

Machine Learning 3: Neuronale Netze III

Ergänzungsfach Informatik, 2021/2022, pro@kswe.ch

20. November 2021

1 Künstliche neuronale Netze¹

Biologische Gehirne führen anspruchsvolle Aufgaben wie fliegen, Nahrung finden, Sprachen lernen und vor Feinden fliehen scheinbar mühelos aus, obwohl sie offensichtlich weniger Speicherkapazität besitzen und wesentlich langsamer laufen als moderne Computer. Biologische Gehirne sind zudem unglaublich robust gegenüber beschädigten und unvollkommenen Signalen, verglichen mit herkömmlichen Computersystemen. Wir möchten nun den nächsten Schritt machen und ein biologisches Gehirn, welches aus **verknüpften Neuronen** besteht, als Inspiration nehmen für künstliche neuronale Netze.

Abbildung 1 zeigt nochmals die Grundeinheit eines biologischen Gehirns - das Neuron.

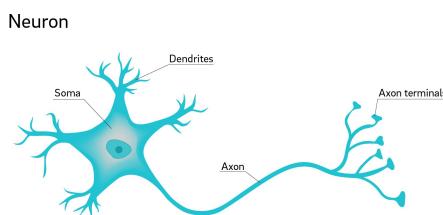


Abbildung 1: Neuron mit Dendriten, Axon und Terminalen.

Die elektrischen Signale werden von Dendriten gesammelt und diese wirken zusammen, um ein stärkeres elektrisches Signal zu bilden. Wenn das Signal stark genug ist, um die Schwelle zu überwinden, feuert das Neuron ein Signal entlang des Axons zu den Terminalen, um es an die Dendriten der nächsten Neuronen weiterzugeben. Abbildung 2 zeigt, wie mehrere Neuronen auf diese Weise miteinander verbunden sind. Es zeigt sich, dass jedes Neuron die Signale von vielen Neuronen vor ihm aufnimmt. Die Stelle an denen sich zwei Neuronen verbinden wird **Synapse** genannt. Auch die Ausgangssignale werden an viele weitere Neuronen weitergegeben, falls das Neuron feuert.

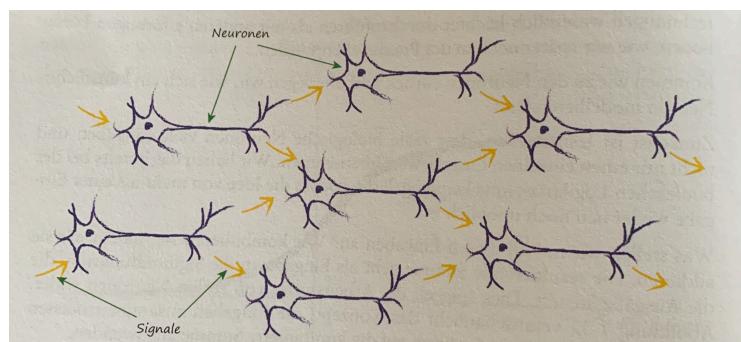


Abbildung 2: Mehrere, miteinander verbundene Neuronen.

Um dieses natürliche Vorbild in einem künstlichen Modell nachzubilden, verwendet man mehrere Schichten von Neuronen, die jeweils zu jedem anderen in der vorhergehenden und nachfolgenden Schicht verbunden sind. Abbildung 3 zeigt ein Beispiel.

¹Auszug aus dem Buch “Neuronale Netze selbst programmieren”, Tariq Rashid”

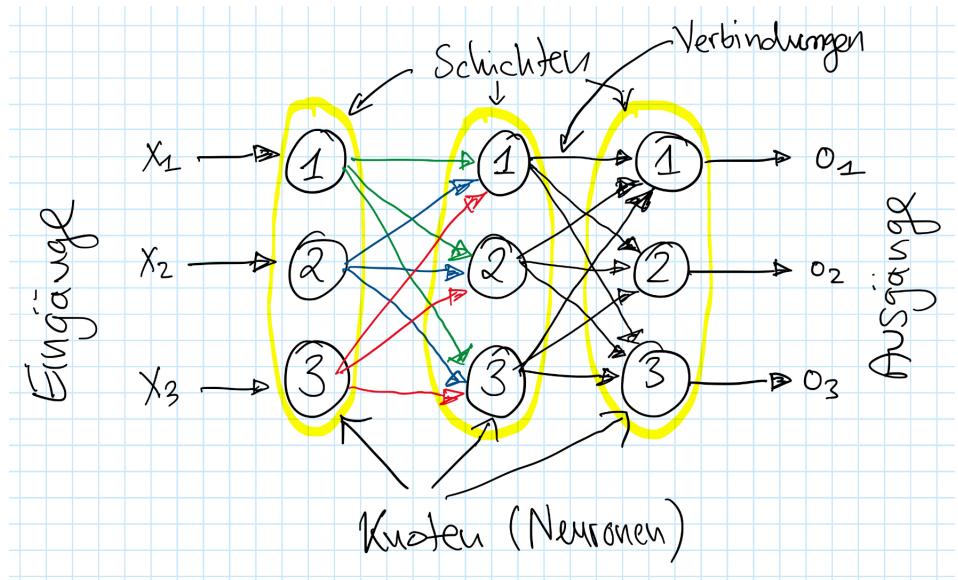


Abbildung 3: Ein künstliches neuronales Netz mit drei Schichten.

Das künstliche neuronale Netz aus Abbildung 3 besteht aus drei Schichten, die jeweils drei künstliche Neuronen enthalten. Der Fachbegriff für ‘‘Neuronen’’ im künstlichen neuronalen Netz (KNN) lautet **Knoten**. Es gibt also neun Knoten in diesem KNN. Außerdem ist jeder Knoten mit jedem anderen Knoten in den vorhergehenden und nachfolgenden Schichten verbunden.

Ähnlich wie beim ‘‘einfachen’’ Neuron möchten wir die Stärke der Verbindungen zwischen den Knoten anpassen können. Wir führen für jede Verbindung ein **Gewicht** ein. Ein geringes Gewicht soll das Signal von einem Knoten zum nächsten Knoten abschwächen. Ein hohes Gewicht soll das Signal verstärken. Abbildung 4 verfeinert das Beispiel KNN aus Abbildung 3.

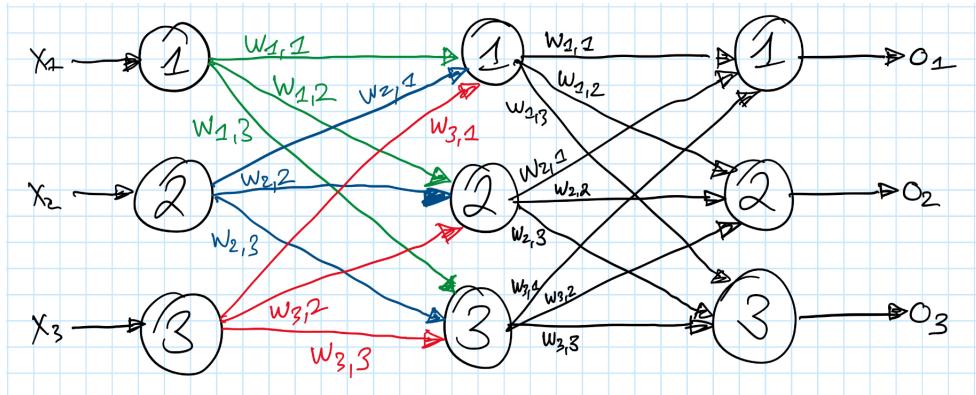


Abbildung 4: Jede Verbindung zwischen zwei Knoten besitzt ein Gewicht.

Wir kürzen die Gewichte mit w (eng. weight) ab. Das Gewicht $w_{2,3}$ ist der Verbindung zugeordnet, die von Knoten 2 in der einen Schicht zu Knoten 3 in der nächsten Schicht existiert. Zwischen zwei Schichten gibt es für jede Verbindung ein Gewicht. Zwischen Schicht 1 und Schicht 2 existiert ein Gewicht $w_{1,1}$ und zwischen Schicht 2 und Schicht 3 existiert ein Gewicht $w_{1,1}$. Diese beiden Gewichte müssen, trotz gleicher Benennung, nicht identisch sein. Für das KNN aus Abbildung 4 gibt es somit 18 unterschiedliche Gewichte.

Berechtigterweise können Sie dieses Design infrage stellen und wissen wollen, warum jeder Knoten mit jedem anderen Knoten in der vorherigen und in der nächsten Schicht verbunden sein sollte. Tatsächlich müssen die Knoten nicht alle verbunden sein; man könnte sehr kreativ sein, was die konkreten Verbindungen angeht. Bei einer vollständigen Verknüpfung ist es aber in der Praxis leichter, die Verbindungen einheitlich in einem Programm zu beschreiben. Selbst wenn es mehr Verbindungen gibt, als für ein spezifisches Machine Learning-Problem absolut nötig sind, schlagen die überflüssigen Verbindungen kaum zu Buche. Der Lernprozess schwächt diese zusätzlichen Verbindungen ab, falls sie tatsächlich nicht benötigt werden. Dies bedeutet, dass manche Gewichte zu null oder fast zu null

werden, während das KNN mit den Trainingsdaten lernt. Trainingsdaten werden bei einem Gewicht von 0 quasi nicht weitergeleitet.

1.1 Architektur eines KNN

Die Architektur eines KNN bestimmt die Anzahl der Schichten und die Anzahl der Knoten pro Schicht. Dies kann je nach Aufgabe unterschiedlich sein. Abbildung 5 zeigt ein KNN mit drei Schichten zu je drei Knoten. Die erste Schicht wird typischerweise als **Input Layer** (dt. Eingabeschicht) bezeichnet. Sie repräsentiert die Eingabe der Daten. Es sind hier keine mathematischen Berechnungen durchzuführen. Der **Output Layer** (dt. Ausgabeschicht) stellt das Ergebnis der Verarbeitung des KNN dar. Die Werte der Knoten für den Output Layer werden für die Klassifizierung benutzt.

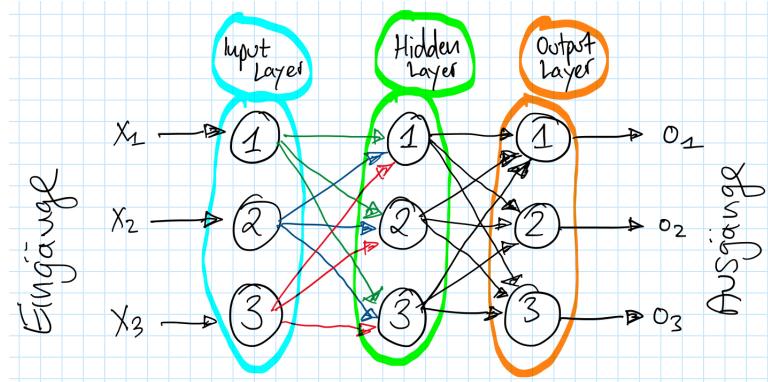


Abbildung 5: Es kann mehrere Hidden Layer geben.

Alle Schichten zwischen dem Input Layer und dem Output Layer werden **Hidden Layer** (dt. versteckte Schicht) genannt. Das mag geheimnisvoll und dunkel klingen, doch leider gibt es dafür keinen geheimnisvollen dunklen Grund. Der Name hat sich einfach gehalten, weil die Ausgänge der mittleren Schicht nicht unbedingt als Ausgänge in Erscheinung treten, also "verborgen" oder "versteckt" sind. Das ist zwar wenig überzeugend, doch es gibt tatsächlich keine bessere Begründung für den Namen.

2 Grundlegender Lernprozess von neuronalen Netzen²

Ein KNN lernt Samples zu klassifizieren durch ein iteratives Verfahren. Trainingsdaten werden "durch" das KNN "bewegt", um eine Ausgabe zu erzeugen. Die Ausgabe wird dann mit der tatsächlichen Ausgabe verglichen. Bei einem Fehler in der Ausgabe werden die Gewichte angepasst und das nächste Sample zum Lernen herangezogen. Aus einer grossen Menge von Trainingsdaten extrahiert das KNN im Lernprozess dann die charakteristischen Eigenschaften der Daten. Diese Eigenschaften der Daten werden durch die Gewichte abgebildet, mit der Hoffnung, dass zukünftige Daten damit ebenfalls korrekt klassifiziert werden. Das Lernen in einem KNN entspricht dabei nicht wirklich dem Lernprozess eines biologischen Gehirns. Es sind mathematische Operationen, welche die Gewichte schrittweise bestimmen. Abbildung 6 zeigt ein Beispiel für die Architektur und den Lernprozess bei der Erkennung von Bildern.

Da ein KNN typischerweise mit Zahlen operiert, müssen die Samples aus dem Daten-Set für das KNN vorbereitet werden. Für das Beispiel aus Abbildung 6 bedeutet dies zum Beispiel, dass die 10×10 Pixel Bilder in einen Vektor mit Zahlen überführt werden müssen. Ein Eintrag im Vektor entspricht dann einem Farbwert. Die Klassen (das Label) für die Samples müssen ebenfalls in Zahlen überführt werden. Im Beispiel aus Abbildung 6 sollen vier verschiedene Buchstaben erkannt werden. Wir können dazu vier verschiedene Vektoren verwenden, wobei jeweils ein Eintrag auf 1 gesetzt wird und alle anderen auf 0.

$$\vec{c}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \vec{c}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad \vec{c}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad \vec{c}_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

²Quelle: <https://sebastiandoern.de/neuronale-netze/>

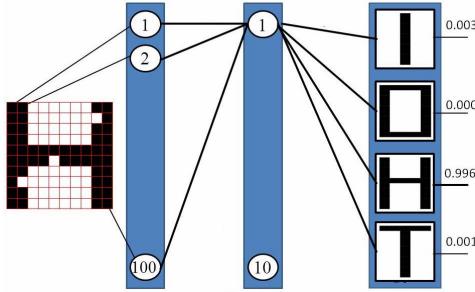


Abbildung 6: Ein dreilagiges Netz zur Klassifizierung von Eingabebildern aus 10×10 Pixeln. Jedes Bild wird in einen 100 dimensionalen Vektor überführt. Der Vektor wird an die Eingabeschicht mit 100 Knoten übergeben. Die verdeckte Schicht besitzt 10 Knoten. Die Ausgabeschicht besitzt vier Knoten. Jeder Knoten repräsentiert ein Buchstabe, den das KNN erkennen soll. In diesem Fall wird das verrauschte Bild des Buchstabens H zu 99.6% als H, zu 0.3% als I und zu 0.1% als T erkannt.

Die Zuordnung von Klasse zu Vektor wird wie folgt vorgenommen:

- Buchstabe I ergibt \vec{c}_1
- Buchstabe O ergibt \vec{c}_2
- Buchstabe H ergibt \vec{c}_3
- Buchstabe T ergibt \vec{c}_4

Dadurch haben wir sowohl die Eingabe als auch die Ausgabe durch Zahlen repräsentiert, die das KNN dann verarbeiten kann.

3 Klassifizieren

Die Architektur eines KNN sieht erstaunlich aus, ist aber auch ein wenig abschreckend und scheint viel Arbeit zu bedeuten. Wir möchten jedoch trotzdem den Aufwand betreiben, das Verarbeitungskonzept im Detail zu verstehen. Damit können wir erfahren, was tatsächlich in einem KNN passiert, selbst wenn wir später die ganze Arbeit vom Computer erledigen lassen. Deshalb versuchen wir, die Abläufe mit einem kleineren neuronalen Netz herauszuarbeiten, das nur aus zwei Schichten besteht, die jeweils zwei Knoten enthalten. Abbildung 7 zeigt das KNN mit den Beispielwerten.

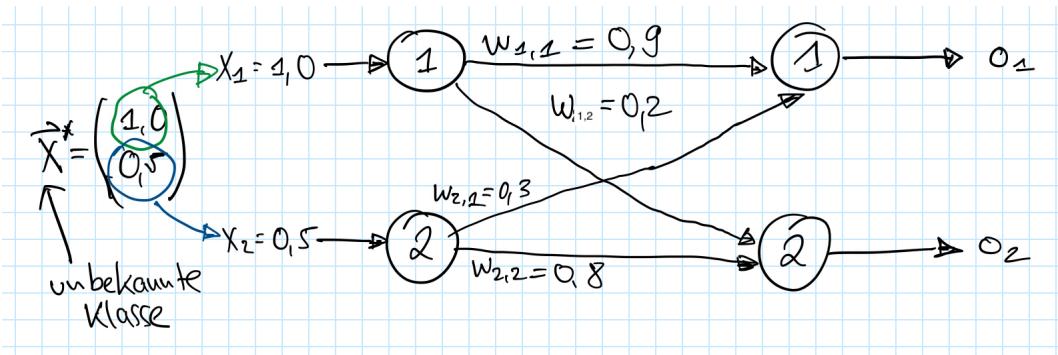


Abbildung 7: Die Gewichte wurden zufällig gewählt. o_1 und o_2 bezeichnen die Werte des Output Layers.

Wir möchten zunächst verstehen, wie man mit einem KNN eine Klassifikation für ein Sample vornehmen kann. Dies ist der einfachere Teil eines KNNS. Im zweiten Schritt setzen wir uns dann damit auseinander, wie man das KNN trainieren kann, das heisst wie man die Gewichte verbessert. Wir gehen also nun davon aus, dass die Gewichte gegeben sind und ein Sample \vec{x}^* klassifiziert werden soll. Es gibt zwei Klassen $c_1 = \text{rot}$ und $c_2 = \text{blau}$. Die Klassen wurden für das Training wie folgt in Zahlen überführt:

$$\vec{c}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \vec{c}_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Die Zuordnung von Klasse zu Vektor wird wie folgt vorgenommen:

- rot ergibt \vec{c}_1
- blau ergibt \vec{c}_2

Wie versuchen nun zu klären, wie das KNN die Klasse für $\vec{x}^* = \begin{pmatrix} 1,0 \\ 0,5 \end{pmatrix}$ bestimmt.

3.1 Input Layer (Schicht 1)

Die erste Schicht der Knoten ist der Input Layer. Ihr fällt lediglich die Aufgabe zu, die Eingaben darzustellen. Das heisst, die Eingabeknoten führen keine Berechnungen durch. Diese Darstellung hängt mit der Entwicklungsgeschichte der KNN zusammen und hat sonst keinen tieferen Grund.

3.2 Output Layer (Schicht 2)

Das KNN aus Abbildung 7 hat **keine** Hidden Layer. Es besteht "nur" aus einer Eingabeschicht und einer Ausgabeschicht. Zwischen diesen beiden Schichten finden nun die Berechnungen statt. Die Knoten der Eingabeschicht leiten die Werte mit den Gewichten als Eingabe zu den Knoten der Ausgabeschicht. Dort wird in jedem Knoten die Summe der gewichteten Eingaben berechnet und die Aktivierungsfunktion eingesetzt. Diese Berechnung entspricht dem Prinzip eines einfachen Neurons aus dem vorherigen Kapitel. Abbildung 8 zeigt die Berechnungen für einen Knoten.

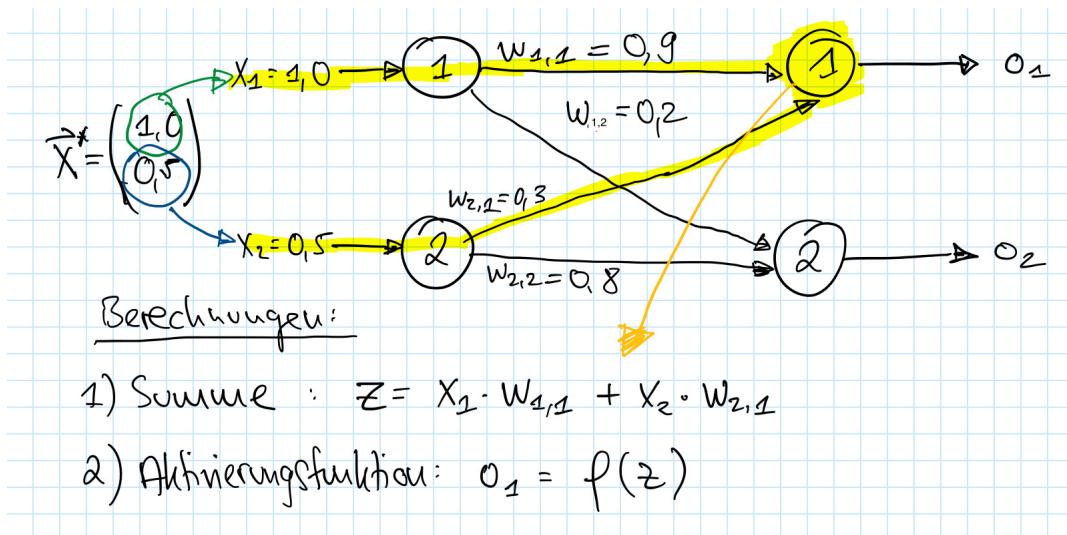


Abbildung 8: Die Berechnungen im ersten Knoten in der zweiten Schicht (Ausgabeschicht).

Wir müssen also die Berechnungen für beide Knoten ausführen, damit wir die Ausgabewerte o_1 und o_2 erhalten.

3.3 Aktivierungsfunktion φ

Welche Aktivierungsfunktion verwenden wir für die Berechnungen? Wir können auf die Stufenfunktion aus dem letzten Kapitel zurückgreifen, jedoch lässt sich die Wahl der Aktivierungsfunktion noch verbessern. Abbildung 9 zeigt die **Sigmoidfunktion**. Sie verläuft sanfter als die abrupte Stufenfunktion, was sie natürlicher und realistischer macht ("die Natur macht keine Sprünge").

Eine solche sanfte s-förmige Sigmoidfunktion werden wir fortan für unsere eigenen neuronalen Netze verwenden. Forscher auf dem Gebiet der künstlichen Intelligenz nehmen auch andere, ähnliche aussehende Funktionen, die Sigmoidfunktion ist aber einfach und tatsächlich sehr gebräuchlich, sodass wir uns in guter Gesellschaft befinden. Die Sigmoidfunktion lässt sich wie folgt beschreiben:

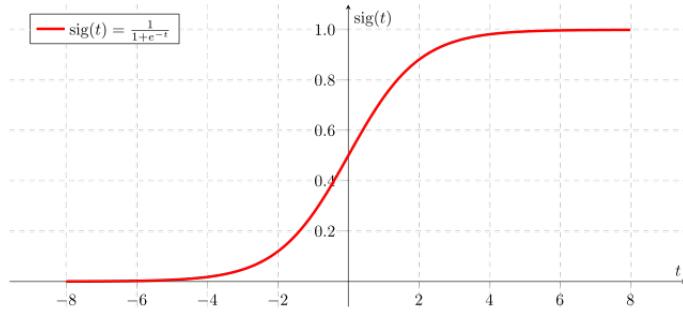


Abbildung 9: Die Funktionswerte $f(x)$ der Sigmoidfunktion befinden sich zwischen 0 und 1, werden jedoch nie erreicht. Das Bild verwendet statt x die Variable t , das heisst, es gilt $f(x) = f(t) = \text{sig}(t)$.

$$f(x) = \frac{1}{1+e^{-x}}$$

Mit e wird hier die **Eulersche Zahl** ($2,71828\dots$) bezeichnet. Es gibt noch einen weiteren sehr wichtigen Grund für die Wahl der Sigmoidfunktion gegenüber den unzähligen anderen s-förmigen Funktionen, die wir für die Ausgabe eines Knotens verwenden könnten: Mit der Sigmoidfunktion sind Berechnungen wesentlich leichter durchzuführen, da für die Funktion eine Ableitung existiert.

3.4 Beispielrechnungen

Wir berechnen nun o_1 und o_2 mit der Sigmoidfunktion als Aktivierungsfunktion.

$$\begin{aligned} z &= x_1 \cdot w_{1,1} + x_2 \cdot w_{2,1} = 1,0 \cdot 0,9 + 0,5 \cdot 0,3 = 1,05 \\ o_1 &= \varphi(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-1,05}} = 0,7408 \end{aligned}$$

$$\begin{aligned} z &= x_1 \cdot w_{1,2} + x_2 \cdot w_{2,2} = 1,0 \cdot 0,2 + 0,5 \cdot 0,8 = 0,6 \\ o_2 &= \varphi(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-0,6}} = 0,6457 \end{aligned}$$

Da die Ausgabe des Knotens o_1 grösser ist als die Ausgabe des Knotens o_2 wird der Sample \vec{x}^* als c_1 klassifiziert, das heisst rot.

3.5 Propagation

Das Berechnen der Ausgaben wird als **Vorwärtsschritt** (eng. **Forward Propagation**) bezeichnet, da die Daten von links nach rechts “durch” das KNN “bewegt” werden. Beim Trainieren des KNN werden wir durch die Ausgaben die Fehler berechnen und damit die Gewichte anpassen. Dieser Schritt wird **Fehlerrückführung** (eng. **Backpropagation of Error**) genannt. Abbildung 10 visualisiert die beiden Prinzipien.

3.6 Matrizen verwenden

Das Berechnen ist recht viel Arbeit. Bereits bei nur zwei Ausgaben von einem sehr vereinfachten KNN. Für ein grösseres KNN möchte man dies auf keinen Fall von Hand durchführen. Auch das eintippen der Anweisungen auf dem Computer erfordert viel Schreibarbeit. Es ist mühselig und fehleranfällig. Wir können jedoch die Berechnungen durch Matrixmultiplikationen ausdrücken.

3.6.1 Gewichtsmatrix

Wir können die Gewichte als Gewichtsmatrix W notieren. Zwischen zwei Schichten gibt es jeweils eine Gewichtsmatrix. Der Aufbau ist wie folgt:

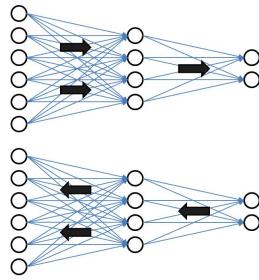


Abbildung 10: Beim Klassifizieren werden die Daten ‘‘nur’’ von links nach rechts ‘‘bewegt’’. Beim Training werden danach zuerst von links nach rechts bewegt, dann werden die Fehler berechnet und anschliessend werden die Gewichte von rechts nach links angepasst.

$$W = \begin{pmatrix} w_{1,1} & w_{2,1} & \cdots & w_{n,1} \\ w_{1,2} & w_{2,2} & \cdots & w_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1,k} & w_{2,k} & \cdots & w_{n,k} \end{pmatrix}$$

In der ersten Spalte werden die Gewichte vom ersten Knoten der einen Schicht zu allen anderen Knoten der anderen Schicht notiert. In der zweiten Spalte werden die Gewichte vom zweiten Knoten der einen Schicht zu allen anderen Knoten der anderen Schicht notiert. Dies geschieht für n Knoten der einen Schicht und k Knoten der anderen Schicht.

3.6.2 Beispiel

Wir drücken die Gewichte für das Beispiel von oben als Matrix aus. Es gibt zwei Knoten in der Eingabeschicht und zwei Knoten in der Ausgabeschicht. Wir erhalten folgende Matrix:

$$W_{\text{input_output}} = \begin{pmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \end{pmatrix} = \begin{pmatrix} 0,9 & 0,3 \\ 0,2 & 0,8 \end{pmatrix}$$

Wir verwenden für die Gewichtsmatrizen immer ein Subskript, damit deutlich wird zwischen welchen beiden Schichten die Gewichte eingesetzt werden. Mit $W_{\text{input_output}}$ sind also die Gewichte zwischen der Eingabeschicht und der Ausgabeschicht gemeint.

3.6.3 Eingabevektor

Der Sample, welcher klassifiziert werden soll, kann durch einen Vektor dargestellt werden. Wir können als Bezeichnung entweder \vec{x} verwenden oder \vec{i} für Input Layer. Wir müssen keine Berechnung durchführen. Bei drei Schichten gibt es nach den Berechnungen zwischen der Eingabeschicht und der Ausgabeschicht ein Ergebnisvektor. Dieser Ergebnisvektor dient als Eingabe für die Berechnungen zwischen der versteckten Schicht und der Ausgabeschicht. Meist wird der Ergebnisvektor ebenfalls mit \vec{i} bezeichnet, da er den Input für die nächste Schicht darstellt.

3.6.4 Beispiel

Der Eingabevektor aus dem Beispiel von oben lässt sich wie folgt darstellen:

$$\vec{i}_{\text{input}} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1,0 \\ 0,5 \end{pmatrix}$$

3.6.5 Vorwärtsschritt als Vektor-Matrixmultiplikation

Wir können die Ausgabe einer Schicht nun durch folgende Vektor-Matrixmultiplikation ausdrücken:

$$\vec{z} = W \cdot \vec{i}$$

Die Summen werden durch die Gewichtsmatrix W multipliziert mit dem Eingabevektor \vec{i} ermittelt und durch den Vektor \vec{z} repräsentiert. Die Aktivierungsfunktion wird dann auf **jedes Element** des Vektors \vec{z} angewendet.

$$\vec{o} = \text{sigmoid}(\vec{z})$$

Mit $\text{sigmoid}(\vec{z})$ ist gemeint, dass wir die Sigmoidfunktion auf jedes Element anwenden. **Dies muss nicht nur für den Output Layer geschehen, sondern auch für jeden Hidden Layer.** Für die Klassifizierung müssen wir dann nur das grösste Element im Vektor \vec{o} bestimmen.

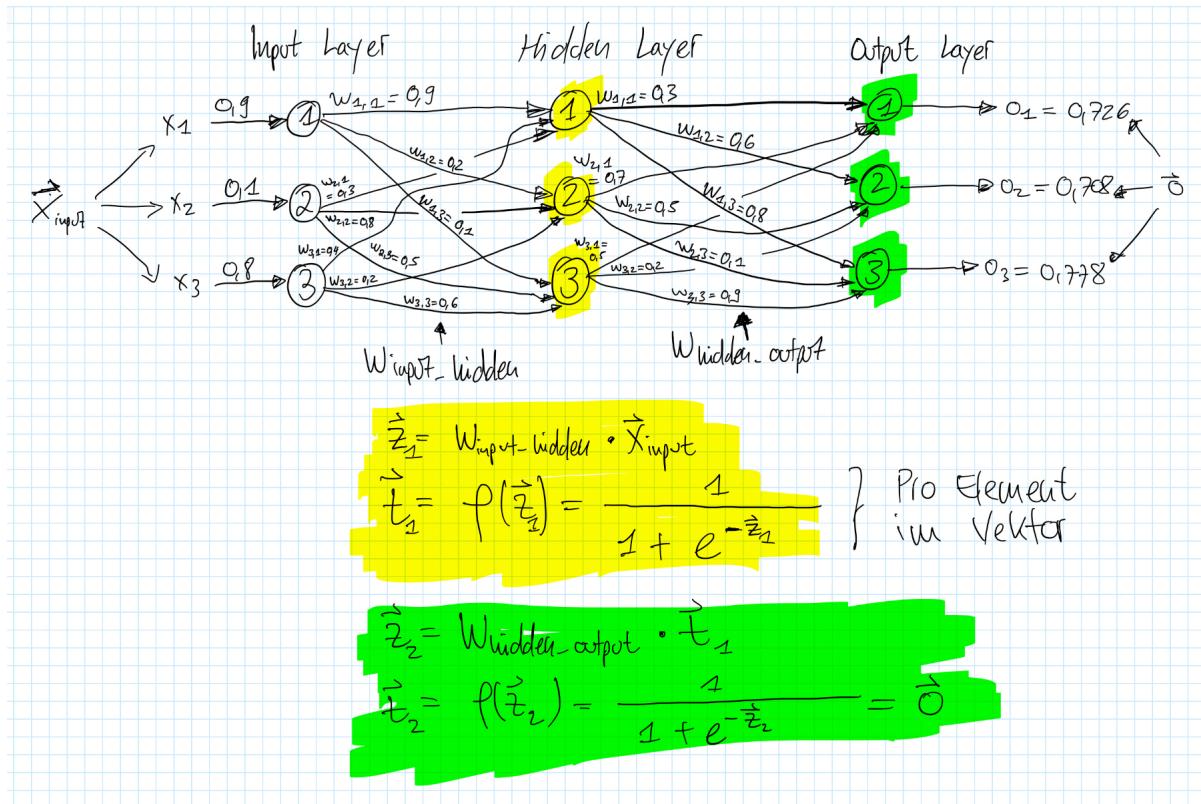
4 Übungen

- Stellen Sie das KNN mit drei Schichten grafisch dar. Verwenden Sie die korrekten Fachbegriffe und folgende Daten:

$$\vec{x}_{\text{input}} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0,9 \\ 0,1 \\ 0,8 \end{pmatrix} \quad \vec{o} = \begin{pmatrix} 0,726 \\ 0,708 \\ 0,778 \end{pmatrix}$$

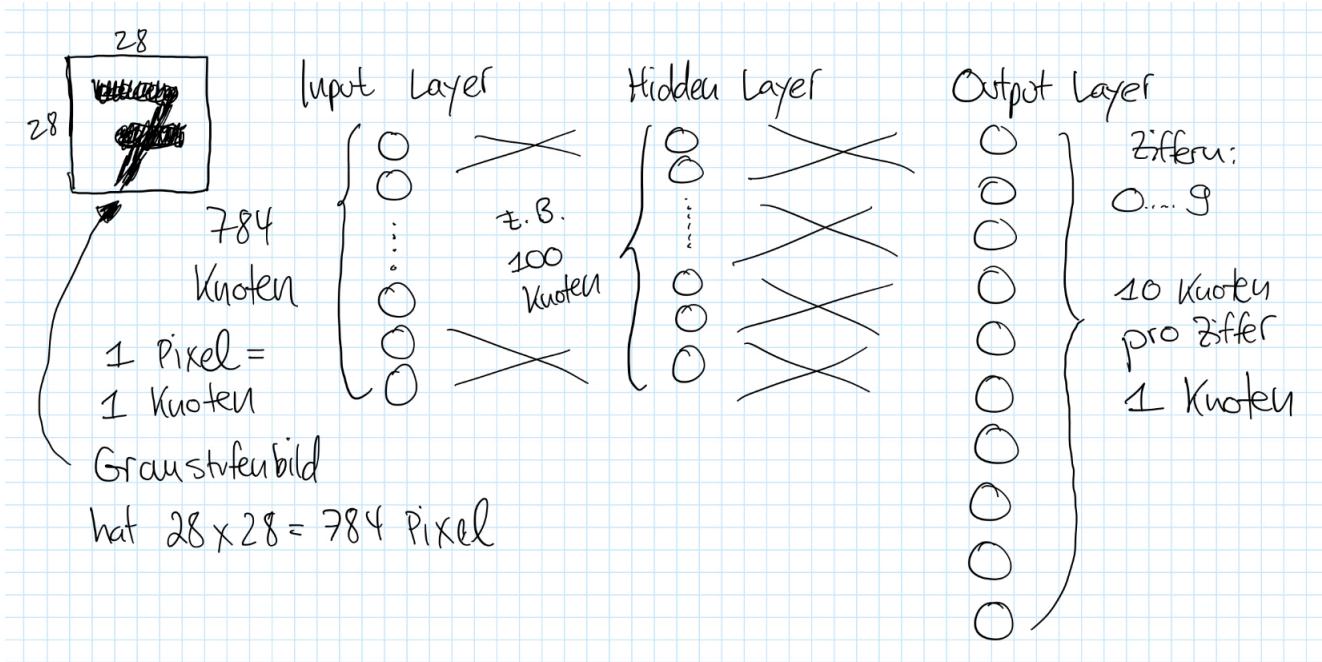
$$W_{\text{input-hidden}} = \begin{pmatrix} 0,9 & 0,3 & 0,4 \\ 0,2 & 0,8 & 0,2 \\ 0,1 & 0,5 & 0,6 \end{pmatrix} \quad W_{\text{hidden-output}} = \begin{pmatrix} 0,3 & 0,7 & 0,5 \\ 0,6 & 0,5 & 0,2 \\ 0,8 & 0,1 & 0,9 \end{pmatrix}$$

Lösungsvorschlag:



- Überprüfen Sie die Ergebnisse, durch ein Python-Programm. Formulieren Sie die Matrizen und Vektoren und berechnen Sie die Multiplikationen.
- Erzeugen Sie in der Konsole (`print`) eine "Klassifizierung" mit drei Klassen c_1 , c_2 und c_3 . Erstellen Sie ein paar zufällige Eingabevektoren und prüfen Sie Ihre Implementation.
- Lösungsvorschlag:** Sie finden die Lösung in der Python-Datei `neuronales_netz_1.py`.
- Informieren Sie sich über die MNIST-Datenbank (handgeschriebene Ziffern) und skizzieren Sie die Architektur für ein KNN. Verwenden Sie drei Schichten.

Lösungsvorschlag:



Wie werden die Output - Vektoren / Samples erzeugt?

Sample: 28x28 Pixel Graustufenbild (Bilddatei)
+ Ziffer als Dezimalzahl in einer CSV-Datei
Bsp. einer Zeile aus der CSV - Datei:

$$\underbrace{7, 0, 0, 0, 0, 0, \dots, 84, 185, 159, 151, \dots, 0, 0}_{\text{Ziffer im Bild}} \quad \underbrace{784 \text{ Graustufenwerte}}$$

$$\vec{x} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \vec{x} \in \mathbb{N}^{784 \times 1} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Eine 1 am Pos 8 steht für Ziffer 7

$$\begin{array}{c} 28 \\ \text{---} \\ \text{0} \end{array} \quad \vec{x} \quad \vec{o} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \begin{array}{c} 28 \\ \text{---} \\ \text{1} \end{array} \quad \vec{x} \quad \vec{o} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$