

# Deep Learning - Lernen von vielschichtigen neuronalen Netzen

KJARTAN FERSTL

DIPLOMARBEIT

eingereicht am

Fachhochschul-Masterstudiengang

INFORMATION ENGINEERING UND -MANAGEMENT

in Hagenberg

im Juni 2014



# Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Hagenberg, am 12. Juni 2014

Kjartan Ferstl

# Inhaltsverzeichnis

<b>Erklärung</b>	<b>i</b>
<b>Kurzfassung</b>	<b>iv</b>
<b>Abstract</b>	<b>v</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Ziele und Aufgaben . . . . .	2
<b>2 Grundlegendes</b>	<b>3</b>
2.1 Definition . . . . .	3
2.2 Entstehung . . . . .	3
2.3 Hürden . . . . .	4
2.4 Neuronale netze . . . . .	4
2.5 Überwacht / Unüberwacht . . . . .	6
<b>3 Algorithmen</b>	<b>7</b>
3.1 Backpropagation . . . . .	7
3.2 Resctricted Boltzmann Maschines . . . . .	9
3.2.1 Grundgedanke . . . . .	10
3.2.2 Abkürzung . . . . .	11
3.2.3 Begründung . . . . .	11
3.2.4 Eignung . . . . .	11
3.3 Deep Autoencoders . . . . .	11
3.4 Faltungscodierte neurale Netze . . . . .	12

Inhaltsverzeichnis	iii
3.5 Sparse coding . . . . .	13
<b>4 Anwendungen</b>	<b>15</b>
4.0.1 Google Projekt . . . . .	15
4.1 Google Street View . . . . .	16
4.2 Spracherkennung . . . . .	17
4.3 Hardware . . . . .	17
4.4 Weitere Anwendungen . . . . .	18
<b>5 Zusammenfassung</b>	<b>19</b>
<b>Literaturverzeichnis</b>	<b>21</b>

# Kurzfassung

Deep Learning-Algorithmen bieten eine Möglichkeit vielschichtige Neuronale Netze zu trainieren. Solche Netze kommen in aktuellen Anwendungen zur Erkennung von Objekten in Bildern, Text und Inhalt in Sprache und vielen anderen Gebieten zum Einsatz. Forschungen an den Algorithmen ermöglichen es, die sehr rechenintensiven Algorithmen zu vereinfachen und in großem Stil auf parallelisierten Computersystemen auszuführen. So ist es in einigen Gebieten bereits möglich mit neuronalen Netzen bessere Ergebnisse als mit optimierten Mathematischen Verfahren zu erzielen. In folgendem werden die wesentlichen Algorithmen und Strukturen von neuronalen Netzen aufgearbeitet und die Schwierigkeiten und Lösungsansätze aufgezeigt. Außerdem werden einige aktuelle Anwendungen aus der Bild- und Sprachanalyse betrachtet.

# Abstract

Deep learning algorithms enable it to train multi player neural networks. Such networks are used to recognize objects in images, to extract text and content in speech and for many other applications. Research in this area made it possible to reduce the complexity of such algorithms and to compute them in parallel on large scale computer networks. Due to these optimization, it is already possible, to produce better results with an neural network than with existing optimized mathematical algorithms for some applications. This paper is concerned about the major algorithms and structures used for neural networks, the problems they come with and how they can be solved. Moreover some applications from object recognition in images and voice analysis are examined.

# Kapitel 1

## Einleitung

### 1.1 Motivation

Komplexe neuronale Netze können bei einigen Problemstellungen, besonders in der Bild- und Spracherkennung ein sehr mächtiges Mittel zur Problemlösung sein. Das Lernen solcher Netze bringt Schwierigkeiten mit sich die in den vergangenen Jahrzehnten schlecht oder gar nicht gelöst werden konnten.

Aufgrund der Weiterentwicklung von PC-Hardware und der Verwendung der Grafik-Recheneinheit (GPU), sind seit den 00er-Jahren bereits bemerkenswerte Ergebnisse möglich. Momentan arbeiten sogar Chiphersteller an preiswerten, dedizierten Chips mit trainierbaren neuronalen Netzen. Qualcomm wird den ersten kommerziellen Chip mit einem integrierten neuronalen Netz noch dieses Jahr (2014) veröffentlichen. Neuronale Netze sind immer nur so gut, wie sie trainiert werden, eines der wesentlichsten Themen im weiteren Fortschritt von neuronalen Netzen sind daher die Algorithmen zum Trainieren.

Aus der eingeleiteten Motivation ergeben sich daher folgende Kernaufgaben für die Seminararbeit:

- Einführung in die Problematik von Deep Learning mit kurzem Blick auf die bisherige Geschichte
- Analyse der Hürden im Deep Learning
- Analyse aktueller Deep Learning-Algorithmen



## 1.2 Ziele und Aufgaben

Folgende zentrale Fragestellungen sollen in der Seminararbeit beantwortet werden:

- Was ist Deep Learning?
- Welche Mustertypen können erkannt werden und welche Anwendungen sind möglich
- Wie sind die gelernten Modelle konkret definiert?
- Welche Verfahren werden verwendet um vielschichtige Modelle zu lernen?
- Was ist aktuell mit Deep Learning in Verbindung mit neuronalen Netzen bei der Mustererkennung in Bildern möglich?

Ziel dieser Arbeit ist es, dem interessierten Leser einen Einstieg in Themen deep learning, mit besonderem Fokus auf neuronale Netze zu vermitteln. Aufbauend auf diesen Erkenntnissen soll gezeigt werden, welche wesentlichen Algorithmen bis heute entwickelt wurden und welche technischen Problemstellungen mit diesen Algorithmen gelöst werden können oder sogar bereits heute damit gelöst werden. Ein wesentlicher Fokus soll dabei auf die Mustererkennung in Bildern gerichtet werden.

# Kapitel 2

## Grundlegendes

### 2.1 Definition

Deep Learning ist ein Gebiet aus dem maschinellen Lernen, das sich mit dem Trainieren von nicht linearen Modellen und hier meist mit vielschichtigen neuronalen Netzen befasst.

Deep Learning wird zum Beispiel zur Klassifizierung von Daten oder zur Extraktion von Merkmalen aus Daten eingesetzt.

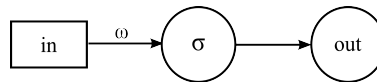
### 2.2 Entstehung

Die Wurzeln des Deep Learnings gehen zurück in die 1950er Jahre, als Frank Rosenblatt eine Maschine namens Perceptron [?] baute. Diese Maschine war in der Lage einige einfache Figuren wie Quadrate und Dreiecke zu erkennen. Für die damalige Zeit war das eine herausragende Maschine und regte den Gedanken, Maschinen zu bauen die Menschen imitieren können weiter an.

Knapp 20 Jahre später schrieb Marvin Minsky [?] ein Buch das die Limitierungen der Maschine aufzeigte und einige scheinbar fundamentale Probleme aufwarf. Diese Erscheinung ließ die neuronalen Netze wieder in den Hintergrund rücken. In den 1980er Jahren, hat der heute renommierte Wissenschaftler Geoff Hinton, eine Maschine gebaut [?], die bereits eine versteckte Schicht besaß und somit in der Lage war, komplexere Aufgaben zu lösen. In dieser Zeit entstand auch der erste Backpropagation Algorithmus. Das trainieren dieser Maschine war sehr aufwendig und so verschwand auch sie bald wieder von der Bildfläche.

Erst mit einer Entdeckung 2006 [?], erneut durch Geoff Hinton, die den Backpropagation Algorithmus erheblich vereinfachte, haben neuronale Netze mit mehreren versteckten Schichten wieder an Bedeutung gewonnen.

ref

**Abbildung 2.1:** Neuron

Heute stellen führende Unternehmen immer häufiger Teams rund um neuronale Netze und das Trainieren dieser zusammen um sich wirtschaftliche Vorteile zu sichern <sup>12</sup>. Erste Erfolge sieht man an Microsofts Spracherkennung Cortana, Siri von Apple, Google Now oder der Bildersuche von Google, die anhand von Bildern ähnlichen Bildern findet.

ref google, apple

## 2.3 Hürden

Die aus der Vergangenheit bekannten Probleme des Lernens neuronaler Netze mit vielen versteckten Schichten strecken sich zum größten Teil bis heute durch. Aktuelle Algorithmen aus dem Deep Learning tendieren zu Überanpassung, Unteranpassung, lokalen Minima oder kämpfen einfach mit der nicht vorhandenen Rechenleistung.

Grundsätzlich kann man für heute verwendete Algorithmen sagen, dass die Performance von neuronalen Netzen sehr stark von der Menge der Trainingsdaten und weniger von dem Algorithmus selbst abhängt. Nicht zuletzt hat die Verfügbarkeit von immensen Datenmengen von Unternehmen und aus dem Internet einen sehr positiven Einfluss auf die Entwicklungen und damit das Interesse von Deep Learning.

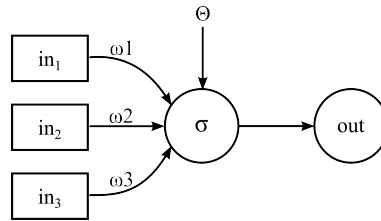
## 2.4 Neuronale netze

Neuronale Netze sind Strukturen aus der Technik, die dem Nervensystem von Lebewesen ähneln. Es sind Modelle, die Eingangsdaten über Neuronen gewichtet kombinieren und daraus einen Ausgang errechnen. Es sind Netze die aus Eingangsdaten Ausgangsdaten über nicht lineare Zusammenhänge berechnen. Diese Netze müssen parametrisiert werden, hier kommt Deep Learning zum Einsatz.

Neuronale Netze sind in der Technik zur Vereinfachung meist in mehreren Schichten aufgebaut. Abbildung 2.1 zeigt ein einfaches Neuron mit nur einem Eingang. Der Ausgang wird aus der Multiplikation des Eingangswerts mit dem Gewicht  $\omega$  und der Übertragungsfunktion errechnet. Als Übertragungsfunktion wird meist Sigmoid-Funktion eingesetzt, sie lässt sich einfach

<sup>1</sup>2013/3 Google stellt Geoffrey Hinton

<sup>2</sup>2013/12 Facebook stellt NYU Professor Yann LeCun ein

**Abbildung 2.2:** Neuron mit mehreren Eingängen**Abbildung 2.3:** Schichten von Neuronen

differenzieren und eignet sich daher besonders gut. Der Ausgang *out* dieses Neurons ergibt sich somit aus der dem Eingang *in* multipliziert mit dem Gewicht  $\omega$  in der Sigmoid-Funktion  $\sigma$ .

$$out = \sigma(\omega * in)$$

Ein Neuron hat in der Regel, wie in Abbildung 2.2 zu sehen, mehrere Eingänge. Außerdem wird ein Bias-Wert  $\theta$  für die Sigmoid-Funktion hinzugefügt. Der Ausgang dieses Neurons kann somit wie folgt berechnet werden:

$$out = \sigma(\omega_1 * in_1 + \omega_2 * in_2 + \omega_3 * in_3 + \theta)$$

Um aus den einzelnen Neuronen ein Netzwerk zu bauen, werden, wie in Abbildung 2.3 dargestellt Schichten gebildet. Eine Schicht ist eine Menge an Neuronen, die, zur Vereinfachung, untereinander meist nicht direkt verknüpft sind. Schichten werden hintereinander gehängt, in dem jedes Neuron einer Schicht als Eingang für jedes Neuron der nächsten Schicht dient. Ein solches Netzwerk ist auch in Abbildung 2.4 zu sehen. Dieses Netzwerk hat eine Eingabeschicht, eine Ausgabeschicht und zwei Schichten dazwischen. Da die Ergebnisse dieser mittleren Schichten in der Regel nicht unmittelbar ein Ergebnis sind, werden sie als unsichtbar betrachtet und auch so bezeichnet. Es handelt sich hierbei also um die versteckten Schichten.

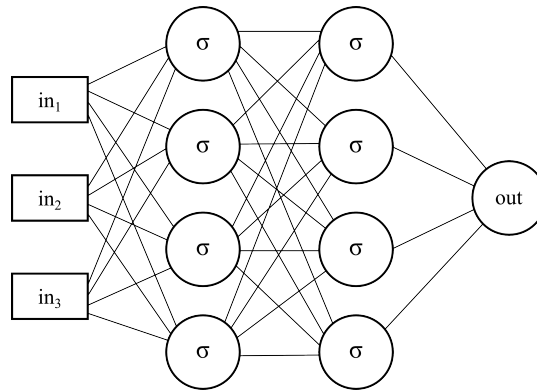


Abbildung 2.4: Ein Neuron

## 2.5 Überwacht / Unüberwacht

Die Algorithmen des Deep Learnings werden im groben in zwei Kategorien geteilt, in die überwachten und die unüberwachten Methoden. Überwacht bedeutet, dass Eingangsdaten an das System angelegt werden für die die gewünschten Ausgangsdaten bekannt sind. Das könnte zum Beispiel heißen, dass das Netz Bilder mit trainiert wird, von denen einige Gesichter enthalten und einige nicht. Am Ausgang wird dann überprüft ob das Netz die Gesichter richtig erkannt hat und wenn nicht, werden die Parameter des Netzes entsprechend angepasst.

Diese Art des Lernens scheint logisch, benötigt aber sehr viele deklarierte Datensätze. Deklarierte Datensätze sind selten in großen Mengen verfügbar und so tendiert diese Art der Algorithmen leicht zur Überanpassung, mehr dazu bei den jeweiligen Algorithmen.

Ein weitere Ansatz ist das unüberwachte Lernen, es handelt sich dabei um Algorithmen, die Eingangsdaten aber keine Ausgabedaten zum Vergleich benötigen. Die Grundlegende Idee dabei ist es, das Netz Merkmale aus den Eingangsdaten lernen zu lassen. Unüberwachtes Lernen ist besonders wegen der großen Verfügbarkeit unkategorisierten Daten interessant und wird häufig als Grundlage vor dem überwachten Lernen eingesetzt. Es hilft einige Probleme des überwachten Lernens zu verbessern, so können die Gewichte aus dem unüberwachten Lernen als Startwert für ein überwachtes Lernen eingesetzt werden. Diese Gewichte haben mehr Aussagekraft über Merkmale der Eingangsdaten als zufällige Werte und können mit weniger kategorisierten Trainingsdaten oft wesentlich bessere Ergebnisse erzielt werden.

## Kapitel 3

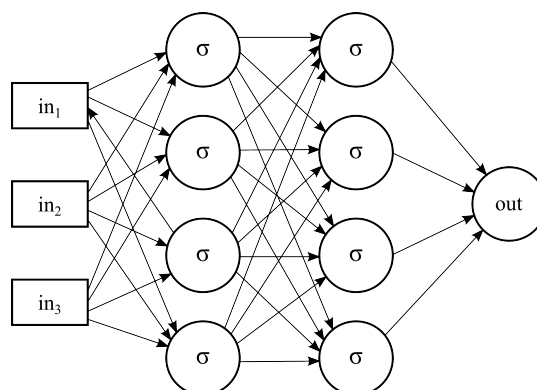
# Algorithmen und Strukturen

### 3.1 Backpropagation

Backpropagation ist ein überwachter Algorithmus zum Trainieren von vereinfachten neuronalen Netzen. Backpropagation setzt voraus, dass sich der Ausgang einer Schicht auf einfachem Weg berechnen lässt. Dies ist bei so genannten Feedforward-Perzeptrons, wie in Abbildung 3.1 zu sehen, der Fall. Überwacht bedeutet, dass das Netzwerk beim lernen anhand von deklarierten Eingabedaten Ausgabedaten zu errechnen, der Fehler der Berechnung des Ausgangs wird beim Backpropagation-Algorithmus über lineare Mathematik zur Adaptierung der Gewichte genutzt.

Der Trainingsprozess beginnt mit zufälligen Gewichten und arbeitet in den folgenden Schritten:

1. Ausgang des Netzes für bestimmte Eingabedaten berechnen



**Abbildung 3.1:** Ein Feedforward-Perzeptron

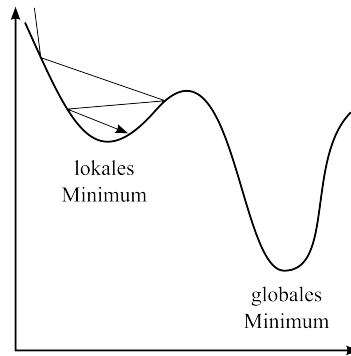


Abbildung 3.2: Lokales Minimum

2. Ausgang mit dem Sollwert vergleichen und daraus den Fehler errechnen

$$error = \frac{1}{2} \sum_{k \in K} (out_k - target_k)^2$$

<i>target</i>	...	Fehler, der später Einfluss in die Gewichte nimmt
<i>out</i>	...	Wert den der Ausgang erreicht hat
<i>target</i>	...	Wert den der Ausgang erreichen sollte
<i>K</i>	...	Anzahl der Neuronen in der Ausgangsschicht

3. Gewichte abhängig von der Größe des Fehlers anpassen

Diese Schritte werden wiederholt, bis die Gewichte so angepasst wurden, dass die Ausgangsdaten möglichst gut den Vorgaben entsprechen. Die Anpassung der Gewichte errechnet sich im wesentlichen anhand der Gradienten des Fehlers und der Gewichte. Zudem wird Tiefe der Schicht des Gewichtes berücksichtigt.

Wie der Name des Algorithmus bereits verrät, wird der Fehler dabei im wesentlichen über eine mathematische Funktion in die Gewichte zurückgeführt. Diese Rückführung bringt das Problem von lokalen Minima mit sich. Wird die Änderung des Fehlers geringer, so wird auch die Anpassung geringer. Befindet man sich auf dem Weg zu einem lokalen Minimum, so kann man durch den abnehmenden Fehler, wie in Abbildung 3.2 zu sehen, leicht darin hängen bleiben. Dieses lokale Minimum kann sehr weit von dem globalen Minimum entfernt sein. Die Wahrscheinlichkeit in dem lokalen Minimum hängen zu bleiben steigt mit seiner Größe. Abhilfe können hier Verfahren wie der Bergsteigeralgorithmus oder zusätzliche Zufallswerte bei der Berechnung der Gewichte schaffen.

Der Algorithmus trainiert vordefinierte Ein- und Ausgabedaten mit dem Ziel für ähnliche Daten ähnliche Ergebnisse zu liefern. Trainiert man das Netz

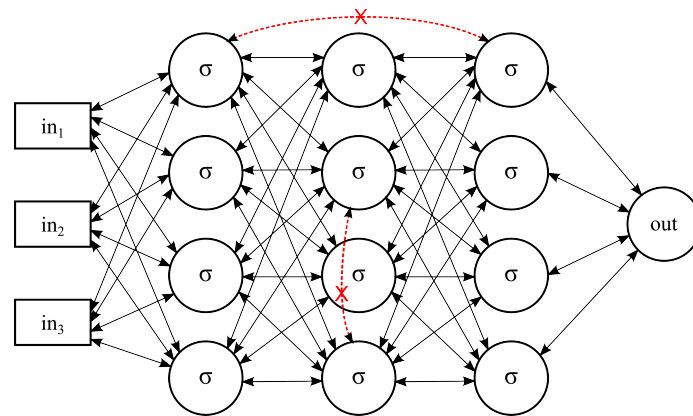


Abbildung 3.3: Eine Restricted-Boltzmann-Maschine

zu intensiv mit diesen Daten, so lernt es sie sehr genau. Dieser Effekt wird Überanpassung genannt und hat zur Folge, dass das Netz im späteren Betrieb schlechtere Ergebnisse liefert als hätte man es weniger lange trainiert. Abhilfe kann hier ein verfrühtes Ende des Trainings, mehr Trainingsdaten oder die Initialisierung der Gewichte durch unüberwachtes Lernen schaffen.

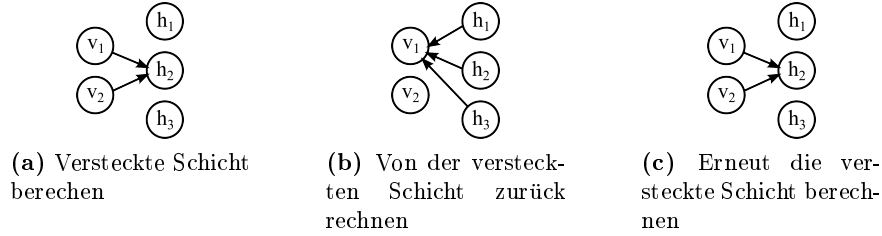
## 3.2 Resctricted Boltzmann Machines

Uneingeschränkte Neuronale-Netze sind schwierig und aufwendig zu trainieren. Restricted-Boltzmann-Maschinen, im weiteren als RBMs bezeichnet, sind eingeschränkte neuronale Netzwerke. Wie in Abbildung 3.3 dargestellt, existieren hierbei nur Verbindungen zwischen aneinander liegenden Schichten. Das Modell erlaubt keine Verbindungen von Neuronen zu sich selbst und alle Verbindungen müssen gleichermaßen in beide Richtungen verlaufen. So lässt sich unter Festlegung der Werte einer Schicht direkt auf die nächste versteckte Schicht weiter rechnen. RBMs arbeiten in der Regel mit boolschen Werten, lassen sich durch Erweiterung aber auch mit anderen Zahlenräumen verwenden.

RBMs wurden bereits 1986 von Paul Smolensky [?] unter dem Namen Harmonium erfunden, fanden jedoch erst 1998, rund 10 Jahre später, durch die Entwicklung von effizienten Lernalgorithmen durch Geoffrey Hinton Anwendung.

Zum trainieren von RBMs existieren verschiedene Algorithmen, die meist auf dem Prinzip beruhen, die Gewichte so anzupassen, dass das hin und her Rechnen zwischen zwei Schichten wieder die Ausgangsdaten ergibt. Im folgenden wird der Algorithmus von Geoffrey Hinton [?], der zugleich als der erste effizienten Lernalgorithmus für RBMs gesehen wird, erklärt.





**Abbildung 3.4:** Berechnungsschritte zum Lernen von RBMs.

### 3.2.1 Grundgedanke

Sind die Werte einer Schicht fixiert, so kann einfach auf die nächste versteckte Schicht weiter gerechnet. Zu Beginn werden ein Trainingsdatensatz an die Eingänge angelegt und die folgenden Operationen, wie auch in Abbildung 3.4 zu sehen, wiederholt ausgeführt:

1. Wahrscheinlichkeit  $p$  für die versteckten Neuronen  $h_j$  anhand der sichtbaren Neuronen  $v_i$  und der Gewichte  $w_{ij}$  berechnen

$$p(h_j = 1) = \frac{1}{1 + e^{-(b_j + \sum_i (v_i * w_{ij}))}}$$

2. Werte für die versteckte Schicht aus dem Mittelwert der Wahrscheinlichkeiten berechnen
3. Gradientenmatrix  $\langle v_i h_j \rangle$  über das dyadische Produkt errechnen

$$\langle v_i h_j \rangle = v * y^T$$

4. Ausgehend von den errechneten versteckten Neuronen, zurück auf die sichtbare Schicht rechnen

Wiederholt man die in der Liste angeführten Schritte sehr oft, so pendeln sich für die Neuronen Werte ein. Diese Werte hängen im wesentlichen vom Modell und den festgelegten Wahrscheinlichkeiten ab, haben jedoch sehr wenig mit den Eingangsdaten zu tun. Anhand der Gradientenmatrix des letzten Durchlaufes lässt sich jedoch eine Differenz zu dem gesuchten Modell herausfinden und so können die Gewichte wie folgt angepasst werden:

$$\Delta w_{ij} = \varepsilon (\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^\infty)$$

Diese Art der Berechnung lieferte sehr gute Modelle. Durch die vielen Iterationen benötigt der Algorithmus sehr viel Rechenzeit und ist daher nicht praxistauglich. Zudem ist es schwierig festzustellen, wie viele Iterationen notwendig sind bis sich die Werte eingependelt haben.

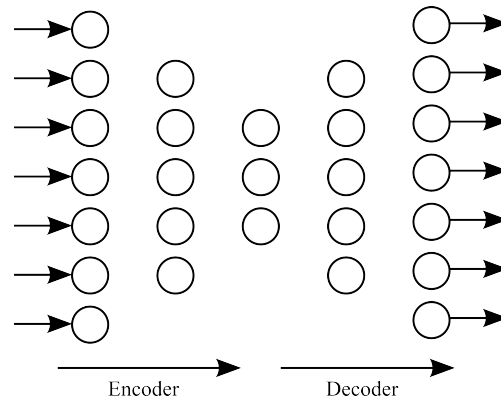


Abbildung 3.5: Deep Autoencoder

### 3.2.2 Abkürzung

Der oben genannte Algorithmus lässt sich in seiner Komplexität erheblich reduzieren, in dem die versteckte Schicht lediglich zwei mal berechnet wird. Geoffrey Hinton hat mit der Vereinfachung gezeigt, dass es bereits mit dem Vergleich der ersten und zweiten Gradientenmatrix möglich ist, gute Ergebnisse zu erzielen. Diese Methode wird *Contrastive Divergence* genannt.

Die Regel zur Anpassung der Gewichte lautet dabei:

$$\Delta\omega_{ij} = \varepsilon(<v_i h_j>^0 - <v_i h_j>^1)$$

### 3.2.3 Begründung

Der Grundgedanke von *Contrastive Divergence* ist es, dass das Modell mit Zufallsgewichten weg von den Eingabedaten, hin zu Daten die ihm besser gefallen, wandert. Wenn man erkennt wo hin das Modell will, kann man die Gewichte so adaptieren, dass das Modell zu den Eingabedaten will.

### 3.2.4 Eignung

Der Algorithmus extrahiert aus den nicht deklarierten Eingabedaten Merkmale die die Daten klassifizieren. Er eignet sich daher um Gemeinsamkeiten und von Eingabedaten, wie zum Beispiel Bildern, zu finden.

## 3.3 Deep Autoencoders

Deep Autoencoder sind neuronale Netze aus mehreren Schichten, die sich in zwei wesentliche Teile trennen lassen. Im ersten Teil, dem Encoder, sinkt

die Anzahl der Neuronen mit jeder Schicht. Im zweiten Teil, dem Decoder, steigt sie, bis am Ende wieder die Anzahl der Eingangsparameter erreicht ist. Abbildung 3.5 zeigt ein solches Netz. Im inneren hat das Netz weniger Neuronen und somit geringere Speicherkapazität als Eingangsparameter. Es wird versucht das Netz so zu trainieren, dass es am Ausgang die gleichen Daten berechnet wie am Eingang angelegt entstehen. Gelingt das Training, so muss das Netz Merkmale etwas aus den Daten gelernt haben.

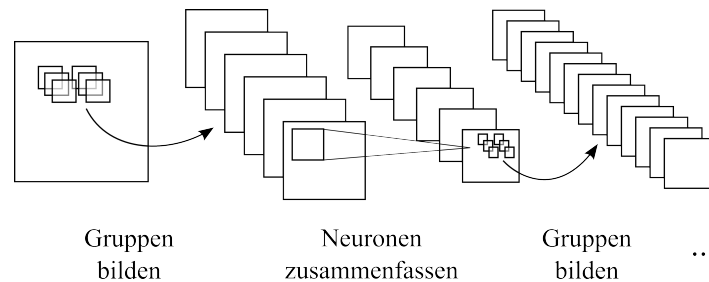
Richtig trainiert, komprimiert das Netz die Daten. So wäre es zum Beispiel möglich Bilder durch das Netz zu schicken, die kleinste Schicht zu speichern und jederzeit über den Decoder wiederherzustellen. Für Bilder, Ton und Videos gibt es einige Kompressionsalgorithmen die sehr fein auf die Charakteristik menschlicher Organe abgestimmt sind und Dinge die wir nicht hören oder Sehen können vernachlässigen. Diese Algorithmen liefern für diesen Anwendungsfall daher meist bessere Ergebnisse. Dennoch demonstriert das Beispiel, dass damit unter Umständen erheblich Speicher und Verarbeitungsleistung eingespart werden kann werden kann. Oft entsprechen die Daten der Dekodierung sogar mehr dem gewünschten Ergebnis als zuvor. Ein Netz das darauf trainiert ist Buchstaben zu lesen, könnte man zum Beispiel im inneren so klein machen, dass es lediglich die einzelnen Buchstaben ohne Manipulationen speichern kann. Beim wiederherstellen würde es dann den Buchstaben erzeugen der seinem Muster am besten entspricht.

Deep Autoencoder liefern sehr vergleichbare Ergebnisse und werden daher gerne als Benchmark für Deep Learning-Algorithmen verwendet. Deep Autoencoders tendieren zu Unteranpassung (eng. underfitting), was auf die geringe Anzahl der Neuronen und damit der geringen Speichermöglichkeit zurückzuführen ist. Eine einfache und effektive Variante ein solches Netz zu trainieren ist es, jeweils zwei Layer separat als restricted Boltzmann Maschine anzusehen.

Lernt man solche Algorithmen mit Bildern und visualisiert die entstandenen Gewichte, so erkennt man, dass solche Algorithmen in erster Instanz meist Kanten und Farbintensitäten finden. In der zweiten Schicht findet man einfache Kombinationen dieser Elemente und ab der dritten Schicht werden meist schon sehr brauchbare Neuronen zur vollständigen Objekterkennung ausgebildet.

### 3.4 Faltungscodierte neurale Netze

Faltungscodierte neuronale Netze (eng. Convolutional Neural Networks) sind neuronale Netze, die zur Reduktion der Gewichte in einzelne Teile zerteilt sind. Bei den bisher angeführten Netzen wurden jeweils alle Neuronen einer Schicht mit jedem Neuron der nächsten Schicht verbunden. Bei faltungscodierten Netzen geht man davon aus, dass weit voneinander entfernte Neuro-



**Abbildung 3.6:** Faltungscodiertes neuronales Netz

nen kaum etwas miteinander zu tun haben, während zwischen nahe aneinander liegende Neuronen Zusammenhänge bestehen. Es werden daher Gruppen von aneinanderliegenden Neuronen gebildet. Diese Gruppen sind untereinander in die nächste Schicht verknüpft. Wie am Beispiel eines Bildes gut verständlich, wird es vorkommen, dass ein Merkmal nicht genau in eine dieser Gruppen passt. Es werden daher viele überlappende Gruppen gebildet.

Nach der Gruppenbildung gibt es eine Funktion, die die Ergebnisse aneinander liegender Werte für die nächste Schicht zusammenfasst. Dies geschieht meist über eine einfache mathematischen oder statistische Funktion und hängt von der Implementierung ab. Diese Funktion könnte zum Beispiel die Kanten in mehrere Richtungen zusammenfassen um die Rotation eines Bildes ignorieren.

Ein faltungscodiertes neuronales Netz wiederholt genau diesen zwei Komponenten, die überlappende Gruppierung von Neuronen so wie das zusammenfassen von Neuronen auf jeder Schicht. Abbildung 3.6 zeigt ein solches Netz.

Dieser Typ von Netzwerk ist bereits 1980 [?] bekannt, wurde in den folgenden Jahrzehnten optimiert und erlebte einen Aufschwung durch die parallelisierte Berechnung auf einer GPU [?]. Nach weiteren Optimierungen in den vergangenen Jahren wurde ein solches Netzwerk unter anderem für ein Projekt von Google [?] eingesetzt, das nicht zuletzt das so genannte face-neuron ausprägte.

### 3.5 Sparse coding

Sparse coding ist ein Algorithmus zum Trainieren eines neuronalen Netzes mit dem Ansatz möglichst aussagekräftige Merkmale zu finden. Anders als beim Deep Autoencoder wird hier nicht nur berücksichtigt wie gut das Ergebnis wiederhergestellt werden konnte, sondern auch wie eindeutig das Ergebnis war. Ziel ist es an möglichst vielen Stellen im Ausgangsvektor 0er zu haben

und dennoch sehr gute Ergebnisse zu finden. Der Fehler setzt sich aus den beiden Faktoren zusammen und wie folgt berechnet:

$$\min \frac{1}{T} \sum_{t=1}^T \min \frac{1}{2} |x^{(t)} - D * h^{(t)}| + \lambda |h_1^{(t)}|$$

$x^{(t)}$  ... Eingangswert

$h^{(t)}$  ... Ausgangswert

$D$  ... Anzahl der Ausgänge

$\lambda$  ... Steuert wie stark sich die Wichtigkeit 0er im Ausgangsvektor auswirken

Die zwei Bedingungen widersprechen sich teilweise und können über  $\lambda$  balanciert werden. Die Berechnung der Gewichte für die versteckten Schichten ist beim Sparse Coding wesentlich aufwendiger als bei Deep Autoencodern, da es sich hier um ein Optimierungsproblem und nicht nur um eine lineare Funktion handelt.

## Kapitel 4

# Anwendungen

In den vergangen Jahren haben sich Unternehmen zunehmend an vielschichtigen neuronalen Netzen interessiert. Wie in Kapitel ?? beschrieben, interessieren sich auch zunehmend Firmen wie Google, Apple und Microsoft. Sie stellten renommierte Wissenschaftler ein und kaufen Unternehmen die in diesem Bereich tätig sind.

### 4.0.1 Objekterkennung in Bildern

- 10 Millionen Trainingsbilder mit je 200x200 Pixel
- Autoencoder mit sparse coding
- 9 versteckte Schichten
- 1 Milliarde Verbindungen
- Training mit asynchronem stochastischem Gradientenverfahren
- Auf auf 1000 Maschinen mit jeweils 16 Rechenkernen drei Tage lang trainiert

Bei einem Projekt [?] 2012 trainierte Google das bisher größte Netz dieser Art mit 10 Millionen Bilder aus dem Internet. Bekannt wurde das Projekt vor allem durch die Ausprägung des in Abbildung 4.1 zu sehenden Gesichts-, Katzen- und Körper-Neurons. Ein paar Eckdaten zum Netz und dem Training:

Es wurde mit relativ kleinen Bildern, dafür aber mit sehr vielen, unbeaufsichtigt trainiert. Ein späterer Benchmark auf ImageNET, eine Datenbank mit Kategorisierten Bildern, ergab einer Trefferquote von 15.8%. Das entspricht einer Steigerung der bisherigen Bestleistung um 70%.



**Abbildung 4.1:** Markante Ausprägungen durch das lernen von Bildern aus dem Internet

Aus technischer Perspektive ist das Netz und damit die verwendete Technik weniger herausragend, als die Menge der Trainingsdaten und die damit verbunden Rechenleistung.

## Kapitel 5

# Zusammenfassung

In der Arbeit wurden die wesentlichen Konzepte und Algorithmen rund um das Deep Learning aufgearbeitet. Deep Learning-Algorithmen basieren im Wesentlichen auf sehr ähnlichen Prinzipien und wurden mit der Entwicklung der Technik an diese angepasst. So sind moderne Algorithmen, wie der zum Beispiel der DistBelief-Algorithmus, im wesentlichen darauf ausgelegt eine parallelisierte Berechnung zu ermöglichen und somit die Hardware bestmöglich auszunutzen.

Aktuelle Netze sind meist sehr vielschichtig und werden mit unbeaufsichtigtem Lernen vorab trainiert. Aufwendiges Feature-Engineering durch Expertenwissen kann meist schon entfallen und wird automatisch durch das unbeaufsichtigte Lernen gemacht.

Bereits heute werden aktiv Netze zur Erkennung von Verkehrsschildern, Hausnummern und Gesichtern eingesetzt. Im Grundsatz zeigen momentan Benchmarks dass die Ergebnisse von neuronalen Netzen im wesentlichen von der Menge der Trainingsdaten und der Größe des Netzes abhängen. Mit zunehmender Vernetzung und dem Internet existieren in vielen Bereichen Daten in sehr großen Mengen. Mit dem Fortschritt der Technik und zunehmendem Interesse an diesem Forschungsgebiet, sind in naher Zukunft noch viele Interessante Anwendungen zu erwarten.

Persönlich finde ich die Entwicklungen in Bereich der Chipherstellung besonders interessant. Die Geschichte hat bereits mehrmals gezeigt, dass die Berechnung auf CPUs über Grafikkarten bis hin zu dedizierten Chips enorme Leistungssprünge bringt. So dürfte auch die Entwicklung von Chips wie sie zum Beispiel Qualcomm momentan entwickelt einige Neuerungen bringen.

Die Leistungsfähigkeit neuronaler Netze ist schwierig zu verstehen und so entsteht immer wieder die etwas philosophische Frage, in wie weit ein solches Netz an die Leistungsfähigkeit des Menschen kommt oder ihn gar eines Tages Konkurrenz macht. Gut Trainiert schlagen diese Netze Menschen schon heute



in der ein oder anderen Aufgabe. An dieser Stelle ist zu erwähnen, dass bereits um 1940 die *Zuse* Maschinen von *Konrad Zuse*, die ersten Rechner der Welt, den Menschen in vielen Rechenaufgaben geschlagen haben. Bis neuronale Netze so groß werden, dass sie sich ein Bild von der Natur machen können, dürfte noch eine ganze Zeit vergehen.

# Literaturverzeichnis

- 1 **Blank u. a. 2008** BLANK, Michael ; BOPP, Thomas ; HAMPEL, Thorsten ; SCHULTE, Jonas: Social Tagging = Soziale Suche? In: GAISER, Birgit (Hrsg.) ; HAMPEL, Thorsten (Hrsg.) ; PANKE, Stefanie (Hrsg.): *Good Tags - Bad Tags: Social Tagging in der Wissensorganisation*, Waxmann, 2008, S. 85–96