# Eine Unterrichtssequenz zum Einstieg in Konzepte des maschinellen Lernens

Vorname1 Nachname1[1], Vorname2 Nachname2[2] und weitere Autorinnen und Autoren in der gleichen Notation

**Abstract:** In diesem Beitrag soll ein Weg aufgezeigt werden, um Grundprinzipien maschinellen Lernens so zu reduzieren, dass sie in der Schule praktisch umzusetzen sind. Dafür werden LEGO-NXT-Roboter so programmiert, dass sie „aus Erfahrungen lernen".

**Keywords:** LEGO-Roboter, maschinelles Lernen, Entmystifizierung des „schlauen" Computers, Konditionierung, Hebb-Regel

## 1 Entmystifizierung des „schlauen" Computers

Grundschüler[3], denen ein Computer Rechenaufgaben stellt und ggf. ein positives Feedback gibt, wurden gefragt, weshalb der Computer immer die korrekte Lösung kennt oder ob er sich auch irren kann. Die einvernehmliche Antwort war, dass Computer sich nicht irren, weil sie „schlau" sind, zumindest schlauer als die Schülerinnen und Schüler. Erfahren sie später in der Sekundarstufe, wenn sie z.B. mit digitaler Elektronik einen Parallel-Addierer im Informatikunterricht gebaut haben, dass sich ein Computer deshalb nicht „verrechnen" kann, weil die Leitungen starr gesteckt sind und deterministisch immer derselben Funktionalität folgen, dann erscheint ihnen diese Maschine sehr viel weniger „schlau". Stattdessen könnten sie jetzt einen Schachcomputer für „intelligent" halten, bis sie lernen, dass in einem Entscheidungsbaum einfach sehr viele Möglichkeiten durchprobiert werden und dass eine der möglichen Aktionen gewählt wird, die zu einem positiven Ergebnis (Gewinn des Spiels) führen. Auch hier folgt der Rechner einem deterministischen Algorithmus. In beiden Beispielen erscheinen die durch einfache Algorithmen gesteuerten Systeme nur deshalb „intelligent", weil die Nutzer von deren Funktionsweise nichts wissen (vgl. [Br93]). Allerdings können beide Systeme noch nichts „dazulernen".

Viele Anwendungen auf dem Smartphone der Lernenden scheinen aber zu lernen. Die Wortergänzung passt sich immer mehr den speziellen Ausdrücken der Schülerinnen und Schüler an, ebenso die Autovervollständigung und weitere. Erst wenn Schülerinnen und Schüler im Informatikunterricht mit Datenbanken arbeiten, verstehen sie, dass es sich hier um einfache Erweiterungen der Datenbasis handelt, also um INSERT-Befehle unter

[1] Einrichtung/Universität, Abteilung, Anschrift, Postleitzahl Ort, emailadresse@author1
[2] Einrichtung/Universität, Abteilung, Anschrift, Postleitzahl Ort, emailadresse@author2
[3] Es wurde eine zweite Klasse mit 21 Grundschülern befragt. Aus dem Ergebnis kann keine allgemeingültige Aussage abgeleitet werden.

bestimmten Bedingungen. Der scheinbare Wissenserwerb wird darauf reduziert, dass sich eine Datenbasis durch das Feedback der Benutzer erweitert.

Im Informatikunterricht lassen sich also viele scheinbar „intelligente“ Systeme entmystifizieren. Das gilt auch für die künstliche Intelligenz. Dieser viel diskutierte Bereich der Informatik wird über kurz oder lang die Schulinformatik erreichen. Er erscheint sehr viel weniger mystisch, wenn gezeigt wird, dass die aktuellen Feed-Forward-Systeme im Wesentlichen Funktionen anwenden, deren Parameter durch viele Beispiele angepasst werden. Wir beschreiben hier einige Beispiele, in denen die Schülerinnen und Schüler lernen, dass auch „neuronale“ Systeme einfach nur deterministischen, vorgegebenen Regeln folgen.

## 2 Bestärkendes Lernen durch Zuordnung Wahrnehmung - Aktion

Bei dem Ansatz des bestärkenden Lernens „lernt“ ein System in einer Lernphase oder Trainingsphase anhand von Beispielen. Dazu ist ein Feedback in der Lernphase notwendig. Wir nutzen folgendes Beispiel dieses Ansatzes dazu, um die Begriffe der Wahrnehmung und Aktion einzuführen.
Wir betrachten ein System, welches mit Hilfe von Sensoren Wahrnehmungen als Eingabe erhält. Dabei sei eine Wahrnehmung eine Kombination der aktuellen Werte aller Sensoren. Gibt es beispielsweise einen Sensor, der hell und dunkel unterscheiden kann und einen Sensor, der in drei Abstufungen Lautstärke unterscheiden kann (leise, mittlere_Lautstärke, laut), dann verfügt das System über sechs verschiedene Wahrnehmungen (hell-leise, hell-mittlere_Lautstärke, hell-laut, dunkel-leise, dunkel-mittlere_Lautstärke, dunkel-laut).
In unserem Programmierbeispiel betrachten wir einen kleinen schwarzen Ball, der sich in einem farbigen Gebiet befindet (Abbildung 1). Dabei kann der Benutzer den grünen Balken mit der Maus von links nach rechts bewegen, ähnlich einem Schläger im Spiel „Pong“.

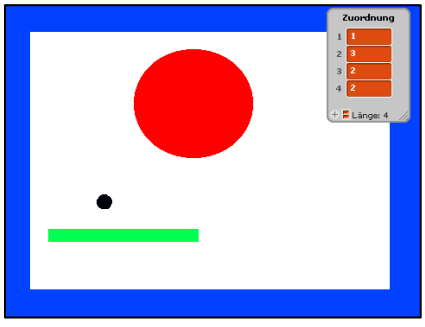


Abbildung 1 : Ball in farbigem Gebiet

Unser Ball soll vier Wahrnehmungen unterscheiden können, nämlich ob er die Farben Rot (1), Blau (2), Grün (3) oder nur Weiß (4) berührt. Weiterhin kann dieser Ball drei verschiedene Aktionen ausführen: er kann langsam vorwärtsfliegen (1), abprallen (2) oder schnell vorwärtsfliegen (3). Eine Zuordnungsliste zwischen Wahrnehmung (1,2,3,4) und Aktion (1,2,3) ist anfangs leer. Der Ball prüft ständig seine Wahrnehmung und führt die Aktion aus, die er in der Zuordnungsliste für diese Wahrnehmung findet. Findet er keine Zuordnung, führt der Ball Aktion 1 aus und „fragt“ anschließend, ob das richtig war. Verneint der Benutzer, macht der Ball die Aktion rückgängig und führt Aktion 2 aus. Auch jetzt fordert er das Feedback des Benutzers. Wir gehen davon aus, dass eine der ausgeführten Aktionen ein positives Feedback bekommt. Diese Aktion wird in der Zuordnungsliste gespeichert. Am Ende der Trainingsphase hat der Roboterball jeder Wahrnehmung eine Aktion zugeordnet, die er fortan ausführt. Damit ist das Spiel ganz unterschiedlich konfigurierbar. Der Ball könnte bei Grün (am Schläger) abprallen und bei Blau (am Rand) und bei Rot seine Geschwindigkeit erhöhen. Er könnte aber auch so trainiert werden, dass er bei Rot abprallt und bei Grün seine Geschwindigkeit erhöht, usw.

## 3 Systeme mit „Neuron“

In dem obigen Beispiel „lernte“ unser Ball sein Verhalten durch Feedback von außen. Jetzt wollen wir Systeme betrachten, die ihre Konfiguration selbst so verändern, dass sie in ihrer Umwelt optimal (re-)agieren (vgl. [MP47]). Unser Beispiel entspricht der klassischen Konditionierung des Pawlowschen Hundes [PH19].

Wir modellieren ein Neuronales Netz als gerichteten Graphen, wobei die Knoten Zellkörper repräsentieren und die Kanten die Verbindungen zwischen den Neuronen. Im Inneren eines Knotens wird ein Schwellenwert notiert. Die Kanten sind gewichtet. Es existieren außerdem Neuronen, die Signale aus der Außenwelt erhalten. Wir verwenden nur binäre Signale (Null oder Eins). Die Ausgabefunktion eines Neurons ist folgendermaßen definiert: ist die Summe aller Eingaben multipliziert mit den zugehörigen Kantengewichten größer oder gleich dem Schwellenwert, so ist die Ausgabe Eins, sonst Null. Abbildung 2-4 veranschaulicht die Modellierung.

Im Folgenden betrachten wir stets nur ein Neuron. Wir können das logische ODER bereits als erste Programmierübung auf dem NXT-Roboter implementieren. In der Schule hilft dies, um sich die Funktionsweise langsam zu erarbeiten, hier können wir den Code aus Platzgründen nicht wiedergeben, er ähnelt aber dem Code im nächsten Kapitel, nur ohne Änderung der Gewichte.

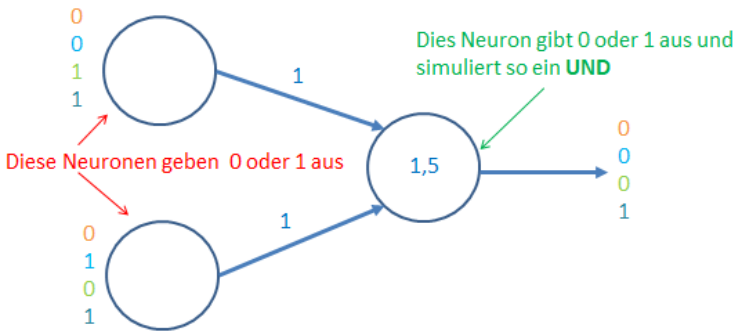


Abbildung 2: Beispiel UND

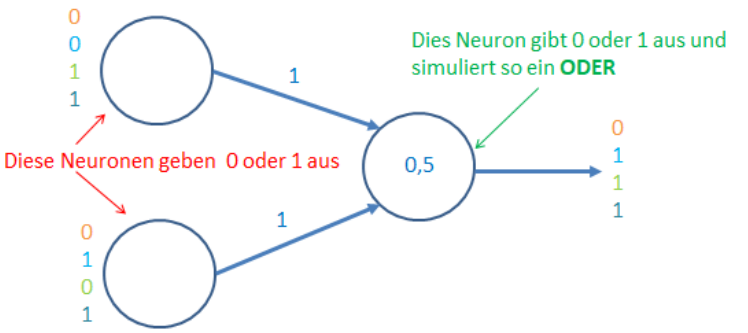


Abbildung 3 : Beispiel ODER

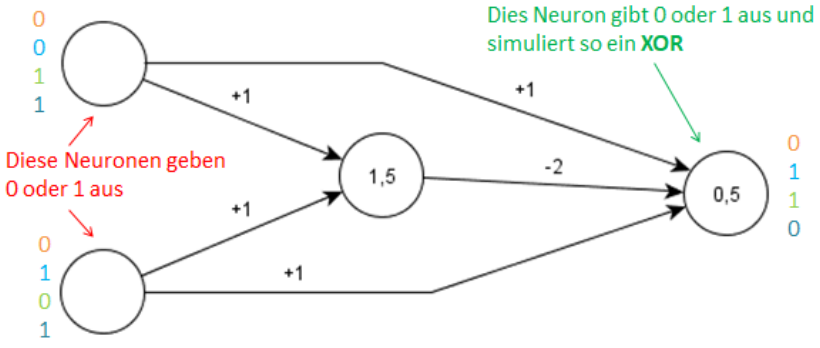


Abbildung 4 : Beispiel XOR

# 4 Der Roboter „lernt“

Unser Roboter soll lernen, aber wie lernt ein Neuronales Netz? Prinzipiell durch die Entwicklung neuer Verbindungen, Löschen bestehender Verbindungen, Ändern der Gewichtung (der Gewichte von Neuron i zu Neuron j), Anpassen der Schwellenwerte der Neuronen, Hinzufügen von Neuronen oder Löschen von Neuronen (vgl. [Wö06]). In unserem reduzierten Modell betrachten wir im Folgenden nur die Möglichkeit der Änderung der Kantengewichte. Dafür gibt es viele verschiedene Lernalgorithmen. Wir verwenden hier die Hebb-Regel, die besagt, dass das Gewicht zwischen zwei Neuronen dann erhöht wird, wenn beide Neuronen gleichzeitig aktiv sind, also beide Neuronen eine Eins am Ausgang liefern (vgl. [Ro96]). Die Erhöhung der Gewichte zwischen den Knoten i und j wird mit einem + dargestellt.

| Aktivität Knoten i | Aktivität Knoten j | Hebb-Regel |
|---|---|---|
| 0 | 0 | = |
| 0 | 1 | = |
| 1 | 0 | = |
| 1 | 1 | + |

Abbildung 5 : Hebb-Regel

Betrachten wir unseren NXT-Roboter mit einem „Neuron“.

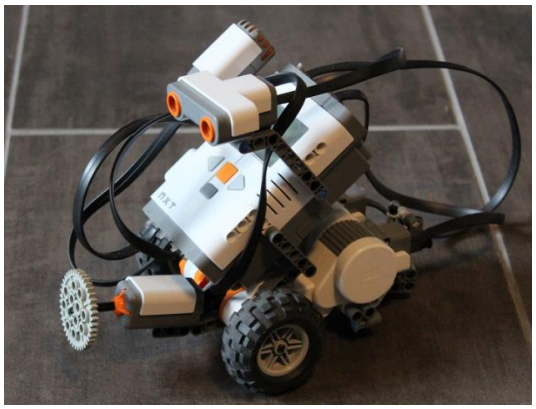

Abbildung 6 : unser Roboter mit Sensoren und Aktoren

Der Roboter besitzt zwei Motoren, mit deren Hilfe er fahren kann. Die beiden möglichen Aktionen sind „vorwärts“ oder „umdrehen“:

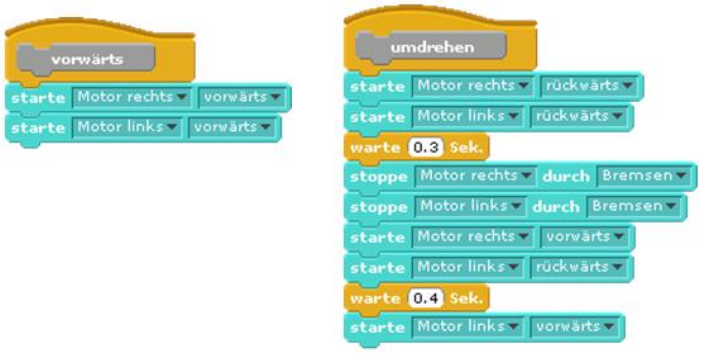


Abbildung 7 : Roboteraktionen

Unser Roboter fährt geradeaus. Liefert sein Neuron eine Eins, so dreht er um. Das Neuron hat nun drei Eingänge. Eingang 1 liefert Eins, wenn der Geräuschsensor laut registriert, Null sonst. Eingang 2 liefert Eins, wenn der Ultraschallsensor nah registriert, sonst Null. Eingang 3 ist Eins, wenn der Berührsensor gedrückt ist und Null sonst.


Abbildung 8 : Wahrnehmungen des Roboters

Mit unserer Lernregel nach Hebb gilt: liefert eine der Eingaben den Wert Eins (ist also aktiv) und das Neuron liefert ebenfalls den Wert Eins, so erhöht sich das Kantengewicht

zwischen Eingabe und Neuron. Für unser Neuron gelten anfangs folgende Kantengewichte:

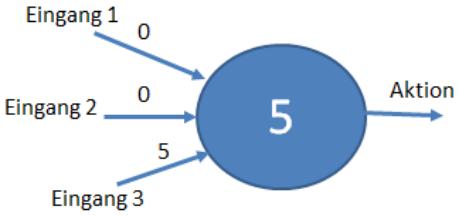


Abbildung 9 : Neuron des Roboters

Ist Eingang 3 aktiv (Berührsensor gedrückt) wird bereits der Schwellenwert erreicht und das Neuron feuert, der Roboter dreht um. Das „angeborene“ Verhalten ist also ein Umdrehen, wenn der Roboter z. B. gegen eine Wand fährt. Die anderen Sensoren beeinflussen die Aktivität des Neurons bisher nicht. Setzen wir nun folgenden Algorithmus um:

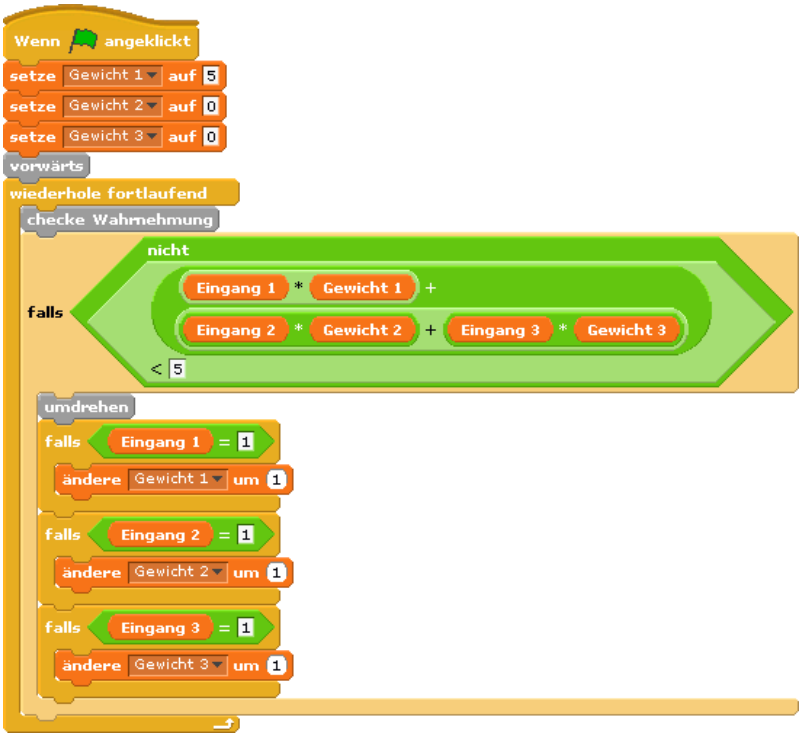


Abbildung 10 : Lernalgorithmus

Man kann Folgendes beobachten: Der Roboter fährt geradeaus. Wenn er gegen eine Wand fährt (Berührsensor löst aus), dreht er um und fährt wieder geradeaus. Das Ganze passiert mehrmals hintereinander. Nach einigen Kollisionen scheint der Roboter etwas „gelernt“ zu haben, denn er dreht schon um, bevor er die Wand berührt. Jedes Mal, wenn das Neuron aktiv ist, also bedingt durch den Berührsensor umdreht, liefert Eingang 2 auch eine Eins, weil der Ultraschallsensor registriert, dass etwas nah ist. Nach der Lernregel wird dadurch die Gewichtung zwischen Eingang 2 und dem Neuron erhöht. Irgendwann erreicht das Gewicht so den Wert Fünf. Jetzt dreht der Roboter bereits um, wenn Eingang 2 eine Eins liefert (eine Wand nah ist), weil die Summe den Schwellenwert erreicht und das Neuron feuert. Mit den Worten eines Schülers: Nachdem sich der Roboter ein paar Mal die Nase gestoßen hat, fährt er nicht mehr gegen eine Wand, sondern dreht vorher um[4]. Der Roboter ist konditioniert wie der Pawlowsche Hund.

## 5 Der „intelligente“ Ball

Steht kein Roboter zur Verfügung, kann man auch andere Systeme zum „Lernen“ bringen. Unser Beispiel ist das Spiel „Pong“ mit einem „intelligenten“ Ball. Unser Ball kann zwei verschiedene Aktionen ausführen: fliegen oder abprallen (Der Code ist aus Platzgründen weggelassen, ist aber wenig komplex).
Außerdem hat sein Neuron zwei Eingänge. Eingang 1 liefert nur dann eine Eins, wenn die Farbe Schwarz berührt wird, Eingang 2 nur dann eine Eins, wenn die Farbe Rot berührt wird. Auch hier lassen wir aus Platzgründen den Code weg, er ist aber ähnlich zu dem Befehl „checke Wahrnehmung“ des Roboters.

Unser Schläger hat drei verschiedene „Kostüme“. Er kann schwarz sein, rot oder rot-schwarz gestreift.

Abbildung 11 : drei verschiedene Kostüme

[4] Natürlich wachsen die Gewichte immer weiter, was unnötig ist, wenn der Wert Fünf überschritten ist. Auch kann der Roboter nicht wieder „verlernen“. Hier finden die Schüler aber aller Erfahrung nach selbstständig individuelle Lösungen.

Wir implementieren analog zu unserem Roboter folgende Funktionalität (siehe Abbildung 11).

Was passiert? Ist der Schläger schwarz, kann man damit das ganz normale Pong spielen. Ist der Schläger nur rot, fällt der Ball hindurch und prallt nicht am Schläger ab. Wählt der Benutzer den rot-schwarz-gestreiften Schläger, so prallt der Ball ab (der schwarzen Farbe wegen). Er berührt dabei aber gleichzeitig auch rot. Der Ball „lernt“, auch bei Rot abzuprallen, denn wenn der zweifarbige Schläger eine längere Zeit verwendet wird und danach der nur rote Schläger, prallt der Ball auch an diesem Schläger ab.

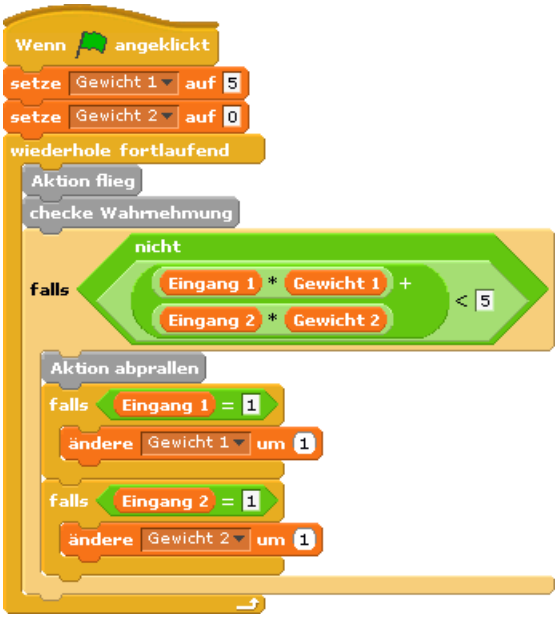


Abbildung 12 : Lernalgorithmus

## 6 Fazit

In diesem Beitrag haben wir eine mehrfach erprobte Unterrichtssequenz aufgezeigt, die didaktisch sehr reduziert die Grundprinzipien maschinellen Lernens verdeutlicht und die praktisch umsetzbar ist. Dabei wurden NXT-Roboter so programmiert, dass sie scheinbar „aus Erfahrungen“ lernen. Diese Sequenz soll einen ersten Einblick in die Grundprinzipien neuronaler Netze ermöglichen. Hinzu kommt die Erkenntnis der Schülerinnen und Schüler, dass auch diese einfach nur deterministischen Regeln folgen und Algorithmen abarbeiten. Sehr viel größere Netze, die heute durch die hohe

Rechenleistung der Rechner zu realisieren sind, verarbeiten auch sehr viel komplexere Situationen, arbeiten aber nicht prinzipiell anders (vgl. [Go16]). Die Frage, ob das menschliche Gehirn prinzipiell mehr leisten kann oder nicht bleibt (natürlich) unbeantwortet, erschließt sich den Schülerinnen und Schülern aber als Fragestellung.

## Literaturverzeichnis

[Br93] Valentin Braitenberg: “Vehikel – Experimente mit kybernetischen Wesen” Rowohlt-Verlag, 1993

[Go16] Ian Goodfellow, Yoshua Bengio, Aaron Courville : “Deep Learning”, MIT Press, 2016

[MP47] W. McCulloch, W. Pitts: „How We Know Universals: the Perception of Auditory and Visual Forms”, Bulletin of Mathematical Biophysics, Vol. 9, 1947

[PH19] https://de.wikipedia.org/wiki/Pawlowscher_Hund, Zugriff: 30.1.19

[Ro96] Raul Rojas: “Neural Networks. A Systematic Introduction”, Springer-Verlag, 1996

[Wö06] P. Manoonpong, T. Gend, F.Wörgötter: „Exploring the dynamic walking range of the biped robot RinBot with an active upper-body component”, IEEE-RAS International Conference of humanoid robots, 2006