fischertechnik

AI Omniwheels Soccer AI

Hightech

Dieser Roboter spielt Fussball!

Hinweis:
*Wir empfehlen, zuerst mit dem Modell **AI Follower** zu arbeiten. Dort wird die Programmierung und die Funktionsweise des neuronalen Netzes genauer erklärt. Bei diesem Modell gehen wir **nicht** tief auf die Programmierung ein – du kannst das Beispiel direkt herunterladen.*
*Wir erklären dir aber, **warum** es so programmiert ist und wie die **Trainingsdaten** funktionieren.*

Dafür verwenden wir:

- 🧠 ein neuronales Netzwerk
- 🤖 ein bisschen klassische Programmierung
- 📷 die Kamera des Roboters

Es ist, als würde der Roboter denken:
„Ich sehe den Ball … wohin soll ich mich bewegen?“

Klassische Programmierung + neuronales Netzwerk

Dieses Beispiel ist besonders, weil es **zwei Arten des Programmierens kombiniert**:

Klassische Programmierung

Die benutzen wir, um:

- den Ball mit der Kamera zu erkennen
- seine Position im Bild zu bestimmen
- die Daten so vorzubereiten, dass das „Gehirn“ sie versteht

Neuronales Netzwerk

Das benutzen wir, um:

- zu entscheiden, wie sich die Motoren bewegen sollen
- aus Beispielen zu lernen
- alle Zwischenbewegungen zu berechnen

👉 Das Beste aus beiden Welten! ✨

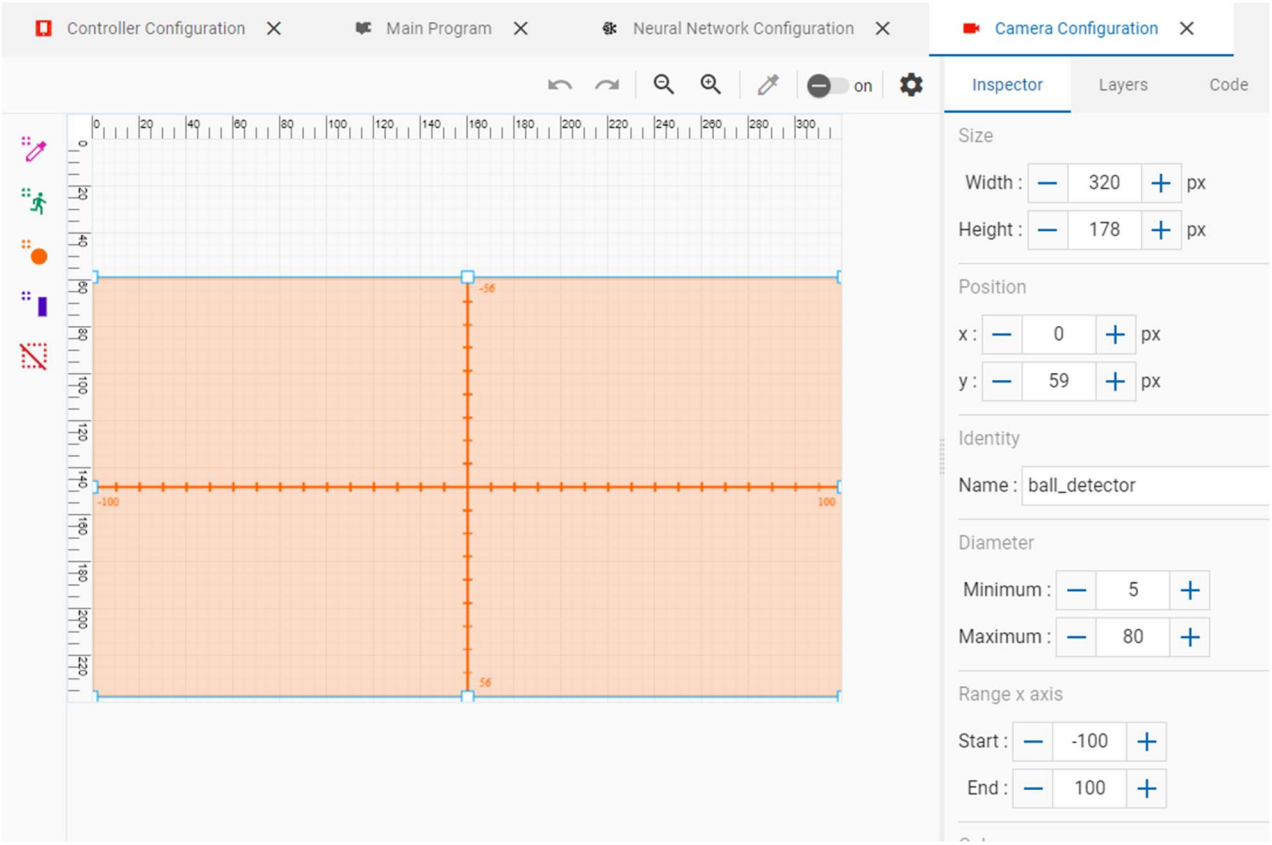
Wenn du schon das Omniwheels-Modell programmiert hast, weißt du:
Die Programmierung ist dort ziemlich komplex.
Hier sind die **Beispiele komplex**, aber die **Programmierung selbst ist sehr einfach**.

Was sieht der Roboter?

Die Kamera erkennt den Ball und liefert:

- X → links oder rechts
- Y → oben oder unten

Schau dir an, wie wir die Kamera konfiguriert haben

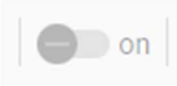


Diese Werte:

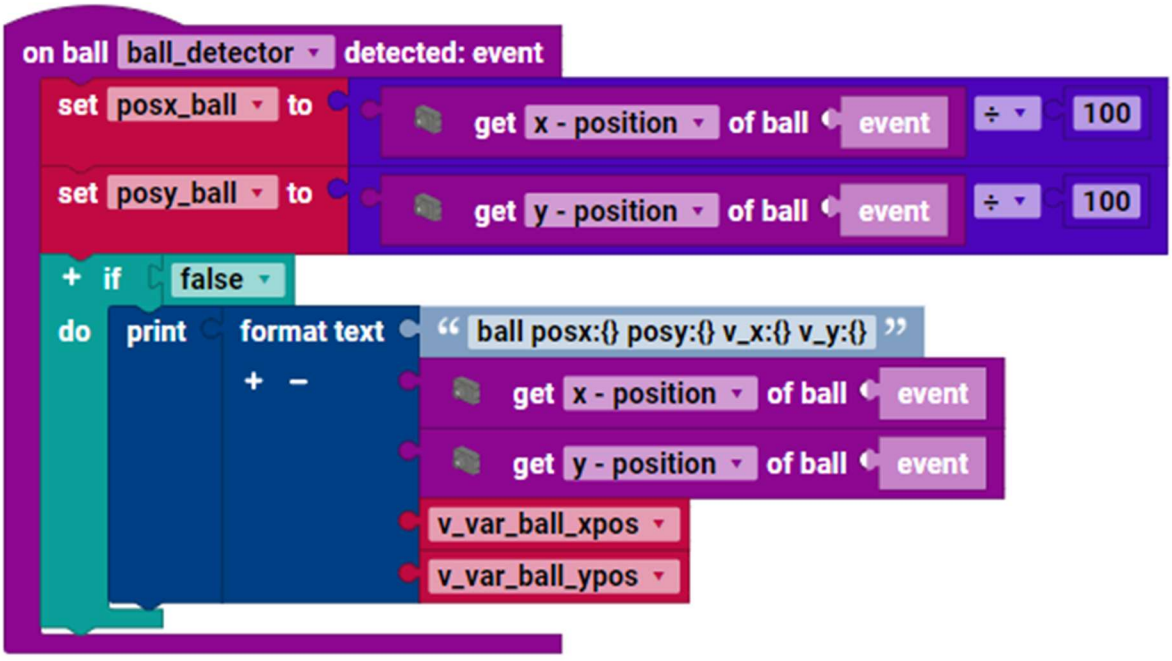
- werden auf einen Bereich von **-1 bis 1** umgerechnet (wir teilen durch 100)
- werden als **Inputs** an das neuronale Netzwerk übergeben

Merke:
Neuronale Netze arbeiten besser mit **kleinen, geordneten Werten**.
Darum verwenden wir immer den Bereich **-1 … 1**.

Aktiviere die **Live-Ansicht der Kamera** über dieses Symbol



Wenn alles läuft, nutze das **„Pick Color"-Werkzeug**, um die Farbe des Balls festzulegen.
Achte darauf, dass ein **blauer Kreis** über dem Ball erscheint – dann wird er richtig erkannt



Hinweis:
Der untere Bereich dient nur dazu zu prüfen, ob die Kamera den Ball korrekt erkennt.

Die Kamerawerte kommen über ein **Event**.
Darum speichern wir sie zuerst in zwei Variablen und fassen sie dann im

Hauptprogramm in einer Liste zusammen, die wir an das neuronale Netzwerk übergeben.

Und was macht der Roboter damit?

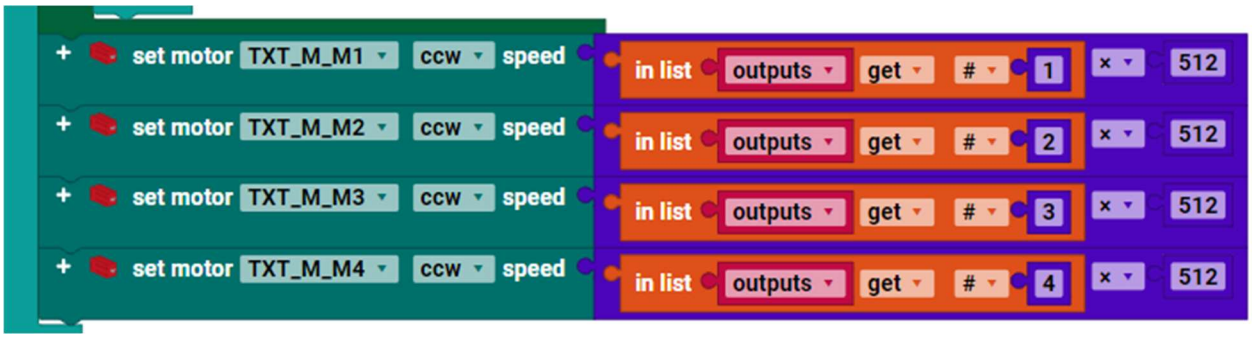
Das „Gehirn“ gibt zurück:

- einen Wert pro Motor

Diese Werte sagen:

- wie schnell sich jeder Motor drehen soll
- in welche Richtung

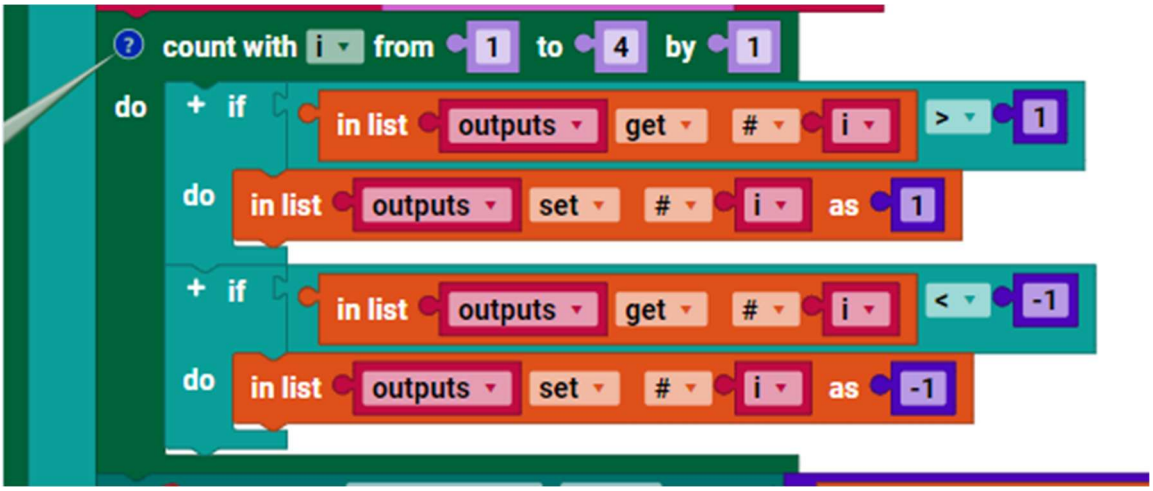
👉 Danach multiplizieren wir sie (z. B. mit **512**)
👉 und machen daraus echte Bewegung 🚗⚙️



Beim Line Follower haben wir mit **400** multipliziert, weil die Netzwerkausgabe größer als 1 sein kann.
Hier prüfen wir die Werte vorher:

- größer als **1** → wird **1**
- kleiner als **-1** → wird **-1**

Wichtig:
Das Netzwerk gibt eine **Liste** zurück – wir prüfen **alle Werte in dieser Liste**.



Aufbau des neuronalen Netzwerks

Wie schon beim Line Follower gilt:
Die **Eingabe- und Ausgabeschichten** hängen von den **Sensoren und Motoren** des Roboters ab.

Unser Roboter hat:

- eine Kamera
- eine Lichtschranke

Die Lichtschranke benutzen wir **nicht** im neuronalen Netzwerk.
Später verwenden wir sie, um den Ball zu schießen ⚽
Jetzt bleiben wir erstmal beim Netzwerk.

Eingabeschicht

Brauchen wir nur **eine** Eingabeneurone?
👉 Nein!
Die Kamera liefert **X und Y**, also **zwei Werte → zwei Neuronen**

Wenn wir z. B. RGB-Farben (Rot, Grün, Blau) verwenden würden, bräuchten wir **drei Neuronen**.

Ein Beispiel:
Ein Bild mit **16 Megapixeln** hätte
👉 **16 Millionen Pixel × 3 Farben = 48 Millionen Eingabeneuronen** 😳

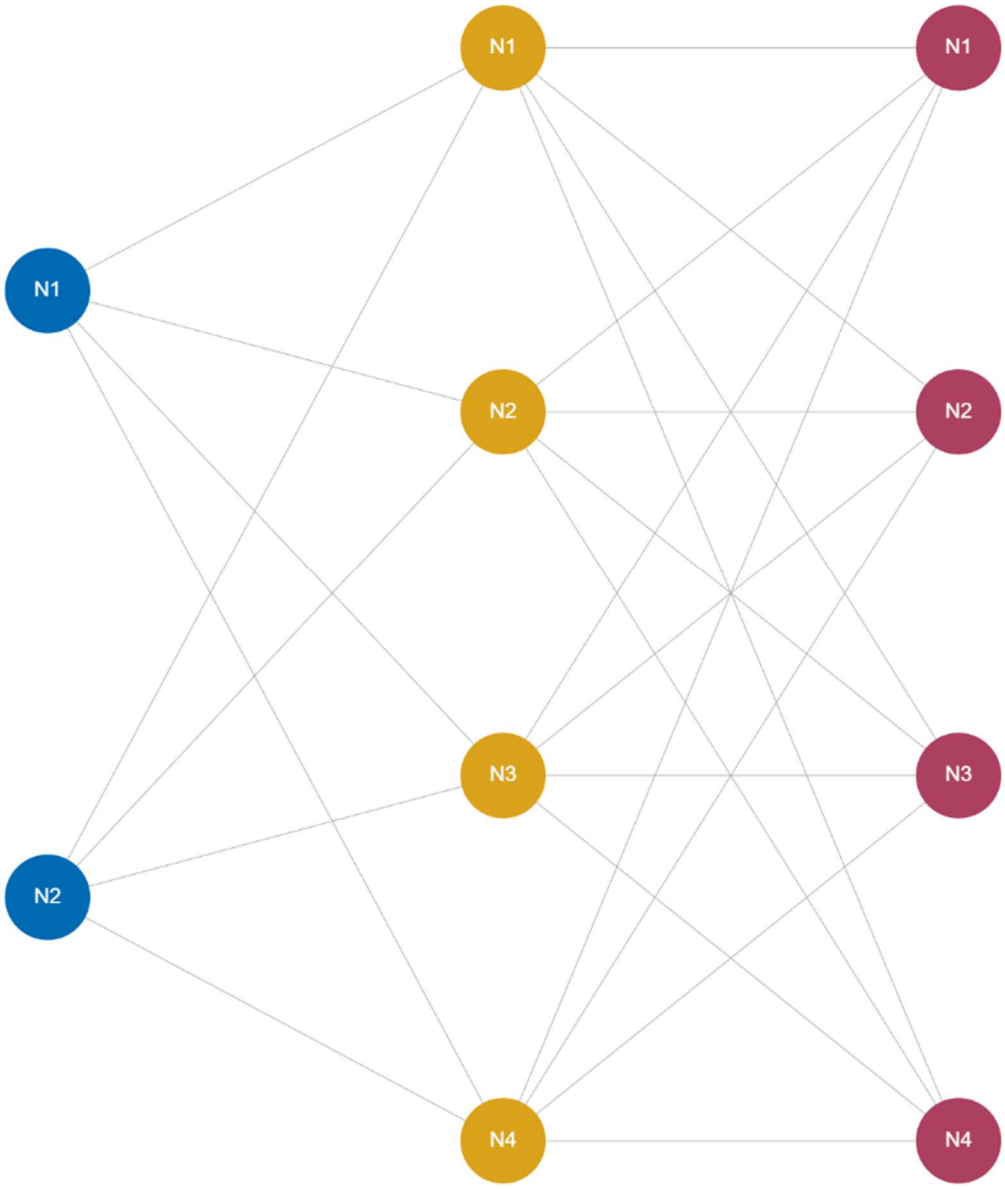
Aber keine Sorge – unser Netzwerk ist viel einfacher 😉

Ausgabeschicht

Wir steuern **4 Motoren** → **4 Neuronen**

Versteckte Schicht

Wir haben uns für **eine Schicht mit 4 Neuronen** entschieden.
Das hat bei uns sehr gut funktioniert 👍



Wie lernt der Roboter?

Das Training läuft so:

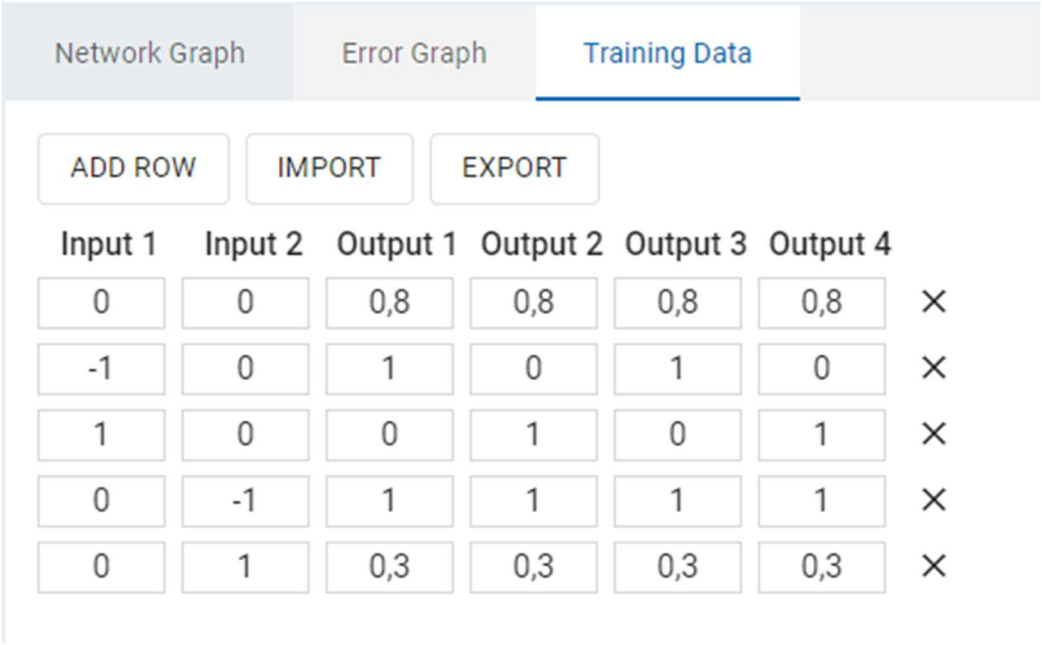
- Wir geben Beispiele wie:
  - Ball links → Motoren drehen so
  - Ball in der Mitte → geradeaus
  - Ball rechts → entgegengesetzt
- Außerdem extreme und zentrale Positionen

Beispiele im Detail:

- Ball **zentriert** (0, 0)
  → Roboter fährt schnell vorwärts
  → alle Motoren: **0,8**
- Ball **links** (-1, 0)
  → Roboter dreht
- Ball **rechts** (1, 0)
  → genau umgekehrt
- Ball **ganz oben** (0, -1)
  → volle Geschwindigkeit
  → alle Motoren: **1**
- Ball **ganz unten** (0, 1)
  → langsam vorwärts

Mehr Beispiele brauchen wir nicht 😊

Alle **Zwischenfälle** berechnet das neuronale Netzwerk **von selbst** – und das ist ein riesiger Vorteil ✨



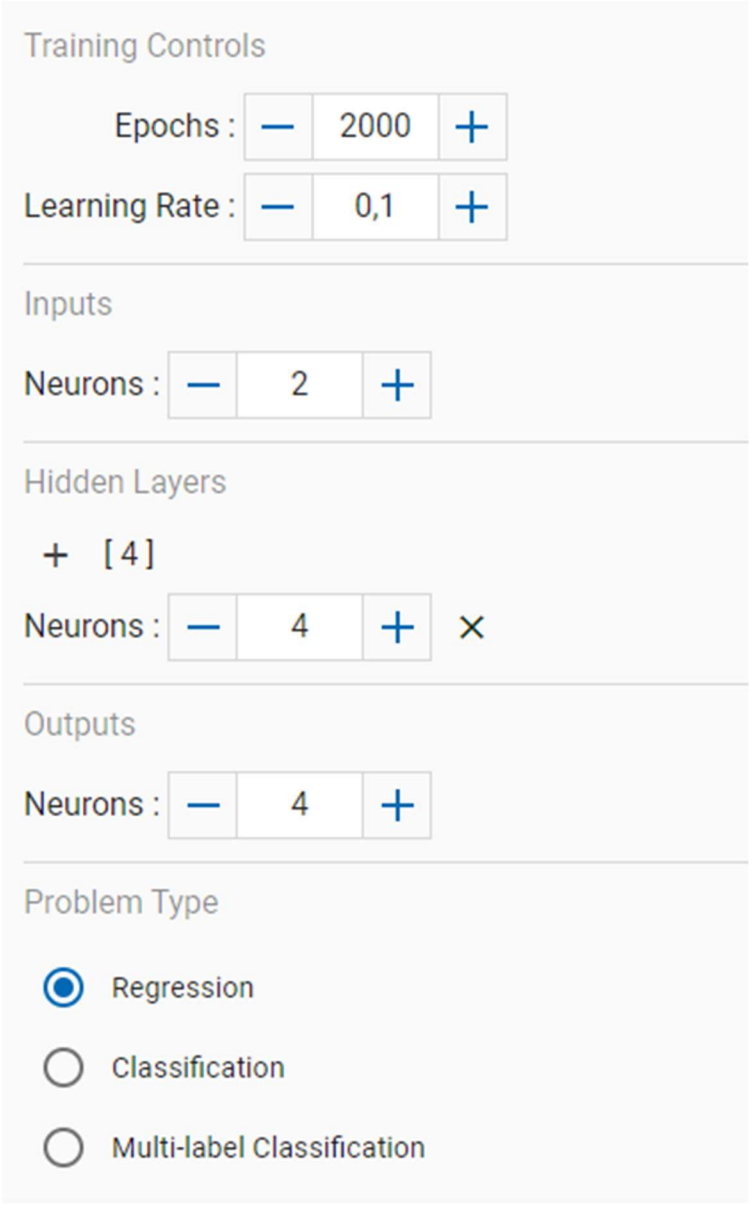
Training starten

Wir haben mit diesen Werten trainiert:

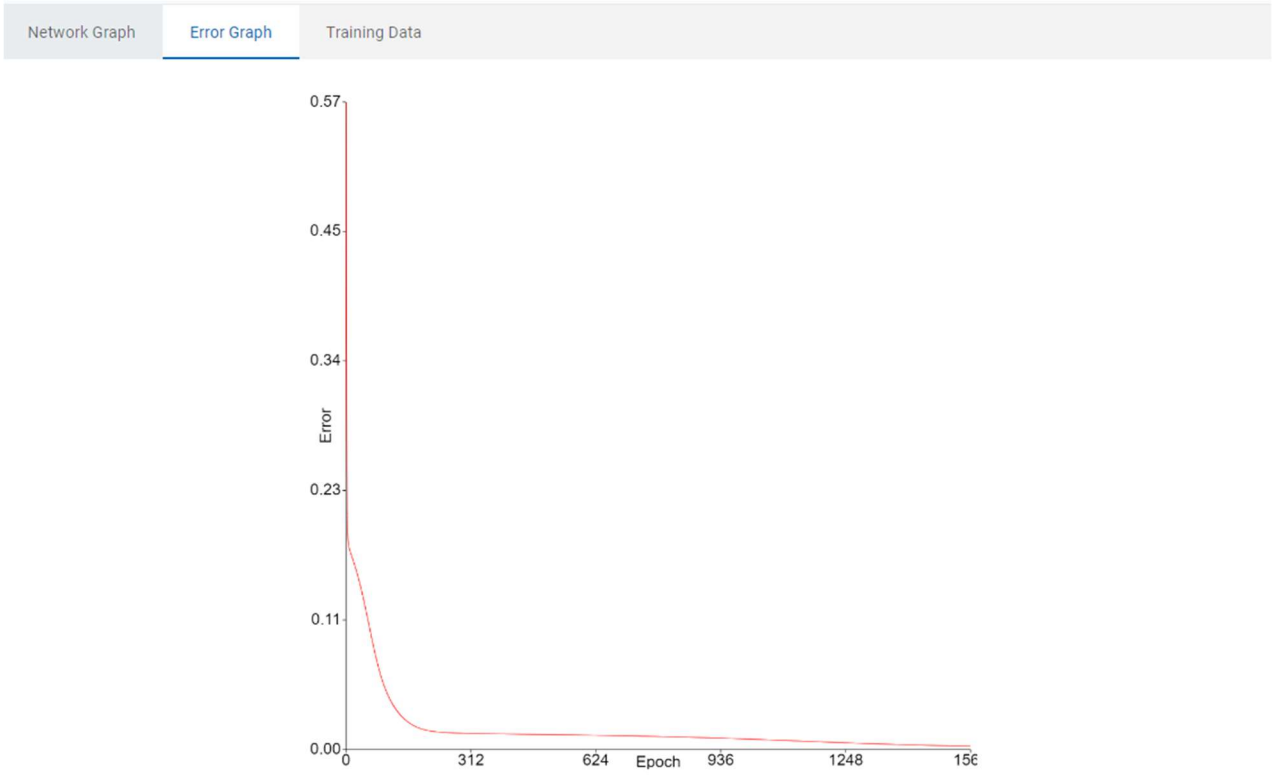
- Epochs: ca. 2000
- Learning Rate: 0,1

Merke:

- große Learning Rate → lernt schnell, aber ungenau
- kleine Learning Rate → lernt langsam
- 0,1 funktioniert in den meisten Fällen sehr gut



Hier siehst du die Trainingsgrafik 📈



Welche Art von Problem ist das?

Das ist ein **Regressionsproblem**.

Das bedeutet:

- keine „Ja / Nein“-Antwort
- kontinuierliche Werte
- kleine Änderung der Ballposition
  → kleine Änderung der Bewegung

Wie beim Lenken:
👉 ein bisschen mehr oder weniger drehen

Den Ball schießen

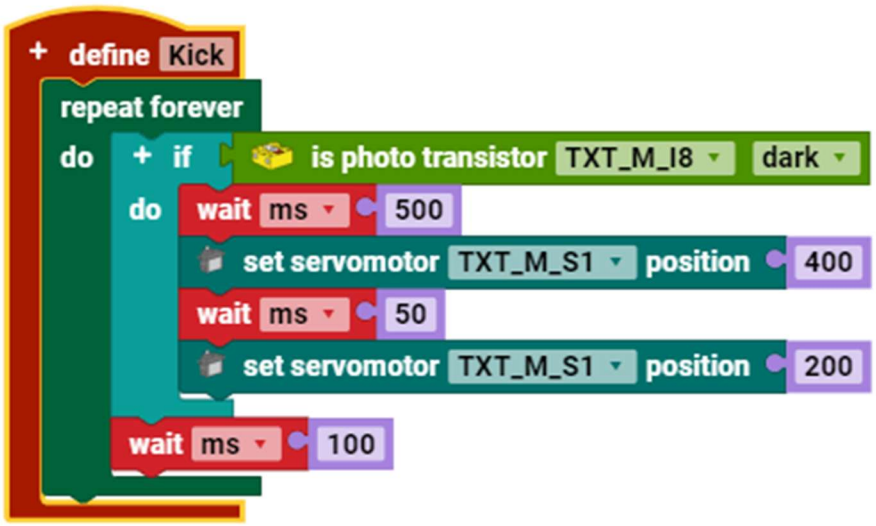
Zum Schießen erstellen wir eine eigene Funktion.

Sie ist ganz einfach:

- Ball erkannt → 0,5 Sekunden warten
- dann schießen!

Die kurze Wartezeit sorgt dafür,
dass der Ball richtig positioniert ist.

Diese Funktion läuft in einer **Endlosschleife**.
Damit sie das Hauptprogramm nicht blockiert,
führen wir sie in einem **Thread** aus.



👉 Ein Thread läuft **parallel** zum Hauptprogramm.
So können:

- das Hauptprogramm
- die Schuss-Funktion
- die Kamera

gleichzeitig arbeiten

Der Thread wird mit diesem Block gestartet.

Ausführen

Jetzt weißt du:

- wie der Roboter funktioniert
- wie er trainiert wird

Starte ihn und beobachte,
wie er den Ball verfolgt und schießt

Zum Schluss

Mit diesen **zwei Modellen** hast du die Grundlagen neuronaler Netzwerke gelernt
und gesehen, wie flexibel man sie bei Robotern einsetzen kann.

Probier doch:

- das Schießen direkt ins Netzwerk einzubauen
- andere Trainingsdaten für andere Bewegungen
- neue Dreh-Strategien

Andere Problemtypen

Klassifikation

Hier wählt das Netzwerk **eine Option aus mehreren**.

Beispiel:

- Farben erkennen (RGB)
- 3 Eingabeneuronen
- 4 Ausgabeneuronen (je eine pro Farbe)

Beim Training:

- alle Ausgänge = 0
- die richtige Farbe = 1

Beim Ausführen:

- die Neurone mit dem **höchsten Wert** (meist > 0,5) gewinnt

Multiklassifikation

Mehrere Neuronen können gleichzeitig aktiv sein.
Das ist eher selten, aber gut zu wissen

Jetzt bist du dran!

Probiere eigene Ideen aus:

- andere Roboter
- andere Sensoren
- Linien erkennen mit der Kamera
- neue Trainingsdaten

Experimentiere, trainiere neu
und finde heraus, was am besten funktioniert.

Und das Wichtigste:
👉 Hab Spaß dabei!