Gymnasium Bäumlihof, 5Bb

MATURAARBEIT

Kann der Computer Werbung erkennen?

Bilderkennung mit einem Neuronalen Netzwerk

Georg Schwan

Betreuungsperson
Test1
Korreferent
Test2

Datum

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

1.2 Aufbau der Arbeit

Kapitel 2

Neuronales Netzwerk

2.1 Konzept

Wenn man ein normales Programm schreiben will muss man das Problem in viele kleiner aufteilen, bis der Computer fähig ist, es zu lösen. In einem Neuronale Netzwerk wird dem Computer nicht gesagt wie es das Problem lösen kann, sondern ein Neuronales Netzwerk versteht das Problem, indem wir es Beispieldaten geben und es daran lernen kann, bis es seine eigene Lösung gefunden hat. Zum Beispiel, wir wollen einem Netzwerk beibringen ob in einem Bild ein Auto vorkommt, dazu geben wir dem Neuronale Netzwerk viele Bilder, mit und ohne Auto. Mit jedem Bild, dass das Neuronale Netzwerk bekommt, lernt es besser wie ein Auto ausschaut.

Das Konzept eines Neuronales Netzwerk ist nicht etwas neuses. Im Jahre 1957 hat Frank Rosenblatt ein erste Idee eines Neuronales Netzwerk vorgestellt, aber erst in den letzten Jahren ist der grosse Hype ausgebrochen, dies liegt daran, dass man erst jetzt die nötigen Daten und Rechenleistung zu verfügung hat.

2.2 Neuron

User Gehirn kann Entscheidungen treffe, da wir billionen von Neuronen haben, die miteinander verbunden sind und sich verständigen können. Aber ein Neuron an sich ist praktisch nutzlos, aber in grosser Anzahl können sie komplexeste Probleme lösen.

Nach dem gleichen Prinzip funktioniert ein Neuronales Netzwerk Es besteht aus vielen Neuronen (daher der Name) die miteinander verbunden sind.

Ein Neuron in einem Neuronales Netzwerk wird als Mathematische funktion definiert wie Abbildung 2.1 verdeutlicht. Ein Neuron hat n verschiedene Eingaben, die als x_j bezeichnet werden und mit einem spezifischen Gewicht w_j multipliziert werden. Die Ausgabe erfolgt indem man alle gewichteten Eingaben, mit einem Bias b, addiert und durch eine so genannte Aktivierungsfunktion durchlaufen läst. Als Gleichung:

$$y = f\left(\sum_{j=1}^{n} x_j w_j + b\right)$$

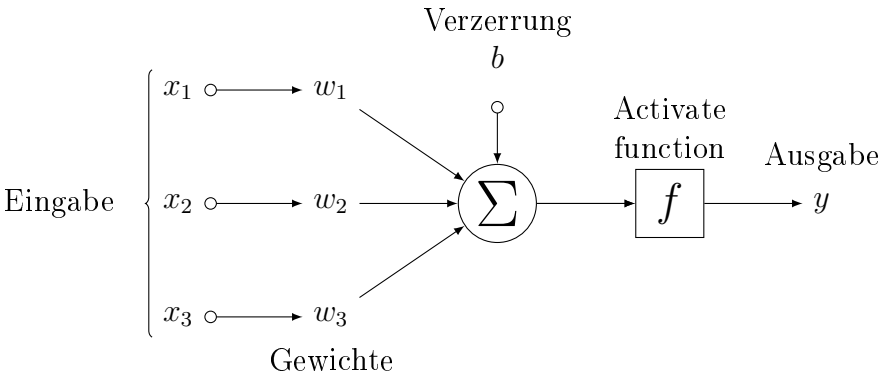

Abbildung 2.1: Einzelner Neuron in einem Neuronalen Netzwerks

Die Gewichte w_j und der Bias b des Neurons sind die Parameter, die angepasst werden und somit das Neuron lernfähig machen.

Quelle: https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f

Eine Aktivierungsfunktion ist nötig, da ohne eine wäre ein Neuronales Netzwerk eine komplett lineare funktion, welches nur lineare Probleme lösen könnte. Die meisten Probleme sind viel komplexer als das man sie linear darstellen könnte und deswegen ist eine Aktivierungsfunktion von nötig. Es wird näher auf die Aktivierungsfunktion eingegangen im Abschnitt 2.5

2.3 Architektur

Wie auch im biologischen Gehirn ist ein Neuron allein nutzlos. Erst wenn man die Neuronen miteinander verbindet kann es komplexe Zusammenhänge modellieren.

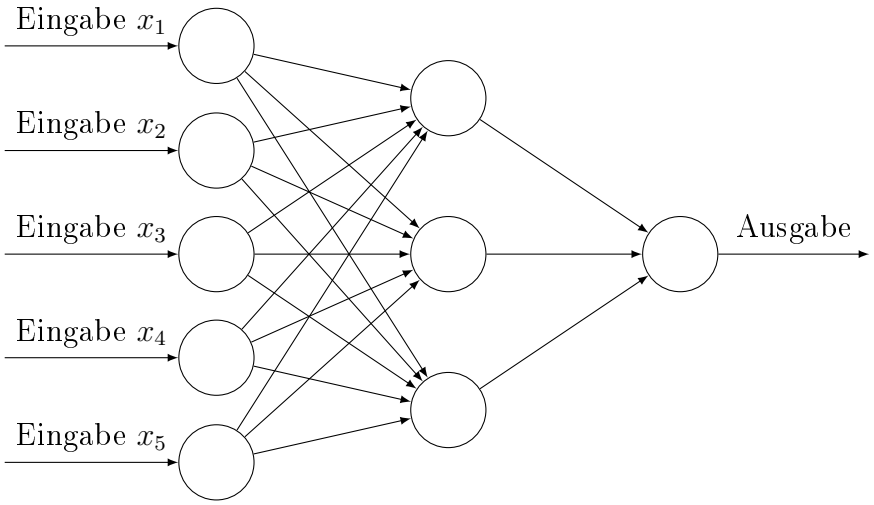

Abbildung 2.2: Mögliche architektur eines Neuronalen Netzwerk

Eine mögliche Architektur kann wie in Abbildung 2.2 ausschauen. Ein Netzwerk wird generell immer in verschiedene Schichten unterteilt. Die linke Schicht wird als eingabe Schicht

bezeichnet und die Neuronen in dieser Schicht werden eingabe Neuronen genannt. Analog da zu wird die rechte Schicht ausgabe Schicht genannt, die die ausgabe Neuronen beinhaltet. Die mittleren Schichten, die von der Anzahl variieren können, werden versteckte Schichten genannt. Die Anzahl der Neuronen in jeder Schicht kann auch variieren. Abbildung 2.3 zeigt eine andere mögliche Architektur für ein Netzwerk, welches 2 versteckte Schichten hat. Jeder Neuron

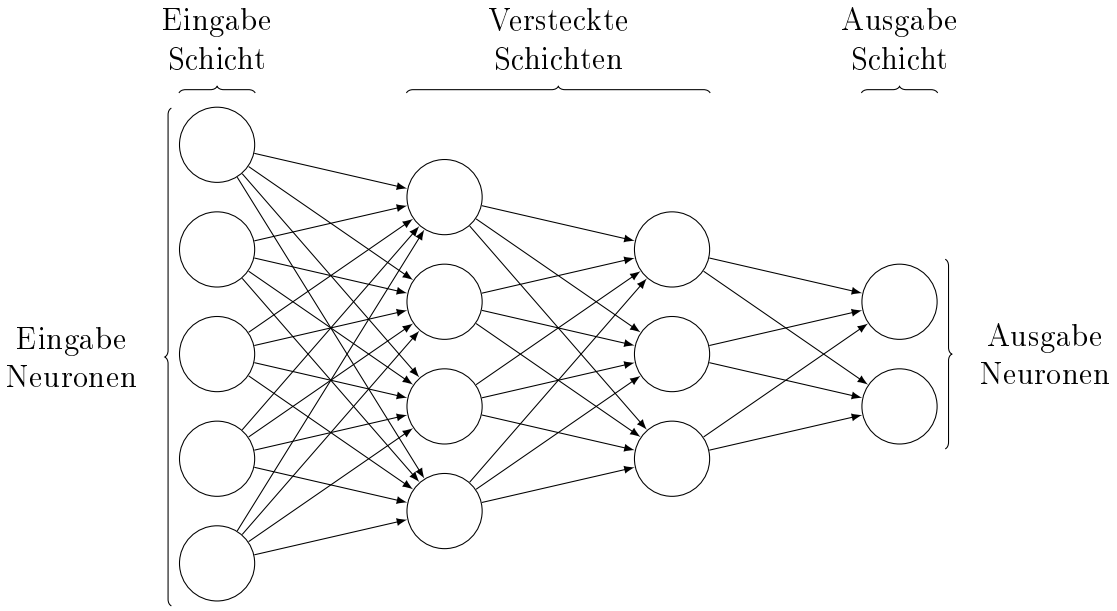

Abbildung 2.3: Neuronales Netzwerk mit 2 versteckten Schichten

von der vorigen Schicht ist mit jedem Neuron der nachfolgenden Schicht verbunden. Dies ist ein klassisches vorwärtsgekoppeltes Netzwerk (im englischen feed forward network). Wichtig zu beachten ist, dass keine Schleifen vorkommen. Es gibt Architekturen in denen Schleifen vorkommen, aber auf diese wird nicht näher eingegangen, da sie für die Bilderkennung nicht weiter relevant sind.

2.3.1 Beschriftung

Um eine allgemeine Gleichung zu bestimmen, muss man zuerst die Elemente des Netzwerks benennen. Wir bezeichnen das Gewicht $w^l_{k,j}$ für die Verbindung des k^{ten} Neuron der $(l-1)^{ten}$ Schicht zu dem j^{ten} Neuron der l^{ten} Schicht. Ähnlich dazu bezeichnen wir die Ausgabe des Neurons als a^l_j und der bias des Neurons als b^l_j, Abbildung 2.4 verdeutlicht diese Notation.

Mit dieser Notation kann eine Gleichung für das Netzwerk aufgestellt werden, welche der Gleichung einem Neuron ähnelt 2.2.

$$a^l_j = f\left(\sum_k a^{l-1}_k w^l_{k,j} + b^l_j\right)$$

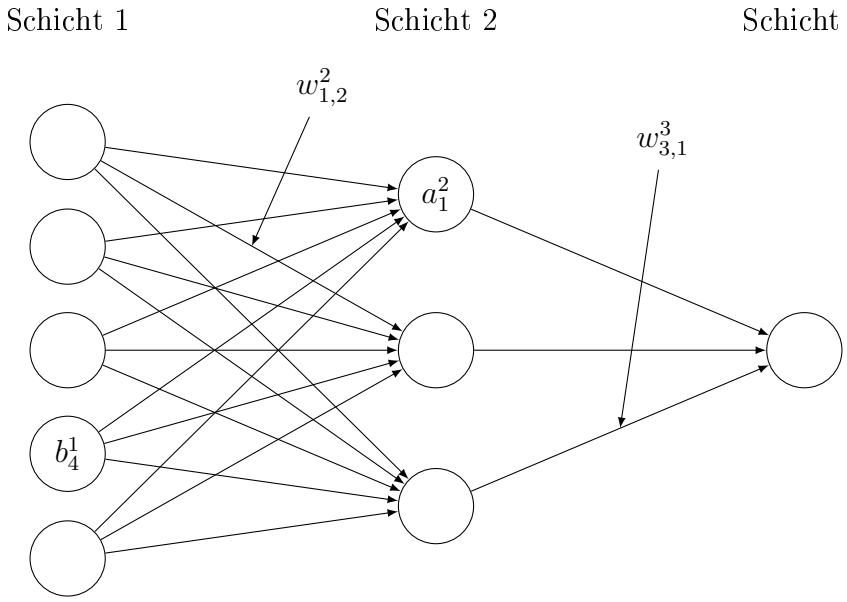

Abbildung 2.4: Bezeichnung der Parameter

2.4 Wie das Netzwerk lernt

Bis jetzt ging es nur darum wie ein Neuronales Netzwerk aufgebaut ist. In dem Abschnitt geht wie ein Neuronales Netzwerk, anhand von Daten, lernen kann

2.4.1 Kostenfunktion

Damit ein Netzwerk lernen kann muss man dem Netzwerk zuerst sagen können wie gut oder wie schlecht es gerade ist. Dazu definieren wir eine Kostenfunktion C, die von allen Gewichten w und allen biases b abhängig ist. Das Netzwerk wird als Funktion $y(x)$ bezeichnet. Den Eingabewert wird als x bezeichnet mit dem dazugehörige Label l. Beachte, dass x und l Vektoren sind. Zum Beispiel würde ein Bild, das 10x10 Pixel gross ist, einen $(10 * 10 =)$100-dimensionaler Vektor haben und das Label, wenn es 3 ausgabe Neuronen gibt, einem 3-dimensional Vektor.

$$C(w, b) = \sum_j (y(x)_j - l_j)^2$$

Das Ziel des Netzwerkes ist diese Kostenfunktion zu minimieren, bis idealerweise $C \approx 0$, dies geschieht wenn die Ausgabe des Netzwerks und des Label ähnlich sind.

2.4.2 Gradient Descent

Um diese Kostenfunktion zu minimieren wird ein Algorithmus names *Gradient Descent* benutzt. Das Konzept basiert darauf, dass man eine Funktion, in abhängigkeit einer Variablen, ableiten kann und so die Steigung and diesem Punkt berechnen kann und die Variable richtung Minimum anpasst.

Bild Einfügen

Zum Beispiel hat man eine Kostenfunktion $C(a, b)$ die von a und b abhängig ist. Wenn man diese Graphisch abbildet, sieht es wie auf Abbildung **ref**.

Die Variablen a und b werden am Anfang zufällige Werte zugeteilt und das Ziel ist sie so anzupassen, dass man das Minimum der Kostenfunktion findet. Um das Minimum zu finden kann man sich einen Ball vorstellen, der in das Minimum herunterrollt. Um dies zu berechnen muss man die Steigung, mithilfe einer Ableitung, herausfinden und die Variable in die gegensätzliche Richtung bewegen, wie auf Abbildung **ref** zu sehen ist.

Bild Einfügen

Für Bewegung ergibt sich:

$$\begin{aligned} a \to a' &= a - \mu \frac{\partial C}{\partial a} \\ b \to b' &= b - \mu \frac{\partial C}{\partial b} \end{aligned}$$

wobei μ eine kleine positive Zahl (learning rate genannt) ist, die die Geschwindigkeit der Bewegung steuert. Ausserdem beachte, dass der Ball keine Beschleunigung hat. Wenn man diese Gleichung nun immer wieder anwendend gelangt man zum Minimum der Kostenfunktion.

Der Algorithmus funktioniert auch bei mehr als nur 2 Variablen und sieht fürs Neuronale Netzwerk identisch aus.

$$\begin{aligned} w^l_{k,j} \to w^{l'}_{k,j} &= w^l_{k,j} - \mu \frac{\partial C}{\partial w^l_{k,j}} \\ b^l_j \to b^{l'}_j &= b^l_j - \mu \frac{\partial C}{\partial b^l_j} \end{aligned}$$

Durch dieses Verfahren kann zwar relativ einfach das Minimum gefunden werden, dabei ist aber zu beachten, dass es nur ein lokales Minimum ist und kein globales.

2.4.3 Backpropagation

Der Algorithmus um $\frac{\partial C}{\partial w^l_{k,j}}$ und $\frac{\partial C}{\partial b^l_j}$ zu berechnen wird als Backpropagation bezeichnet und ist der mathematisch schwerste Teil dieser Arbeit. Es ist aber nicht unbedingt nötig für das Verständnis eines Neuronale Netzwerkes. Es wird auch nicht näher auf die Beweise der Gleichungen eingegangen (oder soll ich???), da es sonst komplizierter wird und im Grunde ist es nur die Anwendung der Kettenregel.

Um die Übersicht zu behalten wird eine Zwischenmenge δ^l_j eingeführt, welches als *Fehler* bezeichnet wird. Der Fehler sagt aus wie gut oder schlecht ein Neuron ist und ist definiert als:

$$\delta^l_j = \frac{\partial C}{\partial z^l_j}$$

wobei z_j^l die Ausgabe von einem Neuron ohne die Aktivierungsfunktion ist, also $a_j^l = f(z_j^l)$. Mit dieser Definition kann man den Fehler in der letzten Schicht L bestimmen:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} f'(z_j^L)$$

In unserem Fall benutzen wir eine Quadratische Kostenfunktion $C = \sum_j (a_j^L - y_j)^2$ bei der die Ableitung $\frac{\partial C}{\partial a_j^L} = 2(a_j^L - y_j)$ ist und können δ_j^L einfacher definieren als:

$$\delta_j^L = 2(a_j^L - y_j) f'(z_j^L)$$

Bei der berechnung des Fehlers δ_j^l Abhängig von δ_j^{l+1} bekommt man:

$$\delta_j^l = \sum_k w_{j,k}^{l+1} \delta_k^{l+1} f'(z_j^l)$$

Beachte, dass bei $w_{j,k}^{l+1}$ das j und k vertauscht sind, so dass man durch alle Neuronen der $(l+1)^{ten}$ Schicht durch iteriert. Mit dieser Gleichung kann man jeden Fehler von jeder Schicht berechnen, indem man von hinten durch das Netzwerk durchläuft. Ähnlich wie wenn man sich beim Netzwerk nach vorne bewegt.

Die Gleichung für die Änderungsrate der Kosten in Bezug auf ein Bias im Netzwerk ist genau der Fehler:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

Die Gleichung für die Änderungsrate der Kosten in Bezug auf ein Gewicht im Netzwerk:

$$\frac{\partial C}{\partial w_{k,j}^l} = a_k^{l-1} \delta_j^l$$

2.5 Aktivierungsfunktionen

Das einzige was noch fehlt ist wie eine Aktivierungsfunktionen genau ausschaut. Wie schon gesagt darf eine Aktivierungsfunktionen nicht linear sein, da sie sonst nichts neues dem Netzwerk beiträgt.

2.5.1 Sigmoid

Ein Beispiel für eine Aktivierungsfunktion ist die Sigmoid Funktion $f(x) = \frac{1}{1+e^{-x}}$, wie sie auf Abbildung 2.5 zu sehen ist. Besonders an dieser Funktion ist, dass sie den Ausgabewert zwischen 0 und 1 eingrenzt, was uns erlaubt den Ausgabewert des ganzen Netzwerkes besser zu deuten, als wenn der Wert zwischen $-\infty$ und ∞ liegt. Ein Problem der Sigmoid Funktion ist, dass wenn die Ausgabe nah bei 1 oder 0 ist, dann ist die Ableitung $f'()$ davon auch nah bei 0, was den Fehler δ_j^l sehr klein hält und so das Netzwerk nur noch langsam lernen lässt. Dieses Problem ist als *Vanishing gradient problem* bekannt.

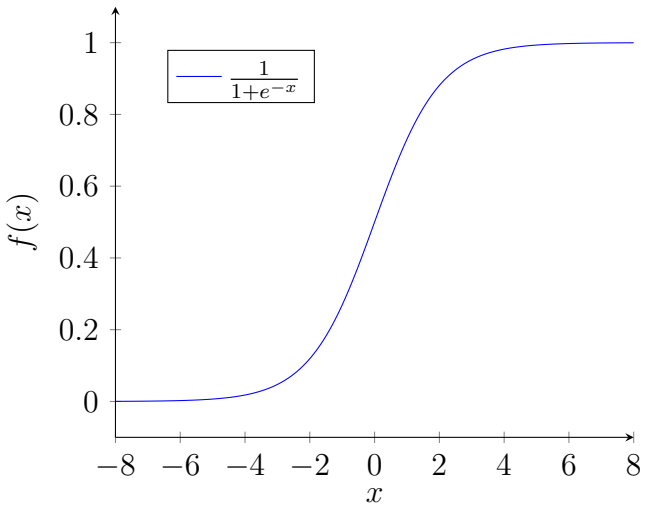

Abbildung 2.5: Sigmoid Aktivierungsfunktionen

2.5.2 Rectified linear units

Eine andere populäre Aktivierungsfunktion ist die Rectified linear units Funktion oder kurz ReLu. Die Funktion $f(x) = \max(0, x)$, wie sie auf Abbildung 2.6 zu sehen ist, löst das Pro-

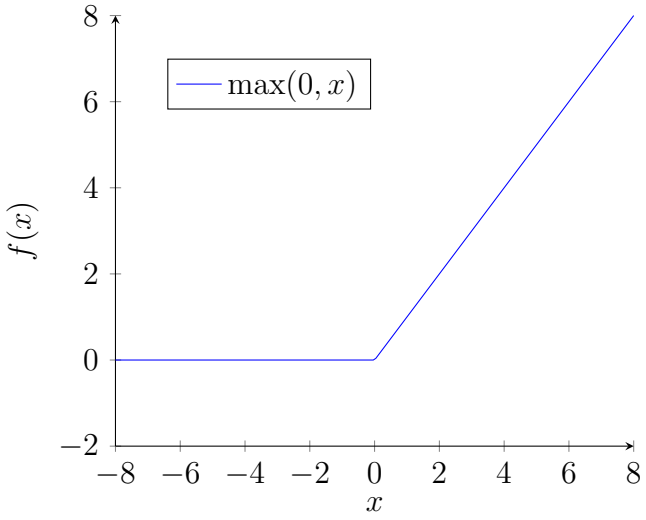

Abbildung 2.6: ReLu Aktivierungsfunktionen

blem des Vanishing gradient und praktisch alle Neuronalen Netzwerke benutzen Relu als ihre Aktivierungsfunktion, da es die besten Ergebnisse erbringt. Ein Nachteil ist, dass sie nur in den Versteckten Schichten gut funktioniert, da der Ausgabewert des Netzwerks in einem unbestimmten Bereich ist.

2.5.3 Softmax

Die Softmax funktion wird verwendet um eindeutige Klassifikationen zu machen und ist definiert als:

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}}$$

Das besondere an dieser Aktivierungsfunktion ist, dass sie nicht nur einen Wert braucht, sondern alle Werte der ganzen Schicht. Ausserdem gibt die Summer aller Resultate von a^L gleich 1 und kann deswegen als eine Wahrscheinlichkeitsverteilung verstanden werden. Dies ist oft sehr hilfreich, da viele Probleme nur ein richtiges Resultat haben, zum Beispiel hat man Bilder von Zahlen, wo immer nur eine Zahl pro Bild zu sehen ist. Und durch die softmax Funktion sieht man dann eine geschätzte Wahrscheinlichkeit vom Netzwerk für jede Zahl.

2.6 Convolution

Bis jetzt ging es nur um Schichten die völlig miteinander verbunden sind. Für die Bilderkennung kann das suboptimal sein, da bestimmte Eigenschaften eines Bildes nicht miteinbezogen werden, wie zum Beispiel die Beziehung von nebeneinander liegenden Pixel und das gesuchte Objekt in einem Bild an verschiedenen Orten vorkommen kann.

2.6.1 Architektur

Die Eingabe für einen convolutional Schicht ist nicht 1-Dimensional, sondern 2-Dimensional. Wie man auf Abbildung 2.7 sieht. Die Neuronen werden normal verbunden, einfach mit dem

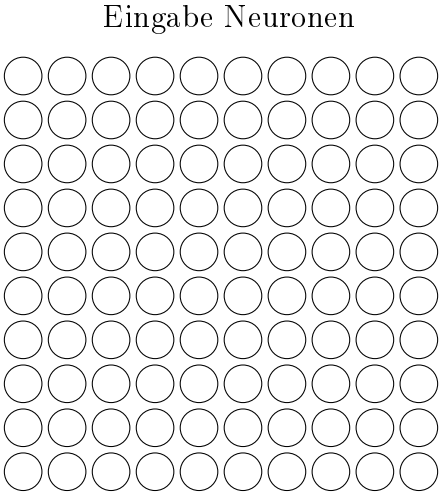

Abbildung 2.7: Eingabe Neuronen für eine convolution Schicht

Unterschied, dass nicht jeder Neuron mit jedem Neuron verbunden wird, sondern dass nur ein bestimmter Bereich zum nächsten Neuron verbunden ist. Dieser Bereich wird als *Filter* bezeichnet und in dem Beispiel auf Abbildung 2.8 wird ein 3x3 Filter benutzt. Der Filter wird dann auf den Eingabe Neuronen um ein Neuron verschoben, um den nächsten Neuron zu verbinden. Und so geht das weiter, auch nach unten, bis die ganze versteckte Schicht gemacht wurde. Dabei wird die versteckte Schicht auch kleiner, in dem Beispiel wird die 10x10 Schicht zu einer 8x8 Schicht, da der Filter irgendwann am anderen Rand anstösst. Abbildung 2.9 verdeutlicht das Prinzip noch einmal.

Der Filter kann auch um mehr als nur einen Neuronen verschoben werden, und man kann

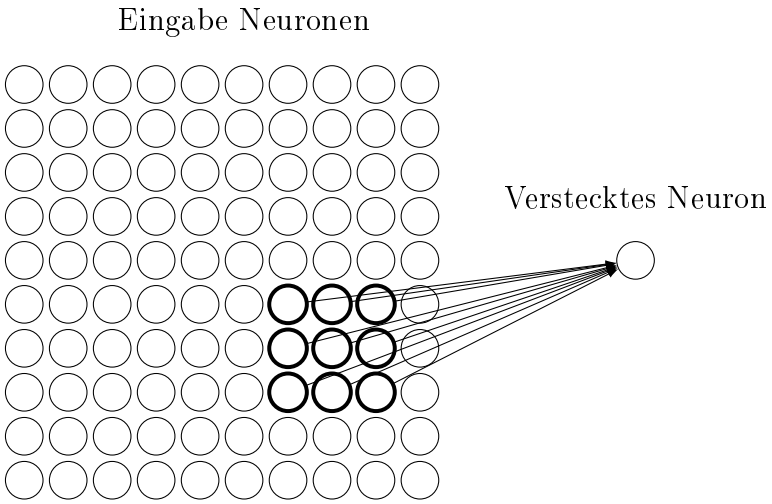

Abbildung 2.8: Verbindung eines versteckten Neurons in einem convolution Schicht

in den beiden Richtungen verschiedene Werte nehmen, zum Beispiel bewegt sich der Neuron nach links um zwei Neuronen und nach unten um drei.

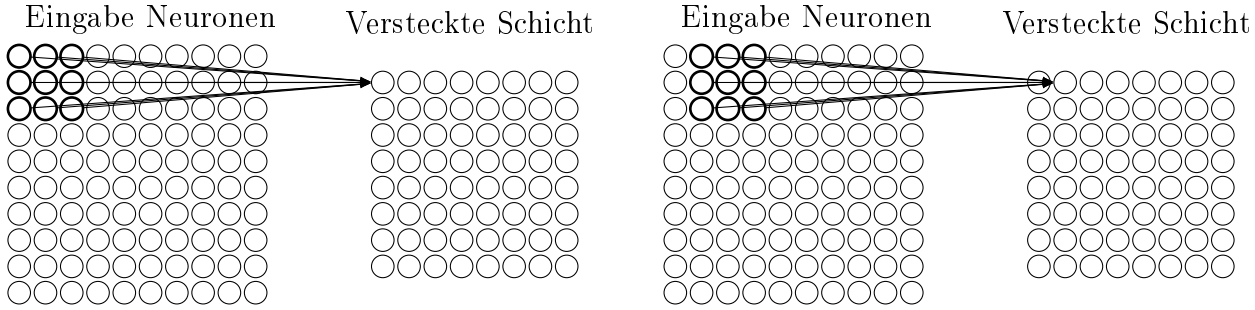

Abbildung 2.9: Bewegung eines Filters über eine convolution Schicht

Das wichtigste am Filter ist, dass er die gleichen Gewichte und Bias verwendet für die Verbindung, d.h bei einem Filter von 5x5 gibt is $(5 * 5 =)25$ verschiedene Gewichte und einen Bias. Wenn nun der Filter bewegt wird werden die gleichen Gewichte und der der gleiche Bias verwendet. Als Gleichung:

$$a^{l+1}_{j,k} = f\left(\sum_{p=0}^{2}\sum_{m=0}^{2} w^{l}_{p,m} a^{l}_{j+p,k+m} + b^{l}\right)$$

Kapitel 3

Lösungsansatz

Kapitel 4

Umsetzung

Kapitel 5

Reflexion

Abbildungsverzeichnis

Ehrlichkeitserklärung

Die eingereichte Arbeit ist das Resultat meiner persönlichen, selbstständigen Beschäftigung mit dem Thema. Ich habe für sie keine anderen Quellen benutzt als die in den Verzeichnissen aufgeführten. Sämtliche wörtlich übernommenen Texte (Sätze) sind als Zitate gekennzeichnet.

Insert Datum