Crash Kurs KI

„Kritische KI-Kompetenz sollte integraler Bestandteil wissenschaftlicher Integrität und philosophischer Bildung sein.“

Leitlinien zum Umgang mit generativer KI am Institut für Philosophie der FU Berlin

Montag 22.6.2026 12:15 Uhr	***Grundlagen*** *Symbolische vs. subsymbolisch KI, Künstliche neuronale Netze, Interpretierbarkeit (Vortrag: Miguel Hoeltje)*
Montag 29.6.2026 12:15 Uhr	***Large Language Models und Chatbots*** *Lern-Paradigmen, Transformer-Architektur, LLMs vs. Chatbots, Reasoner-Models, Multimodalität (Vortrag: Miguel Hoeltje)*
Montag 6.7.2026 12:15 Uhr	***Über Large Language Models hinaus*** *Agentische KI, Vision-Language-Action Modelle, „Bitter Lesson“ & „Era of Experience“, World Models, Neurosymbolische KI (Vortrag: Miguel Hoeltje)*
Montag 13.7.2026 12:15 Uhr	***Zur ethischen Dimension der Entwicklung und Verwendung von KI*** *(Vortrag: Luise Müller)*

Folien, Handouts, ggfs. aktuelle Infos, etc.?

www.crash-ai.de

Prolog: ChatGPT ist *seltsam*…

Wie viele von Ihnen sicherlich auch, bin ich das erste Mal 2023 mit *ChatGPT* in Kontakt gekommen.

Ich war unmittelbar davon beeindruckt, wie flüssig die Kommunikation lief – abgesehen von Menschen gab bis dahin *nichts*, was auch nur *annähernd* eine solche Sprachkompetenz hatte!

(Und jetzt, gerade einmal 3 Jahre später, sind diese Systeme noch *weitaus* leistungsfähiger.)

Aber ChatGPT hat mich auch sofort ziemlich *verwirrt*, da es sich völlig *anders* verhielt, als ich es von Computerprogrammen gewohnt war:

- Dieses System konnte sich Sachen nicht zuverlässig *merken* (obwohl es freimütig „behauptete“ einen Speicher zu haben und sich Dinge für mich „definitiv“ merken zu können…),
- es war teilweise *schwer berechenbar* und *inkonsistent* (mitunter reagierte es auf identische Eingaben mit völlig verschiedenen Antworten),
- es hatte offensichtlich keinen Zugriff auf die eigene Funktionsweise und kein zuverlässiges „Verständnis“ der eigenen Fähigkeiten.
- Alles Dinge, die Computersysteme doch eigentlich *gerade* gut können sollten…?!

Das hat mich dazu gebracht, ein bisschen besser verstehen zu wollen, wie diese Systeme funktionieren und warum sie so *seltsam* sind…

Der Crash Kurs

Zielgruppe Menschen ohne Vorkenntnisse im Bereich KI.

Anspruch Sehr moderat. KI-Forschung ist ein riesiger Bereich und es wird teilweise mathematisch ziemlich komplex. Es wird lediglich darum gehen, einen Einblick in einige kleine Teilbereiche zu geben und ich werde mitunter stark vereinfachen.

Ziel Einen Beitrag zur *KI-Literacy* zu leisten. (Nicht speziell philosophische Fragen, und kein Kurs zu „Wie sollte ich KI verwenden?“, sondern eine Hilfestellung dabei, ein wenig mehr zu verstehen, wie gewisse KI-Systeme funktionieren.)

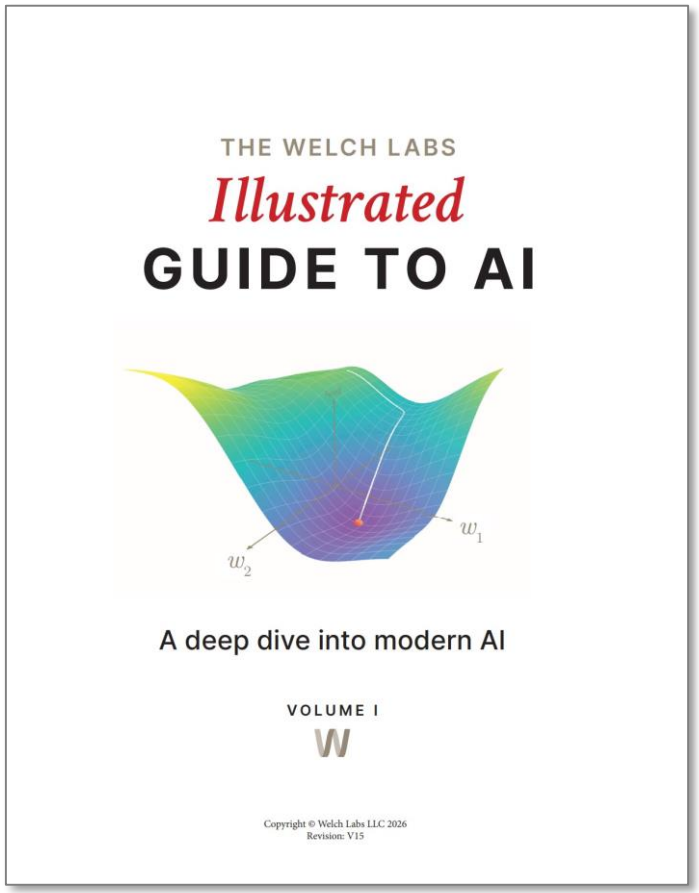
Ausrichtung *Large Language Models* (ChatGPT etc.) als „Aufhänger“: am stärksten im öffentlichen Bewusstsein, am stärksten gehyped, speziell für Uni (insbesondere Geisteswissenschaften) relevant.

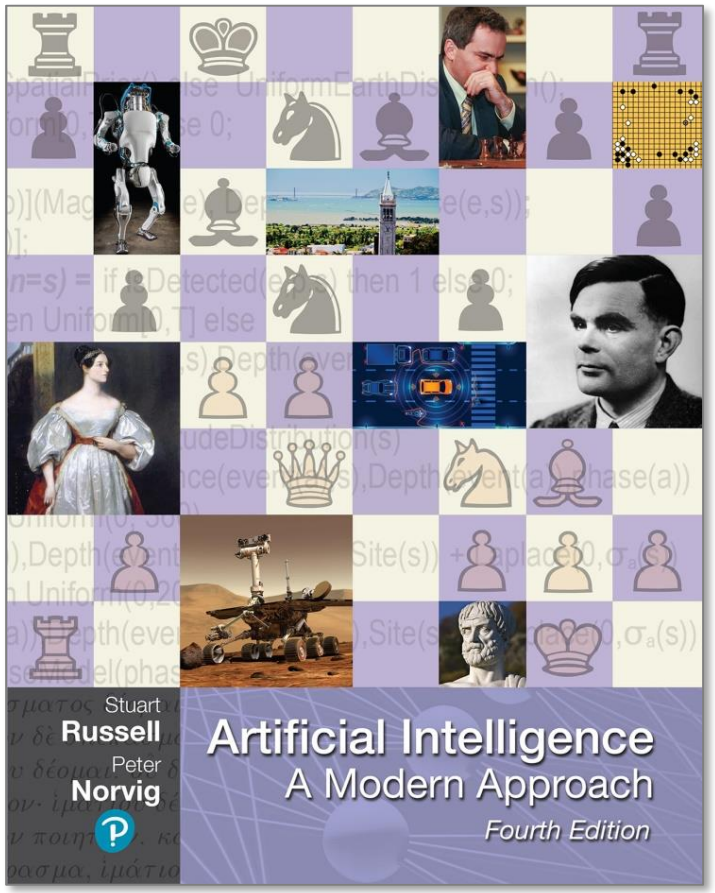
Wie funktionieren LLMs? Wie sind wir bei LLMs gelandet? Wo sind LLMs im größeren Zusammenhang zu verorten? Welche Limitationen haben LLMs und wo könnte es danach hingehen?

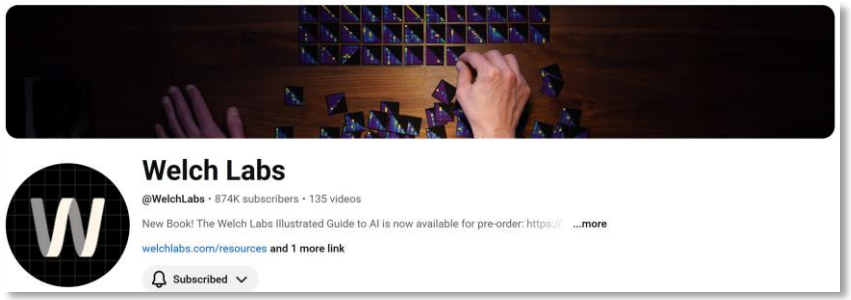
LLMs eignen sich gut als Folie, um im Kontrast andere KI-Systeme einzuordnen (*symbolic vs. non-symbolic AI* etc.), und sie eignen sich ebenfalls gut, ein paar *Big-Picture*-Themen anzureißen.

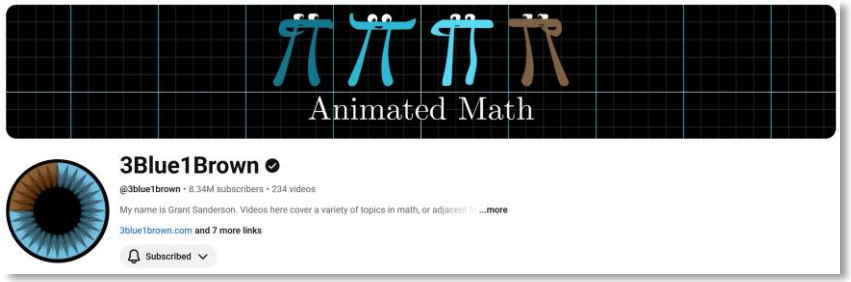
Heute Grundlagen zu *künstlichen neuronalen Netzen*.

Hilfreiche Lektüre und Videos

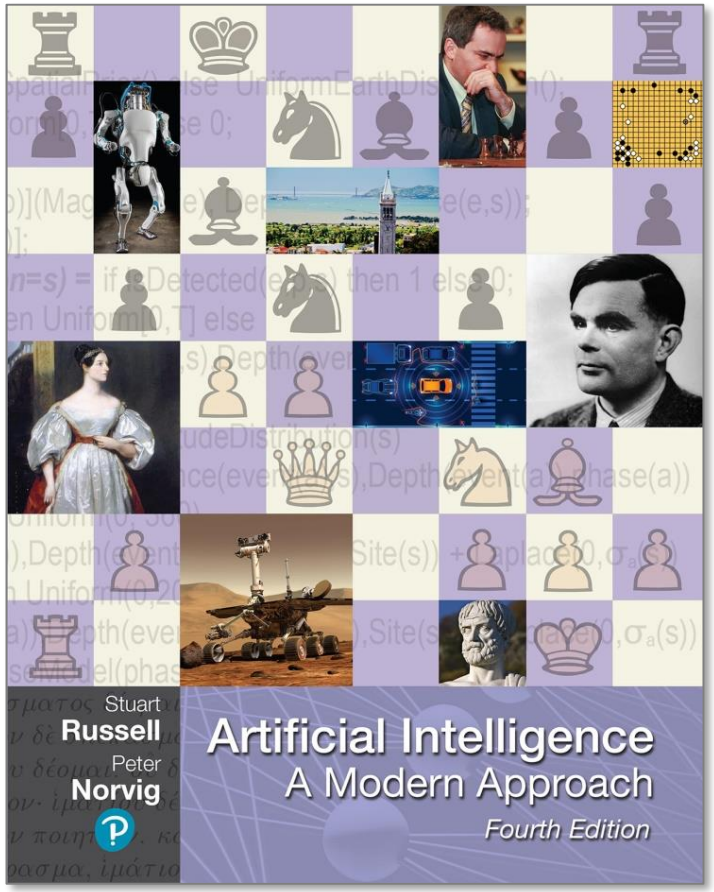








Einstieg: *Lernende Maschinen*



Why would we want a machine to learn? Why not just program it the right way to begin with? There are two main reasons. First, the designers cannot anticipate all possible future situations. For example, a robot designed to navigate mazes must learn the layout of each new maze it encounters; a program for predicting stock market prices must learn to adapt when conditions change from boom to bust. Second, sometimes the designers have no idea how to program a solution themselves. Most people are good at recognizing the faces of family members, but they do it subconsciously, so even the best programmers don't know how to program a computer to accomplish that task, except by using machine learning algorithms.

Russell & Norvig (2022): *Artificial Intelligence: A Modern Approach*, 4th Edition, Global Edition, 669

Neuronale Netze

Der gegenwärtige KI-Boom ist zum Großteil durch die Entwicklung *künstlicher neuronaler Netze* (KNN) gekennzeichnet.

Und auch bei den *Large Language Models*, die im Zentrum der nächsten Sitzung stehen werden, handelt es sich (zumindest weitestgehend) um neuronale Netze.

In der heutigen Sitzung wird es daher darum gehen, zumindest ein grundlegendes Verständnis von KNNs zu entwickeln.

Wir gehen hierzu in 3 Schritten vor:

1. Eine erste Charakterisierung (noch ohne auf die *interne Struktur* von KNNs einzugehen)
2. *Künstliche Neuronen*: Ein Blick auf die *Grundbausteine*, aus denen KNNs aufgebaut sind.
3. Ein konkretes Beispiel für ein neuronales Netz

Neuronale Netze

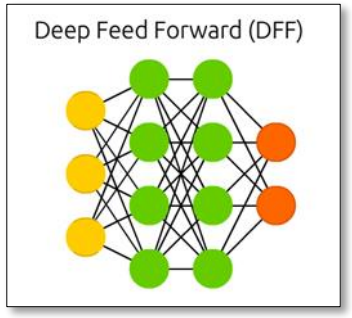
Bevor wir loslegen, noch ein Hinweis.

Es gibt *viele* verschiedene Arten von neuronalen Netzen (und dieser Forschungsbereich entwickelt sich ständig weiter).

Für einen kleinen Eindruck hier das Poster *The Neural Network Zoo*, welches bereits eine Vielzahl von Netz-Typen abbildet

… und es gibt noch *einige* mehr.

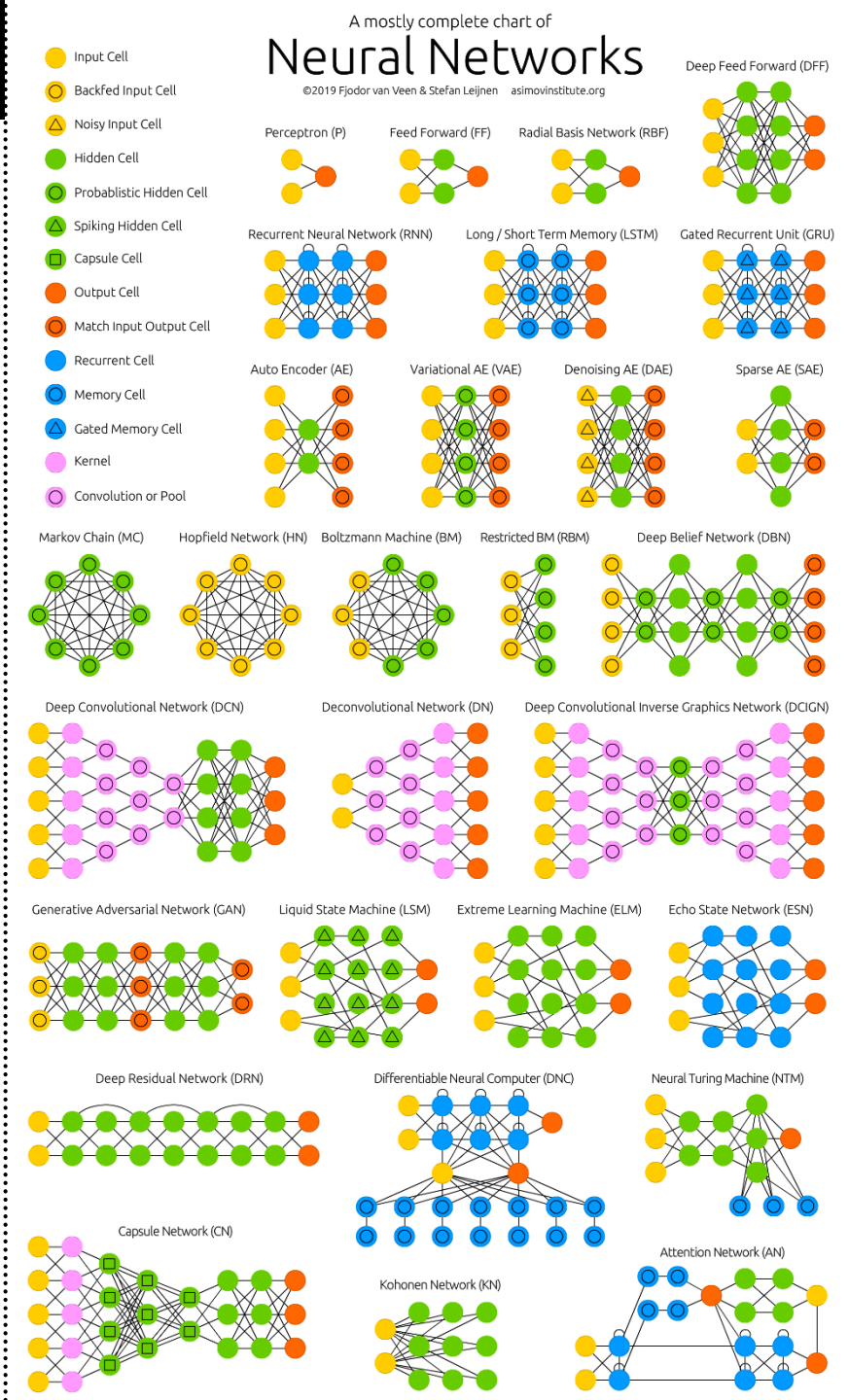
(That is a bit of an overstatement…)



Wir werden uns heute nur eine sehr einfache Art von Netz anschauen (häufig auch *Multi Layer Perceptron* oder schlicht *Feed Forward* Netz genannt).

Aber bereits simple *Feed Forward* Netze sind komplex genug, um anhand von ihnen die grundlegenden Begriffe und das Lernen/Training zu illustrieren…

… zudem bilden Netze dieser Art auch eine Komponente von Transformer-basierten LLMs, um die es nächste Woche gehen wird.

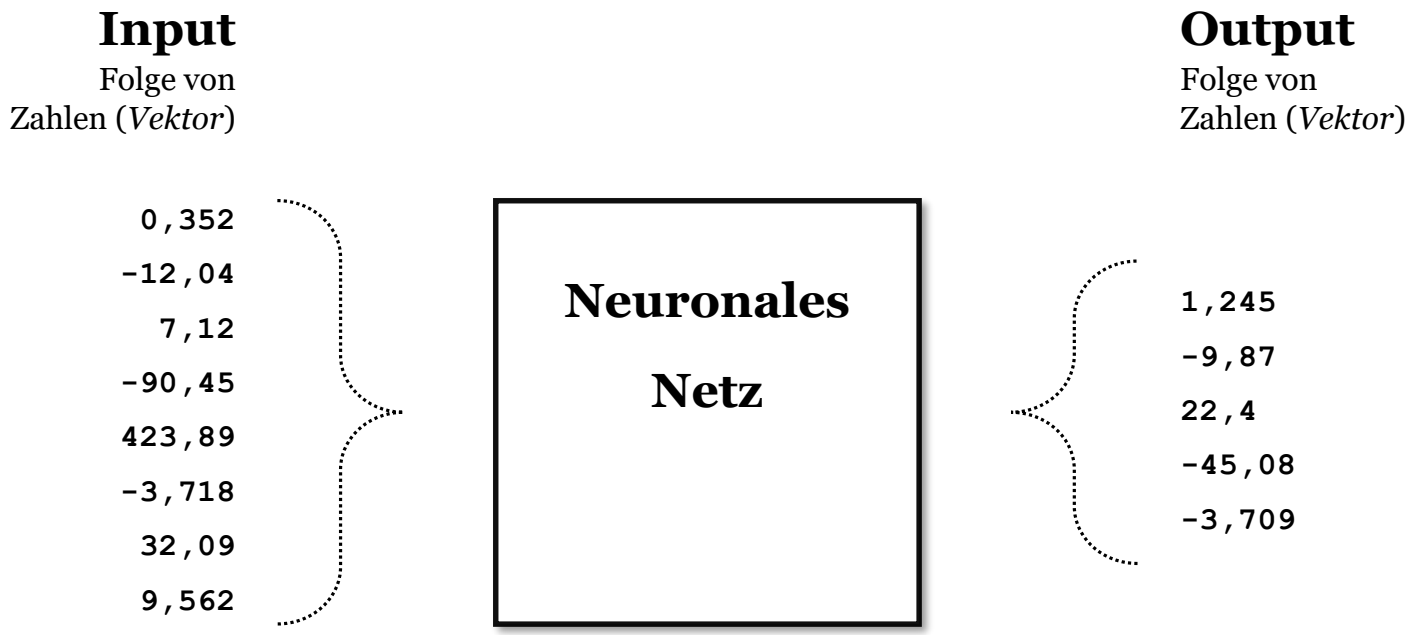


Neuronale Netze | Level 1

Wir können uns ein neuronales Netz als eine *mathematische Funktion* vorstellen:

Es nimmt eine Folge von Zahlen als Input, führt eine Berechnung durch, und liefert eine Folge von Zahlen als Output.

Wie viele Zahlen Input und Output umfassen, ist von Netz zu Netz verschieden und hängt davon ab, was wir mit dem Netz machen wollen.



Neuronale Netze | Level 1

Beispiel:

Wenn wir ein Netz haben wollen, welches sich Bilder der Größe 128×128 Pixel „anschaut“ und sagt, ob es sich um ein Bild eines Hundes oder einer Katze handelt, dann könnten wir nehmen:

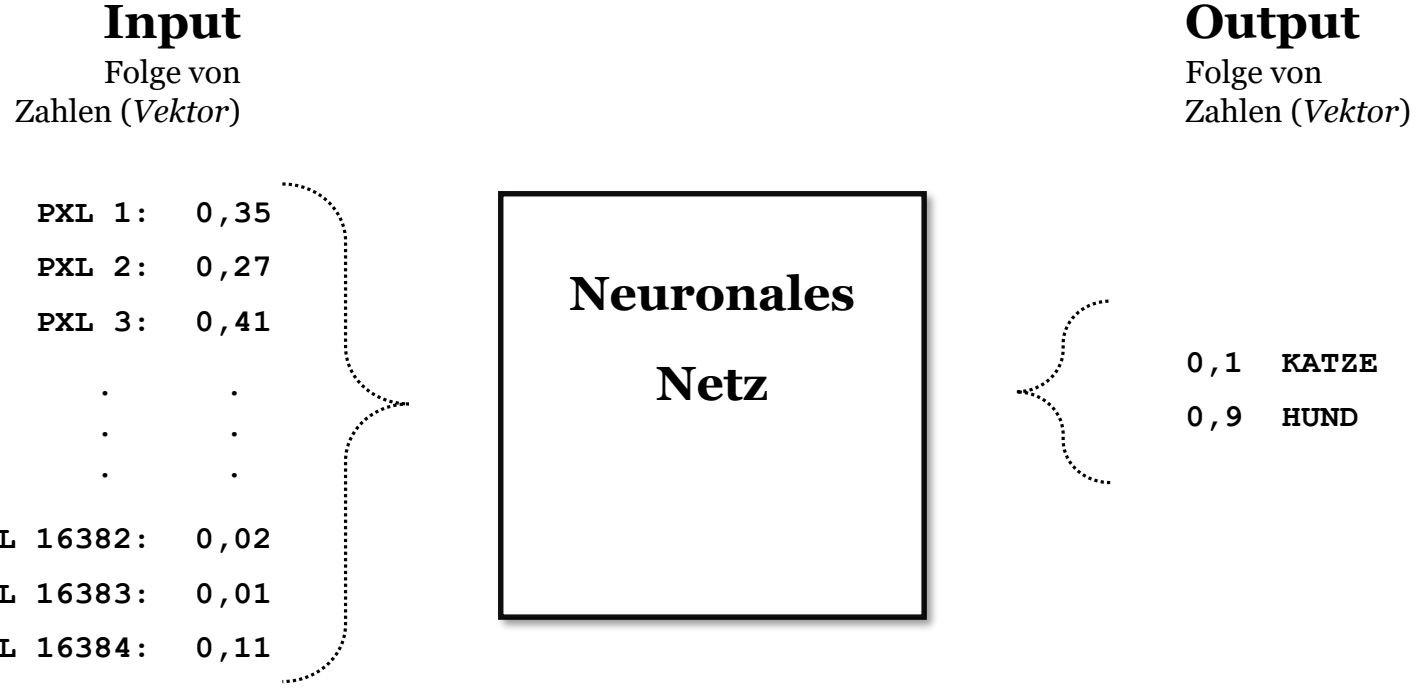
16384 Input-Zahlen:

je eine pro Pixel (Grauwert)

2 Output-Zahlen:

Wahrscheinlichkeit für *Hund*

Wahrscheinlichkeit für *Katze*



Ziel: Ein Netz, welches ausgehend von den Pixelwerten *ausrechnen* kann, ob es sich um Hund oder Katze handelt.

Aber wie soll so eine Berechnung denn bitte aussehen?

Idee: Das soll das Netz selbst herausfinden! Wir zeigen ihm viele Bilder von Hunden bzw. Katzen, und das Netz *lernt*, wie man von den Pixelwerten zur korrekten Bildklassifikation kommt.

Aber wie kann ein Netz denn lernen?

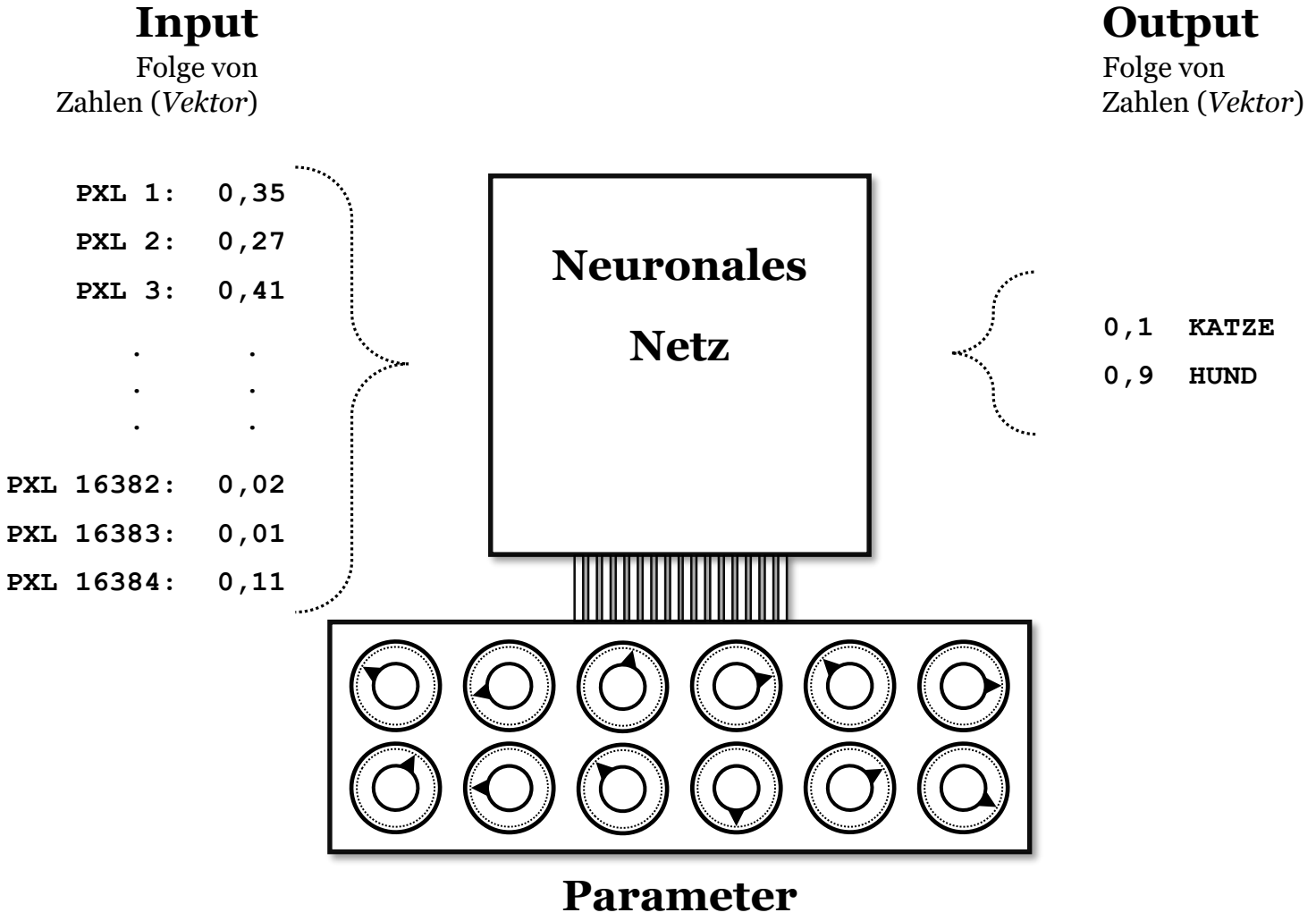
Um das zu verstehen, müssen wir eine Komponente berücksichtigen, die ich bislang ignoriert habe: ***Parameter***.

Neuronale Netze | Level 1

Jedes neuronale Netz hat eine Reihe von *Parametern*.

Welche Berechnung das Netz genau durchführt, kann durch diese Parameter beeinflusst werden.

Wir können uns die Parameter als kleine Stellschrauben oder Drehregler vorstellen, die sich verschieden einstellen lassen.



Neuronale Netze | Level 1

Jedes neuronale Netz hat eine Reihe von *Parametern*.

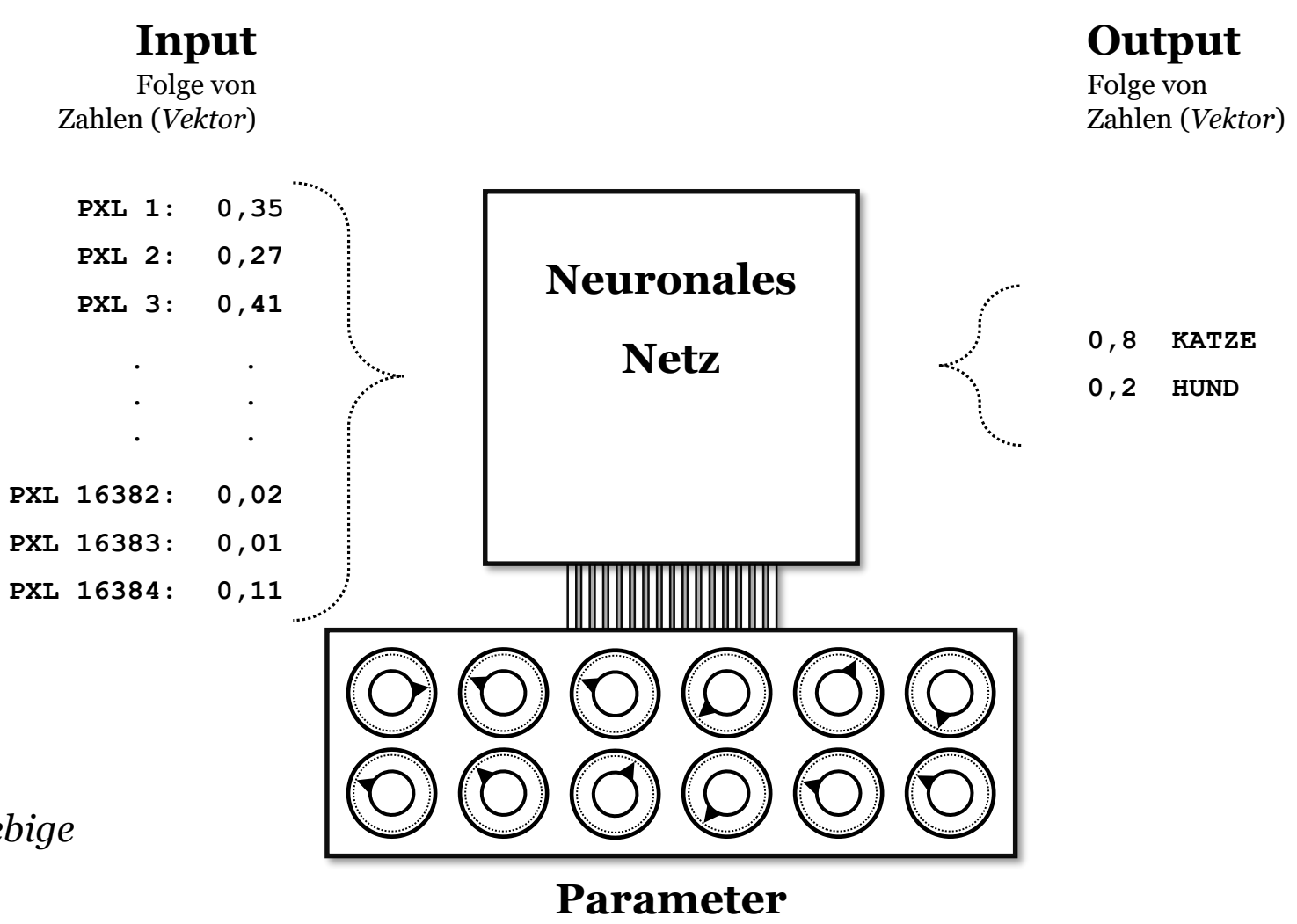
Welche Berechnung das Netz genau durchführt, kann durch diese Parameter beeinflusst werden.

Wir können uns die Parameter als kleine Stellschrauben oder Drehregler vorstellen, die sich verschieden einstellen lassen.

Der *Lernprozess* eines neuronalen Netzes besteht darin, anhand von *Trainingsdaten* zu versuchen, eine geeignete Einstellung der *Parameter* zu finden.

Erfolgreiches Lernen/„Training“ =

Das Finden einer Parametereinstellung, die beliebige Inputs mit adäquaten Outputs verknüpft.



Neuronale Netze | Level 1

Die Illustration zeigt ein Netz mit 12 Parametern. (Das sind *de facto* nicht genug für diese Art von Aufgabe…)

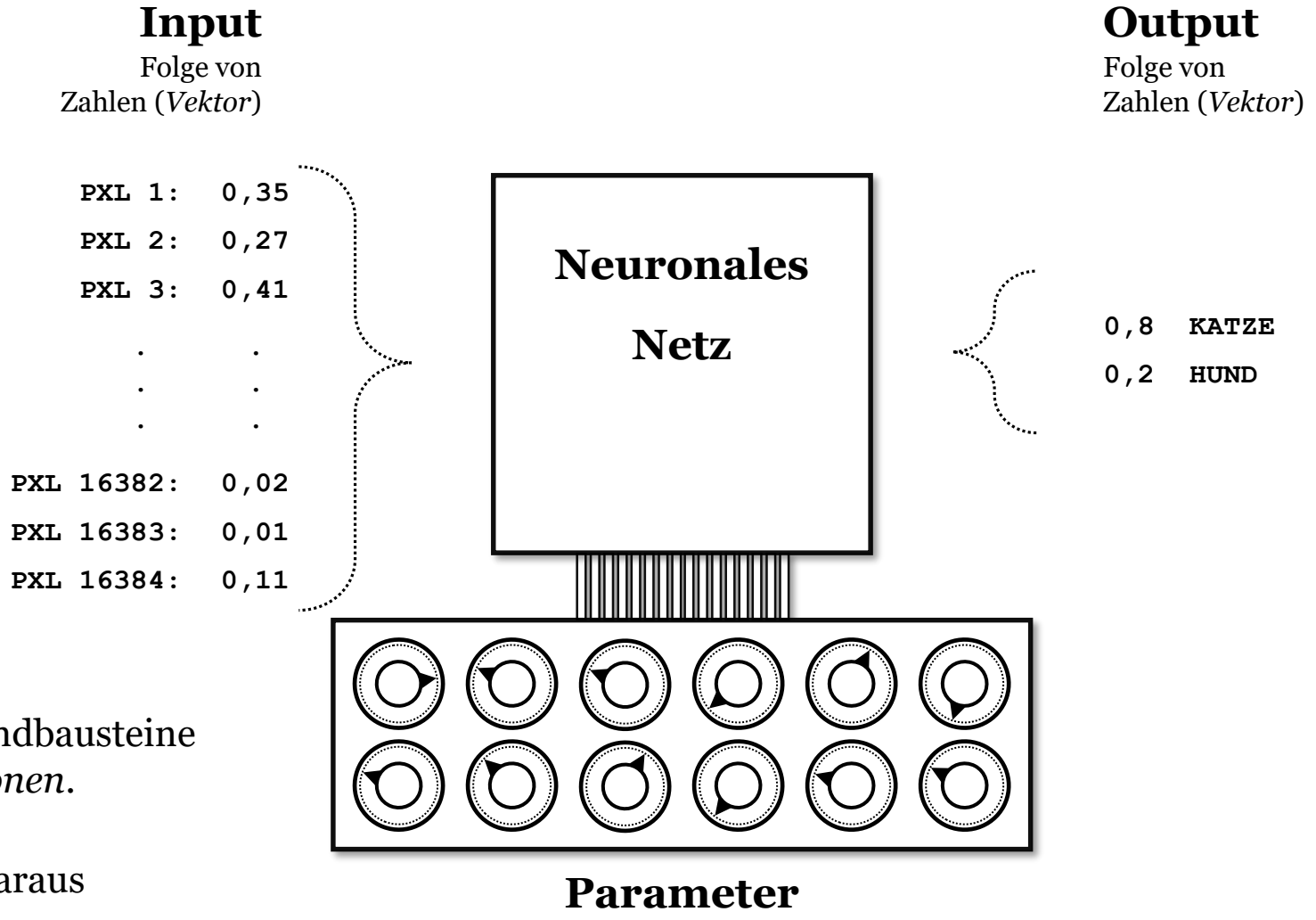
Heutige Netze (etwa in LLMs) haben *viele Milliarden Parameter*.

Aber wie kann man denn eine geeignete Parameter-Einstellung finden?

Um dies etwas besser zu verstehen, gehen wir in 2 Schritten vor:

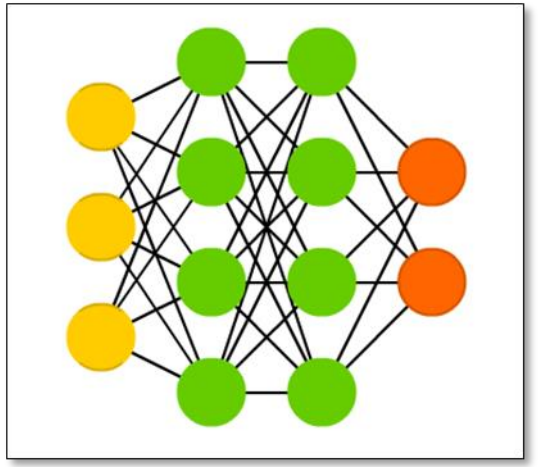
1. Wir werfen zunächst einen Blick auf die Grundbausteine neuronaler Netze: *Einzelne künstliche Neuronen*.

2. Anschließend schauen wir uns an, wie sich daraus komplexe Netze basteln lassen.



Dadurch wird klarer werden, wie neuronale Netze intern aufgebaut sind und welche Rolle Parameter spielen.

Neuronale Netze | Einzelnes Neuron

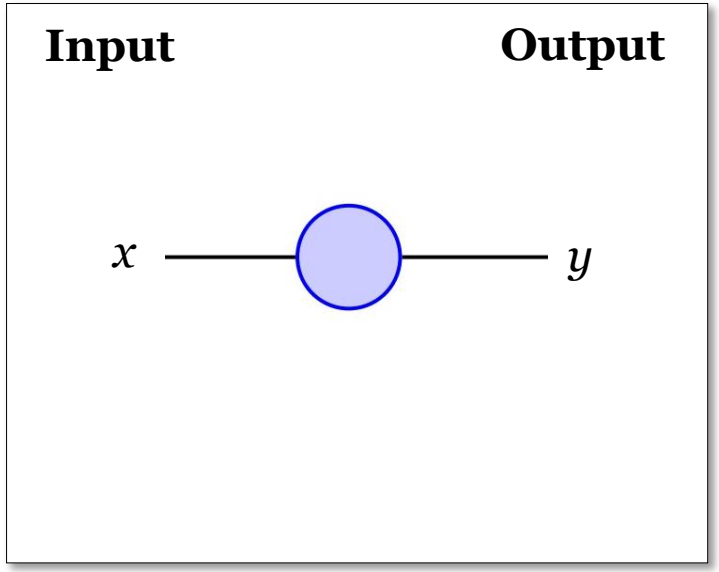


Künstliche neuronale Netze & Neuronen sind durch biologische Neuronen inspiriert.

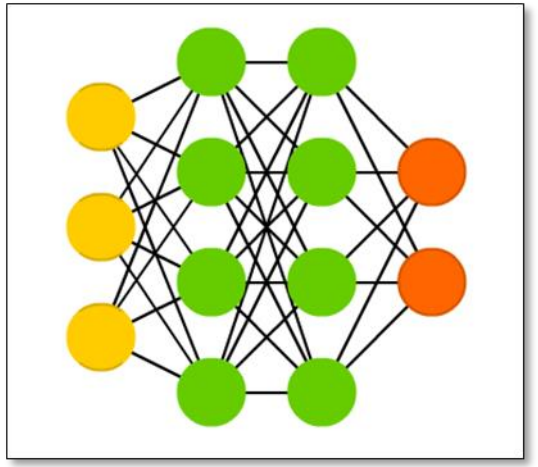
(Auf die teilweise signifikanten Unterschiede zwischen biologischen und künstlichen „Neuronen“ gehen wir hier nicht ein; im Folgenden ist mit „Neuron“ immer gemeint: „*künstliches* Neuron“.)

Ein einzelnes Neuron erhält ein Input, unterzieht das Input einer Rechenoperation, und leitet das Ergebnis der Rechenoperation als Output weiter.

Hier als Beispiel ein Neuron mit nur einem Input: x.



Neuronale Netze | Einzelnes Neuron



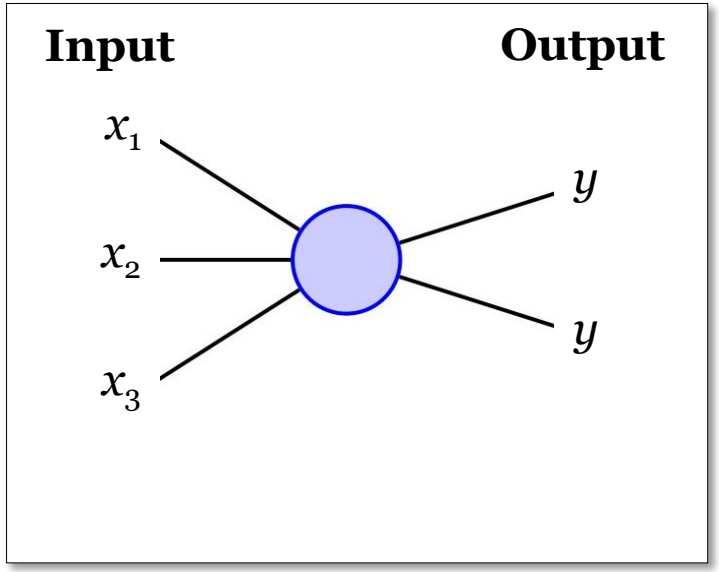
Künstliche neuronale Netze & Neuronen sind durch biologische Neuronen inspiriert.

(Auf die teilweise signifikanten Unterschiede zwischen biologischen und künstlichen „Neuronen“ gehen wir hier nicht ein; im Folgenden ist mit „Neuron“ immer gemeint: „*künstliches* Neuron“.)

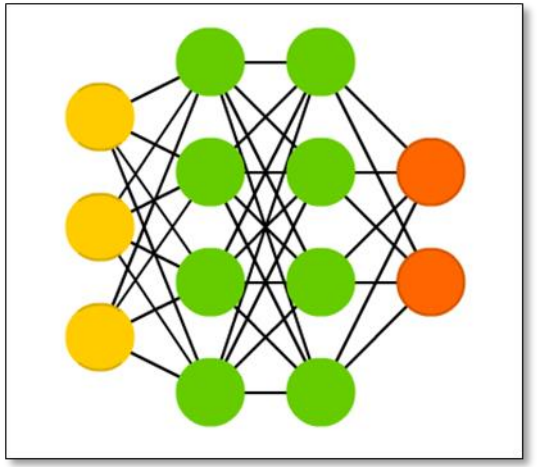
Ein einzelnes Neuron erhält ein Input, unterzieht das Input einer Rechenoperation, und leitet das Ergebnis der Rechenoperation als Output weiter.

Hier ein Neuron mit 3 Inputs (x_1, x_2, x_3), welches das Output (y) an zwei Neuronen weiterleitet.

Prinzipiell kann ein einzelnes Neuron sowohl „nach links“ als auch „nach rechts“ mit einer beliebigen (endlichen) Anzahl von weiteren Neuronen verbunden sein.



Neuronale Netze | Einzelnes Neuron



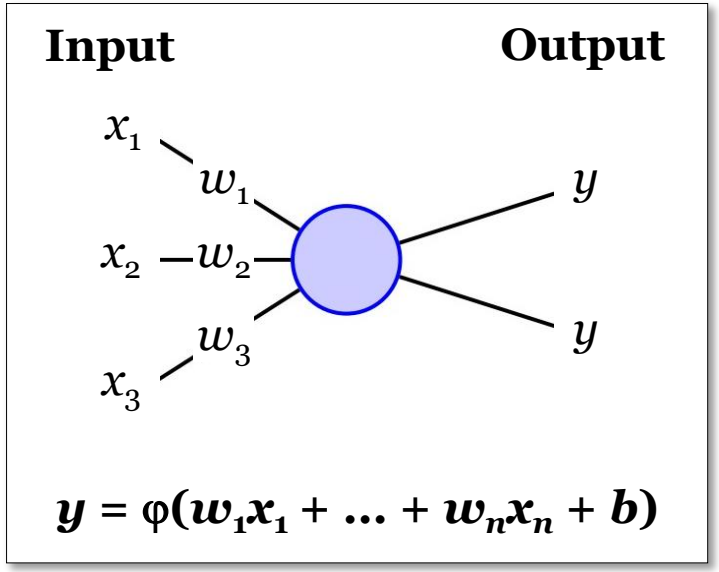
Ähnlich wie in biologischen neuronalen Netzen, können die Verbindungen zwischen einzelnen Neuronen in künstlichen neuronalen Netzen *verschieden stark* sein.

Dies modellieren wir, indem wir jede Input-Verbindung mit einem *Gewicht* versehen.

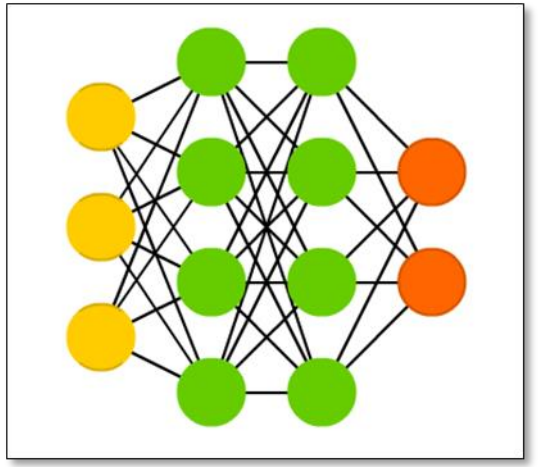
Das Output y eines einzelnen Neuron errechnet sich dann, indem eine Aktivierungsfunktion φ auf die gewichtete Summe der Inputs plus einen Bias-Wert b angewendet wird:

$$y = \varphi(w_1x_1 + \ldots + w_nx_n + b)$$

Ein einzelnes Neuron kann also als eine mathematische Funktion verstanden werden, und ein neuronales Netz als eine Verknüpfung der einzelnen Funktionen, die den Neuronen im Netz entsprechen.



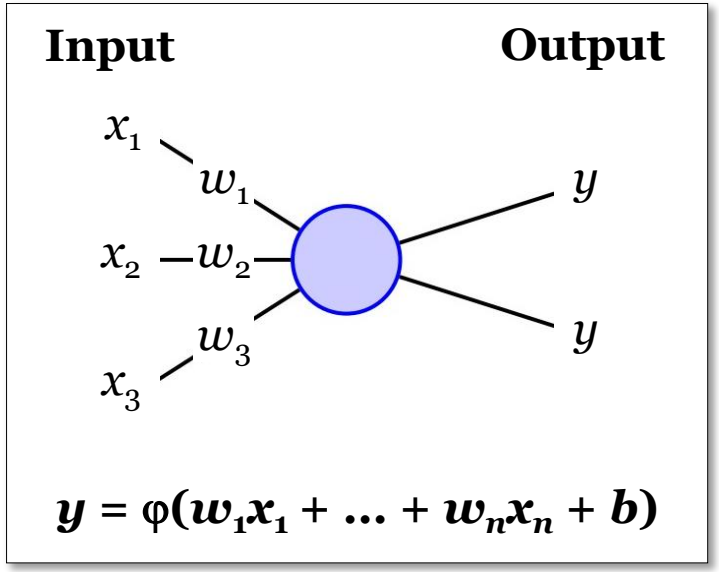
Neuronale Netze | Einzelnes Neuron



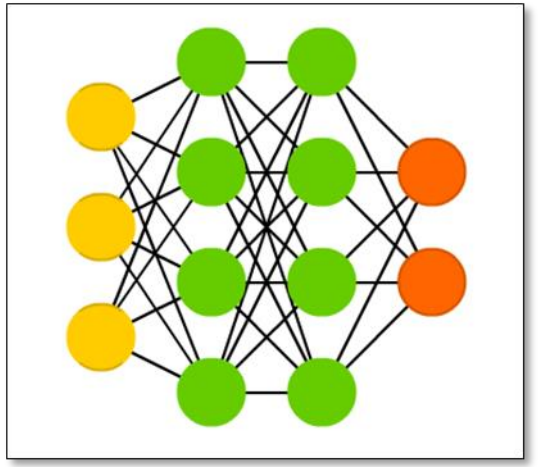
Um nun zu verstehen, wie ein einzelnes Neuron (und darauf basierend: ein neuronales Netz) *lernen* kann, werden wir mit einem vereinfachten Fall beginnen:

- die Aktivierungsfunktion φ werden wir fürs erste ignorieren (darauf können wir ggf. in der Q&A eingehen)
- den Bias-Term *b* lassen wir zu Beginn ebenfalls weg (werden ihn aber im Rahmen des Beispiels dann einführen)
- zudem werden wir uns auf ein Neuron mit nur *einem* Input *x* beschränken
- und entsprechend müssen wir nur ein Gewicht, *w*, berücksichtigen.

Dieser Fall ist ausreichend, um sich einen ersten Eindruck davon zu verschaffen, wie ein Neuron ausgehend von Trainingsdaten lernen kann, und welche Rolle Parameter dabei spielen.



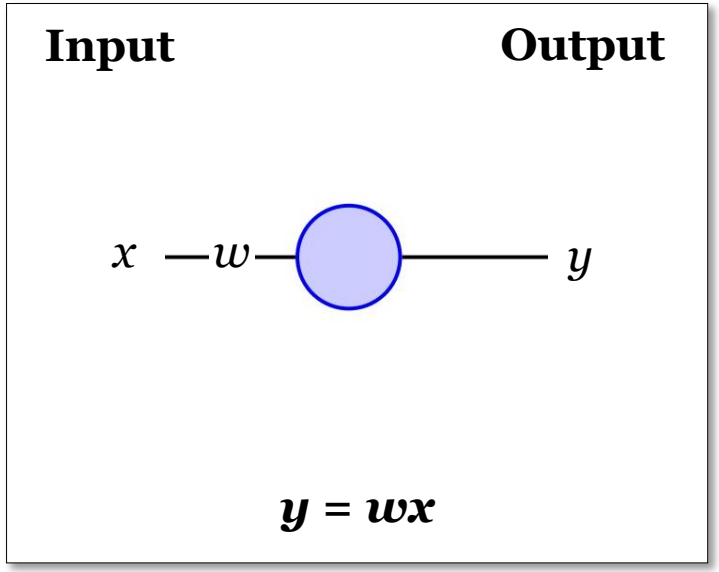
Neuronale Netze | Einzelnes Neuron



Um nun zu verstehen, wie ein einzelnes Neuron (und darauf basierend: ein neuronales Netz) *lernen* kann, werden wir mit einem vereinfachten Fall beginnen:

- die Aktivierungsfunktion φ werden wir fürs erste ignorieren (darauf können wir ggf. in der Q&A eingehen)
- den Bias-Term *b* lassen wir zu Beginn ebenfalls weg (werden ihn aber im Rahmen des Beispiels dann einführen)
- zudem werden wir uns auf ein Neuron mit nur einem Input *x* beschränken
- und entsprechend müssen wir nur ein Gewicht, *w*, berücksichtigen.

Dieser Fall ist ausreichend, um sich einen ersten Eindruck davon zu verschaffen, wie ein Neuron ausgehend von Trainingsdaten lernen kann, und welche Rolle Parameter dabei spielen.



Neuronale Netze | Einzelnes Neuron

Das nun folgende Beispiel können Sie ausführlicher in einem Video von *Primer* nachvollziehen:

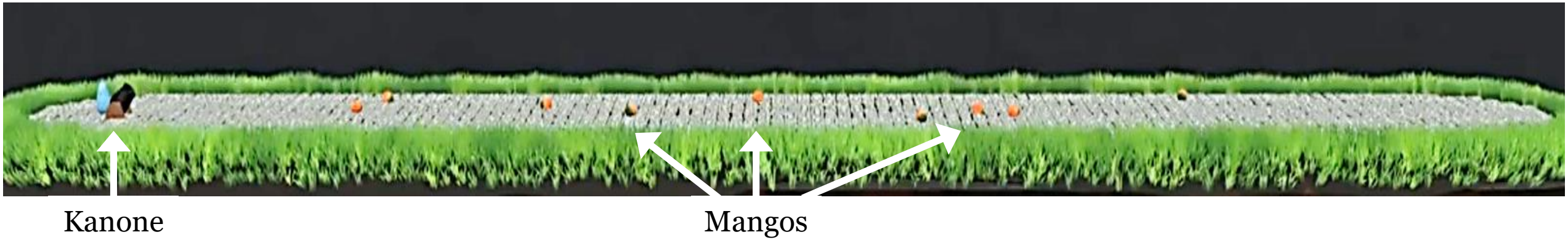
Simulating a single brain cell

https://www.youtube.com/watch?v=GkiITbguoVo

Aufbau: Eine Kanone, die mit unterschiedlich hoher Energie Mangos schießt.

Ziel: Wir wollen einem Neuron beibringen, korrekt vorherzusagen, wie weit eine Mango geschossen werden wird, abhängig davon, mit welcher Energie die Kanone schießt.

Das Neuron bekommt also als *Input* eine Zahl, die für die Energie steht, und produziert als *Output* eine Zahl, die wir als die Vorhersage des Neurons für die Entfernung interpretieren werden.



Neuronale Netze | Einzelnes Neuron

Zunächst werden wir nur einen Parameter berücksichtigen: Das Gewicht w, mit dem der Input-Wert multipliziert wird.

Das Output unseres Neuron berechnet sich also schlicht wie folgt:

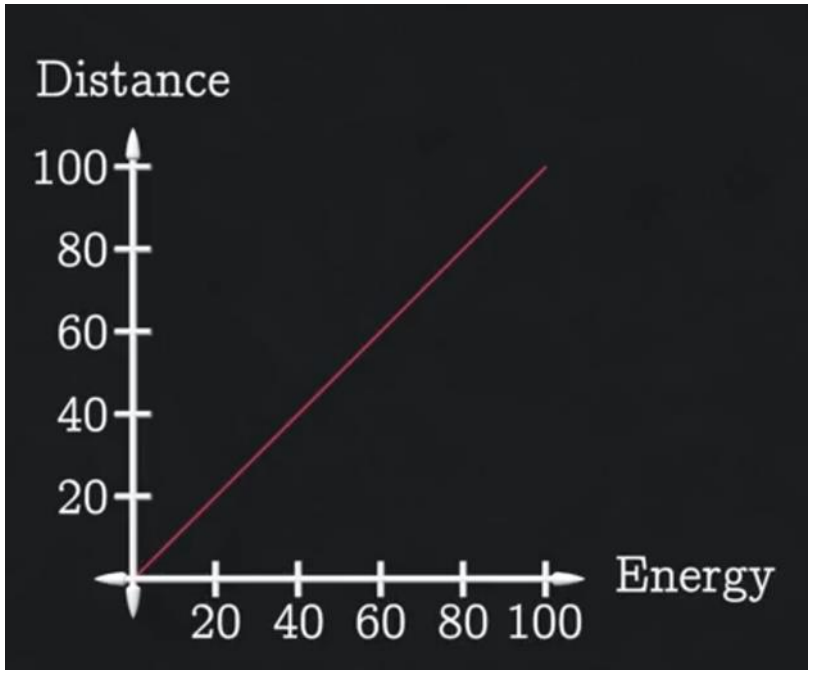
$$y = wx$$

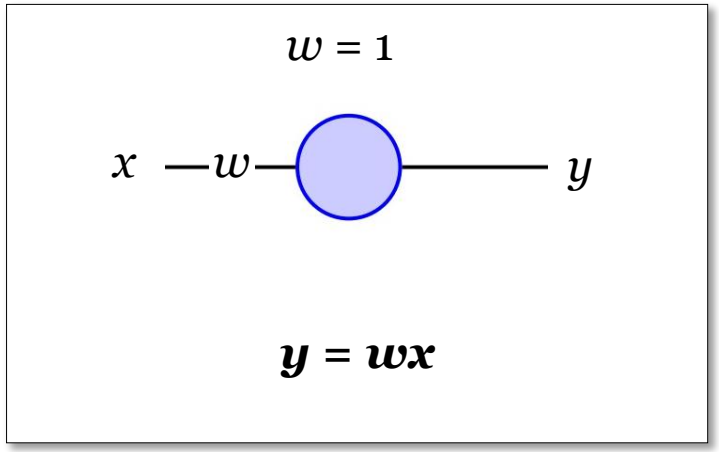
Wir können die entsprechende Funktion also eine Gerade in einem Koordinatensystem darstellen.

Setzen wir den Parameter w zunächst auf 1, so erhalten wir den folgenden Graph.

Da es sich bei dem Gewicht w um einen *Parameter* handelt, kann dieser im Laufe des Trainings geändert werden.

Das Ziel ist es nun, einen Wert für w zu finden, der dem tatsächlichen Verhalten der Kanone möglichst gut gerecht wird.





Neuronale Netze | Einzelnes Neuron

Hierzu sammeln wir *Trainingsdaten:*

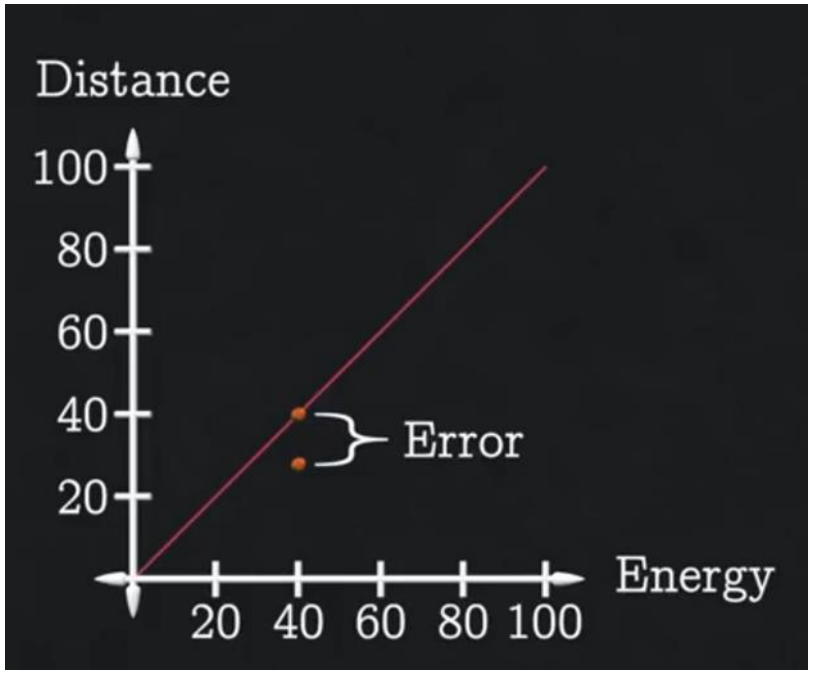
Wie weit schießt die Kanone, abhängig von der Energie, tatsächlich?

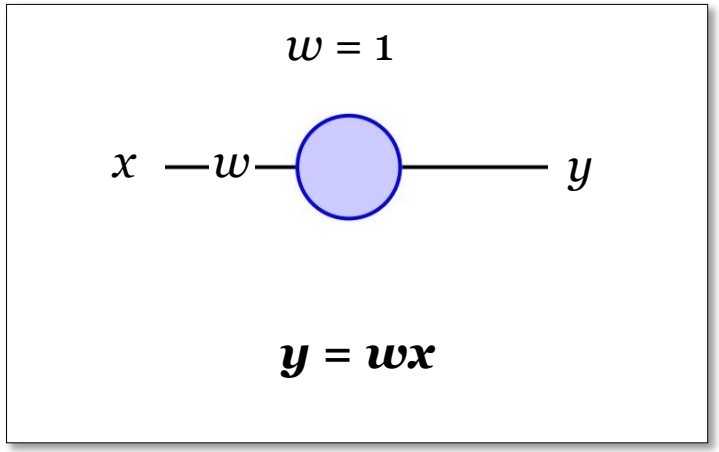
Diese können wir dann mit der „Vorhersage“ des Neurons vergleichen, um einen *Fehlerwert* zu berechnen (oft auch die *Kosten* des Neurons genannt):

Wie weit (und in welche Richtung) weicht der vorhergesagte Wert von dem tatsächlichen Wert ab?

Auf der Basis der Kosten kann nun der Parameter w schrittweise geändert werden.

In diesem Beispiel etwa sollte w *herabgesetzt* werden, da die tatsächliche Entfernung für das Input 40 unter dem Output des Neurons liegt.





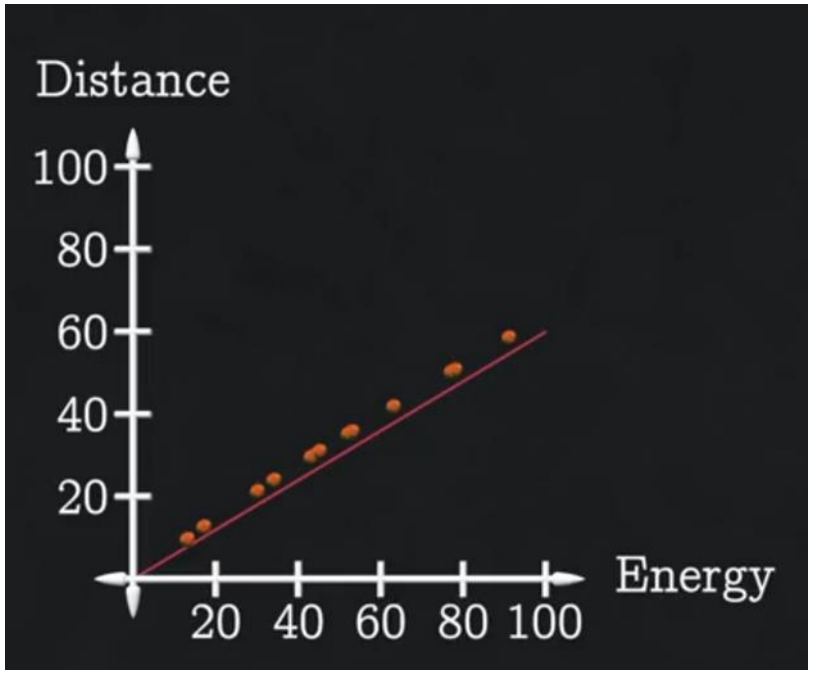
Neuronale Netze | Einzelnes Neuron

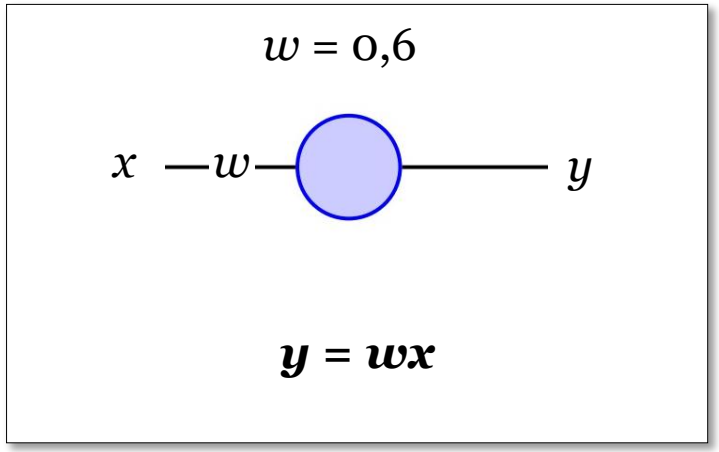
Sofern wir hinreichend viele Trainingsdaten zur Verfügung haben, kann das Neuron durch sukzessive Modifikation des Gewichtes w bei einem adäquaten Wert landen.

In diesem Szenario könnte dieser Wert beispielsweise 0,6 sein.

Upshot:

- Das Gewicht ist ein *Parameter*; es lässt sich im Lichte von Trainings-daten modifizieren.
- Das „Lernen“ des Neurons besteht darin, den Parameter schrittweise so zu modifizieren, dass die *Kosten* relativ zu den Trainingsdaten *minimiert* werden.

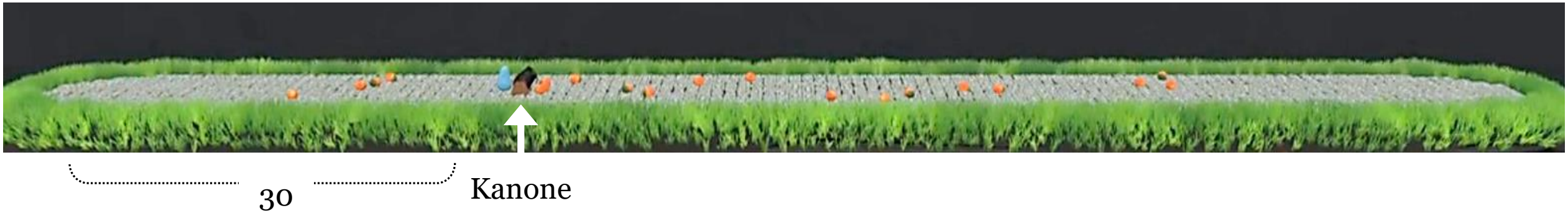




Neuronale Netze | Einzelnes Neuron

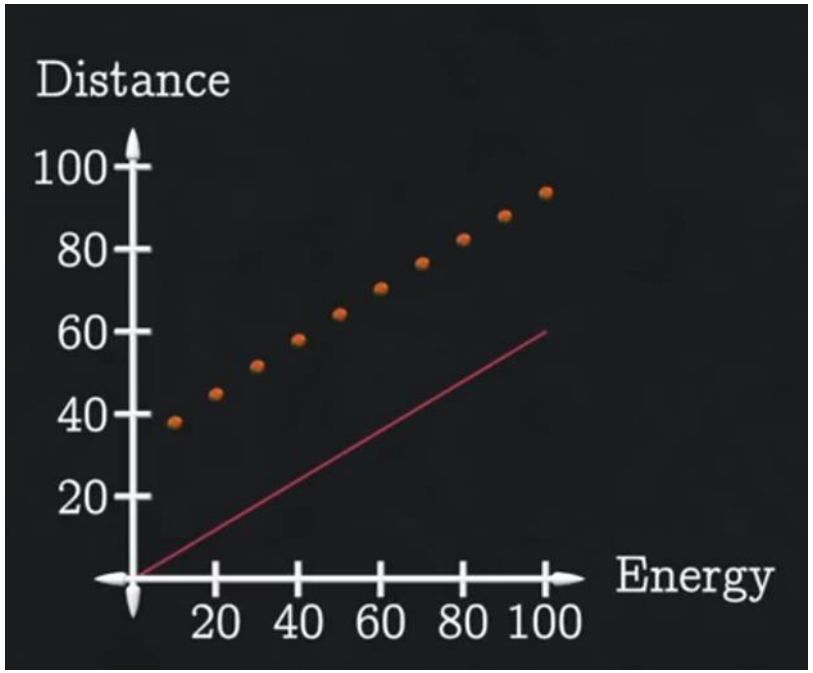
Um zu illustrieren, warum wir zusätzlich zum Gewicht w oft noch eine *weitere Art von Parameter* benötigen – den Bias-Term b – ändern wir den Versuchsaufbau ein wenig.

Rücken wir die Kanone um 30 Entfernungseinheiten nach vorn, während wir aber weiterhin die Entfernung der geschossenen Mangos vom ursprünglichen Ausgangspunkt messen.

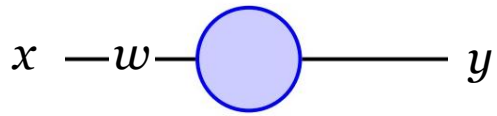


Neuronale Netze | Einzelnes Neuron

In diesem modifizierten Szenario rücken die gemessenen Entfernungen entsprechend um 30 Einheiten nach oben.



$w = 0{,}6$



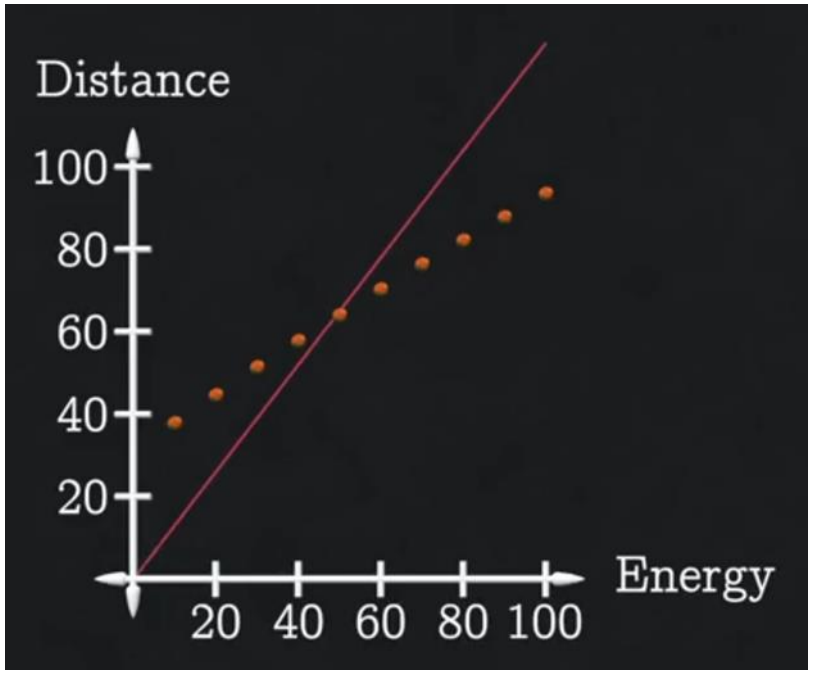
$$y = wx$$

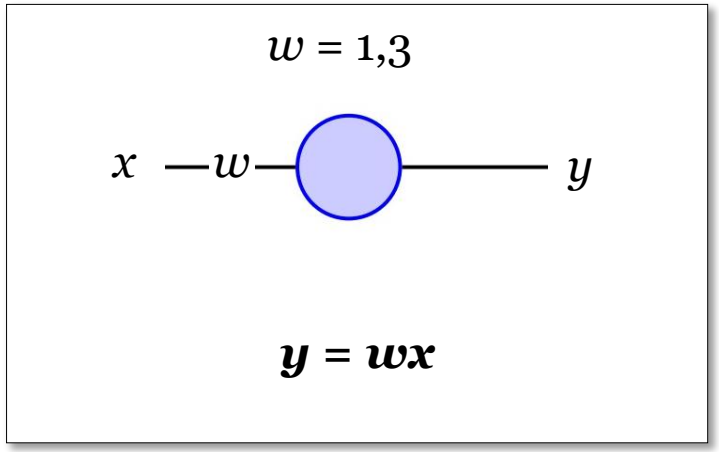
Neuronale Netze | Einzelnes Neuron

Allein mittels der Modifikation von w können wir diesem neuen Versuchsaufbau nicht gerecht werden.

Das Gewicht w kontrolliert die *Steigung* des Graphen;

aber was wir brauchen ist zusätzlich etwas, dass den *y-Achsenabschnitt* kontrolliert (den Punkt, an dem der Graph die y-Achse schneidet).





Neuronale Netze | Einzelnes Neuron

Allein mittels der Modifikation von w können wir diesem neuen Versuchsaufbau nicht gerecht werden.

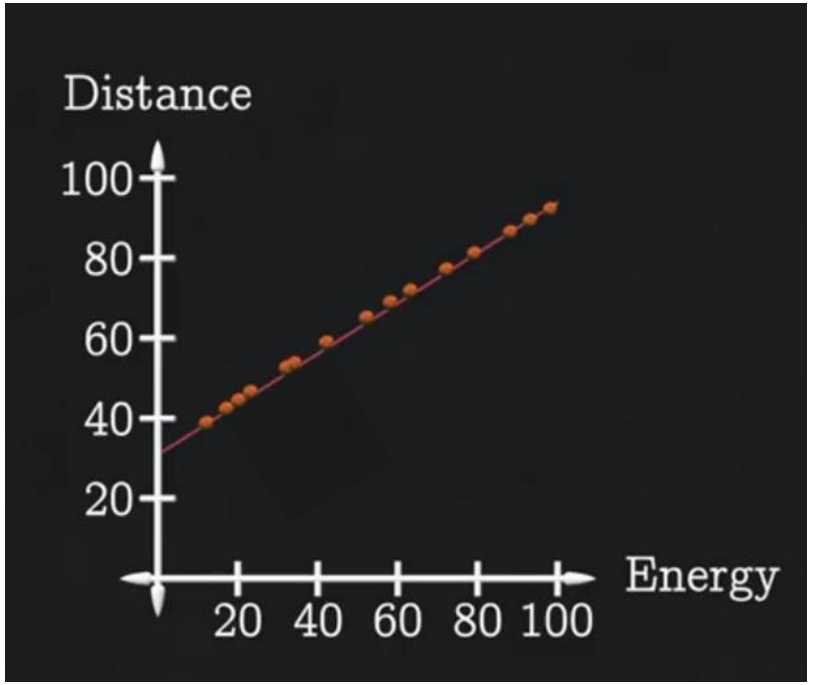
Das Gewicht w kontrolliert die *Steigung* des Graphen;

aber was wir brauchen ist zusätzlich etwas, dass den *y-Achsenabschnitt* kontrolliert (den Punkt, an dem der Graph die y-Achse schneidet).

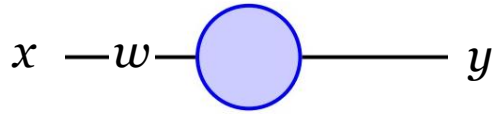
Hierzu dient die Einführung eines weiteren Parameters – der Bias-Term b:

$$\boldsymbol{y = wx + b}$$

Durch diese zusätzliche Komplexität ist es nun möglich, auf der Basis der Trainingsdaten eine Parameter-Einstellung zu finden, die dem neuen Szenario gerecht wird.



$w = 0{,}635 \quad b = 31$



$$\boldsymbol{y = wx + b}$$

Neuronale Netze | Einzelnes Neuron

Zusammenfassung:

- Lernen/Training besteht in der Suche nach Parameter-Einstellungen, die den Trainingsdaten gerecht werden.
- Künstliche Neuronen haben zwei Arten von Parametern:
  - *Gewichte* ($w_1 \dots w_n$), mit denen einzelne Inputs ($x_1 \dots x_n$) multipliziert werden (je ein Gewicht pro eingehender Verbindung)
  - einen Bias-Term b (einer pro Neuron)
- Ein einzelnes Neuron bildet die Summe der gewichteten Inputs und dem Bias-Term:
  - $w_1x_1 + \dots + w_1x_1 + b$
- Anschließend wendet das Neuron noch eine Aktivierungsfunktion φ auf diese Summe an:

$$\boldsymbol{y = \varphi(w_1x_1 + \dots + w_nx_n + b)}$$

Die Aktivierungsfunktion haben wir bislang ignoriert (und werden das vorerst auch weiter tun).

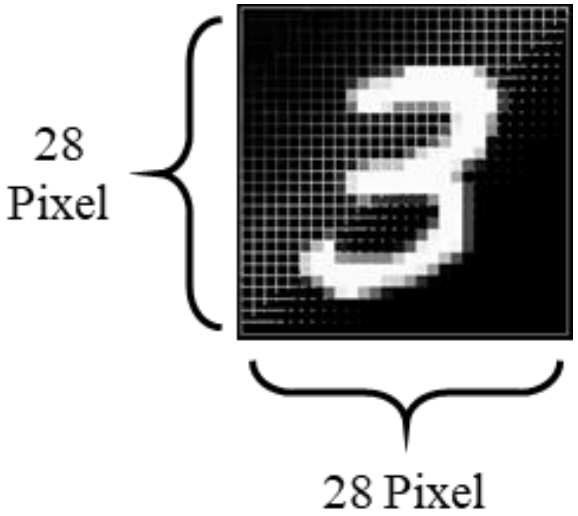
> Ihrer zentrale Rolle besteht darin, dem Neuron zu ermöglichen, auch nicht-linear auf Inputs zu reagieren. Das ist wichtig, weil mehrschichtige Netze dadurch eine größere Ausdrucksstärke bekommen und komplexere Muster lernen können.

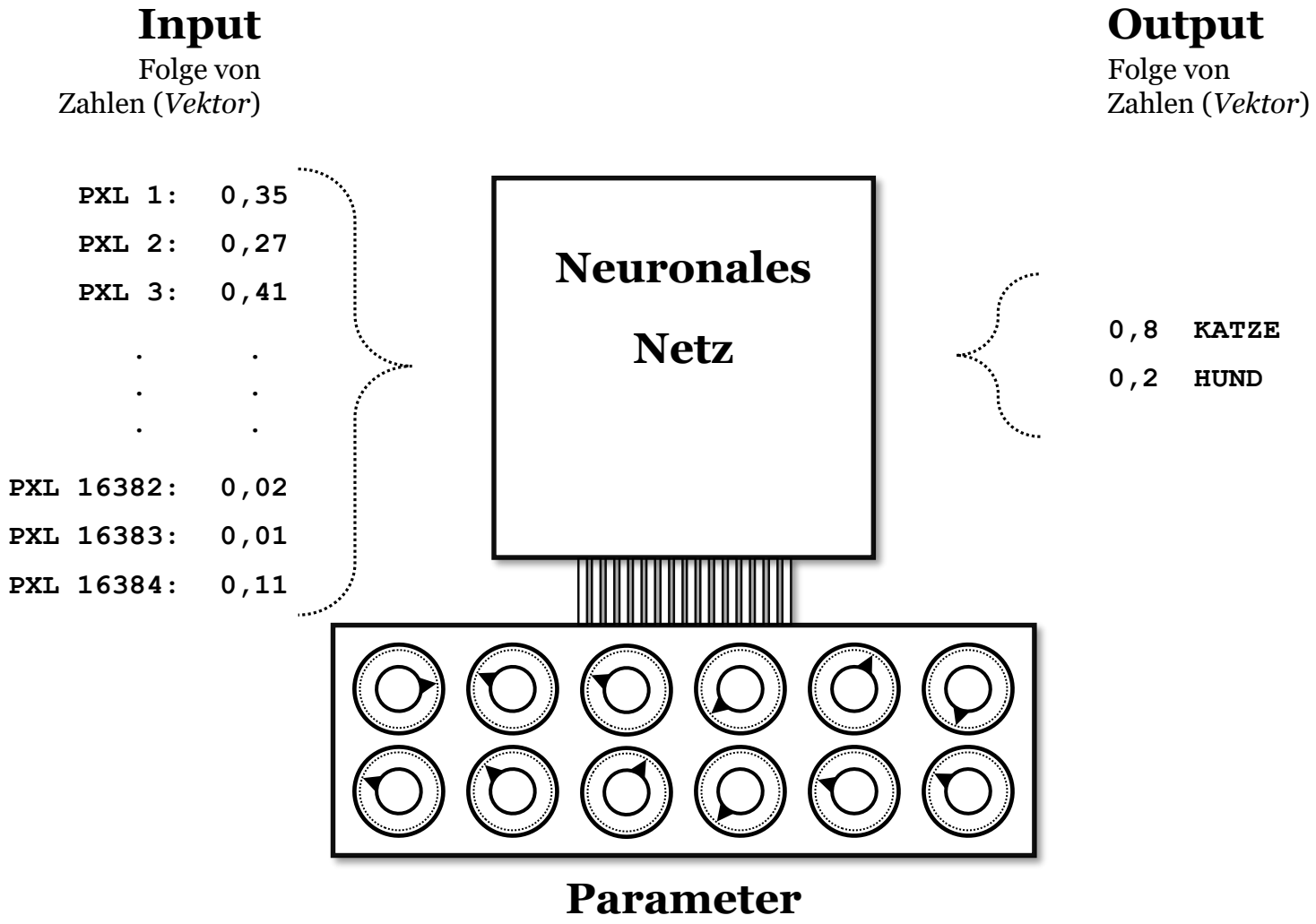
Wichtig für uns: Die Aktivierungsfunktion gehört (typischerweise) nicht zu den veränderbaren Parametern.

Neuronale Netze | Level 2

Kommen wir nun zurück zu neuronalen *Netzen*.

Beispiel: Neuronales Netz, welches handschriftlich geschriebene Ziffern (0, 1, 2, ... 9) erkennen soll.

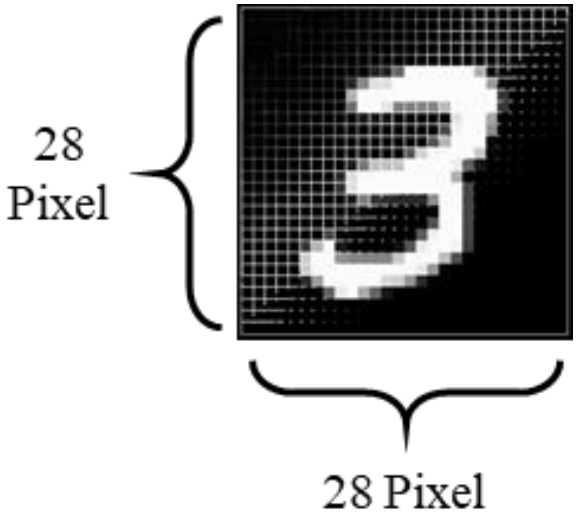


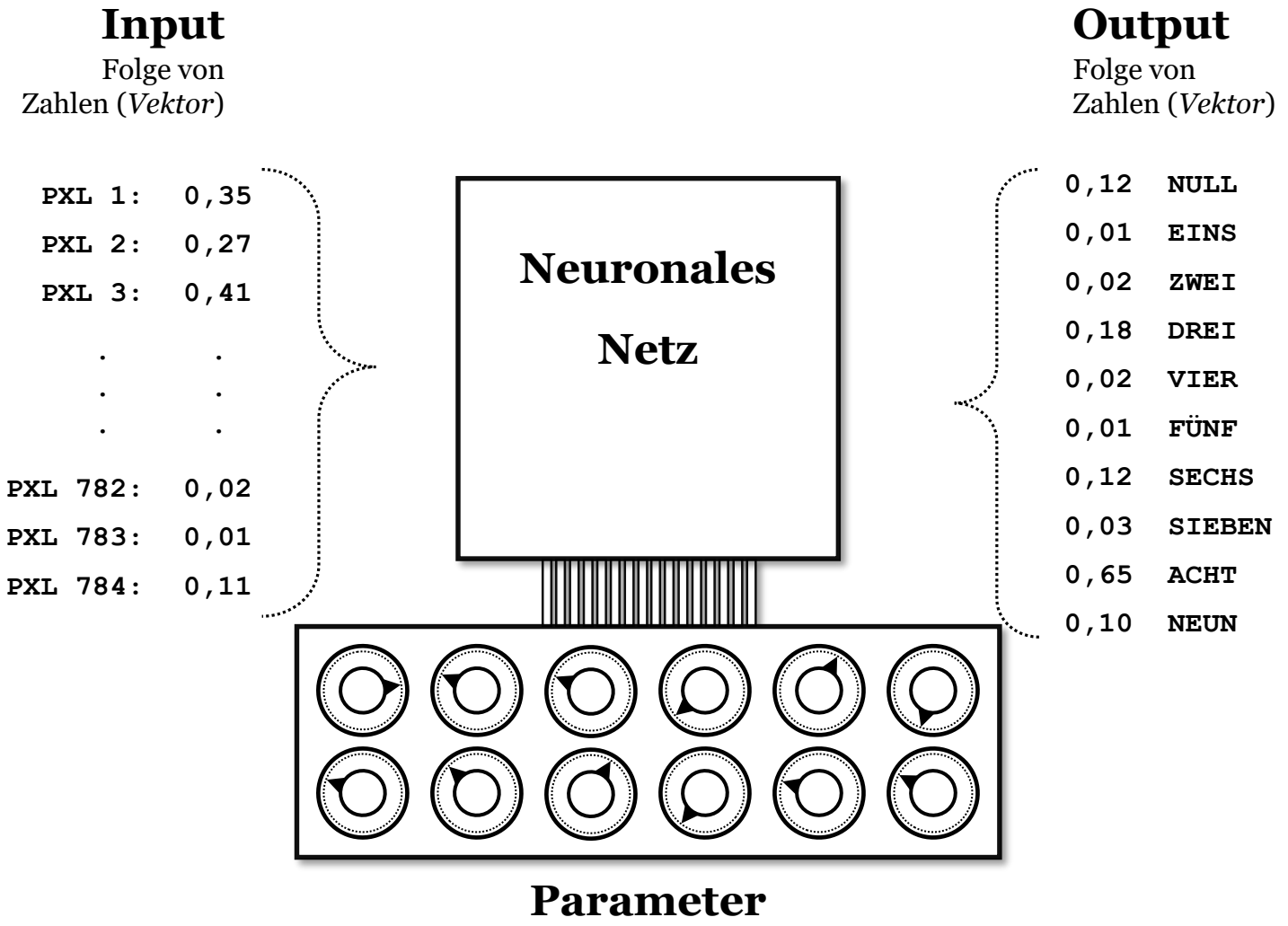


Neuronale Netze | Beispiel-Netz

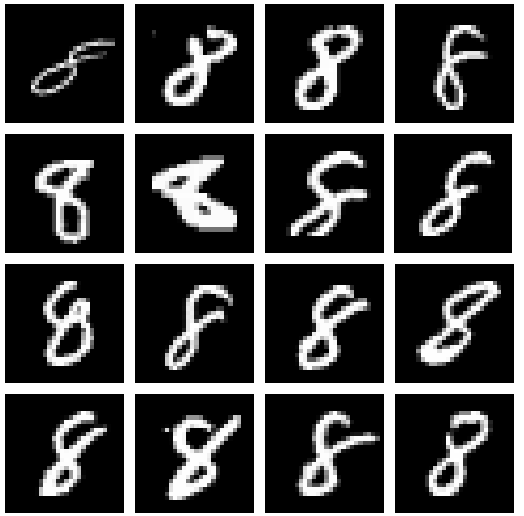
Kommen wir nun zurück zu neuronalen *Netzen*.

Beispiel: Neuronales Netz, welches handschriftlich geschriebene Ziffern (0, 1, 2, ... 9) erkennen soll.





Neuronale Netze | Beispiel-Netz



Aufgabe: Erkennung handschriftlich geschriebener Ziffern (0, 1, 2, … 9).

- Für Menschen stellt das kein großes Problem dar.
- Aber wie machen wir es genau? Formulierung *expliziter Regeln* etwa für ein klassisches Computerprogramm scheint hoffnungslos. (⇨*Symbolic vs. Sub-Symbolic AI*)
- Aber wie sich herausstellt: Neuronale Netze können dies sehr gut lernen!
- Und: Es ist nicht immer ganz leicht zu sehen, wie sie das eigentlich bewerkstelligen… (⇨*Interpretierbarkeit*)

- Um die Präsentation eines Netzes, welches diese Aufgabe bewältigen kann, vorzubereiten, machen wir zunächst ein paar allgemeine Bemerkungen zum Aufbau neuronaler Netze.

Neuronale Netze | Schichten (*Layers*)

Neuronale Netze sind typischerweise in *Schichten* (*Layers*) organisiert.

Die Schicht ganz links ist die ***Input-Layer***. Die Neuronen in dieser Schicht nehmen Werte als Inputs entgegen und geben diese nach rechts weiter.

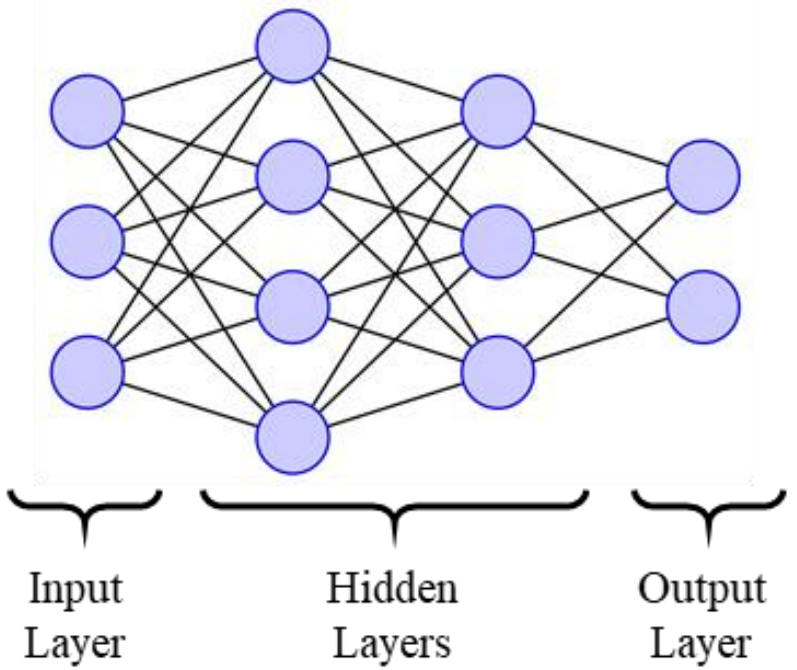
Die Anzahl der Neuronen in der Input-Layer hängt davon ab, mit wie vielen möglichen Inputs wir es zu tun haben.

In den ***Hidden-Layers*** findet in der Regel der Großteil der Berechnungen statt. Hier als Beispiel ein Netz mit zwei *Hidden Layers*.

Die letzte Schicht ist die ***Output-Layer***. Die Anzahl der Neuronen in dieser Schicht hängt davon ab, welche Art von Aufgabe das Netz lösen soll.

Ein Netz welches beispielsweise Bilder in zwei Kategorien einteilen soll (*Hund*, *Katze*) würde hier 2 Output-Neuronen umfassen.

Da wir ein Netz betrachten wollen, welches Ziffern identifiziert, wird das Netz 10 Output-Neuronen haben.



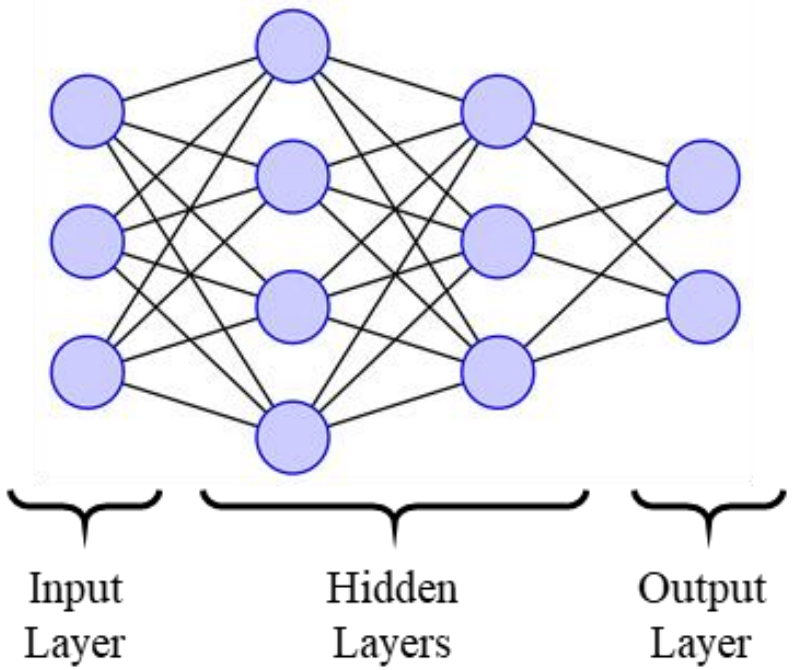
Neuronale Netze | Parameter

Damit ein neuronales Netz lernen kann, muss es sich verändern und anpassen können. Hierzu dienen die *Parameter:* Die veränderbaren Aspekte des Netzes, die im Laufe des Trainings angepasst werden.

Parameter: ***Gewichte*** und ***Biases*** der einzelnen Neuronen.

Während sich die Parameter ändern lassen, sind andere Aspekte eines neuronalen Netzes ***festgelegt/unveränderlich***.

Hierzu gehören: Die Anzahl der Neuronen, die bestehenden Verbindungen, die Organisation in Schichten, die Aktivierungsfunktionen.



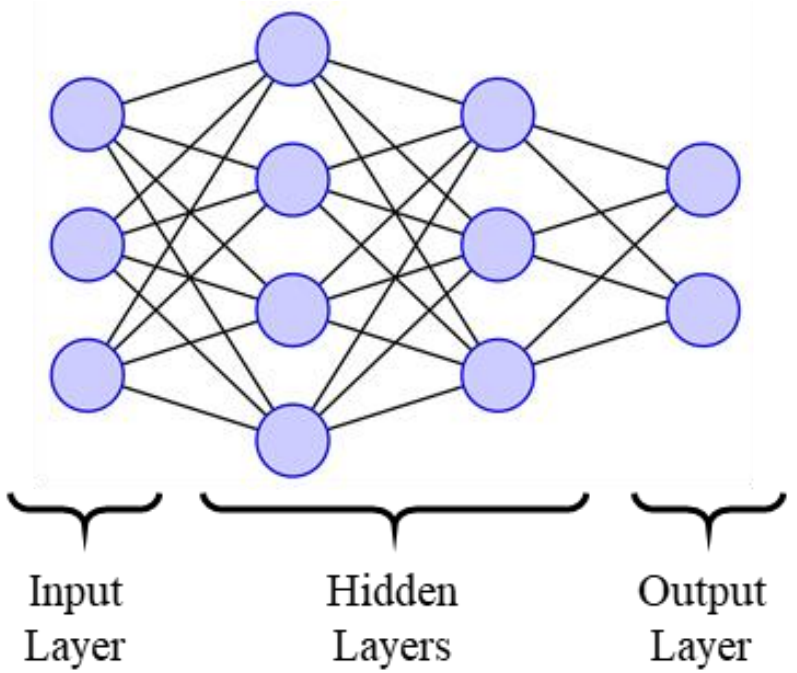
Neuronale Netze | Backpropagation

Wie bereits für einzelne Neuronen dargestellt, besteht das Lernen/Training auch bei einem komplexen Netz darin, die Parameter des Netzes im Lichte von Trainingsdaten schrittweise so anzupassen, dass der Wert einer Kostenfunktion verringert wird.

Bei Netzen mit mehreren Hidden Layers ist es allerdings nicht trivial, wie die einzelnen Parameter systematisch verändert werden sollen.

Hier war die Entwicklung des *Backpropagation*-Algorithmus historisch ein wichtiger Schritt.

Backpropagation erlaubt es, effizient zu berechnen, wie stark einzelne Parameter zum Fehler des Netzes beitragen. Auf dieser Grundlage können die Parameter dann durch Verfahren wie Gradient Descent angepasst werden.



Dadurch wurde das systematische Training von Netzen mit vielen Schichten praktikabel; dies hat wesentlich dazu beigetragen, dass neuronale Netze heute im Zentrum vieler Entwicklungen im Bereich der KI stehen.

(Auf die mathematischen Details gehen wir nicht ausführlich ein. Sehr gute weiterführende Videos hierzu finden Sie beispielsweise bei Welch Labs und 3Blue1Brown.)

Neuronale Netze | Beispiel-Netz

Das nun folgende Beispiel können Sie ausführlicher in einem YouTube-Video von *3Blue1Brown* nachvollziehen:

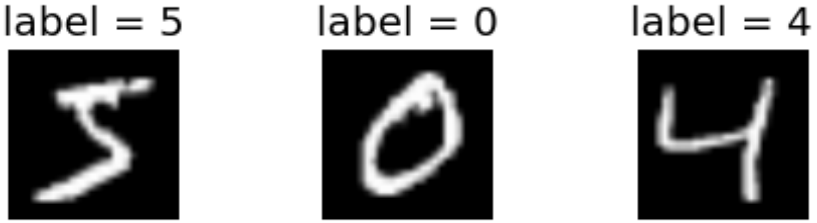
„But what is a neural network? Deep learning chapter 1“

https://www.youtube.com/watch?v=aircAruvnKk

Neuronale Netze | Beispiel-Netz

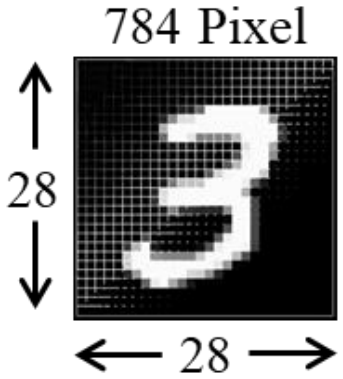
Die Aufgabe

Das Netz soll handschriftlich geschriebene Ziffern (0, 1, 2, … 9) erkennen und korrekt klassifizieren.



Die Trainingsdaten

MNIST-Datenset: Große Sammlung von handschriftlichen Ziffern (28x28 Pixel), jeweils mit *Label* (welche Zahl wird tatsächlich dargestellt?).



Neuronale Netze | Beispiel-Netz

Das Netz

Input: 784 Neuronen (je eins für jeden Pixel), nehmen die Grauwerte der entsprechenden Pixel entgegen.

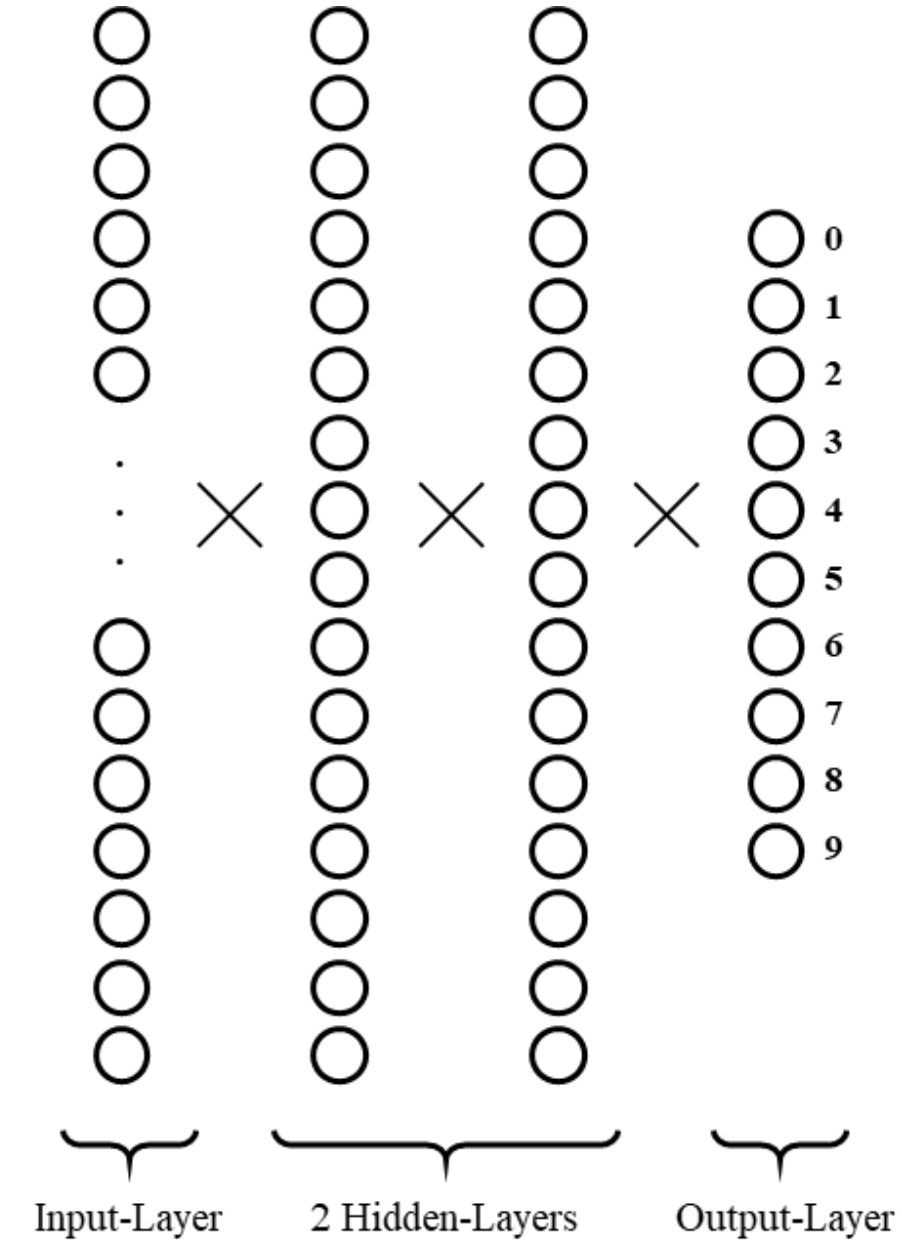
Output: 10 Neuronen (je eins für jede mögliche Klassifikation). Je höher der Wert in einem Output Neuron, desto stärker „glaubt“ das Netz, dass es sich um die entsprechende Zahl handelt.

Hidden: In diesem Fall 2 Hidden-Layers mit je 16 Neuronen.

Das × zwischen den Schichten sollen darstellen, dass alle Neuronen aus einer vorherigen Schicht mit allen Neuronen aus der folgenden Schicht verbunden sind.

Dieses Netz verfügt also bereits über ca. 13.000 Parameter (Gewichte und Biases).

(Aktivierungsfunktion: Sigmoid (führt zu Werten zwischen 0 und 1).)

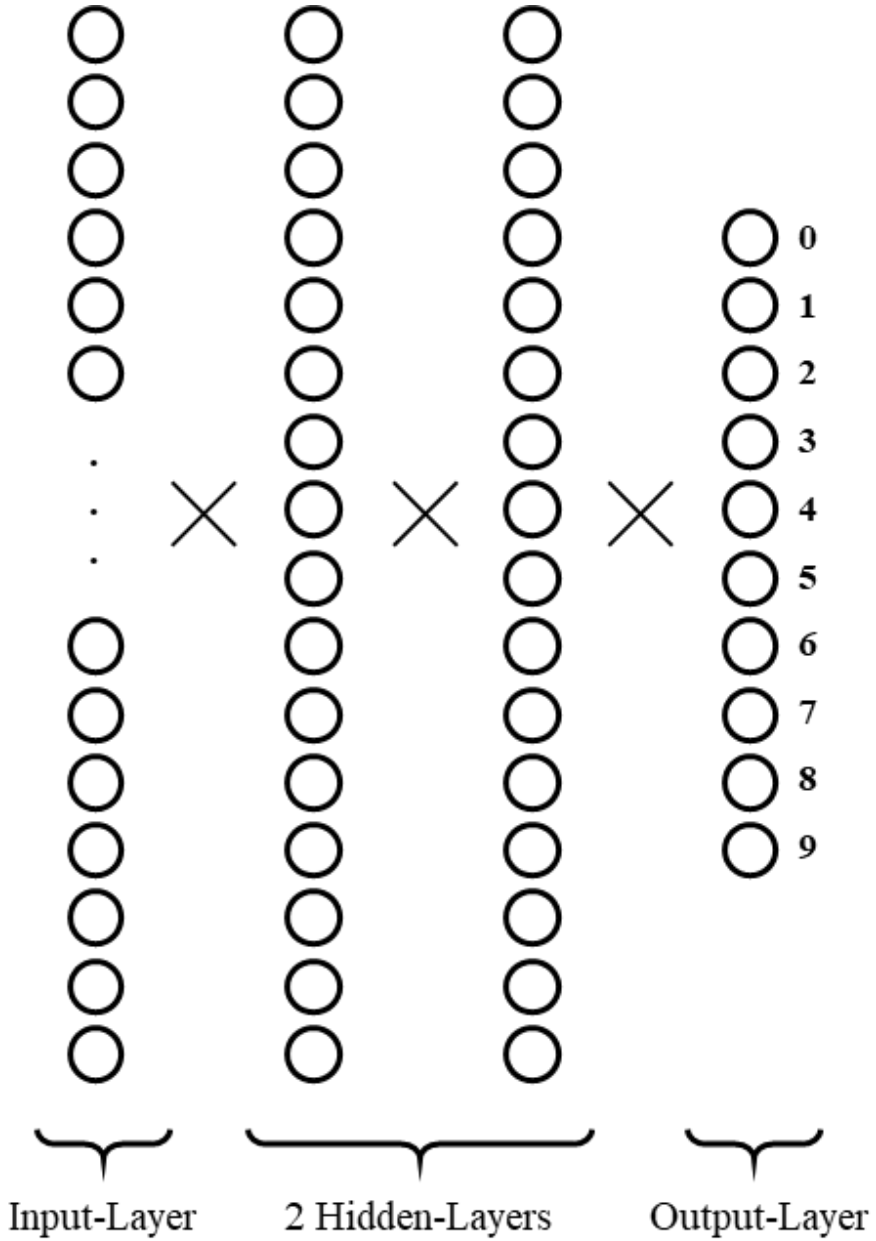


Neuronale Netze | Beispiel-Netz

Training

Zu Beginn (bevor das Training anfängt) sind alle Parameter *zufällig* gewählt.

Das Training besteht nun darin, das Netz mit den Bildern aus den Trainingsdaten zu „füttern“, und dann für jedes Bild zu schauen, wie stark das Urteil des Netzes von dem erhofften/tatsächlichen Urteil abweicht.



Neuronale Netze | Beispiel-Netz

… … …

Das tatsächliche Output für dieses Trainings-Beispiel		Das intendierte Output für dieses Trainings-Beispiel	
0,4	**0**	0	$(0{,}40 - 0)^2$
0,3	**1**	0	$(0{,}3 - 0)^2$
0,9	**2**	0	$(0{,}9 - 0)^2$
0,7	**3**	1	$(0{,}7 - 1)^2$
0,2	**4**	0	$(0{,}2 - 0)^2$
0,3	**5**	0	$(0{,}3 - 0)^2$
0,5	**6**	0	$(0{,}5 - 0)^2$
0,1	**7**	0	$(0{,}1 - 0)^2$
0,2	**8**	0	$(0{,}2 - 0)^2$
0,3	**9**	0	$(0{,}3 - 0)^2$

1,67 ← Die „Kosten“ des Netzes für dieses Trainings-Beispiel

Neuronale Netze | Beispiel-Netz

Dies wiederholen wir nun für alle Trainings-Beispiele im Datenset und ermitteln damit die ***Gesamtkosten des Netzes*** (für dieses Datenset).

Auf dieser Basis können Backpropagation & *Gradient Descent* angewendet werden, um zu ermitteln, auf welche Weise die Parameter des Netzes so geändert werden können, dass sich die Gesamtkosten reduzieren.

Wie sich zeigt, ist bereits dieses relativ einfache Beispiel-Netz in der Lage, mit ausreichendem Training ziemlich gute Ergebnisse zu liefern.

Neuronale Netze | Generalisierung

Das Ziel beim Training eines Netzes ist aber natürlich nicht bloß, im Hinblick auf die Trainingsdaten plausible Ergebnisse zu erhalten.

Idealerweise wollen wir, dass das Netz das Gelernte *übertragen* kann auf Fälle, die *nicht* in den Trainingsdaten enthalten sind.

In unserem Beispiel: Das Netz sollte in der Lage sein, auch „neue“ handschriftliche Ziffern korrekt zu klassifizieren (analog bei anderen Arten von Aufgaben).

Dies nennt man *Generalisierung*.

Was genau Generalisierung ermöglicht, ist eine substanzielle empirische Frage (siehe beispielsweise *Overfitting*).

Es wird interessant sein, auf Generalisierung auch noch einmal im Zusammenhang mit LLMs zurückzukommen.

Das Beispiel-Netz schafft sogar Generalisierung in einem hohen Maße.

Neuronale Netze | Interpretierbarkeit

Heißt das, dass wir es jetzt mit einem „intelligenten“ System zu tun haben? Mit einem System, welches *weiß*, was handschriftlich geschriebene Ziffern sind und wie man diese erkennt?

(Oder entsprechend bei komplexeren Netzen, die Katzen, Hunde, Goldfische etc. identifizieren können – *wissen* diese Systeme, was ein Goldfisch ist?)

Die Beantwortung solcher Fragen wird enorm dadurch erschwert, dass neuronale Netze oft nicht direkt für uns *interpretierbar* sind – es ist oft nicht leicht zu verstehen, was das Netzt eigentlich genau macht und wie es zu seinen Ergebnissen kommt (jedenfalls auf einer für uns Menschen informativen Beschreibungsebene).

In Bezug auf Netze, die Bilder klassifizieren, lässt sich dieser Umstand beispielsweise durch „Optimised Inputs“ illustrieren (andere Beispiele können wir uns dann auch noch in Bezug auf LLMs anschauen).

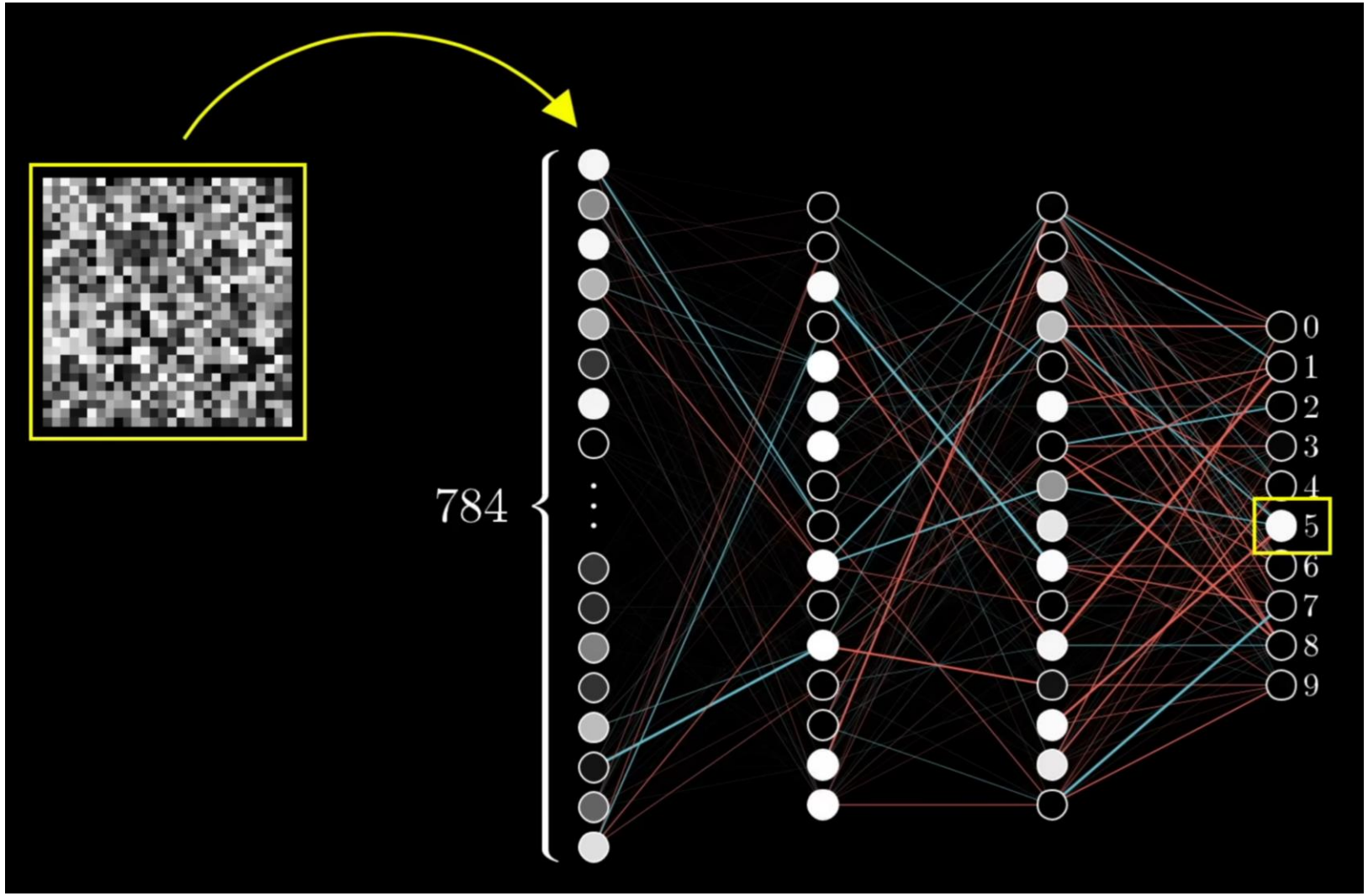
Das Phänomen besteht hier darin, Inputs zu finden, die für ein bestimmtes Output optimiert sind (also hier einen maximal hohen Output-Wert erzielen), die aber dennoch in keinem erkennbaren Zusammenhang zu der intuitiven Klassifikation stehen, auf die wir eigentlich abgezielt hatten.

Neuronale Netze | Interpretierbarkeit

Unser Beispiel-Netz ist sich „sicher“, dass das hier eine 5 darstellt!

Das scheint zu bedeuten, dass das Netz einen ganz *anderen* Begriff einer 5 hat als wir (wenn es überhaupt einen hat…).

(Oder ist dies vielleicht lediglich ein Analogon zu einer „optischen Täuschung“? Die gibt es bei uns Menschen ja schließlich auch…)



3Blue1Brown: „But what is a neural network? Deep learning chapter 1“. https://www.youtube.com/watch?v=aircAruvnKk

Neuronale Netze | Interpretierbarkeit

Hier ein Beispiel mit Bezug auf ein komplexeres *Image-Classifier*-Netz.

Dieses Netz gibt für dieses Bild den maximalen Wert für *Goldfish*.

Goldfish

Tim Sainburg, "Classes visualized from a pretrained VGG network," in Visualizing features, receptive fields, and classes in neural networks from "scratch" with Tensorflow 2. Part 1: Visualizing classes, 19 May 2020. Cropped detail of the VGG16 "goldfish" class visualization.

Neuronale Netze | Interpretierbarkeit

Logic-based AI is the most ambitious approach to AI, because it proposes to understand the common sense world well enough to express what is required for successful action in formulas. Other approaches to AI do not require this. Anything based on neural nets, for example, hopes that a net can be made to learn human-level capability without the people who design the original net knowing much about the world in which their creation learns. Maybe this will work, but then they may have an intelligent machine and still not understand how it works. This prospect seems to appeal to some people.

John McCarthy (1996): „What Computers Still Can't Do“ (Review des gleichnamigen Buches von Hubert Dreyfus)

Zusammenfassung

Halten wir kurz ein paar Punkte aus der heutigen Sitzung fest:

- Künstliche Neuronen können wir als mathematische Funktionen verstehen: Sie bilden zunächst eine gewichtete Summe der Inputs, addieren einen Bias-Term (und wenden darauf eine Aktivierungsfunktion an).
- Die ***Gewichte***, mit denen die Inputs multipliziert werden, sowie der ***Bias***, der hinzuaddiert wird, sind die ***Parameter*** des Neurons.
- Die Parameter sind die „Stellschrauben“, an denen während des Trainings eines Netzes „gedreht“ werden kann.
- Erfolgreiches Lernen bzw. Training besteht darin, eine Parametereinstellung zu finden, durch die Inputs aus dem relevanten Problembereich mit passenden Outputs verknüpft werden.
- Im Idealfall gelingt dies nicht nur für die Trainingsbeispiele, sondern auch für neue, bisher nicht gesehene Beispiele (***Generalisierung***).

Zusammenfassung

Halten wir kurz ein paar Punkte aus der heutigen Sitzung fest:

- Künstliche neuronale Netze sind aus künstlichen Neuronen zusammengesetzt, die typischerweise in Schichten — Layers — organisiert sind.
- Mathematisch entspricht ein neuronales Netz ebenfalls einer Funktion: Es nimmt eine Eingabe entgegen und erzeugt daraus eine Ausgabe.
- Diese Gesamtfunktion ergibt sich aus der Verknüpfung der Rechenoperationen der einzelnen Neuronen und Schichten.
- Für das Training von mehrschichtigen Netzen war die Entdeckung des Backpropagation-Algorithmus zentral. Backpropagation erlaubt es, effizient zu berechnen, wie stark einzelne Parameter zum Fehler eines Netzes beitragen. Auf dieser Grundlage können Parameter dann durch Verfahren wie *Gradient Descent* angepasst werden.