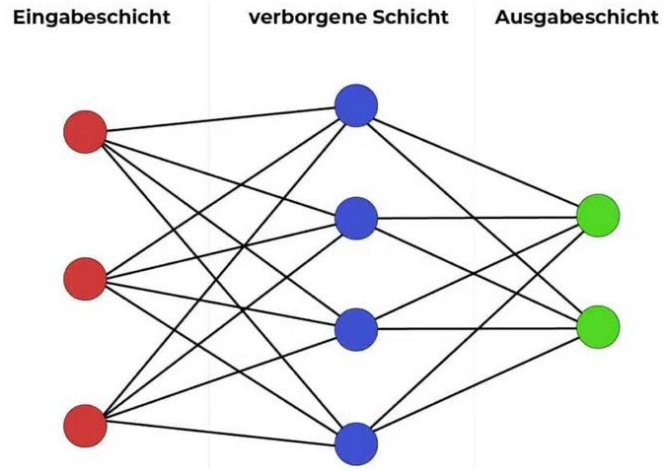
# Neuronale Netze mit fischertechnik


Seit kurzem gibt es einen neuen ft-Baukasten, mit dem Namen STEM Coding Ultimate AI. Dazu gehört als Software die STEM-Suite, die kostenlos heruntergeladen werden kann. In dieser Software sind die Bauanleitungen, die entsprechenden Anleitungen und eine Version vom ROBO Pro Coding enthalten. Als wichtigste Neuerung von RPC ist wohl, das nun auch mit Neuronalen Netzen experimentiert werden kann.
Zur besseren Einordnung erst einmal einige Begriffsbestimmungen.

**Künstliche Intelligenz (KI)**
Das breite Feld, das sich mit der Entwicklung von Maschinen beschäftigt, die Aufgaben ausführen können, die normalerweise menschliche Intelligenz erfordern, wie z.B. das Erkennen, Lernen und Problemlösen.
**Maschinelles Lernen**
Ein Teilbereich der KI, der sich auf die Entwicklung von Algorithmen konzentriert, die es Computern ermöglichen, aus Daten zu lernen und Vorhersagen zu treffen.
**Deep Learning**
Ein Teilbereich des maschinellen Lernens, der neuronale Netzwerke mit vielen Schichten (daher „*tief*“) verwendet, um komplexe Muster in großen Datensätzen zu modellieren.
**Neuronale Netze**
Sie sind das grundlegende Konzept und können auch "*flache*" Strukturen mit wenigen Schichten haben.
Alle Deep-Learning-Modelle sind neuronale Netze, aber nicht alle neuronale Netze sind Deep-Learning-Modelle.

**Grundlagen Künstliches Neuronales Netzwerk**
Künstliche neuronale Netze sind Algorithmen, die dem menschlichen Gehirn nachempfunden sind. Dieses abstrahierte Modell miteinander verbundener künstlicher Neuronen ermöglicht es, komplexe Aufgaben aus den Bereichen Statistik, Informatik und Wirtschaft durch Computer zu lösen.
Neuronale Netze ermöglichen es, unterschiedliche Datenquellen wie Bilder, Töne, Texte, Tabellen oder Zeitreihen zu interpretieren und Informationen oder Muster zu extrahieren, um diese auf unbekannte Daten anzuwenden. Auf diese Weise können datenbasierte Vorhersagen für die Zukunft getroffen werden.




Das Modell des Neuronalen Netzes besteht aus Knoten, auch Neuronen genannt, die Informationen von anderen Neuronen oder von außen aufnehmen, modifizieren und als Ergebnis wieder ausgeben. Dies geschieht über drei verschiedene Schichten, denen jeweils ein Typ von Neuronen zugeordnet werden kann: solche für den Input (Eingabeschicht), solche für den Output (Ausgabeschicht) und so genannte Hidden Neuronen (verborgene Schichten). Die Information wird durch die Input-Neuronen aufgenommen und durch die Output-Neuronen ausgegeben. Die Hidden-Neuronen liegen dazwischen und bilden innere Informationsmuster ab. Die Neuronen sind miteinander über sogenannte Kanten verbunden. Je stärker die Verbindung ist, desto größer die Einflussnahme auf das andere Neuron.

**Eingabeschicht:** Die Eingangsschicht versorgt das neuronale Netz mit den notwendigen Informationen. Die Input-Neuronen (Wert zwischen 0 und 1) verarbeiten die eingegebenen Daten und führen diese gewichtet an die nächste Schicht weiter.
**Verborgene Schicht:** Die verborgene Schicht befindet sich zwischen der Eingabeschicht und der Ausgabeschicht. Während die Ein- und Ausgabeschicht lediglich aus einer Ebene bestehen, können beliebig viele Ebenen an Neuronen in der verborgenen Schicht vorhanden sein. Hier werden die empfangenen Informationen erneut gewichtet und von Neuron zu Neuron bis zur Ausgabeschicht weitergereicht.
**Ausgabeschicht:** Die Ausgabeschicht ist die letzte Schicht und schließt unmittelbar an die letzte Ebene der verborgenen Schicht an. Die Output-Neuronen (Wert zwischen 0 und 1) beinhalten die resultierende Entscheidung, die als Informationsfluss hervorgeht.

**Welche Anwendungen gibt es?**
Typischerweise sind sie prädestiniert für solche Bereiche, bei denen wenig systematisches Wissen vorliegt, aber eine große Menge unpräziser Eingabeinformationen (unstrukturierte Daten) verarbeitet werden müssen, um ein konkretes Ergebnis zu erhalten. Das kann zum Beispiel in der Spracherkennung, Mustererkennung, Gesichtserkennung oder Bilderkennung der Fall sein.
Ein gutes Beispiel wird unter: https://www.youtube.com/watch?v=aircAruvnKk erklärt.

Im Folgenden habe ich das Beispiel Fahrerlose Transport Fahrzeuge aus dem STEM zusammengetragen. Der Zusammenbau des Fahrzeuges erfolgt über die Bauanleitung im STEM.

Dieses Projekt hilft dir, die Funktionsweise künstlicher neuronaler Netze (KNN) zu verstehen.
Während du Grundbegriffe eines neuronalen Netzes wiederholst, lernst du:

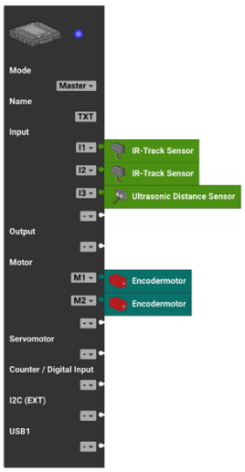
- wie ein neuronales Netz aufgebaut ist,
- was eine Klassifikation ist,
- und wie sich das von einer Mehrfach-Klassifikation (Multi-Label) unterscheidet.

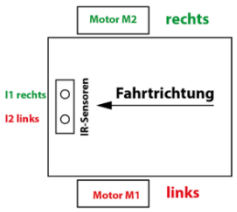
Ob in der Automobilindustrie, in Lagerhallen oder in der Lebensmittelproduktion – die FTF (**F**ahrerlose **T**ransport **F**ahrzeuge) helfen dabei, Arbeitsabläufe zu verbessern, Transportzeiten zu verkürzen und die Arbeit für Menschen sicherer und einfacher zu machen. Die Fahrzeuge fahren Wege automatisch ab, weichen Hindernissen aus und können rund um die Uhr eingesetzt werden.

Inhalt:

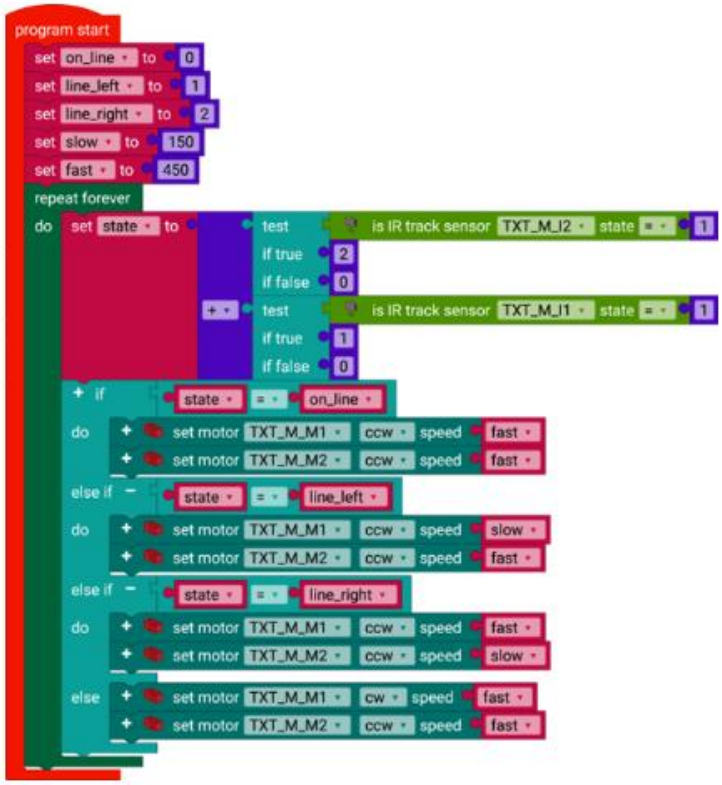


# Steuerung eines FTF – Steuerung mit einem neuronalen Netz



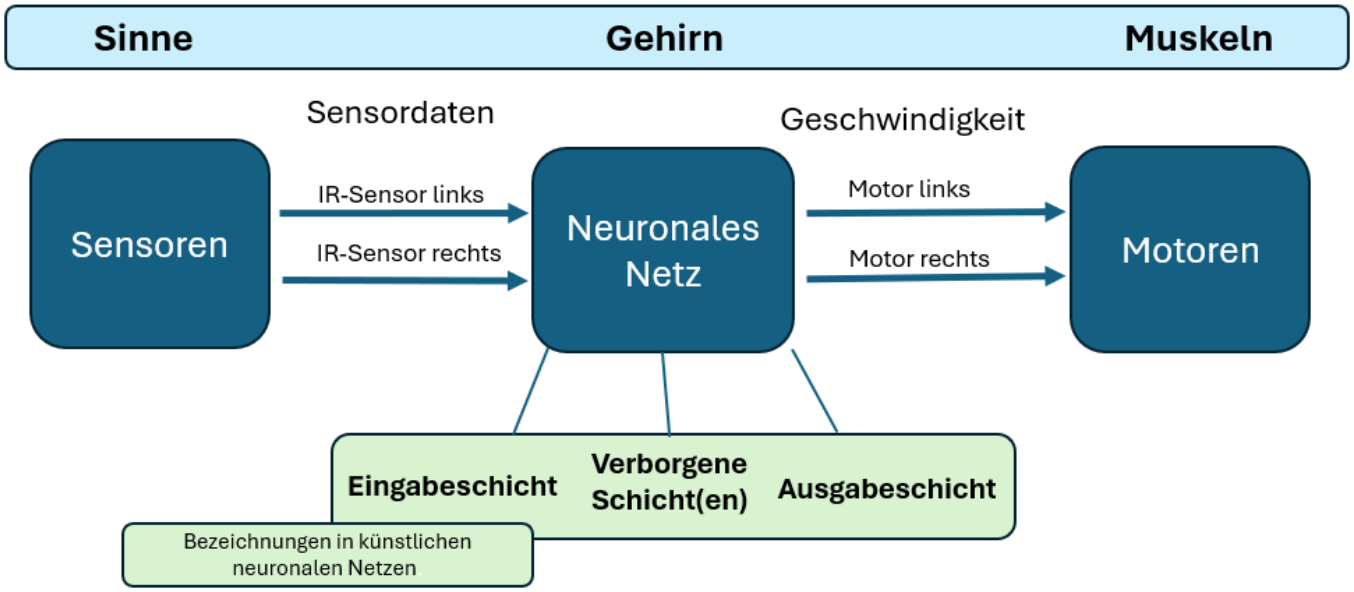




Mit den beiden IR-Sensoren (IR Track Sensor) des Spursensors erkennst du, ob das Fahrzeug sich auf der schwarzen Linie (dem gewünschten Fahrweg) befindet. Mit dem Abstandssensor (Ultrasonic Distance Sensor) erkennst du Hindernisse und den jeweiligen Abstand zum Hindernis.

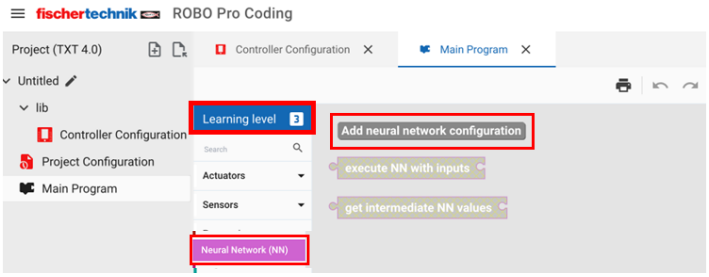



Bisher hast du den Linienfolger mit ausführlicher Programmierung realisiert.

Mit künstlicher Intelligenz in einem neuronalen Netzwerk verringert sich der Programmieraufwand und die Präzision kann gesteigert werden!

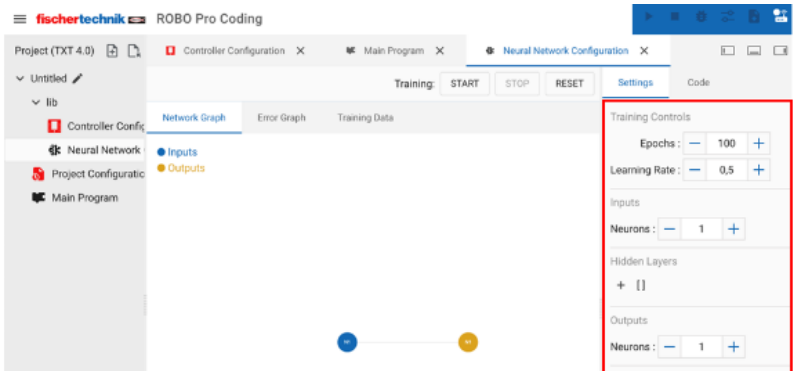



Neuronale Netze ahmen das Gehirn nach, indem sie wie Nervenzellen (Neuronen) Sinnesdaten empfangen, diese über viele Verbindungen weiterleiten und verarbeiten und daraus passende Reaktionen erzeugen – so wie unser Gehirn z. B. Muskelbewegungen oder Sprache steuert.
Überlege, wie du mit einer Fernbedienung das Fahrzeug steuern würdest!

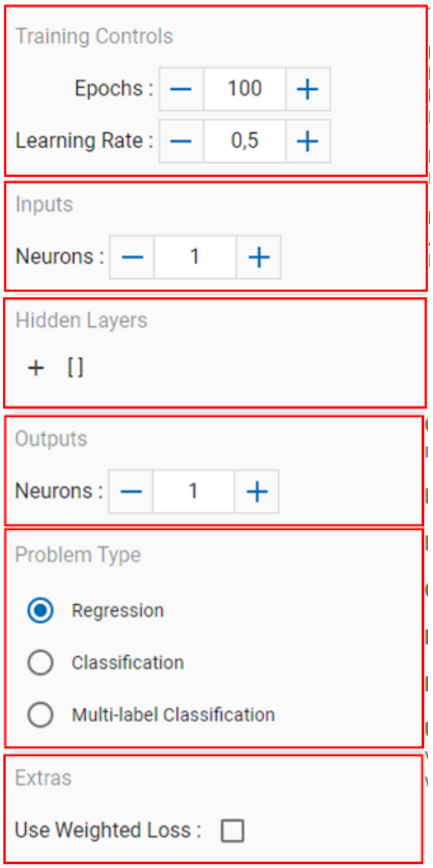



Stelle vor der Einrichtung des neuronalen Netzes sicher, dass „**Learning Level 3**“ ausgewählt ist. Neuronale Netze erzeugst du im Funktionsbereich „**Neural Network (NN)**“.

Klicke auf „**Add neural network configuration**“. Danach erscheint die Konfigurationsoberfläche für das neuronale Netz.




Hier wird ein neuronales Netz (NN) angezeigt, das nur aus einem Eingabeneuron und einem Ausgabeneuron besteht. Mithilfe der Werkzeuge auf der rechten Seite kannst du nun selbst Einstellungen vornehmen und das neuronale Netz entsprechend deiner Aufgabenstellung aufbauen.
Im rot markierten Bereich findest Du die Einstellungen für das gewünschte neuronale Netz.




**Training Controls**

**Epochs:** Gibt an, wie oft das gesamte Trainingsdatenset vom neuronalen Netz durchlaufen und gelernt wird. Mehr Epochen bedeuten oft bessere Anpassung, aber auch längere Trainingszeit – und zu viele können zu Überanpassung (Overfitting) führen.

**Learning Rate:** Bestimmt, wie stark die Gewichte des Netzes bei jedem Lernschritt angepasst werden. Große Werte = schnelleres Lernen, aber Gefahr des „Überschießens“ (man verpasst den optimalen Wert). Kleine Werte = langsamer, aber stabiler.

**Inputs:** Anzahl der Eingabeneuronen. Jedes Neuron in dieser Schicht steht für eine Eingabevariable (z. B. Temperatur, Luftfeuchtigkeit, Messwert). Die Anzahl muss zur Struktur der Eingangsdaten passen.

**Hidden Layers:** Hier können verborgene Schichten hinzugefügt werden, also die Verarbeitungsebenen zwischen Eingabe- und Ausgabeschicht. Jede Schicht kann eine beliebige Anzahl an Neuronen haben, um komplexere Muster zu erkennen. Mehr Schichten und Neuronen = höhere Modellkapazität, aber auch mehr Rechenaufwand und Risiko von Überanpassung.

**Outputs:** Anzahl der Ausgabeneuronen. Bei einer Regression meist 1 Neuron je Aktor (gibt eine Zahl zurück). Bei Klassifikation = Anzahl der möglichen Klassen (z. B. 3 Neuronen für „rot“, „gelb“, „grün“).

**Problem Type**

**Regression:** Vorhersage von kontinuierlichen Werten (z. B. Temperatur, Geschwindigkeit, Entfernung).

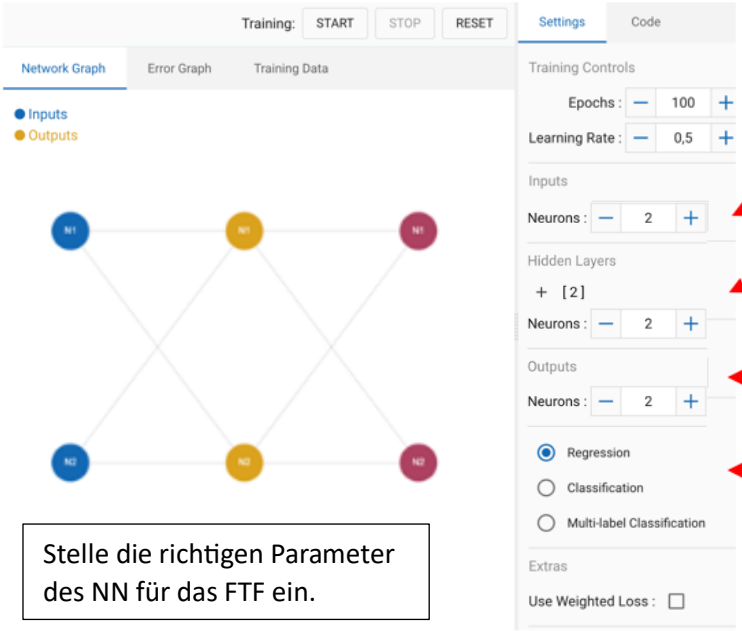
**Classification:** Vorhersage von Klassen, bei denen jedes Beispiel genau einer Klasse zugordnet wird (z. B. Fußgänger oder Rollstuhlfahrer).

**Multi-label Classification:** Einem Beispiel können mehrere Klassen gleichzeitig zugeordnet werden (z. B. ein Bild kann „Ball“ und „rot“ sein).

**Extras**

**Use Weighted Loss** aktiviert gewichtete Fehlerbewertung. Das ist nützlich, wenn manche Klassen oder Werte im Trainingsdatensatz seltener vorkommen und man verhindern will, dass das Netz diese vernachlässigt. Beispiel: Bei einer Klassifikation mit 90 % Klasse A und 10 % Klasse B würde ein Standardnetz Klasse A bevorzugen – durch Gewichtung kann man beide Klassen gleich wichtig machen.

## Konfiguration des neuronalen Netzwerkes für das FTF




Anzahl der Eingangsneuronen: Anzahl der Sensoren, deren Werte verarbeitet werden sollen (hier: IR-Sensor rechts und links).

Zur Verarbeitung wird hier 1 Hidden Layer mit 2 Neuronen eingesetzt.

Anzahl der Ausgangsneuronen: Anzahl der Aktoren (Motoren)

Problemtyp: Regression, jeder Motor soll einen Wert erhalten.

Stelle die richtigen Parameter des NN für das FTF ein.

Im Regressionsverfahren sagt das NN bestimmte **Zahlenwerte** voraus, hier: die Motorgeschwindigkeiten.

Was soll bei verschiedenen Sensorzuständen an den Ausgängen passieren?

ADD ROW IMPORT EXPORT

| Input 1 | Input 2 | Output 1 | Output 2 | |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | × |
| 0 | 1 | 0,9 | 0,2 | × |
| 1 | 0 | 0,2 | 0,9 | × |
| 1 | 1 | 0 | 0 | × |
| IR-Sensor links | IR-Sensor rechts | Motor rechts | Motor links | |

Voll auf Linie:
Motoren fahren Höchstgeschwindigkeit

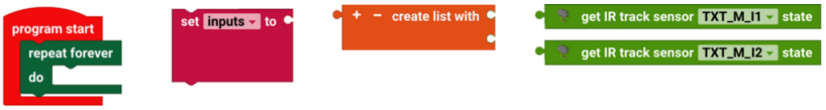
Links auf Linie, rechts neben Linie:
Linkskurve: linker Motor langsam, rechter Motor schnell

Links neben Linie, rechts auf Linie:
Rechtskurve: linker Motor schnell, rechter Motor langsam

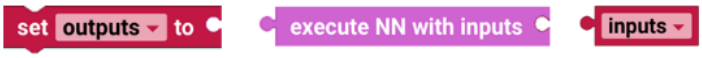
Beide Sensoren nicht auf Linie:
Die Motoren stoppen.

Für das Programm für die Linienfolge musst du zunächst zwei Variablen definieren:


Die 2 Input-Werte werden nun in in der Reihenfolge des angelegten neuronalen Netzes in eine Liste geschrieben.



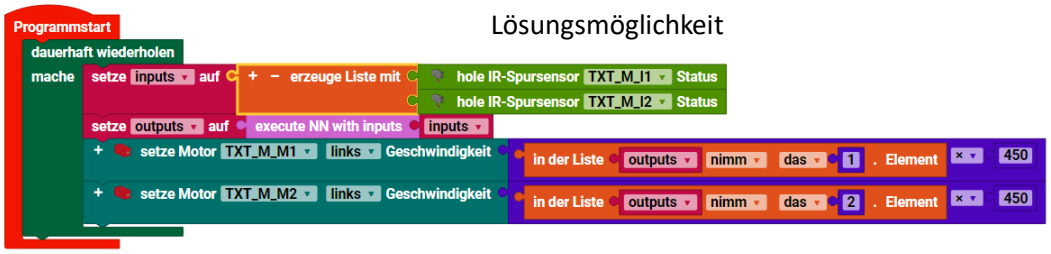

Der Block „**execute NN with inputs**“ erzeugt ebenfalls eine Liste mit Output 1 an erster Stelle und Output 2 an zweiter Stelle. Hier werden dann die vom neuronalen Netz ermittelten Zahlenwerte zwischen 0 und 1 ausgegeben.

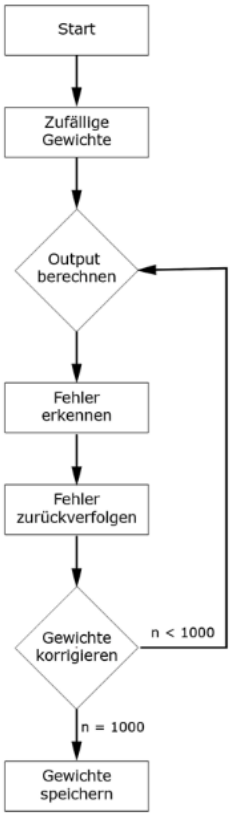



Mit diesen beiden Listenwerten werden die aktuell notwendigen Geschwindigkeiten eingestellt, indem sie mit der Maximalgeschwindigkeit (hier 450) multipliziert werden.


Geduld: das Programm braucht ca. 10s nach dem Hochladen bis zum Start.

Lösungsmöglichkeit



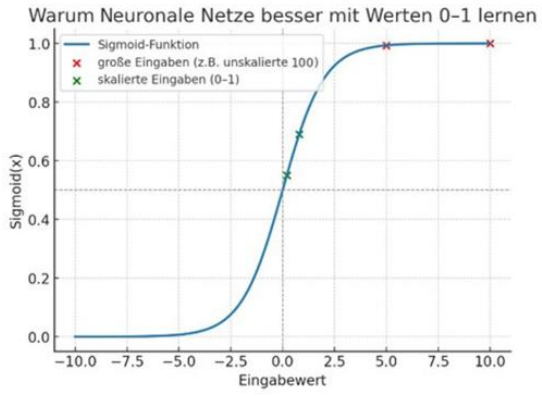




Es kann durchaus sein, dass das Programm schon richtig funktioniert hat.
In der Regel klappt das aber nicht.
Das neuronale Netz versucht, die vorgegebenen Zeilen der Trainingsdaten exakt zu erfüllen. Dabei kann es z. B. „übers Ziel hinausschießen“! Wir müssen es auf jeden Fall richtig trainieren – aber wie? Die Sensoren schicken Rohwerte ins Gehirn des Roboters. Jede Verbindung hat im Programm eine „Stellschraube“, die bestimmt, wie stark das ursprüngliche Signal bei einem bestimmten „Denkneuron“ ankommt. Beim Training dreht man in jeder Runde (Epoche) an diesen Stellschrauben, bis die Motoren im richtigen Moment genau das tun, was sie sollen. Das neuronale Netz schaut also, wie groß sein Fehler war, geht rückwärts durch alle Verbindungen und dreht an den Stellschrauben der Gewichte, damit es beim nächsten Mal besser liegt. Um das komplexe biologische neuronale Netz im Programm zu simulieren, hilft uns die Mathematik.

Jedes Eingangsneuron schickt nach dem Training also seinen eigenen gewichteten Wert an die Neuronen der verborgenen Schicht. Hier werden die Eingangssignale addiert und wiederum gewichtet. Diese „Denkneuronen“ senden Signale an die Ausgangsneuronen, die diese Signale ebenfalls addieren und gewichten.
**Summe = (Eingabe_1• Gewicht_1) + (Eingabe_2 • Gewicht_2) + Bias**
Der **Bias** ist im Prinzip ein kleiner Trick, damit ein Neuron flexibler reagieren kann. Bias verschiebt den Startpunkt, ab wann ein Neuron aktiv wird, er stellt quasi die Grundaktivität eines Neurons ein, auch wenn kein Sensorwert anliegt. Mit einer speziellen mathematischen „Aktivierungsfunktion“ werden die gewichteten Daten abschließend noch in eine dynamische, nicht-lineare Funktion umgerechnet. Das kommt der Biologie sehr nahe!
So entsteht eine Tabelle, die im Programm abgelegt und bei „execute NN with inputs“ zur Vorhersage der benötigten Motorwerte verwendet wird.
**ACHTUNG Fehlerquellen:**
Die Genauigkeit des Trainings hängt von mehreren Faktoren ab:

1. Anzahl der Trainingsdurchläufe (Epochen): je mehr, umso genauer, aber zeitintensiv
2. Einstellung der Lernrate (learning rate)
3. Anzahl der vorgegebenen Situationen in der Trainingstabelle

**Warum sollen die Eingaben für das neuronale Netz zwischen 0 und 1 liegen?**




Neuronale Netze lernen besser, wenn die Eingaben (z. B. Abstandswerte-Werte) zwischen 0 und 1 liegen:
**Große Zahlen** (z.B. 100) führen dazu, dass Aktivierungsfunktionen wie die Sigmoid-Funktion in Sättigungsbereiche geraten - dort reagiert das Netz kaum noch.
**Kleine**, normierte Werte (0–1) liegen im empfindlichen Bereich der Funktion. → Das Netz reagiert stärker und passt seine Gewichte besser an.

**Von der Biologie lernen!**

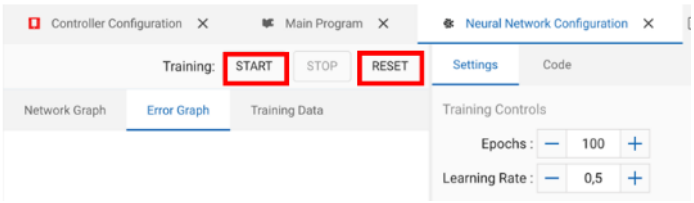
Im menschlichen Nervensystem ist die Lage ähnlich, nur eben in der „Bio-Version“, mit Nervenzellen und chemischen Reaktionen, statt Silizium und Software.

- **Sinneszellen (Sensoren):** z. B. Fotorezeptoren in der Netzhaut, Haarzellen im Ohr, Tastzellen in der Haut. Sie nehmen physikalische Reize auf (Licht, Schall, Druck) und wandeln sie in elektrische Signale um.
- **Eingangsebene (vergleichbar mit Eingangsneuronen):** Die Sinneszellen sind über Nervenbahnen mit anderen Nervenzellen verbunden, die diese Signale an die nächsten Schaltstellen weiterleiten. Dabei kann schon eine erste „Vorverarbeitung“ stattfinden (z. B. Kontrastverstärkung im Auge).
- **„Verborgene Schichten“ im Gehirn (Hidden Layer):** Mehrere Schichten von Neuronen verarbeiten, gewichten und kombinieren die Signale, bevor eine Entscheidung oder Reaktion ausgelöst wird.
- **Ausgangsebene (Motorneuronen):** Lösen dann Muskelbewegungen oder andere Reaktionen aus.

**Der große Unterschied:**

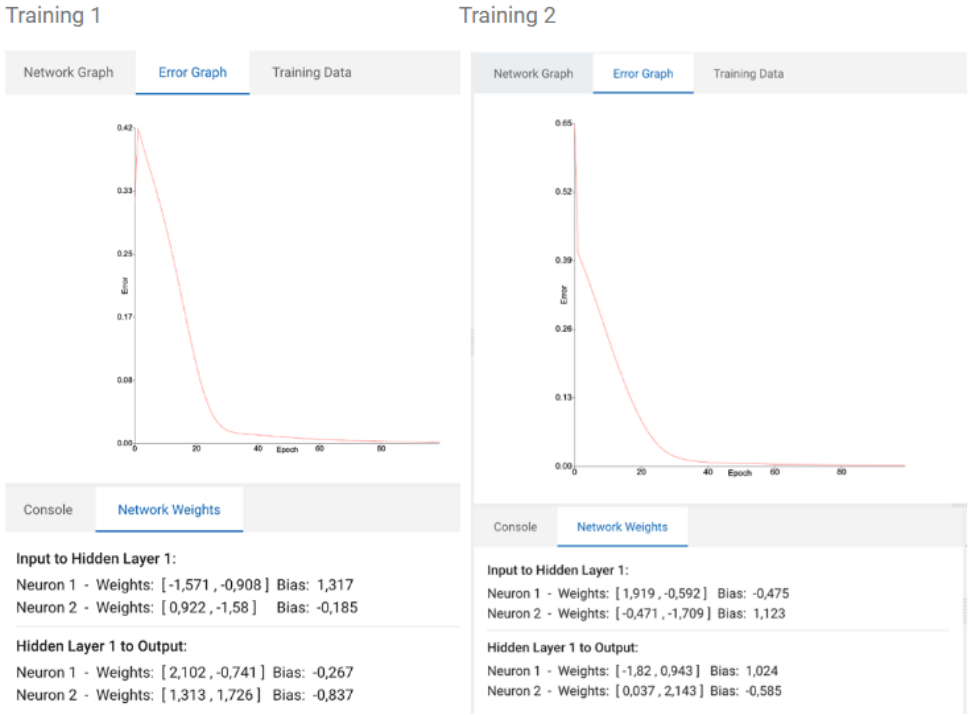
Im Gehirn wird das „Training“ nicht durch Rechenoperationen gesteuert, sondern durch komplexe biochemische Prozesse — z. B. **Synapsenverstärkung oder -abschwächung** (Hebb’sches Lernen) basierend auf Erfahrung.

**Jetzt wird gelernt**



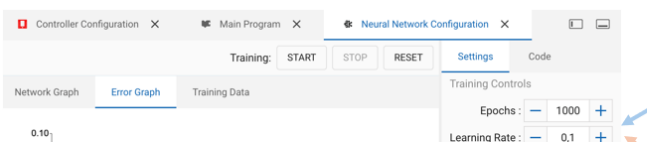

Probiere zunächst die vorgegebenen Grundeinstellungen aus:
Epochsen: 100
Learning Rate: 0.5
Klicke auf Start.

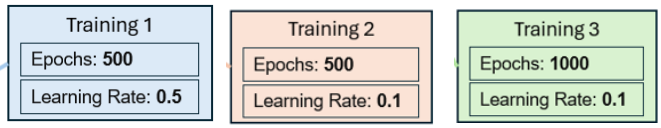
Funktioniert das Programm nun besser? Bei jedem Trainingsdurchlauf werden andere Gewichte und Bias berechnet. Klick auf **Reset** und erneut auf **Start**. Teste das Programm und beobachte, ob es im Ablauf Unterschiede gibt.
Mögliche Ergebnisse:




**Wird es noch besser? Zahl der Trainingsepochen vergrößern**

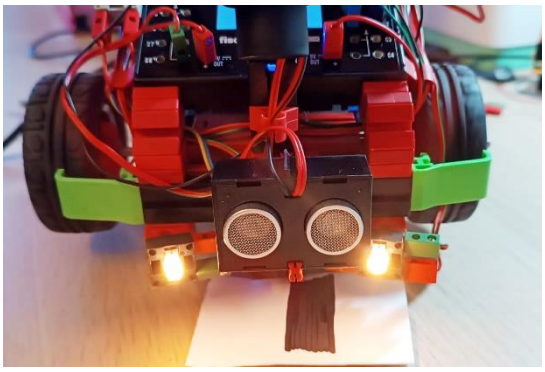
Führe nun weitere Versuche mit geänderten Einstellungen (Epochen, Lernrate) durch. Denke vor dem Start des Trainings daran, den Reset-Button zu betätigen. Nach jedem Training überträgst du das Programm in den TXT 4.0 Controller und machst eine Testfahrt mit dem FTF.




Welche Unterschiede bemerkst du?
Ist evtl. eine Version „übertrainiert“?

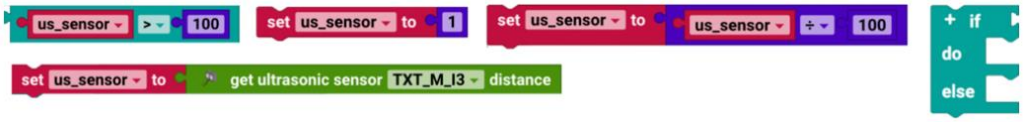
## Abstandssensor dem neuronalen Netz hinzufügen



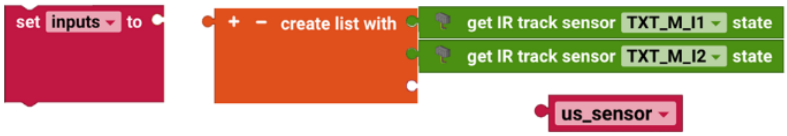
Beim Einsatz eines FTFs kann es vorkommen, dass sich Gegenstände oder langsamere FTFs im jeweiligen Block vor dem schnelleren FTF befinden. Damit es in diesem Fall zu keiner Kollision kommt, soll das FTF nun mit einer Bremsfunktion ausgestattet werden. Zur Simulation wird das FTF und ein Gegenstand (etwa in der Höhe des FTF) auf den Plan gesetzt. Ziel ist es, das Fahrzeug so zu programmieren, dass es das Hindernis erkennt und stoppt. Zur Simulation eines langsameren Fahrzeugs kannst du das Hindernis vor dem FTF bewegen. Das FTF soll abbremsen, bis es die gleiche Geschwindigkeit wie dein bewegtes Hindernis hat.

Der Abstandssensor ist bereits an den Eingang I3 angeschlossen und ist vorne montiert.
Zunächst muss du in deinem bisherigen Programm die Variable „us_sensor“ erstellen.

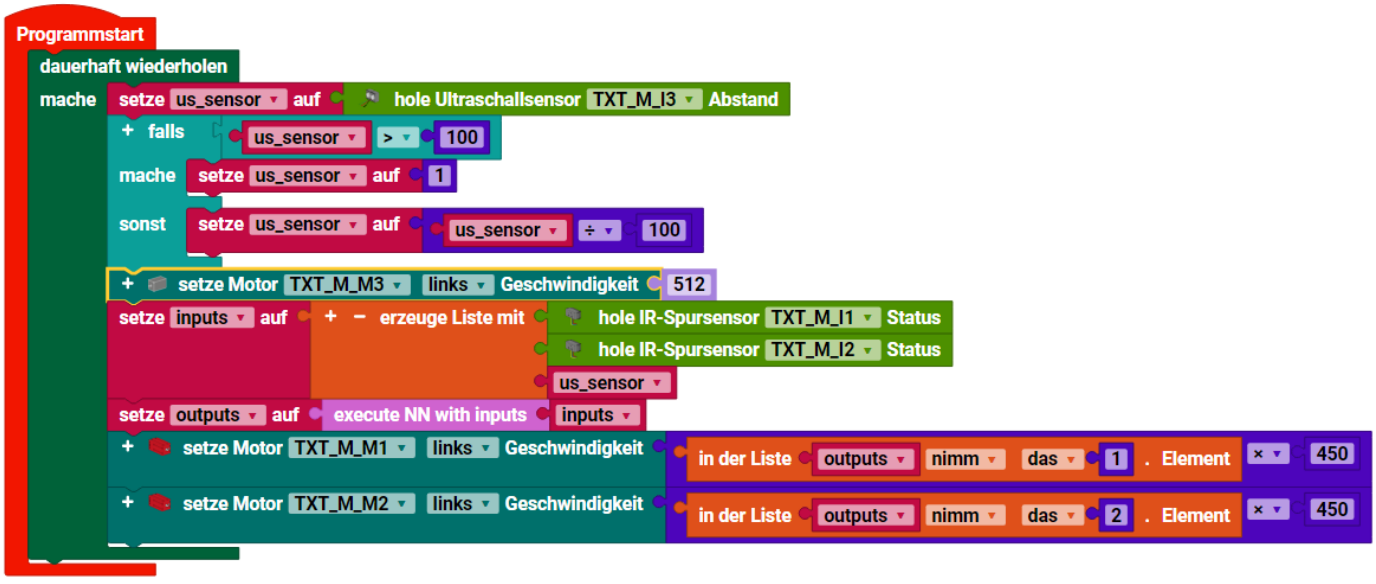

Zu Beginn der „repeat forever“-Schleife soll der Messwert des Abstandssensors in die Variable „us_sensor“ geschrieben werden. Die Messwerte des Abstandssensors sind cm-Angaben. Wir wollen nur den Wertebereich zwischen 0 und 100 cm betrachten. Jeder größere Wert wird grundsätzlich auf 100 cm gesetzt. Das Lernen des neuronalen Netzes funktioniert aber mit Werten zwischen 0 und 1 besser. Deshalb müssen die Werte in der Variablen „us_sensor“ durch 100 geteilt werden, sodass sie nun alle zwischen 0 und 1 liegen.



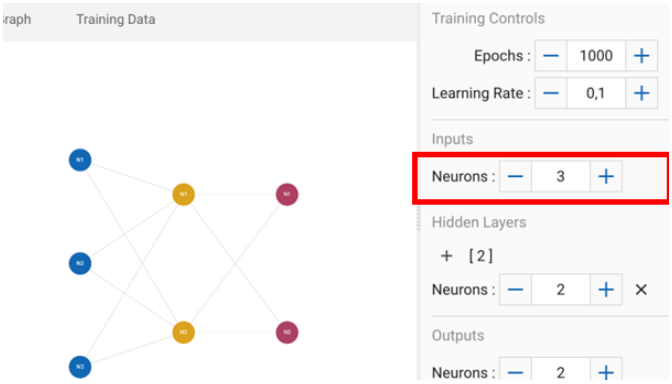

Nun musst du die Listenvariable „inputs“ noch um den Eintrag des „us_sensor“-Wertes erweitern.




Schon ist das erweiterte Programm fertig. Nun musst du nur noch das neuronale Netz entsprechend erweitern und dann erneut trainieren.

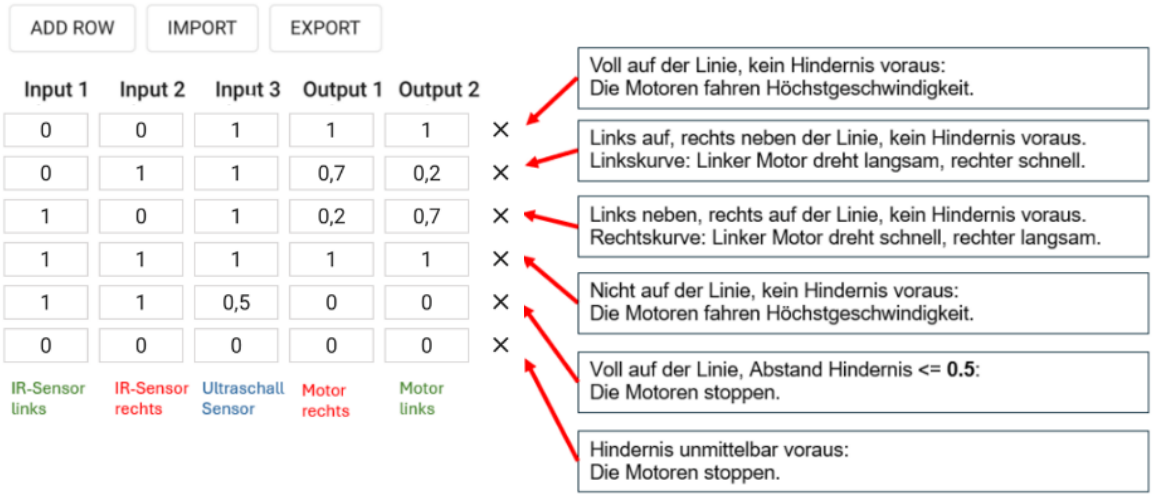


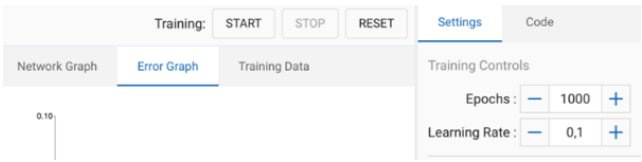
Zunächst musst du ein drittes Eingangsneuron hinzufügen.




Dadurch wir auch die Tabelle „Training Data“ um eine Spalte erweitert.


Beachte, dass die hohe Motorgeschwindigkeit hier auf 0.7 (statt wie bisher 0.9) gesetzt werden sollte.




Nun musst du das neuronale Netz trainieren. Für dieses etwas komplexere Problem sind für die Training Controls die folgenden Einstellungen sinnvoll: 1000 Epochen und Learning Rate 0.1.

Führe mehrere Trainings mit den gleichen Parametern durch. Teste nach jedem Training das Programm und beobachte die Reaktionen des FTFs.
War das Testen der Programme erfolgreich?
Folgende Punkte helfen dir bei der Fehlersuche. Überprüfe die Vorschläge ggf. mit deiner Anlage.

- Stimmt der Programmablauf?
- Stimmen die die Variablennamen?
- Sind alle Funktionen und Variablen korrekt definiert?
- Ist die Tabelle „Training Data“ korrekt?
- Hast du mehrere Trainingsdurchläufe durchgeführt und getestet?

Wenn das FTF wie gewünscht fährt, klicke auf weiter.

## Fazit

**Vor und Nachteile Neuronaler Netze**

Vielleicht denkst du nach diesen ersten Übungen, dass neuronale Netze auf den ersten Blick keinen großen Vorteil bringen. Schließlich hast du ja die Beispiele selbst programmiert und die Programme für diese Modelle wären auch manuell leicht umsetzbar. Aber der entscheidende Punkt, den du dir merken solltest, ist folgender:
Du hast dem neuronalen Netz nur wenige Beispiele gegeben, um zu lernen.
Doch wenn das Netz ausgeführt wird, erhält es unzählige verschiedene Eingabewerte – nicht nur 0,2, 0,4 oder 0,6, sondern auch Werte wie 0,3, 0,444444..., 0,1 usw.
Dennoch entscheidet das Netz richtig, welche LEDs eingeschaltet werden sollen.

**Die eigentliche Stärke der neuronalen Netze**

Diese Fähigkeit, auch Zwischenwerte richtig zu interpretieren, ist das, was neuronale Netze so mächtig macht.
Dasselbe Prinzip findet man in komplexeren Systemen, zum Beispiel bei:

- Sprachmodellen wie ChatGPT,
- Bilderkennungssystemen,
- oder bei der autonomen Fahrzeugsteuerung.

Neuronale Netze lernen aus Beispielen – und je besser und sauberer diese Beispiele sind, desto präziser und verlässlicher arbeiten sie.

**Bedeutung der Trainingsdaten**

Es ist äußerst wichtig, die Trainingsdaten sorgfältig zu kontrollieren. Wenn man ein Sprachmodell mit allen Inhalten des Internets trainieren würde, ohne Filterung, könnte das Modell Unsinn oder gefährliche Aussagen wiedergeben, die es aus Foren oder pseudowissenschaftlichen Quellen gelernt hat.
Selbst wenn man filtert, können sich verzerrte Informationen (Bias) einschleichen.

**Verantwortung und kritisches Denken**

So intelligent und beeindruckend moderne KI-Systeme auch erscheinen, sie sind immer nur so gut wie die Daten, mit denen sie trainiert wurden. Deshalb müssen die Ergebnisse solcher Systeme kritisch zu hinterfragt werden.
Die Beispiele sind alles. Wenn die Trainingsbeispiele nicht korrekt oder ausgewogen sind, kann auch das Ergebnis fehlerhaft oder gefährlich sein.

**Vorteile**:

- Kann Zwischenwerte und Muster erkennen
- Lässt sich auf viele komplexe Systeme anwenden
- Lernt aus Beispielen, ohne exakte Regeln zu benötigen
- Kann sich an neue Daten anpassen

**Nachteile**:

- Abhängig von der Qualität der Trainingsdaten
- Risiko von Vorurteilen (Bias) und Fehlinterpretationen
- Fehlendes Verständnis für den Inhalt – lernt nur Korrelationen
- Training erfordert Kontrolle und Verantwortung

**Wie sicher sind Entscheidungen des Neuronalen Netzes**

In den Übungen zur Klassifikation und Mehrfach-Klassifikation hast du gesehen, dass das neuronale Netz einen Wert zwischen 0 und 1 für jede Neurone in der Ausgabeschicht liefert.
Dieser Wert zeigt an, wie sicher sich das Netz ist, dass eine bestimmte LED leuchten soll.

**Verstehen der Sicherheitswerte**

- Ein Wert von 1,0 bedeutet:
  → Das Netz ist 100 % sicher, dass dieser Fall zutrifft.
- Ein Wert von 0,5 bedeutet:
  → Das Netz ist unsicher, ob dieser Fall korrekt ist.
- Je näher der Wert an 1 liegt, desto zuversichtlicher ist das neuronale Netz in seiner Entscheidung.

Wenn die Werte mit 100 multiplizieren werden, können sie als Prozentwerte der Sicherheit dargestellt werden.