Kai Spriestersbach

NEURONALE NETZE VERSTEHEN – WIE? EINFACH AUSPROBIEREN

Spätestens seit der Ankündigung seitens Google im Oktober 2015, dass deren RankBrain-Algorithmus sogenannte künstliche neuronale Netze für die Verbesserung der Suchergebnisse nutzt, ist das Thema künstliche Intelligenz und maschinelles Lernen in den Fokus der Aufmerksamkeit vieler Online-Marketer gelangt. Aber die dahinter liegenden Konzepte wirklich zu verstehen, ist nicht ganz einfach. Wie immer begreift man Dinge besser, wenn man sie praktisch ausprobieren kann. Das geht jetzt recht einfach und ohne große Vorkenntnisse auf einem von Googles Open-Source-Framework TensorFlow eingerichteten Spielplatz. Dort können Sie ganz einfach direkt in Ihrem Browser mit einem solchen Netzwerk herumexperimentieren und dadurch die Funktionsprinzipien im Detail besser nachvollziehen und damit ein eigenes, besseres Verständnis entwickeln. Quasi neuronale Netze für jedermann – zum Ausprobieren. Und das Beste: Die Nutzung ist völlig kostenlos. Kai Spriestersbach zeigt Ihnen auf, wie Sie das Schritt für Schritt am besten angehen können.

Googles CEO Sundar Pichai verkündete Ende 2015 in der Besprechung der Quartalszahlen mit dem prägnanten Satz „Machine learning is a core transformative way by which we are rethinking everything we are doing“, dass der Suchmaschinengigant sehr großes Potenzial im Einsatz des maschinellen Lernens sieht. Machine Learning eignet sich sowohl zur Verbesserung vorhandener als auch zur Entwicklung vollkommen neuer Produkte. Viele Features und Ansätze werden auf Basis dieser Technologien überhaupt erst möglich. So ist es nicht weiter verwunderlich, dass das Unternehmen aus Mountain View auch über eines der fortschrittlichsten Frameworks für die Erforschung und Entwicklung von Anwendungsszenarien im Bereich des maschinellen Lernens verfügt. Dieses Framework, genannt TensorFlow, wurde einen Monat später unter der Apache-Lizenz 2.0 als Open-Source-Software veröffentlicht und steht seitdem jedermann zur Verfügung.

Googles KI-Spielwiesen

Google hat aus mehreren Gründen großes Interesse daran, dass sich mehr Menschen mit künstlicher Intelligenz beschäftigen. Zum einen soll die Akzeptanz dieser Technologien bei den Anwendern langfristig erhöht werden und zum anderen müssen dringend benötigte Entwickler

DER AUTOR

Kai Spriestersbach ist Online Strategy Consultant bei der eology GmbH sowie Inhaber von SEARCH ONE.

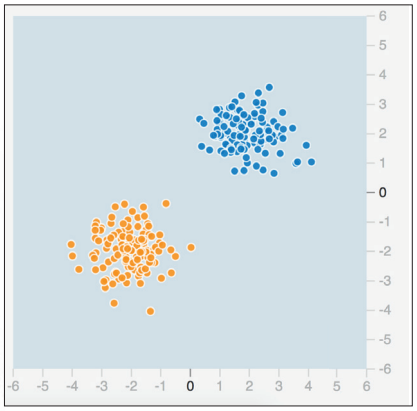

Abb.1: Datensatz für einfache Klassifizierung

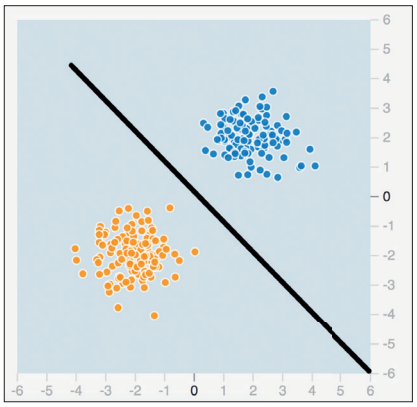

Abb.2: Linienfunktion zu Trennung der Gruppen

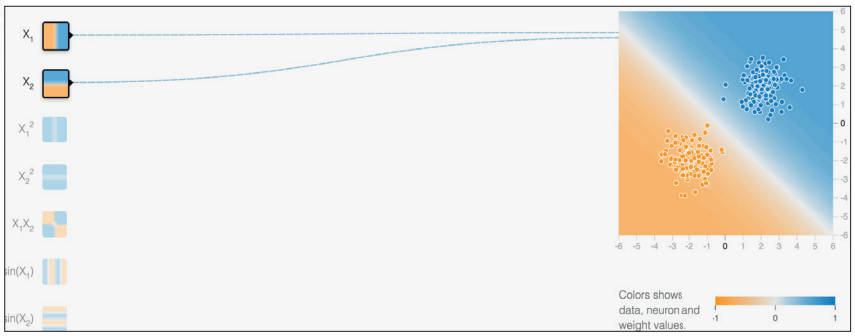

Abb.3: Klassifizierung mittels zweier Eingangsneurone

an den boomenden Technologiezweig herangeführt werden.

Für die normalen Anwender hat Google unter https://aiexperiments.withgoogle.com/ im November 2016 eine KI-Spielwiese mit Experimenten eingerichtet, in denen man die Möglichkeiten künstlicher Intelligenz spielerisch erkunden kann. Leider bleiben die dahinterliegenden Funktionsweisen in diesen zugegebenermaßen sehr interessanten und unterhaltsamen Spielereien dem Nutzer verborgen.

Für interessierte Entwickler, Studenten und Forscher wurde bereits im Juli unter http://playground.tensorflow.org/ ein spezieller „Neural Network Playground" eingerichtet. In dieser JavaScript-Web-App kann man mit einem künstlichen neuronalen Netz (KNN) herumspielen und dadurch ein Verständnis von dessen Arbeitsweise erlangen.

„Vereinfacht gesagt, ist ein neuronales Netz eine Funktion, welches die erwartete Ausgabe mittels Trainingsdaten für bestimmte Eingangswerte erlernt."

Beispiel 1: Klassifizierung von Datenpunkten mit getrennter gaußscher Verteilung

In Abbildung 1 wird ein einfaches Problem der Klassifizierung abgebildet. In dem vorliegenden Datensatz hat jeder Datenpunkt zwei Werte: **x1** (horizontale Achse) und **x2** (vertikale Achse). Es gibt darin zwei Arten von Datenpunkten, die **orangefarbenen** und die **blauen**.

Für dieses relativ unkomplizierte Beispiel könnte man nun einen einfachen Algorithmus schreiben, der mittels diagonaler Linie die beiden Gruppen trennt und an dieser Grenze ermittelt, in welche der Gruppen der jeweilige Datenpunkt gehört. Wie in Abbildung 2 dargestellt, kann die Grenze mit einer einfachen Linienfunktion über die Formel `x1 + x2 > b` bestimmt werden. Für den flexibleren Einsatz wird diese Formel noch um die **Gewichte w1** und **w2** erweitert, sodass sich `w1 x1 + w2 x2 > b` ergibt. Somit kann mittels der Werte für **w1** und **w2** der Winkel der Linie frei festgelegt und mittels des Wertes für b die Position der Linie verändert werden. Diese Formel eignet sich nun für jeden Datensatz, der durch eine gerade Linie in zwei Gruppen getrennt werden kann. Der Programmierer muss nun „nur noch" geeignete Werte für **w1, w2** und **b** finden, um dem Computer zu sagen, wie die Datenpunkte klassifiziert werden sollen. Das neuronale Netz kann diese Werte selbst berechnen, indem es anhand der Trainingsdaten lernt, wo die Grenzen liegen.

Unter *http://einfach.st/tensor2* kann das neuronale Netz nach einem Klick auf den Play-Button dabei beobachtet werden, wie es mit den Eingangswerten **x1** und **x2** die Gewichtungen an zwei Neuronen so lange ändert, bis die Linie im richtigen Winkel steht und die Testdaten korrekt klassifiziert werden. Zu Beginn ändert sich der Winkel der Linie noch sehr schnell, das heißt, das Netzwerk lernt noch sehr schnell, doch es benötigt am Ende mehr als 10.000 Iterationen, um fehlerfrei in Blau und Orange zu unterscheiden (Abbildung 3).

Für dieses Beispiel wäre der Einsatz eines KNN in der Praxis sicher nicht sonderlich effizient. Doch bei nicht linearen Problemen, also wenn die Grenze zwischen den Gruppen nicht durch eine einfache Linie beschrieben werden kann, spielt der selbstlernende Ansatz seine Stärken voll aus. Das Spannende dabei ist, dass die meisten Probleme der echten Welt eben nicht linear sind und die meisten realen Datensätze auch deutlich mehr als zwei Dimensionen umfassen. Dadurch wird die manuelle Erstellung von Algorithmen, Formeln und Gewichtungen extrem aufwendig bis unmöglich.

Beispiel 2: Klassifizierung kreisförmig angeordneter Datenpunkte

Nicht alle Probleme sind linear und damit einfach. Schon am folgenden

DIE GESCHICHTE VON TENSORFLOW

Das TensorFlow-Framework stellt bereits die zweite Generation der Entwicklungswerkzeuge für Machine Learning von Google dar. Der unmittelbare Vorläufer DistBelief wurde 2011 von Jeff Dean und Greg Corrado als Large-scale-deep-learning-Software auf Googles Cloud-Computing-Infrastruktur in einem internen Projekt namens „Google Brain" entwickelt. Nachdem Google den AI-Forscher Geoffrey Hinton für dieses Projekt gewinnen konnte, sorgte dessen neuer Ansatz der sogenannten „Backpropagation" für einen Durchbruch bei der Verbesserung der Ergebnisse. Beispielsweise konnte die Fehlerrate in der bisherigen Spracherkennungssoftware mit einem Schlag um 25 % gesenkt werden. Dean und Hinton schrieben das DistBelief-Framework in eine flexiblere, schnellere, robustere und vor allem einsatzreife Software-Bibliothek um, die anschließend als TensorFlow veröffentlicht wurde. Dieses Framework ermöglicht es, unterschiedliche Ansätze des maschinellen Lernens zu verwenden und ist nicht auf künstliche neuronale Netze festgelegt. Zudem wurde das Framework so konzipiert, dass die Entwicklungs- und Kommunikationsprozesse zwischen Forschern und Entwicklung direkt innerhalb der Anwendung realisiert werden, wodurch sofortige Praxisanwendungen aus neu gewonnenen Erkenntnissen möglich wurden. Letztlich können Anwendungen sogar während der Laufzeit mittels neuer Lerndaten verbessert sowie mehrere Versionen eines Lernmodells parallel getestet werden.

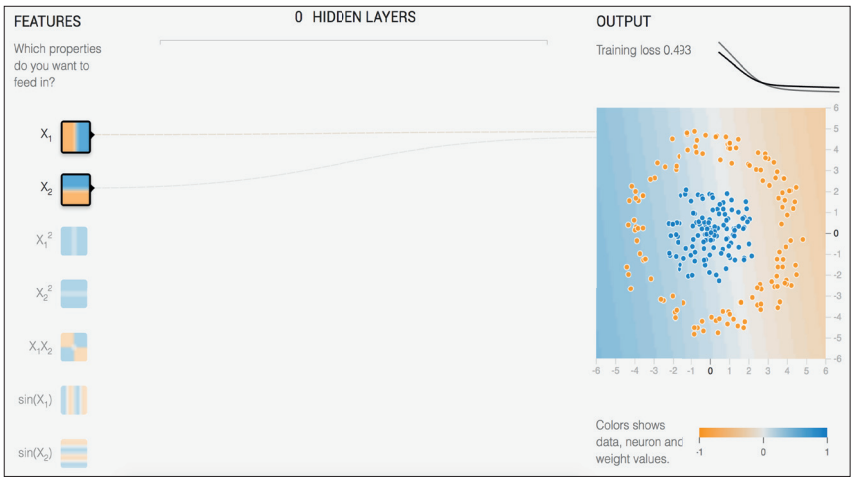

Abb.4: Nicht lineares Problem mit kreisförmig angeordneten Datenpunkten

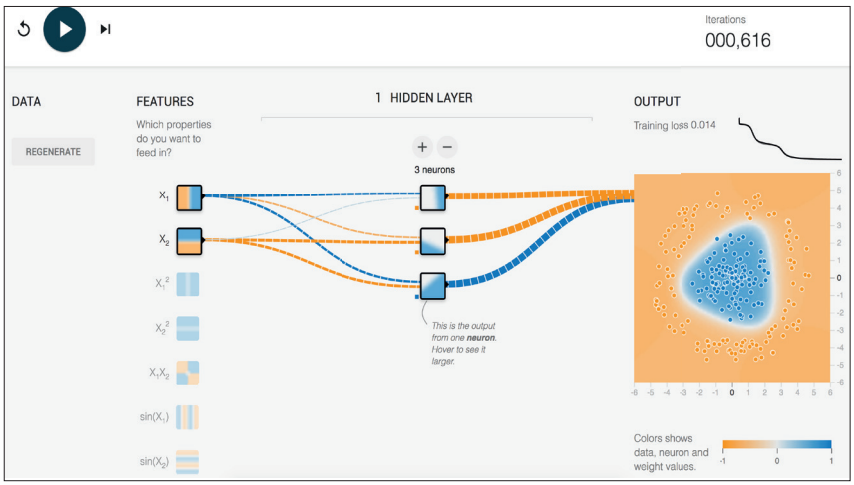

Abb.5: Erfolgreiche Klassifizierung mittels dreier verdeckter Neuronen

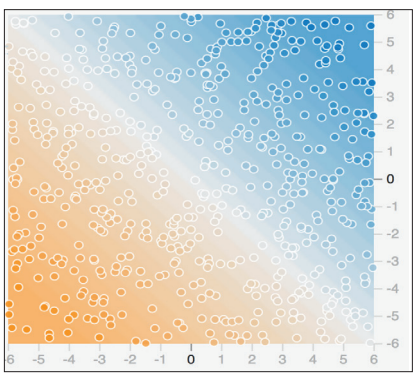

Abb.6: Einfaches lineares Regressionsproblem

Beispiel in Abbildung 4 scheitert der Ansatz, mittels Drehung und Verschieben der Linie eine sinnvolle Grenze festzulegen. Um dieses Problem zu lösen, muss das KNN nun um eine versteckte Schicht von Neuronen erweitert werden, die quasi weitere Linien hinzufügen können.

Fügt man dem Netzwerk nun **drei Neuronen in der verdeckten Schicht** hinzu, gelingt es dem KNN innerhalb weniger Hundert Iterationen, eine Art Dreieck zu finden, also die Kombination aus drei Linien, um den inneren Kreis der blauen Punkte vom äußeren Kreis der orangefarbenen zu trennen (Abbildung 5).

Doch neuronale Netze können nicht nur Klassifizierungen vornehmen. In der Playground-Web-App steht unter Problemstellung (Problem type) auch noch die Regression zur Verfügung. Im Gegensatz zur Klassifizierung, bei der jeder Datenpunkt entweder blau oder orange ist, können die Datenpunkte in der Regression auch jeden Wert dazwischen annehmen. Der einfachste Datensatz (Abbildung 6) kann wiederum mit zwei Eingangsneuronen für **x1** und **x2** ohne verdeckte Schicht gelöst werden.

Beispiel 4: Komplexes Regressionsproblem

Doch der zweite Datensatz, der für die Regression zur Verfügung steht, lässt sich erst mit zwei verdeckten Schichten mit mindestens drei bzw. zwei Neuronen auf Basis der linearen Eingangsneuronen für x1 und x2 lösen. Dabei entstehen quasi sechs offene Dreiecke, die den Datensatz gut, aber nicht fehlerfrei beschreiben (Abbildung 7).

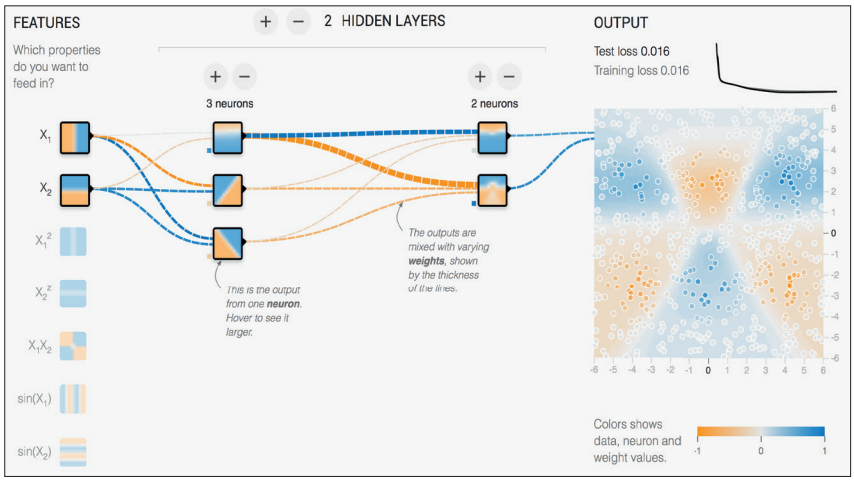

Abb.7: Komplexes Regressionsproblem

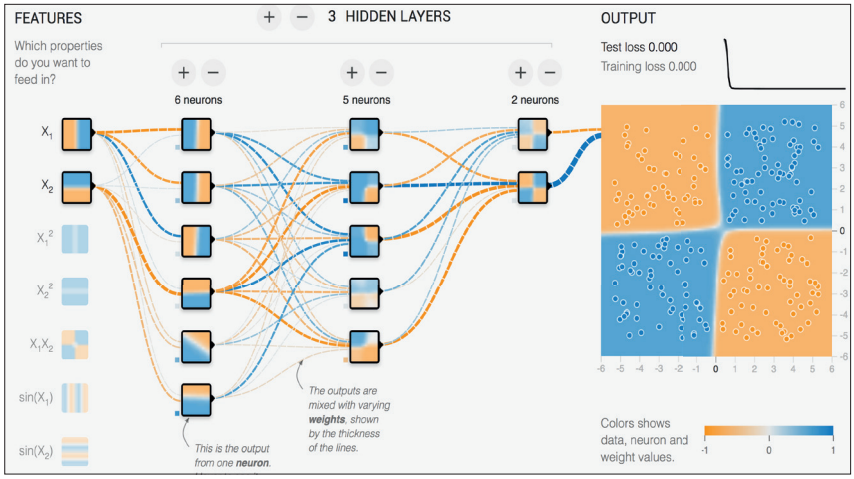

Abb.8: Lösungsvorschlag für das XOR-Klassifizierungsproblem

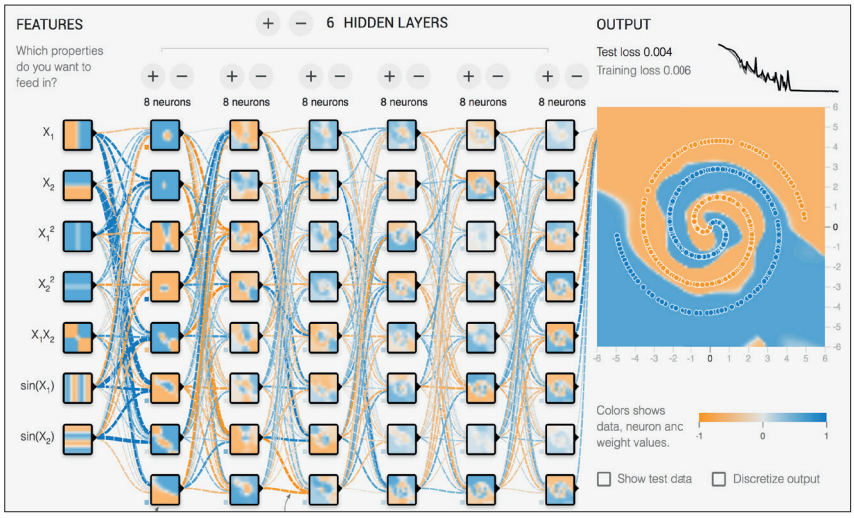

Abb.9: Maximalausstattung des KNN zur Lösung des spiralförmigen Datensatzes

Weitere Datensätze zum Experimentieren

Zusätzlich zu den beiden Datensätzen für die Regression stehen insgesamt vier unterschiedliche Datensätze für das Problem der Klassifizierung zur Verfügung. Der gaußsche Datensatz wurde bereits im ersten Beispiel gelöst (Abbildung 3) und der kreisförmige im zweiten Beispiel (Abbildung 5). Für den Datensatz „exklusives Oder“ (XOR) reichen die linearen Eingangsfunktionen x1 und x2 ebenfalls noch aus. Mittels der Konfiguration in Abbildung 8 wird das Problem durch drei verdeckte Schichten mit sechs, fünf und zwei Neuronen fehlerfrei gelöst.

Zum Problem der spiralförmig angeordneten Datenpunkte, das sich nicht mehr nur mit linearen Eingangsfunktionen lösen lässt, sollten Sie selbst mit den Möglichkeiten des Netzwerks herumexperimentieren. Im einfachsten Falle schalten Sie alle Eingangsfunktionen ein und statten das Netzwerk mit der maximalen Anzahl (sechs) an verdeckten Schichten mit jeweils acht Neuronen pro Schicht aus.

Durch die selbstregulierenden Eigenschaften eines künstlichen neuronalen Netzes werden unnötige Eingangssignale nach und nach durch eine geringe Gewichtung quasi ignoriert und auch unnötige Neuronen werden nach dem Durchlauf der Trainingsdaten nach einigen Hundert Iterationen bereits durch ihre geringe Gewichtung identifizierbar (Abbildung 9). So ist beispielsweise die Eingangsfunktion der Multiplikation (`x1 * x2`) relativ gering gewichtet und kann sogar deaktiviert werden, ohne die Ergebnisqualität entscheidend zu beeinflussen.

Weitere Einstellungsmöglichkeiten des Netzwerks

Für weitere Experimente mit dem Netzwerk lässt sich festlegen, wie viele Prozent des Datensatzes zum Trainieren und wie viele zum Testen des KNN verwendet werden sollen (Ratio of training to test data), wie viel Rauschen es im Datensatz geben soll (Noise) und nach wie vielen Datensätzen (zwischen 1 und 30) das KNN jeweils seine Gewichtungen verändern kann (Batch size). Auch an den Eigenschaften der Neuronen selbst kann gedreht werden. Die Lernrate (Learning rate) wirkt sich auf die

Veränderungsgeschwindigkeit der Gewichtungen aus und kann zwischen 0,00001 und 10 variiert werden. Die Aktivierungsfunktion der Neuronen kann ebenfalls ausgewählt werden.

Kleiner Exkurs: Aktivierungsfunktionen im Detail

Als Aktivierungsfunktionen (Activation) für die Neuronen stehen in der Anwendung neben dem Standardwert Tangens Hyperbolicus (Tanh) noch Sigmoid sowie ReLU und Linear zur Auswahl. Die Variation der Aktivierungsfunktionen führt zu sehr unterschiedlichen Ergebnissen in den Datensätzen und vermittelt ein erstes Verständnis für die Komplexität der Thematik.

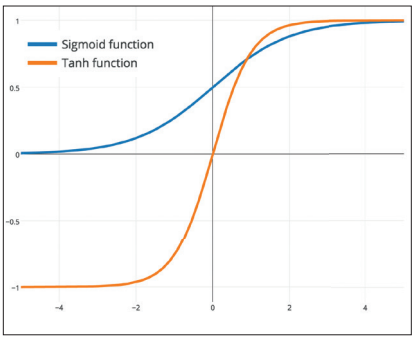

Abb.10: Sigmoide und Tangens Hyperbolicus

Sigmoide und Tangens Hyperbolicus

Der **Tangens Hyperbolicus** ist im Grunde auch eine sigmoide Funktion, da dieser ebenfalls eine S-Kurve beschreibt, jedoch keine negativen Werte liefert (Abbildung 10). **Sigmoide Aktivierungsfunktionen** werden in der Praxis in den meisten Modellen verwendet, die kognitive Prozesse simulieren, da biologische Neuronen in Gehirnen eine ähnliche Charakteristik aufweisen. Im Gegensatz zur linearen Aktivitätsfunktion ist der Aktivitätslevel bei den sigmoiden Funktionen sowohl nach oben als auch nach unten begrenzt. Dies deutet nicht nur auf eine höhere biologische Plausibilität hin (vgl. die begrenzte Intensität des Aktionspotenzials biologischer Neuronen), sondern hat auch den Vorteil, dass die Aktivität im Netz nicht ungewollt „überschwappen" kann und nur noch Fehlerwerte produziert. Durch die vorhandene Horizontalasymptote kommt es jedoch bei der Verwendung von sigmoiden Funktionen häufig zu einem Verschwinden der Übergänge und damit zu einem Lernstillstand in dem jeweiligen Teilbereich des Netzwerks.

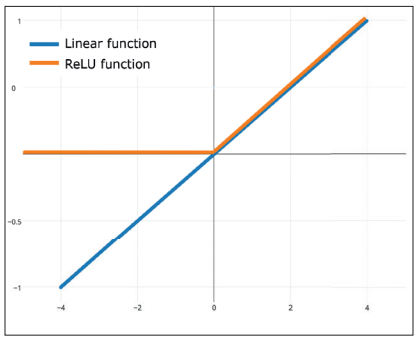

Abb.11: Lineare und ReLU-Funktion

Lineare Funktion und ReLU

Die **lineare Aktivierungsfunktion** ist ideal für die Lösung des linearen Regressionsproblems aus dem Regressionsbeispiel in Abbildung 6. Die sigmoiden Funktionen können diesen linearen Übergang nur annähernd beschreiben, jedoch niemals exakt abbilden. Für nicht lineare Probleme ist die lineare Aktivierungsfunktion jedoch gänzlich ungeeignet, weshalb man diese Aktivierungsfunktion nicht in der Praxis verwendet.

Die **ReLU-Funktion** (rectified linear unit) ist die gängige Aktivierungsfunktion in sogenannten Convolutional Neural Networks (CNN), die beispielsweise bei der Erkennung von Handschriften und Gegenständen in Bildern verwendet werden. Diese Funktion beschreibt eine einfache Gerade, die jedoch jeglichen negativen Input auf 0 projiziert (Abbildung 11). Der konstante Verlauf der Geraden führt zu schnellerem und konstantem Lernen. Der Umstand, dass die ReLU-Funktion keine negativen Werte liefert, sorgt für eine Stabilisierung der Gewichtungen während der Lernphase.

Möglichkeiten der Regulierung

Zu guter Letzt kann im Interface der Web-App noch die Regulierung (**Regularization**) und deren Regulierungsrate (**Regularization rate**) eingestellt werden. Regulierung kann man als eine Form der Glättung der gefundenen Grenzen verstehen. Sie verhindert das Problem des sogenannten Overfitting. **Overfitting** tritt auf, wenn ein KNN auf Basis einer endlichen Anzahl von Trainingsdaten angelernt wird, die festgelegten Grenzen diese dann exakt beschreiben, jedoch zu eng gezogen werden und sich nicht mehr für eine Generalisierung auf weitere Daten eignen.

Fazit: Besonders gelungen im TensorFlow Playground ist die einfach verständliche Visualisierung. Sowohl die Eingangsfunktionen als auch die entstehenden Kombinationen innerhalb der Neuronen werden grafisch dargestellt. Somit können sich auch mathematisch weniger bewanderte Laien etwas unter den entstehenden Grenzen vorstellen. Außerdem werden die Gewichtungen innerhalb des Netzwerks mit zunehmendem Lernerfolg über die Stärke der Verbindungslinien der Neuronen abgebildet und vermitteln so einen Eindruck über wichtigere und unwichtigere Neuronen.

Leider ist der Übergang von diesem Spielplatz in einen echten Einsatz in der Praxis schwierig. TensorFlow ist nicht so unkompliziert wie PHP, mit dem man einfach mal anfangen kann und direkt sinnvolle Ergebnisse erhält. Insbesondere die richtige Architektur des KNN, also wie viele Eingangsneuronen, versteckte Schichten und Neuronen in jeder Schicht angelegt werden, erfordert ebenso tiefe Kenntnisse der Thematik wie die Analyse der vorhan-

denen Daten und deren Normalisierung für einen Einsatz im KNN.

Wie Googles Greg Corrado in einem Interview bei Bloomberg bereits sagte, ist maschinelles Lernen leider kein magischer Sirup, den man einfach auf ein Problem schüttet, um es zu lösen. Es brauche vielmehr viele gute Gedanken und Sorgfalt, um etwas zu bauen, das sich wirklich auszahlt. Für einen ersten Einstieg in die Thematik eignen sich Googles Spielwiesen sicherlich, wer allerdings ernsthaft etwas mit künstlicher Intelligenz oder maschinellem Lernen entwickeln möchte, kommt um eine solide Ausbildung in diesem Bereich leider nicht herum. ¶

KI UND MACHINE LEARNING FÜR QUEREINSTEIGER

Auf der E-Learning-Plattform Udacity findet sich ein sog. **Nanodegree-Programm** zum „**Machine Learning Engineer**“, also eine Art kostenpflichtiger, zertifizierter Online-Lehrgang. Durch dessen Abschluss soll man die Schlüsselkompetenzen erwerben, die einen Bewerber auf eine Stelle in Unternehmen vorbereiten, die nach Machine-Learning-Experten suchen oder die Techniken des maschinellen Lernens einführen wollen. Der Kurs ist mit durchschnittlich 420 Stunden Aufwand zu absolvieren und setzt Kenntnisse der Mittelstufe in Programmieren und Statistik sowie Rechenmethoden und lineare Algebra auf mittlerem Niveau voraus.

Die Anmeldung ist sofort möglich unter: *http://einfach.st/udac2*.

Seit Kurzem gibt es zusätzlich ein umfangreicheres **Nanodegree-Programm zum „Ingenieur für künstliche Intelligenz“**, welches in Zusammenarbeit mit IBM Watson, amazon alexa und DiDi entwickelt wurde. Kursteilnehmer des AI Nanodegree-Programms lernen die Grundlagen der Spielsuche, Logik und Planung, probabilistischer Expertensysteme, Computer Vision, kognitiver Systeme und Methoden der Spracherkennung. Voraussetzungen sind mindestens fortgeschrittene Kenntnisse der Python-Programmierung (oder eine besonders schnelle Auffassungsgabe) sowie der Mathematik. Der 6-monatige Kurs startet im April 2017. Die Anmeldung ist ab sofort möglich unter: *http://einfach.st/udac3*