

Table of Contents

Table of Contents	1
3-D Datenrekonstruktion mit generative Adversarial Networks	3
1 <i>Andreas Wiegand</i>	
1 Einleitung	3
1.1 Objectives	4
1.2 Outline	5
2 Grundlagen und ähnliche Arbeiten	6
2.1 künstliche Neuronale Netzwerke	6
2.1.1 Backpropagation Algorithmus	8
2.1.2 momentum	9
2.1.3 Batch Normalisation	10
2.1.4 Nicht Lineare Gleichungssysteme	16
2.1.5 short title	19
2.2 Datenformat	20
2.2.1 Woher stammen die Daten	21
2.2.2 Objekterkennung	22
2.3 Deep Learning und Informationstheorie	26
2.4 Convolution Neural Networks	27
2.5 Generative Modelle	30
2.5.1 autoencoder	30
2.5.2 Zielfunktion Autoencoder	
.....	30
2.6 Generative Adversarial Network	32
2.6.1 Probleme mit Generative Adversarial Networks	36
2.6.2 Lösungsansätze für Generative Adversarial Networks	
Probleme	37
2.6.3 Conditional Adversarial Networks	38
2.7 Conditional-GAN	39
2.8 3D-GAN	40
3 Methoden	40
3.1 Aufbau	40
3.1.1 Datensatz 1. Versuchsaufbau	40
3.1.2 Datensatz 2. Versuchsaufbau	41
3.1.3 Trainingsaufbau - GAN	41
3.2 Ergebnis	45
4 Evaluation und Ergebnisse	45
5 Zusammenfassung und Diskussion	45
Abbildungsverzeichnis	48

Tabellenverzeichnis	49
----------------------------------	----

Literaturverzeichnis	50
-----------------------------------	----

KDE K Desktop Environment

SQL Structured Query Language

Bash Bourne-again shell

JDK Java Development Kit

VM Virtuelle Maschine

CGAN Conditional Adversarial Networks

PIX2PIX Image-to-Image GAN

3-D Datenrekonstruktion mit generative Adversarial Networks

Andreas Wiegand

Master These in künstlicher Intelligenz

Zusammenfassung. Generative Adversarial Network(GAN) ist ein künstliches neuronales Netzwerk(ANN) aus dem Bereich der generativen Modelle. Die Aufgabe des GANs ist es, die Wahrscheinlichkeitsverteilung von Trainingsdaten zu erlernen und dadurch anschließend neue Samples aus dieser Wahrscheinlichkeitsverteilung zu generieren. Man erhofft sich, den hohen Datenaufwand beim trainieren von ANN zu umgehen und durch GANs neue Trainingsdaten zu generieren. Ziel dieser Arbeit ist es das Konzept von GANs auf 3D-Scans von Tabakblättern anzuwenden um die Wahrscheinlichkeitsverteilung von 3D-Daten zu erlernen. Ein weiteres Ziel ist es heraus zu finden ob mit Hilfe von GANs es möglich ist 3D-Daten welche beispielsweise beim Scannen unvollständig sind ergänzen zu können. Im Folgenden wird auf die Theorie von GANs eingegangen, wie deren Aufbau und deren zugrunde liegendes mathematische Modell. Anschließend werden die Methoden des Experimentes präsentiert sowie die Ergebnisse diskutiert.

1 Einleitung

Um genauere Prognosen über Erntemenge und frühzeitiger Erkennung von Krankheiten zu treffen werden 3D-Scan verfahren eingesetzt welche Pflanzen scannen und damit 3D-Daten liefern welche zur weiterverarbeitung von Machine-Learning Ansätzen benötigt werden. [EVLUT BILD EINBAUEN UND GENAUER AUF SCANVERFAHRENE INGEHEN]. Jedoch sind diese Scanverfahren wie jede Informationsübertragung mit sogenannten rauschen verbunden das heißt die Information vom Empfänger zum Sender wird verändert und behält nicht die ursprünglichen Inhalt. Beispiele dafür waren ein Blatt verdeckt ein anderes Blatt und lässt die Sensoren des Scanner nicht zu das unter trunder liegende Blatt zu erfassen. Um nun diesen Verlust von Daten zu verhindert muss geprüft werden ob die Möglichkeit besteht diese Daten zu reparieren in dem sie ergänzt werden. In dieser Arbeit wird ein Ansatz überprüft welcher es Möglich machen kann dieses Verhalten zu erhalten. Ein Ansatz der dafür Verwendet werden kann ist Deep Learning bei dem man das Deep Learning Modell den unterschied zwischen Reparierten und nicht reparierten Daten erlernt kann es rückschlüsse zeihen und selber Daten reparieren.

Deep learning, however, gained much popularity across many academic disciplines in recent years and has been used in computer vision successfully to

produce state-of-the-art results for various tasks. It is described as the application of multiple processing layers to produce multiple levels of representation. It is therefore capable of learning higher level representations of raw data, that can be used for the intended task, e.g. classification of an image. The most common realization are artificial neural networks (ANN) and especially convolutional neural networks (CNN) for processing data with a grid-like topology, e.g. images. Several publications have shown the effectiveness of CNNs for instance segmentation of plant leaves on images (e.g. Ren and Zemel, 2016). But there is a lack of research in applying deep neural networks to 3D representations of plants.

In den letzten Jahren haben sich im Deep Learning Bereich besonders die discriminativen Modelle hervorgehoben, welche Input Daten wie Bilder, Audio oder Text Daten zu bestimmte Klassen zuordnen. Der Grund für das steigende Interesse liegt in der niedrigen Fehlerrate bei discriminativen Aufgaben, im Vergleich zu anderen Maschine Learning Ansätzen, wie Decision Trees oder Markov Chains (Goodfellow, Bengio, Courville, & Bengio, 2016). Die Modelle lernen eine Funktion welche Input Daten X auf ein Klassen Label Y abbildet. Die Modelle werden dabei von ANN repräsentiert. Man kann auch sagen das Modell lernt die bedingte Wahrscheinlichkeitsverteilung $P(Y|X)$ (Ng & Jordan, 2002). Generative Modelle haben die Aufgabe die Wahrscheinlichkeitsverteilung von Trainingsdaten zu erlernen. Es lernt eine multivariate Verteilung $P(X, Y)$, was dank der Bayes Regel auch zu $P(Y|X)$ umgeformt werden kann und somit kann das Modell auch für discriminativen Aufgaben herangezogen werden. Gleichzeitig können aber neue (x,y) Paare erzeugt werden, was zu dem Ergebnis von neuen Datensätzen führt welche nicht Teil der Trainingssample sind (Ng & Jordan, 2002). In diesem Paper wird speziell auf GAN, aus der Vielzahl von generativen Modellen eingegangen. Diese wurden von Goodfellow (Goodfellow et al., 2014) entwickelt und ebneten den Weg für Variationen, welche auf der Grundidee von GANs aufbauen. 2017 wurden alleine 227 neue Paper zu diesem Thema vorgestellt. Ein Grund weshalb GAN an Popularität gewinnt ist der, dass neuronale Netzwerke mit Zunahme der Trainingsdatenanzahl eine Erhöhung der Performance für die sie Trainiert werden zeigen. Was bedeutet, dass sich mit Zunahme der Datenanzahl die Chance auf eine bessere Performance der Neuronalen Netzwerke ergibt (Halevy, Norvig, & Pereira, 2009). An diesem Punkt erhofft man sich durch GANS mehr qualitative Daten zu erzeugen und somit das Trainingsergebnis zu discriminativen Modelle zu verbessern.

1.1 Objectives

In den letzten Jahren haben sich im Deep Learning Bereich besonders die discriminativen Modelle hervorgehoben, welche Input Daten wie Bilder, Audio oder Text Daten zu bestimmte Klassen zuordnen. Der Grund für das steigende Interesse liegt in der niedrigen Fehlerrate bei discriminativen Aufgaben, im Vergleich zu anderen Maschine Learning Ansätzen, wie Decision Trees oder Markov Chains (Goodfellow et al., 2016). Die Modelle lernen eine Funktion welche Input Daten X auf ein Klassen Label Y abbildet. Die Modelle werden dabei von ANN

repräsentiert. Man kann auch sagen das Model lernt die bedingte Wahrscheinlichkeitsverteilung $P(Y|X)$ (Ng & Jordan, 2002). Generative Modelle haben die Aufgabe die Wahrscheinlichkeitsverteilung von Trainingsdatendaten zu erlernen. Es lernt eine multivariate Verteilung $P(X, Y)$, was dank der Bayes Regel auch zu $P(Y|X)$ umgeformt werden kann und somit kann das Modell auch für discriminativen Aufgaben herangezogen werden. Gleichzeitig können aber neue (x, y) Paare erzeugt werden, was zu dem Ergebnis von neuen Datensätzen führt welche nicht Teil der Trainingssample sind (Ng & Jordan, 2002). In diesem Paper wird speziel auf GAN, aus der Vielzahl von generativen Modellen eingegangen. Diese wurden von Goodfellow (Goodfellow et al., 2014) entwickelt und ebneten den Weg für Variationen, welche auf der Grundidee von GANs aufbauen. 2017 wurden alleine 227 neue Paper zu diesem Thema vorgestellt. Ein Grund weshalb GAN an Popularität gewinnt ist der, dass neuronale Netzwerke mit Zunahme der Trainingsdatenanzahl eine Erhöhung der Performance für die sie Trainiert werden zeigen. Was bedeutet, dass sich mit Zunahme der Datenanzahl die Chance auf eine bessere Performance der Neuronalen Netzwerke ergibt (Halevy et al., 2009). An diesem Punkt erhofft man sich durch GANS mehr qualitative Daten zu erzeugen und somit das Trainingsergebnis zu discriminativen Modelle zu verbessern.

1.2 Outline

Diese Arbeit ist folgender Maßen strukturiert. In Kapitel 2 "Grundlagen und ähnliche Arbeiten" werden theoretische Grundlagen welche für diese Arbeit benötigt wird genauer beleuchtet.

Kapitel 3

Kapitel 4

Kapitel 5 - Gibt eine Zusammenfassung über die Arbeit und wird sie kritisch Diskutieren. Desweiteren wird ein Ausblick erstellt inwiefern das Ergebnis für zukünftige Arbeiten von Relevanz ist.

This thesis is structured as follows Chapter 2 - Methods This chapter provides the experimental design, data set and evaluation metrics used in this thesis. Furthermore the selected approaches that are used for semantic and instance segmentation are introduced. Methods This chapter provides the experimental design, data set and evaluation metrics used in this thesis. Furthermore the selected approaches that are used for semantic and instance segmentation are introduced. Methods This chapter provides the experimental design, data set and evaluation metrics used in this thesis.

Chapter 3 - Methods This chapter provides the experimental design, data set and evaluation metrics used in this thesis. Furthermore the selected approaches that are used for semantic and instance segmentation are introduced.

Chapter 4 - Methods This chapter provides the experimental design, data set and evaluation metrics used in this thesis. Furthermore the selected approaches that are used for semantic and instance segmentation are introduced.

Chapter 5 - Discussion The thesis ends with a discussion of the results and the limitations of this work. In addition, an outlook is given on possible research directions

2 Grundlagen und ähnliche Arbeiten

Im folgenden Kapitel wird auf theoretische Grundlagen eingegangen, welche zum Verständnis für Arbeit benötigt werden. Zunächst werden generative Modelle im Allgemeinen vorgestellt, welche den Grundgedanken der Datengeneration für GANs aufzeigen. Darauf folgend werden Convolutional Neuronale Netzwerke vorgestellt, aus welchen GANs aufgebaut werden können, um mit Bild Daten zu arbeiten.

2.1 künstliche Neuronale Netzwerke

Künstliche Neuronale Netzwerke(ANN) haben das Ziel ein Funktion f^* zu approximieren. Dabei werden Parameter Θ eines Models so angepasst, um die Abbildung von $y = f(x; \Theta)$ zu approximieren. Die Modelle werden auch als feed-forward Neuronale Netzwerke betitelt weil der Informationsfluss des Models von Input zu Output fließt und keine Rekursion von Output zu Input statt findet. ANN können aus mehreren Schichten sogenannten Hidden Layer besteht welche als $f(x) = f^{(2)}(f^{(1)}(x))$ wobei n für die Anzahl der Layer steht und $n \geq 1$ ist. Ein 2-Layer ANN ist dann definiert durch $f^{(n)}$. Es gibt den Input layer welche den Input in das Netzwerk aufnimmt. Im vorherigen Beispiel ist dies $f^{(1)}$ und der letzte layer des Netzwerkes wird Output Layer genannt. Im vorherigen Beispiel wäre das $f^{(2)}$ (Goodfellow et al., 2016).

Jeder Layer besteht aus künstliche Neuronen diese haben ihren Namesgeber von der aus der natur stammende Neuron. Neuronen sind die Bausteine aus denen die Gehirne von Lebewesen, wie Fische, Vögel, Säugetiere zusammen gesetzt sind. Neuronen oder auch Nervenzellen bestehen in eine Zellkern der Zetrum der Zelle um sie herum sind dendriten. Neuronen sind untereinander mit den Axon verbunden. axon haben an den enden Synapen welche die grenze von Axon zur Nervenzelle einen Spalt bilde. Dieser Spalt überwunden indem von der Synapse botenstoffe abgesendet werden welche sich dann an Rezeptor der Zelle anhaften. Diese Übertragung findet statt wenn an der Synapse ein bestimmter Schwellenwert überschritten wurde von elektrischen reizen welche die Zelle abfeuert lässt. Künstliche Neuronen haben diese Schwellenwert durch songenannte Gewichte w_{ij} diese sind auf den Verbindungen zwischen den Neuronen in den unterschiedlichen layern im Netzwerk

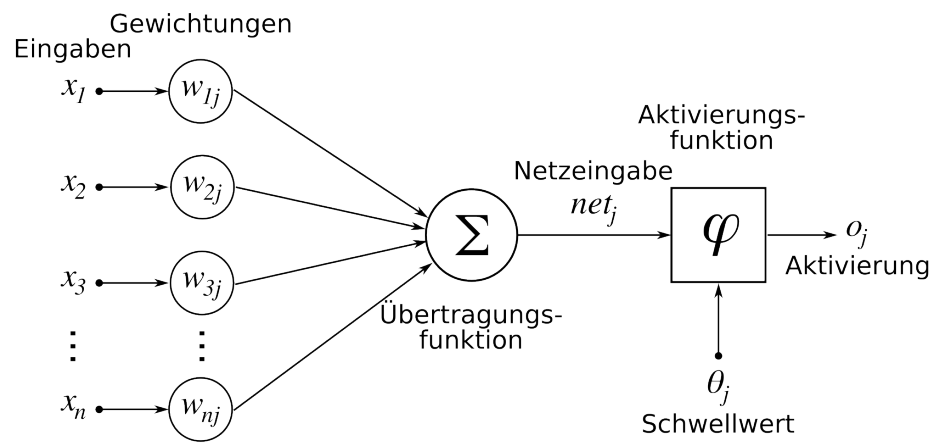


Abb. 1. künstliches Neuron

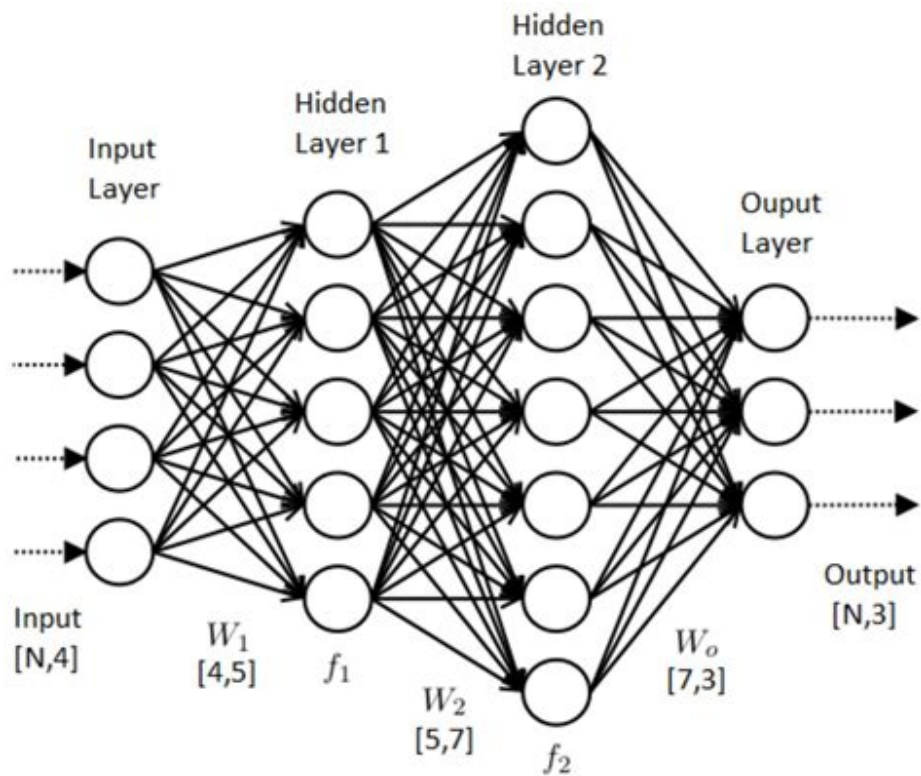


Abb. 2. künstliches neuronales Netzwerk

2.1.1 Backpropagation Algorithmus Um nun ANNs zu trainieren und den gewünschten Output y generieren wird der Backpropagation Algorithmus ergriffen. Dieser wurde von Geoffrey Hinton im Jahre 1989 vorgestellt (Rumelhart, Hinton, & Williams, 1986). Er konnte zeigen dass sein Algorithmus schneller und effizienter auf Neuronale Netze arbeitet als andere vor ihm. Das mathematische zugrundeliegende Konzept ist die partielle Ableitung von $\frac{\partial Z}{\partial w}$, wobei Z die Zielfunktion und w die Gewichte im zu optimierenden Neuronale Netzwerk sind. Für eine Funktion $f(x) = y$ ist die Ableitung definiert als $f'(x)$ oder $\frac{dy}{dx}$ und gibt die Steigung der Funktion an Punkt x an. Dieses Verfahren kann dabei helfen Funktionen zu optimieren. Wenn man weiß wie sehr die Steigung an Punkt x ist kann man x mit einer Änderung von der Ableitung x dahin gehen optimieren (Goodfellow et al., 2016).

Wenn nun $f'(x) = 0$ haben wir keinerlei Information über die Steigung erreicht. Dies ist aber kein Garant dafür dass f ein Optimum erreicht hat. Es könnten wie in Abbildung unten dargestellt. Ein lokales Minimum sein. Das heißt dass wir nur an diesen Punkt ein Minimum erreicht haben aber im Funktionsverlauf ein noch niedrigeres Minimum vorhanden ist. Oder einen Sattelpunkt welche ein Übergang zu einem Anstieg der Funktion bildet. Ist nun die Funktion f definiert als $f: \mathbb{R}^n \rightarrow \mathbb{R}$ und hat als Input mehrere Variablen. Die partielle Ableitung $\frac{\partial f(x)}{\partial x_i}$ zeigt an wie sehr sich $f(x)$ ändert wenn wir x_i ändern. Der Gradient $\nabla_x f(x)$ ist ein Vektor welche alle partiellen Ableitungen von f enthält. Nun kann man f optimieren indem man in die Richtung des Gradienten geht welche negativ ist dieses Verfahren wird Gradient Descent genannt (Goodfellow et al., 2016).

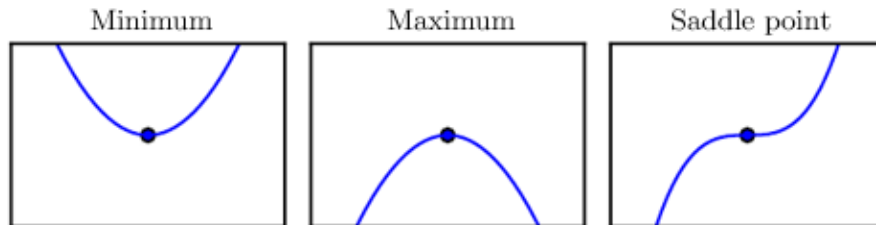


Abb. 3. Minimum Maximum Saddle Point

Algorithmen die für Deep Learning eingesetzt werden, beinhalten eine Art von Optimierung, ohne diese ist eine Umsetzung des Lernprozesses nahezu unmöglich. Diese Minimierungsfunktionen oder auch cost function (welche in vielen Publikationen unterschiedlich bezeichnet wird „loss“- oder „error function“) wollen immer dasselbe Ziel: eine gewisse Target Funktion ermitteln, welche für einen Input den gewünschten Output ausgibt. Das Ziel ist in anderen Worten eine

Menge an Gewichten und Biases zu finden, für welche, die quadratischen Kosten ($C(w,b)$) so gering wie möglich sind. Um dies zu erreichen, wird der Gradient Decient Algorithmus eingesetzt (Nielsen, 2017). Der Grund des Einsatzes dieses Algorithmus ist derer, dass zwar versucht werden könnte die Anzahl der richtig klassifizierten Bilder direkt zu erhöhen. Aber das Problem ist, den Performancegewinn bei Veränderungen der Gewichte festzustellen. Da meisten kleine Veränderungen an den Gewichten und Baises führen zu keinerlei Veränderung bei der Klassifizierung. Somit liegt eine gewisse Schwierigkeit, bei der Ermittlung der richtigen Anpassung dieser Werte. Zuerst muss die quadratische cost function ($C(w,b)$) minimiert werden, bevor sich die Maximierung der Bestimmungsgenauigkeit des Netzwerkes als Ziel gesetzt kann (Nielsen, 2017).

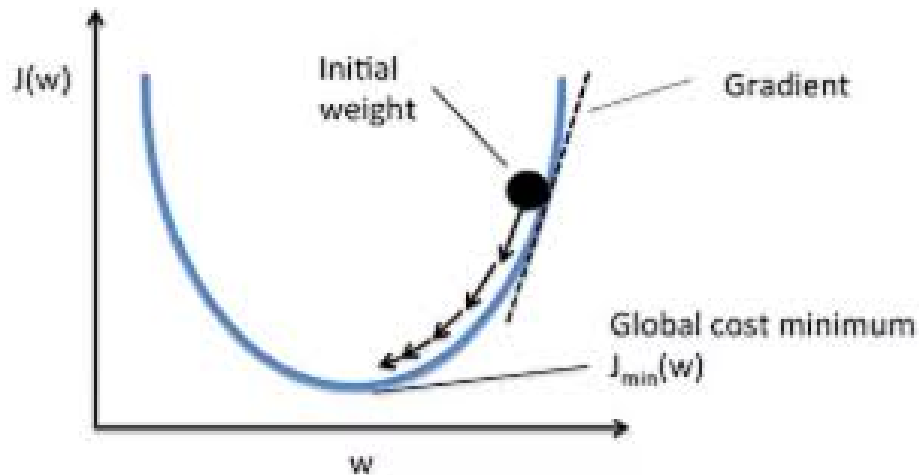


Abb. 4. LossFunktion

Backpropagation Algorithmus wird das Verfahren genannt mit den ANN trainiert werden. Dabei wird der Gradient der Ziel Funktion bestimmt.

2.1.2 momentum Momentum hat das Ziel das Gradientenverfahren zu beschleunigen um eine effizienter Konfergierung der Zielfunktion herbei zu führen und ein schnelleres Lernen erzielt.

$$v_{t+1} = \mu v_t - \epsilon \nabla f(\theta_t)$$

$$\theta_{t+1} = \theta_t + v_{t+1}$$

wobei $\epsilon > 0$ die Lernrate ist und $\mu \in [0,1]$ das Momentum ist und $\nabla f(\theta_t)$ der Gradient von θ_t ist. Je größer das Momentum desto schneller Bewegt sich der

Gradient abwärts. Da am Anfang einer Lernphase der Gradient üblicherweise recht hoch ist empfiehlt sich zunächst mit einem niedrigen Momentum zu arbeiten da sonst die Gefahr besteht über das globale Optimum hinauszuschießen. Wenn nun das Training stagniert welches auf Gründe der Aufbau der Loss-Funktion zurückzuführen ist das zur Nähe des globalen Optimums flache Täler entstehen welche das Trainings verlangsamen. und es zu keiner Verbesserung kommt kann man durch Momentum erzwingen größere Gradienten Sprünge einzugehen und sich somit schneller zu einem globalen Optimum zu bewegen oder aber aus einem lokalen Optimum hinaus Richtung eines globalen Optimums (Sutskever, Martens, Dahl, & Hinton, 2013).

Here's a popular story about momentum gradient descent is a man walking down a hill. He follows the steepest path downwards; his progress is slow, but steady. Momentum is a heavy ball rolling down the same hill. The added inertia acts both as a smoother and an accelerator, dampening oscillations and causing us to barrel through narrow valleys, small humps and local minima. This standard story isn't wrong, but it fails to explain many important behaviors of momentum. In fact, momentum can be understood far more precisely if we study it on the right model. One nice model is the convex quadratic. This model is rich enough to reproduce momentum's local dynamics in real problems, and yet simple enough to be understood in closed form. This balance gives us powerful traction for understanding this algorithm (Sutskever et al., 2013).

For a step-size small enough, gradient descent makes a monotonic improvement at every iteration. It always converges, albeit to a local minimum. And under a few weak curvature conditions it can even get there at an exponential rate.

<https://distill.pub/2017/momentum/>

2.1.3 Batch Normalisation Wie in Kapitel backpropagation Algorithmen gezeigt wurde wird durch den stochastischen Gradienten Vorteile gegenüber dem normalen Gradienten Verfahren ergeben. Durch das der Input in jedem Layer abhängig von den vorherigen Layer abhängig. Dadurch können Änderungen in vorherigen Layer große Auswirkungen in tieferen Layer im Netzwerk haben. Dadurch resultiert dann das in Trainingsläufen die Verteilung von jedem Layers Input sich während des Trainings verändert die verlangsamt das Training erheblich (Ioffe & Szegedy, 2015). Dieses Problem wird als internal covariate shift definiert. Wenn sich die Input Verteilung von einem lernenden System verändert macht es einen covariate shift durch (Ioffe & Szegedy, 2015). Um dies zu verhindern zeigten Sergey Ioffe und Christian Szegedy (Ioffe & Szegedy, 2015) eine Methode welche Batchnormalization genannt wird. Je mehr Layer das Netzwerk hat desto stärker wird der internal covariate shift. Batch normalization besteht aus zwei Algorithmen. Algorithmus 1 verändert den eigentlichen Input von Layer

n zu einen normalisierten Input y und Algorithmus 2 verändert das eigentliche Training eine batch normalisierten Netzwerkes.

Algorithm 1: Batch Normalisierung angewand auf x über Input bei Mini-Batch

```

1 Input: Werte von x über einen Mini-Batch  $B = \{x_1, \dots, x_n\}$ 
2  $\mu_B \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$ 
3  $\sigma_B \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2$ 
4  $\hat{x}_{iB} \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B + \epsilon}}$ 
5  $y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma;\beta}(x_i)$ 
6 Output:  $\{y_i = BN_{\gamma;\beta}(x_i)\}$ 

```

In Schritt 2 des Algorithmus 1 wird der Erwartungswert für alle Input von Mini-Batch B berechnet. In Schritt 3 die Varianze. In Schritt 3 wird nun der der normalisierte x_i berechnet welche dann mit dem β und γ multipliziert werden. Diese Werte sind neue gewichte im Neuronalen Netzwerk welche während des Trainingsporcesses angepasst werden. ϵ in der Gleichung in Zeile 4 ist nur dafür da damit nicht durch 0 geteilt werden kann. In Zeile 8 - 11 werden die Inferenz schritte beschrieben bei den der Minibatch des Trainings ersetzt wird.

Algorithm 2: Training mit Batch-Normalisierungs Netzwerk

```

1 Input: Netzwerk N mit trainierbaren Parameter  $\theta; \{x^{(k)}\}_{k=1} \text{ bis}$ 
2 Ntr BN  $\leftarrow$  N
3 a
4 a
5 a
6 a
7 a
8 a
9 a
10 a
11 a
12 a

```

Schritt 1 bis 5 des Algorithmus 2 baut eigentlich nur das neuronales Netzwerk durch die transformationen aus Algorithmus 1 um. In Schritt 6 und 7 geht es darum die Parameter γ und β zu trainieren. Dieses passiert mit den üblichen Backpropagation Algorithmus wären des Allgemeinen Training des Netzwerkes.

Je mehr Layer das Netzwerk hat desto stärker wird der internal covariate shift

Neuronale Netzwerke sind universelle Funktions approximierer (Hornik, 1991). Es sei erförzuheben das zwar Feedforward Neuronale Netzwerke mit hidden Layer unisversel sind. Aber nicht jede aktivierungsfunktion ist für alle probleme gleich effizient. (Hornik, 1991). In Kapitel BLABLA wurde gezeigt das Neuro-

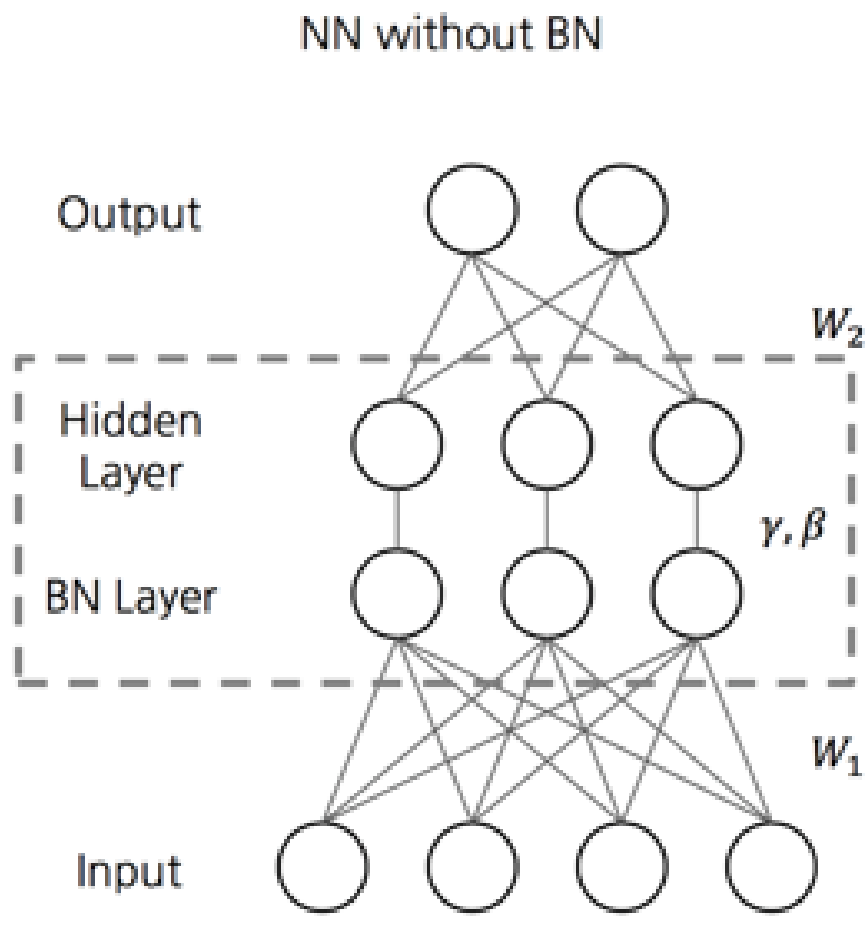


Abb. 5. Neuronales Netzwerk mit Batch Normalization Layer

nale Netzwerke durch Matrizen dargestellt werden können und das durch simple Matrixmultiplikation der Output von Layern berechnet werden können. Die Aktivierungsfunktion sorgen dafür das das ANN auch nicht lineare Funktionen erlernen kann. Also das unser Input X und unser Output Y immer proportional als $Y = X * k$. Wobei k eine Konstante ist. Wie wir in Kapitel BLABLA gesehen haben möchten wir das unser Input linear trennbar ist damit wir unsere Input in Klassen einteilen können. Würde X nun aber nicht linear trennbar sein wie Beispielsweise in Abbildung UNTEN gezeigt Könnten wir durch die lineare transformation des Inputs keine Trennung von X erreichen. <http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/> Mit den einführen von nicht linearen Funktionen können der Raum der durch die Layer dargestellt wird gedreht gewendet und gezogen werden. Aber es scheidet, zerbricht oder faltet in.

Ein Neuron hat einen Input x_n sowie ein Gewicht w_{ij} und einen Bias b. der Output eines Neurons ist definiert durch $y = x_n * w_{ij} + b$

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

$$\zeta(x) = \log(1 + \exp(x))$$

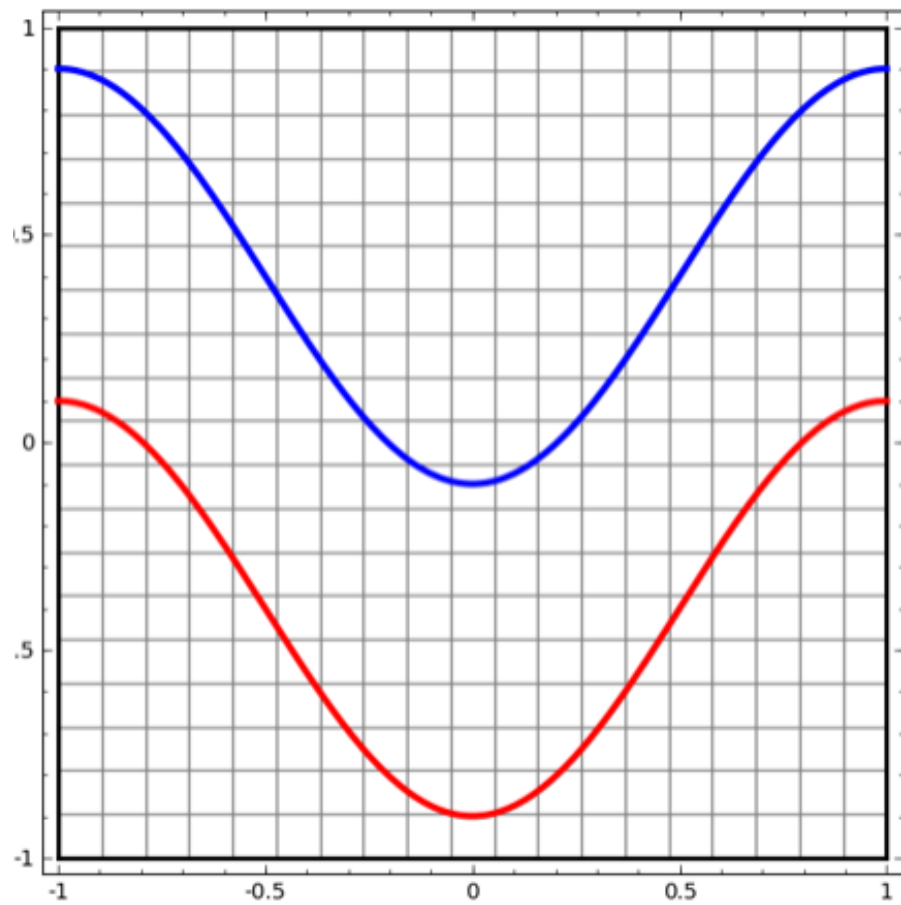


Abb. 6. Nicht trennbar

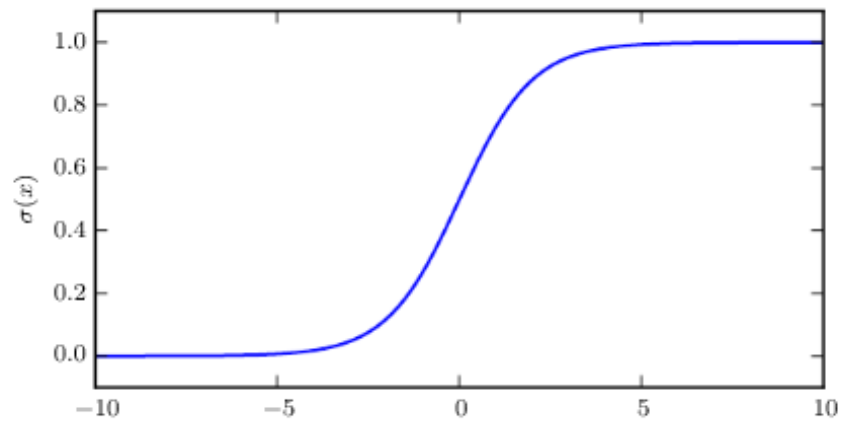


Abb. 7. Sigmoid Funktion

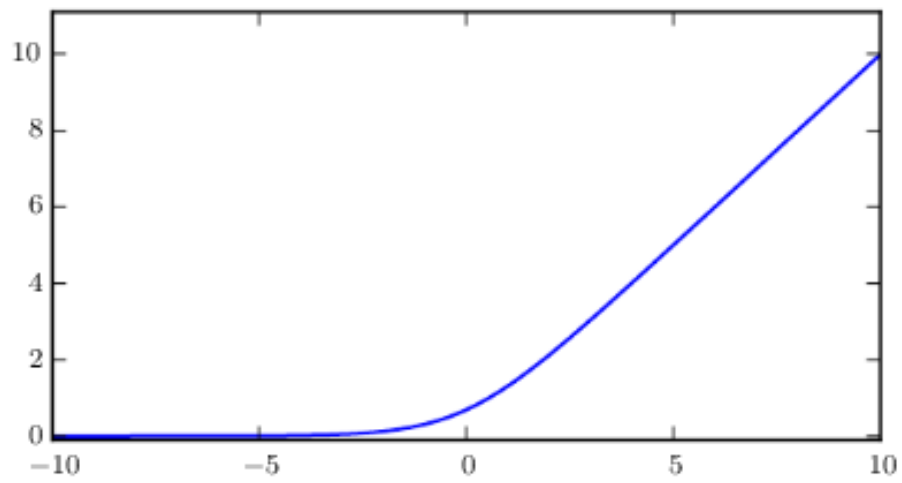


Abb. 8. Softmax

2.1.4 Nicht Lineare Gleichungssysteme Neuronale Netzwerke sind universelle Funktionsapproximierer (Hornik, 1991). Es sei erwähnt, dass zwar Feedforward Neuronale Netzwerke mit hidden Layer universell sind. Aber nicht jede Aktivierungsfunktion ist für alle Probleme gleich effizient. (Hornik, 1991). In Kapitel BLABLA wurde gezeigt, dass Neuronale Netzwerke durch Matrizen dargestellt werden können und das durch simple Matrixmultiplikation der Output von Layern berechnet werden können. Die Aktivierungsfunktionen sorgen dafür, dass das ANN auch nicht lineare Funktionen erlernen kann. Also, dass unser Input X und unser Output Y immer proportional als $Y = X \cdot k$. Wobei k eine Konstante ist. Wie wir in Kapitel BLABLA gesehen haben, möchten wir, dass unser Input linear trennbar ist, damit wir unsere Input in Klassen einteilen können. Würde X nun aber nicht linear trennbar sein wie beispielsweise in Abbildung UNTEN gezeigt, könnten wir durch die lineare Transformation des Inputs keine Trennung von X erreichen. <http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/> Mit dem Einführen von nicht linearen Funktionen können der Raum, der durch die Layer dargestellt wird, gedreht, gewendet und gezogen werden. Aber es scheitert, zerbricht oder faltet in.

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

$$\zeta(x) = \log(1 + \exp(x))$$

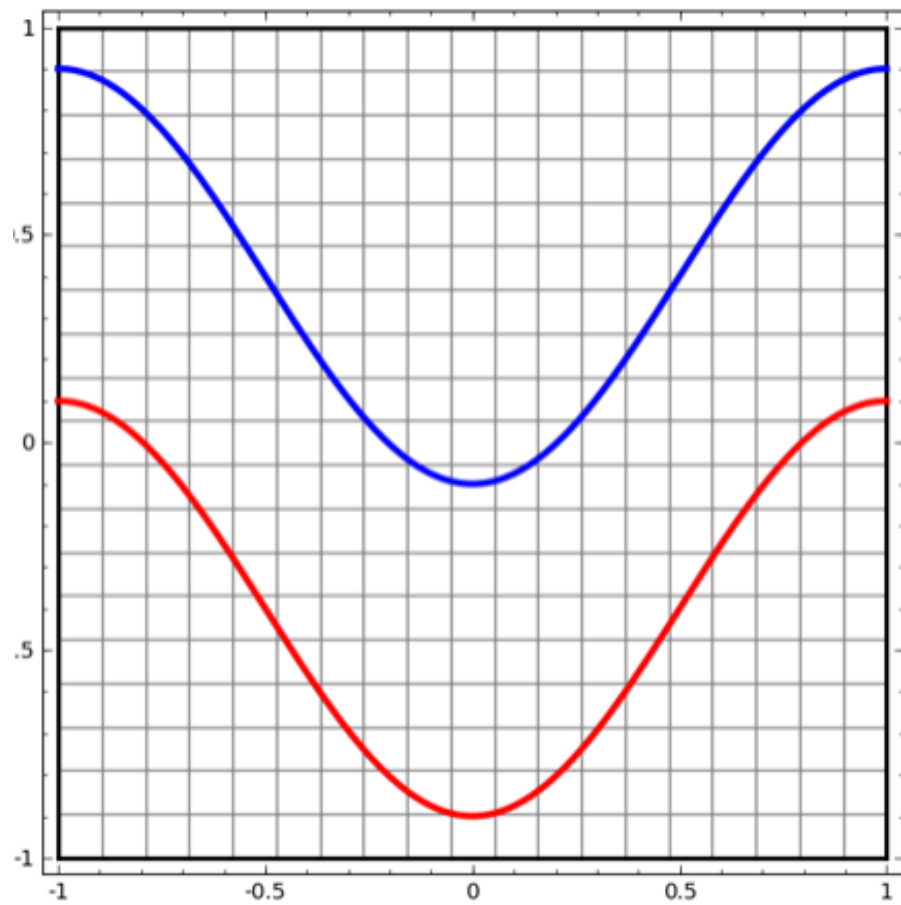


Abb. 9. Nicht trennbar

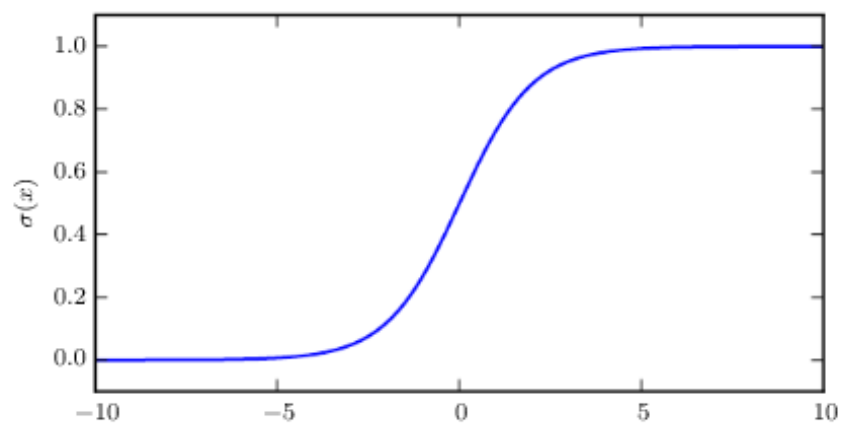


Abb. 10. Sigmoid Funktion

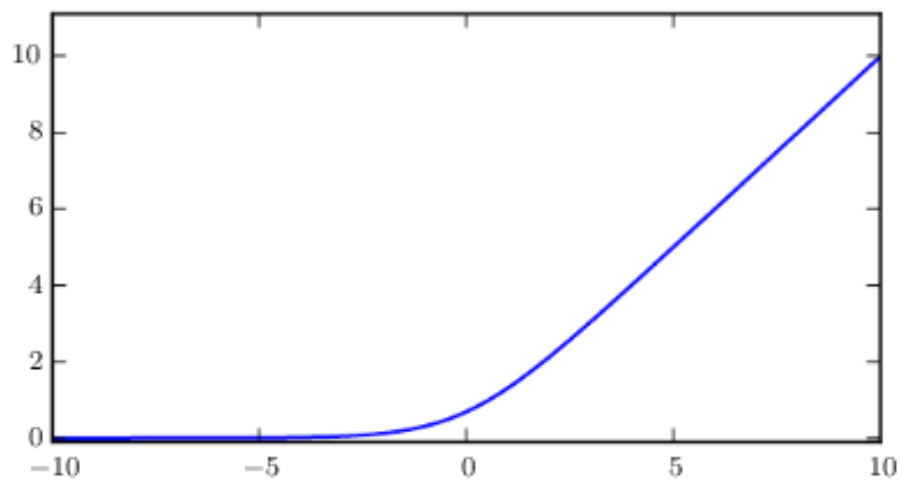


Abb. 11. Softmax

2.1.5 Ziel-Funktion Die Ziel-Funktion muss Differenzierbar sein. Das heißt unseren Funktion $f: D \rightarrow \mathbb{R}$ ist differenzierbar an der Stelle x_0 an der Stelle. Wenn nun $f': x \rightarrow f(x)$ an jeden Punkt x^n ableitbar ist, ist f differenzierbar. Die Aufgabe der Ziel-Funktion ist es zu messen wie gut unsere Model Θ $f^*(x)$ approximiert. Man verwendet auch gerne den Begriff Kosten, also wieviel Kosten unser Modell erzeugt. Daher wird die Ziel-Funktion auch Kost-Funktion genannt. Die Wahl welche Funktion gewählt wird ergibt sich aus der Aufgabe des ANN soll es zur Regression Diese kann für Klassifikation und Regression Aufgaben hergenommen werden. Bei Regression soll eine kontinuierlicher Variablen von Θ als Output generiert werden wohin gegen bei Klassifikationsproblemen der Output Klassen-Labels darstellen. Es gibt verschiedene Ziel Functionen im folgenden wird die Categorical Cross Entropy Loss Function vorgestellt (Golik, Doetsch, & Ney, 2013). Diese wird verwendet wenn man das Ziel hat ein Klassifikationsproblem zu Lösen und $n \geq 1$ Klassen hat. Diese ist definiert als:

$$\hat{q}(c | x) = \arg \min_{q(c|x)} \left\{ - \sum_n \log q(c_n | x_n) \right\}$$

Wobei $x_n: n=1, \dots, N$ die Trainingsdaten sind ist und $c_n: n=1, \dots, N$ die mögliche Klassen.

2.2 Datenformat

Voxel (zusammengesetzt aus dem englischen volume vox und el von elements[1]) bezeichnet in der Computergrafik einen Gitterpunkt („Bild“punkt, Datenelement) in einem dreidimensionalen Gitter. Dies entspricht einem Pixel in einem 2D-Bild, einer Rastergrafik. Wie bei Pixeln wird bei Voxeln üblicherweise die Position nicht explizit gespeichert, sondern implizit aus der Position zu anderen Voxeln hergeleitet. Im Gegensatz dazu werden bei Punkten oder Polygonen die Positionen der Eckkoordinaten gespeichert. Eine direkte Konsequenz dieses Unterschiedes ist, dass man mit Polygonen eine 3D-Struktur mit viel leerem oder homogen gefülltem Raum effizient darstellen kann. Voxel hingegen sind gut bei der Repräsentation eines äquidistant gesampelten Raums, der nicht homogen gefüllt ist. wikipedia kopie



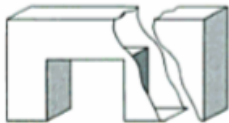

Dimension der Elemente	Element	Modelltyp
0D	<p>Punkt</p> 	Eckenmodell
1D	<p>Linie</p> 	Kantenmodell
2D	<p>Fläche</p> 	Flächenmodell
3D	<p>Volumen</p> 	Volumenmodell (Körpermodell)

Abbildung 1: CAD-Elemente und Modelltypen (Friedrich, 2012, S. 27)

Abb. 12. künstliches Neuron

2.2.1 Woher stammen die Daten Die Daten stammen von TERRA-REF Feld Scanner von der University von Arizona Maricopa Agricultural Center and USD Arid Land Research Station in Maricopa. Es ist der größte Feld crop scanner



Abb. 13. künstliches Neuron

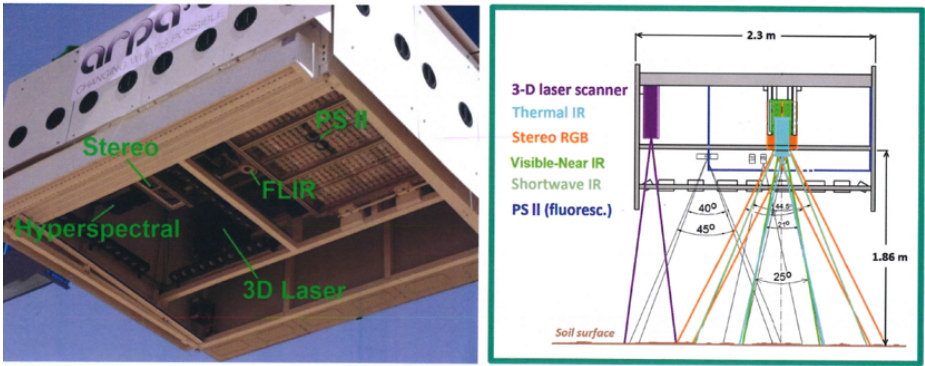


Abb. 14. künstliches Neuron

Sensor	Senor Name	Beschreibung	Field of View	Pixel size(mm)	Beispiel von
3D laser scanner	3d Frauenhofer	a	a	d	
Termal IR	FLIR	a	a	d	
Stereo-RGB	Prosilica GT3300C	a	a	d	
Visual-Near IR(VNIR)	Headwall Inspector	a	a	d	
Shortwave IR(SWIR)	Headwall Inspector	a	a	d	
PSII	PSII Camera LemmnaTec	a	a	d	

Der 3D Laser Scanner erzeugt sogenannte 3D Punktwolken. Dabei werden die Objekte durch den Scanner erfasst und eine 3D Repräsentation welche durch Punkte in einen 3 Dimensionalen Koordinaten System erfasst werden können dargestellt. Dabei wird ein Laser über das zu scannende Objekt gefahren durch geflektierung des Laserstrahls auf der Oberfläche des Objektes können x,y,z Koordinaten des jeweiligen Punktes auf den Objekt bestimmt werden.

Die Objekte werden PLY - Polygon File Format gespeichert.

2.2.2 Objekterkennung Das zweite Modell beruht auf dem Konzept, der Objekterkennung auf Bildern aus den Be-reich Computer Vision. Dabei bekommt das Neuronale Netz (NN) als Input ein Bild und gibt als Output die Bounding-Box Koordinaten und die Klassenbezeichnung des Objekts mit den jeweiligen Prozentangaben zurück.

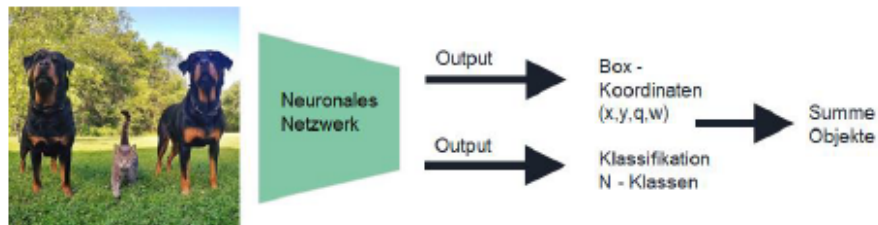


Abb. 15. künstliches Neuron

Im Abbildung 5-1 wird dieser Vorgang Illustriert. Der Output des NN wird in Abbildung 5-2 gezeigt. Bounding-Box Koordinaten sind vier Punkte, welche durch x und y Koordinaten auf dem Bild definiert werden. Diese Punkte kennzeichnen dann den Rahmen, welcher um das zu erkennende Objekt gelegt wird (Ross Girshick, 2014).

Nachdem die Objekte auf dem Bild erkannt worden sind, können die einzelnen Objektklassen auf den Bildern aufsummiert werden. In unseren Versuchsaufbau wird ein Faster Re-gion-based Convolutional Neural Network (Faster R-CNN) eingesetzt. Die Funktionsweise des Faster R-CNN wird in Punkt „5.1.4 Faster R-CNN“ genauer erklärt. In „5.1 Stand der Forschung Objekterkennung mit NN“ wird auf die Entwicklung hin zum Faster R-CNN eingegangen. Der genauere Projektversuchsaufbau wird unter „5.2 Versuchsaufbau“ erläutert.

Um einen genaueren Einblick und das Verständnis für die Objekterkennung mit NN zu schaffen wird zunächst auf die Entwicklung in diesen Bereich eingegangen. Im Allgemeinen arbeiten NN in der Objekterkennung mit den gleichen Layern wie in „2.5 Modellierung neuronaler Netzwerke (Dennis Groß)“ beschrieben. Die Convolutional Layer identifizieren Features welche die Objektklassen unterscheiden. Der Pooling Layer hilft bei der Dimension Reduzierung und dadurch eine Aggregation von Features. Jegliche die Zusammensetzung der Layer

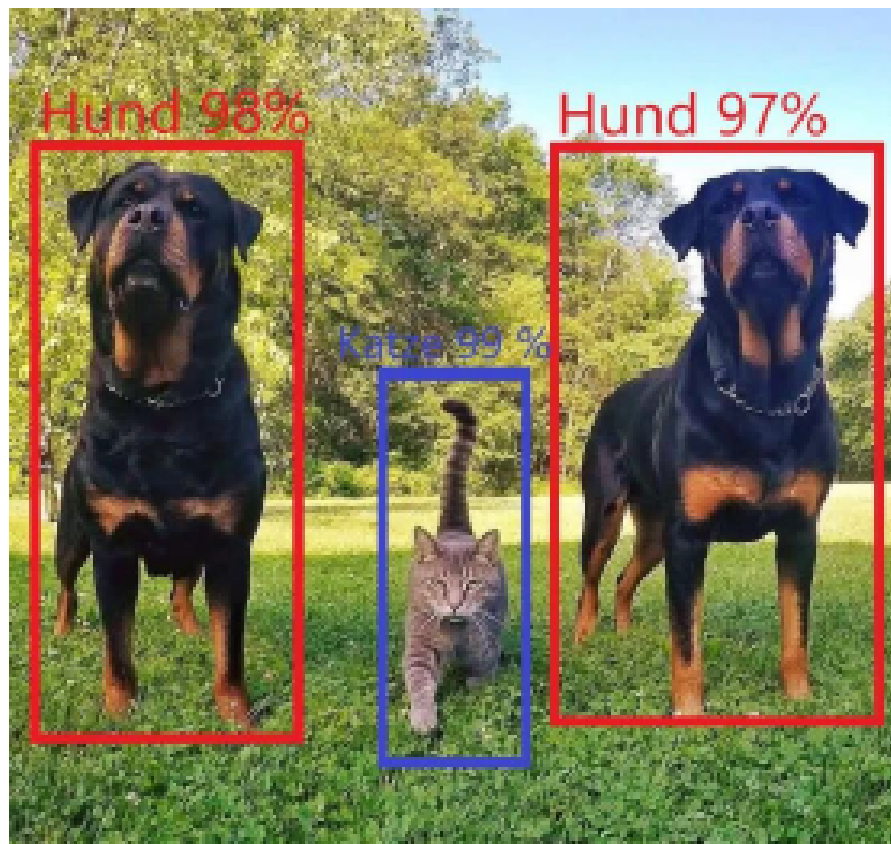


Abb. 16. künstliches Neuron

und die Anzahl unterscheiden sich. Es gibt mehrere Ansätze im Deep Learning Spektrum zur Objekterkennung, beispielsweise YOLO. In diesen Versuchsaufbau wurde Faster R-CNN Modell entschieden da es mehr Objekte auf einem einzigen Bild erkennen kann als YOLO. In dem folgenden Kapitel wird ein Augenmerk auf die stetigen Veränderungen im Aufbau der R-CNN Reihe und die dadurch resultierenden Verbesserungen geworfen werden (Joseph Redmon, 2016).

Selective Search bildet die Grundlage für R-CNN. Eine komplette Ausarbeitung des Algorithmus würde das Ziel dieser Arbeit verfehlen. Es soll lediglich das Grundkonzept veranschaulicht werden um den Leser das Prinzip näher zu bringen. Dieser Algorithmus arbeitet nach dem Prinzip des Clustering und bildet Cluster in Pixelregion, welche im ähnliche Werte haben. Das bedeutet Pixel – die in einem definierten Skalenbereich zueinander liegen – werden als Gruppe zusammengefasst und anschließend mit Bounding-Boxen umrandet. Siehe Abbildung 5-3 veranschaulicht das Prinzip. Der Algorithmus hilft Bilder vorzubereiten und den Suchraum für die NN zu verkleinern, da diese später als Vorhersagen im Training genutzt werden können. Selective Search bietet den Vorteil gegenüber dem Brute Force Ansatz, dass der Suchraum der potentiellen Objekte eingeschränkt wird (R.R. Uijlings, 2012). Der Brute Force Ansatz ist alle möglichen Region Proposal zu erstellen. In einem Fall von einem 128 x 128 Pixel Bild wären das Möglichkeiten Region Proposals. Das Region-Based Convolutional Neural Network (R-CNN) besteht aus drei Modulen. Das Region Proposal-, Feature Extraktion-, und Classification Regions Modul, deren Komposition kann in Abbildung 5-4 entnommen werden. Das Netzwerk folgt die unter 5 Objekterkennung – Modell (Wiegand) – Modell genannten Grundzügen der Objekterkennung (Ross Girshick, 2014). Das Prinzip von Region Proposal wurde in „5.1.1 Region Proposal – Selective Search“ erläutert. R-CNN nutzt diesen Algorithmus um mögliche Objekte auf Bildern vorzuselektieren. Es unterteilt das jeweilige Bild in bis zu 2000 Regionen, welche als Fundament für die Objekterkennung gelten und gibt diese einzelnen Regionen an das Feature Extraktion Modul weiter. Dieses ist mit ähnlichen Layern aufgebaut wie das Modul in „4.1 Aufbau des neuronalen Netzwerks“ beschrieben. Es besteht aus mehreren darauffolgenden Convolutional Layern und Pooling Layern, gefolgt von zwei Fully-Connected Layern. Diese folgen dem gleichen Prinzip, Features auf Bildern zu lernen, welche bei der Inferenz helfen bestimmte Objektklassen zu identifizieren. Dabei werden im Training die Gewichte der Sliding-Windows in den Convolutional Layer so angepasst, um objekttypische Feature zu erkennen (Ross Girshick, 2014). Die Klassifikation erfolgt über Support Vector Machines (SVM). Dabei muss vor dem Training des R-CNN festgelegt werden wie viele Objektklassen der Datensatz enthält und dementsprechend ist die Anzahl der SVM anzupassen (Ross Girshick, 2014). Das Ziel der Bounding Box Regression ist es, anstatt wie bei den Klassifikationsaufgaben im Deep Learning Bereich, eine stetige Variable vorherzusagen. In diesen Fall vier stetige reale Zahlen, welche die Koordinaten der Bounding Boxes darstellen. Der Input im Trainingsdurchlauf ist diese stehen für die vier Eckpunkte der vorhergesagten Bounding-Box Koordinaten der Selective Search. Die Koordinaten der Bounding Box aus den Trainingsdaten entsprechen

Das Ziel ist es am Ende durch die Methode – des kleinsten Quadrats – den Abstand zwischen P und G zu verkleinern. Hervorzuheben sei noch die Zuordnung, von G zu dem entsprechenden P. Da 2000 P durch den Region Proposal Algorithmus erzeugt werden, muss entschieden werden, welches der P am ehesten eine Vorhersage für die G Koordinate liefert. Dies wird 5 Objekterkennung – Modell (Wiegand) 24 dadurch erlangt indem die Flächen der einzelnen P's und G's Bounding Boxes ausgerechnet wird und verglichen wird mit wieviel Prozent das jeweilige P von G abgedeckt wird. Alle P's, welche über einen prozentualen Flächenanteil von 60R-CNN hat schwächen, welche durch Fast R-CNN verbessert werden. Das Training ist eine Multi-stage Pipeline. Das heißt die Bilder werden nach dem Selective Search in unter Bilder zerlegt und erst dann einzeln in das Netzwerk gegeben. Daraus resultierten eine erhöhte Trainingszeit und ein Verlust von Information, wenn man bedenkt, dass die Region Proposals aus den Bildern herausgeschnitten werden und in das NN nacheinander zufällig eingespeist werden. Ein weiterer Nachteil ist der hohe Trainingsaufwand bezüglich Zeit und Speicher. Dadurch das jedes Bild durch Selektive Search in 2000 Region Proposal zerlegt wird und diese einzeln das Netzwerk durchlaufen, müssen jede Objekt Vorhersage und die dazugehörigen Features abgespeichert werden. Dies kann bei großen Datensätzen wie bei „VGG16“ mehrere 100 GB Speicher beanspruchen. Außerdem beträgt die Inferenzzeit bei Bildern um die 47 Sekunden pro Bild (Ross Girshick, 2014). Das „Fast R-CNN“ von Ross Girshick baut auf das Modell „R-CNN“ auf und geht auf die in „5.1.2 R-CNN“ beschriebenen Nachteile ein. Der Aufbau kann Abbildung 5-5 entnommen werden. 5 Objekterkennung – Modell (Wiegand) 25 Abbildung 5-5 Fast R-CNN (Girshick, 2015) Ein großer Vorteil des Modells besteht darin, dass das NN zunächst die Feature Extraktion durch die Convolution-Layers durchgeführt und dann das Region-Proposal auf die Convolution Feature Map durchgeführt wird. Das bedeutet, dass das Trainingsbild nicht zunächst wie in R-CNN Modell in 2000 Bilder zerlegt und in das NN geladen wird, sondern die Featureextraktion das gesamte Bild durchläuft. Um die Region Proposals auf die Convolution Feature Map zu übertragen, wird ein neuer Layer namens Region of Interest Pooling Layer (ROI-Pooling) eingeführt (ROI-Pooling) (Girshick, 2015). Das Faster R-CNN ist derzeit der aktuellste Entwicklungsstand der R-CNN Modell Reihe und wird in unserem Versuchsaufbau verwendet. In diesen wird das Region Proposal Modul durch ein Region Proposal Network abgelöst. Dieses wird durch ein NN, welches als Input die Convolutional Feature Map erhält und als Output die Region Proposal in Form von Bounding Boxes wieder an den ROI Pooling Layer weitergibt. Die Abbildung 5.7 veranschaulicht diesen Aufbau. Dieser übernimmt die komplette Objekterkennung durch ein NN. Der daraus resultierende Vorteil ist ein durchgängig trainierbares Modell und wird durch ein Ähnliches Verfahren wie in Punkt 2.4 Gradient-Descent beschrieben realisiert (Shaoqing Ren, 2016). Das Region Proposal Network bekommt als Inputs zum einem, den Output aus den Convolution Layers generierten Feature Map und zum anderem k Anchor Boxes. Diese sind in einer fixierten rechteckigen Größe. Diese folgen dem gleichen Prinzip der Bounding-Boxes, welche in „5.1.2 R-CNN“ beschrieben worden sind.

K Region Proposal Koordinaten sollen erzeugt werden. Anchor Boxen geben außerdem die Relationen von Länge zu Breite der Sliding-Windons vor, welche auf der Convolution Feature Map angewendet wird. Im Paper wird die Aufgabe das Netzwerk als „Attention“ beschrieben, welche für das NN den Focus 5 Objekterkennung – Modell (Wiegand) 27 auf bestimmte Bildbereiche lenkt. Die cost function für das Region Proposal Network ist zum einem geben aus, Klassifikation Region Proposal und Regression, welche die Koordinaten mit den Ground-Truth Koordinaten der Boxen vergleicht. Die cost function folgt dem gleichen Prinzip der Bounding-Box Regression des R-CNN. Anschließend geht das Verfahren wie beim Fast-RCNN Modell weiter. Die vom RPN ermittelten Koordinaten werden nun auf der Convolutional Feature Map angewandt, darauf folgt das ROI-Pooling und die Klassifikation der einzelnen Bounding Boxen. Die dadurch resultierte Inferenzzeit ist enorm ge-stiegen und liegt im Durchschnitt bei 10 ms pro Bild (Shaoqing Ren, 2016).

2.3 Deep Learning und Informationstheorie

Um das Konzept von Deep Larning besser zu verstehen kann man diese aus den Informationstheoretischen Blickwinkel betrachten. Tischby and Schwarz-Ziv (Shwartz-Ziv & Tishby, 2017) untersuchten dabei Neuronale Netzwerke in Verbindung mit Deep Learning.

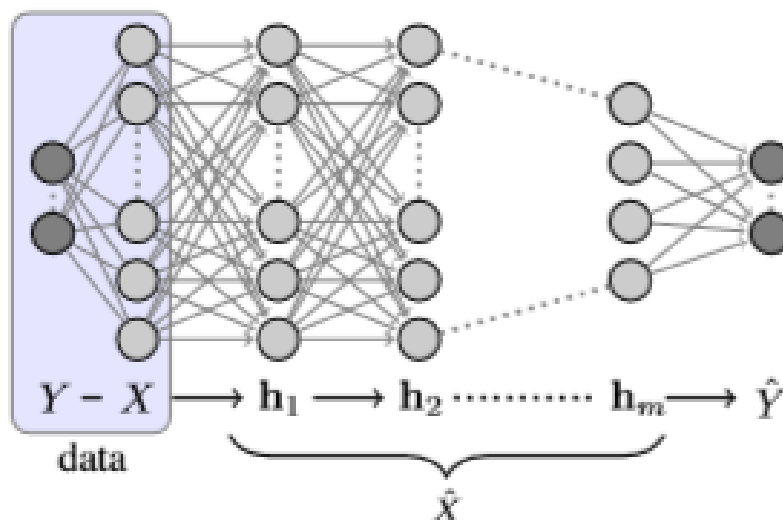


Abb. 17. Neuronal Network als Markov Chain

Das ins 20 dargestellte Netzwerk kann als Markov Kette betrachtet werden. Wobei jeder Layer $h^n:1,...,N$ als ein Zustand einer Markov Kette betrachtet werden kann. Wobei der Übergang durch von Informations als $h_i \rightarrow h_{i+1}$ dargestellt werden kann. Da es keine Rekursion gibt und die Information von Input Layer X durch h_{arg} zu Output Layer Y durch fließt. Dabei untersuchten sie wie sich die Transinformation $I(X;Y)$ von Input Variable X zu Output Variable Y verhält gegeben durch die Multivariate Verteilung $p(X,Y)$. Dieses Prinzip wird Information Bottleneck Prinziple genannt (Tishby & Zaslavsky, 2015).

2.4 Convolution Neural Networks

einabuen wie bild bearbeite ist: <https://arxiv.org/pdf/1801.00634.pdf>

Convolution Neural Networks(CNN) sind eine besondere Art von künstlichen neuronalen Netzwerken, sie sind dafür konzipiert auf Datensätzen zu arbeiten welche in eine Matrix Form gebracht worden sind. Der Input eines CNN können

beispielsweise Bilder sein welche durch die Matrix $A = \begin{bmatrix} a_1 & a_1 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \\ \vdots & n_n & \ddots & \vdots \\ x_1 & x_2 & x_3 & x_n \end{bmatrix}$

dargestellt werden. Jedes Element x_{ij} stellt einen Pixel eines Bildes da, wobei $x_n \in [0,255]$. Die Matrix $A^{w \cdot b \cdot c}$ stellt $w \cdot b \cdot c = N$ dimensionale Matrix da. Wobei w die Länge und b Breite des Bildes entspricht. c sind die Farbspektren eines Bildes und sind in einen RGB-Farbraum 3 beziehungsweise in einen schwarz-weiß Bild 1. Nachdem der Input eines CNN definiert ist kommt nun der Aufbau. CNN setzen sich aus mehrere Schichten von Convolution Layern zusammen. Ein Netzwerk kann mehrere N-Layer haben. Wobei jeder Layer aus mehreren Convolution oder auch Kernels genannt, zusammengesetzt ist. Ein Aufbau kann aus Abbildung 34 entnommen werden(Goodfellow et al., 2016).

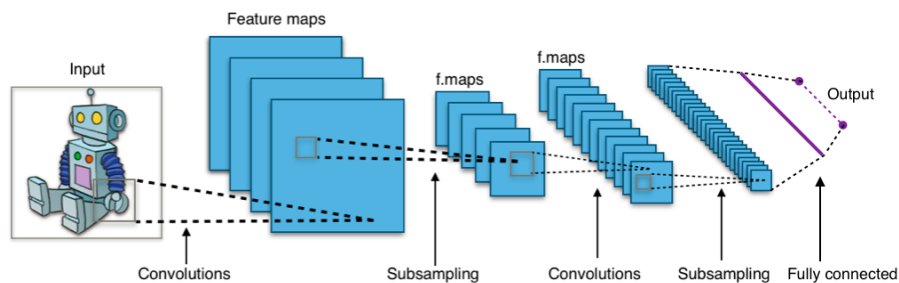


Abb. 18. Convolutional Neural Network

Die Kernels, also die einzelnen Filter, von den jeder der N-Layer k besitzt sind $K^{n \cdot n}$ Matrizen jedes k_{ij} in einem Filter entspricht einen aus der üblichen Neuronalen Netzwerk Architektur bekannten Gewichte. Diese Gewichte werden dann

durch den Backpropagation-Algorithmus in der Trainingsphase des Netzwerkes angepasst um den Verlust der Loss-Funktion durch bestimmten des Gradienten zu minimieren. Das durch die Abbildung 34 dargestellte Subsampling ist der Output aus den Convolutional Layern (Goodfellow et al., 2016).

Da Input und Kernel unterschiedliche Größen haben und man den gesamten Input mit dem Kernel abdecken möchte, bewegt sich der Filter um s Position auf den Input und führt erneut einen Berechnungsschritt durch. Dieser Vorgang wird Stride genannt. An jeder Position wird das Produkt von jedem x_{ij} des Input und k_{ij} des Kernel durchgeführt. Anschließend werden alle Produkte aufsummiert. In Abbildung 19 ist dieser Vorgang verdeutlicht. Zusätzlich gibt es die Möglichkeit für das sogenannte Zero Padding P . Dabei werden mehrere 0 um die Input Feature Map, am Anfang und Ende der Axen anfügt. Dies ist notwendig wenn Kernel und Input Größe nicht kompatibel zueinander sind. Die Anzahl der möglichen Positionen ergeben sich aus Kernel Größe und den Input des jeweiligen Kernel sowie des Strides. Die Output Größe W kann berechnet werden durch $W = (W-F+2P)/s+1$. Wobei F für die Größe des Kernel steht (Dumoulin & Visin, 2016).

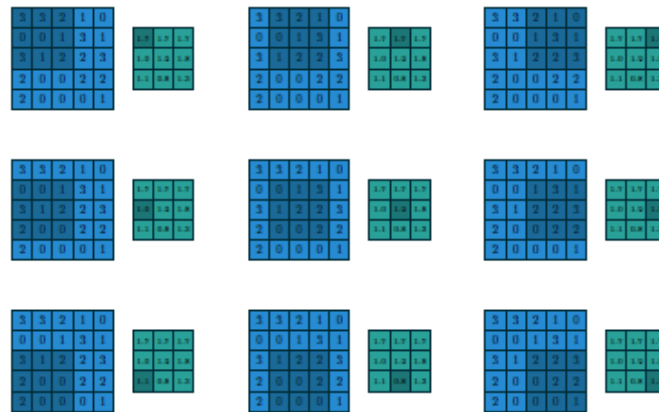


Abb. 19. Convolution Beispiel

Um besser zu verstehen welche Auswirkungen die Anzahl der Kernels in Layer n auf die Größe des Outputs von n und die Anzahl der Kernels in Layer $n+1$ für den nächsten Layer haben, wird ein Beispiel aufgezeigt. Der erste Layer hat 20 Kernels mit der Größe 7×7 und Stride 1. Der Input A für einen Kernel K ist ein 28×28 Matrix. Der Output aus diesen Filter sind 20 22×22 Feature Maps. Würde der Input ein $28 \times 28 \times 3$ Bild mit 3 RGB Channels sein, der Output 60 22×22 Feature Maps. Allgemein kann Convolution Layer als Supersampling gesehen werden und Stride gibt an wieviele Dimensionen bei diesen Prozess pro Convolution Layer entfernt werden soll. Der letzte Layer ist ein fully-connected

Layer welcher den typischen Anforderungen von ANN entspricht (Dumoulin & Visin, 2016).

Transposed Convolution, auch genannt Fractionally Strided Convolution oder Deconvolution ist eine Umkehrfunktion von der üblichen Convolution. Es verwendet die gleichen Variablen wie Convolution. Dabei wird ein Kernel K mit der Größe $N \times N$ definiert der Input I mit der Größe $N \times N$ und Stride $s = 1$. Deconvolution kann wie Convolution angesehen werden mit Zero Padding auf dem Input. Das in Abbildung 20 gezeigte Beispiel zeigt einen deconvolution Vorgang mit eine 3×3 Kernel über einen 4×4 Input. Dies ist gleich mit einen Convolution Schritt mit einen 3×3 kernel auf einen 2×2 Input und einer 2×2 Zero Padding Grenze. Convolution ist Supsampling und mit Deconvolution wird Upsampling betrieben. Durch diesen Schritt kommt es zu einer Dimensionserhöhung des Inputs. Die Gewichte der Kernels bestimmen wie der Input transformiert wird. Durch mehrere Schichten von Deconvolution Layer kann von einer Input Größe $N \times N$ auf eine Output Größe $K \times K$, wobei $K > N$ mit Abhängigkeit von Kernel und Stride abgebildet werden (Dumoulin & Visin, 2016).

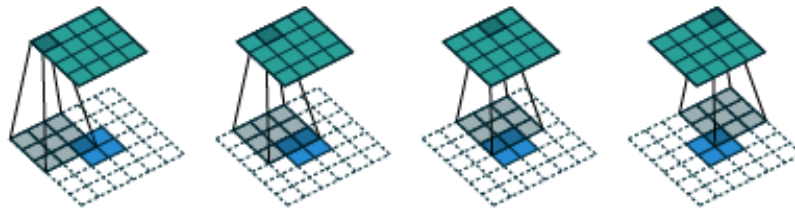


Abb. 20. Deconvolution Beispiel

2.5 Generative Modelle

2.5.1 autoencoder Autoencoder sind ein anderes Modell von generativen Modelle. Ihre Aufgabe besteht darin einen Input zu komprimieren und dann wieder zu erstellen. Die Aufgabe wird als rekonstruktion benannt. Die Technik auf die dabei zurückgegriffen wird nennt sich Dimensionreduktion. Dabei werden die Dimension der Daten so reduziert und Informationen bei zu behalten welche als relevant gelten. Diese Methode finden auch in anderen Machine Learning Ansätzen Anwendung wie in beispielsweise der Principale component anaysis(PCA). Bestandteile eines Autoencoder ist ein Encoder g parametrisiert mit ϕ welcher einen Input x element von x^i wo x ein vektor der länge i ist. Und Damit die Input Dimension bestimmt. Dieser wird durch den Encoder auf einen Vektor z^k abgebildet wobei $k < i$ ist. Der Decoder f θ bekommt nun als Input z^k und mappt z auf x^l wobei $l = i$. Und somit die gleiche Dimension wie der Input. Aufgabe ist es nun das der Encoder den Input z so gut komprimiert das encoder es schafft das $x = z$.

Die Parameter ϕ und θ werden durch erlernt. Und können beispielsweise durch fully-Connected Layer, Convolutional-Layer oder Deconvolutional Layer dargestellt werden. Eine Metrik um zu messen wie gut das Modell seine Aufgabe erfüllt könnte beispielsweise Cross-Entropy functionen sein.

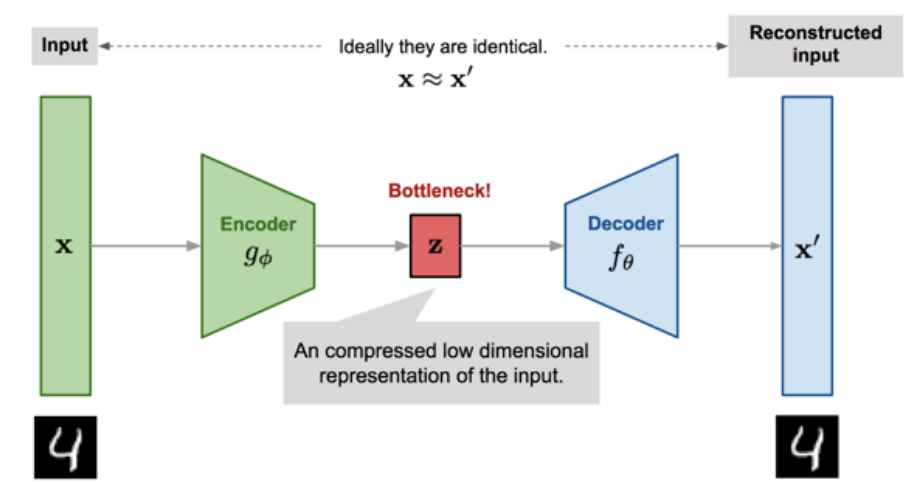


Abb. 21. autoencoder

2.5.2 Zielfunktion Autoencoder

Da in der Arbeit der Input 3d Pointcloud sind und diese Sets und diese invariant

zu ihren Permutationen. Das heißt ändere ich die Anordnung meiner einzelnen Punkte in meien Set bleibt das Ergebniss unverändert. sind sind kann auf übliche Zielfunktionen nicht zürockgegriffen werden da diese mit Sturcturierten Input Daten arbeiten. Das Problem dabei besteht wenn ich 2 unterschiedliche Sets von Punkten habe wie Messe ich um herraus zu finden wie hoch die discrepanc zwischen den Beiden sets ist. <http://graphics.stanford.edu/courses/cs468-17-spring/LectureSlides/L14>

Generative Modelle haben das Ziel eine Wahrscheinlichkeitsverteilungen zu erlernen. Anschließend kann diese als ein Modell genutzt werden und Samples zu erzeugen. Die Modelle können dabei beispielsweise auf ANN oder Markov Chains trainiert werden(Goodfellow et al., 2016). Im folgenden liegt der Fokus auf ANNs. Ein mögliches Anwendungsbeispiele wären es dem generativen Model, Bilder von bestimmten Objekten zunächst als Trainingsdaten zu geben. Anschließend können von er erlernten hochdimensionalen Wahrscheinlichkeitsverteilung Samples gezogen werden und neue Bilder erzeugt werden, welche nicht im Trainingsdatensatz vorhanden waren. Allgemein gehalten können jegliche Typen von Daten wie Text, Bild oder Audiodateien für generative Modelle herangezogen werden. Es gibt unterschiedliche Typen von generativen Modellen, welche sich vom Aufbau des Neuronalen Netzwerk und der Zielfunktion unterscheiden. Beispiele dafür sind Boltzmann Maschine, Autoencoder oder Deep Belief Networks(Goodfellow et al., 2016). Diese Arbeit beschäftigt sich mit einen anderen Vertreter, dem Generativ Adverserial Network(GAN). In den letzten Jahren konnte sich das GAN als best practice Ansatz bei den generativen Modellen herausarbeiten was Performancegründe bei der Trainierbarkeit und Qualität der generierbaren Daten zu Grunde liegt(Goodfellow et al., 2016). Die Modelle arbeiten nach der Maximum Likelihood Schätzverfahren(ML-Schätzer) in dem die Parameter θ dahingegen angepasst werden, dass die unsere beobachtetet Daten am ehesten passen. Man kann ML-Schätzer als Kulback-Leibler(KL) Divergenz darstellen und das generative Modelle das Ziel haben die KL Divergenz zwischen den Trainingsdaten P_r und den generierten Daten P_g zu minimieren. In Abbildung 22 ist dieser Prozess dargestellt. Diese Modelle versuchen dann die diese Cost Funktion

$$KL(P_r||P_g) = \int_x P_r \log \frac{P_r}{P_g} dx$$

zu minimieren. Wenn nun beide Verteilungen $P_r = P_g$ sind, hat das Model sein Minimum Loss erreicht und Θ muss nicht mehr angepasst werden. Interessant wird es, wenn $P_r \neq P_g$. Wenn $P_r > P_g$ führt, das dazu dass das Integral schnell gegen unendlich konvergiert. Was dazuführt, das hohe Kosten entstehen wenn die Verteilung von generativen Modell erzeugt, nicht die Daten abdeckt. Wenn nun $P_r < P_g$ ist, dann bedeutet das, dass x eine niedrige Wahrscheinlichkeit hat aus unseren Trainingsdaten zu kommen aber eine hohe Wahrscheinlichkeit von den Generator erzeugt zu werden. Dann würde sich die KL gegen 0 konvergieren. Was zur Folge hat, dass der Generator Falsch ausschauende Daten geniert aber keine Kosten dafür erzeugt werden und im Umkehrschluss es zu keinen Veränderungen unseren Θ kommt. Man vermutet das dies der Grund für die Problematiken von

generativen Modellen wie Autoencoder und CO sind. Dies ist aber noch kein abgeschlossenes Problem und wird weiterhin erforscht (Salimans et al., 2016). Unter GAN werden wir auf dieses Problem erneut aufgreifen und es wird gezeigt inwiefern sich GAN dieses Problem angeht. Generative Modelle gehören zu einem Bereich des unüberwachten Lernens, da keine Labels für die Trainingsdaten gebraucht werden. Probleme welche diese Modelle haben sind beispielsweise, dass Autoencoder zwar mit wenig Trainingsaufwand trainiert werden können jedoch sind die generierten Bilder sehr trüb. Allgemein haben Autoencoder und Co. Vorteile im Lernen des latenten Raums von Objektklassen, weisen aber Probleme beim Generieren von neuen Daten auf. Da gezeigt wurde das im Deep Learning Bereich die discriminativen Modelle mit Zunahme der Daten stark an Leistung zunehmen. Und GANs Stärke in der Datengeneration in guter Qualität liegt. Kann es seine Vorteil gegenüber den anderen Modellen ausspielen (Salimans et al., 2016).

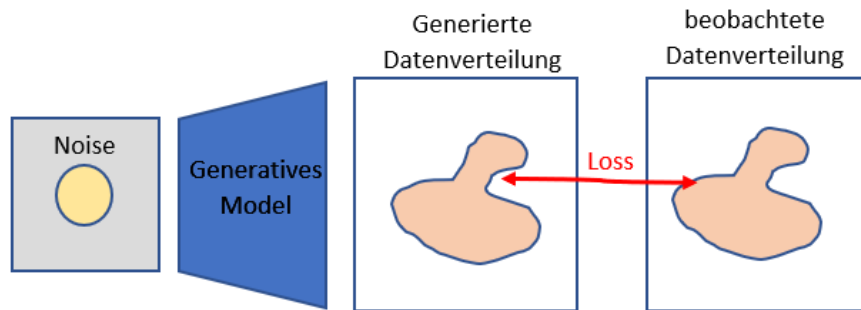


Abb. 22. Generative Modelle

2.6 Generative Adversarial Network

Ein GAN besteht aus zwei ANN, dem Discriminator D und dem Generator G . Das Ziel des G ist es, Daten x zu erzeugen, welche nicht von Trainingsdaten y unterschieden werden können. Dabei wird eine vorangegangene Input Noise Variable $p_z(z)$ verwendet, welche eine Abbildung zum Datenraum $G(z; \Phi_g)$ herstellt. Dabei sind Φ_g die Gewichte des neuronalen Netzwerkes von G . Der Discriminator hat die Aufgabe zu unterscheiden, ob der jeweilige Datensatz von G erzeugt wurde und somit ein fake Datensatz ist, oder von Trainingsdaten y stammt (Goodfellow et al., 2014). Die Zusammensetzung zwischen den beiden Netzwerken kann aus Abbildung 23 entnommen werden.

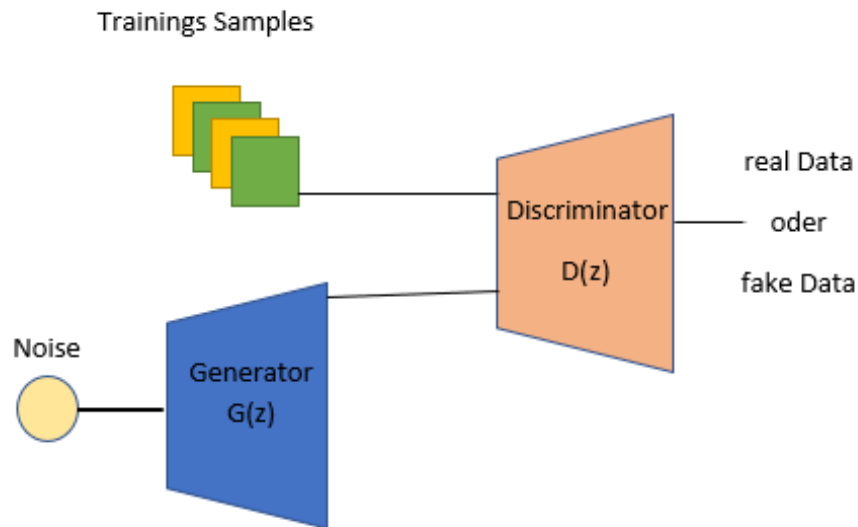


Abb. 23. Generativ Adverserial Network

Der Discriminator ist definiert durch $D(x; \Phi_d)$. Wobei Φ_d die Gewichte des Discriminators sind und $D(x)$ die Wahrscheinlichkeit ist, dass x von den Trainingsdaten stammt und nicht von p_g . Die Wahrscheinlichkeitsverteilung für unsere Trainingsdaten ist p_r . Im Training werden dann Φ_d so angepasst, dass die Wahrscheinlichkeit Trainingsbeispiele richtig zu klassifizieren maximiert wird. Und Φ_g wird dahingehen trainiert die Wahrscheinlichkeit zu minimieren, so dass D erkennt dass Trainingsdatensatz x von G erzeugt wurde. Mathematisch ausgedrückt durch $\log(1 - D(G(z)))$. Die gesamte Loss-Funktion des vanilla GAN ist definiert als

$$\min_G \max_D V(D, G) = E_{x \sim p_{\text{data}}(x)} [\log D(x)] + E_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

diese beschreibt ein Minmax Spiel zwischen G und D . Welches das globale Optimum erreicht hat wenn $p_g = p_r$. Das heißt, wenn die Datenverteilung, welche von G erzeugt wird, gleich der unserer Trainingsdaten ist (Goodfellow et al., 2014).

Das Training erfolgt durch den folgenden Algorithmus:

Algorithm 3: Minibatch stochastic gradient descent Training für Generative Adversarial Networks. Die Anzahl der Schritte welche auf den Discriminator angewendet wird ist k

```

1 for Anzahl von Training Iterationen do
2   for k Schritte do
3     • Sample minibatch von m noise Samples  $z^{(1)}, \dots, z^{(m)}$  von noise
       $p_g(z)$ 
4     • Sample minibatch von m Beispielen  $x^{(1)}, \dots, x^{(m)}$  von Daten
      Generationsverteilung  $p_{\text{data}}(x)$ 
5     • Update den Discriminator zum aufsteigenden stochastischen
      Gradienten:
6      $\nabla_{\Phi_d} \frac{1}{m} \sum_{i=1}^m [\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))]$ 
7   end
8   • Sample minibatch von m noise Samples  $z^{(1)}, \dots, z^{(m)}$  von noise  $p_g(z)$ 
9   • Update den Generator mit den absteigenden stochastischen
      Gradienten:
10   $\nabla_{\Phi_g} \frac{1}{m} \sum_{i=1}^m \log(1 - D(G(z^{(i)})))$ 
11 end
```

Beim Training wird ein stochastischer Minibatch von mehreren Trainingsdaten gleichzeitig erstellt. Dies soll dabei helfen, dass der Generator sich nicht auf bestimmte Gewichte fest fährt und auf Trainingssätze kollabiert. So weisen die erzeugten Daten mehr Variationen auf (Salimans et al., 2016). D wird zunächst in einer inneren Schleife auf n Trainingsätzen trainiert, womit man Overfitting von D vermeiden will, was zur Folge hätte, dass D nur den Trainingsdatensatz kopieren würde. Deshalb wird k mal D optimiert und ein mal G in der äußeren Schleife.

Ein möglicher Aufbau von GAN wird in Abbildung 24 dargestellt. Dies ist das sogenannte Deep Convolution GAN(DC GAN), welches dafür konzipiert wurde auf Bilddaten zu arbeiten. Dabei besteht der Generator aus mehreren Schichten von Deconvolution Layern. Welche den Input Noise Variable $p_z(z)$ auf y abbildet. D besteht aus mehreren Schichten von Convolution Layern und bekommt als Input die Trainingsdaten, oder die von G erzeugten Y , und entscheidet über die Klassifikation(Radford, Metz, & Chintala, 2015).

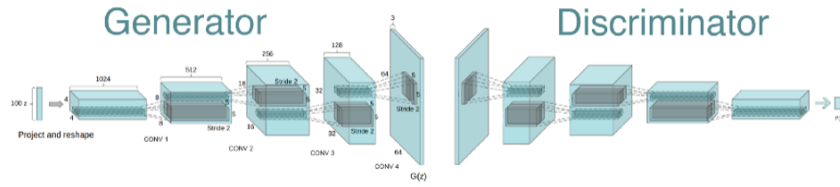


Abb. 24. Deep Convolutional GAN

Wie unter Generativen Modellen gezeigt wurde kann das asymmetrische Verhalten der KV Divergenz zu schlechten Trainingsergebnissen führen. Goodfellow (Goodfellow et al., 2014) zeigte, dass sich die MinMax Loss-Funktion des GAN auch als Jensen-Shannon Divergenz(JS Divergenz) darstellen lässt. Diese ist definiert als

$$D_{JS}(P_r || P_g) = \frac{1}{2} D_{KL}(P_r || \frac{P_g + P_r}{2}) + D_{KL}(P_g || \frac{P_g + P_r}{2})$$

wobei P_r die Wahrscheinlichkeitsverteilung der Trainingsdaten ist und P_g die des Generators. Huzár (Huzár, 2015) zeigte, dass durch das symmetrische Verhalten der JS Divergenz ein potentiell besseres Trainingsergebnis entstehen kann, im Vergleich zu der KL Divergenz. Damit zeigte weshalb GANs im Vorteil gegenüber anderen generativen Modellen sind. Abbildung 25 veranschaulicht dieses Konzept. Der linke Graph zeigt 2 Normal Verteilungen. In der Mitte wird die KV Divergenz der beiden Normal Verteilungen dargestellt. Rechts ist die JS Divergenz der Beiden dargestellt. Man sieht sehr gut das asymmetrische Verhalten der KV und das symmetrische der JS. Dadurch lassen sich aussagekräftigere Gradienten, bestimmen welche zum Optimieren von D und G benötigt werden(Huzár, 2015).

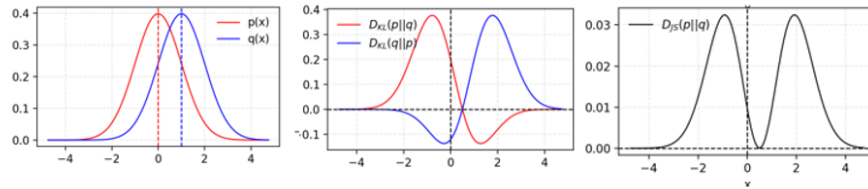


Abb. 25. KL Divergenz und JS Divergenz

2.6.1 Probleme mit Generative Adversarial Networks Wie auch die anderen generativen Modelle haben auch GANs noch Schwächen bezüglich der Trainingsabläufe und der Qualität der generierten Daten. Im Folgenden wird auf einige Probleme eingegangen.

Equilibrium D und G betreiben ein MinMax Spiel. Beide versuchen das Nash Equilibrium zu finden. Dies ist der bestmögliche Endpunkt in einem nicht kooperativen Spiel. Wie in dem Fall von GAN wäre das wenn $p_g = p_r$. Es wurde gezeigt, dass das Erreichen dieses Punktes sehr schwierig ist, da durch die Updates der Gewichte mit den Gradienten der Loss-Funktion starke Schwingungen der Funktion entstehen können. Dies kann zur Instabilität für das laufende Training führen (Salimans et al., 2016).

Vanishing gradient Dies beschreibt das Problem, wenn D perfekt trainiert ist mit $D(x) = 1, \forall x \in p_r$ und $D(x)=0 \forall x \in p_g$. Die Loss-Funktion würde in diesem Fall auf 0 fallen und es gäbe keinen Gradienten, für den die Gewichte von G angepasst werden können. Dies verlangsamt den Trainingsprozess bis hin zu einem kompletten Stopp des Trainings. Würde D zu schlecht trainiert mit $D(x) = 0, \forall x \in p_r$ und $D(x)=1 \forall x \in p_g$. Bekommt G kein Feedback über seine Leistung bei der Datengeneration hat er keine Möglichkeit p_r zu erlernen (Salimans et al., 2016).

Mode Collapse Während des Trainings von GAN kann es dazu kommen, dass der Generator möglicherweise auf eine Einstellung seiner Gewichte fixiert wird und es zu einem sogenannten Mode Collapse führt. Was zur Folge hat, dass der Generator sehr ähnliche Samples produziert (Arjovsky, Chintala, & Bottou, 2017).

Keine aussagekräftigen Evaluations Metriken Die Loss Funktion der GANs liefert keine aussagekräftigen Evaluationsmöglichkeit über den Fortschritt des Trainings. Bei discriminativen Modellen im üblichen Machine Learning besteht die Möglichkeit Validierungsdatensätze zu verwenden und an diesen die Genauigkeit des Modells zu testen. Diese Möglichkeit besteht bei GANs nicht (Huang et al., 2018).

2.6.2 Lösungsansätze für Generative Adversarial Networks Probleme

Nun werden einige Techniken aufgezeigt, welche die unter Abschnitt Probleme mit GAN genannten Schwierigkeiten angehen und zu einem effizienteren Training führen, damit eine schnellere Konvergenz während des Trainings erreicht wird.

Feature matching Dies soll die Instabilität von GANS verbessern und gegen das Problem des Vanishing Gradient angehen. G bekommt eine neue Loss-Funktion und ersetzt die des üblichen Vanilla GAN. Diese soll G davon abhalten, sich an D über zu trainieren und sich zu sehr darauf zu fokussieren, D zu täuschen und gleichzeitig auch versuchen die Datenverteilung der Trainingsdaten abzudecken(Salimans et al., 2016).

Minibatch discrimination Um das Problem des Mode Collapse zu umgehen, so dass es nicht zu einem Festfahren der Gewichte von G kommt, wird beim Trainieren die Nähe von den Trainingsdatenpunkten gemessen. Anschließend wird die Summe über der Differenz aller Trainingspunkte genommen und dem Discriminator als zusätzlicher Input beim Training hinzugegeben (Salimans et al., 2016).

Historical Averaging Beim Training werden die Gewichte von G und D aufgezeichnet und je Trainingsschritt i verglichen. Anschließend wird an die Lossfunktion je Trainingsschritt die Veränderung zu $i-1$ an die Loss-Funktion addiert. Damit wird eine zu starke Veränderung bei den jeweiligen Trainingsschritten bestraft und soll gegen ein Model Collapse helfen (Salimans et al., 2016).

One-sided Label Smoothing Die üblichen Label für den Trainingsdurchlauf von 1 und 0 werden durch die Werte 0.9 und 0.1 ersetzt. Dies führt zu besseren Trainingsergebnissen. Es gibt derzeit nur empirische Belege für den Erfolg, jedoch nicht weshalb diese Technik besser funktioniert(Salimans et al., 2016).

Adding Noises Noise an den Input von D zu hängen kann gegen das Problem des Vanishing gradienten helfen und das Training verbessern(Salimans et al., 2016).

Use Better Metric of Distribution Similarity Die JS Divergenz von vanilla GAN sorgt für bessere Trainingsergebnisse im Vergleich zu der KL Divergenz von anderen generativen Modelle. Jedoch weist die JS immernoch Probleme auf. Es wird vorgeschlagen diese durch die Wasserstein Metric zu ersetzen, da diese bessere Ergebnisse bei disjunkten Wahrscheinlichkeitsverteilungen liefern kann(Salimans et al., 2016).

2.6.3 Conditional Adversarial Networks Conditional Adversarial Networks (CGAN) beruhen auf dem Grundkonzept von vanilla GAN (vgl. Kapitel GAN). Es wird zusätzlich die extra Information y hinzugefügt. Diese kann jegliche Information sein, welche auf x abgestimmt ist. Beispielsweise kann y ein Klassenlabel zu den gelernten $P(x)$ sein. Die Zielfunktion des CGAN ist

$$\min_G \max_D V(D, G) = E_{x \sim p_{\text{data}}(x)} [\log D(x|y)] + E_{z \sim p_z(z)} [\log(1 - D(G(z|y)))]$$

und beschreibt eine bedingte Wahrscheinlichkeit, dass ein Trainingsdatensatz x oder ein Datensatz, welcher von dem Generator erzeugt wurde, von y abhängt (Mirza & Osindero, 2014).

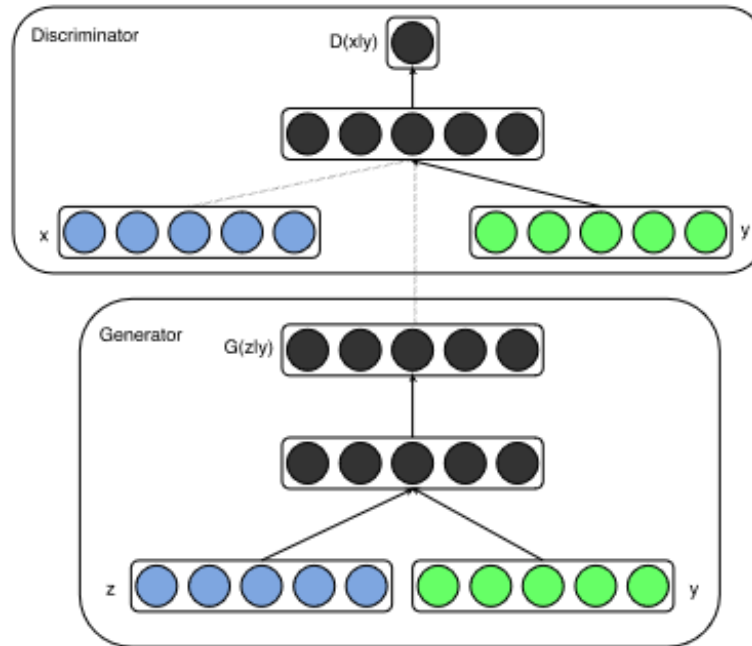


Abb. 26. Conditional Adversarial Network

Ein beispielhafter Anwendungsfall ist das Erlernen von selbstgenerierten Zahlen. Der MNIST Datensatz besteht aus 10000 handschriftlich eingescannten Zahlen von 0 - 9. Die Daten x können nun während des Trainings mit den dazugehörigen Zahlenlabels y trainiert werden und anschließend kann der Generator verwendet werden, um selbst die gewünschten x zu erzeugen, indem y gewählt wird (Mirza & Osindero, 2014). Das Konzept von CGAN wird nun in folgender Arbeit weiterentwickelt, um das Ziel zu haben, selber Bilder zu verändern.

2.7 Conditional-GAN

Conditional-GAN(C-GAN) ist eine modification des ursprünglichen GAN welches erlaubt bedingte Wahrscheinlichkeiten in Datensätze zu erlernen. Dass heißt zusätzliche Informationen in den lern Prozess einzuspeisen um den Output zu modifizieren. Im ursprünglichen GAN gibt es keine Möglichkeit auf den Output des GANs einfluß zu nehmen. Dabei wird das Modell so verändert das eine zusätzliche Information y zusätzlich als Input in den Discriminator oder den Generator zugefügt wird. Dabei kann y jedliche Information sein wie Label. Die Ziel funktion wird bedingt geändert. $\min_G \max_D V(D, G) = E_{x \sim p_{\text{data}}(x)} [\log D(x)] + E_{z \sim p_z(z)} [\log(1 - D(G(z)))]$

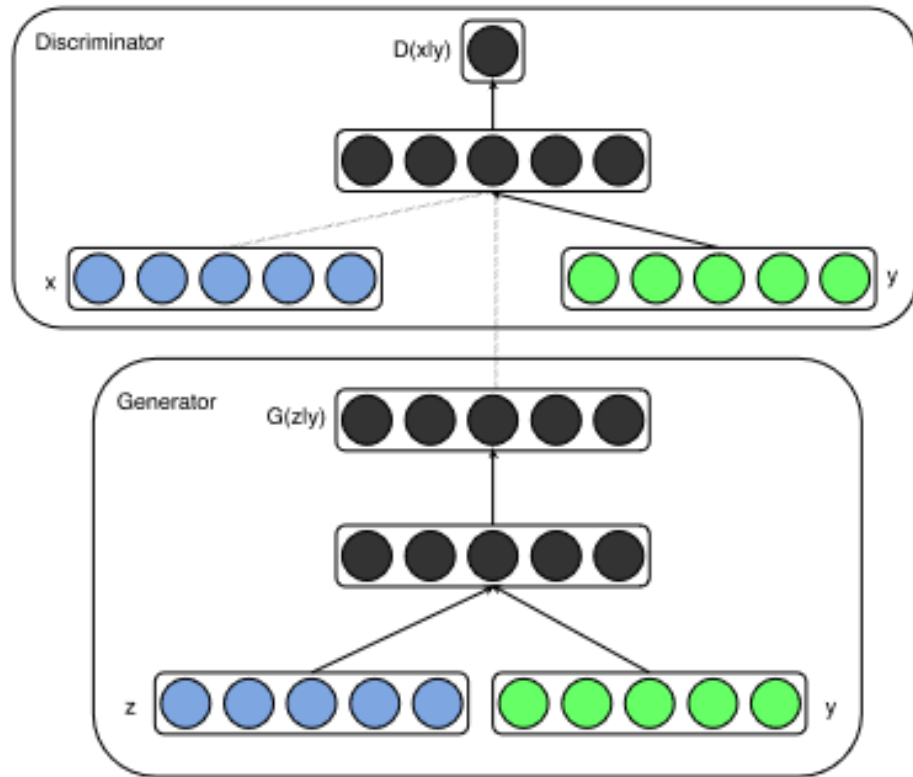


Abb. 27. Conditional Adversarial Network

2.8 3D-GAN

Das besondere an den 3D Raum im Vergleich zu Normalen 2D Bildern ist die hohe steigerung der Dimension und zu gleich der Hohe Informationsgehalt welcher in 3D Objekten steckt. Das Ziel von 3D-GANS ist es Modelle von Objekten zu erhalten. Dabei wird der latente Objektraum erfasst und soll damit die Wahrscheinlichkeiten für einzelne Objektklassen enthalten.

Die Architektur des typischen 3D-GAN ist dem vanilla GAN von Goodfellow ähnelt. Durch den Aufbau von 3D-Objekten unterscheidet sich nur die Dimensionalität der einzelnen Layer. Dadurch das 3d Objekte ins Räumliche Verhältnis gestellt werden müssen braucht man eine dritte Dimenson in den Layern welche die Tiefe der Objekte Darstellt. (Wu, Zhang, Xue, Freeman, & Tenenbaum, 2016a).

Die Zielfunktion bleibt die gleiche wie bei den üblichen GANS $\min_G \max_D V(D, G) = E_{x \sim p_{\text{data}}(x)} [\log D(x|y)] + E_{z \sim p_z(z)} [\log(1 - D(G(z|y)))]$

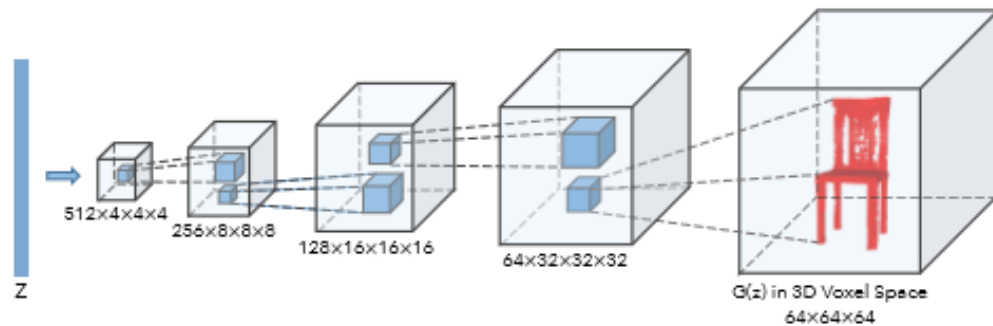


Abb. 28. Conditional Adversarial Network

(Wu et al., 2016a)

Das latent GAN hingegen benutzt eine andere Technik. Zunächst wird ein Autoencoder(siehe Autoencoder) verwebdet um Trainingsdaten zu trainieren. Ziel dabei ist den latenten Raum der Trainingsdaten zu erlernen und eine kompression der Daten um den suchraum zu verringern und dadurch das Training des GANs zu erleichtern. Nachdem das Trainingsabgeschlossen hat wird der Trainingsdaten satz durch den Encoder komprimiert und das Training vom GAN wird mit dem komprimierten Datensatz getan.

(Wu et al., 2016a)

3 Methoden

3.1 Aufbau

3.1.1 Datensatz 1. Versuchsaufbau Der erste Datensatz "SStühle" besteht aus 4014 Punktwolken mit je 2056 Punkten. Der Datensatz wurde aus von den

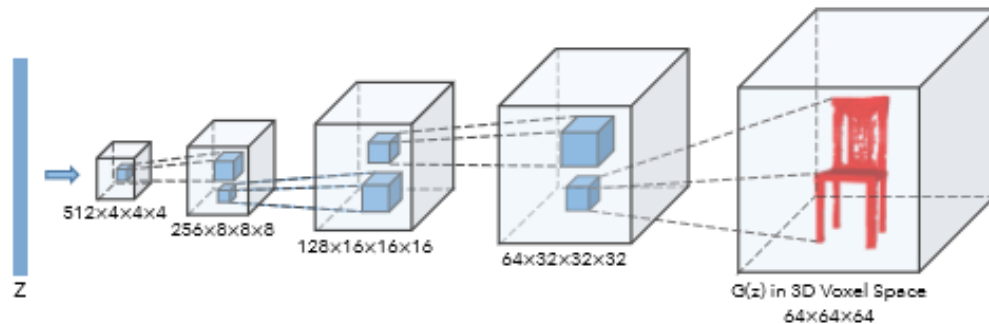


Abb. 29. Conditional Adversarial Network

Shapenet Datensatz entnommen. Ein Beispielpunktwolke kann aus Abbildung entnommen werden. Die Daten für den zweiten Datensatz "Blätterstämme vom Fraunhofer Institut. Der Grunddatensatz bestand aus mehreren 3D-Scans von Tabakpflanzen bei den die Blätter der Pflanze zu einen Datensatz zusammengefügt wurden sind. Der "Blätter" Datensatz besteht aus 420 Punktwolken mit je 2056 Punkten.

3.1.2 Datensatz 2. Versuchsaufbau Für den zweiten Versuchsaufbau wird auf den in Datensatz Versuchsaufbau 1. Blatt Datensatz "Blätterfürück gegriffen. Dabei wird in den Blättern punkte heraus genommen welche das verdecken oder zerstört sein eines Blattes simulieren soll. Dies passiert indem eine Gäußverteilung auf einer 3D-Sphere erzeugt wird. Anschließend wird das Komplement zwischen einem 3D-Blatt und einer 3D-Sphere berechnet das Ergebnis sind zerstörte Blätter mit Kreisrunden Löchern auf der Oberfläche. Der Algorithmus zum erzeugen der Trainingsdaten kann im Anhang entnommen werden. In Abbildung sind die 3 Abschnitte zu erkennen wie die Trainingsdaten erzeugt werden.

Insgesamt besteht der Trainingsdatensatz aus 450 unzerstörten Blättern sowie 1024 zerstörten paaren.

3.1.3 Trainingsaufbau - GAN Der Aufbau besteht aus zwei unterschiedlichen Versuchen. Aufbau 1 ist der RAW - GAN Aufbau. Dabei wird auf den herkömmlichen Aufbau von GAN zurückgegriffen. Die Allgemeinen Meta-Trainingsvariablen sind Learningrate mit 0,005 und einen AdamOptimizer, mit einem Beta1 von 0.5 und einem Beta2 von 0,5Der Discriminator besteht aus 4-Layern welche 1-Dimensionaler Convolutional Layer bestehen. Mit einer Kernel Größe von 1 und stride von 1. Die Aktivierungsfunktion sind Relus. Darauf folgt 3 fully connected Layer mit der Größe 128, 54 und 1 alle mit einer Relu Aktivierungsfunktion. Der Generator besteht auf 5 fully-connected Layern mit [64,128,512,1024,1536] neuronen jeweils mit einer Relu Aktivationsfunktion. Der Aufbau kann in Ab-



Abb. 30. 3D Punktwolke einer Tabakpflanze

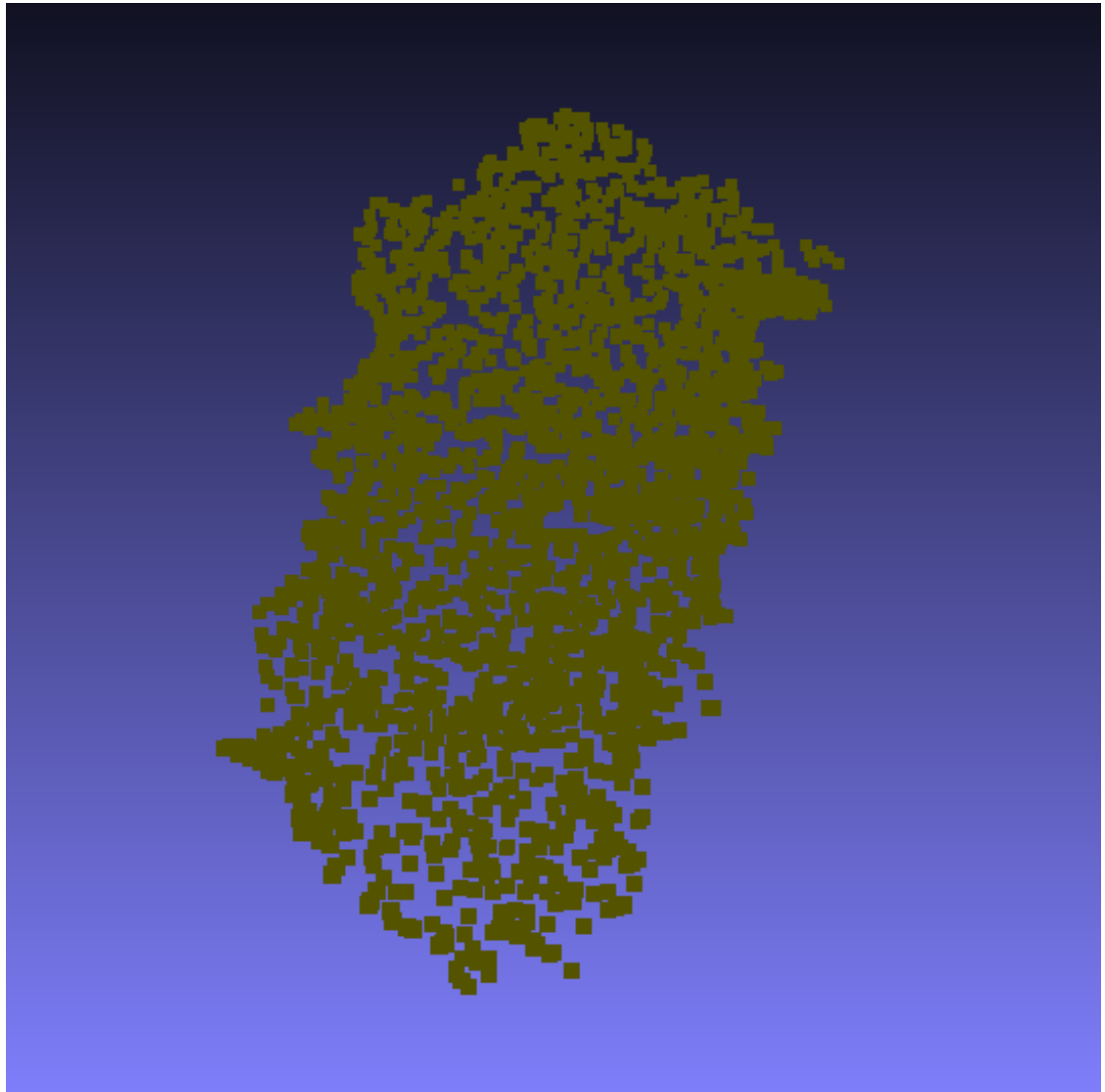


Abb. 31. 3D Punktwolke einer Tabakpflanze

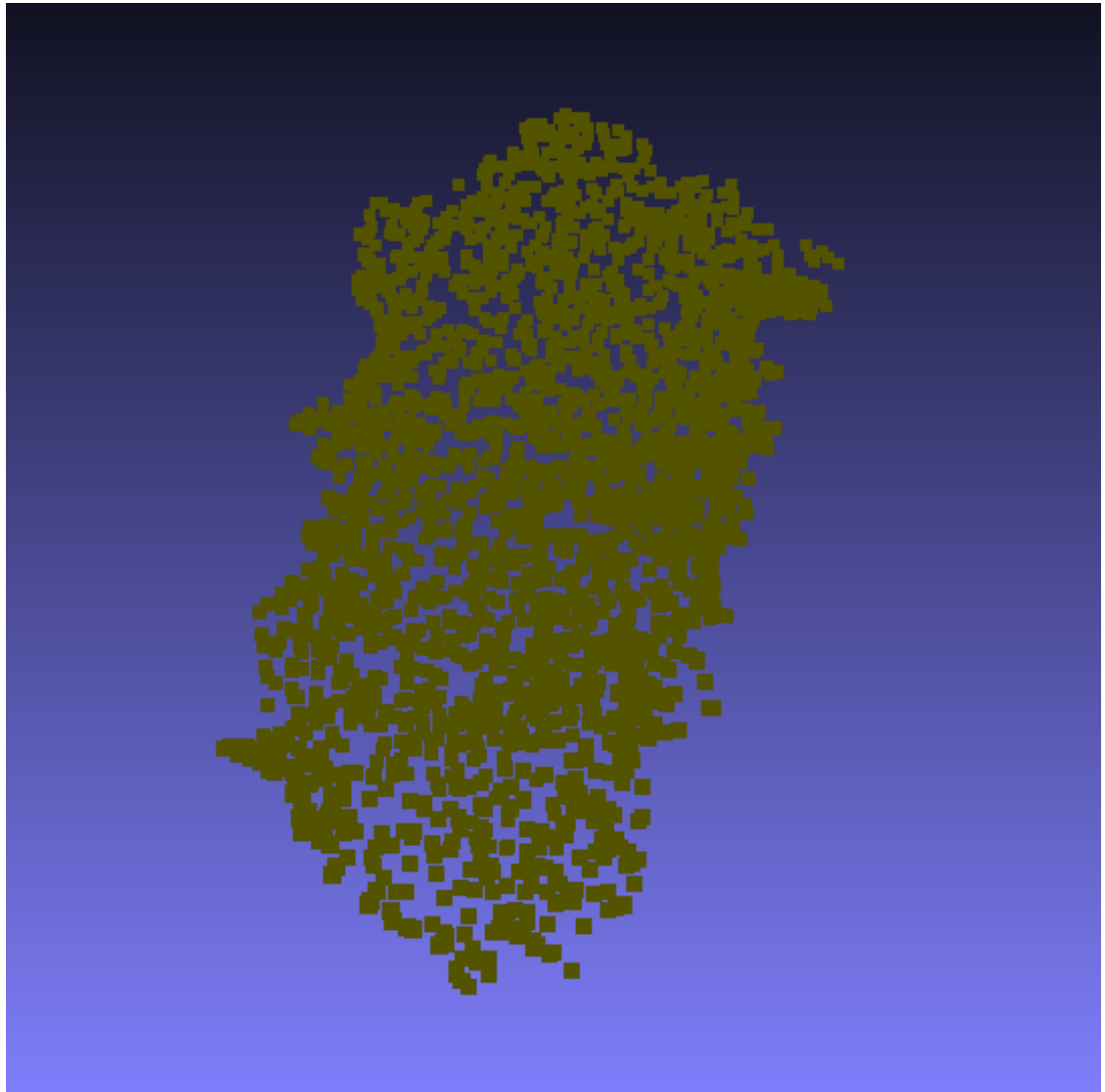


Abb. 32. 3D Punktwolke einer Tabakpflanze

bildung entnommen werden. Die Zielfunktion ist die Wasserstein Metric und die übliche GAN Zielfunktion.

Der Versuchsaufbau ist das Latent-GAN zunächst wird dabei der Autoencoder mit den Trainingsdaten trainiert. Der Encoder des Autoencoders wird mit einer Learning Rate von 0.0005 trainiert. und einer Batch Größe von 50. Der Encoder besteht dabei besteht dabei aus 4 1-D Convolutional Layern mit [64,128,256,1024] Filtern Stride von 1 und Size von 1. Die Aktivierungsfunktion ist relu. Der Letzte Layer ist ein Max Layer. Der Decoder besteht aus [256,256,[614]

3.2 Ergebnis

4 Evaluation und Ergebnisse

5 Zusammenfassung und Diskussion

Ein GAN besteht aus zwei ANN, dem Discriminator und dem Generator. Das Ziel des G ist es Daten zu erzeugen, welche nicht von Trainingsdaten unterschieden werden können. Der Discriminator hat die Aufgabe zu unterscheiden ob der jeweilige Datensatz von G erzeugt wurde und somit ein fake Datensatz ist, oder von den Trainingsdaten stammt (Goodfellow et al., 2014). GANs werden derzeit noch erforscht. Es gibt noch einige offene Fragen, beispielsweise bezüglich der Performance hochauflösender Bilder (Wang et al., 2017). Es wurden in diesem Paper einige Probleme, welche beim Trainieren von GAN auftreten können und mögliche Lösungsansätze, vorgestellt. Es gibt derzeit einige praktische Ansätze, welche in der Anwendung auf GANs zurück greifen. Beispielsweise durch Textbeschreibung eigene Bilder als Output generiert werden (Reed et al., 2016), oder 3D Daten erzeugt werden (Wu, Zhang, Xue, Freeman, & Tenenbaum, 2016b). GANs finden Anwendung in unterschiedlichen Bereichen des Deep Learnings, da sie als Lösung des Problems angesehen werden, dass Neuronale Netzwerke eine hohe Menge an Trainingsdaten benötigen und GANs dieses Problem durch ihre Fähigkeit, neue Daten zu generieren, umgehen. GANs lernen eine Art "versteckte Repräsentation von Klassen, was dazu beitragen kann auch Modelle von komplexen Prozessen zu erlernen. Es gibt erste Ansätze bei denen im Reinforcement Learning durch GANs versucht wird Modelle von der Umwelt eines Agenten zu erlernen, welche dann auf Grundlage dieser Modelle Vorhersagen über zukünftige Ereignisse treffen kann (Pinto, Davidson, Sukthankar, & Gupta, 2017).

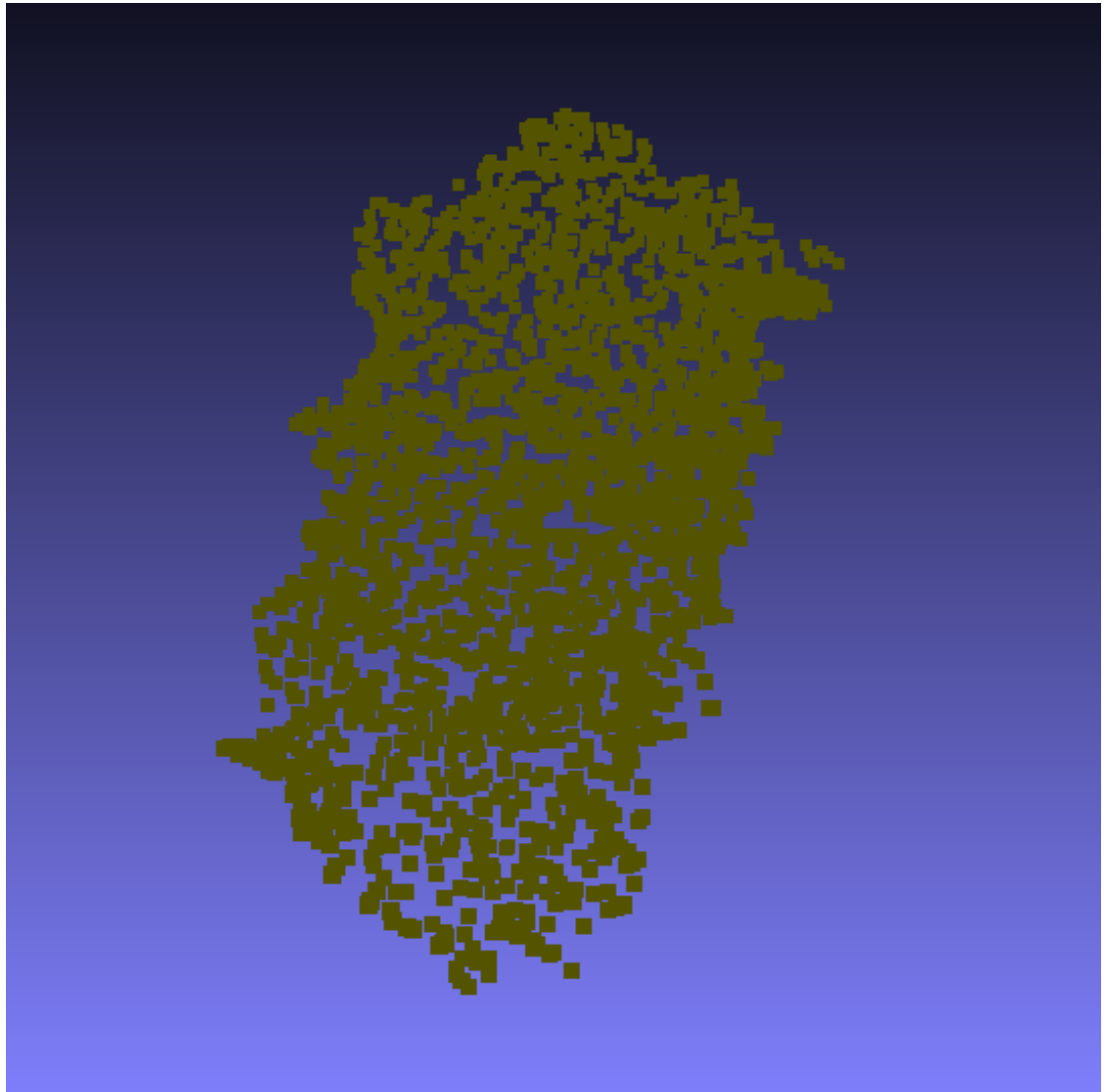


Abb. 33. 3D Punktwolke einer Tabakpflanze

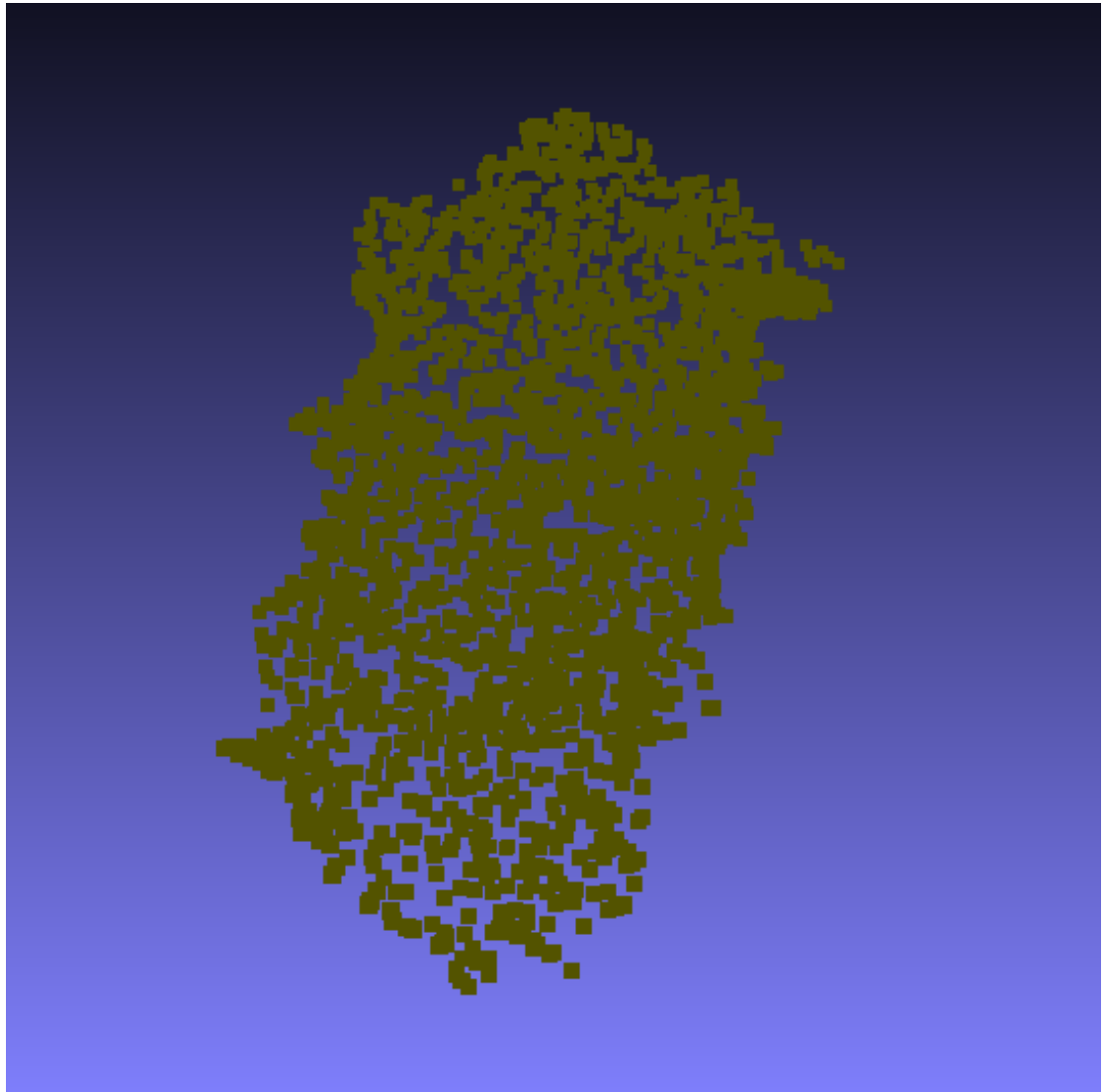


Abb. 34. 3D Punktwolke einer Tabakpflanze

Abbildungsverzeichnis

1	künstliches Neuron	7
2	künstliches neuronales Netzwerk	7
3	Minimum Maximum Saddle Point	8
4	LossFunktion	9
5	aaaa	12
6	Nicht trennbar	14
7	Sigmoid Funktion	15
8	Softmax	15
9	Nicht trennbar	17
10	Sigmoid Funktion	18
11	Softmax	18
12	künstliches Neuron	20
13	künstliches Neuron	21
14	künstliches Neuron	21
15	künstliches Neuron	22
16	künstliches Neuron	23
17	Neuronal Network als Markov Chain	26
18	Convolutional Neural Network	27
19	Convolution Beispiel	28
20	Deconvolution Beispiel	29
21	autoencoder	30
22	Generative Modelle	32
23	Generativ Adversarial Network	33
24	Deep Convolutional GAN	35
25	KL Divergenz und JS Divergenz	36
26	Conditional Adversarial Network	38
27	Conditional Adversarial Network	39
28	Conditional Adversarial Network	40
29	Conditional Adversarial Network	41
30	3D Punktwolke einer Tabakpflanze	42
31	3D Punktwolke einer Tabakpflanze	43
32	3D Punktwolke einer Tabakpflanze	44
33	3D Punktwolke einer Tabakpflanze	46
34	3D Punktwolke einer Tabakpflanze	47

Tabellenverzeichnis

Literaturverzeichnis

- Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein gan. *arXiv preprint arXiv:1701.07875*.
- Dumoulin, V., & Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.
- Golik, P., Doetsch, P., & Ney, H. (2013). Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *Interspeech* (Vol. 13, pp. 1756–1760).
- Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1). MIT press Cambridge.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672–2680).
- Halevy, A., Norvig, P., & Pereira, F. (2009). The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2), 8–12.
- Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2), 251–257.
- Huang, G., Yuan, Y., Xu, Q., Guo, C., Sun, Y., Wu, F., & Weinberger, K. (2018). *An empirical study on evaluation metrics of generative adversarial networks*. Retrieved from <https://openreview.net/forum?id=Sy1f0e-R-Huszár>.
- Huszár, F. (2015). How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *arXiv preprint arXiv:1511.05101*.
- Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.
- Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. *arXiv preprint*.
- Karras, T., Aila, T., Laine, S., & Lehtinen, J. (2017). Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*.
- Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.
- Ng, A. Y., & Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems* (pp. 841–848).
- Pinto, L., Davidson, J., Sukthankar, R., & Gupta, A. (2017). Robust adversarial reinforcement learning. *arXiv preprint arXiv:1703.02702*.
- Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.
- Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. (2016). Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*.
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088), 533.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., & Chen, X. (2016). Improved techniques for training gans. In *Advances in neural information processing systems* (pp. 2234–2242).

Shwartz-Ziv, R., & Tishby, N. (2017). Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*.

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International conference on machine learning* (pp. 1139–1147).

Tishby, N., & Zaslavsky, N. (2015). Deep learning and the information bottleneck principle. In *Information theory workshop (itw), 2015 ieee* (pp. 1–5).

Wang, T.-C., Liu, M.-Y., Zhu, J.-Y., Tao, A., Kautz, J., & Catanzaro, B. (2017). High-resolution image synthesis and semantic manipulation with conditional gans. *arXiv preprint arXiv:1711.11585*.

Wu, J., Zhang, C., Xue, T., Freeman, B., & Tenenbaum, J. (2016a). Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in neural information processing systems* (pp. 82–90).

Wu, J., Zhang, C., Xue, T., Freeman, B., & Tenenbaum, J. (2016b). Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in neural information processing systems* (pp. 82–90).

Zhang, C., Bengio, S., Hardt, M., Recht, B., & Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*.