Universität Regensburg

Deep Learning zur Schätzung absorbierter Strahlungsdosis für die nuklearmedizinische Diagnostik

An der Fakultät für Sprach-, Literatur- und Kulturwissenschaften der
Universität Regensburg zur Erlangung des akademischen Grades des
Master of Arts (M.A.) eingereichte Arbeit

von: Luciano Melodia
geboren in: Regensburg

Erstgutachter: Prof. Dr.-Ing. Bernd Ludwig
Zweitgutachter: Prof. Dr. rer. nat. Elmar Lang

Eingereicht am: 14. Juni 2020
Matrikelnummer: 1625974

A

ABSTRAKT

Im vergangenen Jahrzehnt gab es große Fortschritte im maschinellen Lernen. Diese Beiträge haben auch andere Wissenschaften und die Industrie in hohem Maß beeinflusst, sodass oft interdisziplinäre Gruppen an Verfahren der künstlichen Intelligenz geforscht haben.

Diese Verfahren werden zur Klassifikation, Mustererkennung oder Annäherung von Funktionen verwendet. Letzteres ist besonders in der nuklearmedizinischen Diagnostik für die Dosimetrie von Interesse. Krebspatienten sind durch die Therapie mit Radiopharmaka einer großen Strahlenbelastung ausgesetzt. Umso bedeutsamer sind Verfahren zur genauen Berechnung der Dosis. Doch gängige Verfahren benötigen lange Berechnungszeiten zur Simulation der durch das injizierte Isotop deponierten Energie.

In der nachfolgenden Arbeit wird ein neues Verfahren vorgestellt, welches mit Hilfe von Bildgebungen eine möglichst präzise Näherung für die absorbierte Strahlungsdosis im Körper schätzt. Für diesen Zweck kommen Bildgebungen aus dem *CT* und *PET* oder *SPECT* zum Einsatz. Aus der *CT*-Bildgebung wird die Massendichte für das Gewebe des Patienten gewonnen. Die Aufnahmen aus dem *CT* sind in kleine Volumeneinheiten zerschnitten, die sich Massenkern nennen. Daraus simuliert man anschließend die Verteilung der deponierten Energie mit Hilfe von Monte-Carlo-Verfahren. Es resultieren sogenannte Dosis-Voxel-Kerne welche mit der Zerfallsverteilung, die man aus dem *SPECT* gewinnt, verrechnet werden. Daraus wiederum resultiert die Verteilung der absorbierten Strahlungsdosis.

Ein neuronales Netz hat hierfür aus den Dichtekernen der Patientendaten die absorbierte Strahlungsdosis schätzen gelernt. Es wurde mit Massendichten und der zugehörigen absorbierten Strahlungsdosis trainiert. Das Netz wurde mit verschiedenen Metriken evaluiert und erreicht unter anderem einen Jaccard-Koeffizienten von 0.86. Abschließend wird diskutiert, wie das neuronale Netz für den klinischen Einsatz vorbereitet und evaluiert werden kann.

DANKSAGUNG

„Mein Abiturjahrgang war 2012.

Fünf Jahre hat es gedauert.

Vier Fächer durfte ich studieren.

Dreimal durfte ich verreisen.

Zwei Wege wurden mir eröffnet.

Für diesen einen habe ich mich entschieden.“

Ich möchte zu Beginn dieser Arbeit das Wort an diejenigen richten, die zu ihrem Entstehen beigetragen haben. Manch einer von ihnen bekam keinen literarischen Vermerk.

Als erstes möchte ich Elmar Lang danken. Professor Lang hat diese Arbeit betreut, zu allen Tageszeiten für Benachrichtigungen Zeit gefunden und mit einer schönen Weihnachtsfeier die brenzligste Zeit erhellt. Durch seine Vorlesungen habe ich besonders viel mitnehmen können. Wenn auch „*de boa Buildl*“ für das Selbststudium „*dahoam*“ meist noch übrig geblieben sind.

Weiterhin möchte ich meinen Dank an Bernd Ludwig richten. Professor Ludwig ist ebenfalls Betreuer dieser Arbeit und machte die Kooperation zwischen der Informationswissenschaft und der *Computational Intelligence and Machine Learning Group* (CIML) überhaupt erst möglich. Herrn Ludwig lernte ich bereits im Bachelor kennen. Bei ihm begann das maschinelle Lernen. Ich erinnere mich noch genau an die Seminarstunden, in denen *Food Recommender Systeme* besprochen wurden. Irgendwann ging es dann um das Mensa Essen. Ich vergesse das deshalb nicht so leicht, weil mich das experimentelle Weihnachtsmenü in der Mensa – „Spaghetti mit Bananensoße“ – nicht nur einmal erschauern ließ. Vielen Dank für Ihre Zeit!

Am Nachmittag nach der Mensa setzte ich mich dann oft noch an das Training der unterschiedlichen Modelle und habe versucht, dem Rechner die größten Mühen abzunehmen. Letztendlich reichte es mit der Hardware nicht ganz, da sprang Martin in die Bresche und stellte 4.2 GHz Intel Core i7 Prozessoren und Radeon Pro 580 8GB Grafikkarten zur Verfügung. „*Spuilzeich*“ würde er sagen. Danke an Martin Böddecker, dass ich auch an Wochenenden die Büros und die gesamte Hardware nutzen konnte und grundsätzlich für die sehr angenehme Zeit in deiner Firma!

Besonders wichtig für die Zeit, in der ich dann auch mal Pause machen musste, war das Essen. Gutes Essen, einen heißen Kaffee und Geschichten aus der Kindheit, wie sie den Traktor vom Urgroßvater stahl, konnte nur meine Mutter erzählen. Ich möchte Beata Melodia namentlich nennen, um nicht Gefahr zu laufen, vorgeworfen zu bekommen, ich hätte nur über „meine Mutter“ geschrieben. Mama ist doch viel

schöner. Danke Mama!

An der Uni gab es innerhalb der CIML Gruppe ebenfalls reichlich Hilfe. Besonders möchte ich mich bei Theresa Götz und Saad Al-Baddai bedanken, die mir die physikalischen Grundlagen näher gebracht haben und stundenlang mit mir über die Architekturen und Metriken in den verwendeten neuronalen Netzen diskutiert haben. Grundsätzlich bedanke ich mich bei der ganzen CIML. Mords Zeit!

Eine Person gab es, die auch die Launen eines misslungenen Versuchs ertragen musste. Sie peppelte mich wieder auf und brachte Schokolade. Dann hieß es, ich solle zeitig ins Bett gehen. Vielen Dank an die beste Freundin und Unterstützung, die man sich vorstellen kann: Marie-Louise Isenberg.

Dann wären noch die Freunde und Geschwister zu nennen. Sie waren für Abendunterhaltung, Korrekturen und ein Bier nach Feierabend zuständig. Vielen Dank an Dominique Melodia, Sebastian Müller, Thomas Büttner, Tobias Baron und Philipp Gäbelein.

Abschließend bedanke ich mich bei meinem Vater. Er hat mir Wissenschaft als Gegenstand näher gebracht und auch Wissenschaft als Philosophie. Nach dem Abitur war es eine schwere Entscheidung. Wohin? Was studieren? Mein Vater hat mir das erklärt. Es kommt nur darauf an was man verstehen möchte und wie sehr man es verstehen möchte. Welche Bezeichnung das Fach hat, ist unerheblich.

Danke dir ganz besonders Papa.

Dir widme ich diese Arbeit.

*In Andenken an
Dr. phil. Domenico Melodia*

Inhaltsverzeichnis

1	Problemstellung	1
1.1	Grundprinzip	2
1.2	Dichte spezifische Dosis-Voxel-Kerne	4
2	Deep Learning	7
2.1	Überwachtes Lernen mit neuronalen Netzen	8
2.2	Lernen mittels Gradientenabstieg	10
2.3	Vorwärtspropagation	10
2.4	Rückwärtspropagation	11
2.5	Rekurrente neuronale Netze (RNN)	13
2.5.1	Einfache rekurrente neuronale Netze (SRNN)	14
2.5.2	Lernen von Sequenzen mit rekurrenten neuronalen Netzen	15
2.5.3	Langzeit-, Kurzzeitspeicher mit rekurrenten neuronalen Netzen (LSTM)	17
2.6	Faltungsnetze (CNN)	19
2.6.1	Theorem der Faltung	19
2.6.2	Faltungsmatrizen	20
2.6.3	Übersetzungsinvarianz der Faltung	20
2.6.4	Pooling	22
2.7	Stochastischer Gradientenabstieg (SGD)	23
2.8	Adaptives Moment niedriger Ordnung (ADAM)	24
2.9	Nesterov adaptives Moment niedriger Ordnung (NADAM)	25
3	Experiment	29
3.1	Arbeiten zur Bildsegmentierung	29
3.2	Metriken zur Messung der Segmentierung und Rekonstruktion	31
3.3	Verschiebung der Kovarianz	32
3.4	Residuenetze	35
3.5	Dropout	36
3.6	Lernen komplexer Transferfunktionen mit U-Netzen	38
3.7	Ergebnisse	40
4	Diskussion	43
	Literatur	47

Abbildungsverzeichnis

1.1	Schematische Darstellung der Faltung von Zerfallskarte und Dosis-Voxel-Kernen	3
1.2	Schematische Darstellung des Prozesses von der CT-Bilgebung bis zur Dosisverteilung	4
2.1	Aktivierungsfunktionen: Tangens Hyperbolicus, Rectifier Linear Unit, Soft-Exponential-Funktion	8
2.2	Binärer Klassifikator: einschichtiges einzelnes Perzeptron	9
2.3	Vorwärtspropagation bei einem neuronalen Netz.	11
2.4	Rückwärtspropagation bei einem neuronalen Netz für Klassifikationsaufgaben	13
2.5	Einfaches rekurrentes neuronales Netz mit einer Eingangs-, einer Ausgangs- und einer tieferen Schicht. Die grünen Punkte entsprechen einfachen Assoziativspeichern in Form von Summen.	15
2.6	Rekurrentes neuronales Netz, wie es für das Lernen von Sequenzen verwendet wird	16
2.7	LSTM-Perzeptren in einem rekurrenten neuronalen Netz	17
2.8	Graphische Veranschaulichung der Faltungsoperation diskreter Mengen	20
2.9	Darstellung der gemeinsamen Gewichte wie sie in Faltungsnetzen umgesetzt werden	21
2.10	Weite und enge Formen der Faltung	22
3.1	<i>Skip Connection</i>	35
3.2	Architektur des U-Netzes zur Schätzung der Dosis-Voxel-Kerne	39
3.3	Evaluation der Trainingsdurchgänge mit <i>MSE</i> , <i>MAE</i> und <i>IoU</i>	41
4.1	Evaluation der klinisch motivierten Metrik	44
4.2	<i>Scree</i> -Plot der Massendichten.	54
4.3	Querschnitt der Massendichten und absorbierten Strahlungsdosis . . .	56
4.4	Schätzergebnisse des <i>Deep Learning</i> Systems	57

PROBLEMSTELLUNG

“During the development of the whole-body CT scanner, it became clear that the availability of an accurate cross-sectional picture of the body, the CT ‘slice,’ would have an important effect on the precision and implementation of radiotherapy treatment planning.

— Godfrey Hounsfield

NUKLEARE Verfahren werden in der Medizin sowohl in der Diagnostik als auch in der Krebstherapie immer bedeutsamer. Krebs ist eine in zunehmendem Alter sich häufende Erkrankung in der Bevölkerung und bedarf möglichst aufschlussreicher Diagnostik und wirksamer Behandlung. Durch den Ansatz von hybriden bildgebenden Verfahren wie *SPECT/CT* und *PET/CT* lassen sich Informationen über Gewebedichten und gleichzeitig über die Verteilung der prädiagnostisch verabreichten radioaktiven Isotope im Körper erhalten. Die räumliche Auflösung solcher Verfahren befindet sich im *mm* bis *cm* Bereich.

Stoffwechselaktive Gewebe wie beispielsweise Tumore sind gut erkennbar in den dreidimensionalen Verteilungen. Die Nuklearmedizin ist nicht nur nützlich um Tumore zu erkennen, sie kann auch dazu verwendet werden sie zu behandeln. Dazu wird dem Patienten ein radioaktives Präparat verabreicht, was sich anschließend an den Tumor anlagern soll und ihn somit bestrahlt. Bei einer Therapie mit einem Lutetium-markierten Radiopharmakon werden in den Zielregionen β^- -Strahlen emittiert. Die besonders energiereichen Teilchen mit einer sehr kurzen Reichweite müssen genau dosiert werden, sodass eine patientenspezifische Dosimetrie gefordert ist vgl. [FB11; AT13; JG15; MF13; IS13a]. Um die patientenspezifische Dosisverteilung abschätzen zu können werden *SPECT*-Bilder zu verschiedenen Zeitpunkten (4, 24, 48 und 72 Stunden nach Injektion) akquiriert. Um aus den so erhaltenen Aktivitätsverteilung eine Verteilung der Anzahl an Zerfällen pro Voxel zu bekommen, müssen die Bilder über die Zeit-Aktivitäts-Kurve integriert werden. Es gibt verschiedene Ansätze die statistisch unsicheren Bilder zu integrieren, das soll aber nicht in dieser Arbeit diskutiert werden. Im Folgenden wird die Zerfallsverteilung als bekannt vorausgesetzt. Zunächst wird das etablierte Grundprinzip der Dosisabschätzung erläutert und anschließend ein neuer Ansatz der Problemlösung vorgestellt.

1.1 GRUNDPRINZIP

Ausgehend von der Verteilung der Zerfälle wird die Dosisverteilung geschätzt. Ein weit verbreitetes Verfahren ist die Faltung mit einem sog. Dosis-Voxel-Kern. Dabei wird ein Würfel mit – in diesem Fall – 9^3 Voxel erzeugt.

Eine übliche Verfahrensweise Dosisverteilungen zu berechnen, ist die Verwendung von sog. S-Werten oder eng. *S-Values* vgl. [NPH07].

Unter den S-Werten versteht man den Bruchteil der mittleren kinetischen Energie für jeden radioaktiven Zerfall, welche von einer Volumeneinheit absorbiert wird. Die Energie, die in einer spezifischen Volumeneinheit gemessen werden kann, ergibt sich aus der linearen Superposition von jedem mitwirkenden Voxel innerhalb der räumlichen Verteilung um eine Strahlungsquelle. Die Energiedosis, welche um eine kreisförmig strahlende isotrope Quelle in einem homogenen Medium als Punkt gemessen werden kann, nennt sich Dosis-Punkt-Kern [GS08; LB08].

Bei der Berechnung von hochauflösenden Voxel-S-Werten werden Monte-Carlo-Simulationen verwendet. Die Quelle wird dabei als Voxel definiert, wobei der Winkel und der Ort der emittierten Strahlung stichprobenartig ermittelt werden. Die Teilchen werden auf ihren Emissionsursprung zurück berechnet (Material mit $p = 1.04 \frac{g}{cm^3}$) und die absorbierte Energie wird in jedem Voxel gespeichert, mit einer Auflösung von üblicherweise $5mm$. Die durchschnittliche absorbierte Dosis für jedes simulierte Teilchen wird in $MeV \cdot cm^{-3}$ angegeben und anschließend in $mGy \cdot MBq^{-1} \cdot s^{-1}$ umgerechnet, um als Voxel-S-Wert verwendet werden zu können.

Die erzeugten S-Werte können in einer geeigneten Größe für die Berechnung von absorbierter Strahlungsdosis verwendet werden vgl. [WB99]. Die Dosis für eine bestimmte Gewebeprobe $\mathbf{r}_T$ ist die Summe der Eigenaktivität und der Aktivität des umliegenden Gewebes $\mathbf{r}_U$ gemäß:

$$\langle D(\mathbf{r}_T) \rangle = \sum_{\mathbf{r}_U} \tilde{A}_{\mathbf{r}_U} \cdot S(\mathbf{r}_T \leftarrow \mathbf{r}_U) \quad (1.1)$$

Die Quelle und die berechnete Zieldosis sind Voxel aus dem Gewebe des Patienten, entsprechend der kummulierten Aktivitätenkarte aus dem *SPECT* Detektorfeld. Demenstprechend ist für ein bestimmtes Voxel $\mathbf{v}_T$ die durchschnittlich absorbierte Dosis gegeben durch die Summe der Beiträge aller Quellvoxel $\mathbf{v}_U^{(n)}$ mit der Anzahl N . Die Aktivität von $\mathbf{v}_U^{(n)}$ ist gegeben durch das Produkt der zeitlich eingebetteten Aktivität und der S-Werte von Voxel zu Voxel gemäß:

$$\langle D(\mathbf{v}_L) \rangle = \sum_{U=0}^{N-1} \tilde{A}_{\mathbf{v}_U} \cdot S(\mathbf{v}_T \leftarrow \mathbf{v}_U) \quad (1.2)$$

Die resultierende absorbierte Dosis kann durch Faltung erhalten werden. Eine Möglichkeit der Implementierung wäre der eng. *Fast Hartley Transform Algorithmus* [Bra84].

Für jedes verwendete Radionuklid wird eine S-Wert-Matrix berechnet, welche als Faltungskern verwendet wird um die mittlere absorbierte Dosis für jedes Voxel zu bestimmen. Dieses Verfahren birgt einen immensen Nachteil, denn es können keine Inhomogenitäten innerhalb des Gewebes berücksichtigt werden. Die Voxel haben unterschiedliche Gewebssklassen, z.B. eng. *soft tissue* für weiches Gewebe und somit eine zugehörige Massendichte von $1.004 \frac{g}{cm^3}$. In das Voxel im Zentrum des Würfels, welcher im weiteren Verlauf als Tensor bezeichnet wird, wird ein radioaktives Lutetium¹⁷⁷ Isotop platziert. Mit Hilfe einer Monte-Carlo-Simulationssoftware (GAMOS) vgl. [SC06; NR02; ORNL02] werden anschließend 1200×10^7 Zerfälle für die Dichtekerne des ganzen Körpers eines Patienten simuliert oder 20×10^7 für einen Dosis-Voxel-Kern. Als Resultat erhält man eine Dosisverteilung im Tensor der Einheit *Gray* in Bezug auf einen Zerfall. Es kann ebenfalls ein statistischer Fehler für jedes Voxel bestimmt werden. Dieser Dosis-Voxel-Kern wird mit der Zerfallsverteilung

Abb. 1.1: Schematische Darstellung der Faltung von Zerfallskarte und Dosis-Voxel-Kern. In der mittleren Darstellung ist die Verteilung der Zerfälle für einen Patienten dargestellt, die mit einem homogenen Dosis-Voxel-Kern gefaltet wird um zur Dosisverteilung, welche im Bild rechts zu sehen ist, zu gelangen.

gefaltet, um die Dosisverteilung zu erhalten vgl. Abb. 1.1. Formal schreibt sich die Faltungsoperation wie folgt vgl. Kap. 2.6.1:

$$T_{x,y} = \sum_n \sum_m \mathbf{K}_{nm} \mathbf{X}_{(x+n-\hat{z})(y+m-\hat{z})} \quad (1.3)$$

Die Einheit der Energiedosis ist $Gray = \frac{J}{kg} = 6,25 \cdot 10^6 \text{ TeV}$. Die Dosis kann wie folgt berechnet werden:

$$D = \frac{\partial \mathcal{L}}{\partial m} = \frac{1}{\rho} \frac{\partial \mathcal{L}(\mathbf{r}_T)}{\partial V} \quad (1.4)$$

Wobei $\mathcal{L}(\mathbf{r}_T)$ die deponierte Energie in einem Voxel, m die Masse des Voxels, ρ die Massendichte innerhalb des Voxels und V das Volumen des Voxels bezeichnet.

Aus der Gleichung (1.2) lässt sich schließen, dass die Dosisverteilung in einem Dosis-Voxel-Kern stark dichteabhängig ist. Bei der Faltung mit der Zerfallsverteilung wird vorausgesetzt, dass der Patient vollständig aus Weichgewebe besteht, was zu erheblichen Fehlern bei der Dosisverteilung führt. Im Anschluss wird dies für die zu diskutierende Methode berücksichtigt.

1.2 DICHTESPEZIFISCHE DOSIS-VOXEL-KERNE

Die Faltung mit einem Dosis-Voxel-Kern bestehend aus Weichteilgewebe ist eine zeiteffektive Methode zur Abschätzung der im Körper deponierten Strahlungsenergie, jedoch werden Dichteunterschiede von z.B. Knochen, Weichgewebe und Luft nicht berücksichtigt. Da von jedem Patienten ein *SPECT/CT* akquiriert wurde, ist die Information der Dichteverteilung vorhanden. Um diese zu nutzen, müsste man anstatt mit nur einem Kern die verschiedenen Zerfallsverteilungen auch mit verschiedenen Kernen falten.

In Abb. 1.2 wird die Vorgehensweise schematisch dargestellt. Zunächst wird das *CT*-Bild in kleine Volumeneinheiten zerlegt, um die Dichteverteilungskerne zu erhalten. Dabei ist das *CT*-Bild auf die Ortsauflösung der *SPECT*-Aufnahme skaliert. Anschließend wird die absorbierte Strahlungsdosis der einzelnen Dichteverteilungskerne mit Hilfe von Monte-Carlo-Simulationen berechnet. Sowohl die Zerfallsverteilung als auch die Dichteverteilung aus dem *CT* gehen in die Monte-Carlo-Simulation ein und heraus kommt die gewünschte Dosisverteilung [WB99]. Da die Berechnung und

Abb. 1.2: Aus der *CT*-Bildgebung lassen sich die Elektronenmassen messen, aus denen die Massendichte berechnet werden kann. Einzelne Ausschnitte ergeben die Dichteverteilungskerne. Mittels Monte-Carlo-Simulation können daraus Dosis-Voxel-Kerne berechnet werden. Um schließlich die Dosisverteilung zu erhalten, wird die Karte der Zerfallsverteilungen, die man aus der *SPECT*-Bildgebung erhält, mit den Dosis-Voxel-Kernen gefaltet.

Simulation mit den unterschiedlichen Dichteverteilungskernen immens kostspielig und zeitintensiv ist, jedoch die Faltung mit nur einem Kern zu unzureichenden Ergebnissen führt, soll ein neuer Ansatz vorgestellt werden. Ziel ist es, den Transfer, welcher bei der Monte-Carlo-Simulation berechnet wird, aus den Daten der unterschiedlichen Dichteverteilungen lernen zu können. Anschließend sollen die gelernten Filter für die Zerfallsverteilung zur Faltung eingesetzt werden, um die

Dosisverteilung zu erhalten.

Für dieses Verfahren werden Techniken aus dem *Deep-Learning* verwendet. Das nachfolgende Kapitel soll einen Überblick zu den gängigen Techniken geben und gleichzeitig zur im Experiment verwendeten Architektur hinführen. In den Erläuterungen zu den neuronalen Netzen soll für die Notation die fettgedruckte Schreibweise für Vektoren verwendet werden. Fettgedruckte Großbuchstaben entsprechen dabei Matrizen bzw. Tensoren.

“All problems in computer science can be solved by another level of indirection, except of course for the problem of too many indirections.

— David Wheeler

UNTER *Deep Learning* versteht man im Fachbereich der Informatik Berechnungsmodelle, die im gleichen Atemzug mit neuronalen Netzen genannt werden wollen. Dabei handelt es sich um mehrschichtige Methoden der Informationsverarbeitung, bei denen die Repräsentation der Daten mit unterschiedlichen Ebenen der Abstraktion gelernt werden soll. Es handelt sich um Techniken der Merkmalsextraktion oder eng. *Feature Extraction* [YL15]. Diese Methoden haben starken Einfluss auf die Verbesserung von Spracherkennung, Sprachsynthese, Grafikverarbeitung, Genomik, Objekterkennung, medizinische Bildgebung und vielen weiteren Fachgebieten ausgeübt.

Deep Learning kann dabei Strukturen in sehr großen Datensätzen erkennen, indem mittels *Backpropagation*-Algorithmus kalkuliert wird, wie eine Maschine ihre Parameter verändern muss, um eine gewünschte Repräsentation in einer Schicht aus der vorherigen Schicht berechnen zu können [YL15]. Für die Sprachverarbeitung führte das rekurrente neuronale Netz zunächst zu großen Durchbrüchen, da innerhalb einer Verarbeitungsschicht nicht nur afferente, sondern auch laterale Kopplungen mit der Rückwärtspropagation oder Anti-Hebb-Lernregel [Lis89] belernt werden und somit semantische Informationen zwischen morphosyntaktischen Einheiten eines Satzes kodiert werden können. Für die Grafikverarbeitung ist es das *Convolutional Neural Network* oder Faltungsnetz, welches nicht nur sequentielle Information, sondern auch räumliche Bezüge kodieren kann [YL15]. Maschinelles Lernen wird dabei in zwei grobe Kategorien eingeteilt: überwachtes und unüberwachtes Lernen. Man stellt schnell fest, dass überwachte Lernverfahren für Klassifikations- und Regressionsprobleme eingesetzt werden und unüberwachtes Lernen hervorragend geeignet ist, um Daten vorzuverarbeiten. Bei diesem Experiment wird das überwachte Lernen zur Regression verwendet, um die Transferfunktion von einer räumlichen Darstellung in eine andere zu lernen.

Für *Deep Learning* gibt es jedoch viele nennenswerte Technologien. Das nachfolgende Kapitel beschreibt die sehr vielfältig in dieser Arbeit verwendeten Algorithmen und Optimierungsverfahren. Dazu gehören: rekurrente neuronale Netze, die Funktionsweise von Faltung in neuronalen Netzen, besondere Eigenschaften der Faltung in Kombination mit dem *Pooling*, sowie zwei Optimierungsalgorithmen.

2.1 ÜBERWACHTES LERNEN MIT NEURONALEN NETZEN

Die vermutlich am besten erforschte und populärste Technik des maschinellen Lernens ist das überwachte Lernen. Wenn eine Klassifizierungs- oder Regressionsaufgabe vorliegt, z.B. eine Zuordnung von Bildern in verschiedene Gruppen, wird dabei zunächst ein Datensatz für jeden Vertreter gesammelt. Während des Trainings werden der Maschine die Bilder gezeigt. Diese berechnet einen Vektor von Zielwerten, dabei entspricht jede Dimension einer zugeordneten Kategorie oder Klasse [YL15]. Es wird entweder ein Vektor erzeugt, dessen Werte 0 oder 1 annehmen können, oder sich im Wertebereich zwischen 0 und 1 befinden. Ersteres gibt für jedes Bild eine feste Klassenzugehörigkeit an, letzteres eine Wahrscheinlichkeit. 1 entspräche beispielsweise der Zugehörigkeit zu einer Klasse und 0 eben nicht.

Ziel der Aufgabe ist es, den Informationseingang so zu verrechnen, dass dessen Wert an der entsprechenden Dimension des Vektors maximiert wird vgl. [YL15]. Jeder Wert in diesem Vektor steht dabei für eine Kategorie, zu der die Eingangsinformation zugeordnet werden soll. Vor dem Trainingsbeginn werden die Gewichte zufällig initialisiert, sodass es beliebig unwahrscheinlich ist, dass bei der Eingabe des Bildes die entsprechende Klasse resultiert. Deshalb wird eine Fehlerfunktion eingesetzt, die den Fehler oder den metrischen Abstand zwischen Eingabe und Ausgabe berechnet. Diese Funktion gilt es anschließend zu maximieren. Im Falle neuronaler Netze modifiziert die Maschine mittels Rückwärtspropagation in jedem Trainingsschritt ihre Parameter, sodass sich der Fehler verringert vgl. [YL15]. Die Parameter sind reelle Zahlen, die häufig auch Gewichte genannt werden. Modellhaft wird die Analogie zur Funktionsweise und Dynamik in menschlichen Hirnzellen gesucht, den Perzeptren vgl. [Ros62]. Deshalb wird eine einzelne verarbeitende Zelle auch als Perzeptron oder Neuron bezeichnet [Gro88].

Abb. 2.1: Aktivierungsfunktionen: Tangens Hyperbolicus, Rectifier-Linear-Unit [LM13], Soft-Exponential-Function [LG16] (Bildquelle: Wikipedia [Wik17]).

$$\mathbf{y} = h\left(\sum_n \mathbf{w}_{jn} \cdot \mathbf{x}_n\right) = h(\mathbf{W}\mathbf{x}) \quad (2.1)$$

In der obigen Gleichung (2.1) ist die Aktivität einer Schicht formalisiert. Dabei ist $\mathbf{y}$ das Ausgangssignal als explizite Funktion der Eingangssignale $\mathbf{x}_n$, $n \in \mathcal{D}$, wobei $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ die Anzahl der Beispiele bezeichnet und $\mathcal{D}$ die Menge der zu Grunde liegenden Daten. $\mathbf{w}_{jn}$ ist das Gewicht in der i -ten Schicht am n -ten Neuron,

entsprechend ist $\mathbf{W}$ die Gewichtsmatrix des Netzes. h ist eine Aktivierungsfunktion, bspw. Signumfunktion, Tangens Hyperbolicus oder Fermifunktion vgl. [Lip87]. Verwendet man die Signumfunktion für Klassifikationsaufgaben, folgt die Bedingung $|y| = 1$ und für jedes Muster μ muss gelten $t^{(\mu)} = \text{sign}(\mathbf{w} \cdot \mathbf{x}^{(\mu)})$, wobei $\mathbf{t}$ einen Vektor meint, welcher in Richtung der entsprechenden Klasse zeigt. Betrachtet man ein ideales Musterbeispiel $\mathbf{x}_*$, so gilt $[\mathbf{x}_*^{(\mu)} = \mathbf{t}^{(\mu)} \cdot \mathbf{x}^{(\mu)}] : \Leftrightarrow [\mathbf{w} \cdot \mathbf{x}_*^{(\mu)} \geq 0, \forall \mu]$. Geometrisch ist dies so zu interpretieren, dass mit $\mathbf{w} \cdot \mathbf{x}_* = \mathbf{w} \cdot \mathbf{x} = 0$ eine Gerade, Ebene oder Hyperebene im Raum definiert wird, welche die Muster im Eingangsraum in zwei Klassen einteilt. Ein einzelnes Perzeptron kann demzufolge nur linear separable Probleme lösen vgl. Abb. 2.2 [Lip87, vgl. S. 9-11]. Herkömmliche neuronale Netze arbeiten mit

Abb. 2.2: Binärer Klassifikator: einschichtiges einzelnes Perzeptron. (1) Vektor $\mathbf{w}$ ist orthogonal zur Trennebene zwischen zwei Klassen, (2) Forderung nach $\mathbf{w} \cdot \mathbf{x}_*^{(\mu)} \geq 0, \forall \mu$, (3) mit einem einzelnen einschichtigen Perzeptron nicht lösbares, nicht linear separables Problem.

einer asymmetrischen eng. *Feedforward*-Kopplung oder Vorwärtskopplung. Es gibt für gewöhnlich keine Rückkopplung zwischen Ausgangs- und Eingangsschicht und auch keine lateralen Kopplungen zu den Nachbarperzeptren. Der Algorithmus erhält Muster als Eingabe und strebt in Richtung geeigneter Gewichte. Für die Anpassung dieser Gewichte wird eine Lernschrittweite definiert, welche üblicherweise als η geschrieben wird. Daraus ergibt sich die Update-Regel [Lip87, S. 12]:

$$\mathbf{w}_{jn}^{(neu)} = \mathbf{w}_{jn}^{(alt)} + \eta \delta_j^{(\mu)} \cdot \mathbf{x}_n^{(\mu)} \quad (2.2)$$

Für die obige Gleichung gibt es immer dann eine genaue Lösung, wenn eine Richtung $\mathbf{w}$ gefunden werden kann, für die alle Projektionen der Muster $\mathbf{x} \in \mathcal{D}$ positiv sind $f(\mathbf{w}) \geq 0 \forall \mathbf{x}_*^{(\mu)}$. Für die Schwierigkeit eine Trennebene zu finden, lässt sich folgendes Maß angeben: $f(\mathbf{w}) = \frac{1}{|\mathbf{w}|} \min_{\mu} (\mathbf{x} \cdot \mathbf{x}_*^{(\mu)})$. Ein solches Perzeptron ist geeignet, um einfache binäre Klassifikationsaufgaben zu bewerkstelligen. Mehrlagige Netze mit mindestens zwei Schichten, den sog. eng. *Hidden Layer*, können auch nicht linear separable Probleme lösen. Dabei erfolgt die Verarbeitung in zwei Schritten.

2.2 LERNEN MITTELS GRADIENTENABSTIEG

Mittels Gradientenabstieg werden die Gewichte eines neuronalen Netzes iterativ angepasst, bis der Algorithmus konvergiert, ein Stop-Kriterium erfüllt ist oder die Kostenfunktion gänzlich verschwindet, was in der Praxis selten der Fall ist. Die Kostenfunktion wird häufig als Distanz oder Divergenz zwischen Eingabe und Ausgabevektor formalisiert. Einfache Distanzmaße bieten sich an, wie etwa die euklidische Norm, oder der mittlere quadratische Fehler. Im Folgenden soll eine Liste einen Überblick über gängige Kostenfunktionen schaffen, welche hauptsächlich für Klassifikationsprobleme verwendet werden [She05; LR03; HMS14].

$$D(\mathbf{x}, \mathbf{y}) = \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \|\mathbf{y} - \mathbf{x}\|_2^2 \quad (\text{mittlerer quadratischer Fehler})$$

$$D(\mathbf{x}, \mathbf{y}) = \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{p(\mathbf{y})} \quad (\text{Kullback-Leibler-Divergenz})$$

$$D(\mathbf{x}, \mathbf{y}) = \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \mathbf{x} \log \frac{\mathbf{x}}{\mathbf{y}} + (1 - \mathbf{x}) \log \frac{1 - \mathbf{x}}{1 - \mathbf{y}} \quad (\text{logarithmisch basierte Funktion})$$

Durch den Einsatz des Logarithmus (logarithmische Stauchung) besitzt die Funktion weniger lokale Minima als der *MSE*.

$$D(\mathbf{x}, \mathbf{y}) = - \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \mathbf{x} \log \mathbf{y} + (1 - \mathbf{x}) \log 1 - \mathbf{y} \quad (\text{Entropie-ähnliche Funktion})$$

Für die Entropie-ähnliche Kostenfunktion gilt, dass der Fehler der Ausgabeschicht direkt proportional zur Differenz von Zielausgabe und tatsächlicher Ausgabe ist:

$$\frac{\partial \mathcal{L}(\mathbf{x}, \mathbf{y})}{\partial \mathbf{y}} = \frac{\mathbf{y} - \mathbf{x}}{\mathbf{y}(1 - \mathbf{y})} \quad (2.3)$$

2.3 VORWÄRTSPROPAGATION

Unter Vorwärtspropagation versteht man die Präsentation eines Reizmusters in der Eingangsschicht und die sukzessive Berechnung der gewichteten Eingaben aller darauffolgenden Schichten, bis zur Ausgabe.

Der hier beschriebene Algorithmus ist in [YL15] einführend in die *Deep Learning* Thematik vorgestellt worden.

Sei $\mathbf{x}$ für die nachfolgenden Beispiele ein Vektor der Länge 4. Jede Dimension steht für ein Merkmal der Stichprobe. Die durchgezogenen Linien symbolisieren Gewichte von einem Perzeptron zum nächsten. Perzeptren addieren sämtliche gewichteten

Abb. 2.3: Vorwärtspropagation bei einem neuronalen Netz.

Muster aus der vorherigen Schicht auf, und transformieren diese mittels Aktivierungsfunktion. Dieser Vorgang wird bis zur Ausgabe fortgesetzt. Hier verändert sich die Aktivierungsfunktion. Für Klassifikationsaufgaben ist es sinnvoll zu entscheiden, ob ein Beispiel einer Klasse zugehörig ist oder nicht. Dies geht am besten mittels Wahrscheinlichkeiten. Die Softmax-Funktion $softmax(\mathbf{x}) = \frac{e^{\mathbf{x}}}{\sum_{\mathbf{x} \in \mathcal{D}} e^{\mathbf{x}}}$, die Werte im Intervall $[0, 1]$ ausgibt, kann dafür gewählt werden. Die Aktivität eines neuronalen Netzes mit zwei Schichten für Klassifikationsaufgaben im *Feedforward*-Schritt ist gegeben durch:

$$\mathcal{H}(\mathbf{x}, \mathbf{W}) = softmax \left(\sum_j \mathbf{W}_{ij} h \left(\sum_n \mathbf{w}_{jn} \mathbf{x}_n \right) \right) \quad (2.4)$$

2.4 RÜCKWÄRTSPROPAGATION

Die Rückwärtspropagation ist ein Algorithmus für die Korrektur der Gewichte bei jedem Iterationsschritt. Die Erläuterungen die nun folgen entstammen der 1987 veröffentlichten Publikation [Lip87] für Berechnungsverfahren mit neuronalen Netzen, sowie dem von Hinton et. al. 1986 publizierten Aufsatz über das Lernen der Repräsentation von Daten mittels rückwärts propagierter Fehlerrechnung [DR86]. Der eng. *Backpropagation*-Algorithmus ist äquivalent zu einem iterativen Gradientenabstieg für eine Kostenfunktion, hier in den Beispielen ein Distanzmaß (z.B. *MSE*). Die Kosten berechnen sich aus der tatsächlichen Ausgabe der kumulierten Aktivität eines neuronalen Netzes und der gewünschten Zielausgabe.

Die Nichtlinearität wird durch eine Transformation mittels Aktivierungsfunktion gewährleistet. Im Folgenden soll $h(x)$ als Aktivierungsfunktion gehandhabt werden. Der Faktor $\frac{1}{2}$ wird gerne vorangestellt, weil die Aktivierungsfunktion so analytisch leichter zu differenzieren ist, diese Veränderung ist jedoch nicht zwangsläufig notwendig.

Unterhalb der Schichten ist die entsprechende Kostenfunktion notiert. Für die Upda-

tes der Kostenfunktion wird die δ -Regel verwendet. Der gesamte Fehler zwischen tatsächlicher Ausgabe und dem Zielmuster ist durch $\mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{w})$ gegeben:

$$\mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{w}) = \frac{1}{2} \sum_{i, \mu} \left(\delta_i^{(\mu)} \right)^2 = \quad (2.5)$$

$$= \frac{1}{2} \sum_{i, \mu} \left(\mathbf{y}_i^{(\mu)} - \mathbf{x}_i^{(\mu)} \right)^2 = \quad (2.6)$$

$$= \frac{1}{2} \sum_{i, \mu} \left(\mathbf{y}_i^{(\mu)} - h \left(\sum_j \mathbf{W}_{ij} \mathbf{v}_j^{(\mu)} \right) \right)^2 = \quad (2.7)$$

$$= \frac{1}{2} \sum_{i, \mu} \left(\mathbf{y}_i^{(\mu)} - h \left(\sum_j \mathbf{W}_{ij} h \left(\sum_n \mathbf{w}_{jn} x_n^{(\mu)} \right) \right) \right)^2 \quad (2.8)$$

$\mathbf{v}_i^{(\mu)}$ meint die Aktivität am i -ten Perzeptron, während das Muster μ präsentiert wird. Es wird vorausgesetzt, dass die Kostenfunktion kontinuierlich und differenzierbar ist. Für die erste Schicht berechnet sich das $\Delta \mathbf{w}_{jn}$ wie folgt:

$$\Delta \mathbf{w}_{jn} = \eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}_{jn}} = \quad (2.9)$$

$$= \eta \sum_{i, \mu} \frac{\partial \mathcal{L}}{\partial \mathbf{v}_j^{(\mu)}} \frac{\partial \mathbf{v}_j^{(\mu)}}{\partial \mathbf{w}_{jn}} = \quad (2.10)$$

$$= \eta \sum_{i, \mu} (\mathbf{y}_i^{(\mu)} - \mathbf{x}_i^{(\mu)}) h'(\mathbf{h}_i^{(\mu)}) \mathbf{W}_{ij} h'(\mathbf{h}_j^{(\mu)}) \mathbf{x}_n^{(\mu)} \quad (2.11)$$

$$= \eta \sum_{i, \mu} \delta_i^{(\mu)} \mathbf{W}_{ij} h'(\mathbf{h}_j^{(\mu)}) \mathbf{x}_n^{(\mu)} \quad (2.12)$$

$$= \eta \sum_n \delta_j \mathbf{x}_n^{(\mu)} \quad (2.13)$$

In der Ausgabeschicht ergibt sich der Fehler der Vorwärtspropagation wie folgt:

$$\delta_i^{(\mu)} = (\mathbf{y}_i^{(\mu)} - \mathbf{x}_i^{(\mu)}) h' \left(\sum_j \mathbf{W}_{ij} \mathbf{v}_j^{(\mu)} \right) \quad (2.14)$$

Dieser Fehler ist der fortgepflanzte Fehler aus den *Hidden-Layer*-Schichten:

$$\delta_j^{(\mu)} = h' \left(\sum_i \mathbf{W}_{ij} \mathbf{v}_i^{(\mu)} \right) \sum_i \mathbf{W}_{ij} \delta_i^{(\mu)} = \quad (2.15)$$

$$= h' \left(\sum_j \mathbf{W}_{ij} \mathbf{v}_j^{(\mu)} \right) \sum_i \mathbf{W}_{ij} (\mathbf{y}_i^{(\mu)} - \mathbf{x}_i^{(\mu)}) h' \left(\sum_j \mathbf{W}_{ij} \mathbf{v}_j^{(\mu)} \right) \quad (2.16)$$

Eine Sammlung an Trainingsbeispielen wird Ensemble, eine vollständige Präsentation aller Beispiele Epoche genannt. Die Präsentation der Muster erfolgt zufällig. Die Anzahl der Muster, die für einen Iterationsschritt verwendet werden, nennt sich eng. *Batch-Size* und ließe sich mit Paketgröße sinngemäß übersetzen. Die Konvergenz des Algorithmus ist nicht bewiesen, ebenso wenig, wie ein wohldefiniertes Stop-

Kriterium. Im Allgemeinen sollte gelten, dass der Gradient $\|\nabla_{\mathbf{w}} \mathcal{L}\|$ einen kleinen Wert annimmt, oder aber der Gewichtsvektor keine signifikante Veränderung über eine bestimmte Anzahl an Iterationsschritten macht $\|\sum_t \mathbf{w}(t+1) - \mathbf{w}(t)\| < \epsilon$. Zusammenfassend ergibt sich folgender Algorithmus für ein m -schichtiges Netzwerk, wobei $m \in \{1, \dots, M\}$ und $\mathbf{v}_i^{(M)}$ die Aktivität des i -ten Perzeptrons in der m -ten Schicht bezeichnet. Dabei gilt, dass die Aktivität in der Eingangsschicht dem Netto-Input entspricht $\mathbf{v}_i^0 \equiv \mathbf{x}_i$ und mit $\mathbf{w}_{ij}^{(M)}$ das synaptische Gewicht der entsprechenden afferenten Kopplung ($\mathbf{v}_j^{(m-1)} \rightarrow \mathbf{v}_i^{(M)}$) bezeichnet wird: Zunächst werden die Gewichte mit kleinen Werten initialisiert, z.B. $10^{-2} \leq \mathbf{w}_{ij} \leq 10^{-4}$. Anschließend wird das erste Muster $\mathbf{x}_i^{(\mu)}$ präsentiert, sodass jede Eigenschaft des Musters in einem numerischen Wert kodiert ist ($\mathbf{v}_i^0 = \mathbf{x}_i^{(\mu)}, \forall i$). Jetzt berechnet man die Aktivitäten höherer Schichten mit $\mathbf{v}_i^{(M)} = h(\sum_j \mathbf{w}_{ij}^{(M)} \mathbf{v}_j^{(m-1)})$ bis zur letzten Schicht $\mathbf{v}_i^{(M)}$.

Abb. 2.4: Darstellung des *Feedforward*-Algorithmus (links) und Verlauf der Rückwärtspropagation des Fehlers (rechts).

An dieser Stelle ist die Vorwärtspropagation abgeschlossen und es greift der Fehlerberechnungsalgorithmus. Üblicherweise berechnet man den Fehler der Ausgangsschicht zuerst, sodass die Terme, welche mittels Kettenregel aus der Ableitung entstehen [vgl. 2.12], für die Berechnung in früheren Schichten wieder verwendet werden können. Der Fehler der Ausgangsschicht ist dann $\delta_i^{(M)} = h'(A_i^{(M)})(\mathbf{y}_i^{(\mu)} - \mathbf{v}_i^{(M)})$, wobei $A_i^{(M)}$ die Aktivität am i -ten Perzeptron in der m -ten Schicht meint. Somit ergeben sich auch die Fehler tieferer Schichten: $\delta_i^{(m-1)} = h'(A_i^{(m-1)}) \sum_j \mathbf{w}_{ij}^{(M)} \delta_j^{(M)}$. Zuletzt werden alle Gewichte mittels δ -Regel verbessert $\Delta \mathbf{w}_{ij}^{(M)} = \eta \delta_i^{(M)} \mathbf{v}_j^{(m-1)}$ und durch die neuen Gewichte ersetzt: $\mathbf{w}_{ij}^{(neu)} = \mathbf{w}_{ij}^{(alt)} + \Delta \mathbf{w}_{ij}$. Diese Prozedur wird für alle Muster im Trainingsensemble wiederholt.

2.5 REKURRENTE NEURONALE NETZE (RNN)

Rekurrente neuronale Netze sind eine Architekturform, die eine große Bandbreite von Berechnungsmodellen meint. Im Kern unterscheiden sich diese Modelle nur geringfügig in der Architektur von klassischen neuronalen Netzen. Dieser Unter-

schied manifestiert sich jedoch in ihrer mathematischen Definition. Im Gegensatz zu afferenten neuronalen Netzen, mit klassischer Vorwärtspropagation, besitzen rekurrente neuronale Netze zyklische Strukturen [ML09], die bei der Iteration berechnet werden.

Durch die Verwendung lateraler Kopplungen sind neuronale Netze auch Ausgangspunkt für die eng. *Skip-Connections* gewesen, welche maßgeblichen Einfluss auf die Bewegung des Gradienten haben. Diese Verbindungen werden im *Deep Learning* als Überbrückung verwendet und bieten dem Gradienten einen anderen Pfad an. Die *Skip-Connections* tauchten zuerst bei den sog. Residuennetzen auf und wurden später in die sog. U-Netze überführt [KH16; ÖÇ16]. Dazu in Kap. 3 mehr.

Ein rekurrentes neuronales Netz verhält sich so, dass es ein eigenes dynamische Aktivierungsverhalten innerhalb der rekurrenten Strukturen entwickelt. Versteht man afferente Netze als Transferfunktion $f : D \rightarrow Z : x \rightarrow f(x)$, so wäre ein rekurrentes neuronales Netz ein dynamisches System [ML09].

Weiterhin verfügt das rekurrente Netz über eine Art nicht lineare Historie zu den bereits trainierten Eingabemustern. Es kann als eine Art Kurzzeitgedächtnis verstanden werden [ML09]. Die Kopplungen können entweder in Form einer direkten Rückkopplung (Rekursion innerhalb eines Perzeptron), einer indirekten Rückkopplung (Rekursion zum vorherigen Perzeptron), einer lateralen Kopplung (zum benachbarten Perzeptron) oder eines vollständig verbundenen Graphen auftreten.

Derartige Modelle haben eine Kosten- oder Energiefunktion zu Grunde, die mit einem stochastischen Prozess minimiert werden soll. Die afferenten und rekurrenten Kopplungen sind dabei vollständig symmetrisch vgl. [ML09]. Die bekanntesten Vertreter dieser Netze sind Hopfield Netze [Hop82], Boltzmann Maschinen [DA85], oder die in jüngster Vergangenheit aufgekommenen eng. *Deep Belief Networks* [GH12]. Typische Anwendungen in diesem Gebiet sind Assoziativspeicher, Datenkompression, unüberwachte Modellierung von Verteilungen und Modellierung von zeitlich abhängigen Mustern, wie Bewegungsabläufen [ML09; GT07].

2.5.1 EINFACHE REKURRENTE NEURONALE NETZE (SRNN)

Einfache rekurrente neuronale Netze arbeiten mit dem in Kap. 2.4 erläuterten Rückwärts-propagation-Algorithmus. Elman hat 1991 die Funktionsweise dieser Algorithmen beschrieben [Elm91].

In der versteckten Perzeptrensicht im neuronalen Netz können laterale Kopplungen eingeführt werden, die zu sog. Kontexteinheiten führen. Diese Kontexteinheiten sind Perzeptren, deren Gewicht konstant bei 1 bleibt. Die Anzahl dieser Kontexteinheiten entspricht der Anzahl Einheiten in der versteckten Schicht, mit der die laterale Kopplung besteht. Die versteckten Einheiten koppeln anschließend an selbige Perzeptren zurück, von denen sie ihre Information beziehen. Jedes kontextuelle Perzeptron erhält von genau einer Perzeptreneinheit die gewichtete Information. Die Kontexteinheiten beinhalten eine exakte Kopie der Information der mit ihnen ge-

Abb. 2.5: Einfaches rekurrentes neuronales Netz mit einer Eingangs-, einer Ausgangs- und einer tieferen Schicht. Die grünen Punkte entsprechen einfachen Assoziativspeichern in Form von Summen.

koppelten Perzeptren zum Iterationszeitpunkt t . Zum Iterationsschritt $t + 1$ wird der Inhalt an die gekoppelte Schicht zurückgegeben. Jedes kontextuelle Perzeptron ist afferent mit jedem Perzeptron der verborgenen Schicht verbunden. Die Addition und Gewichtung erfolgt mittels Vorwärtspropagation vgl. Kap. 2.3. Diese Kontexteinheiten werden als Assoziativspeicher betrachtet, da in ihnen sämtliche Informationen, die vom neuronalen Netz verarbeitet wurden, beinhaltet sind. Die Information zum Iterationszeitpunkt t bedingt den gewichteten Ausgang der verborgenen Schicht, welcher wiederum für den nächsten Iterationsschritt $t + 1$ in die Kontexteinheiten gespeichert wird.

2.5.2 LERNEN VON SEQUENZEN MIT REKURRENTEN NEURONALEN NETZEN

Um Zeitreihen zu lernen werden modifizierte Architekturen verwendet. Im Folgenden wird ein Überblick über Techniken für rekurrente neuronale Netze gegeben, um zeitliche Abhängigkeiten zu kodieren. Als Zeitreihe versteht sich jegliche zeitliche Abhängigkeit, wie ein steigender oder fallender Aktienkurs, das Lernen bestimmter Bewegungsabläufe oder medizinische Anwendungen: Herzschlag, Blutdruck oder Hirnaktivitäten. Wie kann man zum Beispiel lernen, wann der Blutdruck steigt bzw. fällt, abhängig von der vorherigen Aktivität? Angenommen es liegen Daten von verschiedenen Patienten vor. Einer der Patienten ist stationär behandelt worden, dessen Werte liegen über eine geraume Zeit vor, anders als bei Patienten die eine 24 Stunden Messung absolvieren. Die erste Herausforderung ist also die Vereinheitlichung der Dimension, da eine unterschiedliche Menge an Daten von jeder Testperson vorliegt vgl. [Le15].

Eine Möglichkeit zur Vereinheitlichung der Dimensionen wäre die größte Datenmatrix als Ausgangsdimension zu verwenden. Anschließend werden alle anderen Matrizen auf dieselbe Größe formatiert, indem die fehlenden Belegungen mit Nullen aufgefüllt werden. Es ist von Vorteil, die Daten auf dieselbe Art und Weise mit Nullen zu füllen. Ausgehend von einer Ecke oder dem Zentrum der Matrix. Im Englischen nennt sich dieser Vorgang *Zero-Padding*. Daraufhin kann mittels Faltung der unterschiedlich dimensionale Eingang an Information in eine einheitliche Größe reduziert werden vgl. [Le15]. Um die Dimensionsreduktion zu gewährleisten und die

wichtigsten Aktivitäten zu berücksichtigen, ist es möglich nach der Faltung das eng. *Max-Pooling* zu verwenden (vgl. 2.6.4). Unter so drastischen Kriterien kann jedoch der Informationsverlust für die Bilderkennung zu hoch sein. Für das beschriebene Szenario ist diese Methode also ungeeignet, da man von einer präzisen zeitlichen Information ausgehend Rückschlüsse auf zukünftige Ereignisse schließen möchte vgl. [Le15].

Eine alternative Möglichkeit mit unterschiedlich großer Eingangsinformation umzugehen, sind rekurrente Netze. Ein geeignetes Modell für die Vorhersage solcher abhängiger Variablen könnte wie folgt aussehen vgl. Abb. 2.6 [Le15]: Dabei sind

Abb. 2.6: Rekurrentes neuronales Netz, wie es für das Lernen von Sequenzen verwendet wird [Le15]. Dabei kann der Eingang eine beliebige Anzahl an Dimensionen haben. Gibt es einen Datensatz, dem Messungen fehlen (Datenpunkte oder Vektorpositionen), werden diese Belegungen mit Nullen aufgefüllt. Die nicht lineare Aktivität innerhalb der Perzeptren mit lateralen Bindungen ist eine sigmoide Funktion. Als Ausgabe wird eine Kodierung mit Information über die sequenziellen Bezüge innerhalb der Daten generiert. Die Ausgabe muss nicht zwangsläufig dieselbe Dimension wie der Eingang haben. Es ist ebenso möglich die Daten niedriger- oder höherdimensional zu kodieren.

$x_0, \dots, x_T$ Messpunkte des Blutdrucks zum Zeitpunkt t der Gesamtzeit T . $h_0, \dots, h_T$ bezeichnen die Aktivitäten im neuronalen Netz, häufig auch lokales Feld genannt. Das Gewicht vom Eingang in die erste tiefere Schicht heißt w , das Gewicht zwischen den tieferen Schichten wird mit u bezeichnet und die Gewichte aus der tieferen Schicht heraus zur Ausgabe mit v , analog zu [Le15]. Diese Gewichte sind für alle Perzeptren gleich, sie werden also geteilt, sind Skalare und erhalten deshalb keinen Index. Durch das Teilen der Variablen kann der rekurrente Eingang beliebig lang gewählt werden. Nach der Vorwärtspropagation werden die Gewichte durch Gradientenabstieg auf der Kostenfunktion mit dem *Backpropagation*-Algorithmus optimiert.

Die Aktivitäten in den versteckten Schichten können anschließend sukzessive wie folgt angepasst werden [Le15], wobei h die Aktivität im lokalen Feld bezeichnet:

$$\mathbf{y} = v\mathbf{h}_T \quad (2.17)$$

$$\mathbf{h}_t = h(u\mathbf{h}_{t-1} + w\mathbf{x}_t), \text{ für } t = 1, \dots, T \quad (2.18)$$

$$\dots \quad (2.19)$$

$$\mathbf{h}_0 = h(w\mathbf{x}_o) \quad (2.20)$$

2.5.3 LANGZEIT-, KURZZEITSPEICHER MIT REKURRENTEN NEURONALEN NETZEN (LSTM)

Für die Langzeit-, Kurzzeitspeicher oder eng. *Long Short-Term Memory*, gibt es besondere Anwendungsfälle für das Lernen von Transfers. Leider ist die Rückwärtspropagation über die Zeit ein schwieriges Unterfangen, welches häufig zu explodierenden oder verschwindenden Gradienten führt [SH01]. Der Blutdruck bei einem

Abb. 2.7: LSTM-Perzeptren in einem rekurrenten neuronalen Netz. Der rote Pfad ist ein nicht linearer Pfad. Der Gradient kann auch über den linearen, einfacheren Pfad verbessert werden.

gesunden Menschen ist für gewöhnlich über Nacht durchgehend stabil. Morgens hat der Mensch im Allgemeinen leicht erhöhten Blutdruck, der über den Tag hinweg schwankt.

Entsprechen die Daten diesem Szenario, würde das System in der ersten Dimension, also am Morgen, immer einen hohen Wert für jeden Datenpunkt zum Training erhalten. Das entspräche einem niedrigen Gradienten, der sich über den Verlauf der Optimierung kaum noch ändern würde [Le15]. Dieses Verhalten ist nicht wünschenswert, da eine ähnliche Gewichtung für alle Tageszeiten erwartet wird.

Le formuliert dies wie folgt: Wenn der Gradient in einer bestimmten Dimension äußerst klein ist und in anderen Dimensionen extrem groß, dann sieht die Landschaft der Kostenfunktion aus wie ein Tal, mit sehr steilen Wänden und einem tiefen Becken vgl. [Le15]. Der Grund für dieses Verhalten des Gradienten, so Le weiter, ist die Nutzung von sigmoiden Aktivierungsfunktionen. Durch die iterative Verbesserung der Kostenfunktion werde diese auch mit der Ableitung der sigmoiden Funktion multipliziert. Dies könne schnell zur Sättigung führen.

Eine Möglichkeit dies zu verhindern wäre die *selu*-Funktion als Aktivierung zu ver-

wenden [SH17]. Diese verhindert die Bildung von Extremwerten beim Gradienten. In Kap. 3 wird im Detail vorgestellt, welche Funktion für den experimentellen Anwendungsfall gewählt wurde, um diesem Problem zu entgehen.

Abgesehen davon ist für das rekurrente Lernen wohl das LSTM die größte Verbesserung [FG03; SH97; Le15]. Die Idee hinter LSTM ist eine Veränderung der rekurrenten Struktur, sodass der Gradient bei der Rückwärtspropagation stabiler bleibt. Die Kerne dieser Assoziativspeicher sind die rekurrenten Perzeptren, die eine Integration über die Zeit darstellen vgl. [Le15]. Die anschließenden Erläuterungen folgen den Aufsätzen [SH97] und [Le15].

Angenommen die Daten haben zum Zeitpunkt t die Werte $\mathbf{x}_t$ und die tieferen Einheiten hatten zum vorangegangenen Zeitschritt die Aktivität $\mathbf{h}_{t-1}$, dann haben die Perzeptren, welche den Assoziativspeicher bilden, folgende Werte:

$$\mathbf{m}_t = \alpha \odot \mathbf{m}_{t-1} + \beta \odot h(\mathbf{x}_t, \mathbf{h}_{t-1}) \quad (2.21)$$

Dabei meint der Operator $\odot$ die elementweise Multiplikation zweier Vektoren. Das neue Gewicht der Assoziativspeicher ist somit eine gewichtete Linearkombination aus $\mathbf{m}_{t-1}$, also dem Gewicht aus dem vorangegangenen Zeitschritt und f . Dies entspricht zwei unterschiedlichen Pfaden, denen der Gradient bei der Rückwärtspropagation folgen kann, sollte eines der beiden Bestandteile extreme Werte annehmen. Häufig wird für die Funktion h eine sigmoide Funktion wie der $\tanh$ gewählt, um explodierende Gradienten zu verhindern. Von diesem Assoziativspeicher ausgehend, lässt sich der Zustand der versteckten Schichten wie folgt berechnen:

$$\mathbf{h}_t = \gamma \odot h(\mathbf{m}_t) \quad (2.22)$$

$\mathbf{h}_t$ und $\mathbf{m}_t$ werden für die nachfolgenden Zeitschritte zur Berechnung der Aktivität benötigt. Dies gibt dem Gradienten wiederum mehr Möglichkeiten unterschiedliche Pfade einzuschlagen. Die Terme α, β und γ werden im Englischen häufig auch *Gates* genannt, zu Deutsch entspricht das den Schranken. Diese Schranken sind Faktoren, die den Beitrag des jeweiligen Terms, also der Aktivität zum Schritt t bzw. $t - 1$ gewichten. Sie werden wie folgt implementiert:

$$\alpha(t) = h(\mathbf{W}_{x\alpha}\mathbf{x}_t + \mathbf{W}_{h\alpha}\mathbf{h}_{t-1} + \mathbf{W}_{m\alpha}\mathbf{m}_{t-1} + \mathbf{b}_\alpha) \quad (2.23)$$

$$\beta(t) = h(\mathbf{W}_{x\beta}\mathbf{x}_t + \mathbf{W}_{h\beta}\mathbf{h}_{t-1} + \mathbf{W}_{m\beta}\mathbf{m}_{t-1} + \mathbf{b}_\beta) \quad (2.24)$$

$$\gamma(t) = h(\mathbf{W}_{x\gamma}\mathbf{x}_t + \mathbf{W}_{h\gamma}\mathbf{h}_{t-1} + \mathbf{W}_{m\gamma}\mathbf{m}_{t-1} + \mathbf{b}_\gamma) \quad (2.25)$$

$$f(\mathbf{x}_t, \mathbf{h}_{t-1}) = h(\mathbf{W}_{xm}\mathbf{x}_t + \mathbf{W}_{hm}\mathbf{h}_{t-1} + \mathbf{b}_m) \quad (2.26)$$

Mit diesen Parametern und dieser Architektur können folgende Eigenschaften erzielt werden [Le15]:

- Es ist möglich die Verbindungen zu den rekurrenten Perzeptren bereits zu Beginn zu nutzen, sodass sich das LSTM-Netz ähnlich wie ein Faltungsnetz

verhält. Die Gradienten können zu den Werten jedes Zeitschrittes hin konvergieren.

- Es ist ebenfalls möglich die Verbindungen zu den rekurrenten Perzeptren von Beginn an zu initialisieren, sodass man im schlimmsten Fall ein einfaches rekurrentes Netz trainiert.
- LSTM-Einheiten sind numerisch sehr stabil. Durch die Aktivierungsfunktion sind die addierten Werte in einem normierten Bereich, z.B. zwischen -1 und 1 .

2.6 FALTUNGSNETZE (CNN)

Die eng. *Convolutional Neural Networks* oder Faltungsnetze sind eine Technik des *Deep Learning*, welche die Bildverarbeitung in den vergangenen Jahren maßgeblich geprägt und weiterentwickelt haben. Besonders wurde die Bildklassifikation [KS14], sowie Objekterkennung [JR15; SR15] und semantische Zuordnung von Bildinhalten [JL15a] verbessert.

Faltungsnetze zeigen beeindruckende Performanz bei standardisierten Benchmarktests mit Datensätzen wie z.B. ImageNet [JD09] oder MSCOCO [TYL14]. Die Fähigkeit der semantischen Repräsentation macht sie ebenso wertvoll für das Belernen einer Funktion für den Bild zu Bild Transfer. Im Folgenden werden die Prinzipien von Faltungsnetzen erläutert und gängige Modelle vorgestellt.

2.6.1 THEOREM DER FALTUNG

In der mathematischen Disziplin der Funktionalanalysis ist die Faltung ein Operator der zwei Funktionen $f(x)$ und $g(x) \in \mathbb{R}^n \rightarrow \mathbb{C}$ auf eine Dritte abbildet $(f \otimes g)(x)$. Faltung ist ein sehr nützliches Werkzeug, möchte man feststellen, wie viel von den Eigenschaften einer Funktion in einer anderen enthalten sind. Die Faltung ist wie folgt definiert vgl. [Wer07]:

$$(f \otimes g)(x) = \int_0^x f(\tau) \cdot g(x - \tau) d\tau \quad (2.27)$$

Der Wert des bestimmten Integrals ist eine veränderliche Funktion von x , somit eine Integralfunktion. Zur Erläuterung soll die Faltung der Funktionen $f(x) = \sin(x)$ und $g(x) = \cos(x)$ dienen. Dabei ist die Faltung so zu verstehen, dass die Ergebnisfunktion $(f \otimes g)(x)$ angibt, wie stark der Wert der Funktion $g(x)$ zum Iterationszeitpunkt $x - \tau$ im Funktionswert der Gewichtungsfunktion $f(x)$ zum Iterationsschritt x enthalten ist. Ein analytisches Beispiel für die Faltung von $\sin(x)$ und $\cos(x)$ ist im Anhang unter 4 gegeben.

Neuronale Netze verwenden ebenfalls das Prinzip der Faltung. Für diskrete Größen wird die Multiplikation der Faltungsmatrix, auch Kern genannt, im nächsten Abschnitt erläutert.

2.6.2 FALTUNGSMATRIZEN

Für die Faltung von Bildern verarbeitet man diskrete Mengen. Die Informationen über einen bestimmten Zustand werden in eine Faltungsmatrix geschrieben. Faltungsmatrizen sind quadratisch mit einer ungeraden Anzahl an Spalten- bzw. Zeilenvektoren. Für viele Operationen der Bildverarbeitung wird das Faltungsprinzip aus Kap. 2.6.1 verwendet. Für eine Menge an Pixel in einem Bild, oder einer beliebigen diskreten Menge an Datenpunkten, lässt sich die Faltungsoperation wie folgt formalisieren:

$$\mathbf{T}^{(x,y)} = \sum_n \sum_m \mathbf{K}_{nm} \mathbf{X}_{(x+n-\hat{z})(y+m-\hat{z})} \quad (2.28)$$

$\mathbf{T}$ ist die gefaltete Matrix, wobei die Koordinaten $x, y \in \mathbb{N}$ den jeweiligen Eintrag definieren. $\mathbf{K}$ bezeichnet den Kern, der zur Faltung verwendet wird. $\mathbf{X}$ bezeichnet die Matrix der Datenpunkte des ursprünglichen Datensatzes. Faltet man ein Bild, so entspricht $\mathbf{X}$ dem ursprünglichen Bild. Mit $\hat{z}$ wird die Matrixmitte des Kerns bezeichnet.

Abb. 2.8: Graphische Veranschaulichung der Faltungsoperation diskreter Mengen. Die Faltung wird mit einem Filter von 6 und einer Schrittweite (eng. *Stride* von 1 durchgeführt. Bei der Faltung wird zunächst von links nach rechts gefaltet. Die Werte innerhalb der rot markierten Matrix werden aufsummiert und in das hervorgehobene Feld in der rechten kleineren Darstellung der Matrix geschrieben. Die Matrix nach der Faltung ist kleiner, gemäß der Anzahl an Schritten entlang einer Achse.

2.6.3 ÜBERSETZUNGSINVARIANZ DER FALTUNG

Deep Learning wird häufig für große Datenstrukturen verwendet. Dabei ist entscheidend, dass der Datenhaushalt zu bewältigen ist. Bei großen Datenmengen sind die Dimensionen des Eingangs sehr hoch, dementsprechend würde jedes Perzeptron eine

Summe der gewichteten Dimensionen für jedes Beispiel annehmen. Angenommen es liegt ein $150px \cdot 150px$ Bild vor, so wäre der Eingangsraum 22500 Dimensionen groß und jedes Neuron würde ebensoviele Komponenten für die Aktivität aufaddieren. Eine mögliche Lösung ist das sog. Teilen der Gewichte oder eng. *weight sharing* vgl. [Le15].

Ein Beispiel soll das Prinzip verdeutlichen. Ausgehend von einer beliebigen Schicht und neun eingehenden Gewichten $w_1, \dots, w_9$, von denen jeweils drei den gleichen (oder einen ähnlichen) Wert haben:

$$w_1 = w_2 = w_3 \quad (2.29)$$

$$w_4 = w_5 = w_6 \quad (2.30)$$

$$w_7 = w_8 = w_9 \quad (2.31)$$

So sollen die Aktivitäten in die nächste Schicht propagiert werden vgl. [Le15]. Dadurch, dass die Gewichte dreier Verbindungen gleich sind, können sich diese den Wert teilen. So muss nicht der gesamte Datensatz verarbeitet werden. Statt alle Gewichte $w_1, \dots, w_9$ zu speichern, reicht es ein Gewicht aus jedem Tupel zu verarbeiten, z.B. w_1, w_4 und w_7 .

Diese Idee, Gewichte zu teilen, entspricht der Operation der Faltung, bei der eine Matrix, also ein Satz von Gewichten, für mehrere Positionen der Eingangssignale einer Schicht angewendet wird [Le15]. Diese Matrizen nennt man auch Filter. Filter haben eine weitere nützliche Eigenschaft für die Verarbeitung von Daten.

Abb. 2.9: Darstellung der gemeinsamen Gewichte wie sie in Faltungsnetzen umgesetzt werden (Bildquelle: [NK14]).

Innerhalb der *Pooling*-Schicht, dazu in Abschnitt 2.6.4 mehr, spielt die Reihenfolge der Informationen in den Perzeptren keine Rolle vgl. [Le15]. In natürlichen Daten ist der Übersetzungstransfer die häufigste Ursache für verrauschte Daten. Beim *Max-Pooling* wird von einer festgelegten Anzahl Perzeptren nur eines, welches die höchste Aktivität aufweist, in die nächste Schicht geleitet. Deshalb eignen sich solche Architekturen besonders für natürliche Daten. Bei diesem Vorgang von Faltung und *Pooling* wird die Dimension in den Daten drastisch reduziert. Viele kürzlich aufgekommene Architekturen haben zusätzlich eine weitere Art von Transformation, die lokale Kontrastnormalisierung, oder eng. *local contrast normalization* vgl. [Le15;

Abb. 2.10: Weite und enge Formen der Faltung. Der Filter in diesem Beispiel hat die Größe 5 (Bildquelle: [NK14]).

KJ09]. Diese Schicht erhält als Eingabe die Aktivität der *Max-Pooling*-Schicht. Es wird der Mittelwert subtrahiert und die Standardabweichung der eingehenden Perzeptren dividiert [Le15]. Das erlaubt eine Helligkeitsinvarianz, was für Bildsegmentierung und Objekterkennung ein nützliches Werkzeug ist. Diese Konstruktion wird für die Vorwärtspropagation analog zu herkömmlichen neuronalen Netzen trainiert. Bei der Rückwärtspropagation werden die Gewichte, die sich die gruppierten Perzeptren teilen, gemittelt vgl. [Le15]:

$$\mathbf{w}_1^{(neu)} = \mathbf{w}_1^{(alt)} - \alpha \left(\frac{\partial \mathcal{L}}{\partial \mathbf{w}_1^{(alt)}} + \frac{\partial \mathcal{L}}{\partial \mathbf{w}_4^{(alt)}} + \frac{\partial \mathcal{L}}{\partial \mathbf{w}_7^{(alt)}} \right) \quad (2.32)$$

$$\mathbf{w}_4^{(neu)} = \mathbf{w}_4^{(alt)} - \alpha \left(\frac{\partial \mathcal{L}}{\partial \mathbf{w}_1^{(alt)}} + \frac{\partial \mathcal{L}}{\partial \mathbf{w}_4^{(alt)}} + \frac{\partial \mathcal{L}}{\partial \mathbf{w}_7^{(alt)}} \right) \quad (2.33)$$

$$\mathbf{w}_7^{(neu)} = \mathbf{w}_7^{(alt)} - \alpha \left(\frac{\partial \mathcal{L}}{\partial \mathbf{w}_1^{(alt)}} + \frac{\partial \mathcal{L}}{\partial \mathbf{w}_4^{(alt)}} + \frac{\partial \mathcal{L}}{\partial \mathbf{w}_7^{(alt)}} \right) \quad (2.34)$$

2.6.4 POOLING

In der bisherigen Literatur sind die populärsten Ansätze das *Max*-, *Min*- und das *Average-Pooling*, sowie *Stochastic-Pooling*. Komplexere *Pooling*-Schichten, wie das räumlich pyramidale *Pooling* sind zuletzt in den Fokus der Untersuchungen gerückt [YB11; YBL10; MZ13; CP16]. Der Begriff *Pooling* kommt aus dem Englischen und entspricht übersetzt ins Deutsche einer Bündelung. Üblicherweise werden *Pooling*-Schichten innerhalb von Faltungsnetzen eingesetzt. Diese Schichten werden nach der Faltung verwendet, um besondere Merkmale entlang bestimmter Achsen oder Bereiche der verarbeiteten Matrix auszumachen. Ein Faltungsnetz berechnet Filter, die ein besonderes Merkmal am Eingangsobjekt kodieren und später bei der Vorhersage entdecken. Anders als mit der herkömmliche Aktivierungsfunktion bei der sämtliche Werte der Matrix separat verarbeitet werden, kann mittels *Pooling* die Aktivität über eine ganze Achse gelernt werden, oder einen Teil der Matrix. *Pooling*-Schichten leiten nur eine bestimmte Menge an Informationen weiter. Es kommt also zu einem Verlust. In welcher Form diese Informationen an die nächste Schicht weiter gegeben werden können, wird in diesem Kapitel beschrieben.

Das *Pooling* ist eine Technik, bei der eine Größe vordefiniert wird, die üblicherweise nicht veränderlich ist. Diese Größe kann einem Vektor, einer Matrix oder einem Tensor entsprechen, je Dimension mit der gebündelt wird. Normalerweise wird eine

$n \times n$ Matrix für ein *2D-Pooling* verwendet.

Beim *Max-Pooling* wird diese vordefinierte Größe wie eine Schablone am ersten Zeilen- und Spalteneintrag der Eingangsmatrix angelegt. Aus diesem Bereich wird die größte Aktivität gewählt. Diese Aktivität entspricht dem ersten Eintrag der Ausgangsmatrix. Dieses Verfahren wird für die gesamte Eingangsinformation iterativ wiederholt. Dabei muss eine zweite Größe festgelegt werden, die angibt, um wie viele Pixel die Schablone für jeden Iterationsschritt verrückt wird. Diese Größe nennt sich Schrittweite oder eng. *stride*. Die Schrittweite wird für jede Dimension der Eingangsinformation angegeben, für eine Matrix entspräche sie einem $2D$ -Vektor, für einen Tensor einem $3D$ -Vektor, entsprechend der Schrittweite in jede Raumrichtung.

2.7 STOCHASTISCHER GRADIENTENABSTIEG (SGD)

Der stochastische Gradientenabstieg stellt eines von vielen Optimierungsverfahren dar, die häufig in neuronalen Netzen verwendet werden. Bisher wurden verschiedene Techniken der Zusammensetzung von Perzeptren-Netzen behandelt. Die Optimierung der Parameter spielt jedoch ebenfalls eine zentrale Rolle. Als nächstes werden gängige Algorithmen vorgestellt, die auch im experimentellen Teil dieser Arbeit zum Einsatz kommen.

Im Gegensatz zum klassischen Gradientenabstieg ist der stochastische Gradientenabstieg eine starke Vereinfachung. Die Erklärungen und Beispiele für den SGD sind dem Aufsatz »*Large Scale Machine Learning with Stochastic Gradient Descent*« von Bottou entnommen [Bot10].

Anstatt den ganzen Gradienten $\nabla_{\mathbf{w}}(f_t)$ zu berechnen, wird für den SGD nur der Gradient zu einem Beispiel $\mathbf{x}$ während des Iterationsschritts t berechnet:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta_t \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}_t, \mathbf{w}_t) \quad (2.35)$$

Im Gegensatz dazu formalisiert man den Gradientenabstieg wie folgt:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta_t \frac{1}{n} \sum_{i=1}^n \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}_i, \mathbf{w}_t) \quad (2.36)$$

Im Gegensatz zum Gradientenabstieg in Gleichung (2.36) besteht die Hoffnung darin, dass sich Gleichung (2.35) durch die zufällige Auswahl eines Beispiels so verhält wie Gleichung (2.36), trotz des hinzugefügten Rauschens verursacht durch die Vereinfachung vgl. [Bot10].

Da der SGD keinen Bezug zu Beispielen aus vorhergehenden Iterationsschritten hat, kann er online berechnet werden. Der SGD ist eine direkte Minimierung des erwarteten Risikos, bzw. der erwarteten Kosten und wird aus der tatsächlichen Verteilung der zu Grunde liegenden Stichprobe gezogen.

In der Literatur wurde das Konvergenzverhalten des SGD weitreichend untersucht. Verständlicherweise ist die Konvergenz des SGD limitiert, durch die verrauschte

Schätzung des tatsächlichen Gradienten. Wenn der Gradient zu langsam abnimmt, nimmt die Varianz des Parameters $\mathbf{w}_t$ ebenso langsam ab. Unter ausreichender Regularisierung entspricht das bestmögliche Konvergenzverhalten $\eta_t \sim t^{-1}$. Die Erwartung des Residuenfehlers konvergiert mit einer ähnlichen Geschwindigkeit, $\mathcal{L}\rho \sim t^{-1}$ vgl. [Bot10; Mur98].

2.8 ADAPTIVES MOMENT NIEDRIGER ORDNUNG (ADAM)

Das zunehmende Interesse an maschinellem Lernen bedarf robuster Lösungen zur Optimierung von Kostenfunktionen von nicht stationären Daten.

Die Methode wurde so gestaltet, dass zwei sehr erfolgreiche Ansätze miteinander kombiniert wurden. Dazu gehört der AdaGrad Algorithmus [JD11] und der RMSProp Algorithmus [TT12]. AdaGrad zeigte bis zu diesem Zeitpunkt gute Performanz mit sparsam kodierten Daten, während RMSProp sehr gut mit online und mit stationären Daten umgehen konnte [DK15]. Einige Vorteile des ADAM Algorithmus: Die Anzahl

---

Algorithmus 1 ADAM Algorithmus aus dem Aufsatz von Kingma und Lei Ba [DK15], $\mathbf{g}_t^2$ ist die elementweise Quadrierung von $\mathbf{g}_t \odot \mathbf{g}_t$. In Ihrem Aufsatz geben die beiden Autoren als gute Initialparameter $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ und $\epsilon = 10^{-8}$ an. Im folgenden Algorithmus sind alle Vektoroperationen elementweise zu verstehen [DK15].

```

1: Prozedur: ADAPTIVES MOMENT NIEDRIGER ORDNUNG
2: Parameter:  $\alpha$ : Lernschrittweite
3: Parameter:  $\beta_1, \beta_2 \in [0, 1]$ : exponentielle Strafterme für die Momentschätzungen
4: Parameter:  $\mathcal{L}(\Theta)$ : Stochastische Kostenfunktion mit den Parametern  $\Theta$ 
5: Parameter:  $\Theta_0$ : Initialisierungsparameter
6:    $\mathbf{m}_0 \leftarrow 0$  (initialisiere Vektor für Moment erster Ordnung)
7:    $\mathbf{v}_0 \leftarrow 0$  (initialisiere Vektor für Moment zweiter Ordnung)
8:    $t \leftarrow 0$  (Iterationszeitschritt)
9:   while  $\Theta_t$  nicht konvergiert do
10:      $t \leftarrow t + 1$ 
11:      $\mathbf{g}_t \leftarrow \nabla_{\Theta} \mathcal{L}_t(\Theta_{t-1})$  (Berechne den Gradienten zum Zeitpunkt  $t$ )
12:      $\mathbf{m}_t \leftarrow \beta_1 \cdot \mathbf{m}_{t-1} + (1 - \beta_1) \cdot \mathbf{g}_t$  (Update Schätzung des ersten Moments)
13:      $\mathbf{v}_t \leftarrow \beta_2 \cdot \mathbf{v}_{t-1} + (1 - \beta_2) \cdot \mathbf{g}_t^2$  (Update Schätzung des zweiten Moments)
14:      $\widehat{\mathbf{m}}_t \leftarrow \mathbf{m}_t / (1 - \beta_1^t)$  (Fehlerkorrektur des Moments erster Ordnung)
15:      $\widehat{\mathbf{v}}_t \leftarrow \mathbf{v}_t / (1 - \beta_2^t)$  (Fehlerkorrektur des Moments zweiter Ordnung)
16:      $\Theta_t \leftarrow \Theta_{t-1} - \alpha \cdot \widehat{\mathbf{m}}_t / (\sqrt{\widehat{\mathbf{v}}_t} + \epsilon)$ 
  return  $\theta_t$ 

```

---

an Update-Schritten ist invariant in Bezug auf die Skalierung der Daten, die Veränderung des Gradienten ist nahezu gänzlich an die Schrittweite α gebunden, es werden keine stationären Daten benötigt, der Algorithmus funktioniert auch mit Matrizen sparsamer Kodierung und der Algorithmus vollführt eine schrittweise Annäherung, ähnlich dem *simulated annealing* vgl. [DK15].

Im beschriebenen Algorithmus 1 entspricht der Term $\mathcal{L}(\Theta)$ einer Energie- bzw. Kostenfunktion, die von einem Satz Parametern Θ abhängt [DK15]. Insgesamt soll der

Erwartungswert der Funktion $\mathcal{L}(\Theta)$ minimiert werden. Der Algorithmus wird als stochastisch bezeichnet, durch die Verwendung von zufällig gewählten Beispieldaten, anhand derer der Gradient berechnet wird. Eine weitere Möglichkeit den Algorithmus als stochastisch zu verstehen, wäre das inherente Rauschen der Funktion [DK15]. $\mathbf{g}_t = \nabla_{\Theta} \mathcal{L}_t(\Theta)$ bezeichnet den Gradienten der Funktion $\mathcal{L}(\Theta)$, die zum Iterationszeitpunkt t berechnet wurde. $\mathcal{L}_1 \dots \mathcal{L}_T$ bezeichnen analog die jeweilige Funktion zum Zeitschritt $t \in \{1, \dots, T\}$. Bei der Berechnung des Gradienten für einen neuen Zeitschritt wird dieser in Richtung der Momente erster und zweiter Ordnung gelenkt, dabei sind β_1 und β_2 Regularisierungsterme vgl. [DK15].

Kingma und Lei Ba testeten den Algorithmus auch unter experimentellen Bedingungen mit Faltungsnetzen. Nicht zuletzt soll der Algorithmus wegen der guten Benchmark Ergebnisse verwendet werden. In der Literatur gibt es viele Erweiterungen für ADAM. Von besonderem Interesse ist jedoch die Einbindung des Nesterov Moments in den Optimierungsalgorithmus. In [Doz16] zeigte diese Variante hervorragende Testergebnisse, weshalb der Algorithmus im nächsten Kapitel ebenfalls vorgestellt wird.

2.9 NESTEROV ADAPTIVES MOMENT NIEDRIGER ORDNUNG (NADAM)

Möchte man ein bestehendes Deep Learning System verbessern, so gibt es unterschiedliche Möglichkeiten wie man vorgehen kann vgl. [Doz16]: Entweder man macht das Netz tiefer, man ersetzt gewöhnliche rekurrente Einheiten durch LSTM-Zellen oder man setzt Methoden der Vorverarbeitung ein, um die Daten möglichst rauschfrei zu präsentieren [Doz16].

Weiterhin konnte gezeigt werden, dass auch eine sorgfältige Auswahl der Verteilung bei der Initialisierung von Faltungsnetzen ausschlaggebend für die frühe Entwicklung des Gradienten ist und somit auch für die Performanz des Systems [IS13b].

Dabei werden die Parameter des Systems durch Optimierungsalgorithmen in jedem Iterationsschritt angepasst. Der Gradientenabstieg ist der prominenteste Algorithmus und inzwischen sehr gut verstanden.

Für die Beschleunigung des Gradientenabstiegs wurde die Einführung eines Moments vorgeschlagen [Pol64]. Dabei wird, ähnlich dem physikalischen Verständnis von Moment, eine Art Geschwindigkeitsvektor, der eine entsprechende Raumrichtung angibt, verwendet. Dieser Vektor m wird mit einem Kostenterm (meist einer Konstante, hier und in den Ausführungen von Dozat und Polyak als μ bezeichnet) multipliziert.

Dies hat den Vorteil, dass der Gradient in Raumrichtungen, die eine rasante Veränderung und einen starken Abfall aufweisen, drastisch verlangsamt wird und vice versa in Richtungen beschleunigt wird, in denen der Gradient oszilliert [Doz16]. Sutskever et al. schlugen bereits die Einbindung des Nesterov-Moments in den Algorithmus des Gradientenabstiegs vor [Doz16; IS13c]. So zeigt sich, betrachtet man die erste

---

Algorithmus 2 Gradientenabstieg mit klassischem Moment [Doz16].

- 1: $\mathbf{g}_t \leftarrow \nabla_{\Theta_{t-1}} \mathcal{L}(\Theta_{t-1})$ (Berechnung des Gradienten $\mathbf{g}$ zum Zeitschritt t)
  - 2: $\mathbf{m}_t \leftarrow \mu \mathbf{m}_{t-1} + \mathbf{g}_t$ (Berechnung des Moments $\mathbf{m}$ mit Kosten μ)
  - 3: $\Theta_t \leftarrow \Theta_{t-1} - \eta \mathbf{m}_t$ (Moment mit Lernschrittweite η)
-

Gleichung der Momentberechnung, dass der Update-Schritt gleichbedeutend mit einem Gradientenabstieg in Richtung des Moments zum Zeitschritt $t - 1$ ist und ein weiterer Schritt in Richtung des Gradienten zum Zeitschritt t gemacht wird vgl. Algorithmus 1 [Nes83].

$$\Theta_t = \Theta_{t-1} - (\mu \mathbf{m}_{t-1} + \alpha_t \mathbf{g}_t) \quad (2.37)$$

Dozat postuliert, dass der Momentterm $\mu \mathbf{m}_{t-1}$ nicht vom derzeitigen Gradienten abhängt ($\mathbf{m}_t \leftarrow \mu \mathbf{m}_{t-1} + \alpha_t \mathbf{g}_t$), sondern lediglich vom Gradienten des letzten Iterationsschritts vgl. Algorithmus 1 [Doz16]. Sutskever et al. schlugen deshalb vor, dass die Parameter noch vor Berechnung des Gradienten angepasst werden sollten, um eine Qualitätsverbesserung gegenüber dem klassischen Gradientenabstieg zu erreichen vgl. [IS13c].

$$\mathbf{g}_t \leftarrow \nabla_{\Theta_{t-1}} \mathcal{L}_t(\Theta_{t-1} - \mu \mathbf{m}_{t-1}) \quad (2.38)$$

$$\mathbf{m}_t \leftarrow \mu \mathbf{m}_{t-1} + \alpha_t \mathbf{g}_t \quad (2.39)$$

$$\Theta_t \leftarrow \Theta_{t-1} - \mathbf{m}_t \quad (2.40)$$

Während für den klassischen Gradientenabstieg das Moment als eine gewichtete Summe der letzten Update-Schritte verstanden werden kann, entspricht es bei ADAM einem gewichteten Mittelwert der Gradienten der letzten Iterationen [DK15; Doz16].

$$\mathbf{m}_t \leftarrow \mu \mathbf{m}_{t-1} + (1 - \mu) \mathbf{g}_t \quad (2.41)$$

$$\Theta_t \leftarrow \Theta_{t-1} - \alpha_t \frac{\mathbf{m}_t}{(1 - \mu^t)} \quad (2.42)$$

Indem die Gradienten statt der Updates verwendet werden, ist es dem Algorithmus möglich kontinuierlich die Richtung zu wechseln, selbst wenn die Lernrate bereits sehr klein ist, was wiederum zu einer präziseren Konvergenz führt [DK15; Doz16]. Der Algorithmus korrigiert auch den Initialisierungsfehler, der durch die Initialisierung der Momente mit Nullwerten entsteht, durch den Nenner $(1 - \mu^t)$ [Doz16]. Im Folgenden möchte ich dem Aufsatz von Dozat [Doz16] folgen und erläutern, wie es gelingt, diese Modifikationen in den ADAM-Algorithmus zu integrieren. μ bezeichnet dabei die Kosten einer Richtung, in welche der Gradient während eines Update-Schritts gelenkt werden soll. Statt den Gradienten zuerst zu berechnen, anschließend zu den Ausgangswerten der Parameter zurückzukehren, um dann wiederum einen Schritt in Richtung des Moments zu machen, wird das Moment zum

Zeitschritt $t + 1$ nur ein Mal berechnet, während des Updates des Zeitschritts t . Dies geschieht wie folgt vgl. [Doz16; Nes83]:

$$\mathbf{g}_t \leftarrow \nabla_{\Theta_{t-1}} \mathcal{L}_t(\Theta_{t-1}) \quad (2.43)$$

$$\mathbf{m}_t \leftarrow \mu_t \mathbf{m}_{t-1} + \alpha_t \mathbf{g}_t \quad (2.44)$$

$$\Theta_t \leftarrow \Theta_{t-1} - (\mu_{t+1} \mathbf{m}_t + \alpha_t \mathbf{g}_t) \quad (2.45)$$

Dieselbe Umformung kann nun auch für ADAM (Alg. 1) angewendet werden:

$$\Theta_t \leftarrow \Theta_{t-1} - \alpha_t \left(\frac{\mu_t \mathbf{m}_{t-1}}{(1 - \prod_{i=1}^t \mu_i)} + \frac{(1 - \mu_t) \mathbf{g}_t}{(1 - \prod_{i=1}^t \mu_i)} \right) \quad (2.46)$$

$$\Theta_t \leftarrow \Theta_{t-1} - \alpha_t \left(\frac{\mu_{t+1} \mathbf{m}_t}{(1 - \prod_{i=1}^{t+1} \mu_i)} + \frac{(1 - \mu_t) \mathbf{g}_t}{(1 - \prod_{i=1}^t \mu_i)} \right) \quad (2.47)$$

Daraus entsteht dann das Nesterov adaptive Moment niedriger Ordnung. In Alg. 3 wird der vollständige Algorithmus noch einmal zusammengefasst.

Der Algorithmus wurde mit verschiedenen Optimierungsalgorithmen empirisch

---

Algorithmus 3 Schätzung des Nesterov adaptiven Moments [Doz16].

```

1: Prozedur: NESTEROV ADAPTIVES MOMENT
2: Parameter:  $\alpha_0, \dots, \alpha_T; \mu_0, \dots, \mu_T; \nu; \eta$  : Initialisierungsparameter
3:    $\mathbf{m}_0, \mathbf{n}_0 \leftarrow 0$  (Vektoren des ersten/zweiten Moments)
4:   while  $\Theta_t$  nicht konvergiert do
5:      $\mathbf{g}_t \leftarrow \nabla_{\Theta_{t-1}} \mathcal{L}_t(\Theta_{t-1})$ 
6:      $\mathbf{m}_t \leftarrow \mu_t \mathbf{m}_{t-1} + (1 - \mu_t) \mathbf{g}_t$ 
7:      $\mathbf{n}_t \leftarrow \nu \mathbf{n}_{t-1} + \mathbf{g}_t^2 (1 - \nu)$ 
8:      $\hat{\mathbf{m}} \leftarrow \mu_{t+1} \mathbf{m}_t / (1 - \prod_{i=1}^{t+1} \mu_i) + (1 - \mu_t) \mathbf{g}_t / (1 - \prod_{i=1}^t \mu_i)$ 
9:      $\hat{\mathbf{n}} \leftarrow \nu \mathbf{n}_t / (1 - \nu^t)$ 
10:     $\Theta_t \leftarrow \Theta_{t-1} - \alpha_t / (\hat{\mathbf{m}}_t \sqrt{\hat{\mathbf{n}}_t + \epsilon})$ 
  return  $\Theta_t$ 

```

---

verglichen, darunter ADAM und dem klassischen SGD. In Kombination mit Faltungsnetzen zeigte er die beste Performanz unter den getesteten Verfahren [Doz16].

Da dieser Algorithmus sowohl den SGD als auch den ADAM in der Geschwindigkeit der Konvergenz übertroffen hat, wird er im weiteren Verlauf der Arbeit zur Optimierung der Parameter der neuronalen Netze in Kap. 3 verwendet.

“If you thought that science was certain - well,
that is just an error on your part.

— Richard Feynman

FÜR das Experiment gilt es einen Transfer zu lernen. Ausgangspunkt sind vorher berechnete Matrixkerne der unterschiedlichen Gewebedichten von Gewebe eines Patienten. Mittels Monte-Carlo Simulation wurde die absorbierte Strahlungsdosis des Gewebes berechnet, wie in Kap. 1 beschrieben wurde. Die Einheit der absorbierten Strahlung ist *Gray*. In der Nuklearmedizin wird für weitere Berechnungen die Zerfallsverteilung aus der Bildgebung mit den entsprechenden Matrixkernen gefaltet, um eine neue Karte der Dosisverteilung zu erhalten. Der status quo ist eine Faltung nach Gewebssklassen. Das bedeutet, dass eine Faltungsmatrix für jede Gewebssklasse vorliegt. Ein zentrales Problem dieser Vorgehensweise ist gemischtes Gewebe, welches im Körper vorhanden ist, jedoch durch das Stückeln der CT-Aufnahme in Dichteverteilungskerne nicht adäquat repräsentiert werden kann. Die Dosisverteilung eines Gewebes mit unterschiedlicher Dichte wird so mit einem hohen Fehler geschätzt. Dieses Experiment zeigt, wie Verfahren des maschinellen Lernens eingesetzt werden können, um mit neuronalen Netzen eine bessere Schätzung der absorbierten Strahlungsdosis für ein bestimmtes Gewebe und ein bestimmtes Isotop erhalten zu können. Dabei werden Architekturen aus der Bilderkennung und Bildsegmentierung verwendet. Die bildgebenden Verfahren der Medizin machen große Fortschritte in diesem Bereich. Im Nachfolgenden soll Bezug auf aktuelle Publikationen der Bildsegmentierung und Bildrekonstruktion genommen werden.

3.1 ARBEITEN ZUR BILDSEGMENTIERUNG

Bei der Bildsegmentierung ist es die Aufgabe einen semantisch zusammengehörigen Teil eines Bildes zu erkennen und diesen abzugrenzen. Verschiedene wissenschaftliche Disziplinen nutzen Bildsegmentierung und Bilderkennung zur Analyse. Die Medizin insbesondere benutzt diese Techniken häufig als diagnostisches Werkzeug.

Besondere Fortschritte machte die Einführung der Faltungsnetze, die in den letzten Jahren für visuelle Aufgabenfelder den bisherigen Standard übertroffen haben [RG14; AK12]. Zu diesem Zeitpunkt wurden Faltungsnetze hauptsächlich für Klassifikationsaufgaben verwendet, jedoch gehört zur Bildsegmentierung mehr. Durch die Zuordnung von jedem Pixel zu einer bestimmten Klasse lässt sich aus der Klassifi-

kationsaufgabe eine Lokalisierungsaufgabe umformulieren vgl. [ÖÇ16]. Dadurch wurden Faltungsnetze für die Bildsegmentierung interessant.

Ab diesem Zeitpunkt gab es großen Andrang die Benchmarks zu übertreffen. Zuletzt sind die reinen Faltungsnetze (eng. *Fully Connected Convolutional Neural Networks*, kurz *FCNN*), die auf die klassische Struktur gänzlich verbundener Perzeptren verzichten, zum Standard erklärt worden vgl. [LC15; JD15; GL16].

FCNNs wurden weitreichend für verschiedene Aufgaben eingesetzt. So eigneten sie sich für die Tiefenschätzung [JL15b], Bildrestoration [DE13], der Rekonstruktion von Bildern für die Verbesserung der Auflösung [CD13] und nicht zuletzt der Schätzung von Dichten [FL15], weshalb sie auch in den Fokus dieser Untersuchung gerückt sind.

Diese Netze erhielten viel Aufmerksamkeit und konnten deshalb schnell weiterentwickelt werden. Durch die Kombination der LSTM-Zellen mit den Faltungsnetzen entstanden noch performantere Netze zur Extraktion von Merkmalen [GG17]. Eine besondere Architekturform der Faltungsnetze ist das U-Netz, welches von Ronneberger et. al. 2015 vorgestellt wurde [ÖÇ16; OR15]. Die Architektur besteht aus einem symmetrischen Aufbau. Wie bei *Autoencodern* auch, wird die Information über eine Dekodierungsschicht zunächst verarbeitet. Hierbei wird sukzessive die Dimension in den Daten durch Faltung und Pooling verringert. Dabei wird jedoch die Filteranzahl in jeder Faltungsschicht zeitgleich erhöht. Dies ist der kontraktive Pfad für die Lokalisierung der Muster vgl. [ÖÇ16]. Anschließend erfolgt eine Schicht zur Entkodierung des Gelernten. Die Anzahl der Schichten entspricht dabei der Anzahl der Dekodierungsschichten. Jede Schicht enthält dieselbe Anzahl an Filter wie bei der Dekodierung, jedoch in umgekehrter Reihenfolge. Durch *Upsampling* wird dabei die Dimension der Daten wieder auf den Ausgangszustand gebracht. Das Besondere an diesem Netz, was auch den entscheidenden Unterschied zu herkömmlichen *Autoencodern* macht, sind die Verkettungen zwischen den Schichten der Dekodierung und Entkodierung. Dabei wird jede Schicht der Dekodierung mit einer Entkodierungsschicht verknüpft, die dieselbe Dimension hat. Die Verknüpfung erfolgt über einen Pfad mit einer möglichst einfachen Aktivierung, beispielsweise eine lineare Aktivierungsfunktion, oder gar ohne Aktivierung. Es konnte gezeigt werden, dass diese Art von neuronalen Netzen in der Segmentierung die bisherigen Standards bei weitem übertreffen [ÖÇ16]. Weiterhin haben diese Netze eine besonders günstige Berechnungszeit, sie sind also schnell vgl. [ÖÇ16].

Aufgrund dessen wird diese Architekturform für den hier vorgestellten Rekonstruktionsvorgang adaptiert. Durch die Kombination verschiedener Techniken aus der Rekonstruktion und Segmentierung soll nicht nur eine erfolgreiche Lokalisation erfolgen, sondern eine möglichst voxelgenaue Transferfunktion gelernt werden.

3.2 METRIKEN ZUR MESSUNG DER SEGMENTIERUNG UND REKONSTRUKTION

Für das Experiment werden verschiedene Metriken verwendet, um das Ergebnis korrekt interpretieren zu können. Diese Untersuchung ist die erste ihrer Art, weshalb versucht wird den *State of the Art* in Anbetracht anderer Problemstellungen anzugeben. Bei den folgenden Metriken ist es gelegentlich wichtig die Achsen des Tensors separat zu erläutern. Zu diesem Zweck wird die erste Dimension mit i , die zweite mit j und die Dritte mit k benannt, sodass gilt: $\forall i, j, k = \{i, j, k \in \mathbb{N} | 1 \leq i, j, k \leq I, J, K\}$ und $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{I \times J \times K}$.

MSE (eng. *mean squared error*) Der mittlere quadratische Fehler ist definiert als der Erwartungswert des quadrierten Fehlers von Eingang und Zielausgabe, $MSE(\mathbf{X}, \mathbf{Y}) = \frac{1}{N} \sum_{i,j,k} (\mathbf{Y}_{ijk} - \mathbf{X}_{ijk})^2$. Diese Art Fehler zu messen gewichtet Fehler $|e| > 1$ im Quadrat höher und Fehler $|e| < 1$ im Quadrat niedriger. Dieses Maß ist ein gängiges Maß für Regressionsprobleme. Als Metrik für die Qualität des neuronalen Netzes ist diese Art Fehler am wenigsten geeignet, da der Fehler nicht auf ein Intervall zwischen $[0, 1]$ normiert ist. Aufgrund mangelnder Untersuchungen in diesem Bereich gibt es auch keine Referenzwerte auf die Bezug genommen werden kann. Der mittlere quadratische Fehler wird trotzdem gemessen, um Eigenschaften der Daten aus dessen Beziehung zu den anderen beiden verwendeten Metriken abzuleiten. Der *MSE* wird mit dem *MAE* verglichen. Für normalisierte Daten im Intervall $[0, 1]$ ist zu erwarten, dass der Fehler deutlich kleiner ausfällt als der *MAE*. Für Daten mit großer Varianz würde er entsprechend größer ausfallen.

MAE (eng. *mean absolute error*) Der mittlere absolute Fehler gibt ein gutes relatives Maß zum mittleren quadratischen Fehler und ist definiert als der Betrag der Differenz von Eingang und Zielausgabe: $MAE(\mathbf{X}, \mathbf{Y}) = \frac{1}{N} \sum_{i,j,k} |\mathbf{Y}_{ijk} - \mathbf{X}_{ijk}|$. Der *MAE* ist für auf $[0, 1]$ normierte Daten größer als der *MSE*. Auch diese Messung gibt jedoch nicht die Möglichkeit eines standardisierten Vergleichs her, denn der *MAE* ist nicht auf die Werte in den Daten normiert, sodass er als Metrik keine Aussagekraft hat. Die Messung soll trotzdem durchgeführt werden, da der *MAE* den Fehler linear gewichtet und somit ein Referenzwert zum *MSE* darstellt. Liegen bspw. Normalisierungsfehler vor, so können diese durch das Beobachten des Verhaltens der beiden Metriken entdeckt werden.

IoU (eng. *intersection over union*) In der Bildsegmentierung und dem Fachbereich *Computer Vision* ist der Jaccard-Koeffizient oder eng. *Intersection Over Union* sehr weit verbreitet. Im Allgemeinen wird damit der gelernte Überlapp zweier Mengen gemessen. Der Jaccard-Koeffizient ist definiert als: $J(\mathbf{X}, \mathbf{Y}) = \frac{|\mathbf{X} \cap \mathbf{Y}|}{|\mathbf{X} \cup \mathbf{Y}|}$.

Die folgende Implementierung der *IoU* wird für das neuronale Netz als Kostenfunktion verwendet:

$$J(\mathbf{X}, \mathbf{Y}) = \frac{\sum_{i,j,k} \min(\mathbf{X}_{ijk}, \mathbf{Y}_{ijk})}{\sum_{i,j,k} \max(\mathbf{X}_{ijk}, \mathbf{Y}_{ijk})} \quad (3.1)$$

Es wird jeweils zwischen Vorhersage und Zieltensor für jedes Pixel paarweise der kleinste und größte Wert genommen. Anschließend wird der kleinere Wert durch den größeren dividiert. Mit dieser Implementierung erhält man einen Fehler normiert auf das Intervall $[0, 1]$. Bei der Berechnung des Fehlers werden auch die räumlichen Positionen im Tensor berücksichtigt.

Weiterhin ist dieses Maß für Segmentierungsverfahren weit verbreitet und kann dadurch auf einfache Weise Aufschluss über die Qualität der produzierten Vorhersage geben [ZB16; TJ12].

Bei der Auswertung spielt als zweiter Faktor die physikalische Interpretierbarkeit eine zentrale Rolle. Etwa 60% der gesamten deponierten Energie liegt im Zentrum, wo das simulierte Isotop deponiert wurde. Aus diesem Grund ist es sinnvoll eine Kostenfunktion zu berechnen, welche den Fehler mit dem Anteil an deponierter Energie gewichtet. Indem aus beiden Tensoren paarweise das Minimum mit dem Kehrwert des Maximums gewichtet wird, kann dieser Effekt erzielt werden ($\frac{\min(x,y)}{\max(x,y)} = \min(x,y) \cdot \frac{1}{\max(x,y)}$). Die *IoU* bietet somit eine Metrik, welche den Fehler proportional zur deponierten Energie angibt.

Normalisierung Vor der Verarbeitung werden die Daten auf ein bestimmtes Intervall normiert. Da in der letzten Schicht des neuronalen Netzes eine sigmoide Funktion angewendet wird, welche ebenfalls Werte im Intervall $[0, 1]$ zurück gibt, eignet sich eine Min-Max-Normalisierung besonders. Um jedoch zu verhindern, dass der Gradient sättigt, werden die Werte auf ein etwas kleineres Intervall $[0.1, 0.9]$ normiert. Dadurch erfolgt die Aktivierung nicht nahe den Grenzwerten 0 und 1. Diese Bereiche sind Flach vgl. Abb. 2.1, was eine Sättigung des Gradienten bedeutet.

$$\mathbf{X} = (b - a) \times \frac{\mathbf{X}_{ijk} - \min(\mathbf{X})}{\max(\mathbf{X}) - \min(\mathbf{X})} + a \quad (3.2)$$

3.3 VERSCHIEBUNG DER KOVARIANZ

Faltungsnetze zu trainieren birgt viele Herausforderungen. Dadurch dass eine große Anzahl Parameter optimiert werden muss, gibt es auch einen großen Suchraum. Deshalb können die Filter während des Gradientenabstiegs sehr unterschiedliche Verteilungen annehmen, ebenso wie der Informationseingang von Schicht zu Schicht. Dies macht die richtige Initialisierung der Gewichte und Parameter äußerst schwierig und hat auch Konsequenzen für die gewählte Lernschrittweite vgl. [SI15].

Ioffe und Szegedy definieren die interne Verschiebung der Kovarianz oder eng. *Internal Covariate Shift* als die Veränderung der Verteilung der Aktivitäten des Netzwerks, während die Parameter des Netzes im Trainingsdurchlauf adaptiert werden [SI15].

LeCun, aber auch Wiesler und Ney, publizierten bereits früh ihre Erkenntnisse, dass neuronale Netze mit zentrierten Daten, linearen Transformationen mit Mittelwert 0 und Varianz 1, sowie dekorrelierten Merkmalen bessere Ergebnisse erzielen können [YL98; SW14].

In diesem Kapitel möchte ich auf die Normalisierung der *Batches* eingehen und erläutern, wie dadurch der *Covariate Shift* verhindert werden kann. Ich stütze mich dabei auf die Ausführungen von Ioffe und Szegedy [SI15].

Die *Batch*-Normalisierung kann für jede affine Transformation gefolgt von einer Nichtlinearität der folgenden Form verwendet werden:

$$\mathbf{y} = h(\mathbf{W}\mathbf{x} + \mathbf{b}) \quad (3.3)$$

Wobei $\mathbf{W}$ die gelernten Gewichte, $\mathbf{b}$ die gelernte Verzerrung und h eine beliebige Nichtlinearität meint. Die Verzerrung $\mathbf{b}$ kann vernachlässigt werden, da die Daten im nächsten Schritt normalisiert werden und durch die Subtraktion des Mittelwerts der Einfluss der Verzerrung eliminiert wird vgl. [SI15]. Anders als bei Ioffe und Szegedy wird die *Batch*-Normalisierung nach der Aktivierung durchgeführt. Dies hat den Hintergrund, dass die *LeakyReLU*-Nichtlinearität verwendet wird. *Batch*-Normalisierung zentriert die Daten aus einer Faltungsschicht entsprechend den gelernten Mustern. Negative Aktivierungen, welche durch die Verwendung eines Skalierungsterms in der Aktivierungsfunktion erlaubt werden, können nicht länger berücksichtigt werden. *Batch*-Normalisierung nach der Aktivierung normalisiert die positiven Daten ohne diese durch die negativen statistisch zu verzerren. Der Gedanke dabei ist, dass diese Merkmale in der nächsten Faltung ohnehin durch die Aktivierungsfunktion entsprechend skaliert werden, bspw. im Falle einer *ReLU*-Funktion auf 0 gesetzt und somit nicht länger berücksichtigt werden.

Daraus ergibt sich folgende Normalisierungsgleichung für die Aktivitäten:

Algorithmus 4 *Batch*-Normalisierung für die Faltung in neuronalen Netzen nach Ioffe und Szegedy [SI15].

- 1: **Prozedur:** *Batch*-NORMALISIERUNG
  - 2: **Parameter:** $\mathcal{B} = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}$: Werte für $\mathbf{x}$ über einen Satz an Beispielen
  - 3: **Parameter:** $\Theta = \{\gamma, \beta\}$: zu lernende Parameter
  - 4: $\bar{\mathbf{x}}_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i$ (Mittelwert für *Mini-Batch*)
  - 5: $\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (\mathbf{x}_i - \bar{\mathbf{x}}_{\mathcal{B}})^2$ (Varianz für *Mini-Batch*)
  - 6: $\hat{\mathbf{x}}_i \leftarrow \frac{\mathbf{x}_i - \bar{\mathbf{x}}_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$ (Normalisierung)
  - 7: $\mathbf{y}_i \leftarrow \gamma \hat{\mathbf{x}}_i + \beta \equiv \text{BatchNorm}(\mathbf{x}_i, \Theta)$ (Skalierung und Verschiebung)
  - 8: **return** $\mathbf{y}_i$
-

$$\mathbf{y} = \text{BatchNorm}(h(\mathbf{W}\mathbf{x})) \quad (3.4)$$

Für Faltungsschichten ist es ebenfalls wichtig zu berücksichtigen, dass Merkmale aus demselben Filter, mit einer anderen Position, dieselbe Normalisierung erhalten. Um dies zu gewährleisten wird eine *Mini-Batch* Normalisierung für alle Positionen durchgeführt [SI15]. Im Algorithmus 4 ist $\mathcal{B}$ eine Menge von Werten aus einer kleinen Menge an Beispielen und deren räumlicher Aktivität. Ist also die Größe der *Mini-Batch* mit m benannt, und die Merkmalskarte (oder Filter) mit $p \times q$, so wird die Größe des *Mini-Batch* auf $m' = |\mathcal{B}| = m \cdot pq$ festgelegt [SI15].

Würde man den Informationseingang einer Faltungsschicht auf diese Weise normalisieren, beeinflusst dies die Repräsentation durch die Transferfunktion. Verwendet man bspw. eine sigmoide Funktion, würde die Aktivität auf den linearen Teil der Nichtlinearität eingeschränkt sein [SI15]. Um zu gewährleisten, dass die Normalisierung auch die ursprüngliche Identität abbilden kann, sollte diese optimal sein, sollen zwei Parameter $\gamma^{(\mu)}, \beta^{(\mu)}$ für die Muster $\mathbf{x}^{(\mu)}$ gelernt werden vgl. [SI15; YL98]. Diese skalieren und verzerren die normalisierten Werte entsprechend des Gradienten:

$$\mathbf{y}^{(\mu)} = \gamma^{(\mu)} \hat{\mathbf{x}}^{(\mu)} + \beta^{(\mu)} \quad (3.5)$$

Diese Parameter werden mit dem ursprünglichen Modell iterativ gelernt. Während des Trainingsvorgangs muss die Rückwärtspropagation der Kostenfunktion $\mathcal{L}$ entsprechend der hier vorgestellten Transformation erfolgen. Ebenso wird der Gradient mit Berücksichtigung der *Batch*-Normalisierung verändert. Hierfür wird die Kettenregel wie folgt angepasst vgl. [SI15]:

$$\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{x}}_i} = \gamma \frac{\partial \mathcal{L}}{\partial \mathbf{y}_i} \quad (3.6)$$

$$\frac{\partial \mathcal{L}}{\partial \sigma_B^2} = \sum_{i=1}^m \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{x}}_i} (\mathbf{x}_i - \bar{\mathbf{x}}_B) - \frac{1}{2} (\sigma_B^2 + \epsilon)^{-\frac{3}{2}} \quad (3.7)$$

$$\frac{\partial \mathcal{L}}{\partial \bar{\mathbf{x}}_B} = \sum_{i=1}^m \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{x}}_i} \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \quad (3.8)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_i} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{x}}_i} \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial \mathcal{L}}{\partial \sigma_B^2} \frac{2(\mathbf{x}_i - \bar{\mathbf{x}}_B)}{m} + \frac{1}{m} \frac{\partial \mathcal{L}}{\partial \bar{\mathbf{x}}_B} \quad (3.9)$$

$$\frac{\partial \mathcal{L}}{\partial \gamma} = \sum_{i=1}^m \frac{\partial \mathcal{L}}{\partial \mathbf{y}_i} \hat{\mathbf{x}}_i \quad (3.10)$$

$$\frac{\partial \mathcal{L}}{\partial \beta} = \sum_{i=1}^m \frac{\partial \mathcal{L}}{\partial \mathbf{y}_i} \quad (3.11)$$

3.4 RESIDUENNETZE

Residuenetze verwenden die Idee von Einheiten mit deren Hilfe eine zusätzliche Residuenfunktion $\mathcal{H}$ in Abhängigkeit von dem Informationseingang $h(\mathbf{x}_n)$ in einer Schicht n gelernt werden soll [KH16].

Daraus lässt sich der Ausgang aus einer Residueneinheit wie folgt formalisieren:

$$\mathbf{y}_n = f(\mathbf{x}_n) + \mathcal{H}(\mathbf{x}_n, \mathbf{W}_n) \quad (3.12)$$

$$\mathbf{x}_{n+1} = h(\mathbf{y}_n) \quad (3.13)$$

Wobei $\mathbf{x}_n$ und $\mathbf{x}_{n+1}$ den Eingang in der n -ten Schicht meint und $\mathcal{H}$ eine Residuenfunktion. $f(\mathbf{x})$ ist eine identische Abbildung. Auf diese Weise kann der gesamte Inhalt eines überbrückten Teils des neuronalen Netzes durch die Summe des propagierten ungewichteten Inhalts und einer Residuenfunktion dargestellt werden. Die Realisierung dieser Technik erfolgt durch eine Verbindung, welche einen Teil des Netzes überbrückt und die Information von einem Teil des Netzes in einen tieferen propagiert vgl. [KH16]. Bei der Verwendung von U-Netzen wird der Inhalt von Schichten überbrückt, die bei der Verarbeitung der Daten als Ausgabe dieselbe Dimension liefern.

Für diese Modifikation gibt es in der Literatur viele Bezeichnungen, zuletzt wurden *Deep Learning* Systeme als Residuenetze bezeichnet, welche eine lineare Überbrückung zwischen jeder Schicht anbieten [KH16]. Dadurch kann theoretisch der Inhalt durch das gesamte Netz linear rückwärts propagiert werden. Die Verbindungen selbst werden *Skip Connections* genannt und sollen fortwährend sinngemäß mit Überbrückung bezeichnet werden. Jüngst wurden verschiedene Aktivierungsfunktio-

Abb. 3.1: Die Grafik zeigt zwei tiefere Schichten mit einer Überbrückung vgl. [KH16].

nen zur Propagation mittels Überbrückung getestet [SH97; RS15a; RS15b], wobei die besten Ergebnisse mit der Verkettung von Matrizen bzw. Tensoren oder einer linearen Aktivierung erzielt wurden. Andere Aktivierungsfunktionen zeigten größere Fehlerraten im empirischen Versuch.

Falls also h eine identische Abbildung ist, sodass $\mathbf{x}_{n+1} \equiv \mathbf{y}_n$, kann Gleichung (3.12) in (3.13) eingesetzt werden und es ergibt sich vgl. [KH16]:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \mathcal{H}(\mathbf{x}_n, \mathbf{W}_n) \quad (3.14)$$

Rekursiv fortgesetzt erhält man für die tieferen Schichten:

$$\mathbf{x}_{n+2} = \mathbf{x}_{n+1} + \mathcal{H}(\mathbf{x}_{n+1}, \mathbf{W}_{n+1}) \quad (3.15)$$

$$= x_n + \mathcal{H}(\mathbf{x}_n, \mathbf{W}_n) + \mathcal{H}(\mathbf{x}_{n+1}, \mathbf{W}_{n+1}) \quad (3.16)$$

Oder allgemein:

$$\mathbf{x}_N = x_n + \sum_n \mathcal{H}(\mathbf{x}_n, \mathbf{W}_n) \quad (3.17)$$

Dies gilt für die Schicht mit der Überbrückung n und die dazwischen befindlichen Schichten. Die Merkmale tieferer Aktivitäten $\mathbf{x}_N$ der Schicht N können dargestellt werden durch den Netto-Input der Schicht $\mathbf{x}_n$ und eine Residuenfunktion $\sum_n \mathcal{H}(\mathbf{x}_n, \mathbf{W}_n)$ zwischen zwei beliebigen Perzeptreineinheiten n und N .

Die Merkmale $\mathbf{x}_N = \mathbf{x}_0 + \sum_n \mathcal{H}(\mathbf{x}_n, \mathbf{W}_n)$ jeder tieferen Einheit $n \in \{n, \dots, N\}$ ist die Summe der Ausgangsinformation aller vorherigen Residuenfunktionen und $\mathbf{x}_0$. Dies steht im Kontrast zu herkömmlich „flachen“ Netzen, in denen ein Merkmal $\mathbf{x}_N$ das Produkt aus dem gewichteten Eingang $\prod_n \mathbf{W}_n \mathbf{x}_n$ ist (bei gleichzeitiger Vernachlässigung von Normalisierung und Regularisierung) [KH16].

Diese Verbindungen haben auch einen günstigen Einfluss auf den rückwärts propagierten Fehler, wie die folgende Modifikation der Kettenregel entsprechend Gleichung (3.17) zeigt vgl. [KH16]:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_n} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_N} \frac{\partial \mathbf{x}_N}{\partial \mathbf{x}_n} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_N} \left(1 + \frac{\partial \sum_n \mathcal{H}(\mathbf{x}_n, \mathbf{W}_n)}{\partial \mathbf{x}_n} \right) \quad (3.18)$$

Gleichung (3.18) führt an, dass der Gradient $\nabla_{\mathbf{x}_n} \mathcal{L}(\mathbf{x}_n, \mathbf{W}_n)$ durch zwei Terme dargestellt werden kann: $\frac{\partial \mathcal{L}}{\partial \mathbf{x}_N}$ welcher die Information direkt propagiert, ohne eine Gewichtung vorzunehmen, und einen zweiten Term $1 + \frac{\partial \sum_n \mathcal{H}(\mathbf{x}_n, \mathbf{W}_n)}{\partial \mathbf{x}_n}$ welcher durch die Gewichtung propagiert wird [KH16].

Aus Gleichung (3.17) und (3.18) folgt, dass das Signal direkt in die überbrückte Schicht weiter propagiert werden kann. In diesem Experiment wird die Technik nach der letzten Faltungsschicht und vor *Dropout* und *Pooling* eingebettet. Dadurch kann der Gradient auf die gelernte Information des ersten Teils auch nach weiteren Faltungsoperationen zurückgreifen.

In den U-Netzen zeigte sich diese Methode bereits als sehr effektiv [ÖÇ16; OR15] und auch im empirischen Versuch der Schätzung der Dosis-Voxel-Kerne konnte die bestehende Performanz des Systems nur mit Hilfe von Überbrückungen erreicht werden.

3.5 DROPOUT

Neuronale Netze mit vielen Schichten sind zunehmend schwerer zu trainieren. *Dropout* anzuwenden ist eine einfache Möglichkeit das Phänomen des sog. *Overfitting*

zu verhindern, bei dem die Trainingsdaten in gewisser Weise auswendig gelernt werden und die Generalisierungsfähigkeit verloren geht. *Dropout* dünnt dabei das Netzwerk aus, indem es durch eine Zufallsvariable aus einer Bernoulli-Verteilung bestimmt, welche Perzeptren innerhalb einer Schicht weiterhin aktiv bleiben und somit auch propagiert werden. Ein neuronales Netz mit n Perzeptreneinheiten kann als Sammlung von 2^n ausgedünnten Subnetzen verstanden werden. Diese Netze sind weiterhin so konzipiert, dass alle Gewichte aus der vorherigen Schicht geteilt werden, sodass die Anzahl Parameter $\leq O(n^2)$ ist vgl. [NS14].

Gegeben sei ein neuronales Netz mit N tieferen Schichten und sei $n \in \{1, \dots, N\}$ der Index der tieferen Schichten. Weiterhin sei I die Anzahl Perzeptreneinheiten in einer der tieferen Schichten und analog $i \in \{1, \dots, I\}$ der Index der Perzeptreneinheit innerhalb einer Schicht. Weiterhin bezeichne $\mathbf{y}_n$ den Informationsausgang $\mathbf{y}$ von Schicht n , wobei für $\mathbf{y}_0 = \mathbf{x}$ das ungewichtete Muster $\mathbf{x}$ der Input ist. $\mathbf{b}_n$ steht für *Bias* und bezeichnet die statistische Verzerrung in der n -ten Schicht. $\mathbf{w}_n$ ist der Gewichtsvektor $\mathbf{w}$ für die n -te Schicht. Weiterhin sei $h(\mathbf{x})$ eine nicht lineare Aktivierungsfunktion und $\odot$ die elementweise Multiplikation von Vektoren. So kann der *Feedforward*-Algorithmus wie folgt formalisiert werden nach [NS14]:

$$\mathbf{z}_n = \mathbf{w}_n \odot \mathbf{y}_{n-1} + \mathbf{b}_n \quad (3.19)$$

$$\mathbf{y}_n = h(\mathbf{z}_n) \quad (3.20)$$

Mit *Dropout* wird die Operation wie folgt verändert:

$$\mathbf{r}_n \sim \text{Bernoulli}(p) \quad (3.21)$$

$$\tilde{\mathbf{y}}_n = \mathbf{r}_n \odot \mathbf{y}_n \quad (3.22)$$

$$\mathbf{z}_n = \mathbf{w}_n \odot \tilde{\mathbf{y}}_n + \mathbf{b}_n \quad (3.23)$$

$$\mathbf{y}_n = h(\mathbf{z}_n) \quad (3.24)$$

Die Motivation hinter diesem Verfahren kommt aus der Evolutionstheorie. Bei der sexuellen Reproduktion wird die Hälfte der Gene zweier Elternteile mit einer geringen Wahrscheinlichkeit der Mutation miteinander rekombiniert vgl. [NS14]. Dies bildet das Erbgut des Nachwuchses.

Eine Alternative in der Biologie ist die asexuelle Replikation von einem Elternteil, bei dem eine Kopie des Erbguts erzeugt wird, die ebenfalls durch geringe Veränderungen mutiert. Es scheint plausibel zu sein, dass die asexuelle Fortpflanzung die bessere alternative darstellt, weil so gewährleistet werden kann, dass ein funktionierendes Gen ohne Umwege weitergegeben wird. Weiterhin würde die sexuelle Fortpflanzung durch die Rekombination funktionierende Gene womöglich aufbrechen vgl. [NS14]. Nichts desto trotz ist sexuelle Reproduktion die häufigste Form der Fortpflanzung bei komplexeren Organismen.

Eine mögliche Erklärung für die Überlegenheit der sexuellen Fortpflanzung ist der Vorteil der Mischungsfähigkeit von Genen. Betrachtet man eine evolvierende Popula-

tion über einen längeren Zeitraum, könnte die Fitness eines Individuums zweitrangig sein. Für die Überlebensfähigkeit der Population wäre es von Vorteil, dass ein Gen mit verschiedenen anderen Genen zusammenpasst. Die Fähigkeit eines Gens mit einem zufälligen anderen Gen funktionieren zu können macht es sehr robust vgl. [NS14]. Dieser Theorie zu Folge wäre es wichtig innerhalb der Population nicht nur ein vorteilhaftes Gen möglichst weit zu verbreiten, sondern dessen Robustheit zu gewährleisten.

Eine ähnliche Idee verfolgt dabei das *Dropout*. Jede Perzeptreineinheit soll lernen mit einer zufällig gewählten Menge an Perzeptreineinheiten aus der vorherigen Schicht die benötigten Merkmale zu generieren. Das macht die Perzeptreineinheit robuster und sorgt dafür, dass Fehler weniger durch andere Perzeptreineinheiten korrigiert werden können. So wird das richtige Lernen der Muster forciert vgl. [NS14].

3.6 LERNEN KOMPLEXER TRANSFERFUNKTIONEN MIT U-NETZEN

Das Modell des U-Netzes mit lateralen Kopplungen zwischen den Schichten implementiert eine Art Assoziativspeicher, welcher unmittelbar mit der extrahierten Information einer konvolutionellen Ebene im Netz zusammenhängt vgl. [ÖÇ16; OR15]. Als Ebene kann ein hierarchisches Plateau verstanden werden, in denen die Dimension der verarbeiteten Daten nicht drastisch verändert wird oder gleich bleibt, also kein *Downsampling* durchgeführt wird. Für Daten, die viel Information enthalten – verteilt auf eine geringe Anzahl an Dimensionen vgl. [4] – ist es wichtig die Architektur so zu modifizieren, dass ein möglichst verlustfreies Lernen erfolgen kann.

Der hier verwendete Datensatz ist zum Teil für dieses Experiment künstlich erzeugt worden und entsprechend neu. Dadurch gibt es keine vorliegenden Metaanalysen. Ein erster Schritt soll darin getan werden, ein Modell zu entwickeln, welches in der Lage ist Transfers zwischen den beiden Größen zu schätzen. Informationen zu den Daten, die im Rahmen dieser Arbeit entstanden sind, befinden sich im Anhang [4]. Die Implementierung des neuronalen Netzes erfolgt mit Keras in der Version 2.1.2 [al.15] und TensorFlow r1.5 [Aba+15]. Der Code für den Algorithmus wird auf Github publiziert unter <https://github.com/karhunenloeve/DeepLearningCNN> [Mel18].

Architektur Insgesamt hat das U-Netz 45 Schichten. Die Filter in den Faltungsschichten wurden mit der Lecun-Gleichverteilung initialisiert [YL64]. Jede Faltungsschicht ist verbunden mit einer *Leaky-ReLU*-Aktivierungsfunktion, sowie einer Schicht zur Normalisierung in Bezug auf jeden *Batch*. Die Konstante für die Multiplikation mit negativen Werten ist für die *Leaky-ReLU*-Funktion auf 5.5 festgelegt worden. Jede Faltungsschicht wurde regularisiert. Dabei erhielt die erste Schicht einen Regu-

larisierungsterm der L_1 -Norm, mit einem Faktor $\gamma = 0.005$ und jede weitere Schicht eine Regularisierung der L_2 -Norm mit einem Faktor $\gamma = 0.001$.

Abb. 3.2: Architektur des U-Netzes zur Schätzung der Dosis-Voxel-Kerne. Alle Faltungsschichten bis auf die letzte (blau) sind mit der *LeakyReLU*-Aktivierungsfunktion und einer Schicht zur *Batch*-Normalisierung versehen. Die laterale Kopplung aus der letzten Faltungsschicht des ersten Blocks (grauer Pfeil) ist eine einfache Verkettung der Matrizen ohne Aktivierung.

Aktivierungsfunktion Die sog. eng. *Rectifier Linear Units* sind im Kontext neuronaler Netze Funktionen, die den positiven Wert ihres Arguments liefern, dies entspricht $f(x) = x_+ = \max(0, x)$ für den Eingang in einem Perzeptron. Das Problem dieser Funktionen ist, dass Werte, die < 0 sind, nicht einbezogen werden. Die *ReLU*-Funktion setzt diese Werte grundsätzlich auf 0, sodass kein Einfluss über den Gradienten ausgeübt wird. Eine elegantere Lösung ergab sich aus empirischen Versuchen, sodass statt der *ReLU*-Funktion die sog. *LeakyReLU*-Funktion verwendet wird. Es konnte gezeigt werden, dass derartige Aktivierungsfunktionen die Performanz neuronaler Faltungsnetze maßgeblich verbessern konnten [LM13]. Diese Funktion erlaubt einen skalierten Gradienten, wenn der Eingang kleiner als Null ist:

$$f(x) = \begin{cases} x, & \text{wenn } x \geq 0 \\ \alpha x, & \text{sonst} \end{cases}, \text{ mit } \alpha = 5.5 \quad (3.25)$$

Daten und Gewebeklassen Der Datensatz setzt sich zu gleichen Teilen zusammen aus Massendichten bzw. absorbierter Strahlungsdosis von Lunge, Niere, Leber, Knochen und Milz.

Zu jeder Gewebeklasse gibt es 2.000 Proben von Massendichten und absorbierter Strahlungsdosis, also insgesamt 10.000 Beispiele. Da die unterschiedlichen Gewebearten sich stark in ihrer Struktur unterscheiden, wird die Grundgesamtheit zuerst gemischt. Anschließend werden die Datensätze daraus gezogen. Insgesamt gibt es

zwei Datensätze. Ein Trainingsdatensatz mit 7.000 Beispielen und ein Validierungsdatensatz mit 3.000 Beispielen, welcher völlig unabhängig ist. Die *Batch-Size* für den Trainingsvorgang ist auf 128 festgelegt.

3.7 ERGEBNISSE

Für das Experiment wurde als Optimierungsalgorithmus wie in Kap. 2.9 beschrieben der NADAM-Algorithmus verwendet. Insgesamt zeigt das neuronale Netz eine gute Generalisierungsfähigkeit mit einer *IoU* von 0.86 nach 308 Epochen. Gelernt wurde mit einer Lernschrittweite von 10^{-4} . Die Lernschrittweite wurde einmalig angepasst und halbiert, nachdem 15 Epochen ohne eine Veränderung von $\epsilon \geq 10^{-6}$ vergangen sind. Als Informationseingang wurde für das neuronale Netz die normalisierte Massendichte verwendet und als Zieltensor die ebenfalls normalisierten Dosis-Voxel-Kerne.

Tab. 3.1 zeigt die Ergebnisse für die unterschiedlichen Gewebeklassen. Die besten Resultate erzielte das Lungengewebe. In Abb. 4.3 werden verschiedene Massendichtekerne und Dosis-Voxel-Kerne für die unterschiedlichen Gewebeklassen aufgezeigt. Beide Datensätze wurden über den gesamten Tensor normalisiert, wie in Kap. 3.2 erläutert wurde.

Es ist gelungen ein Modell zu konstruieren, welches die Dosis-Voxel-Kerne mit einer an den derzeitigen Standard hinreichenden Genauigkeit schätzt vgl. [ZB16; TJ12]. Durch die Verwendung unabhängiger Daten kann zusammengefasst werden, dass die vorgestellte Architektur fähig ist komplexe Transferfunktionen zu lernen, wie sie in der Monte-Carlo-Simulation verwendet werden.

Aus der Vorbereitung der Daten für den Trainingsdurchlauf wurde Erkenntnis darüber gewonnen, wie die Varianz in den Daten verteilt ist. Zur Dimensionsreduktion wurde die Hauptachsentransformation getestet. Es zeigte sich jedoch, dass die Varianz in den Daten bis auf eine Dimension gleichverteilt ist. Dies ist nicht außergewöhnlich für künstlich erzeugte Daten. Aus diesem Grund wurde keine Dimensionsreduktion für das Training durchgeführt. Im Anhang befindet sich die Erläuterung zur *PCA* und der dazugehörige *Scree-Plot*, welcher den relativen Anteil der Varianz in jeder Dimension zeigt vgl. Kap. Appendix 4.

	IoU		$\text{MAE} \times 10^{-3}$		$\text{MSE} \times 10^{-4}$	
Gewebe	Training	Test	Training	Test	Training	Test
Knochen	0.55	0.47	3.91	4.23	1.18	1.21
Lunge	0.94	0.90	0.12	0.19	0.78	0.97
Niere	0.73	0.72	3.10	3.30	1.12	1.82
Leber	0.82	0.79	2.41	2.78	1.00	1.26
Milz	0.64	0.61	4.05	5.01	1.68	1.81
Total	0.96	0.86	2.29	2.12	1.18	1.24

Tab. 3.1: Ergebnisse für die Schätzung der Strahlungsdosis nach Gewebeklassen.

Abb. 3.3: Evaluation der Trainingsdurchgänge mit *MSE*, *MAE* und *IoU*. Das U-Netz wurde mit einem Trainings- zu Validierungsverhältnis von 7 : 3 trainiert. Bei der Initialisierung wurde eine Lernschrittweite von 10^{-4} gewählt. Nach 262 Epochen markiert eine schwarze vertikale Linie im Plot den Zeitpunkt, an dem die Lernschrittweite halbiert wurde (0.5×10^{-4}). Als Kostenfunktion wurde die *IoU* verwendet. Insgesamt wurde das Netz 308 Epochen trainiert. Ein Stop-Kriterium wurde festgelegt für die Kostenfunktion, sodass die Mindestverbesserung innerhalb von 15 Epochen bei $\epsilon \geq 10^{-6}$ liegen musste.

DISKUSSION

„ *The computing scientist's main challenge is not to get confused by the complexities of his own making.*

— Edsger Wybe Dijkstra

IN dieser Arbeit konnte empirisch gezeigt werden, dass *Deep Learning* Architekturen für das Lernen komplexer Transferfunktionen geeignet sind, insbesondere für die Anwendung in der Dosimetrie. Das entwickelte Verfahren ist fähig Gewichte für eine Linearkombination zu lernen, welche eine Funktion der Monte-Carlo-Simulation approximiert und dadurch auch neue, unabhängige Beispiele verarbeiten kann.

Durch die ausgewählten Fehlermessungen konnte überprüft werden, ob das Muster tatsächlich gelernt wurde. Zur Debatte steht jedoch die Größe des Fehlers und wie diese mit einer physikalisch plausiblen Argumentation gemessen werden kann. Der *IoU* gibt einen vollständigen Einblick in die Fehlerverteilung für einzelne Beispiele. Dabei werden jedoch die Positionen innerhalb eines DVKs gleich gewichtet, sodass ein Fehler im äußeren Bereich des Kerns genauso ausschlaggebend für den Gesamtfehler ist, wie ein Fehler im zentralen Bereich.

Um zu überprüfen, ob der Algorithmus für den klinischen Einsatz geeignet ist, müssen in weiteren Untersuchungen vollständige Simulationen von mehreren Menschen generiert und für den Trainingsprozess verwendet werden. Anschließend kann man das neuronale Netz mit der Monte-Carlo-Simulation und den derzeitigen Standards für die Berechnung von Dosisverteilungen vergleichen um ein Fazit zu ziehen.

Für den Patienten spielt dabei die exakte Berechnung der Dosisverteilung im Zentrum eines DVKs die größte Rolle. An der Stelle, wo die isotrope Quelle deponiert wurde, ist auch die Strahlungsenergie am höchsten. Aus diesem Grund fällt in diesem Bereich der Fehler besonders schwerwiegend aus.

Idealerweise erfolgt die Gewichtung so, dass das Zentrum besonders starken Einfluss auf den Fehler hat und nach außen hin zunehmend geringer gewichtet wird. Ein Vorschlag für diese Umsetzung wäre die Gewichtung des vorhergesagten Kerns der absorbierten Strahlungsdosis mit der berechneten Energiedosis aus der Monte-Carlo-Simulation. Formalisiert ergibt dies folgende Kostenfunktion, wobei $\odot$ wiederum die elementweise Multiplikation der Tensoren notiert:

$$\mathcal{L}(\mathbf{X}, \mathbf{Y}) = \sum_{i,j,k} (\mathbf{X}_{ijk} - \mathbf{Y}_{ijk})^2 \odot \frac{\mathbf{X}_{ijk}}{\sum_{i,j,k} \mathbf{X}_{ijk}} \quad (4.1)$$

Dabei bezeichnet $\mathbf{X}$ den Tensor mit der tatsächlichen Dosisverteilung (berechnet mit einer Monte-Carlo-Simulation) und $\mathbf{Y}$ die Vorhersage des neuronalen Netzes. Gleichung (4.1) ist der mittlere quadratische Fehler, gewichtet mit dem relativen Anteil an Strahlungsdosis zum gesamten DVK. Aus der physikalischen Simulation ergibt sich, dass im Zentrum die Strahlungsdosis immer am höchsten ist und nach außen hin sukzessive abnimmt. Neben dieser Messung soll die entgegengesetzt gewichtete Metrik als Referenzwert dienen:

$$\mathcal{L}'(\mathbf{X}, \mathbf{Y}) = \sum_{i,j,k} (\mathbf{X}_{ijk} - \mathbf{Y}_{ijk})^2 \odot \left(0.9 - \frac{\mathbf{X}_{ijk}}{\sum_{i,j,k} \mathbf{X}_{ijk}} \right) \quad (4.2)$$

Abb. 4.1: Evaluation der klinisch motivierten Metrik. Die dick hervorgehobenen Linien der Metrik, welche mit der umgekehrten Verteilung gewichtet wurde, korrelieren stark mit der normal gewichteten Metrik, sind jedoch stets höher. Für die Messung wurde der alte Trainingsstand aus Kap. 3 entnommen und bis zur Epoche 453 fortgeführt.

Die Berechnung der Metrik $\mathcal{L}'$ ist eine Art Negativ der Strahlungsdosis. Da die Werte normiert sind, erhält man durch das Subtrahieren des relativen Anteils der deponierten Energie einen Tensor, der entgegengesetzt zur ersten Metrik gewichtet ist. Der äußere Bereich des Tensors wird dabei stärker gewichtet als der innere. Betrachtet man nun den Abstand der beiden gemessenen Größen zueinander, so kann man Aussagen, dass falls $\mathcal{L}' < \mathcal{L}$, der Fehler im Zentrum geringer ausfällt als

der Fehler zum Rand hin. Dies trifft aufgrund der Verteilung der Strahlungsdosis in den DVKs aus der Simulation zu. In Abb. 4.1 wurden diese Metriken für etwas mehr als 150 Epochen berechnet. Daraus ergibt sich, dass stets $\mathcal{L}' < \mathcal{L}$ gilt und somit der Fehler zum Zentrum im Mittel geringer ausfällt.¹

Diese Feststellung gibt ein Indiz zur Beschaffenheit der Daten und legitimiert ihre Verwendungsweise, enthält jedoch keine Metrik, die ein geeignetes Vergleichsobjekt darstellt. In weiteren Untersuchungen sollen nun Vergleiche mit Verfahren gemacht werden, die derzeit in Verwendung sind. Ergeben sich daraus gute Resultate, können anschließend Experten aus dem Gebiet der Dosimetrie entscheiden, zu welchem Zeitpunkt die Strahlungsdosis mittels neuronaler Netze geschätzt werden kann.

---

¹Für eine Normierung im Intervall $[0, 1]$ lautet die Formel entsprechend $\mathcal{L}'(\mathbf{X}, \mathbf{Y}) = \sum_{i,j,k} (\mathbf{X}_{ijk} - \mathbf{Y}_{ijk})^2 \odot \left(1 - \mathbf{X}_{ijk} : \sum_{i,j,k} \mathbf{X}_{ijk}\right)$. In dem in Kap. 3 beschriebenen Algorithmus wurde aus technischen Gründen auf ein Intervall von $[0.1, 0.9]$ normiert. Dem entsprechend ist die Gleichung (4.2) modifiziert worden.

LITERATUR

- [AK12] G.E. Hinton A. Krizhevsky I. Sutskever. „Imagenet classification with deep convolutional neural networks“. In: *NIPS*. S. 1106–1114 (2012) (zitiert auf Seite 29).
- [AT13] C. Avigo M. Sollini P. Erba G. Mariani A. Traino S. Marcatili. *Dosimetry for nonuniform activity distributions: A method for the calculation of 3d absorbed-dose distribution without the use of voxel s-values, point kernels, or monte carlo simulations*. 2013 (zitiert auf Seite 1).
- [Bot10] L. Bottou. „Large-Scale Machine Learning with Stochastic Gradient Descent“. In: *NEC Labs America, Princeton NJ 08542, USA* (2010) (zitiert auf den Seiten 23, 24).
- [Bra84] R.N. Bracewell. *The Fast Hartley Transform*. 1984 (zitiert auf Seite 2).
- [CD13] K. He X. Tang C. Dong C.C. Loy. „Learning a deepconvolutional network for image super-resolution“. In: *ECCV* (2013) (zitiert auf Seite 30).
- [CP16] Z. Tu Ch.L. Patrick W.Gallagher. „Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree“. In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS)* (2016) (zitiert auf Seite 22).
- [DA85] T.J. Sejnowski D.H. Ackley G.E. Hinton. *A learning algorithm for Boltzmann machines*. 1985 (zitiert auf Seite 14).
- [DE13] R. Fergus D. Eigen D. Krishnan. „Restoring an imagetaken through a window covered with dirt or rain“. In: *ICCV* (2013) (zitiert auf Seite 30).
- [DK15] J. Lei Ba D.P. Kingma. „ADAM: A Method for Stochastic Optimization“. In: *ICLR* (2015) (zitiert auf den Seiten 24–26).
- [DR86] R.J. Williams D.E. Rumelhart G.E. Hinton. *Learning representations by back-propagating errors*. 1986 (zitiert auf Seite 11).
- [Doz16] T. Dozat. „Incorporating Nesterov Momentum into Adam“. In: *ICLR* (2016) (zitiert auf den Seiten 25–27).
- [Elm91] J.L. Elman. *Distributed Representations, Simple Recurrent Networks, and Grammatical Structure*. 1991 (zitiert auf Seite 14).
- [FB11] G. Battistoni M. Cremonesi A. Di Dia A. Fasso A. Ferrari M. Ferrari G. Paganelli G. Pedrolì F. Botta A. Mairani. *Calculation of electron and isotopes dose point kernels with fluka monte carlo code for dosimetry in nuclear medicine therapy*. 2011 (zitiert auf Seite 1).
- [FG03] J. Schmidhuber F.A. Gers N.N. Schraudolph. „The Journal of Machine Learning Research“. In: *A Field Guide to Dynamical Recurrent Neural Networks* (2003) (zitiert auf Seite 18).
- [FL15] G. Lin F. Liu C. Shen. „Deep convolutional neural fields for depth estimation from a single image“. In: *CVPR* (2015) (zitiert auf Seite 30).

- [GG17] B. Schuller G. Geren. „Convolutional RNN: an Enhanced Model for Extracting Features from Sequential Data“. In: *arXiv:1602.05875v3 [stat.ML]* (2017) (zitiert auf Seite 30).
- [GH12] D. Yu G.E. Dahl A. Mohamed N. Jaitly A. Senior V. Vanhoucke P. Nguyen T.N. Sainath B. Kingsbury G. Hinton L. Deng. *Deep Neural Network for Acoustic Modelin in Speech Recognition*. 2012 (zitiert auf Seite 14).
- [GL16] A. van den Hengel I. Reid G. Lin C. Shen. „Efficientpiecewise training of deep structured models for semanticsegmentation“. In: *CVPR* (2016) (zitiert auf Seite 30).
- [GS08] R. Wahl B. He A. Prideaux R. Hobbs G. Sgouros E. Frey. *Three dimensional imaging based radiobiological dosimetry*. 2008 (zitiert auf Seite 2).
- [GT07] S. Roweis G.W. Taylor G.E. Hinton. *Modeling human motion using binary latent variables*. 2007 (zitiert auf Seite 14).
- [Gro88] S. Grossberg. *Nonlinear Neural Networks: Principles, Mechanisms, and Architectures*. 1988 (zitiert auf Seite 8).
- [HMS14] N. Vasconcelos H. Masnadi-Shirazi. *On the Design of Loss Functions for Classification: theory, robustness to outliers, and SavageBoost*. 2014 (zitiert auf Seite 10).
- [Hop82] J.J. Hopfield. *Neural networks and physical systems with emergent collective computational abilities*. 1982 (zitiert auf Seite 14).
- [IS13a] P. Pérez I. Scarinci M. Valente. *Dose point kernel calculation and modelling with nuclear medicine dosimetry purposes*. 2013 (zitiert auf Seite 1).
- [IS13b] G. Dahl G. Hinton I. Sutskever J. Martens. „On the importance of initialization and momentum in deep learning“. In: *Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA. JMLR: W&CP Bd. 28* (2013) (zitiert auf Seite 25).
- [IS13c] G. Dahl G. Hinton I. Sutskever J. Martens. „On the importance of initialization and momentum in deep learning“. In: *In Proceedings of the 30th International Conference on Machine Learning (ICML-13). S. 1139–1147* (2013) (zitiert auf den Seiten 25, 26).
- [JD09] R. Socher L.-J. Li K. Li L. Fei-Fei J. Deng W. Dong. *ImageNet: A Large-Scale Hierarchical Image Database*. 2009 (zitiert auf Seite 19).
- [JD11] Y. Singer J. Duchi E. Hazan. „Adaptive subgradient methods for online learning and stochastic optimization“. In: *The Journal of Machine Learning Research, 12:2121–2159* (2011) (zitiert auf Seite 24).
- [JD15] J. Sun J. Dai K. He. „BoxSup: Exploiting boundingboxes to supervise convolutional networks for semantic segmentation“. In: *ICCV* (2015) (zitiert auf Seite 30).
- [JG15] M. Cox M. Ljungberg L. Johansson K. Gleisner J. Gustafsson G. Brodin. *Uncertainty propagation for spect/ct-based renal dosimetry in ¹⁷⁷lu peptide receptor radionuclide therapy*. 2015 (zitiert auf Seite 1).
- [JL15a] T. Darrell J. Long E. Shelhamer. *Fully convolutional networks for semantic segmentation*. 2015 (zitiert auf Seite 19).

- [JL15b] T. Darrell J. Long E. Shelhamer. „Fully convolutional networks for semantic segmentation“. In: *CVPR* (2015) (zitiert auf Seite 30).
- [JR15] R.B. Girshick A. Farhadi J. Redmon S.K. Divvala. *You only look once: Unified, real-time object detection*. 2015 (zitiert auf Seite 19).
- [KH16] Sh. Ren J. Sun K. He X. Zhang. *Identity Mappings in Deep Residual Networks*. 2016 (zitiert auf den Seiten 14, 35, 36).
- [KJ09] M.A. Ranzato Y. LeCun K. Jarrett K. Kavukcuoglu. *What is the best multi-stage architecture for object recognition?* 2009 (zitiert auf Seite 22).
- [KS14] A. Zisserman K. Simonyan. *Very deep convolutional networks for large-scale image recognition*. 2014 (zitiert auf Seite 19).
- [LB08] M.Ferrari L. Bodei M. Cremonesi. *Long-term evaluation of renal toxicity after peptide receptor radionuclide therapy with ⁹⁰Y-dotatoc and ¹⁷⁰Lu-dotatate: The role of associated risk factors*. 2008 (zitiert auf Seite 2).
- [LC15] I. Kokkinos K. Murphy A.L. Yuille L. Chen G. Papandreou. „Semantic image segmentation with deep convolutional nets and fully connected CRFs“. In: *ICLR* (2015) (zitiert auf Seite 30).
- [LG16] M.S. Gashler L.B. Godfrey. *A continuum among logarithmic, linear, and exponential functions, and its potential to improve generalization in neural networks*. 2016 (zitiert auf Seite 8).
- [LM13] A.Y. Ng L.A. Maas A.Y. Hannun. *Rectifier nonlinearities improve neural network acoustic models*. 2013 (zitiert auf den Seiten 8, 39).
- [LR03] A. Caponnetto M. Piana L. Rosacco E. De Vito. *Are Loss Functions All the Same?* 2003 (zitiert auf Seite 10).
- [Le15] Qu. V. Le. *A Tutorial on Deep Learning. Part 2: Autoencoders, Convolutional Neural Networks and Recurrent Neural Networks*. 2015 (zitiert auf den Seiten 15–18, 21, 22).
- [Lip87] R.P. Lippmann. *An Introduction to Computing with Neural Nets*. 1987 (zitiert auf den Seiten 9, 11).
- [Lis89] J. Lisman. „A mechanism for the Hebb and the anti-Hebb processes underlying learning and memory“. In: *Proc. Natl. Acad. Sci. USA, Neurobiologie. Bd. 86, S. 9574-9578* (1989) (zitiert auf Seite 7).
- [MF13] T. Mauxion M. Bardiès P. Kletting G. Glatting M. Lassmann M. Fernández H. Hänscheid. *A fast method for rescaling voxel s values for arbitrary voxel sizes in targeted radionuclide therapy from a single monte carlo calculation*. 2013 (zitiert auf Seite 1).
- [ML09] H. Jaeger M. Lukoševičius. *Reservoir Computing Approaches to Recurrent Neural Network Training*. 2009 (zitiert auf Seite 14).
- [MZ13] R. Fergus M.D. Zeiler. „Stochastic Pooling for Regularization of Deep Convolutional Neural Networks“. In: *arXiv preprint arXiv:1301.3557* (2013) (zitiert auf Seite 22).
- [Mur98] N. Murata. „A Statistical Study of On-line Learning“. In: *Online Learning and Neural Networks, Cambridge University Press* (1998) (zitiert auf Seite 24).

- [NK14] Ph. Blunsom N. Kalchbrenner E. Grefenstetter. „A Convolutional Neural Network for Modelling Sentences“. In: *arXiv:1404.2188v1 [cs.CL]* (2014) (zitiert auf den Seiten 21, 22).
- [NPH07] M. Zankl G. Sgouros B. Wessels N. Petoussi-Henss W. Bolch. *Patient-specific scaling of reference s-values for cross-organ radionuclide s-values: what is appropriate?* 2007 (zitiert auf Seite 2).
- [NR02] H. Thiersen R. Jeraj N. Reynaert H. Palmans. *Parameter dependence of the mcnp electron transport in determining dose distributions*. 2002 (zitiert auf Seite 3).
- [NS14] A. Krizhevsky I. Sutskever R. Salakhutdinov N. Sristava G. Hinton. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. 2014 (zitiert auf den Seiten 37, 38).
- [Nes83] Y. Nesterov. „A method of solving a convex programming problem with convergence rate $O(1/k^2)$ “. In: *In Soviet Mathematics Doklady, Bd. 27, S. 372–376* (1983) (zitiert auf den Seiten 26, 27).
- [OR15] T. Brox O. Ronneberger P. Fischer. „U-Net: Convolutional Networks for Biomedical Image Segmentation“. In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Bd. 9351. LNCS. 2015, S. 234–241 (zitiert auf den Seiten 30, 36, 38).
- [Pol64] B.T. Polyak. „Some methods of speeding up the convergence of iteration methods“. In: *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964 (1964) (zitiert auf Seite 25).
- [RG14] T. Darrell J. Malik R. Girshick J. Donahue. „Rich feature hierarchies for accurate object detection and semantic segmentation“. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2014) (zitiert auf Seite 29).
- [RS15a] J. Schmidhuber R.K. Srivastava K. Greff. „Highway networks“. In: (2015) (zitiert auf Seite 35).
- [RS15b] J. Schmidhuber R.K. Srivastava K. Greff. „Training very deep networks“. In: (2015) (zitiert auf Seite 35).
- [Ros62] F. Rosenblatt. *Principles of Neurodynamics*. 1962 (zitiert auf Seite 8).
- [SC06] A. Bitar A. Lisbona J. Barbet D. Franck J.R. Jourdan M. Bardies S. Chiavasse I. Aubineau-Laniece. *Validation of a personalized dosimetric evaluation tool (Oedipe) for targeted radiotherapy based on the Monte Carlo MCNPX code*. 2006 (zitiert auf Seite 3).
- [SH01] P. Frasconi J. Schmidhuber S. Hochreiter Y. Bengio. „Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.“ In: *A Field Guide to Dynamical Recurrent Neural Networks* (2001) (zitiert auf Seite 17).
- [SH17] Th. Unterthiner A. Mayr S. Hochreiter G. Klambauer. „Self-Normalizing Neural Networks“. In: *31st Conference on Neural Information Processing Systems (NIPS 2017)* (2017) (zitiert auf Seite 18).
- [SH97] J. Schmidhuber S. Hochreiter. „Long short-term memory“. In: *Neural Computation* (1997) (zitiert auf den Seiten 18, 35).

- [SI15] Ch. Szegedy S. Ioffe. „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift“. In: *Proceedings of the 32nd International Conference on Machine Learning*. Hrsg. von Francis Bach und David Blei. Bd. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, 2015, S. 448–456 (zitiert auf den Seiten 32–34).
- [SR15] R.B. Girshick J. Sun S. Ren K. He. *Faster R-CNN: towards real-time object detection with region proposal networks*. 2015 (zitiert auf Seite 19).
- [SW14] R. Schlüter H. Ney S. Wiesler R. Alexander. „IEEE International Conference on Acoustics, Speech, and Signal Processing, Firenze, Italia“. In: (2014), S. 180–184 (zitiert auf Seite 33).
- [She05] Y. Shen. *Loss Functions for Binary Classification and Class Probability Estimation*. 2005 (zitiert auf Seite 10).
- [TJ12] A. Torralba T. Judd F. Durand. „A Benchmark of Computational Models of Saliency to Predict Human Fixations“. In: *MIT Technical Report*. 2012 (zitiert auf den Seiten 32, 40).
- [TT12] G. Hinton T. Tieleman. „Technical Report. Lecture 6.5 - RMSProp“. In: *COURSERA: Neural Networks for Machine Learning* (2012) (zitiert auf Seite 24).
- [TYL14] S. Belongie J. Hays P. Perona D. Ramanan P. Dollr C.L. Zitnick T.-Y. Lin M. Maire. *Microsoft coco: Common objects in context*. 2014 (zitiert auf Seite 19).
- [WB99] J.S. Robertson W.E. Bolch L.G. Bouchet. *MIRD pamphlet no. 17: the dosimetry of nonuniform activity distributions – radionuclide s-values at the voxel level*. 1999 (zitiert auf den Seiten 2, 4).
- [Wer07] D. Werner. *Funktionalanalysis*. 2007⁶ (zitiert auf den Seiten 19, 53).
- [YB11] F. Bach J. Ponce Y. LeCun Y. Boureau N. Le Roux. „Ask the locals: multi-way local pooling for image recognition.“ In: *ICCV* (2011) (zitiert auf Seite 22).
- [YBL10] J. Ponce Y. Boureau und Y. LeCun. „A Theoretical Analysis of Feature Pooling in Visual Recognition“. In: *ICML* (2010) (zitiert auf Seite 22).
- [YL15] G. Hinton Y. LeCun Y. Bengio. *Deep learning*. 2015 (zitiert auf den Seiten 7, 8, 10).
- [YL64] G.B. Orr K.-R. Müller Y. Lecun L. Bottou. „Efficient BackProp“. In: *Neural Networks: Tricks of the trade, Springer* (1964) (zitiert auf den Seiten 38, 55).
- [YL98] G. Orr K. Muller Y. LeCun L. Bottou. „Efficient backprop“. In: *Springer* (1998) (zitiert auf den Seiten 33, 34).
- [ZB16] A. Oliva A. Torralba F. Durand Z. Bylinskii T. Judd. „What do different evaluation metrics tell us about saliency models?“ In: *arXiv preprint arXiv:1604.03605* (2016) (zitiert auf den Seiten 32, 40).
- [ÖÇ16] S.S. Lienkamp T. Brox O. Ronneberger Ö. Çiçek A. Abdulkadir. „3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation“. In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Hrsg. von M.R. Sabuncu G. Unal S. Ourselin W.S. Wells und L. Joskowicz. Bd. 9901. LNCS. 2016, S. 424–432 (zitiert auf den Seiten 14, 30, 36, 38).

WEBSEITEN

- [Aba+15] M. Abadi, A. Agarwal, P. Barham et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: <https://www.tensorflow.org/> (zitiert auf Seite 38).
- [Mel18] L. Melodia. *DVK-uNet – Neuronales Netz zur Schätzung von Dosis-Voxel-Kernen*. 2018. URL: <https://github.com/karhunenloeve/DeepLearningCNN> (zitiert auf Seite 38).
- [ORNLO2] U.S. Department of Energy Oak Ridge National Laboratory. *Monte Carlo N-Particle Transport Code System for Multiparticle and High Energy Applications*. 2002. URL: <http://www.mcnp.ir/admin/imgs/1354175991.C715.PDF> (besucht am 9. Nov. 2017) (zitiert auf Seite 3).
- [Wik17] The Free Encyclopedia Wikipedia. *Activation Function*. 2017. URL: https://en.wikipedia.org/wiki/Activation_function (besucht am 23. Nov. 2017) (zitiert auf Seite 8).
- [al.15] F. Chollet et. al. *Keras*. 2015 (zitiert auf Seite 38).

A

APPENDIX

ZUSÄTZLICHES ZUM KAPITEL 2: DEEP LEARNING

Im Nachfolgenden soll ein analytisches Beispiel der Faltung von $\cos(x)$ mit dem $\sin(x)$, also stetiger Funktionen, gegeben werden. Dadurch wird verständlicher gemacht, was das Resultat der Faltung hervorbringt.

$$(f \otimes g)(x) = \int_0^x \underbrace{\sin x - \tau}_{\sin x \cos \tau - \sin \tau \cos x} \cdot \cos \tau d\tau = \quad (4.3)$$

$$= \int_0^x (\sin x \cos \tau - \sin \tau \cos x) \cdot \cos \tau d\tau = \quad (4.4)$$

$$= \int_0^x \sin x \cos^2 \tau - \cos x \sin \tau \cos \tau d\tau = \quad (4.5)$$

$$= \int_0^x \underbrace{\sin x}_{\text{Konstante}} \cos^2 \tau d\tau - \int_0^x \underbrace{\cos x}_{\text{Konstante}} \sin \tau \cos \tau d\tau \quad (4.6)$$

Für die weiteren Umformungen werden die Identitäten $\cos^2 \tau = \frac{1}{2}(1 + \cos 2\tau)$ und $\frac{d \sin \tau}{d\tau} = \cos \tau$ vgl. [Wer07] verwendet.

$$= \sin x \left(\int_0^x \cos^2 \tau d\tau \right) - \cos x \left(\int_0^x \sin \tau \cos \tau d\tau \right) = \quad (4.7)$$

$$= \frac{1}{2} \sin x \left(\int_0^x (1 + \cos 2\tau) d\tau \right) - \cos x \left(\int_0^x \sin \tau \cos \tau d\tau \right) = \quad (4.8)$$

$$= \frac{1}{2} \sin x \left[\left(\tau + \frac{1}{2} \sin 2\tau \right) \right]_0^x - \cos x \left[\frac{1}{2} \sin^2 \tau \right]_0^x = \quad (4.9)$$

$$= \frac{1}{2} \sin x \left(x + \frac{1}{2} \sin 2x \right) - \cos x \left(\frac{1}{2} \sin^2 x - 0 \right) = \quad (4.10)$$

$$= \frac{1}{2} x \sin x + \frac{1}{4} \sin x \underbrace{\sin 2x}_{2 \sin x \cos x} - \frac{1}{2} \sin^2 x \cos x = \quad (4.11)$$

$$= \frac{1}{2} x \sin x + \frac{1}{4} \sin x (2 \sin x \cos x) - \frac{1}{2} \sin^2 x \cos x = \quad (4.12)$$

$$= \frac{1}{2} x \sin x + \frac{1}{2} \sin^2 x \cos x - \frac{1}{2} \sin^2 x \cos x = \quad (4.13)$$

$$= \frac{1}{2} x \sin x \quad (4.14)$$

Für diskrete Mengen ist die analytische Herangehensweise analog. Vereinfacht ausgedrückt, ist das Resultat einer Faltung der Anteil einer Funktion an der anderen, versetzt um einen bestimmten Zeitschritt, welcher vorher meist als Konstante definiert wurde.

Abb. 4.2: Scree-Plot der Massendichten.

Unter der Hauptachsentransformation oder eng. *Principal Component Analysis* (PCA) versteht man eine Methode zur Zusammenfassung der Eigenschaften einer Menge multivarianter Datenmuster. Die PCA ist dabei eine lineare Transformationsmethode, welche häufig zur Vorverarbeitung oder Datenkomprimierung eingesetzt wird. Das Ziel einer PCA ist die Bestimmung von n normierten orthogonalen Vektoren $\mathbf{u}_i, i = 1, \dots, n$ im Eingangsraum, welche möglichst viel von der Varianz der Daten repräsentieren. Hierfür werden zunächst die Vektoren auf Null zentriert. Betrachtet man das Orthonormalsystem $\{\mathbf{u}_i\}$ gemäß:

$$\langle \mathbf{x} \rangle = \int \mathbf{x} p(\mathbf{x}) d^m \mathbf{x} = 0 \quad (4.15)$$

$$\|\mathbf{u}\| = \left(\sum_i \mathbf{u}_i^2 \right)^{\frac{1}{2}} = 1, \text{ und } \langle \mathbf{x}^T \mathbf{u} \rangle = 0 \quad (4.16)$$

Ziel ist es eine Darstellung $\mathbf{u}^*$ zu finden, sodass $\langle \mathbf{x}^T \mathbf{u}^* \rangle$ die Varianz der Projektionen von $\mathbf{x}$ auf $\mathbf{u}^*$ nach Maßgabe der Wahrscheinlichkeitsverteilung $p(\mathbf{x})$ maximal wird. Gesucht wird die Lösung zu folgendem Optimierungsproblem:

$$\mathcal{L}(\mathbf{w}) = \left\langle \left(\frac{\mathbf{x}^T \mathbf{w}}{\|\mathbf{w}\|} \right)^2 \right\rangle = \langle (\mathbf{x}^T \mathbf{u})^2 \rangle = \mathbf{u}^T \langle \mathbf{x} \mathbf{x}^T \rangle \mathbf{u} = \mathbf{u}^T \mathbf{C} \mathbf{u} \quad (4.17)$$

Wobei $\mathbf{C}$ die Kovarianzmatrix der Datenmuster meint. Die Eigenwerte erhält man mittels PCA in absteigender Reihenfolge. Die Eigenwerte geben dabei die Varianz entlang der Richtung des entsprechenden Eigenvektors an. Die Lösung für dieses Optimierungsproblem wurde vollständig berechnet und ist in Abb. 4.2 zu sehen.

Schicht (Typ)	# Dim.	Maske	Aktivierung	# Param.	Vorher
Eingang	(9, 9, 9)	–	–	0	–
Reshape	(27, 27, 1)	–	–	0	Eingang
UpSampling-1	(54, 54, 1)	–	(2, 2)	0	Reshape
Conv2D-1 [‡]	(52, 52, 8)	(3, 3)	LeakyReLU	80	UpSampling-1
Conv2D-2 [‡]	(50, 50, 8)	(3, 3)	LeakyReLU	584	Conv2D-1
Conv2D-3 [‡]	(48, 48, 16)	(3, 3)	LeakyReLU	1168	Conv2D-2
Conv2D-4 [‡]	(46, 46, 16)	(3, 3)	LeakyReLU	2320	Conv2D-3
Conv2D-5 [‡]	(44, 44, 32)	(3, 3)	LeakyReLU	4640	Conv2D-4
Conv2D-6 [‡]	(42, 42, 32)	(3, 3)	LeakyReLU	9248	Conv2D-5
AVGPooling	(21, 21, 32)	(3, 3)	–	0	Conv2D-6
Conv2D-7 [‡]	(19, 19, 32)	(3, 3)	LeakyReLU	9248	AVGPooling
Conv2D-8 [‡]	(17, 17, 32)	(3, 3)	LeakyReLU	9248	Conv2D-7
Conv2D-9 [‡]	(15, 15, 64)	(3, 3)	LeakyReLU	18496	Conv2D-8
Conv2D-10 [‡]	(13, 13, 64)	(3, 3)	LeakyReLU	36928	Conv2D-9
Conv2D-11 [‡]	(11, 11, 32)	(3, 3)	LeakyReLU	18464	Conv2D-10
Conv2D-12 [‡]	(9, 9, 32)	(3, 3)	LeakyReLU	9248	Conv2D-11
Conv2D-13 [‡]	(7, 7, 32)	(3, 3)	LeakyReLU	9248	Conv2D-12
Dropout	–	–	0.2	–	Conv2D-13
UpSampling-2	(42, 42, 32)	–	(6, 6)	0	Dropout
Concat	(39, 39, 32)	–	–	0	UpSampling-2
Conv2D-14 [‡]	(37, 37, 32)	(3, 3)	LeakyReLU	32800	Conv2D-13
Conv2D-15 [‡]	(35, 35, 16)	(3, 3)	LeakyReLU	4624	Conv2D-14
Conv2D-16 [‡]	(33, 33, 16)	(3, 3)	LeakyReLU	2320	Conv2D-15
Conv2D-17 [‡]	(31, 31, 8)	(3, 3)	LeakyReLU	1160	Conv2D-16
Conv2D-18 [‡]	(29, 29, 8)	(3, 3)	LeakyReLU	584	Conv2D-17
Conv2D-19 [‡]	(27, 27, 4)	(3, 3)	LeakyReLU	292	Conv2D-18
Conv2D-20	(27, 27, 1)	(1, 1)	Sigmoid	5	Conv2D-19
Reshape	(9, 9, 9)	–	–	0	Conv2D-20

Parameter: 182.017

trainierbare Parameter: 180.985

nicht trainierbare Parameter: 1.032

Tab. 4.1: Für die Rekonstruktion der absorbierten Strahlungsdosis wurden unterschiedliche Trainingsvorgänge miteinander verglichen. Die Regularisierungen und Modifikationen sollen in der obigen Tabelle illustriert werden. Die Reihenfolge innerhalb der Faltungsschicht ist: 1. Faltung, 2. *LeakyReLU*-Aktivierung ($\alpha = 5.5$), 3. *Batch*-Normalisierung. Alle Matrixkerne dieses Netzes wurden für die Experimente mit der Lecun-Normalverteilung initialisiert [YL64]. Die Daten wurden mittels *Min-Max*-Normalisierung ($\mathbf{X}_{ijk} = (b - a) \times \frac{\mathbf{X}_{ijk} - \min(\mathbf{X})}{\max(\mathbf{X}) - \min(\mathbf{X})} + a, \forall i, j, k$) auf ein Intervall $[a, b]$ abgebildet. Die in Kap. 3 erwähnten Regularisierungen wurden wie folgt umgesetzt, wobei die Symbole die entsprechenden Schichten in der Tabelle markieren: [‡] In dieser Schicht wurde eine Regularisierung der L_1 -Norm vorgenommen, mit einer Strafkonstante $\gamma = 0.005$. [†] Hier wurde bei der Faltung eine Regularisierung der L_2 -Norm für jeden Filter durchgeführt, mit der Konstante $\gamma = 0.001$.

Abb. 4.3: Querschnitt der Massendichten und absorbierten Strahlungsdosis entlang der i -Achse. Im groß hinterlegten Bild sieht man die Massendichten einer spezifischen Gewebeklasse als Ausschnitt aus einem Massenkern. Im jeweils kleineren Bild ist die dazugehörige absorbierte Strahlungsdosis. Im Bild erkennt man deutlich zusammengehörige Quadrate. Jedes Quadrat ist dabei ein unabhängiger Ausschnitt des Gewebes der zufällig aus der Gesamtmenge gezogen wurde. Beispiel *A* ist die Dichte und absorbierte Strahlungsdosis des Knochens, *B* die der Niere, *C* ist die der Leber, *D* die der Lunge und *E* die der Milz. Für jedes Bild sind 36 Querschnittspaare zu sehen. Je dunkler die Fläche ist, desto dichter ist das Gewebe. Für die absorbierte Strahlung ist die helle Fläche die höchste Strahlungsdosis und die dunkle die geringste.

Abb. 4.4: Schätzergebnisse des *Deep Learning* Systems. Der Querschnitt *A* stammt aus der Leber, *B* aus dem Nierengewebe und *C* zeigt Knochengewebe. Die Schnitte der Dichtekerne wurden entlang der *i*-Achse des Datentensors an der fünften von neun Stellen entnommen. Dies hat den Hintergrund, dass die meiste Aktivität im Zentrum des Dichtekerns stattfindet, wo das radioaktive Nuklid simuliert wurde. Besonders deutlich ist zu sehen, dass die Skalierungen von Vorhersage (*Prediction*) und Sollausgabe (*Output*) sehr gut übereinstimmen. In *A*, *C* ist ebenfalls eine gute Rekonstruktion der detaillierten Verteilung der Strahlungsenergie in sehr kleinem Maßstab gelungen.

ERWÄHNUNG

Die Daten für das Projekt wurden vom Universitätsklinikum in Erlangen bereitgestellt. Für die Berechnungen und die Zurverfügungstellung der Daten möchte ich mich herzlich bedanken.