Seminararbeit

B.Sc.-Studiengang „Wirtschaftsingenieurwesen“

Universität Hamburg
Fakultät für Betriebswirtschaft
Lehrstuhl für Mathematik und Statistik in den Wirtschaftswissenschaften

Long Short-Term Memory (LSTM)

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Neuronale Netzwerke sind spätestens seit der Veröffentlichung des Large Language Models *ChatGPT* im November 2022 in den Mittelpunkt des öffentlichen Interesses gerückt. Die beeindruckenden Fähigkeiten dieser Modelle haben nicht nur ein breites gesellschaftliches Interesse geweckt, sondern auch zu einem Wettrennen zwischen den führenden KI-Unternehmen ausgelöst. Seitdem werden enorme Ressourcen in die Weiterentwicklung neuronaler Netzwerke investiert, was zu immer leistungsfähigeren Anwendungen führt und gleichzeitig in der Forschung Diskussionen über ihre möglichen Auswirkungen auf unsere Gesellschaft anstößt[1]. Die rapiden Fortschritte, die in den letzten Jahren gemacht wurden, täuschen jedoch darüber hinweg, dass die Anfänge der Neuronalen Netzwerke bereits 1943 gelegt wurden. Damals entwickelten der Neuropsychologe *Warren McCulloch* und der Mathematiker *Walter Pitts* mit dem sogenannten „McCulloch-Pitts Neuron (M-P Neuron)“[2] das erste formale Modell eines künstlichen Neurons und legten damit den Grundstein für die heutige Forschung auf diesem Gebiet.(Géron 2023, S. 334)

Diese Arbeit untersucht die Funktionsweise und die mathematischen Grundlagen von Long Short-Term Memory (LSTM)-Zellen und stellt die besonderen Fähigkeiten von LSTM-Netzwerken dar.

Zunächst werden die grundlegenden Konzepte klassischer neuronaler Netze vorgestellt. Darauf folgt eine Einführung in rekurrente neuronale Netzwerke, um die Entwicklung hin zu LSTM-Architekturen nachvollziehbar zu machen. Anschließend werden der Aufbau und die mathematischen Hintergründe der LSTM-Struktur detailliert dargestellt. Im weiteren Verlauf wird anhand einer Zeitreihenprognose die praktische Implementierung von LSTM-Modellen mit der Keras-Bibliothek demonstriert und verschiedene Ausführungen dieser verglichen. Abschließend wird die Bedeutung der LSTM-Architektur im Kontext aktueller Deep-Learning-Entwicklungen diskutiert und ein Vergleich mit der Transformer-Architektur gezogen.

[1] **International AI Safety Report 2025**
[2] **The Logical Calculus of the Ideas Immament in Nervous Activity on how neurons might work**

2 Theoretische Grundlagen

Frühe Modelle wie das M-P-Neuron wurden entwickelt, um logische Funktionen mit künstlichen Neuronen nachzubilden. Das M-P-Neuron arbeitet ausschließlich mit binären Eingabe- und Ausgabewerten und kann durch die Verschaltung mehrerer Neuronen beliebige logisch linear trennbare Ausdrücke darstellen(Géron 2023, S. 337). Aufbauend auf diesen frühen Konzepten wurden im Laufe der Zeit leistungsfähigere Architekturen entwickelt, die heute für komplexe Aufgaben eingesetzt werden. Im folgenden wird ein Überblick über die Architektur und die mathematischen Grundlagen neuronaler Netze gegeben.

2.1 Klassische neuronale Netze: Struktur und Funktionsweise

Die einfachste neuronale Architektur ist das „Perzeptron“, welches 1957 von Frank Rosenblatt erfunden wurde. Dieses verwendet als Neuronen „Threshold Logic Units (TLU)“, welche mit Reellen Zahlen arbeiten können. Das Perzeptron besitzt nur eine Eingabeschicht und eine Ausgabeschicht. Im einfachsten Fall, in dem die Ausgabeschicht aus nur einem Neuron besteht, kann es wie in Abbildung 1 dargestellt werden.
Die Eingaben $x_i \in \mathbb{R}$ werden jeweils mit einem Gewicht $w_i \in \mathbb{R}$ multipliziert, bevor sie in einer linearen Funktion aufsummiert werden, wobei noch ein Bias Wert b addiert wird: $a = \sum(w_i \cdot x_i) + b$. Dieser interne Wert a, wird einer Aktivierungsfunktion ϕ übergeben und dann ausgegeben $y = \phi(a)$. In Abbildung 2, ist dies in Vektorschreibweise abgebildet.(Aggarwal 2023, 5f)

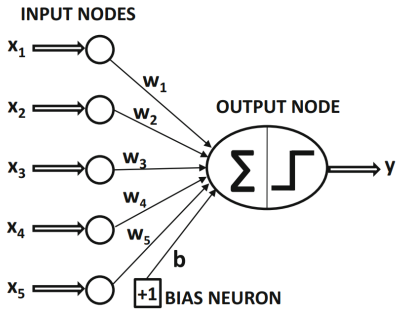


Abbildung 1: Perzeptron (Aggarwal 2023, S. 5)

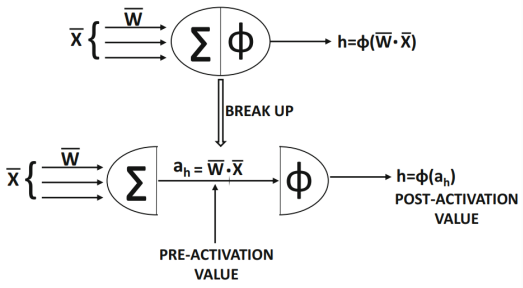


Abbildung 2: Perzeptron in Vektorschreibweise(Aggarwal 2023, S. 11)

Alle folgenden Neuronen können im Aufbau wie dieses betrachtet werden. Bias-Werte b werden in den weiteren Formeln nicht weiter explizit aufgeführt, da sie als gewichtete Ausdrücke behandelt werden können.

Die Aktivierungsfunktion ist essentiell für die Funktion und Effektivität des Netzwerks. In der Ausgabeschicht muss sie auf die jeweilige Aufgabe abgestimmt sein, zum Beispiel kann eine Schrittfunktion nur binäre Klassen klassifizieren. In tiefen neuronalen Netzwerken sorgen nicht-lineare Aktivierungsfunktionen dafür, dass jede Schicht eigene Aussagekraft behält und das Netzwerk komplexe Muster erlernen kann, da ein Netzwerk mit nur linearen Aktivierungen auf eine einzelne Schicht reduziert werden könnte.[3]

2.2 Trainieren von Neuronalen Netzen

Im Folgenden werden die mathematischen Grundlagen neuronaler Netzwerke auf Basis von Computational Graphs erläutert, angelehnt an *Neural Networks and Deep Learning*(Aggarwal 2023, Kapitel 1 und 2).

Neuronale Netzwerke speichern ihre erlernte Zusammenhänge in ihren Parametern (Gewichte und Bias), die bei der Modell-Initiierung normalerweise zufällig Standard-normalverteilt generiert werden. Das Training besteht anschließend daraus, diese Parameter so anzupassen, dass es seine Aufgabe besser erfüllt. Hierfür wird dem Modell eine *Verlustfunktion* $\mathcal{L}$ übergeben, die das Ziel des Netzwerks in mathematischer Form repräsentiert.

Sie entspricht der Güte der Vorhersage $\hat{y}$ des NN und ermöglicht die objektive Erfolgs-Bewertung.

$$\mathcal{L} = (y, \hat{y}) \tag{2.1}$$

Der *Optimierer* bestimmt das Lernverfahren um diese Verlustfunktion zu minimieren. Durch das verändern der einzelnen Gewichte w_i, wird die Weitergabe der Informationen h_i gesteuert. Das NN lernt, indem es einen Datenpunkt $\mathbf{X_i}$ (Instanz) aufnimmt, den Zielwert $\hat{y}_i$ ausgibt, und die Verlustfunktion $\mathcal{L}_i$ für diese Instanz berechnet. Anschließend werden die Gewichtungen $\mathbf{W}_i$ mit dem **Gradientenabstiegsverfahren(GV)** angepasst, um $\mathcal{L}_i$ zu minimieren. Dies geschieht, indem die partielle Ableitung der Verlustfunktion, nach den einzelnen Parametern $(w_1, w_2, ..., w_n) = \mathbf{W}_i$ berechnet wird. Der Gradient

$$\nabla \mathcal{L}_{i(\mathbf{W}_i)} = \frac{\partial \mathcal{L}_i}{\partial \mathbf{W}_i} = [\frac{\partial \mathcal{L}_i}{\partial w_1}, ..., \frac{\partial \mathcal{L}_i}{\partial w_n}]^T \in \mathbb{R}^n \tag{2.2}$$

gibt die Steigung jedes Parameters für den aktuellen Wert der Verlustfunktion wieder. Mit anderen Worten, wie stark ein Parameter zum Fehler beigetragen hat. Diese Steigung wird anschließend mit der Lernrate α multipliziert und von den aktuellen Parameterwerten ab-

[3] "Theorem 1.5.1 A multi-layer network that uses only the identity activation function in all its layers reduces to a single-layer network."(Aggarwal 2023, S. 17)

gezogen.

$$\mathbf{W}_{i+1} = \mathbf{W}_i - \alpha \frac{\partial \mathcal{L}_i}{\partial \mathbf{W}_i} \tag{2.3}$$

Durch die Verwendung nicht-linearer Aktivierungsfunktionen, können neuronale Netze meistens nicht in geschlossener Form dargestellt werden, weshalb der Backpropagation Algorithmus[4] eingesetzt wird. Dieser nutzt die Kettenregel,

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x} \tag{2.4}$$

um die Gradienten schichtweise rückwärts durch das Netzwerk zu berechnen. Der Ablauf der Backpropagation gestaltet sich wie folgt:

1. Die Eingaben $\mathbf{X}_i$ werden dem NN übergeben, welches jede Schicht durchläuft und die Vorhersage $\hat{y}_i$ berechnet.
2. Die Verlustfunktion $\mathcal{L}_i$ wird berechnet.
3. Die Gradienten werden berechnet, indem das Netzwerk von der Ausgabeschicht ausgehend schichtweise rückwärts durchlaufen wird

2.3 Rekurrente neuronale Netze: Konzept und Limitationen

Eine großer Schwäche der konventionellen Neuronalen Architekturen stellt das Verarbeiten von sequentiellen Daten dar. Hierzu gehören beispielsweise Zeitreihen (Daten die zeitlich geordnet sind und Bezug aufeinander nehmen) und Sprachen (Wörter die aufeinander Bezug nehmen).(Aggarwal 2023, S.265)

Rekurrente neuronale Netze (RNN) wurden entwickelt um mit diesen umzugehen. In einem RNN werden die Zeitschritte einer Eingabesequenz nacheinander in chronologischer Reihenfolge verarbeitet. Hierbei wird das Ergebnis jedes Zeitschritts, der sogenannte Hidden State $h^{(t)}$, als zusätzliche Eingabe an den nächsten Zeitschritt $(t+1)$ übergeben. Dadurch kann das Netzwerk Informationen aus der Vergangenheit $(t-n)$ in die aktuelle Vorhersage $\hat{y}^{(t)}$ miteinbeziehen und so auch zeitliche Muster erkennen.

Solche Zellen eines NN, in denen Informationen aus der Vergangenheit gespeichert bleiben, nennt man „Gedächtniszellen“. Da jedes Neuron einer Schicht die Information an das nächste

[4]Kombination aus Gradientenabstiegsverfahren und Reverse Mode Automatic Differentation

Neuron weitergibt, steht in der Theorie selbst dem letzte Neuron die Information aus dem ersten Zeitschritt zur Verfügung. In der Praxis, überlagern die neuen Datenpunkte jedoch die älteren zunehmend, was zu dem Problem des „Short Term Memory“ führt und nach ungefähr ~ 10 Zeitschritten , ist die Information verloren.(Géron 2023, S.580)

Dies liegt am Problem der *verschwindenden* bzw. *explodierenden Gradienten*, welcher auch die Trainierbarkeit eines RNN erschwert. Analog zu NN wird zum optimieren der Gradienten eines RNN, die Backpropagation Through Time (BPTT) genutzt:
Dabei wird im Vorwärtsdurchgang $\hat{y}^{(t\,=\,n)}$ für die Sequenz $t \in (1, .., n)$ berechnet. Anschließend wird die Verlustfunktion $\mathcal{L} = (y, \hat{y})$ berechnet und das Netzwerk rückwärts durch die Zeit durchpropagiert. Da die Gewichte $\mathbf{W}_h = [\mathbf{W}_{hh},\ \mathbf{W}_{hx}]$ während der gesamten Sequenz konstant bleiben, führt dies zu folgenden Problem im Gradientenabstiegsverfahren:

1. $|\mathbf{W}_{hh}| < 1$ führt im Training zu einem verschwindenden Gradienten:
   Mit voranschreitenden Epochen: $E \uparrow \quad \Rightarrow \quad \nabla\mathcal{L}_{(\mathbf{W}_h)} = \frac{\partial\mathcal{L}}{\partial\mathbf{W}_h} \approx 0$
   $\rightarrow$ Vorschnelles lokales Optimum: $\mathbf{W}_{h_{E+1}} = \mathbf{W}_{h_E} - 0 = const$

2. $|\mathbf{W}_{hh}| > 1$ führt zu einem explodierenden Gradienten:
   Mit voranschreitenden Epochen: $E \uparrow \quad \Rightarrow \quad \nabla\mathcal{L}_{(\mathbf{W}_h)} = \frac{\partial\mathcal{L}}{\partial\mathbf{W}_h} \approx \pm\,\infty$
   $\rightarrow$ Gewichte divergieren im Training: $\mathbf{W}_{h_{E+1}} = \mathbf{W}_{h_E} - (\pm\,\infty) = \pm\,\infty$

Am Beispiel eines RNN mit nur einer Zelle pro Schicht, ist es mathematisch sehr gut darstellbar, dass bei einem gleichbleibendem Gewicht $W_{(hh)} = const$ und über eine Sequenz mit T Zeitschritten, der Faktor $W^T_{(hh)}$ entstehen wird: (Aggarwal 2023, S. 279)

$$\frac{\partial\mathcal{L}}{\partial h^{(t-1)}} = \frac{\partial\mathcal{L}}{\partial h^{(t)}} \cdot \frac{\partial h^{(t)}}{\partial h^{(t-1)}} = \frac{\partial L}{\partial h^{(t)}} \cdot \left[\phi'(W^{(t)}_{(hh)} h^{(t-1)}) \cdot W^{(t)}_{(hh)}\right] \tag{2.5}$$

3 Long Short-Term Memory (LSTM) - Netzwerke

An diesem Problem setzt die Architektur der „Long Short-Term Memory"-Zelle an, welche 1997 von Sepp Hochreiter und Jürgen Schmidhuber vorgestellt wurde[5]. Ihre Weiterentwicklung der RNN-Zelle baut darauf auf, dem Netzwerk zu ermöglichen ein Langzeitgedächtnis zu formen um relevante Informationen über viele Zeitschritte hinweg zu speichern und gezielt zu nutzen. Diese Funktion übernimmt der sogenannte Cell-State C_t.
Mit einfachen Worten, befähigt die LSTM-Architektur das NN dazu, die zukünftige Relevanz von Datenpunkten, sowie die aktuelle Relevanz von vorherigen Datenpunkten, zu erlernen. Wodurch diese Variable ausschlaggebend dafür ist, dass Problem des Vanishing Gradient im Prozess der Backpropagtion Through Time zu verhindern.

3.1 Architektur und Komponenten von LSTM-Netzwerken

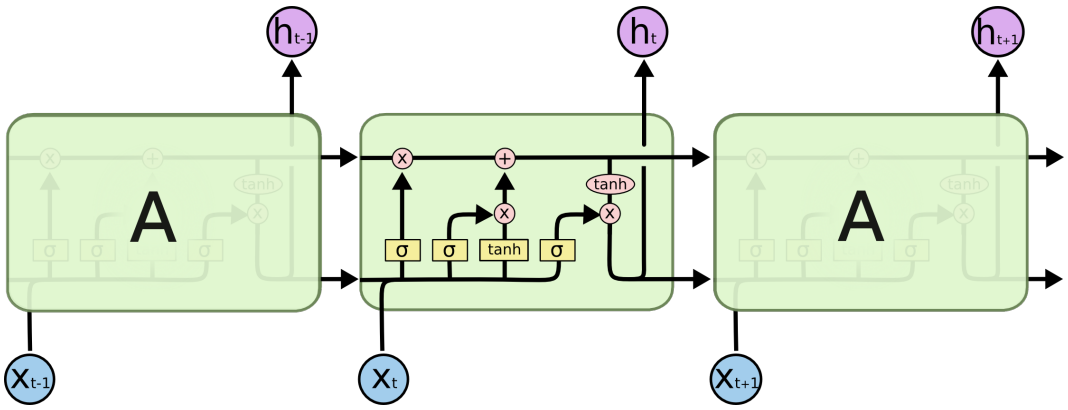
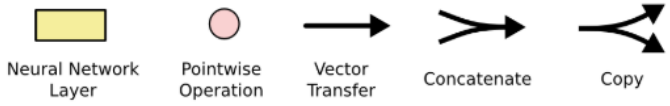


Abbildung 3: einfache LSTM-Zelle mit 3 Zeitschritten
(Olah 2015)



In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

Abbildung 4: Notation
(Olah 2015)

[5] **Long Short-Term Memory** DOI:10.1162/neco.1997.9.8.1735

Die Fähigkeit zur Relevanz-Abschätzung in LSTM-Netzwerken wird durch drei interne Gates realisiert: das Forget Gate (f_t), das Input Gate (i_t) und das Output Gate (o_t). Diese Gates steuern, welche Informationen im Cell State C_t gespeichert, überschrieben oder gelöscht werden. Die zugehörigen Gewichtsmatrizen (W_f, W_i, W_o) werden während des Trainings optimiert und bleiben anschließend konstant.

Im Unterschied zu klassischen RNNs, die Informationen ausschließlich über den Hidden-State h_t weitergeben und dadurch ältere Informationen schnell überlagern, besitzt die LSTM-Zelle durch die manipulation des Cell-State C_t die Fähigkeit, wichtige Informationen über viele Zeitschritte hinweg zu bewahren, gezielt abzurufen oder zu vergessen, wenn sie nicht mehr relevant sind.

Mathematisch hat das auf die BPTT folgenden Einfluss:(eigene Herleitung, nach den Formeln von (Olah 2015))

Durch den Cell-State C_t kann der Hidden-State h_t dargestellt werden als:

$$h_t = o_t \odot \tanh(C_t) = o_t \odot \tanh(f_t \odot C_{t-1} + i_t \odot \tilde{C}_t) \tag{3.1}$$

wodurch die Möglichkeit entsteht, die folgende partielle Ableitung aufzustellen:

$$\frac{\partial h_t}{\partial C_t} \cdot \frac{\partial C_t}{\partial C_{t-1}} \cdot \frac{\partial C_{t-1}}{\partial \tilde{C}_{t-1}} \cdot \frac{\partial \tilde{C}_{t-1}}{\partial h_{t-2}} \tag{3.2}$$

$$= [o_t \odot \tanh'(C_t)] \odot [f_t] \odot [i_{t-1}] \odot [\tanh'(W_{Cx} \cdot x_{t-1} + W_{Ch} \cdot h_{t-2}) \odot W_{Ch}] \tag{3.3}$$

Hier entsteht im Gegensatz zu den klassischen RNN nur einmal der konstante Faktor W_{Ch}. Dadurch bleibt der Gradient auch über viele Zeitschritte hinweg stabil, was das Problem des verschwindenden bzw. explodierenden Gradienten effektiv verhindert.

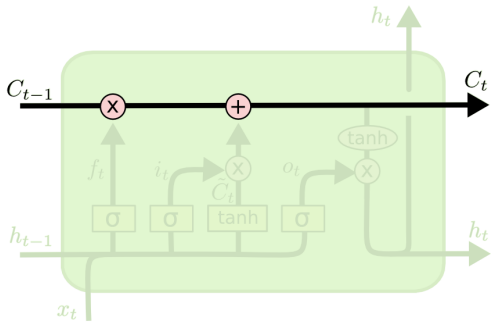


Abbildung 5: Cellstate-Highway
(Olah 2015)

3.2 Funktionsweise der Gates (Forget, Input, Output)

In jedem Zeitschritt werden einer LSTM-Zelle drei Vektoren übergeben:

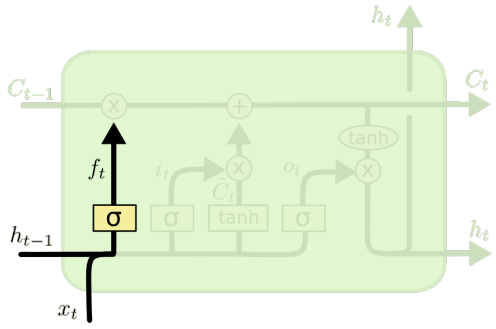
- x_t: Merkmale zum Zeitpunkt t
- h_{t-1}: Hidden State (Kurzzeitgedächtnis) vom vorherigen Zeitschritt
- C_{t-1}: Cell State (Langzeitgedächtnis) vom vorherigen Zeitschritt

Die Gates sind jeweils die Ausgabe eines kleinen neuronale Netze mit einer Sigmoid-Aktivierung σ, die x_t und h_{t-1} als Eingabe erhalten. Ihre Bezeichung als Gate kommt daher, dass die einzelnen Werte in diesen Vektoren zwischen 0 und 1 liegen und dadurch steuern, wie stark Informationen weitergegeben, gespeichert oder gelöscht werden. Nach ihrer Berechnung werden sie in einer Hadamard-Multiplikation verwendet, wo sie am Beispiel des Forget-Gates folgende Auswirkung haben:
Für $f_1 = 0$, wird die Information in $C_{1_{t-1}}$ gelöscht, während bei $f_4 = 1$ die Information $C_{4_{t-1}}$ unverändert übernommen wird. Visuell kann man das sehr gut veranschaulichen:

$$\mathbf{C}_{t-1} \odot \mathbf{f}_t \Leftrightarrow \begin{bmatrix} C_{1_{t-1}} \\ C_{2_{t-1}} \\ C_{3_{t-1}} \\ C_{4_{t-1}} \end{bmatrix} \odot \begin{bmatrix} 0 \\ 0,2 \\ 0,6 \\ 1 \end{bmatrix} \Leftrightarrow \begin{bmatrix} 0 \\ 0,2 \cdot C_{2_{t-1}} \\ 0,6 \cdot C_{3_{t-1}} \\ C_{4_{t-1}} \end{bmatrix} \tag{3.4}$$

Forget Gate f_t

Im ersten Schritt, wird das Forget Gate berechnet. Dieses Gate beinhaltet, wie bereits im Beispiel beschrieben, die Anweisung welche Informationen aus dem bisherigen Langzeitgedächtnis C_{t-1} veraltet sind und gelöscht werden sollen.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

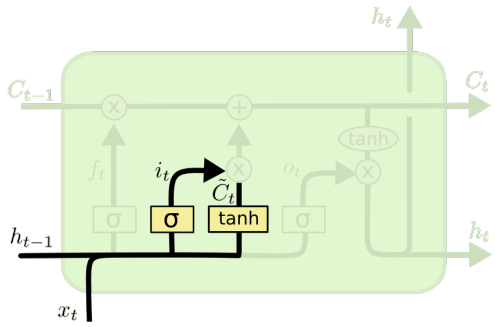
Abbildung 6: Aktivierung des Forget-Gates
(Olah 2015)

New Cell-State $\tilde{C}_t$

Dieser Vektor repräsentiert die Informationen aus dem Kurzzeitgedächtnis h_{t-1} und der Eingabe x_t, welche die LSTM-Zelle als wichtigen Zusammenhang definiert hat. Im Gegensatz zu den Gates, ist hier die Aktivierungsfunktion eine *tanh* Funktion, die Werte können im Bereich $[-1, 1] \in \mathbb{R}$ liegen

Input Gate i_t

Das Input Gate gibt die Informationen aus demNew Cell-State $\tilde{C}_t$ für das Langzeitgedächtnis C_t frei. Es entspricht einem Gradgeber, der überwacht, **ob** ein Zusammenhang relevant genug ist, um ihn zu speichern.

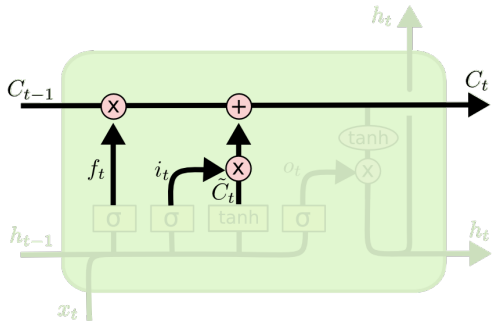


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Abbildung 7: Aktivierung des Input-Gates
(Olah 2015)

Langzeitgedächtnis C_t

Im Anschluss wird, wie in Abbildung 8 dargestellt, das Langzeitgedächtnis C_t aktualisiert. Dabei werden durch zwei Hadamard-Multiplikationen (elementweise Multiplikation) zunächst veraltete Informationen in C_{t-1} und irrelevante Informationen in $\tilde{C}_t$ entfernt. Anschließend werden die resultierenden Vektoren elementweise zu C_t addiert und an die nächste LSTM-Zelle der Schicht weitergeleitet.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

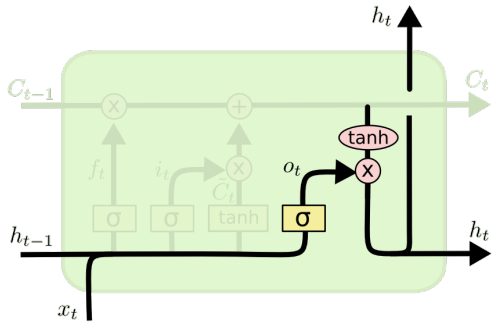
Abbildung 8: Berechnung des neuen Cell-State
(Olah 2015)

Output Gate o_t

Dieses Gate fungiert als Entscheider darüber, welche Informationen aus dem aktualisierten Langzeitgedächtnis C_t für den aktuellen Zeitschritt relevant sind und als Ausgabe bzw. Kurzzeitgedächtnis weitergegeben werden soll.

Ausgabe/Kurzzeitgedächtnis h_t

In der letzten Berechnung werden die Werte des Vektors C_t zunächst mit einer tanh-Aktivierungsfunktion in den Bereich $[-1, 1] \in \mathbb{R}$ transformiert[6]. Anschließend bestimmt das Output Gate o_t durch elementweise Multiplikation , welche Werte in den Hidden State h_t übernommen werden. Der resultierende Hidden State h_t wird sowohl an die nächste Schicht des neuronalen Netzes weitergegeben, als auch dem nächsten Zeitschritt $t+1$ bereitgestellt.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Abbildung 9: Berechnung des neuen Hidden-State
(Olah 2015)

[6]Durch die *elementweise-Addition*, kann der $\mathbf{C}_t$ Vektor außerhalb des Bereichs $[-1, 1]$ liegen

4 Anwendungsgebiete von LSTM

Durch ihre beschriebene Architektur haben LSTM-Netzwerke nach ihrer Einführung schnell das Einsatzspektrum RNN maßgeblich erweitert und dominiert. Sie werden überall dort eingesetzt, wo sequenzielle Daten analysiert und verarbeitet werden müssen. Zu den typischen Anwendungsgebieten zählen(Yu u. a. 2019):

Sprache:	Texterzeugung, Übersetzung, Sentiment-Analyse
Audio:	Spracherkennung, Musikerzeugung
Zeitreihen:	Finanzprognose, Wettervorhersage, Energieverbrauch
Anomalien:	Betrugserkennung, Netzwerküberwachung

Im folgendem wird anhand einer univarianten Zeitreihenanalyse die Anwendung von LSTM-Netzwerken demonstriert und auf verschiedene Modellvarianten eingegangen. Der Schwerpunkt liegt auf dem Vergleich verschiedener Modellarchitekturen. Aus diesem Grund wird eine autoregressive Zeitreihe ohne ausgeprägte Trends oder starkes Rauschen verwendet, sodass keine aufwändige Vorverarbeitung notwendig ist.

4.1 Modellaufbau

Der Datensatz basiert auf öffentlich verfügbaren Daten[7] zur deutschen Stromerzeugung der letzten zehn Jahre. Da es sich um eine autoregressive Zeitreihe handelt, werden keine weiteren Eingabedaten verwendet. Zur Steigerung der Vorhersagegenauigkeit könnten jedoch weitere, mit der Stromerzeugung korrelierte Variablen (Feiertage, Wetterprognosen, Wirtschaftskennzahlen) mit einbezogen werden.

Das Ziel der Analyse ist die stündliche Vorhersage der Stromerzeugung für die nächsten 24 Stunden. Als Eingabe erhält das Modell die stündlichen Erzeugungsdaten, der letzten 336 Stunden (28 Tage). Die Qualität der Vorhersagekraft des Modells wird anhand des Mean Absolute Error (MAE) auf dem Validierungsdatensatz bewertet.

Verglichen werden folgende Modellvarianten:

1. Architektur: LSTM-Layer vs. GRU-Layer
2. Netzwerktiefe: Ein-Layer, mit je 64 Neuronen vs. (Zwei-Layer, mit je 32 Neuronen)
3. Prognoseart: 24h direkt vs. 1h iterativ über 24h

[7] **Bundesnetzagetur**

Das Training der Modelle erfolgte über 100 Epochen mit folgenden Parametern:

Optimierer:	ADAM: lernrate $\alpha = 0.005$
ReduceLearningrate:	Faktor 0.5, bei Plateu der Verlustfunktion
Regularisierung:	Dropout: 0.1, (Recurrent-Dropout: 0.3)
Batchsize:	64, (128)
Verlustfunktion:	Huber mit $\delta = 1$

Die Huber-Verlustfunktion[8] ist definiert als: (Hastie, Tibshirani und Friedman 2009, S. 349)

$$\mathcal{L}_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{, falls } |y - \hat{y}| \leq \delta \\ \delta \cdot (|y - \hat{y}| - \frac{1}{2}\delta) & \text{, sonst} \end{cases} \tag{4.1}$$

4.2 Modellevaluation

Die visuelle Darstellung der Trainingsverläufe, ist im Anhang: *Modellvergleich* zu finden. Zur Modellevaluation sind in der folgenden Tabelle die besten MAE-Werte angegeben, welche die Modelle auf dem Validierungsdatensatz erreicht haben. Inklusive der Epoche, in welcher das Ergebnis erzielt wurde:

Tiefe	Architektur	Prognoseart	MAE (Validation)	Epoche
1 Layer	LSTM	24h direkt	0.3646	54
1 Layer	GRU	24h direkt	0.3543	100
1 Layer	LSTM	1h iterativ	0.1580	98
1 Layer	GRU	1h iterativ	0.1480	67
2 Layer	LSTM	24h direkt	0.3249	100
2 Layer	GRU	24h direkt	0.2991	100
2 Layer	LSTM	1h iterativ	0.0980	42
2 Layer	GRU	1h iterativ	0.0987	54

Tabelle 1: Tabellarischer Vergleich der besten MAE-Werte, welche die LSTM- und GRU-Modelle auf dem Validierungsdatensatz erreicht haben. Mit der Angabe der Epoche, in welcher das Ergbenis erzielt wurde.

Die Ergebnisse aus Tabelle 1 lassen mehrere charakteristische Muster erkennen, die für das Verständnis der verschiedenen Architekturentscheidungen von zentraler Bedeutung sind.

Einfluss der Netzwerktiefe:

Eine zusätzliche rekurrente Schicht führt durchgängig zu einer Verbesserung des Validierungs-MAE. Besonders bemerkenswert ist dabei das unterschiedliche Konvergenzverhalten: Die 24h-direkten Prognosemodelle zeigen nach 100 Epochen noch keine Anzeichen von Overfitting und erreichen ihr Validierungsminimum erst am Ende des Trainingszyklus. Im Ge-

[8] Keras `https://keras.io/api/losses/regression_losses/#huber-function`

gensatz dazu konvergieren die 1h-iterativen Varianten deutlich früher (Epochen 42-67) und zeigen anschließend typische Overfitting-Symptome. Dieses Phänomen erklärt sich durch die unterschiedliche Aufgabenkomplexität: Während die simultane 24h-Prognose eine hochdimensionale Ausgabe erfordert und somit die erweiterte Modellkapazität sinnvoll nutzen kann, führt die geringere Komplexität der 1h-Einzelvorhersage bei tieferen Architekturen schneller zur Überanpassung.

Architekturvergleich LSTM vs. GRU:

Die GRU-Architektur erzielt in nahezu allen Konfigurationen leicht bessere oder gleichwertige Ergebnisse (einzige Ausnahme: 2-Layer iterativ mit marginalem Unterschied). Diese Überlegenheit ist bei der relativ einfachen autoregressiven Vorhersageaufgabe durchaus erwartbar, da die reduzierte Parameterzahl der GRU-Zellen bei begrenzter Aufgabenkomplexität eine bessere Bias-Varianz-Balance ermöglicht und das Risiko der Überanpassung verringert.

Prognosestrategie-Evaluation:

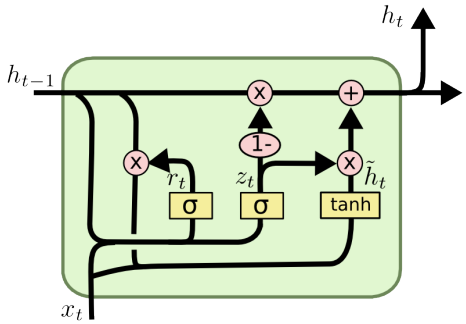
Die erheblichen MAE-Unterschiede zwischen 1h-iterativen und 24h-direkten Modellen sind primär methodisch bedingt. Während bei der direkten Strategie der MAE als Mittelwert über alle 24 prognostizierte Stunden berechnet wird, erfasst die iterative Bewertung ausschließlich die Abweichung der jeweils nächsten Stunde. Diese fundamentale Unterschied in der Evaluationslogik macht die Werte nicht direkt vergleichbar.
Aus praktischer Sicht ist jedoch die 24h-direkte Prognose der iterativen Variante vorzuziehen, da sich bei der iterativen Vorhersage Prognosefehler kumulativ aufaddieren und zu einem zunehmenden Abweichen von den tatsächlichen Werten führen.

Fazit:

Die Analyse demonstriert eindeutig den Vorteil zusätzlicher Netzwerktiefe bei angemessener Aufgabenkomplexität. Für die vorliegende Anwendung der Stromerzeugungsprognose erweist sich eine 2-Layer-GRU-Architektur mit 24h-direkter Prognose als optimale Konfiguration, da sie die beste Balance zwischen Modellkapazität, Generalisierungsfähigkeit und praktischer Anwendbarkeit bietet.

5 Evaluation und Vergleich

Wie in Kapitel 3 gezeigt, löst die LSTM-Architektur das Problem der schlechten Trainierbarkeit klassischer RNNs und ermöglicht die Verarbeitung längerer Sequenzen(Aggarwal 2023, S.285). Durch ihren komplexeren Aufbau erhöht sich die Rechenlast allerdings deutlich: Für die Berechnung der Gates, sowie die mehrmalige Manipulation des Langzeitgedächtnisses C_t wird erheblich mehr Rechenkapazität als bei einer einfachen RNN-Zelle benötigt. Zusätzlich steigt durch die vielzahl interner Gewichte die Gesamtanzahl der zu trainierenden Parameter (Aggarwal 2023, S.287f).
Um diese Komplexität zu reduzieren, wurde 2014 die Gated Recurrent Unit (GRU) als eine Variation der LSTM-Zelle eingeführt.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Abbildung 10: Gated Recurrent Unit (GRU)
(Olah 2015)

Sie verringert die Komplexität durch zwei Veränderungen erheblich:

1. Die Forget- und Output-Gates werden zu einem Update Gate z_t zusammengefasst.
2. Es wird keine zusätzliche Langzeitgedächtnis-Variable verwendet, sondern ausschließlich der Hidden-State h_t weitergegeben.[9]

In ihrer Funktion ist die GRU dem LSTM aber sehr ähnlich. Das Reset-Gate r_t entscheidet welche Informationen in h_{t-1} in der jetzigen Sequenz wichtig sind. Das Update Gate z_t entscheidet darüber, ob Informationen relevant genug sind, um sie in h_t zuspeichern und welche Informationen aus h_{t-1} gelöscht werden sollen. Allerdings bezieht es sich auf den internen Hidden-State $\tilde{h}_t$, der analog zum New-Cell-State $\tilde{C}_t$ im LSTM, dass Ergebnis einer *tanh*-Aktivierungsfunktion ist. Anders als im LSTM, wird in der Berechnung aber der bereits vom Reset-Gate r_t gefilterte Hidden-State $(h_{t-1} \odot r_t)$ zusammen mit dem Input x_t für die Berechnung verwendet. Zum Schluss wird das Ergebnis aus $(z_t \odot \tilde{h}_t)$ dann mit $((1 - z_t) \odot h_{t-1})$

[9] Dies ist gleichbedeutend damit, dass die GRU-Zelle kein separates internes Gedächtnis ($\tilde{C}_t$) hat.

elementweise addiert und als neuer Hidden-State h_t weitergegeben.
In der Praxis zeigt sich, dass GRU-Modelle eine höhere Recheneffizienz sowie eine einfachere Implementierung bieten. Aufgrund diese geringeren Komplexität sind sie bei kleineren Datensätzen resistenter gegen ein Overfitting auf dem Trainingssatz, und bieten infolgedessen eine bessere Generalisierung. Bei großen Datensätzen hingegen profitieren LSTM-Modelle von ihrer höheren Parameterzahl, da sie komplexere Zusammenhänge zwischen den Daten abbilden können(Aggarwal 2023, S.287-289). Zurückkommend auf die Einleitung dieser Arbeit bildet die Transformer-Architektur die Grundlage für Modelle wie den General Purpose Transformer (GPT) und bietet deutliche Vorteile gegenüber klassischen RNNs. Wie RNNs können Transformer Sequenzen als Eingaben verarbeiten, benötigen jedoch kein rekursives Feedback. Stattdessen verwenden sie eine Kombination aus einem „Attention"- Mechanismus und einer Encoder-Schicht. Diese Berechnungen sind im Gegensatz zu rekursiven Aufrufen hochgradig parallelisierbar, was eine effiziente Nutzung moderner Graphics Processing Units (GPUs) ermöglicht. Dies stellt eine der größten Schwächen der RNN dar, deren rekursive Struktur sequentielle Berechnungen und damit eine geringere Trainingsgeschwindigkeit erzwingt. Aus diesem Grund können Transformer auf vielen GPUs gleichzeitig trainiert werden, was einer größeren Rechenkraft entspricht, wodurch größere Datensätze verwendet werden können, was nahezu zwangsläufig in einer besseren Performance des Modells mündet.(Aggarwal 2023, S.446)

6 Fazit

Im Verlauf dieser Arbeit wurden die Funktionsweise und die mathematischen Grundlagen von LSTM-Netzwerken umfassend dargestellt. Es wurde gezeigt, dass LSTM-Architekturen durch ihre spezielle Struktur in der Lage sind, langfristige Abhängigkeiten in sequenziellen Daten zuverlässig zu erfassen und das Problem des verschwindenden Gradienten, das bei klassischen rekurrenten neuronalen Netzen häufig auftritt, wirksam zu umgehen. Die Analyse der internen Gates und des Cell-States verdeutlichte, wie LSTM-Zellen relevante Informationen über längere Zeiträume hinweg speichern und selektiv abrufen können.
Die praktische Vorhersage einer Zeitreihenprognose mit der Keras Bibliothek hat die Leistungsfähigkeit von LSTM-Modellen in realen Anwendungsfällen bestätigt.

Mit dem Aufkommen der Transformer-Architektur hat sich das Feld der sequenziellen Datenverarbeitung weiterentwickelt. Transformer-Modelle bieten insbesondere bei sehr langen Sequenzen und großen Datenmengen Vorteile hinsichtlich Parallelisierbarkeit und Modellleistung. Dennoch bleiben LSTM-Netzwerke in vielen spezialisierten Anwendungsbereichen relevant, etwa bei begrenzten Ressourcen oder spezifischen Aufgabenstellungen.

Zusammenfassend lässt sich festhalten, dass LSTM-Netzwerke einen wichtigen Meilenstein in der Entwicklung neuronaler Netze darstellen und weiterhin eine bedeutende Rolle in der Analyse sequenzieller Daten spielen. Für zukünftige Arbeiten erscheint es vielversprechend, hybride Ansätze oder spezialisierte LSTM-Varianten zu erforschen, um die jeweiligen Stärken verschiedener Architekturen gezielt zu kombinieren.

Anhang: Datensatz

Bundesnetzagetur: `https://www.smard.de/home/downloadcenter/download-marktdaten/`

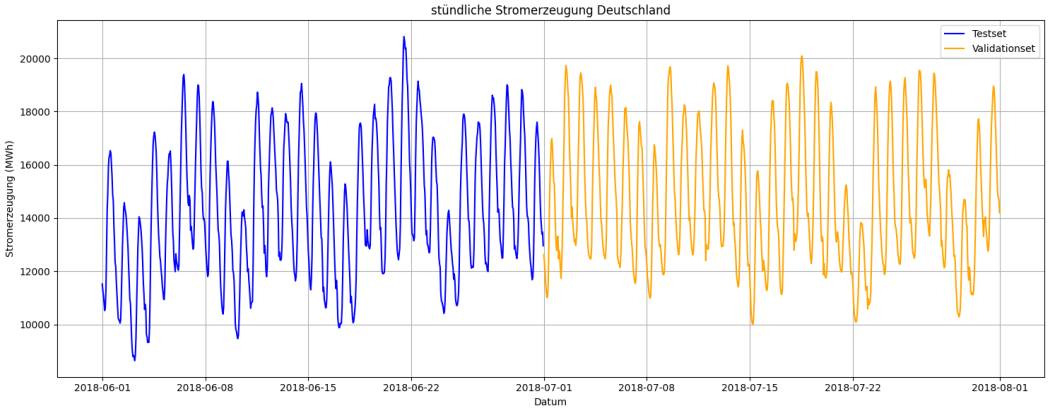


Abbildung 11: Datensatz, stündliche Auflösung, Zeitraum:

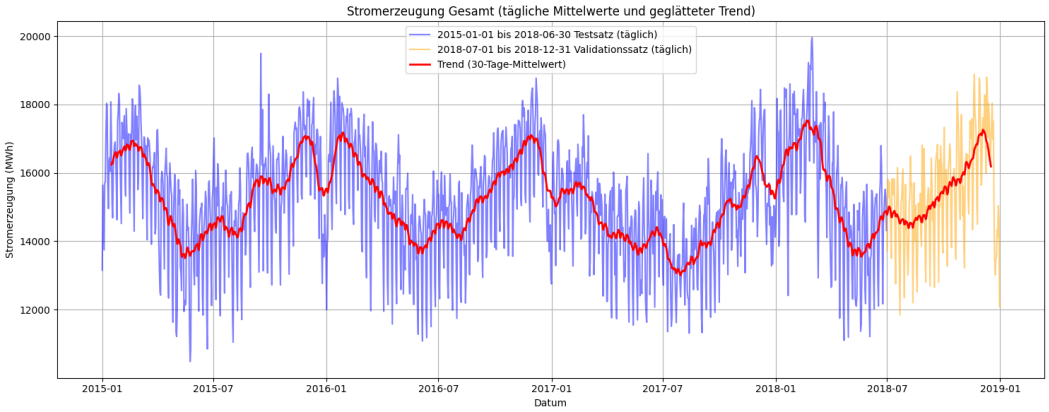


Abbildung 12: Datensatz, tägliche Auflösung, Zeitraum:

	Stromerzeugung Gesamt
2015-01-01 00:00	12884.25
2015-01-01 01:00	12552.0
2015-01-01 02:00	12394.5
2015-01-01 03:00	12112.5
2015-01-01 04:00	12042.5
2015-01-01 05:00	12081.0
2015-01-01 06:00	11494.25
2015-01-01 07:00	11655.0
2015-01-01 08:00	11859.0
2015-01-01 09:00	11780.75

Abbildung 13: Datensatz, die ersten zehn Einträge

Anhang: Modellvergleich

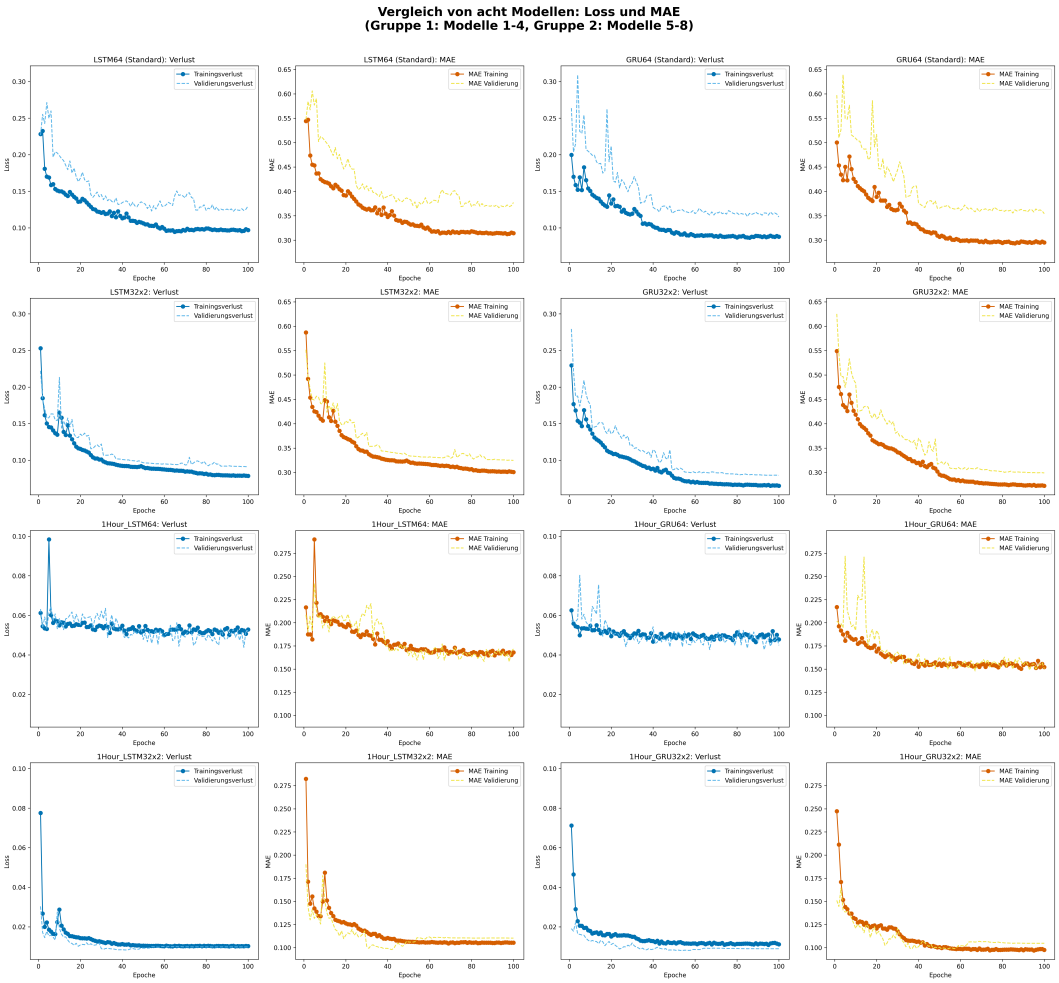


Abbildung 14: Modellvergleich

Literaturverzeichnis

Aggarwal, Charu C. (2023). *Neural networks and deep learning: a textbook*. Second Edition. Cham: Springer International Publishing AG. 529 S. ISBN: 978-3-031-29642-0.

Chollet, François (2018). *Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek*. 1. Ausgabe. mitp Professional. Frechen: mitp. 1 S. ISBN: 978-3-95845-838-3.

Géron, Aurélien (2023). *Praxiseinstieg Machine Learning mit Scikit-Learn, Keras und TensorFlow: Konzepte, Tools und Techniken für intelligente Systeme*. Übers. von Kristian Rother und Thomas Demmig. 3., aktualisierte und erweiterte Auflage. Heidelberg: O'Reilly. 1 S. ISBN: 978-3-96009-212-4.

Hastie, Trevor, Robert Tibshirani und Jerome Friedman (2009). *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY: Springer New York. ISBN: 978-0-387-84857-0. DOI: 10.1007/978-0-387-84858-7. URL: http://link.springer.com/10.1007/978-0-387-84858-7 (besucht am 06. 06. 2025).

McCulloch, Warren S. und Walter Pitts (Dez. 1943). „A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* 5.4, S. 115–133. ISSN: 0007-4985, 1522-9602. DOI: 10.1007/BF02478259. URL: http://link.springer.com/10.1007/BF02478259 (besucht am 21. 05. 2025).

Olah, Christopher (27. Aug. 2015). *Understanding LSTM Networks*. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (besucht am 06. 06. 2025).

Yu, Yong u. a. (Juli 2019). „A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures". In: *Neural Computation* 31.7, S. 1235–1270. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco_a_01199. URL: https://direct.mit.edu/neco/article/31/7/1235-1270/8500 (besucht am 29. 05. 2025).