Deep Learning

Christoph Wick

Jährlich veröffentlichen Forscher neue Zahlen und Bestwerte ihrer Lernverfahren in verschiedensten Bereichen, mit welchen sie sich den menschlichen Fähigkeiten nähern oder diese sogar bereits übertreffen. Hierbei ist *Deep Learning* als Schlagwort prominent vertreten. Das bekannteste Beispiel darunter ist sicherlich der von Googles DeepMind-Gruppe entwickelte AlphaGo-Computer, der erstmals professionelle menschliche Spieler im Go-Spiel, das weitaus komplexer als Schach ist, bezwang (vgl. [8]). Insbesondere die Mainstreammedien griffen das Thema auf und postulierten eine neue Ära von Künstlicher Intelligenz.

Trotz wachsender Popularität des Deep Learning löst dieser Ansatz nicht pauschal jedes ungelöste oder nur unzufrieden gelöste Problem des maschinellen Lernens. Vielmehr ist es als eines von vielen Werkzeugen zu verstehen, das zum überwachten oder unüberwachten Lernen von insbesondere sehr großen Datensätzen verwendet werden kann. Deep Learning kann hierbei den geschichteten Aufbau hierarchischer Features automatisch sehr gut abbilden, weshalb es vor allem in der Bild- und Sprachverarbeitung eine wichtige Rolle spielt. Weitere Bereiche des maschinellen Lernens wie iML [5] und OCR [9], in denen Deep Learning eingesetzt wird, wurden bereits in Artikeln des aktuellen Schlagworts tituliert.

Struktur eines tiefen Neuronalen Netzwerks

Der Grundbaustein eines jeden Deep-Learning-Ansatzes ist das 1958 von Frank Rosenblatt [7] vorgestellte Perzeptron. Dieses ist eine Konstruktion von mehreren sogenannten künstlichen Neuronen, die mit gewichteten Verknüpfungen und einem Schwellwert miteinander gekoppelt sind. Bei einem einschichtigen Perzeptron sind dabei mehrere Eingabeneuronen mit einem oder mehreren Ausgabeknoten voll verknüpft, d. h. jede Eingabe mit jeder Ausgabe. Die gewichteten Verbindungen sind hierbei die lernbaren Parameter des Verfahrens, welche ursprünglich mit einer sehr einfachen Regel gelernt wurden.

Der Grundbaustein des Perzeptrons ist leicht erweiterbar, indem man mehrere Schichten einzelner Perzeptronen mittels einer nichtlinearen Aktivierungsfunktion koppelt. Eine solche Struktur wird aufgrund ihrer Vielschichtigkeit Multi-Layer-Perzeptron (MLP) genannt.

Darauf aufbauend gibt es eine Vielzahl weiterer spezieller Netzstrukturen, z. B. Convolutional Neuronale Netze, bei denen teilweise ein Neuron einer Schicht mit nur wenigen Neuronen der Vorgängerschicht verbunden ist, um lokale Muster zu erkennen, was u. a. bei der Bildverarbeitung und Mustererkennung sehr erfolgreich eingesetzt wird (s. Abschn. „Deep Learning in der Praxis“).

DOI 10.1007/s00287-016-1013-2

Christoph Wick
Julius-Maximilians-Universität Würzburg,
Lehrstuhl für Künstliche Intelligenz
und angewandte Informatik,
Am Hubland, 97074 Würzburg
E-Mail: christoph.wick@uni-wuerzburg.de

*Vorschläge an Prof. Dr. Frank Puppe
<puppe@informatik.uni-wuerzburg.de>
oder an Dr. Brigitte Bartsch-Spörl
<brigitte@bsr-consulting.de>

Alle „Aktuellen Schlagwörter“ seit 1988 finden Sie unter:
http://www.is.informatik.uni-wuerzburg.de/as

Trainieren eines Neuronalen Netzwerks

Ein Neuronales Netz mit vielen Schichten wird standardmäßig mittels eines Gradientenabstiegs, beispielsweise SGD (Stochastic Gradient Descent) oder dessen Abwandlungen wie AdaGrad oder Nesterov, trainiert. Dazu wird eine Loss-Funktion L definiert, die bei fehlerfreiem Lernen des Trainingsdatensatzes minimal ist. Die Anwendung des Gradientenabstiegs auf L durch Änderung der trainierbaren Gewichte W wird durch die Formel

$$W \leftarrow W - \eta \nabla_W L$$

beschrieben, wobei η der Lernrate und ∇_W der Ableitung nach den Gewichten W entsprechen. Diese sucht und findet ein lokales Minimum von L, welches jedoch nicht global sein muss, da der Trainingsdatensatz nicht zwangsweise auch perfekt gelernt werden kann. Häufig wird als Loss-Funktion die über alle Trainingsbeispiele summierte euklidische Distanz gewählt. Stimmen alle Vorhersagen überein, sind sowohl die Einzelabstände der Beispiele als auch deren Summe gleich Null. Je stärker jedoch die Abweichung eines Beispiels, desto größer (hier quadratisch) geht dieser Fehler in die Loss-Funktion ein.

Als Bild für diese mathematische Formel stellt man sich beispielsweise eine Kugel in einer Gebirgslandschaft L mit allen Gipfeln, Tälern und Gebirgsseen vor. Diese Kugel rollt aufgrund der Schwerkraft in ein Tal als lokales Minimum und zwar dem geringsten Widerstand, d. h. dem steilsten Abstieg, also dem Gradienten ∇_W, folgend. In diesem Bild wird bereits eine modifizierte Version des Gradientenabstiegs verwendet, in welchem der aktuelle Impuls und die Trägheit berücksichtigt werden. Die Kugel kann so aufgrund ihrer Geschwindigkeit aus einem flachen lokalen Minimum entkommen, um ein besseres zu finden.

Die Lernrate η hat in diesem Bild die Bedeutung der Schrittweite in einer physikalischen Simulation. Dieser Parameter hat starke Auswirkungen auf das Ergebnis und ist abhängig von der Landschaftsbeschaffenheit L, denn ein zu kleiner Wert erfordert viele winzige Berechnungsschritte und birgt die Gefahr, in einem sehr flachen lokalen Minimum stecken zu bleiben, wohingegen ein zu großer Wert zu wilden Sprüngen in der Landschaft führen kann und das Verfahren nicht konvergiert. Ein geeigneter Wert sollte deshalb zunächst groß genug sein, um eine grobe Orientierung zu finden und um kleinere Minima zu überspringen, jedoch im Laufe der Simulation gesenkt werden, um zu einem lokalen Minimum zu konvergieren. In der Anwendung wird deshalb die Lernrate η während des Trainings gesenkt.

Zwar ist nun ersichtlich, wie das Lernverfahren funktioniert, jedoch fehlt zur Implementierung des Algorithmus die Berechnung des Gradienten $\nabla_W L(N)$. Dazu soll ein einfaches MLP mit drei lernbaren Schichten und deren Gewichtsmatrizen W_i betrachtet werden. Dies wird in einer vereinfachten mathematischen Schreibweise notiert, da dadurch sowohl das Vanishing-Gradient-Problem, als auch der Backpropagation-Algorithmus leicht erklärt werden können. Als Aktivierungsfunktion wird der Platzhalter $\sigma(x)$ verwendet, beispielsweise $\tanh(x)$.

$$N(x) = W_1 \cdot \sigma(\overbrace{W_2 \cdot \sigma(\underbrace{W_3 \cdot x}_{N_3})}^{N_2})$$

Gut erkennbar ist die Verkettung der einzelnen Schichten, indem die Gewichtsmatrix der i-ten Ebene auf die Ausgabe der Aktivierungsfunktion der darüber liegenden angewandt wird, hier notiert als N_2 bzw. N_3. $N(x)$ bezeichnet hier nur die Ausgabe des Netzwerks ohne eine Loss-Funktion, welche $N(x)$ als Argument erhält $L(N(x))$.

Da L deshalb eine Verkettung von mehreren Funktionen ist, findet die Ableitungsregel $\frac{\mathrm{d}f(g(x))}{\mathrm{d}x} = \frac{\mathrm{d}g(x)}{\mathrm{d}x}\frac{\mathrm{d}f(g)}{\mathrm{d}g}$ mehrmals Anwendung:

$$\frac{\mathrm{d}L}{\mathrm{d}W_1} = \sigma(N_2) \cdot \frac{\mathrm{d}L}{\mathrm{d}N}$$
$$\frac{\mathrm{d}L}{\mathrm{d}W_2} = \sigma(N_3)\frac{\mathrm{d}\sigma(N_2)}{\mathrm{d}N_2} \cdot W_1 \cdot \frac{\mathrm{d}L}{\mathrm{d}N}$$
$$\frac{\mathrm{d}L}{\mathrm{d}W_3} = x \cdot \frac{\mathrm{d}\sigma(N_3)}{\mathrm{d}N_3} \cdot W_2 \cdot \frac{\mathrm{d}\sigma(N_2)}{\mathrm{d}N_2} \cdot W_1 \cdot \frac{\mathrm{d}L}{\mathrm{d}N}$$

Man erkennt, dass viele Terme wiederkehren, je tiefer man in das Netz vordringt. Hier beispielsweise die Ableitung der Loss-Funktion $\frac{\mathrm{d}L}{\mathrm{d}N}$ und die der ersten Schicht $\frac{\mathrm{d}\sigma(N_2)}{\mathrm{d}N_2}$. Der sogenannte Backpropagation-Algorithmus speichert nun die wiederkehrenden Faktoren zur Berechnung der Ableitungen in tieferen Ebenen und ist somit eine effiziente Implementierung der Kettenregel nach

den einzelnen Lernparametern. So wird beispielsweise ab der zweiten Schicht jedes Mal der Faktor $\frac{d\sigma(N_2)}{dN_2} \cdot W_1 \cdot \frac{dL}{dN}$ benötigt, sodass dieser nach unten weitergereicht werden kann. Lehrbücher schreiben vereinfacht, dass der Fehler gemäß den Gewichten aufgeteilt und zurück propagiert wird, woher der Algorithmus seinen Namen trägt.

Zwar können mit dem SGD, welcher in dieser Form noch heute verwendet wird, effizient Neuronale Netze trainiert werden, jedoch zeigen sich Probleme, die das Trainieren von tiefen Strukturen aufgrund eines verschwindenden Gradienten erschwert.

Vanishing-Gradient-Problem

Das von Sepp Hochreiter im Jahre 1991 [4] in seiner Diplomarbeit erstmals beschriebene Problem des verschwindenden Gradienten ist eine der Hauptursachen, warum es anfänglich unmöglich war, tiefe Netze zu trainieren. Dies lässt sich leicht bei näherer Betrachtung der oben berechneten verketteten Ableitungen verstehen. Je tiefer das Netz wird, desto mehr partielle Ableitungen der Aktivierungsfunktion, d. h. Faktoren, müssen miteinander multipliziert werden (z. B. $\frac{d\sigma(N_2)}{dN_2}$). Sind diese während des Trainings < 1, dann multiplizieren sich mehrere Faktoren zu einer Zahl, die schnell gegen Null strebt, weswegen die Gewichtsänderungen in tiefen Schichten deutlich langsamer sind als die in höheren. Da ursprünglich zumeist der $\tanh(x)$ als Aktivierungsfunktion verwendet wurde, führte dies unausweichlich zu diesem Problem, denn alle Funktionswerte und auch die Ableitungen an allen Stellen sind stets betragsmäßig < 1. Besonders relevant ist das Problem auch bei rekurrenten Netzen, die als zweite Dimension die Zeit besitzen, weshalb der Gradient ebenso in Zeitrichtung verschwindet.

Bei Verwendung von Aktivierungsfunktionen mit Gradienten > 1 kann stattdessen das Exploding-Gradient-Problem auftreten. Dort werden Zahlen > 1 multipliziert, wodurch der Wert des Gradienten in tiefen Schichten explodiert und das Netz nicht konvergiert.

Heutzutage gibt es mehrere Ansätze, die Probleme des Vanishing Gradient zu bekämpfen. Der einfachste ist die Verwendung von purer Rechenkapazität, insbesondere von Grafikkarten, die die Anzahl an Flops seit dem Jahr 1991 bis heute (2016) vermillionenfacht haben. So ist es möglich, durch enormen Rechenaufwand selbst kleine Gradienten zur Optimierung eines Netzes zu verwenden. Weiterhin wird in modernen Netzarchitekturen meist zusätzlich eine ReLU-Aktivierungsfunktion verwendet, die als $\sigma(x) = \max(0, x)$ definiert ist. Da die Ableitung dieser nichtlinearen Funktion für $x > 0$ exakt 1 ist, wird sowohl ein Explodieren als auch ein Verschwinden des Gradienten verhindert. So erkennt man im obigen Formalismus, dass die Ableitungen der Aktivierungsfunktionen, falls deren Argument > 0 ist, auf 1 gesetzt wird und nur noch die Gewichtsmatrizen Einfluss auf die Gewichtsänderungen haben. Nicht zu vergessen ist hierbei, dass die Gewichtsänderung 0 ist, sobald ein Argument der Aktivierungsfunktion < 0 ist, jedoch trifft dies nur auf einen Teil der lernbaren Gewichte in den Gewichtsmatrizen zu.

Die übrigen Parameter erhalten nun aber eine Gewichtsänderung, die nicht dem Vanishing-Gradient-Problem unterliegen, sondern ausschließlich von den Gewichtsmatrizen und Netzausgaben der höheren und tieferen Schichten W_i, bzw. $\sigma(N_i)$ und der Ableitung der Loss-Funktion $\frac{dL}{dN}$ abhängen. Die Größenordnung eines Gradienten liegt deshalb stets in der des zugehörigen Gewichts, multipliziert mit der Ableitung der Loss-Funktion, d. h. die Gewichtsänderung ist proportional zum Fehler und aktuellen Gewicht, weshalb ein Nichtverschwinden garantiert ist, solange ein Fehler existiert.

Andere Ansätze, wie der von Geoffrey Hinton im Jahr 2006 [2] vorgestellte, trainieren das Netzwerk zunächst schichtenweise unüberwacht, woraufhin diese in einem Feintuning auf den eigentlichen Datensatz trainiert werden. Tiefe rekurrente Netze können beispielsweise durch Verwendung sogenannter LSTM-Zellen, die auf Hochreiter und Schmidhuber [3] zurückgehen, trainiert werden. Diese fügen dem Netzwerk ein lernbares Gedächtnis hinzu, weshalb Gradienten so über mehrere Zeitschritte gespeichert werden können, ohne dass diese verschwinden.

Durch diese und weitere moderne Techniken ist es seit einigen Jahren nun möglich, effizient sehr tiefe Neuronale Netze mit mehreren Millionen Parametern zu trainieren. Eine weitere wichtige Rolle spielen hierbei die riesigen Datensätze, die aufgrund von BigData nun verfügbar sind und einer Überanpassung der Gewichte entgegenwirken. Auf das Problem des Overfittings wird in diesem Artikel jedoch nicht weiter eingegangen.

Für eine vertiefte Auseinandersetzung mit Deep Learning sei hier auf einen Artikel über Deep Learning von LeCun et al. [6] und auf ein neu erschienenes Buch von Goodfellow et al. [1] verwiesen.

Deep Learning in der Praxis

Aufgrund der Popularität des Deep Learning existieren bereits mehrere mächtige freie Open-Source-Frameworks zum Trainieren tiefer Netzstrukturen. Zu nennen sind hier Caffe[1], CNTK[2], Tensorflow[3] und Torch[4] (in alphabetischer Reihenfolge), wobei bemerkenswert ist, dass die Softwaregiganten Google und Microsoft ihre intern entwickelten Frameworks Tensorflow und CNTK frei zur Verfügung stellen. Alle Frameworks bieten eine Schnittstelle zu NVIDIAs Cuda und CUDNN zum effizienten Training auf Grafikkarten. Die von AMD entwickelte Alternative OpenCL zu NVIDIAs Cuda wird beispielsweise von Caffe experimentell unterstützt. Die Einrichtung der Frameworks setzt standardmäßig einen Linux-basierten Computer oder Grafikkartenserver voraus, wobei verschiedene inoffizielle Portierungen auf Windows existieren.

Die einzelnen Frameworks haben neben der nativen C/C++-Schnittstelle entweder eine Python- oder Lua-Anbindung, die es erlaubt, mit wenigen Zeilen Code Netzwerkstrukturen zu erstellen und zu trainieren. Das Training ist aufgrund der verschiedenen Hyperparameter, darunter die Lernrate η oder die Netzstruktur $N(x)$, nicht trivial und erfordert einige Erfahrung, die man jedoch beispielsweise durch Ausprobieren der Beispiele der einzelnen Frameworks erlangen kann.

Anwendung finden die genannten Frameworks in den verschiedensten Bereichen des Deep Learnings. Ein Hauptgebiet ist dabei die Bildverarbeitung zur Klassifikation (z. B. Schildererkennung oder Gesichtserkennung) oder Segmentierung (z. B. Lokalisieren von Objekten). Insbesondere die speziellen Convolutional Neuronalen Netze (CNN), siehe dazu z. B. [6], spielen hier eine wichtige Rolle, da diese aufgrund ihrer speziellen Struktur spezifische lokale Eigenschaften hierarchisch finden können. So können in der ersten Ebene des Netzes Konturen, Linien oder Bögen gefunden werden, die in tieferen Ebenen komplexere Strukturen bilden, um schließlich das gewünschte Zielobjekt zu erkennen. Moderne Netzstrukturen, wie das bekannte GoogleNet, das zur Klassifikation von Bildern eingesetzt werden kann, besitzen dabei bereits mehr als 20 Schichten.

Eine Benchmark auf diesem Bereich ist der MNIST-Datensatz[5] von handgeschriebenen Ziffern, auf welchem CNNs Genauigkeiten von über 99,7 % erzielen. Die falsch klassifizierten Ziffern sind hierbei auch für den Menschen nicht eindeutig zuordenbar, da diese sehr unsauber und dadurch mehrdeutig geschrieben sind. Faszinierend ist dennoch, dass Deep Learning, das keinerlei Vorwissen über die Aufgabe besitzt, selbstständig durch das Lernverfahren die Gewichte so optimiert und dadurch eigene Features ausbildet, die klassische Ansätze des maschinellen Lernens übertreffen.

Ein weiteres wichtiges Gebiet ist die Verarbeitung von zeitlichen Sequenzen wie z. B. Spracherkennung oder OCR von geschriebenem Text. Letzteres wurde bereits im aktuellen Schlagwort von Uwe Springmann [9] vorgestellt. Dazu werden tiefe rekurrente Netze (RNN), beispielsweise LSTM-Modelle, genutzt, um eine Eingabe- auf eine Ausgabesequenz abzubilden, mithilfe derer beispielsweise gesprochener Text transkribiert oder geschriebener Text erkannt werden kann. Bei einer Überlappung von Bild- und Sequenzverarbeitung, z. B. um in Videos Bewegungen oder Abläufe zu erkennen, können bereits RNNs und CNNs kombiniert und gemeinsam trainiert werden.

Da riesige Netze aus vielen frei lernbaren Parametern im Millionenbereich bestehen, wächst die Gefahr einer Überanpassung auf den Trainingsdatensatz. Um dem entgegenzuwirken, werden teils riesige Datensätze verwendet, die zusätzlich durch Generierung neuer Beispiele aus den existierenden augmentiert werden. Dies ist beispielsweise durch Addieren von Rauschen oder Anwendung von Transformationen zu erreichen. Das Training eines tiefen Netzwerkes dauert trotz Grafikkartenbeschleunigung und effizienten Implementierungen teils mehrere Tage, weshalb verständlich ist, dass vor allem der technische Fortschritt den Boom des Deep Learning in den vergangenen Jahren ermöglicht hat.

[1] http://caffe.berkeleyvision.org/
[2] https://www.microsoft.com/en-us/research/product/cognitive-toolkit/
[3] https://www.tensorflow.org/
[4] http://torch.ch/
[5] http://yann.lecun.com/exdb/mnist/

Spannend hierbei bleibt, wann die nächsten größeren Durchbrüche in der Künstlichen Intelligenz durch den Einsatz von Deep Learning eintreten.

Literatur

1. Goodfellow I, Bengio Y, Courville A (2016) Deep Learning. The MIT Press
2. Hinton GE, Osindero S, Teh Y-W (2006) A fast learning algorithm for deep belief nets. Neural Comput 18:1527–1554
3. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
4. Hochreiter S (1991) Untersuchungen zu dynamischen neuronalen netzen. Diplomarbeit, TU München
5. Holzinger A (2016) Interactive machine learning (iml). Informatik-Spektrum 39(1):164–168
6. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521:436–444
7. Rosenblatt F (1957) The Perceptron – A Perceiving and Recognizing Automaton. Cornell Aeronautical Laboratory, 85-460-1
8. Silver D, Huang A, Maddison CJ et al. (2016) Mastering the game of go with deep neural networks and tree search. Nature 529:484–503
9. Springmann U (2016) OCR für alte drucke. Informatik-Spektrum 39(6):459–462