsten Layer zur Verfügung gestellt.

Dense Layer

Der oder die Dense Layer stehen am Ende eines CNNs und stellen den eigentlichen Klassifikator zur Klasseneinteilung oder Regressor für numerische Werte dar. Die Ausgabe des letzten Pooling Layers wird dabei als Eingabe für den Dense Layer verwendet. Je nachdem ob eine Klassifizierung oder eine Regression durchgeführt werden soll, befinden sich ein oder mehrere Ausgabeneuronen im Dense Layer. Die Ausgabe ist demnach die vom CNN erkannte zugehörige Klasse oder im Fall einer Regression eine Zahl. Bei einem Dense Layer handelt es sich demnach um ein Single-Layer Perzeptron oder ein Multi-Layer Perzeptron.

3.3.2 Generative Adversarial Networks (GAN)

GANs werden in erster Linie zum Erzeugen von Bildern oder zur Manipulation von Bildern verwendet. So lassen sich dadurch beispielsweise täuschend echt wirkende, aber falsche Bilder erzeugen. Ein prominenter Sammelbegriff hierfür sind die sogenannten DeepFakes, mit welchen sich die Gesichter von Prominenten auf Bild und Video austauschen lassen.

Ein GAN besteht aus zwei tiefen neuronalen Netzen – einem Diskriminator und einem Generatornetzwerk. Der Generator erzeugt dabei stets Bilder, welche einem bestimmten Datensatz ähnlichsehen sollen. Der Diskriminator muss nun klassifizieren, ob es sich bei dem Bild um ein echtes Bild oder ein erzeugtes Generatorbild handelt. Dementsprechend konkurrieren die beiden Netzwerke, da der Generator stets seine Performance erhöhen möchte, dazu allerdings die Performance des Diskriminators verringert beziehungsweise dieser ausgetrickst werden muss.

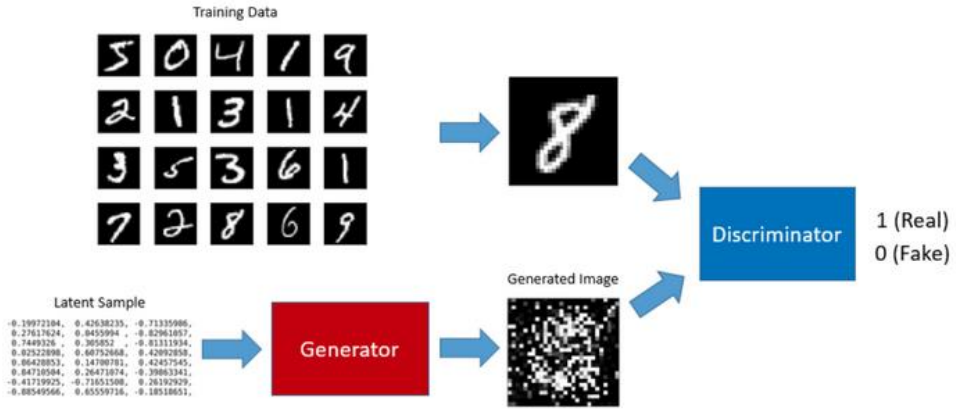


Abbildung 6: GAN Beispiel MNIST Datensatz

In Abbildung 6 werden Trainingsdaten aus dem MNIST-Datensatz dem Diskriminator übergeben. Der MNIST-Datensatz umfasst dabei 70.000 handgeschriebene Buchstaben von 0 bis 9. Der Generator erhält am Anfang des Trainings lediglich Gleitkommazahlen zwischen -1 und 1, woraus ein verrauschtes Bild entsteht. Der Diskriminator klassifiziert dieses als Fake, wodurch der Generator die Rückmeldung erhält, dass dieser seinen Prozess optimieren muss.

3.3.3 Recurrent Neuronal Networks (RNN)

RNNs haben ein umfangreiches Einsatzgebiet. So lassen sich unter anderem Vorhersagen treffen, automatische Übersetzer trainieren oder Image Captioning durchführen. Unter Image Captioning versteht man das automatische Generieren einer textuellen Beschreibung zu einem gegebenen Bild.
Die Besonderheit von RNNs ist, dass auf Grund ihrer internen Vernetzung diese über eine Art Gedächtnis verfügen. Dabei sind bei einem RNN sämtliche Neuronen einer Schicht miteinander oder sogar mit Neuronen von vorherigen Schichten verbunden. Dadurch erhält ein RNN in jeder Schicht Zugriff auf das Wissen aus bereits vorherigen Schichten, was in anderen Modellen wie beispielsweise bei CNNs nicht der Fall ist.

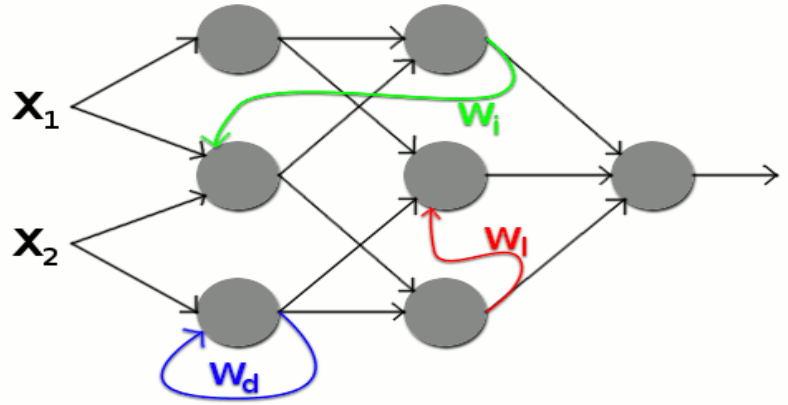


Abbildung 7: RNN Beispiel

Das einfache in Abbildung 7 dargestellte RNN stellt die drei möglichen Arten der Rückkopplung zum Zugriff auf bereits gelerntes Wissen dar. Wd zeigt hierbei eine direkte Rückkopplung auf. Der Output eines Neurons wird erneut dem Input des Neurons zurückgegeben. Wi koppelt dabei den Output eines Neurons auf den Input eines Neurons der vorherigen Schicht zurück, während mit Wl eine seitliche Rückkopplung stattfindet, sprich der Output eines Neurons wird auf den Input eines Neurons der selbigen Schicht gekoppelt.

3.4 Weiteres & Fazit

Neben der lokalen Programmierung von TensorFlow lassen sich auch Modelle in einer Cloud erstellen und trainieren. Beispiele hierfür sind unter anderem die Google ML Cloud oder Amazon AWS Sagemaker. Der Vorteil hierbei ist, dass die Hardware zur Berechnung und zum Training von dem Cloudanbieter bereitgestellt wird und somit ein sehr schnelles Training komplexer Modelle ermöglicht wird.

Außerdem verfügt TensorFlow über das nützliche Visualisierungstool *TensorBoard.* Dabei handelt es sich um eine Webapplikation, mit welcher sich beliebige Parameter eines TensorFlow Programmes, wie beispielsweise die Verringerung der Fehlerfunktion, visuell über die Zeit darstellen lassen oder sich der in der Regel sehr komplexe aufgebaute Graph darstellen lässt. Dadurch erlangt der Nutzer eine sehr nützliche Transparenz über das von ihm entwickelte neuronale Netz.

Mit TensorFlow liefert Google ein sehr effizientes und nützliches Framework zur Erstellung von tiefen neuronalen Netzen. Damit kann potentiell jeder der über

Programmierkenntnisse und Kenntnisse über künstliche Intelligenz verfügt sich eigene intelligente Anwendungen erstellen. Dabei sind die Anwendungsmöglichkeiten von TensorFlow nahezu grenzenlos.

4 Quellenverzeichnis

Sämtliche Quellen wurden zuletzt am 19.09.2018 überprüft.

1. Was ist Deep Learning?
   https://www.bigdata-insider.de/was-ist-deep-learning-a-603129/

2. So funktioniert Google TensorFlow
   https://www.bigdata-insider.de/so-funktioniert-google-tensorflow-a-669777/

3. Einführung TensorFlow
   https://www.statworx.com/de/blog/data-science/einfuehrung-tensorflow/

4. Get Started with TensorFlow
   https://www.tensorflow.org/tutorials/

5. TensorFlow: Der Weg des maschinellen Lernens
   https://jaxenter.de/tensorflow-der-weg-des-maschinellen-lernens-35956

6. Build a Convolutional Neural Network using Estimators
   https://www.tensorflow.org/tutorials/estimators/cnn

7. A Beginners Guide to Generative Adversarial Networks (GANs)
   https://skymind.ai/wiki/generative-adversarial-network-gan

8. Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs
   http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns

9. Image Captioning in Deep Learning
   https://towardsdatascience.com/image-captioning-in-deep-learning-9cd23fb4d8d2

10. Wie Programme lernen
    https://www.heise.de/ix/heft/Wie-Programme-lernen-3807263.html

5 Abbildungsverzeichnis

Sämtliche Bildquellen wurden zuletzt am 19.09.2018 überprüft.

Abbildung 1.
http://jsfeeds.com/details/neural-networks-in-javascript-5a37a3e104959cec4923fa1d

Abbildung 2.
https://datawarrior.wordpress.com/2017/10/31/interpretability-of-neural-networks/

Abbildung 3.
https://www.statworx.com/at/blog/einfuehrung-tensorflow/

Abbildung 4.
http://nbviewer.jupyter.org/urls/maucher.home.hdm-stuttgart.de/nb/ML/ConvolutionNeuralNetworks.ipynb

Abbildung 5.
http://nbviewer.jupyter.org/urls/maucher.home.hdm-stuttgart.de/nb/ML/ConvolutionNeuralNetworks.ipynb

Abbildung 6.
https://towardsdatascience.com/understanding-generative-adversarial-networks-4dafc963f2ef?gi=dec974b1b8a2

Abbildung 7.
http://www.wikiwand.com/de/Rekurrentes_neuronales_Netz