Numerikseminar SS2020: Modellierung beliebiger Input-Output Relationen durch Neuronale Netze vom Feed-Forward Typ

Lukas Schulth
lukas.schulth@uni-konstanz.de

2. September 2020

Inhaltsverzeichnis

Zusammenfassung

Diese Ausarbeitung ist die ausführliche Dokumentation des ersten Teils des Doppelvortrags, den ich zusammen mit Tizian Weber am 14.Juli 2020 im Rahmen des Numerikseminars bei Herrn Schropp, Herrn Junk und Herrn Frei im Sommersemester gehalten habe.

In meinem ersten Teil wird eine kurze Einführung über das Perzeptron hin zu typischen Feed-Forward-Netzwerken gegeben. Der Aufbau solcher Netze wird ausführlich betrachtet. Auf die theoretische Qualität solcher Netze wird im letzten Abschnitt von Abschnitt 3 kurz eingegangen. Abschließend werden einige Anwendungsgebiete und ein Beispiel betrachtet.

Im zweiten Teil des Vortrags wurde der Trainingsprozess neuronaler Netzwerke ausführlich von Herrn Weber erklärt und ein praktisches Beispiel vorgestellt.

1 Einführung und historische Entwicklung

Seit über tausenden von Jahren beschäftigt man sich mit der Funktionsweise des menschlichen Gehirns. Mit der Verfügbarkeit moderner Elektronik wurde schon früh versucht, das Gehirn zu modellieren. Die ersten Schritte in Richtung künstlicher neuronaler Netzwerke geht auf den Neurophysiologen Warren McCulloch und den Mathematiker Walter Pitts zurück, die im Jahre 1943 ein Paper verfassten, das beschreibt, wie das Gehirn funktionieren könnte. Sie benutzen elektrische Schaltkreise, um ein einfaches Modell eines Neurons (Nervenzelle) zu beschreiben. Dieses konnte binäre Eingabewerte $x_1, ..., x_m$ in eine Ausgabe y überführen. [1]

Im Jahr 1958 stellte der amerikanische Psychologe und Informatiker Frank Rosenblatt das sogenannte Perzeptron vor, das die Grundlage für heutige Neuronale Netzwerke bildet und die logischen Operation UND, ODER und NICHT modellieren konnte.

2 Perzeptron

Im Jahr 1958 publizierte Frank Rosenblatt (US-amerikanischer Psychologe und Informatiker) das Perzeptron Modell, welches die Grundlage für Neuronale Netze bildet.[2]
Das Perzeptron besteht aus einem einzelnen künstlichen Neuron mit Gewichten und einem Schwellenwert. Ein Perzeptron erhält n binäre Eingaben $x_1, ..., x_n$. Ziel ist es, daraus eine binäre Ausgabe zu erhalten.[3]. Um die Ausgabe zu berechnen, werden die Eingaben mit Gewichten $w_i, i \in \{1, ..., n\}$ multipliziert und aufsummiert. Liegt diese Summe über einem Schwellwert S, ist die Ausgabe 1. Andernfalls ist die Ausgabe 0:

$$\hat{y} = \begin{cases} 0, & \text{falls } \sum w_i x_i \leq S \\ 1, & \text{falls } \sum w_i x_i > S \end{cases} \tag{1}$$

Um anhand der Eingabewerte die richtige Ausgabe zu bekommen, müssen die passenden Gewichte gefunden werden. Dies erfolgt iterativ durch die Berechnung der Ausgabe und dem Update der Gewichte[4]
Die Funktionsweise eines Perzeptrons ist graphisch in Abbildung 1 dargestellt. Die Entscheidung anhand eines Schwellenwertes S erfolgt in der Abbildung im vorletzten Block. Dabei wird auf die gewichtete Summe eine sogenannte Aktivierungsfunktion $g : \mathbb{R} \to \mathbb{R}$ angewendet, um die Ausgabe $\hat{y}$ zu erhalten. Es gibt viele verschiedene Funktionen, die als Aktivierungsfunktionen verwendet werden. Darauf werden wir später genauer eingehen. Im obigen Beispiel ist unsere

[1] `http://www2.psych.utoronto.ca/users/reingold/courses/ai/cache/neural4.html`
[2] https://de.wikipedia.org/wiki/Perzeptron
[3] `http://neuralnetworksanddeeplearning.com/chap1.html`
[4] `https://en.wikipedia.org/wiki/Perceptron`

Aktivierungsfunktion g, die sogenannte Heavyside-Funktion, die Werte oberhalb eines Schwellwertes auf 1 und unterhalb des Schwellwertes auf 0 abbildet.

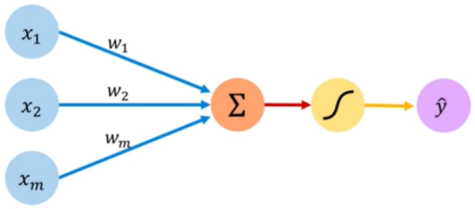


Abbildung 1: Die Gewichte $w_1, ..., w_m$ werden mit den Eingaben $x_1, ..., x_m$ multipliziert. Die anschließende Anwendung einer Aktivierungsfunktion liefert die Ausgabe $\hat{y}$

Wir verschieben den Schwellwert S in Gleichung 1 auf die linke Seite der Gleichung als Bias w_0 und erhalten damit die allgemeine Schreibweise für die Ausgabe eines einzelnen Neurons:

$$\hat{y} = g(w_0 + \sum_{i=1}^{m} w_i x_i)$$

Dies ist wieder graphisch in Abbildung 2 dargestellt.

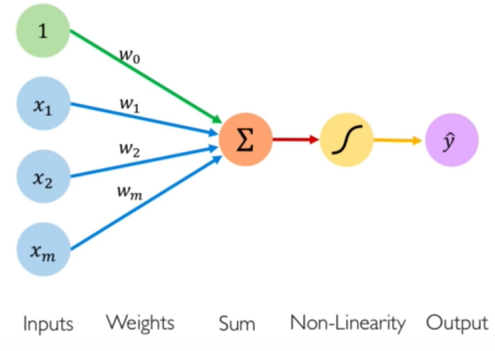


Abbildung 2: Durch Hinzunahme eines Bias w_0 können wir die Entscheidungsgrenze eines einzelnen Neurons linear verschieben

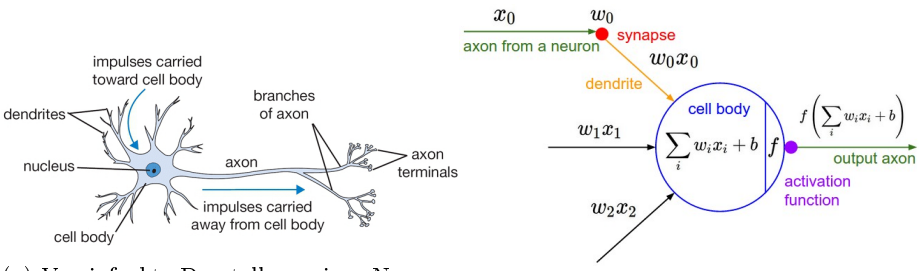
2.1 Biologische Inspiration

Wir beschreiben (sehr vereinfacht) das biologische Vorbild eines künstlichen Neurons und dessen Funktionsweise.

Eine Neuron oder eine Nervenzelle ist eine auf Erregungsleitung und Erregungsübertragung spezialisierte Zelle, die als Zelltyp in Gewebetieren vorkommt. Diese Neuronen bilden zusammen mit den Gliazellen das Nervensystem. Zum zentralen Nervensystem gehören das Gehirn und das Rückenmark.

Im menschlichen Gehirn existieren etwa 86 Millionen Neuronen, die durch $10^{14} - 10^{15}$ Synapsen miteinander verbunden sind[5].

Nervenzellen bestehen aus einem Zellkörper, Dendriten und Neuriten/Axon (Zellfortsätze). Sie unterscheiden sich stark zu anderen Zellen im Körper. Dendriten nehmen Informationen auf und leiten diesen an den Zellkörper weiter. Im Zellkörper einer Nervenzellen befinden sich der Zellkern und der Großteil anderer Zellorganellen. Hier sammeln sich die Informationen aller Dendriten, die am Axonhügel miteinander verrechnet werden. Der Axonhügel ist der Beginn des Axons, dem Zellfortsatz zur Weiterleitung der Informationen. Dieses Axon ist bei Wirbeltieren von einer sogenannten Myelinscheide umhüllt, wodurch die Informationen schneller über das Axon geleitet werden. Das Axon geht in die Präsynaptische Endigung (auch: Synapsenendknöpfchen oder Axonterminale) über, die eine Verbindung zu anderen Nervenzellen oder Muskelfasern über sogenannte Synapsen herstellt.



(a) Vereinfachte Darstellung eines Neurons

(b) Mathematisches Modell eines künstlichen Neurons

Abbildung 3: Vereinfachte Darstellungen von natürlichen und künstlichen Neuronen

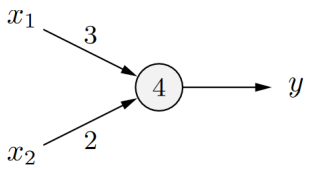
Abbildung 3 zeigt eine Zeichnung eines biologischen Neurons (links) und das mathematischen Modells eines Neurons (rechts).

2.2 Darstellung einfacher logischer Operationen

Im Folgenden zeigen wir, wie Gewichte inklusive Bias gewählt werden müssen, um mit Hilfe eines Perzeptrons die Aussagen $x_1 \wedge x_2$ und $x_1 \rightarrow x_2$ dargestellt werden können.

[5] `https://www.nature.com/scitable/blog/brain-metrics/are_there_really_as_many/`

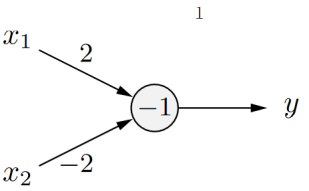
Für $x_1 \wedge x_2$ wählen wir die Gewichte $w_0 = 4, w_1 = 3, w_2 = 3$. Für die Eingabewerte $(x_1, x_2) = (0, 0)$ in der ersten Zeile der Tabelle in Abbildung 4 erhalten wir $3 \cdot 0 + 2 \cdot 0 \leq 0$ und damit folgt $y = 0$.



x_1	x_2	$3x_1 + 2x_2$	y
0	0	0	0
1	0	3	0
0	1	2	0
1	1	5	1

Abbildung 4: Modellierung der UND-Funktion mit Hilfe eines Perzeptrons

Die Aussage $x_1 \rightarrow x_2$ erhalten wir mit den Gewichten $w_0 = -1, w_1 = 2, w_2 = -2$ wie in Abbildung 5 dargestellt.



x_1	x_2	$2x_2 - 2x_1$	y
0	0	0	1
1	0	-2	0
0	1	2	1
1	1	0	1

Abbildung 5: Modellierung einer Implikation

2.3 XOR-Problem: Perzeptron ist ein linearer Klassifikator

Marvin Minsky und Seymour Papert wiesen 1969 nach, dass ein einlagiges Perzeptron die XOR-Funktion (exclusive or: ausschließendes Oder) nicht auflösen kann. Dabei wird ein Perzeptron f_θ gesucht, das die Funktionspaare (x_i, y_i) mit $y_i = f_\theta(x_i)$ korrekt abbildet. Die vorliegenden Paare sind dabei $((1,0),1), ((1,1),0), ((0,1),1)$ und $((0,0),0)$. Stellen wir die beiden Kategorien in einem zweidimensionalen Koordinatensystem dar, so ist es nicht möglich. Beide Kategorien mit nur einer Geraden zu trennen. Wir benötigen eine nicht-lineare Komponente. (Problem der linearen Separierbarkeit). Die Funktionspaare sind in Abbildung 6 links in Tabellenform und rechts als Koordinaten dargestellt

2.4 Lösung: Multi-Layer Perceptrons (MLP)

`http://deeplearning.net/tutorial/mlp.html` Die Modellierung der *xor*-Funktion wird wieder möglich, wenn wir eine weitere Schicht an Neuronen einfügen. Dabei

x_1	x_2	y
0	0	1
1	0	0
0	1	0
1	1	1

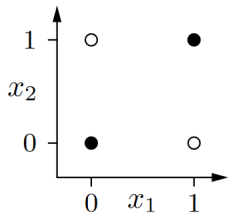


Abbildung 6: Links: Funktionspaare. Rechts: Funktionspaare in einem Koordinatensystem. Die beiden ausgefüllten Punkte besitzen den Wert 1. Die beiden nicht ausgefüllten Punkte besitzen den Wert 0.

werden die Eingabepaare (x_1, x_2) an zwei Neuronen weitergegeben, bevor diese dann eine Ausgabe erzeugen. Dieses mehrschichtige Perzeptron (Multi-Layer Perceptron (MLP)) ist in Abbildung 7 abgebildet.

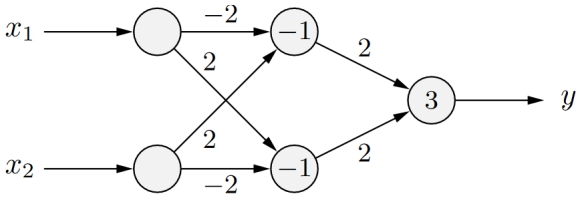


Abbildung 7: Mehrschichtiges Perzeptron löst XOR-Problem

Dabei werden die beiden folgenden Gewichtsmatrizen benutzt:
$W^{(1)} = \begin{pmatrix} -2 & 2 \\ 2 & -2 \end{pmatrix} \in \mathbb{R}^{2x2}$ und
$W^{(2)} = \begin{pmatrix} 2 \\ 2 \end{pmatrix} \in \mathbb{R}^{2x1}$.
Wir schreiben $xor = g(W^{(2)}(g(W^{(1)})))$.
Durch das Einfügen dieser neuen Schicht, erhalten wir eine sogenannte Zwischenschicht.

3 Neuronale Netzwerke vom Typ Feed-Forward

Künstliche Neuronale Netze sind von Neuronalen Netzen (dem Gehirn) inspiriert. Neuronale Netze sind eine Klasse von Funktionen, die aus einfacheren Funktionen zusammengesetzt sind, zusammen mit nichtlinearen Aktivierungsfunktionen, sodass eine nichtlineare Abbildung $f_\theta : \mathbb{R}^n \to \mathbb{R}^m$ mit Parametern θ entsteht.

3.1 Notation

Wir legen einige Notationen fest, um Neuronale Netzwerke im Folgenden beschreiben zu können:

- $x^{(i)}$: i-ter Datenpunkt der Eingabe
- $W^{(l)}$: Gewichte für die Abbildung von Schicht l auf $l+1$
- $w_0^{(l)}$: Vektor an Verschiebungen von Schicht l auf $l+1$
- $g^{(l)} : \mathbb{R} \to \mathbb{R}$: Nicht-lineare Aktivierungsfunktion komponentenweise angewendet.

3.2 Aufbau

Ein Neuronales Netzwerk besteht grundsätzlich aus mehreren Schichten, die miteinander verbunden sind. Die Eingabe läuft durch die verschiedenen Schichten von links nach rechts bis zur letzten Schicht, der sogenannten Ausgabeschicht. Im Folgenden beschreiben wir die einzelnen Bestandteile eines Neuronalen Netzwerkes:

- Eingabeschicht: Hier wird keine Berechnung durchgeführt. Die Eingabe wir an die nächste Schicht weitergegeben.
- Ausgabeschicht: Es wir eine Aktivierungsfunktion benutzt, die auf die gewünschte Ausgabeform des Netzes abbildet.
- Zwischenschicht: Befindet sich zwischen Ein- und Ausgabeschichten. Die Information wird von Schicht l nach $l+1$ abgebildet. Dabei fließt die Information nur in eine Richtung (immer in Richtung der Ausgabeschicht).Es gibt keine Verbindungen zwischen zwei nicht aufeinanderfolgenden Schichten. Außerdem gibt es keine Rekurrenzen, d.h. die Eingabe in eine Schicht kann nicht die Ausgabe derselben Schicht sein (wie beispielsweise im Fall von rekurrenten neuronalen Netzwerken). Es gibt k viele Zwischenschichten mit Breiten $d_1, ..., d_k$
- Verbindungen und Gewichte: Das Netzwerk besteht aus Verbindungen zwischen einzelnen Neuronen aufeinanderfolgender Schichten. Dabei beschreibt $W^{(l)} = W_{ij}^{(l)}$ die Gewichte und damit die Stärke der Verbindung zwischen Neuron i in Schicht l und Neuron j in Schicht l+1.
- Forward-Pass: Unter dem Forward-Pass verstehen wir die Auswertung eines Datenpunktes $x^{(i)}$ mit Hilfe unseres Netzwerks f_θ.
- Die Größe eines Netzwerkes wir charakterisiert durch die Tiefe $d = k + 1$ (Anzahl an Zwischenschichten +1) und die Breite eines Netzwerkes $w = \max\{d_i, i = 1, ..., k\}$ (Anzahl der Neuronen in der Schicht mit den meisten Neuronen).

Für den Übergang von Schicht l auf l+1 ergeben sich die folgenden Schritte:

$$A^{(l)} = W^{(l)}x + w_0^{(l)}$$

.
Dabei beschreibt $A^{(l)}$ die Ausgabe der l-ten Schicht, bevor die Aktivierungsfunktion angewendet wird. Anschließend wenden wir die Aktivierungsfunktion $g^{(l)}$ punktweise an und erhalten die Aktivierungen der l-ten Schicht des Netzwerkes.

Wir erhalten damit die folgende Schreibweise für ein neuronales Netz mir r Schichten:

$$f_\theta = (g^{(r)} \circ A^{(r)}) \circ (g^{(r-1)} \circ A^{(r-1)}) \circ ... \circ (g^{(1)} \circ A^{(1)}).$$

3.3 Beispiel

In diesem Netzwerk wird eine 3-dimensionale Eingabe $x0(x_1, x_2, x_3)$ in eine 2-dimensionale Ausgabe $y = (y_1, y_2)$ überführt. Das Netzwerk besitzt eine Zwischenschicht und damit die Tiefe 2 und die Breite 4. Die Graphische Darstellung ist in Abbildung 8 dargestellt.

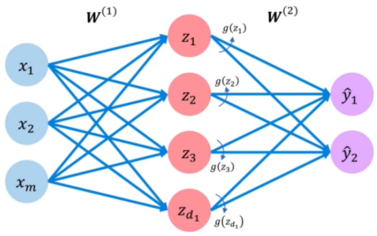


Abbildung 8: Beispiel eines zweischichtigen Neuronalen Netzwerkes

Für die Ausgaben der Zwischen- bzw. Ausgabeschichten erhalten wir:

$$z_i = g\left(w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j w_{j,i}^{(1)}\right)$$

$$\hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j w_{j,i}^{(2)}\right)$$

Für die Dimensionen gilt:
$m = 3, w = d_1 = 4, d = 2, W^{(1)} \in \mathbb{R}^{4x3}, W^{(2)} \in \mathbb{R}^{2x4}$.
$w_0^{(1)} \in \mathbb{R}^4, w_0^{(2)} \in \mathbb{R}^2$.
Insgesamt werden also 26 Parameter benötigt, um das Netzwerk zu betreiben.

3.4 Aktivierungsfunktionen

In unserem ersten Beispiel hatten wir die Heavyside-Funktion als mögliche Aktivierungsfunktion gesehen. Neben dieser gibt es noch viele andere mögliche Aktivierungsfunktionen, die häufiger Verwendung finden. Wir betrachten einige Beispiele der häufigsten Aktivierungsfunktionen in neuronalen Netzwerken.

Sigmoid:

- $\sigma(x) = \frac{1}{1+e^{-x}}$
- Im Allgemeinen ist eine Sigmoidfunktion $\sigma(x)$ eine beschränkte und differenzierbare reelle Funktion, für die gilt: $\sigma'(x) > 0$ oder $\sigma'(x) < 0$ f.a. $x \in \mathbb{R}$ und $\sigma(x)$ besitzt genau einen Wendepunkt.
- Einfache Differenzierbarkeit: $\sigma'(x) = \sigma(x)(1 - \sigma(x))$
- Es gilt: $0 < \sigma(x) < 1$. Damit eignet sich diese Aktivierungsfunktion zur Darstellung einer Wahrscheinlichkeitsverteilung.

tanh:

- $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Tangens hyperbolicus ist eine skalierte Sigmoidfunktion: $tanh(x) = 2\sigma(2x) - 1$
- Einfache Differenzierbarkeit: $tanh'(x) = 1 - tanh(x)^2$
- Es gilt: $-1 < tanh(x) < 1$

ReLU:

- ReLU steht für Rectified Linear Unit
- $relu(x) = max(0, x)$
- $relu'(x) = \begin{cases} 0 & \text{für } x \leq 0 \\ 1 & \text{für } x > 0 \end{cases}$
- Stochastischer Gradientenabstieg konvergiert bei einem Netzwerk zur Bildklassifikation auf ImageNet mit ReLU-Aktivierungen bis zu 6 Mal schneller als ein identisches Netzwerk mit sigmoid- oder tanh-Aktivierungen[6]
- Sehr einfach zu berechnen.

[6] `http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf`

- Laut Fei-Fei Li von der Stanford University deutet aktuelle Forschung darauf hin, dass ReLus zu den Aktivierungsfunktionen gehören, die sich am ähnlichsten zu tatsächlichen Neuronen im menschlichen Gehirn verhalten[7]
- Absterbende ReLUs: Es kann vorkommen, dass beim Training des Netzwerkes einzelne Neuronen mit ReLU-Aktivierungsfunktionen "absterben", d.h. die Gewichte werden so verändert, dass das Neuron bei keiner einzigen Eingabe aus dem Datensatz mehr aktiviert wird. Dies tritt häufig im Zusammenhang mit zu großen Lernrate auf.

Mögliche Aktivierungsfunktionen, um das Absterben von Neuronen zu verhindern:

Leaky ReLU:

- $f(x) = max(\alpha x, x)$
- Eine Idee, um das Absterben von Neuronen zu verhindern.
- Erfolg ist nur in einigen Fällen sichtbar.

Maxout:

- $f(x) = max(w_1^T x + b_1, w_2^T x + b_2)$
- Verallgemeinert ReLU und Leaky ReLU
- Parameteranzahl steigt deutlich: Pro Neuron werden doppelt so viele Parameter verwendet.

Eine der wichtigsten Eigenschaften von Aktivierungsfunktionen ist, dass ein glatter, differenzierbarer Übergang von verschiedenen Eingaben gegeben ist, beispielsweise im Unterschied zur Heavyside-Funktion als Aktivierungsfunktion.

Mit Hilfe von Aktivierungsfunktionen können wir nicht-lineare Zusammenhänge in unseren Daten modellieren. Damit können zum Beispiel die Punkte in Abbildung 9 erfolgreich klassifiziert werden. Benutzen wir jedoch nur lineare Aktivierungsfunktionen, können wir die Daten nicht fehlerfrei klassifizieren. Selbst die Wahl eines größeren Netzwerkes liefert keine Verbesserung.

3.5 Universal Approximation Theorem [Hornik 1991]

Es stellt sich nun die Frage, wie gut die Approximation einer unbekannten Funktion mit Hilfe eines Neuronalen Netzwerkes ist.
Erste Resultate wurden diesbezüglich von Kurt Hornik[8][9] 1991 zunächst für einschichtige, dann für mehrschichtige Netzwerke veröffentlicht.

[7] https://www.youtube.com/watch?v=d14TUNcbn1k&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv&index=4

[8] https://web.njit.edu/~usman/courses/cs675_spring20/hornik-nn-1991.pdf

[9] https://deeplearning.cs.cmu.edu/document/readings/Hornik_Stinchcombe_White.pdf

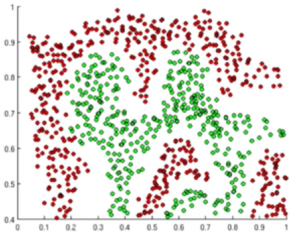
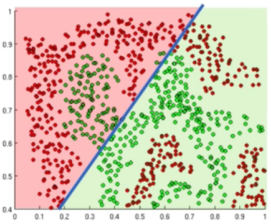
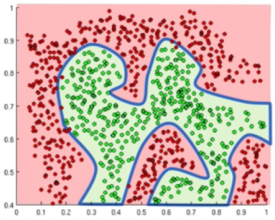


Abbildung 9: Beispielaufgabe: Rote und grüne Punkte sollen mit Hilfe eines Neuronalen Netzes klassifiziert werden



(a) Lineare Aktivierungsfunktionen führen zu linearen Entscheidungen



(b) Mit Hilfe von nicht-linearen Aktivierungsfunktionen können wir beliebig komplexe Funktionen approximieren.

Abbildung 10: Lineare und nicht-lineare Aktivierungsfunktionen

Theoretisch betrachtet unterscheidet sich die Approximationsgüte einschichtiger Netzwerke nicht zu der Qualität mehrschichtiger Netzwerke. In der Praxis zeigt sich jedoch, dass größere Netzwerke meist einfach zu trainieren sind.
Wir zitieren im Folgenden einen Satz, der die Approximationsgüte sogenannter ReLU-Netzwerke beschreibt[10]. Dabei bestehen die Aktivierungen ausschließlich aus ReLU-Funktionen. In der ursprünglichen Arbeit von Kurt Hornik wird eine große Klasse von Aktivierungsfunktionen betrachtet.

Theorem 1 (Approximation mit ReLU-Netzwerken)
Sei $g^{(i)} = ReLU, i = 1, ..., k+1$.
Sei $K \subset \mathbb{R}^m$ *kompakt.*
Sei $f : K \to \mathbb{R}^n$ *stetig.*
Sei $\epsilon > 0$.

Dann existiert ein k-schichtiges neuronales Netzwerk
$f_\theta = (g^{(k+1)} \circ A^{(k+1)}) \circ (g^{(k)} \circ A^{(k)}) \circ ... \circ (g^{(1)} \circ A^{(1)})$ *mit* $A^{(i)} : \mathbb{R}^{d_i} \to \mathbb{R}^{d_{i+1}}, i = 1, ..., k$, *wobei* $m = d_1, d_2, ..., d_k, d_{k+1} = n$ *und* $d_2, ..., d_k$ *den Breiten der Zwi-*

[10] `https://arxiv.org/pdf/1710.11278.pdf`

schenschichten entspricht. mit

$$\sup_{x \in K} |f_\theta(x) - f(x)| < \epsilon$$

4 Andwendungsbeispiele

Einige Anwendungsbeispiele, bei denen Neuronale Netze eingesetzt werden:

- Gesichtserkennung
- Klassifikation von Bildern
- Spracherkennung
- Maschinelle Übersetzung
- Medizinische Diagnosen
- Autonomes Fahren
- Digitale Assistenten
- Werbung, Suchen, Empfehlungen
- General Game Playing

Wir gehen kurz auf ein Beispiel aus der Kategorie der Bildklassifikation ein, um den typischen Workflow zu verdeutlichen.
Wir wollen in diesem Beispiel ein neuronales Netzwerk trainieren, das handgeschriebene Ziffern korrekt den Klassen $0, ..., 9$ zuordnen kann. Dies soll in der späteren Anwendung auch auf solchen Ziffern funktionieren, die das Netzwerk vorher nie verarbeitet hat. Dazu benutzen wir den MNIST-Datensatz, der aus 70.000 Bildern der Pixelgröße $28x28$ mit Pixelwerten zwischen 0.0 (weiß) und 255.0 (schwarz). Für das Training werden 60.000 Bilder benutzt, um die Parameter des Netzwerkes zu optimieren. Dazu werden die Parameter zu Beginn des Trainings zufällig initialisiert. Mit Hilfe einer Verlustfunktion wird die Approximationsgüte des Netzwerkes f_θ bewertet und ein Optimierungsschritt ausgeführt(Näheres dazu im zweiten Teil der Ausarbeitung).
Nach Abschluss des Trainings wird die Qualität des Netzwerkes in der Testphase auf 10.000 Testbildern bewertet.
In Abbildung 11 ist ein Eingabe x dargestellt, die das Label 8 besitzt. Ein erfolgreich trainiertes Netzwerk liefert zu dieser Eingabe die folgende Ausgabe: $f_\theta(x) = \big(0\,0\,0\,0\,0\,0\,0\,0\,1\,0\big)$, d.h. das Bild wird erfolgreich der Klasse 8 zugeordnet.

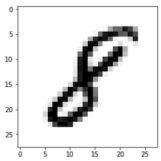


Abbildung 11: Datenpunkt x aus der Klasse mit den Labels '8'.

5 Deep Learning

Unter Deep Learning versteht man die Verwendung neuronaler Netze mit einer deutlich größeren Anzahl an Zwischenschichten (hidden layers). Im Bereich der Bilderkennung werden deutlich größere Netze verwendet. In Abbildung 12 sind einige der bekanntesten Netzwerke zur Klassifikation von Bildern und deren Größe als Anzahl der zu wählenden Parametern dargestellt.

Das Netz Inception-v3 besteht beispielsweise aus 159 Schichten und über 23 Millionen Parametern[11].

VGG16 war ein Durchbruch im Bereich von Convolutional Neural Networks (CNN) nach den Netzwerken LeNet-5 (1998), AlexNet (2012), ZFNet (2013), GoogleNet/Inception (2014). Es besteht aus 16 Schichten mit über 138 Millionen Paramtern.

Aufgrund der sehr hohen Parameteranzahl ist das Training solcher Netzwerke sehr zeitaufwändig. Eine Möglichkeit, die Trainingszeit zu reduzieren ist die Verwendung sogenannter vortrainierter Netzwerke. Soll beispielsweise eine Netzwerk zur Klassifikation von Bilder eingesetzt werden, können Netzwerke verwendet werden, die bereits auf Datensätzen wie beispielsweise Imagenet [12] trainiert wurden. Anschließend wird die letzte Schicht des Netzwerkes auf das vorliegende Problem angepasst. Nach nur wenigen Epochen (Trainingsdurchläufe) werden sehr gute Ergebnisse erzielt trainiert. Dabei wird das sogenannte Transfer-Lernen[13][14] ausgenutzt.

6 Andere Arten von Neuronalen Netzwerken

Als rekurrente bzw. rückgekoppelte neuronale Netze bezeichnet man neuronale Netze, die sich im Gegensatz zu den Feedforward-Netzen durch Verbindungen von Neuronen einer Schicht zu Neuronen derselben oder einer vorangegangenen Schicht auszeichnen. Durch diese rückgerichtete (rekurrente) Kanten (feedback loops) erhalten die neuronalen Netzwerke somit eine Rückkopplung. Im Gehirn

[11] https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d
[12] http://www.image-net.org/
[13] https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html
[14] https://cs231n.github.io/transfer-learning/

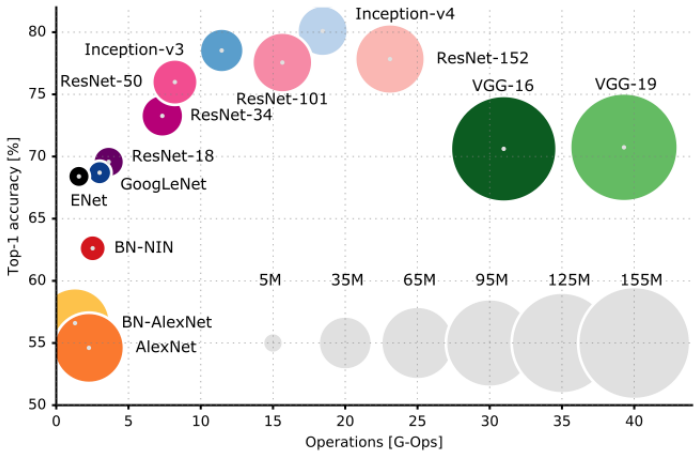


Abbildung 12: Dargestellt sind verschiedene Netze mit Parametern zwischen 5 und 155 Millionen. Auf der x-Achse sind die Operationen notiert, die für einen einzelnen forward-pass benötigt werden. Auf der y-Achse wir die TOP-1-Genauigkeit dargestellt.

ist dies die bevorzugte Verschaltungsweise neuronaler Netze, insbesondere im Neocortex. In künstlichen neuronalen Netzen werden rekurrente Verschaltung von Modellneuronen benutzt, um zeitlich codierte Informationen in den Daten zu entdecken[15].

Bei sogenannten Convolutional Neural Networks(CNN) besteht die Struktur aus einem oder mehreren Convolutional Layer, gefolgt von einem Pooling Layer. Diese Einheit kann sich prinzipiell beliebig oft wiederholen. In der Regel liegt die Eingabe als zwei- oder dreidimensionale Matrix (z. B. die Pixel eines Graustufen- oder Farbbildes) vor. Dementsprechend sind die Neuronen im Convolutional Layer angeordnet. Die Aktivität jedes Neurons wird über eine diskrete Faltung (daher der Zusatz convolutional) berechnet. Dabei wird schrittweise eine vergleichsweise kleine Faltungsmatrix (Filterkernel) über die Eingabe bewegt. Die Eingabe eines Neurons im Convolutional Layer berechnet sich als inneres Produkt des Filterkernels mit dem aktuell unterliegenden Bildausschnitt. Dementsprechend reagieren benachbarte Neuronen im Convolutional Layer auf sich überlappende Bereiche (ähnliche Frequenzen in Audiosignalen oder lokale Umgebungen in Bildern)[16][17].

[15] https://de.wikipedia.org/wiki/Künstliches_neuronales_Netz
[16] http://deeplearning.net/tutorial/lenet.html
[17] https://de.wikipedia.org/wiki/Convolutional_Neural_Network

Abbildungsverzeichnis

Literatur

[1] University of Toronto: Artificial Neural Networks Technology: 3.0 History of Neural Networks
http://www2.psych.utoronto.ca/users/reingold/courses/ai/cache/neural4.html

[2] Michael Nielsen. Neural Networks and Deep Learning. Online Book. December 2019.
http://neuralnetworksanddeeplearning.com/chap1.html

[3] deeplearning.net. Multilayer Perceptron.
http://deeplearning.net/tutorial/mlp.html

[4] Lex Fridman. Massachusetts Institute of Technology. Deep Learning Basics: Introduction and Overview.
https://www.dropbox.com/s/c0g3sc1shi63x3q/deep_learning_basics.pdf?dl=0

[5] Alexander Amini. Introduction to Deep Learning. MIT 6.S191. 27.Januar,2020.
http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

[6] Fei-Fei Li, Justin Johnson, Serena Yeung. Stanford University. Lecture4: Backpropagation and Neural Networks. 13.April,2017.
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture4.pdf

[7] Christian Borgelt. Paris Lodron University of Salzburg. Artificial Neural Networks and Deep Learning.
https://borgelt.net/slides/nn.pdf

[8] Boris Hanin, Mark Sellke. Approximating Continuous Functions by ReLU Nets of minimal width.
https://arxiv.org/pdf/1710.11278.pdf

[9] Hanhe Lin. Universität Konstanz. Lecture 3: Neural Networks, Lecture 4: Deep Neural Networks.
https://www.mmsp.uni-konstanz.de/teaching/courses/summer-2018/deep-learning-programming/

[10] Kurt Hornik, Maxwell Stinchombe, Halber White. Technische Universität Wien, University of California, San Diego. Multilayer Feedforward Networks are Universal Approximators. März 1989.
https://deeplearning.cs.cmu.edu/F20/document/readings/Hornik_Stinchcombe_White.pdf

[11] Alfredo Canziani, Eugenio Culurciello, Adam Paszke. An Analysis of Deep Neural Network Models for Practical Application.
https://arxiv.org/pdf/1605.07678.pdf