Einführung in Deep Learning – LSTM & CNN

Proseminar Data Mining

Jonas Knupp
Fakultät für Informatik
Technische Universität München
Email: knupp@in.tum.de

***Kurzfassung*— Aufbauend auf das Grundverständnis zu neuronalen Netzen (NNs) gibt diese Ausarbeitung einen kurzen Einblick in das Themengebiet Deep Learning (DL) und stellt als zwei verbreitete Architekturen Long Short-Term Memories (LSTMs) und Convolutional Neural Networks (CNNs) vor, welche gegenwärtig in der Praxis effektiv eingesetzt werden.**

***Schlüsselworte*— Deep Learning, Vanishing Gradient Problem, Recurrent Neural Network, Backpropagation Through Time, Long Short-Term Memory, Convolutional Neural Network**

I. Einleitung

Der Begriff Deep Learning (DL) hat sich mittlerweile zum Synonym der aktuellen Erfolgsgeschichte neuronaler Netze (NNs) entwickelt. Er beschreibt den Trend zu vielschichtigen Architekturen, Deep Neural Networks (DNNs), welche auf diverse Art der komplexen Entscheidungsfindung durch maschinelles Lernen (ML) in realen Anwendungsgebieten gerecht werden sollen.

Vergangene Forschungsarbeiten haben das Potential von DNNs wiederholt belegt und insbesondere in praxisnahen Bezug gestellt (Kapitel III-C und IV-B). Somit ist maschinelles Lernen marktreif geworden und wird längst im Alltag eingesetzt, nicht zuletzt aufgrund steigender Rechenleistung sowie effizienterer Modelle und Algorithmen.

Die Netztiefe erhält erst durch die problemspezifische Anpassung der Modellarchitektur ihr volles Potential. Zwei gegenwärtig prominente Beispiele sind Long Short-Term Memory (LSTM) und Convolutional Neural Network (CNN). Während LSTMs zur Kategorie der Recurrent Neural Networks (RNNs) gehören und prädestiniert sind für sequentielle Prozesse, etwa Spracherkennung oder Textübersetzung, werden CNNs vorwiegend bei visueller Mustererkennung jeglicher Art eingesetzt.

Diese Ausarbeitung soll eben diese Modellbausteine erläutern. Zunächst wird in Kapitel II die Idee von Deep Learning begründet und damit einhergehende Herausforderungen aufgezeigt, insbesondere das Vanishing Gradient Problem. Kapitel III stellt das grundlegende Konzept von Recurrent Neural Networks vor, den in diesem Kontext oft verwendeten Trainingsalgorithmus Backpropagation Through Time (BPTT), sowie Aufbau, einige Eigenschaften und Anwendungsgebiete von Long Short-Term Memories. Kapitel IV geht auf Aufbau, Eigenschaften, Training und Anwendungsgebiete von Convolutional Neural Networks ein.

II. Deep Learning (DL)

Das Universal Approximation Theorem beweist [1], dass bereits ein neuronales Netz mit nur einer einzigen endlich großen Zwischenschicht ausreicht, um eine arbiträre stetige Funktion (unter milden Voraussetzungen) beliebig genau approximieren zu können. Insofern sind tiefe neuronale Netze mit hunderten Schichten, wie sie heutzutage in der Praxis vorkommen, keine theoretisch mathematische Notwendigkeit.

Jedoch kann so das Entscheidungsproblems nur als Art "Normalform" darstellen, was zu exponentiellem Wachstum der notwendigen Knotenanzahl hinsichtlich der zu erkennenden Musteranzahl führt. Zudem ist die Rauschempfindlichkeit aufgrund der Ad-hoc-Mustererkennung recht hoch. Somit sind flache Netze meist nicht praxistauglich.

Durch mehrere Schichten erlaubt man DNNs hingegen Funktions- bzw. Feature-Kompositionen, welche eine schrittweise Problemlösung ermöglichen. Mit den Schichten wächst der Raum zur Umstrukturierung und Faltung der Daten. Die Eigenheiten natürlicher Entscheidungsfindung können so naturgetreuer und stabiler modelliert werden. Oft ist damit ein signifikant kompakteres, bestenfalls linear statt exponentiell wachsendes, Netz erreichbar.

A. Feature-Komposition

Letztlich wird die potentiell unstrukturierte, vieldimensionale Eingabe durch das trainierte Netz in einen sogenannten Feature-Vektor transformiert. Ziel ist eine zur weiteren Verarbeitung optimierte Darstellung, also eine Repräsentation der Eingabe (Representation Learning), welche die darin versteckten semantischen Aspekte möglichst direkt zugänglich macht, etwa zur Klassifikation des Eingabemusters auf Basis grober, relevanter Eigenschaften/Features (Abb. 1).

Ähnlich zu PCA (Principal Component Analysis) ist es oft sinnvoll, Dimensionen mit hoher Varianz oder statistischer Unabhängigkeit zu formen. Zur semantischen Aufbereitung muss das Netz die versteckten Zusammenhänge verstehen und dekorrelieren bzw. isolieren können. Fasst man die Eingabe als Komposition und Überlappung solcher Muster auf, so bietet sich deren schrittweise Erkennung und Separierung an. Eben dies wird durch die Mehrschichtigkeit realisiert.

Dabei generiert jede Zwischenschicht eine semantisch umstrukturierte Repräsentation der ursprünglichen Eingabe. Sie dient der jeweils nachfolgenden Schicht, welche ihre Features wiederum aus vorherigen zusammensetzt (Feature-Komposition). So entsteht eine Feature-Hierarchie (Abb. 1).

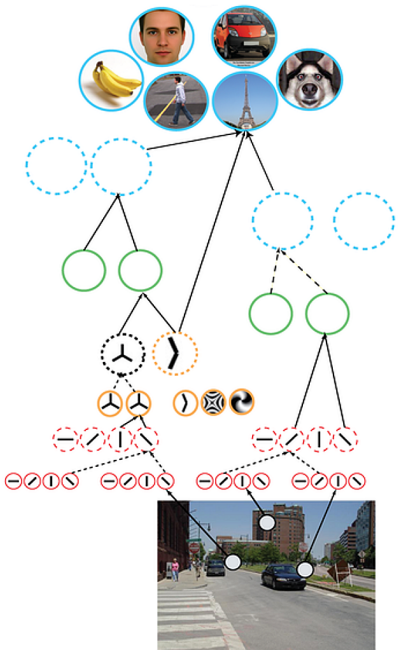


Abbildung 1: Feature-Komposition bei CNNs. Komplexe Muster werden aus einfacheren zusammengesetzt. Grafik aus [2].

Oft weisen allerdings besonders die tiefen Schichten keine direkt verständlichen Features auf und sind meist dense/distributed statt sparse/class-based. Sie können als Embedded-Vektor zur semantischen Repräsentation der Eingabe genutzt werden (z.B. bei Word2Vec), aber erst nach weiterer Abstraktion und Konkretisierung, bis hin zur Klassifizierung, ist eine exakte Interpretation der einzelnen Dimensionen durch den Menschen möglich. Das macht es bislang schwierig, Gelerntes in DNNs vollständig nachzuvollziehen, geschweige denn ihre Korrektheit und Zuverlässigkeit zu verifizieren.

B. Herausforderungen

Neben ihrem enormen Potential und ihrer teils schwierigen Interpretierbarkeit bergen DNNs auch einige neue Herausforderungen bezüglich des Trainings. Ihre Realisierung ist unter anderem dank zunehmend leistungsstarker Hardware möglich geworden. Dennoch bleibt als wohl entscheidendster Faktor für erfolgreich lernende Modelle die mathematische Architektur und Algorithmik.

- $\gg Mio.$ Parameter, GB − TB Datenmenge
  $\rightarrow$ Parameter Server, Distributed Computing, ...
- Konvergenz (komplexer Suchraum)
  $\rightarrow$ optimierte Initialisierung, langes Training, ...
- Performance (großer Rechenaufwand)
  $\rightarrow$ GPU, Parallelisierung, ...
- Overfitting (Detaillierung statt Generalisierung)
  $\rightarrow$ viele Daten, Regularisierung, Early Stop, ...
- Vanishing Gradient (exponentieller Fehlerschwund)
  $\rightarrow$ Modellanpassung, ...

$\Longrightarrow$ **effiziente Modelle und Algorithmen**

C. Vanishing Gradient

Wie bereits 1991 von Hochreiter anhand RNNs erläutert [3], haben DNNs ein zentrales Problem beim Backpropagation, bekannt als “vanishing (/exploding) gradient problem”.

Zur Berechnung der Ableitungen, welche als Grundlage des Gradient-Descent-Algorithmus dienen, fällt durch wiederholt rekursives Anwenden der Kettenregel die Ableitung der Aktivierungsfunktion an jedem ihrer Anwendungsstellen multiplikativ ins Gewicht (Gleichung 1). Da sich diese bei Aktivierungsfunktionen wie $sigmoid$ oder $tanh$ im Bereich $sigmoid' \in (0; 0.25]$ oder $tanh' \in (0; 1]$ bewegt (Abb. 2), wird δ mit steigender Backpropagation-Tiefe betragsmäßig immer kleiner, sofern die Gewichte nicht entsprechend ausgleichend entgegenwirken, also $W \cdot act' \approx 1$. Letzteres wäre jedoch im Hinblick auf die Aktivierungsfunktionen, welche durch zu hohe Gewichte in sehr flachem (satured) Terrain landen (Abb. 2), nicht hilfreich. Im Endeffekt lernen also tiefere Schichten aufgrund des tendenziell exponentiellen Fehlerschwunds signifikant langsamer.

$$
\begin{aligned}
\delta_l &:= \frac{\partial E}{\partial net_l} \\
&= \frac{\partial E}{\partial act_L} \frac{\partial act_L}{\partial net_L} \frac{\partial net_L}{\partial act_{L-1}} \cdots \frac{\partial net_{l+1}}{\partial act_l} \frac{\partial act_l}{\partial net_l} \\
&= diag(act_l'(net_l)) \\
&\quad \cdot \left(\prod_{k=l+1}^{L} W_k^T \cdot diag(act_k'(net_k)) \right) \cdot \frac{\partial E}{\partial act_L}
\end{aligned} \tag{1}
$$

Eine naheliegende und momentan populäre Lösung ist die Wahl einer weniger problematischen Aktivierungsfunktion, beispielsweise Rectified Linear Unit ($ReLU$, Gleichung 2). Deren Ableitungen ($ReLU' \in \{0; 1\}$) führt zur effektiven Vermeidung des exponentiellen Fehlerschwunds. Nachteilig ist ihr konstanter 0-Wert im negativen Definitionsbereich. Dem entgegenwirken können weiche Varianten wie $softplus$, $leakyReLU$ oder ELU.

$$
ReLU(x) = max(0, x), \quad ReLU'(x) = \begin{cases} 0 & x < 0 \\ 1 & x > 1 \end{cases} \tag{2}
$$

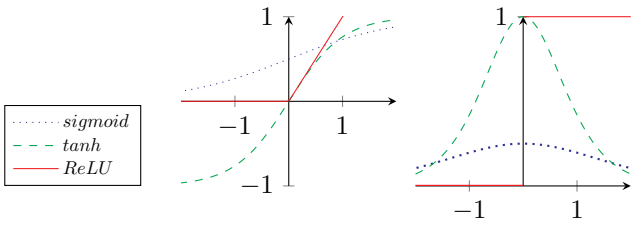
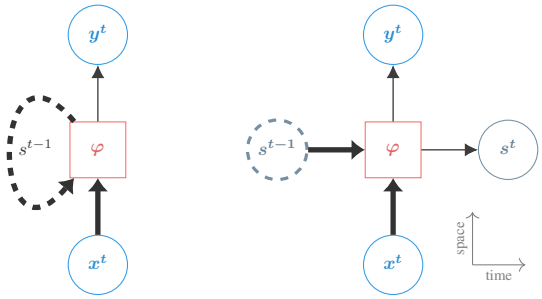


Abbildung 2: Aktivierungsfunktionen (links) und deren Ableitungen (rechts). Man beachte die Erwartungswerte Letzterer.

III. Recurrent Neural Network (RNN)

Die bisher angesprochene Erweiterung zu Deep (Feed-Forward) Neural Networks (DFNNs) gibt einem Modell zur Entscheidungsfindung mehr Platz im Informationsraum (vertikal sequentielle Verarbeitung). Allerdings fehlt ihm nach wie vor die Fähigkeit zur zeitlich sequentiellen Herangehensweise (horizontal), etwa für Textverarbeitung oder Echtzeitanwendungen. Diese kann selbst durch beliebig große DFNNs nicht erreicht werden. Stattdessen ist eine Ausweitung des Modells auf eine zeitliche Dimension erforderlich.

Eine Möglichkeit dafür sind Recurrent Neural Networks (RNNs). Sie stellen genau wie FNNs Funktionsapproximierer dar, jedoch verwalten sie zudem einen inneren Status s (Hidden-State). Rekurrente Netzschichten können zusätzlich zur üblichen Eingabe bzw. Ausgabe eben jenen aktuellen Status lesen bzw. für den nächsten Zeitschritt neu schreiben (Abb. 3, Gleichung 3). Durch gezielte Nutzung dieses Zwischenspeichers, der als akutes "Gedächtnis" oder "Arbeitsspeicher" verstanden werden kann, erhalten RNNs bewiesenermaßen [4] das Potential zur Turing-Vollständigkeit und somit die theoretische Befähigung zur zeitlich gegliederten Lösung von Problemen verschiedenster Art.



(a) RNN-Schicht (b) Entfaltete RNN-Schicht

Abbildung 3: Allgemeine äußere Struktur einer RNN-Schicht

$$
\begin{aligned}
&x \in \mathbb{R}^n, \quad y \in \mathbb{R}^m, \quad s \in \mathbb{R}^{m \times k} \\
&(y^t, s^t) = \varphi(x^t, s^{t-1}) \\
&\text{Bsp.: } s^t = y^t = \varphi(x^t, y^{t-1}) = act_l(net^t) \\
&\qquad net^t = Wx^t + Ry^{t-1} + b
\end{aligned}
\tag{3}
$$

A. Backpropagation Through Time (BPTT)

Um RNNs zu trainieren, können sie über die zurückliegenden Zeitschritte entfaltet werden (Unfolded RNN), sodass der daraus resultierende gerichtete azyklische Graph (DAG) wie üblich mittels Backpropagation traversiert werden kann (Backpropagation through time, BPTT). Da eine Variable über mehrere Zeitschritte hinweg wirkt (shared parameters), ergibt sich eine Addition der partiellen Ableitungen über alle zurückliegenden Zeitschritte (Gleichung 4, Abb. 4).

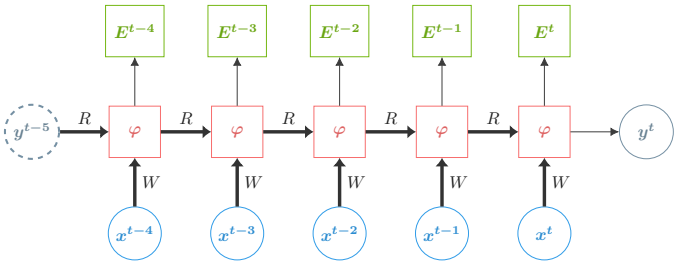


Abbildung 4: Backpropagation Through Time (BPTT). Je Fehler E^t wird die partielle Ableitung nach W, R und b über zurückliegende Zeitschritte $[0;t]$ akkumuliert (Gleichung 4).

$$
\begin{aligned}
\delta W &:= \frac{\partial E}{\partial W} = \sum_{t=0}^{T} \sum_{k=0}^{t} \frac{\partial E^t}{\partial net^k} (x^k)^T = \sum_{t=0}^{T} \delta net^t (x^t)^T \\
\delta R_l &:= \frac{\partial E}{\partial R} = \sum_{t=1}^{T} \delta net^t (y^{t-1})^T, \quad \delta b := \frac{\partial E}{\partial b} = \sum_{t=0}^{T} \delta net^t
\end{aligned}
\tag{4}
$$

Hinsichtlich der Tiefe des aufgefalteten Netzes droht bei RNNs im besonderen Maße Vanishing Gradient. Zudem fällt es einfachen RNNs schwer, über längere Zeit eine bestimmte Information konstant zu speichern und zum richtigen Zeitpunkt gezielt zu nutzen oder zu verwerfen. Darum sind in der Praxis erweiterte RNNs notwendig, etwa LSTMs oder GRUs.

B. Long Short-Term Memory (LSTM)

Mit der Einführung von Long Short-Term Memories (LSTMs) haben Hochreiter und Schmidhuber 1997 [5] maßgeblich zum heutigen Erfolg von RNNs beigetragen. Die Architektur adressiert zuvor genannte Probleme einfacher RNNs und ermöglicht so den effektiven Umgang mit Langzeitabhängigkeiten. Spätere Anpassungen wie Forget-Gate [6] und Peepholes [7] gehören zur aktuellen Standard-Variante (Abb. 5, Gleichung 5) unter diversen Variationen.

Äußerlich entspricht eine LSTM- einer RNN-Schicht mit Hidden-State s bestehend aus der vorherigen Ausgabe y^{t-1} sowie separatem Speicher (Cell) c^{t-1} (Gleichung 5a). Um Letzteren kontrolliert beschreiben, löschen und lesen zu können, werden entsprechend Input-, Forget- und Output-Gate eingeführt, welche als Prozentwerte $g_{\{i,f,o\}} \in [0;1]$ (meist durch $sigmoid$) verstanden werden können und multiplikativ die Menge des Informationsdurchflusses an ihrer jeweiligen Position steuern (Abb. 5, Gleichung 5c). Die Gates werden durch einfache Perceptrons berechnet, aus aktueller Eingabe x^t und vorherigem Status $s^{t-1} = (y^{t-1}, c^{t-1})$ (wobei die darin enthaltenen Verbindungen von c zu den Gates als Peepholes bezeichnet werden), bzw. für das Output-Gate g_o^t bereits mit c^t (Gleichung 5b). Darüber hinaus wird durch gewöhnliche nichtlineare Aktivierungsfunktionen $g_{\{z,y\}}$ (z.B. $tanh$) vor dem Schreiben in und Lesen aus c die bereits von FNNs bekannte Fähigkeit zur universellen Funktionsapproximation gewährleistet (Abb. 5, Gleichung 5b,c).

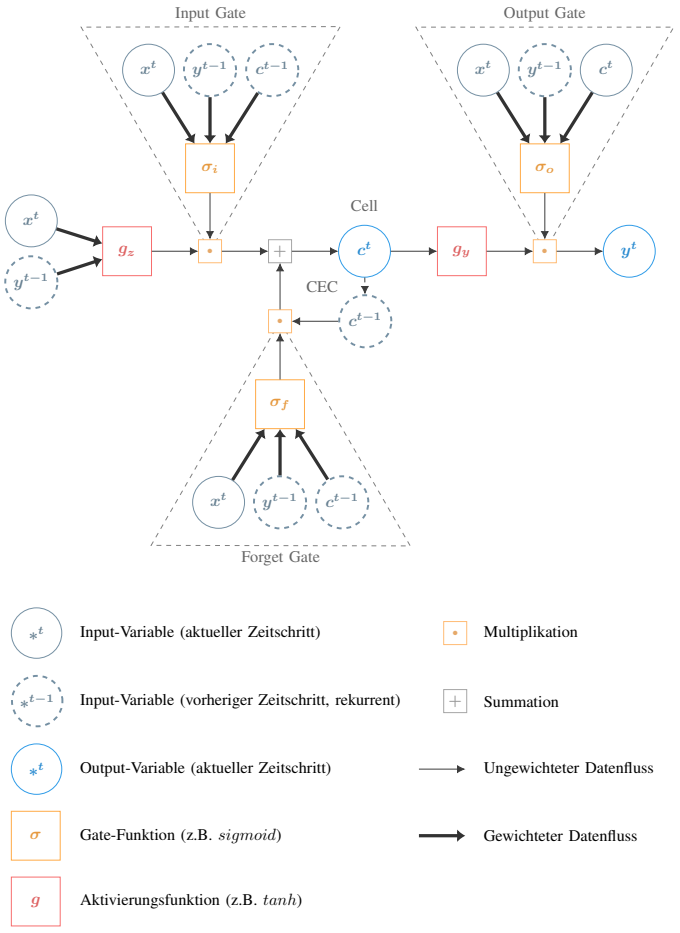


Abbildung 5: Standard LSTM-Schicht

$$x \in \mathbb{R}^n, \quad y, c \in \mathbb{R}^m$$

$$(y^t, s^t) = \varphi(x^t, s^{t-1}) \text{ mit } s = \begin{pmatrix} c \\ y \end{pmatrix} \tag{5a}$$

φ :

$$W_{\{z,i,f,o\}} \in \mathbb{R}^{m \times n}$$
$$R_{\{z,i,f,o\}} \in \mathbb{R}^{m \times m}$$
$$b_{\{z,i,f,o\}}, p_{\{i,f,o\}} \in \mathbb{R}^m$$
$$\sigma_{\{i,f,o\}} = sigmoid \text{ (o.ä.)}$$
$$g_{\{z,y\}} \in \{tanh, sigmoid, ReLU, ...\}$$

$$\begin{pmatrix} z \\ i \\ f \\ o \end{pmatrix} = \begin{pmatrix} g_z(W_z x^t + R_z y^{t-1} + b_z) \\ \sigma_i(W_i x^t + R_i y^{t-1} + p_i \odot c^{t-1} + b_i) \\ \sigma_f(W_f x^t + R_f y^{t-1} + p_f \odot c^{t-1} + b_f) \\ \sigma_o(W_o x^t + R_o y^{t-1} + p_o \odot c^t + b_o) \end{pmatrix} \tag{5b}$$

$$s^t = \begin{pmatrix} c^t \\ y^t \end{pmatrix} = \begin{pmatrix} f \odot c^{t-1} + i \odot z \\ o \odot g_y(c^t) \end{pmatrix} \tag{5c}$$

BPTT funktioniert auch bei LSTMs, dank durchgehender Differenzierbarkeit. Eine positive Besonderheit dabei ist das Constant Error Carousel (CEC, Abb. 5, Gleichung 6). Solange das Forget-Gate annähernd offen ist ($g_f \approx 1$), bleibt die Ableitung beim BPTT über diese Zeitschritte hinweg dementsprechend konstant. Vanishing Gradient "through time" wird somit gezielt kontrolliert, sodass bei Bedarf auch lange zurückliegende Informationen Einfluss auf den Lernprozess nehmen und somit Langzeitabhängigkeiten erlernt werden können.

$$\begin{aligned} \frac{\partial E^t}{\partial net^k} &= \frac{c^k}{net^k} \cdot \left(\prod_{j=k+1}^{t} \frac{\partial c^j}{\partial c^{j-1}} \right) \cdot \frac{\partial E^t}{\partial c^t} + \dots \\ &= \frac{c^k}{net^k} \cdot \left(\prod_{j=k+1}^{t} diag(f^j) + \dots \right) \cdot \frac{\partial E^t}{\partial c^t} + \dots \end{aligned} \tag{6}$$

C. Anwendungen

RNNs, insbesondere LSTMs und GRUs, werden in einer Vielzahl von Anwendungsgebieten eingesetzt, bei denen zeitlich oder räumlich sequentielle Verarbeitung bzw. Ein- und/oder Ausgaben variabler Länge eine Rolle spielen:

1-zu-* / Vektor-zu-Sequenz (Sequenz-Generierung) – Eine einzige Eingabe wird zu einer Ausgabe-Serie übersetzt. Beispielsweise kann so zu einem mittels CNN in einen Vektor codiertes Bild eine Bildbeschreibung generiert werden [8].

***-zu-1 / Sequenz-zu-Vektor (Sequenz-Codierung)** – Eine Eingabe-Serie wird zu einer einzigen Ausgabe übersetzt, etwa um einen Satz variabler Länge thematisch zu klassifizieren. Auch zur Berechnung von Satzwahrscheinlichkeiten für Neural Network Language Models (NNLMs) im Kontext von Natural Language Processing (NLP) sind RNNs einsetzbar [9].

***-zu-* / Sequenz-zu-Sequenz (Sequenz-Übersetzung)** – Eine Sequenz-zu-Sequenz-Übersetzung stellt den universellen Fall dar. Man kann zudem zwischen **synchroner** und **asynchroner** Übersetzung unterscheiden. Denkbare Einsatzgebiete sind beispielsweise Textübersetzung (Text-zu-Text, Abb. 6) [10], Spracherkennung (Sprache-zu-Text) [11] und Vorhersagen (*-zu-*) diverser Prozesse wie Wetter [12] oder Verkehrsfluss [13], sowie Anomalie-Erkennung, etwa von EKGs [14].

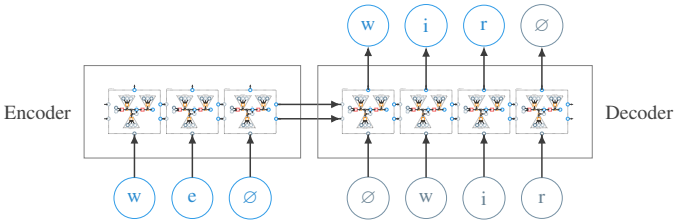


Abbildung 6: Encoder-Decoder. Eine einfache Architektur zur asynchronen Übersetzung, z.B. von Text. Die Eingabe wird in einen Embedding-Vektor codiert (Encoder) und daraus anschließend in die Ziel-Sprache decodiert (Decoder).

IV. Convolutional Neural Network (CNN)

Zum Verständnis der Umwelt ist die Interpretation visueller Daten respektive Bilder von Vorteil. Dafür ist weniger die absolute räumliche Position einer jeden Informationseinheit (z.B. Pixel) wichtig, sondern vielmehr deren relative Anordnung zueinander. Den visuellen Cortex eines biologischen Gehirns als Vorbild [15] [16], haben sich Convolutional Neural Networks (CNNs) entwickelt, welche neue Maßstäbe bei automatisierter Bildverarbeitung setzen. Noch allgemeiner können sie auf beliebige lokalitätsbedingte, also neben räumliche etwa auch zeitliche (z.B. akustische), Muster angewandt werden.

Kernstück bildet die diskrete Convolution-Operation (Gleichung 7[1], Abb. 7). Hierbei wird die bisherige Mustererkennung auf ein Raster, etwa ein Pixel-Bild, angewandt, indem ein Kernel K, welcher in diesem Fall als räumlich angeordnete Gewichte einer üblichen Perceptron-Schicht verstanden werden kann, über das Eingaberaster X gestreift wird und die damit gewichtete Summe des gerade betrachteten Bereichs den skalaren Wert an entsprechender Position im Ausgaberaster (Feature Map) Y darstellt.

Durch die Convolution-Operation ist eine gewisse Translationsinvarianz gewährleistet [16], die Verschiebung eines Musters beeinflusst also nicht dessen Erkennung. So kann jenes unabhängig von seiner Position mittels eines einzigen Filters erkannt und zugleich die relative räumliche Position dessen Abstraktion im Ausgaberaster beibehalten werden.

Zur Unterscheidung verschiedener Muster benötigt der Kernel entsprechend viele Filter. Jeder erzeugt ein eigenes Ausgaberaster. Übereinandergelegt ergibt sich daraus die Tiefendimension (Kanäle/Channels) der Ausgabe.

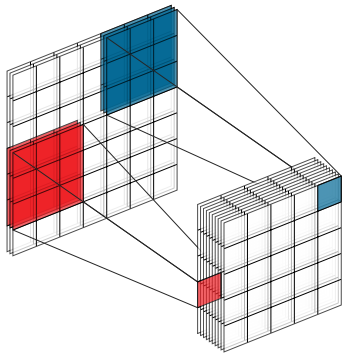


Abbildung 7: CNN-Schicht mit 3 Eingabe- und 10 Ausgabekanälen. (Übersichtshalber sind nur zwei Filter gezeigt.)

$$X \in \mathbb{R}^{h_x \times w_x \times d_x}, \quad K \in \mathbb{R}^{d_y \times h_k \times w_k \times d_x},$$
$$Y \in \mathbb{R}^{h_x - h_k + 1 \times w_x - w_k + 1 \times d_y}$$

$$Y = K * X = \left(\sum_m \sum_n K^T_{k,m,n} \cdot X_{i+m,j+n} \right)_{i,j,k} \tag{7}$$

[1]Gleichung 7 zeigt eigentlich Cross-Correlation (Kernel nicht gespiegelt), wird jedoch in dem Kontext dennoch als Convolution bezeichnet.

Der Prozess kann äquivalent zu DFNNs vielschichtig wiederholt (Abb. 8) und somit eine schrittweise Abstraktion durch Komposition kleiner, einfacher (z.B. senkrechter Strich) zu großen, komplexen (z.B. Auto) Mustern realisiert werden (Abb. 1).

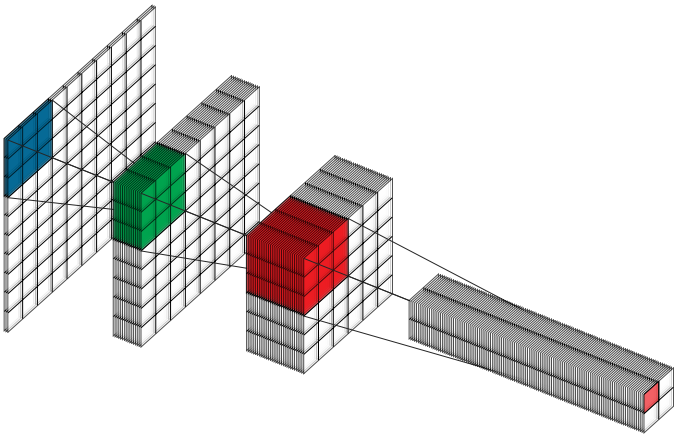


Abbildung 8: Mehrschichtiges CNN. Padding: valid, Stride: 1. (Übersichtshalber ist nur ein Filter pro Schicht gezeigt.)

Die Filtergröße sollte entsprechend so gewählt sein, dass das Sichtfeld zum zu erkennenden Muster passt. Dabei ist zu beachten, dass Filter höherer Schichten indirekt ein größeres Sichtfeld auf das ursprüngliche Eingaberaster haben (Abb. 9). Hohen Schichten erschließen sich somit größere bis alle Teile des Bildes, sodass sie diesbezüglich komplexere Abstraktionen vornehmen können (Abb. 1).

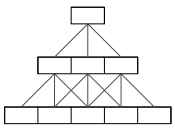
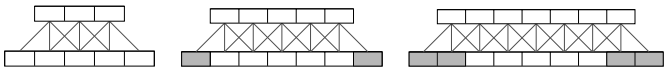


Abbildung 9: 1D-CNN. Der Filter mit Breite 3 in Schicht 2 hat indirekt ein Sichtfeld der Breite 5 auf Schicht 0.

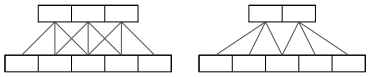
Padding - Da der Kernel nicht ohne Weiteres komplett über die Randbereiche der Eingabe gestreift werden kann, sind Muster dort meist schlechter erkennbar. Zudem schrumpfen die Schichten mit steigender Netztiefe ungewollt (Abb. 8). Eine Lösung ist Padding, wobei entlang der Ränder das Raster erweitert wird, etwa durch Nullen, Zero-Padding (Abb. 10).



(a) valid (b) same (c) full

Abbildung 10: 1D-CNN mit verschiedenem Padding.

Striding - Zur Beschleunigung sowie als einfache Downsampling-Methode “on the fly” kann die Schrittweite (Stride) während der Convolution vergrößert werden (Abb. 11). Kleine Teilmuster werden dadurch ggf. übersprungen, jedoch ist dies oft vernachlässigbar zu Gunsten der Effizienz.



(a) stride = 1 (b) stride = 2

Abbildung 11: 1D-CNN mit verschiedenem Striding.

Pooling - Mit etwas mehr Qualität, Kontrolle und Rechenaufwand verbunden ist das Pooling, eine weitere Downsampling-Variante. Hierfür wird eine separate Convolution-Operation (mit Stride > 1) durchgeführt, bei der das Eingabesichtfeld statt auf seine gewichtete Summe beispielsweise seinen Durchschnitt (Average-Pooling) oder sein Maximum (Max-Pooling) abgebildet wird.

Für die Basis-Variante eines vollständigen Netzes werden Convolution, nichtlineare Aktivierungsfunktion, evtl. Normalisierung sowie Pooling wiederholt kombiniert (Abb. 12). Letztlich wird die Ausgabe durch eine übliche voll verbundene Schicht zur weiteren Verarbeitung vektorisiert.

Abbildung 12: Beispiel eines einfachen CNN-Bausteins.

A. Backpropagation bei CNNs

CNNs sind durchgehend differenzierbar, also für Backpropagation geeignet. Ähnlich wie bei RNNs sind für die Ableitungen nach den Gewichten K (Gleichung 8a) die "shared parameters" zu beachten, die sich in diesem Fall dadurch ergeben, dass der selbe Kernel für alle Positionen der Ausgabeschicht verwendet wird. Die für die nächsttiefere Schicht notwendige Ableitung nach X entspricht wieder einer Convolution-Operation (Gleichung 8b).

$$\begin{aligned}\frac{\partial E}{\partial K} &= \left(\sum_i \sum_j \frac{\partial E}{\partial Y_{i,j}} \frac{\partial Y_{i,j}}{\partial K_{m,n}}\right)_{m,n} \\ &= \left(\sum_i \sum_j \frac{\partial E}{\partial Y_{i,j}} X_{i+m,j+n}\right)_{m,n} \qquad (8a)\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial X} &= \left(\sum_m \sum_n \frac{\partial E}{\partial Y_{i-m,j-n}} \frac{\partial Y_{i-m,j-n}}{\partial X_{i,j}}\right)_{i,j} \\ &= \left(\sum_m \sum_n \frac{\partial E}{\partial Y_{i-m,j-n}} K_{m,n}\right)_{i,j} \\ &= \left(\sum_m \sum_n K_{h_k-1-m,w_k-1-n} \right. \\ &\qquad \left. \cdot \frac{\partial E}{\partial Y_{i-h_k+1+m,j-w_k+1+n}}\right)_{i,j} \\ &= flip(K) * pad(Y) \qquad (8b)\end{aligned}$$

B. Anwendungen

CNNs sind prädestiniert für Bildverarbeitung. Angefangen mit Handschrifterkennung [17] haben sich immer komplexere Anwendungen ergeben, nicht zuletzt aufgrund zunehmender Rechenleistung, insbesondere mit GPUs:

***D-zu-0D / Bild-zu-Klasse (Bild-Klassifizierung)** - Das Labeling von Bildern und Objekten ist eine der momentan alltäglichsten Einsatzgebiete von CNNs, etwa zur Klassifizierung von handschriftlichen Zeichen [17], Straßenschildern [18], Gesichtern [19] und anderem [20]. Damit und durch automatische Erkennung von Aktionen [21] hat zudem die Überwachung eine neue Effizienzquelle gefunden. Anwendungen von CNNs sind aber beispielsweise auch für medizinische Bildanalyse denkbar, etwa zur Tumor-Segmentierung [22].

***D-zu-*D / Bild-zu-Vektor (Bild-Codierung)** - Ebenso gut können Bilder in einen Feature-Vektor codiert und anderweitig weiterverarbeitet werden. So etwa zur Bildbeschreibung (z.B. mittels RNN) [8] oder automatisierten Durchführung grafischer Spiele (z.B. Go [23], Atari [24]). Überdies können CNNs, da auch Text/Sprache lokalitätsbedingte Semantik aufweist, zwecks Spracherkennung mittels Convolution über das Audio-Frequenzband o.a. genutzt werden [25].

***D-zu-*D / *-zu-Bild (Bild-Generierung)** - Umgekehrt kann das Konzept von Convolution auch zur Generierung bzw. Manipulation von Bildern genutzt werden, beispielsweise zum Nachahmen von Kunst-Stilen [26] oder für Sprachsynthese [27]. Zudem können aufgrund semantischer Einbeziehung der Pixelumgebung verbesserte Methoden für Bild-Upsampling (Super Resolution, SR) realisiert werden [28].

V. Zusammenfassung und Ausblick

Diese Ausarbeitung hat grundlegende Aspekte zu Deep Learning (DL), insbesondere zweier seiner erfolgreichen Repräsentanten, Long Short-Term Memories (LSTMs) und Convolutional Neural Networks (CNNs), erläutert und Potential sowie Vielfalt anhand einiger Anwendungsgebiete dargestellt.

Während CNNs durch Convolution eine lokalitätsbedingte Mustererkennung für den effektiven Umgang mit visuellen Informationen ermöglichen, lassen sich mit LSTMs Langzeitabhängigkeiten bei sequentieller Verarbeitung erlernen und nutzen. Zusammen stellen sie zwei Deep-Learning-Architekturen dar, deren Essenz gegenwärtig eine zentrale Rolle hinsichtlich des Ziels der Simulation kognitiver Fähigkeiten zuzusprechen sein dürfte.

Auch in Anbetracht verstärkter Popularität in Industrie und bei Endnutzern kommt dem Themenkomplex rund um Deep Learning gegenwärtig viel Aufmerksamkeit zugute. Anhaltende Forschung wird weiterhin noch fortschrittlichere Methoden hervorbringen und so zur Lösung komplexer Aufgaben beitragen. Neben Hardware-Verbesserungen bleibt dabei die Entwicklung effizienter Algorithmen sehr bedeutsam. Derzeitige Grenzen wie vollständiges Videoverständnis, Multi-Task-Learning, notwendige Datenmengen und Energieeffizienz werden sich auf diese Weise zukünftig gewiss zu weiteren Erfolgen wandeln.

LITERATUR

[1] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

[2] J. Zweig, "Deep neural networks go to the movies," Feb. 2016. [Online]. Available: https://www.strong.io/blog/deep-neural-networks-go-to-the-movies

[3] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen," Diplomarbeit, Technische Universität München, 1991.

[4] H. T. Siegelmann and E. D. Sontag, "Turing computability with neural nets," *Applied Mathematics Letters*, vol. 4, no. 6, pp. 77–80, 1991.

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[7] F. A. Gers and J. Schmidhuber, "Recurrent nets that time and count," in *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, vol. 3. IEEE, 2000, pp. 189–194.

[8] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[9] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, "Exploring the limits of language modeling," *CoRR*, vol. abs/1602.02410, 2016.

[10] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," *CoRR*, vol. abs/1609.08144, 2016.

[11] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 2013, pp. 6645–6649.

[12] S. Balluff, J. Bendfeld, and S. Krauter, "Short term wind and energy prediction for offshore wind farms using neural networks," in *Renewable Energy Research and Applications (ICRERA), 2015 International Conference on*, 2015, pp. 379–382.

[13] Y. Tian and L. Pan, "Predicting short-term traffic flow by long short-term memory recurrent neural network," in *Smart City/SocialCom/SustainCom (SmartCity), 2015 IEEE International Conference on*, 2015, pp. 153–158.

[14] S. Chauhan and L. Vig, "Anomaly detection in ecg time signals via deep long short-term memory networks," in *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*, 2015, pp. 1–7.

[15] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cats visual cortex," *The Journal of Physiology*, vol. 160, no. 1, pp. 106–154, 1962.

[16] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[17] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[18] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. D. Shet, "Multi-digit number recognition from street view imagery using deep convolutional neural networks," *CoRR*, 2013.

[19] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE transactions on neural networks*, vol. 8, no. 1, pp. 98–113, 1997.

[20] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[21] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 221–231, 2013.

[22] M. Havaei, A. Davy, D. Warde-Farley, A. Biard, A. Courville, Y. Bengio, C. Pal, P.-M. Jodoin, and H. Larochelle, "Brain tumor segmentation with deep neural networks," *Medical image analysis*, vol. 35, pp. 18–31, 2017.

[23] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016, cited By 295.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[25] O. Abdel-Hamid, A. rahman Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 22, no. 10, pp. 1533–1545, oct 2014.

[26] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *CoRR*, vol. abs/1508.06576, 2015.

[27] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *CoRR*, vol. abs/1609.03499, 2016.

[28] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 2, pp. 295–307, 2016.