Deep Learning

Grundlagen Grundbegriffe Grundübungen

Alfatraining Kurs 2024-08 Dozent: Karsten Flügge

Themen des Tages

Tag 16

Agentensysteme, Spiel Agenten
Training durch Belohnungen
Deep Reinforcement Learning (RL)
Policy Gradients,
Deep-Q-Learning

Steuerung dynamischer Systeme

Deep Reinforcement Learning

Anwendung der exakten Techniken welche wir kennen in neuem Setting!

Agentensysteme, Spiel-Agenten

Ein **Agent** ist ein System, das in einer Umgebung **autonom Entscheidungen** trifft und durch seine **Aktionen Ziele** verfolgt. In der Regel reagiert ein Agent auf **Umweltreize** und hat die Fähigkeit, **selbstständig** zu **lernen** und sich an neue Situationen anzupassen.

Agentensysteme

Anwendungen:

- **Spiele**
- **Robotik** &
- **Steuerung**stechnik
- **Simulation** von Menschenmengen
- **Wirtschaft**, z.B. für autonome Handelssysteme.

Spiel-Agenten

Erfolgsgeschichte

Schach: Der Schach-Computer **Deep Blue**, der Garry Kasparov besiegt hat, ist ein Beispiel für einen deliberativen Spiel-Agenten, der durch umfassende Berechnungen Züge analysiert.
Jeopardy: When IBM was great
DeepMind: 2013 verschiedenste **Atari** Spiele mit dem **selben Netz** geschlagen (Transfer Learning von Meta-Strategien!)
Go: AlphaGo, basierend auf Deep Learning und Reinforcement Learning, besiegte 2016 den weltbesten Go-Spieler.

Moderne

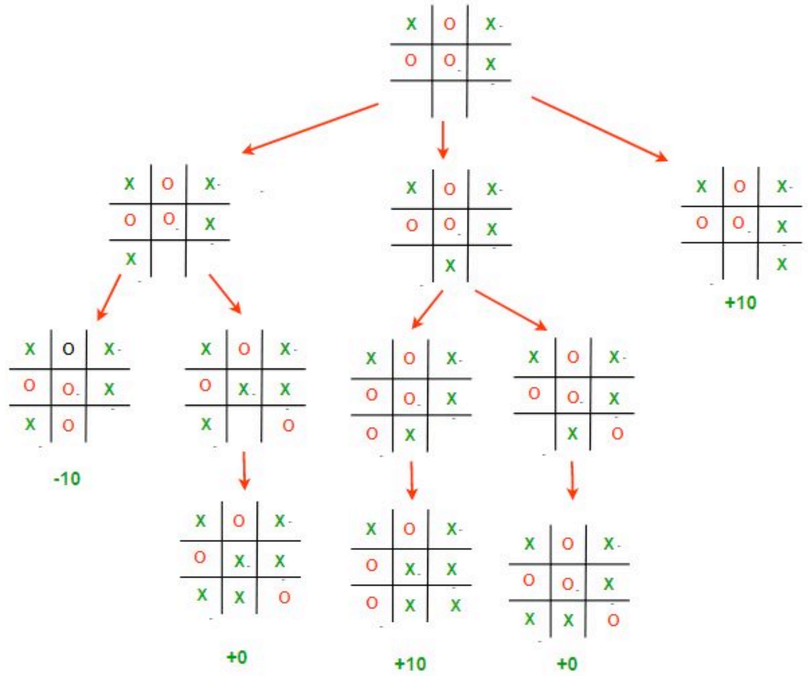
Videospiele: NPCs (Non-Player **Characters**) in Computerspielen sind oft KI-gesteuerte Agenten, die auf bestimmte Umgebungen oder Spieleraktionen reagieren.

Training durch Belohnungen

Intuitiver Ansatz:
Merke bei jedem Spiel Verlauf alle **Positionen**
Zähle wie oft eine Position zu **Siegen** führte
Wähle neue Positionen teils zufällig nach Historie

Entscheidungsbaum -
Decision tree

Probleme:
- Keine guten Züge für **unbekannte Situationen**
- **zu viele** mögliche **Felder * Züge**



Reinforcement Learning

Reinforcement Learning (Verstärkungslernen) ist eine Methode des maschinellen Lernens, bei der ein Agent in einer **Umgebung** agiert, um durch Interaktionen zu lernen. Das Ziel ist es, eine **optimale Strategie (Policy)** zu finden, die zu maximalen kumulierten Belohnungen führt. Diese Art des Lernens unterscheidet sich von *Supervised* Learning (überwachtem Lernen), da der Agent nicht explizit mit den richtigen Aktionen trainiert wird, sondern durch Versuch und Irrtum lernt, **basierend auf den Rückmeldungen (Belohnungen) der Umgebung.**

- 💡 **Belohnung nicht nur bei Sieg sondern auch für neue Situationen**
- 💡 **Belohnung für verschiedenste Fortschritts-Metriken**

Reinforcement Learning

Kernkonzepte:

- **Agent:** Die Entität, die lernt und handelt.
- **Umgebung (Environment):** Der Raum, in dem der Agent agiert.
- **Zustand (State):** Repräsentation der aktuellen Situation der Umgebung.
- **Aktion (Action):** Eine Entscheidung des Agents, die den Zustand verändert.
- **Policy (Strategie):** Regel oder Funktion, die die Aktionen des Agents bestimmt.
- **Belohnung (Reward):** Rückmeldung der Umgebung nach einer Aktion.
- **Wertfunktion (Value Function):** Erwarteter kumulierter Belohnungswert eines Zustands.

Q-Value: Erwartete Belohnung / Expected reward

💡 Letzteres nennt sich auch **Q**-Wert für **qu**ality **ku**muliert ;)

Reinforcement Learning

Idee Implementierung Pseudo Code

```
while not is_game_over():
  state = environment.get_state()
  action = agent.choose_action(state) # via policy
  reward, next_state = environment.step(action)
  agent.update_policy(state, action, reward, next_state)
```

hat erst mal nichts mit Deep Learning zu tun, aber …

Reinforcement Learning

update_policy(state, action, **reward**, next_state)

1.) via **Heuristik:**

sammle alle durchgeführten Aktionen und die Zustände vorher und nachher

2.) policy update:
jene Aktionen welche zu einem positiven reward führten werden in Zukunft häufiger ausgewählt

Primitive algorithmische Policy:
Wähle diejenige **Aktion** welche in der Vergangenheit den **höchsten Reward** erzeugt hat.
Nachteile:
- Vorhersehbarkeit
- missed opportunities: es könnte viel bessere Aktionen geben
- Exploration: schon alle Aktionen ausprobiert??

Bessere algorithmische Policy:
Wähle **zufällig** aus den **n besten eine Aktion** welche in der Vergangenheit den **höchsten Reward** erzeugt hat.

Noch Bessere algorithmische Policy:
Wie oben aber führe ab und zu komplett neue Aktionen aus.

Noch noch bessere algorithmische Policy:
Schätze mögliche Belohnung für alle Aktionen und wähle davon die wahrscheinlich beste

hat erst mal nichts mit Deep Learning zu tun, aber …

Aufgabe:

in tic_tac_toe_reinforcement.py :
ändere policy in chose_action:

a) Bessere algorithmische Policy:

Wähle **zufällig** aus den **n besten eine Aktion** welche in der Vergangenheit den **höchsten Reward** erzeugt hat.

und/oder:

b) Noch Bessere algorithmische Policy:

Wie oben aber führe ab und zu **zufällig** komplett **neue Aktion** aus.

c) 4*4 Tic-Tac-Toe

Reinforcement Learning

Beispiele für konkrete **Policies**:

Temperatur Steuerung:

- Wenn die Temperatur im Raum um 5 Grad unter dem Zielwert liegt, heize mit mittlerer Leistung
- **Wenn** Temperatur ≥ Solltemperatur: **Heizung ausschalten**.

Roboter-Navigation:

- **Zustand:** Position und Orientierung des Roboters.
- **Policy:** Wenn das Hindernis innerhalb von 1 Meter ist, drehe um 90 Grad nach rechts. Wenn kein Hindernis, bewege dich geradeaus.

2. **Finanzportfolio-Management:**
   - **Zustand:** Marktdaten (z.B. Aktienpreise, Volatilität).
   - **Policy:** Wenn Aktienpreis steigt um 5%, kaufe mehr Anteile. Wenn der Preis um 10% fällt, verkaufe.
3. **Spielsteuerung (z.B. Schach):**
   - **Zustand:** Aktuelle Brettposition.
   - **Policy:** Wenn König bedroht, ziehe eine Figur zur Verteidigung. Wenn der Gegner eine Schwachstelle hat, setze eine aggressive Figur ein.
4. **Versicherung**
   - Zustand: Kunden mit Verträgen
   - Policy: Wenn Kunde Unfall erzeugt erhöhe Prämie
   - Belohnungsfunktion: Maximiere Gewinne der Versicherung
5. Eigen Ernährung:
   - Zustand: Gesundheit
   - Policy: Esse süsses
6. Sicherheit
   - Zustand: Friede?
   - Policy: Tit for tat

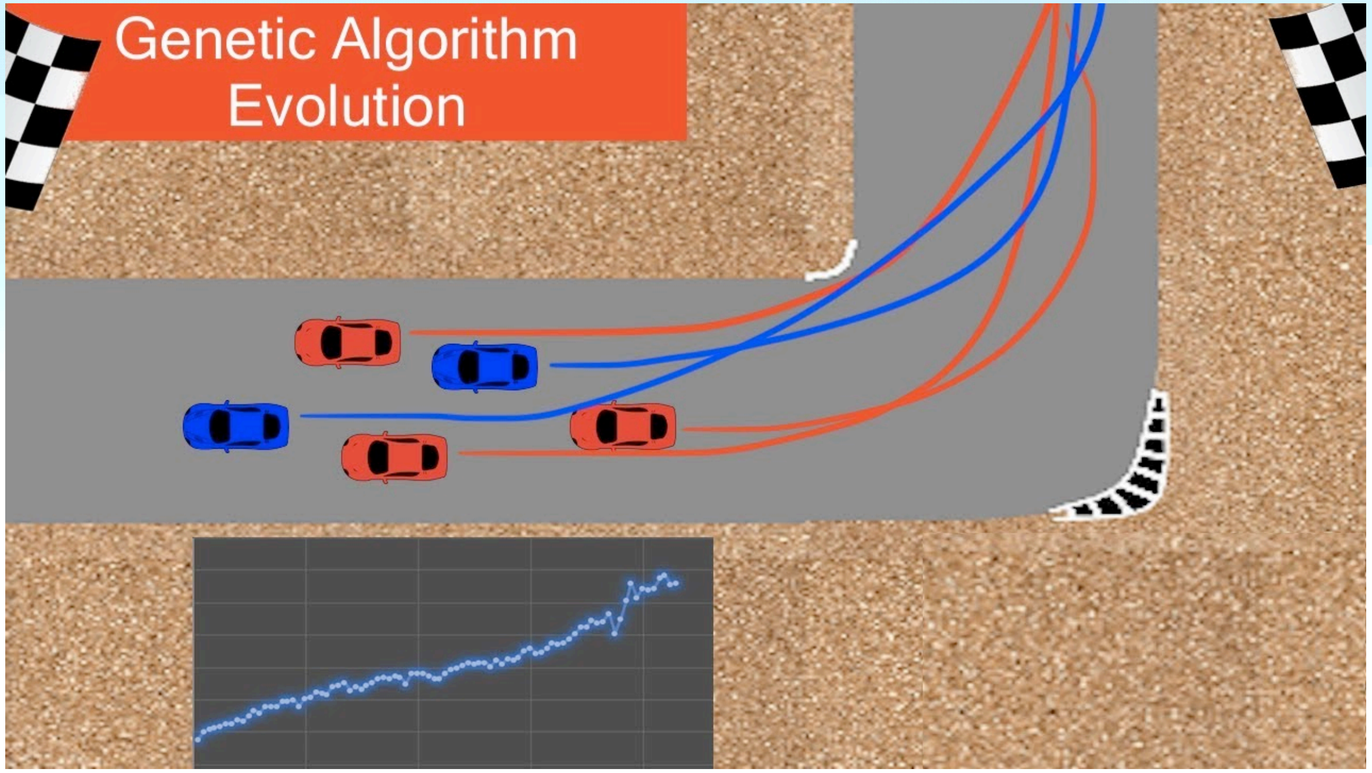
hat erst mal nichts mit Deep Learning zu tun, aber …

Reinforcement Learning

"***genetische Algorithmen***." zufällig eine erste Generation von 100 Policies erzeugen, ausprobieren und die **schlechte**sten 80 **Policies eliminieren**.

Die 'überlebenden' Policies können dann automatisch leicht verändert werden und das Spiel beginnt von neuem.

So könnten etwa selbst fahrende Autos im spiel an verschiedenen Positionen zur Kurve einschwenken...



`hat erst mal nichts mit Deep Learning zu tun, aber …`

Deep Reinforcement Learning

Deep Reinforcement Learning erweitert klassische Reinforcement Learning (RL) Ansätze durch die Verwendung von tiefen neuronalen Netzen, um komplexe Zustandsräume besser zu verarbeiten und zu generalisieren. Bekannte Algorithmen sind **Deep Q-Learning (DQN)**, **Policy-Gradient-Methoden**, und **Actor-Critic-Modelle**.

💡 Der **Zustandsraum** wird bei Spielen schnell quasi **unendlich**, so dass wir auf unser Netz für **Schätzungen** angewiesen sind.

Deep Q-Learning

Es gibt eine lange Historie von komplexer Theorie, wie man die **quality (Q-Wert)** von Aktionen schätzt.

quality ist der erwarteten **kumulierten Belohnungswert** (Reward)

statt diesen mühselig zu berechnen wird der **Q-Wert** nun **gelernt:**

💡 Der **Zustandsraum** wird bei Spielen schnell quasi **unendlich**, so dass wir auf unser **Netz für Schätzungen** angewiesen sind. (Schätzungen der eventuell theoretisch existierenden optimalen Spielzüge…).

Hier kommt die Kraft von **Mustererkennung** zum tragen!

💡Damit wird ein riesiger Berg von Theorie hinfällig:

💡 Deep-Mind (*https://homl.info/dqn*) konnte im Jahr 2013 demonstrieren, dass Deep Neural Networks besonders bei komplexen Aufgaben viel besser funktionieren als klassische Schätzverfahren und keinerlei Extraktion von Merkmalen nötig ist.

Deep Reinforcement Learning

Damit spart man es sich mühsam eine **optimale Strategie (Policy)** selbst zu Programmieren, das Netz lernt die *passenden* Aktionen automatisch!

Was gibt das **Netz** konkret aus?

Aktionen codiert als **Klassen:**
z.B. Joystick Bewegungen (links:0 rechts:1 …)
Tic-Tac-Toe **9 Positionen** als 9 Klassen
illegale Positionen (belegt oder not-in-reach) mit loss Funktionen bestrafen!

Aktionen codiert als **Werte** (Regression):
z.B. Kaufpreis, Beschleunigung(Geschwindigkeit)…

Wie wird es trainiert?

Actor-Critic-Modelle

Wie wird es trainiert?

In **Actor-Critic-Modellen** wird nicht jede einzelne Aktion bewertet sondern das Gesamtresultat aller Aktionen.

Actor wird auch **Policy Net** genannt: wählt Aktion
Critic wird auch **Value Net** genannt:
sagt **Q**ualität der Aktion(en) voraus

Objektiver reward am **Ende des Spiels => Training**

💡 **Critic** ist also so ähnlich wie unser **Diskriminator** beim GAN.

Credit-Assignment-Problem

Welche der gewählten **Aktionen** hat das End-Resultat **positiv oder negativ** beeinflusst?

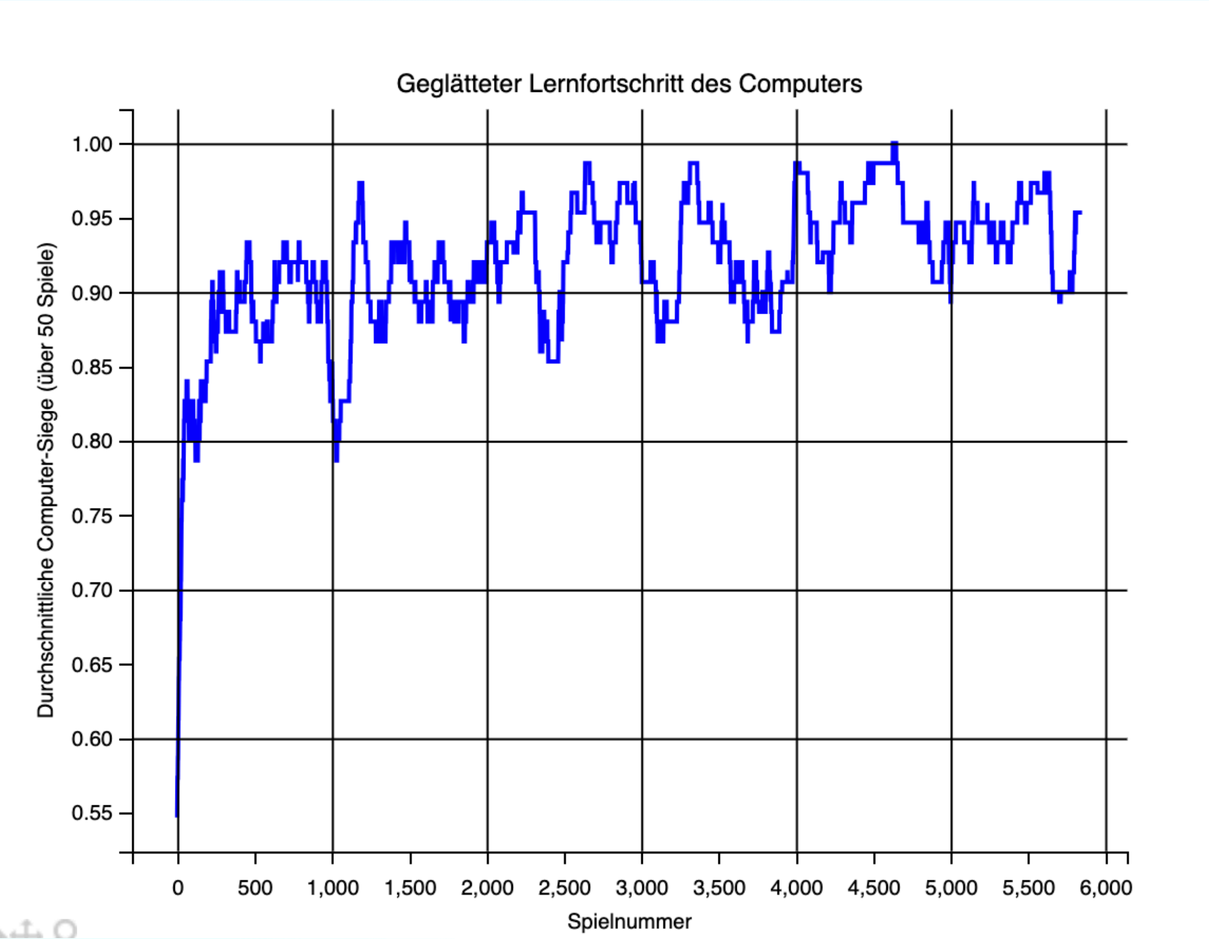
1) Egal, **Hauptsache gewonnen.** "absurde" Züge können oft überraschend zum Sieg führen.

2) **Discount-Faktor:** die **letzten Aktionen** führten unmittelbar zum Sieg? Belohne frühe Züge weniger, da sie Umwege darstellen könnten.

3) **Gesetz der großen Zahlen:** wenn wir oft genug spielen werden die positiven und negativen Aktionen in Summe schon in die richtigen Töpfe fallen

Deep Reinforcement Learning

Deep Reinforcement Learning in Tic-Tac-Toe
Gegen dummen Gegenspieler der zufällig spielt:



Aufgabe:

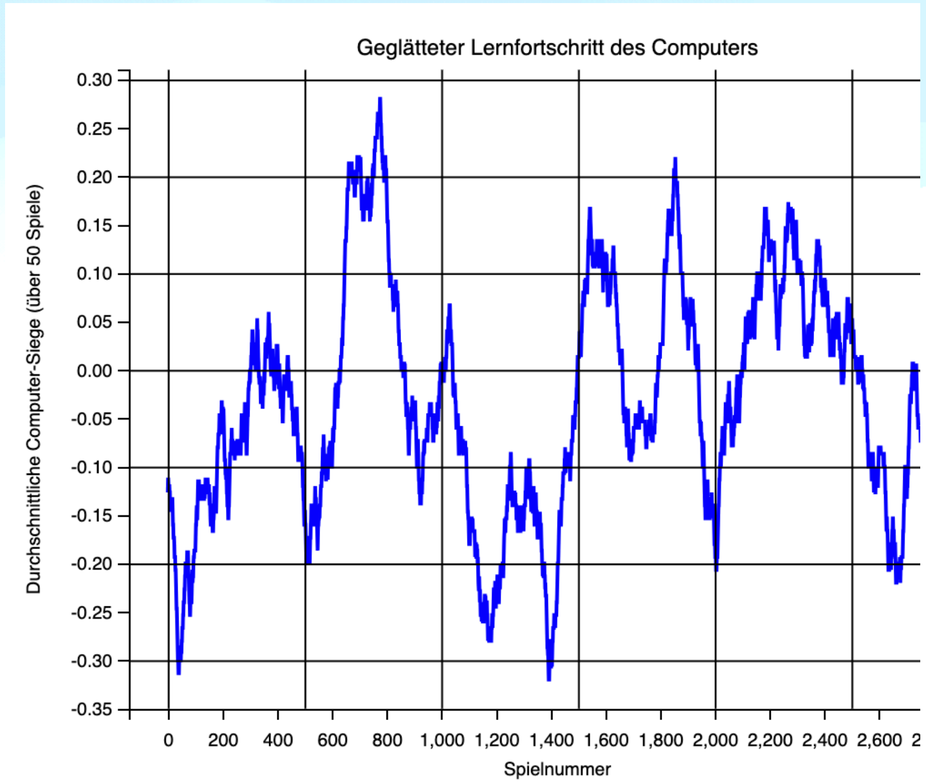
a) Mit tic_tac_toe_deep-q.py vertraut machen.

b) Welchen Einfluss haben in tic_tac_toe_deep-q.py veränderte Hyperparameter?

Nach ≈1000 Spielen tut sich nicht viel

c) Lasst Computer gegen sich selber spielen! ≈ Zeile 112 ff

Keiner kann gewinnen!



Vorteile und Nachteile von RL

Vorteile:

- **Anwendung in komplexen Umgebungen:**
  Kann komplexe, hochdimensionale Zustände verarbeiten, wie in Spielen (z.B. AlphaGo) oder Robotik.
- **Lernen ohne explizite Daten:**
  Kein annotiertes Trainingsset notwendig, da der Agent durch Interaktion mit der Umgebung lernt.

Nachteile:

- **Hohe Rechenleistung:** Erfordert oft viel Rechenleistung und Zeit für Training.
- **Instabilität:** Training kann instabil oder schwer zu optimieren sein, besonders bei kontinuierlichen Zustandsräumen.
- **Explorations-Exploitation-Dilemma:**
  Schwierigkeit, zwischen Entdeckung neuer Aktionen und Ausnutzung gelernter Strategien zu balancieren.

reward = 1 / loss

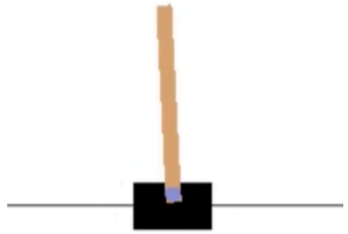
Im Grunde sind die Begriffe **reward** und **value-function** an welchen sich die Netze orientieren und worüber sie trainiert werden nichts anderes als **error/loss/…** Funktionen in neuem Gewand:

Unser Optimizer **maximiert reward** statt ihn zu minimieren.

Steuerung dynamischer Systeme

Anwendung von RL (reinforcement learning) auf RL (real life)

Dynamisches System: Ein System, das seinen Zustand über die Zeit basierend auf internen Regeln oder physikalischen Gesetzen verändert. Beispiele sind mechanische, elektrische, biologische oder chemische Systeme.
Steuerung: Der Prozess der Beeinflussung eines dynamischen Systems durch gezielte Eingaben (Kontrolleingaben), um ein bestimmtes Verhalten zu erreichen.
Regelung: Ein Spezialfall der Steuerung, bei dem ein Sollwert (z.B. eine bestimmte Position oder Temperatur) automatisch erreicht wird.



Auch im Computer: resource management, task scheduling, or control loops within operating systems.

Steuerung dynamischer Systeme

Dynamische Systeme und Modellierung

1. **Zustandsraum**darstellung: Ein häufig verwendetes Modell dynamischer Systeme, das das System durch Zustandsvariablen und Differentialgleichungen beschreibt.
   - Beispiel: Ein Pendel kann durch Winkel und Winkelgeschwindigkeit als Zustandsvariablen modelliert werden. als **Input/Output Vektor**
2. **Diskrete vs. kontinuierliche Systeme**: Dynamische Systeme können entweder kontinuierlich (z.B. durch Differentialgleichungen) oder diskret (z.B. durch Differenzengleichungen) beschrieben werden.

Steuerung dynamischer Systeme

Beispiele und Anwendungen

- **Autonome Fahrzeuge**: Benötigen eine präzise Steuerung dynamischer Systeme für Navigation und Stabilität.
- **Robotik**: In Robotern müssen Arme und Greifer oft mit hoher Genauigkeit gesteuert werden, was dynamische Modelle und Steuerung erfordert.
- **Prozessindustrie**: Chemische Reaktionen oder Produktionslinien sind dynamische Systeme, die optimal gesteuert werden müssen, um Effizienz und Qualität zu maximieren.

Simulation

Da Reinforcement Learning in der realen Welt teilweise sehr **hohe Kosten** verursacht gibt es eine große Motivation mit **Simulationen** zu arbeiten.

Von kleinen Spiel Umgebungen zu industriellen Ansätzen gibt es eine sehr große Bandbreite

OpenAI Gym (https://gym.openai.com/) # pip3 install gymnasium as gym

(Atari-Spiele, Brettspiele, physikalische Simulationen in 2-D und 3-D und so weiter)

NVIDIA & Siemens Digital Twin

ganze Industriehallen

WayMo autonome Autos (virtuelle Teststrecken)

…

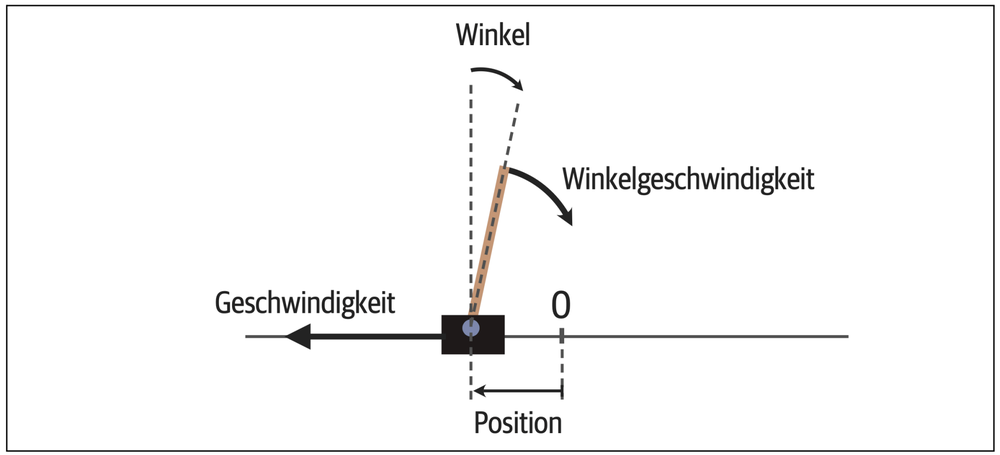


Abbildung 18-4: Die CartPole-Umwelt

Deep Q

Erweiterung des Konzept der *Heuristic*:

Memory Replay, auch **Experience Replay** genannt, speichert vergangene Erlebnisse (Erfahrungen) des Agenten während des Trainings in einem Speicher, der als Replay Buffer bezeichnet wird. Eine Erfahrung besteht normalerweise aus einem Tupel (s,a,r,s′), das den aktuellen Zustand **state s**, die gewählte **Aktion a**, die erhaltene Belohnung **reward r** und den nächsten Zustand **s′** beschreibt.

Im Training werden dann nicht nur die neuesten Erfahrungen genutzt, sondern zufällig ausgewählte Erfahrungen aus diesem Buffer. Dies hilft, Korrelationen in den aufeinanderfolgenden Daten zu reduzieren und das Training stabiler zu machen.

Deep Q

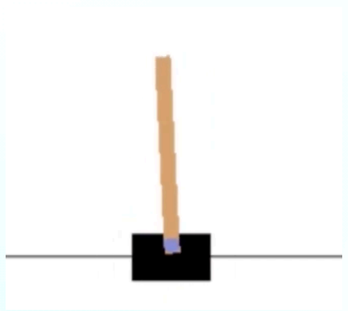
Exploration

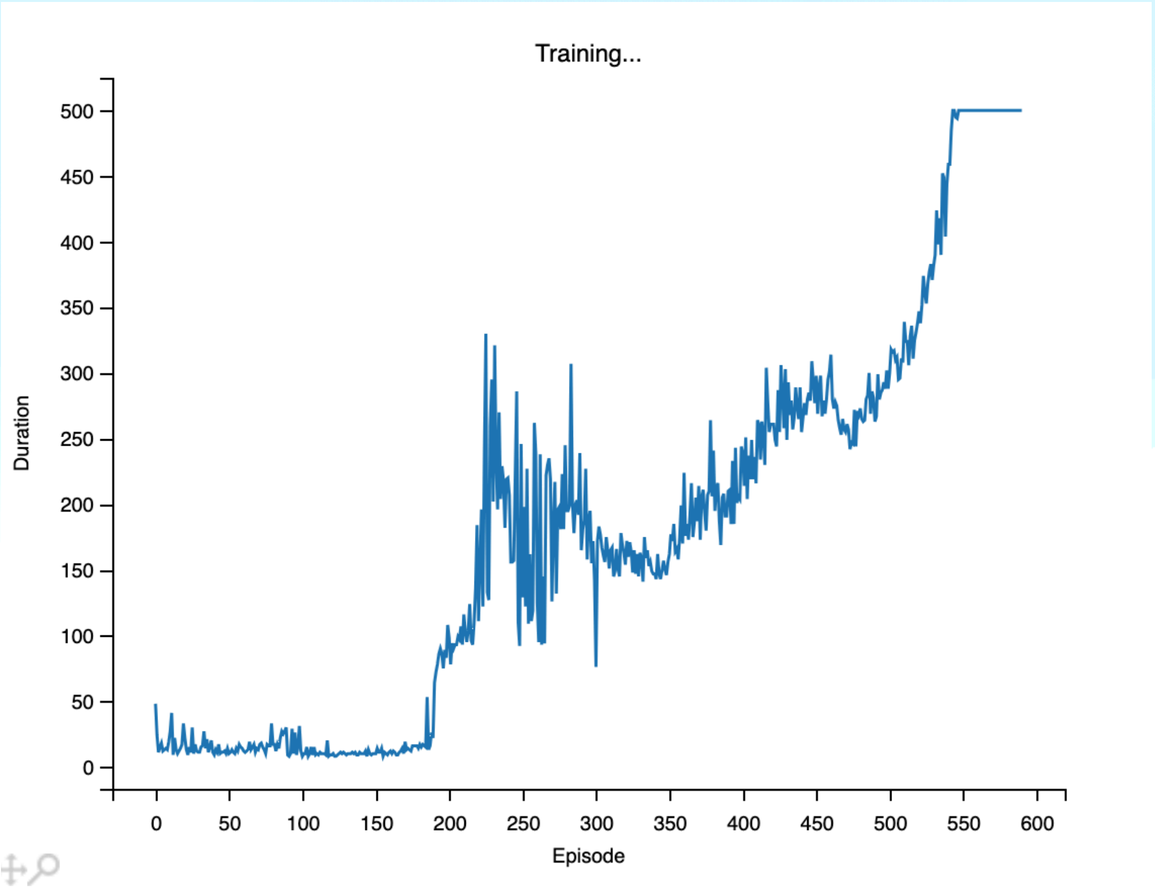
Anstatt sich beim Erkunden auf den Zufall zu verlassen, könnte die Erkundungs-Policy auch gezielt **noch nicht ausprobierte Aktionen** wählen.

Complicated **Exploration** policies and **rewards** je nach Spiel / Umgebung

Deep Q

Selbst balancierender Pole





Deep Q

https://github.com/sweetice/Deep-reinforcement-learning-with-pytorch

https://github.com/higgsfield/RL-Adventure/blob/master/1.dqn.ipynb Atari