Deep Reinforcement Learning

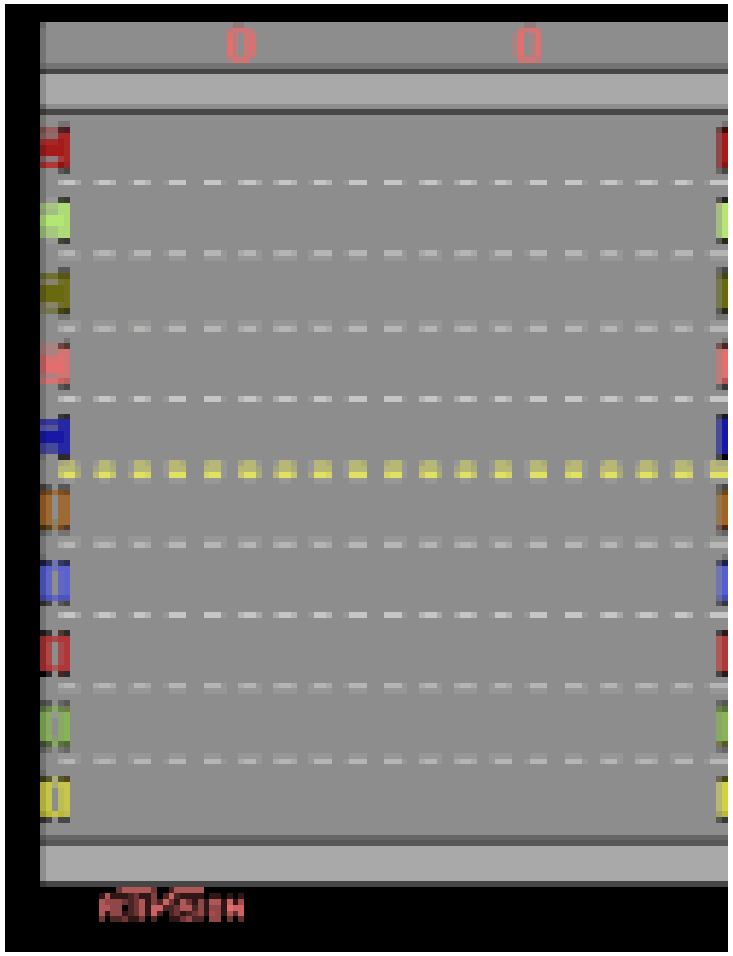
Finale Präsentation

Simon Lausch, Jan Felix Fuchs, Paul Jansen, Tobias Papen

Inhaltsverzeichnis

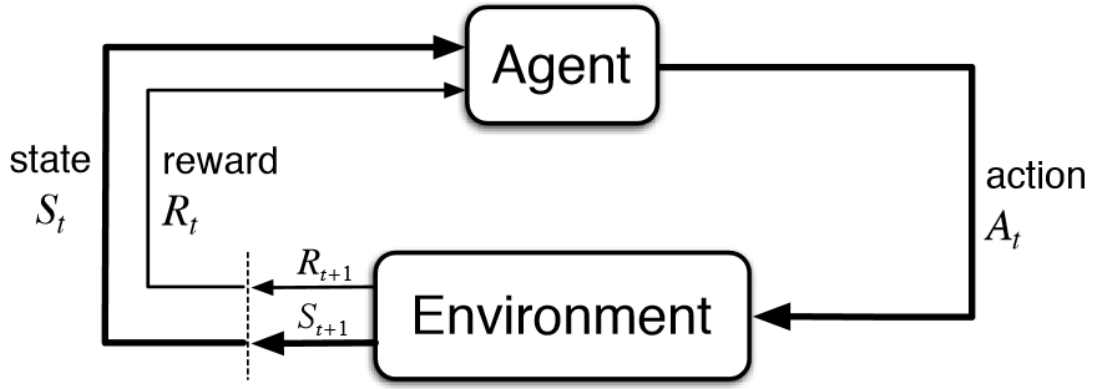
Wdh.: Environment

- Atari: Freeway
  - OpenAI Gym Library
  - Spieldauer 2:30 Minuten (2048 Steps)
  - 8 verschiedene Spielmodi
  - 2 verschiedene Schwierigkeiten
  - Observation
    - RGB
    - Grayscale
    - RAM



Wdh.: Reinforcement Learning

- Methode des maschinellen Lernens
- Bestandteile:
  - Agent
  - Environment
  - State
  - Action
  - Reward
- Exploration vs. Exploitation
- Ziel: Gesamtbelohnung maximieren
- Ansätze: model-based vs. model-free



Deep Q-Learning

Wdh.: Q-Learning

- Q-Table:
  - Matrix mit State-Action Paaren
- Q-Values:
  - Initialisierung mit dem Wert null
  - Temporal difference

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}_{\text{current value}}\Big)}^{\text{temporal difference}}$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

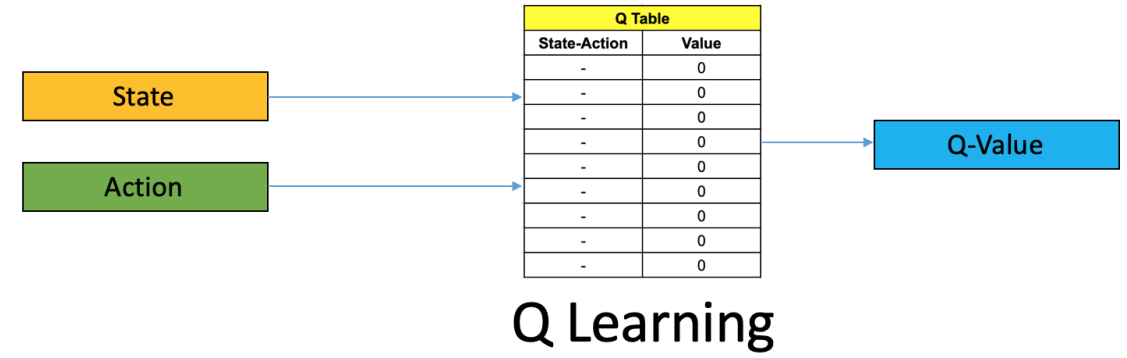
Actions

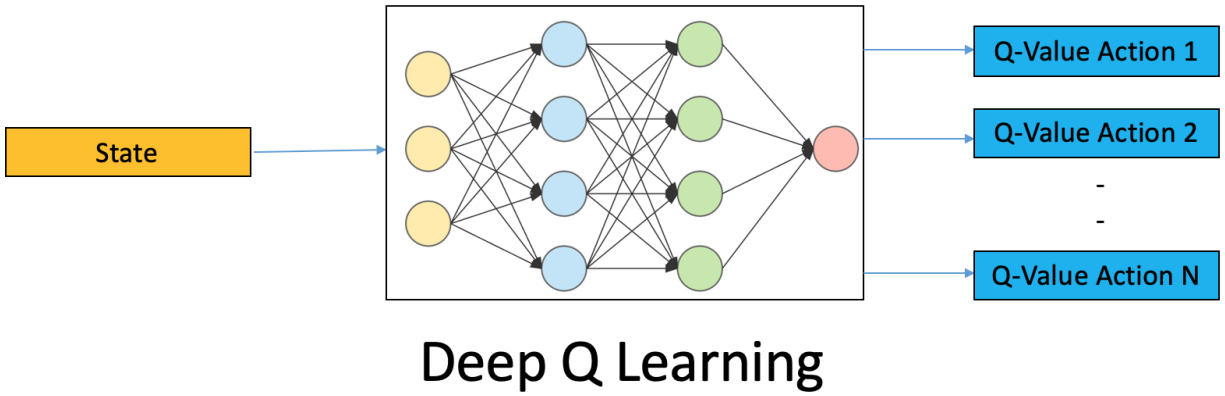
States	A_1	A_2	...	A_M
S_1	$Q(S_1, A_1)$	$Q(S_1, A_2)$		$Q(S_1, A_M)$
S_2	$Q(S_2, A_1)$	$Q(S_2, A_2)$		$Q(S_2, A_M)$
⋮			⋱	⋮
S_N	$Q(S_N, A_1)$	$Q(S_N, A_2)$	...	$Q(S_N, A_M)$

- Parameter:
  - Learning rate
  - Discount factor

Wdh.: Von Q-Learning zu Deep Q-Learning

- Berechnung der Q-Values durch ein tiefes neuronales Netz
  - Eingabe: State
  - Ausgabe: Approximierte Q-Values für die Actions
- Anpassung der Gewichte
  - Berechnung des "Loss"
  - Backpropagation
  - Optimierungsalgorithmen





Baseline - Aufbau

- Fully connected 3 layers
- ReLU for the first two layers
- Adam Optimizer
- Mittlere quadratische Abweichung (engl.: MSE) als loss function

```
#Lineare Transformation
#Pro Layer wird eine lineare Transformation angewandt
self.fc1 = nn.Linear(*self.input_dims, self.fc1_dims) #* entpackt die input_dims
self.fc2 = nn.Linear(self.fc1_dims, self.fc2_dims)
self.fc3 = nn.Linear(self.fc2_dims, self.n_actions)
self.optimizer = optim.Adam(self.parameters(), lr=lr)
```

- Store transition
- Batch sample learning
- Wahl des Epsilon Decrease
- Reward function

```
agent.store_transition(observation, action, reward, observation_, done)
```

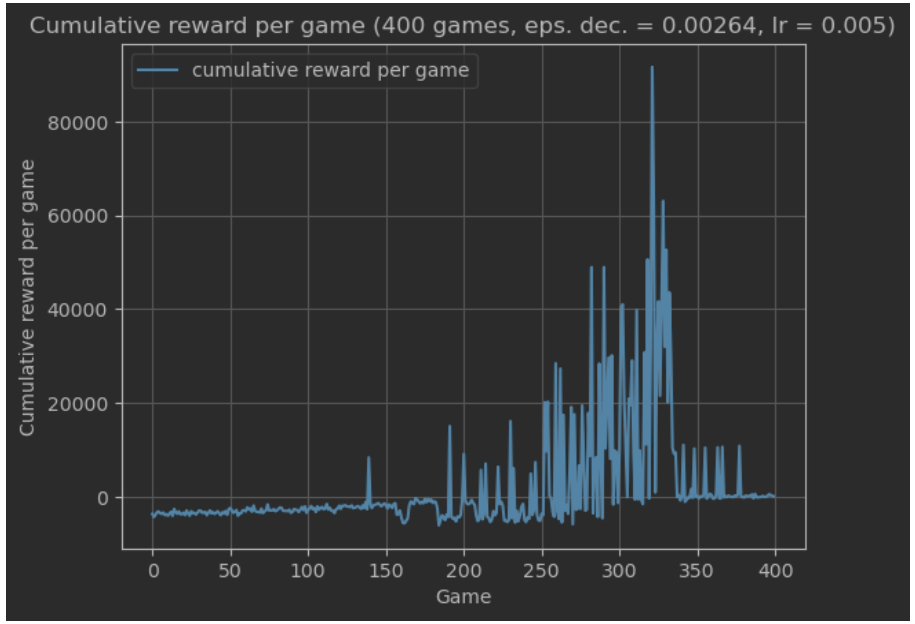
Baseline – Versuche

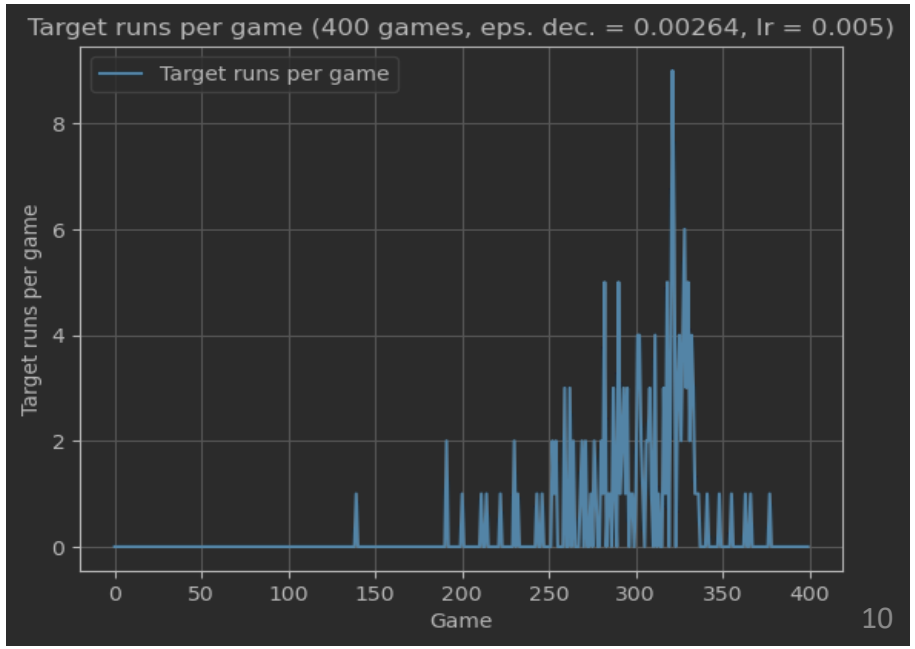
- Ziele der Versuche:
  - Optimale Hyperparameter finden
    - Epsilon abstieg
    - Learning rate
    - Discount factor
  - Optimale reward function finden
  - Ansätze auf erste Performance untersuchen

Baseline – Versuch 1

- Epsilon Decrease: 0.00264
- Learning rate: 0.005
- Durchläufe (Games): 400
- Reward function: Bewegungsabhängig

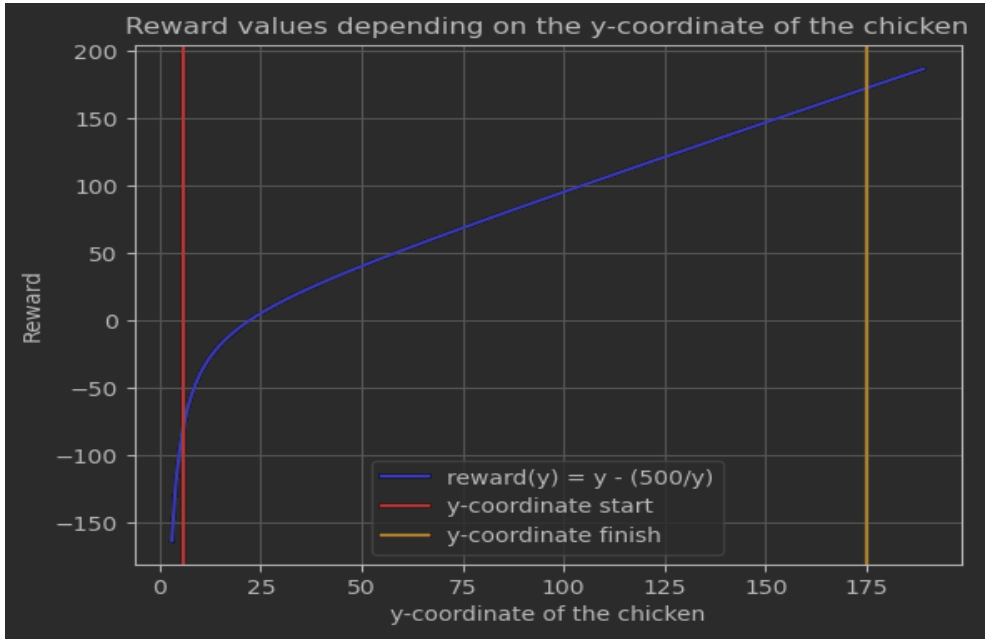
```
reward_goal = 10000                 #Reward for reaching the goal
reward_forwards = 10                #Reward for moving forwards
reward_backwards = -10              #Reward for moving backwards
reward_nooperation = 0              #Reward for standing still
reward_crash = -100                 #Reward for colliding with a car
```

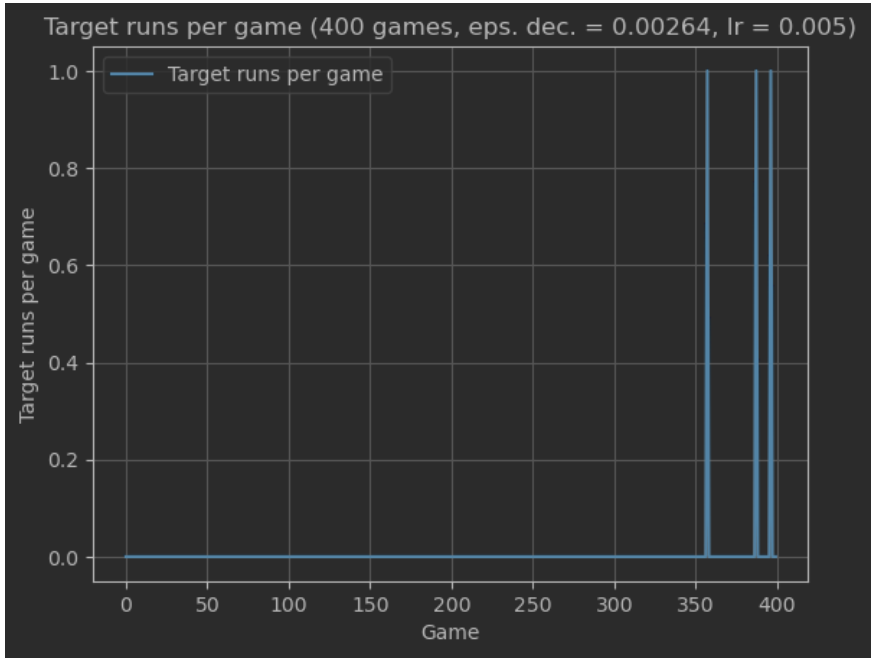


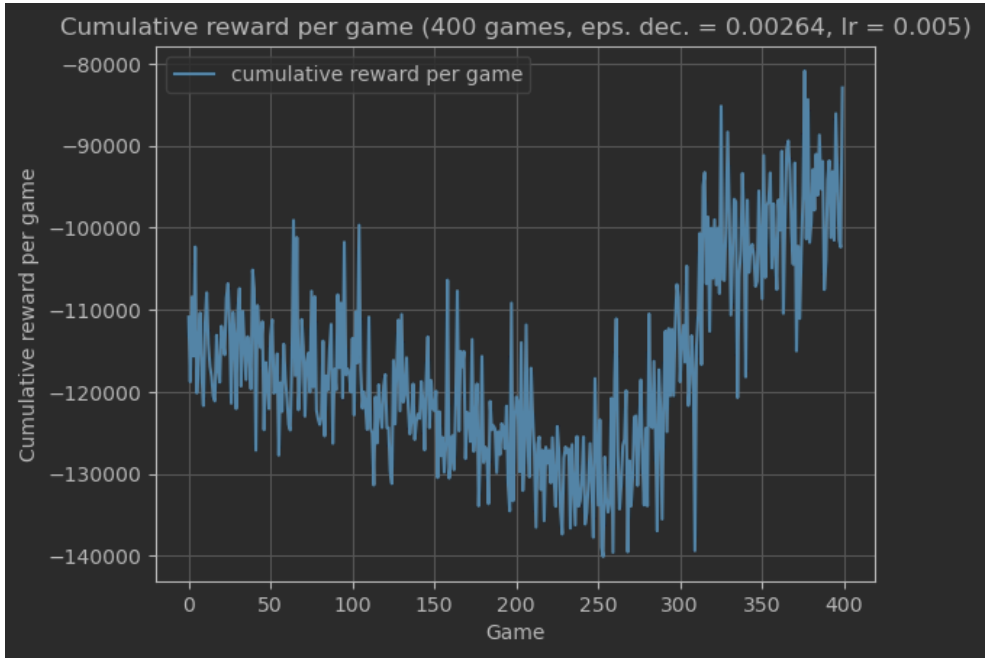


Baseline – Versuch 11

- Epsilon Decrease: 0.00264
- Learning rate: 0.005
- Durchläufe (Games): 400
- Reward function: nicht linear

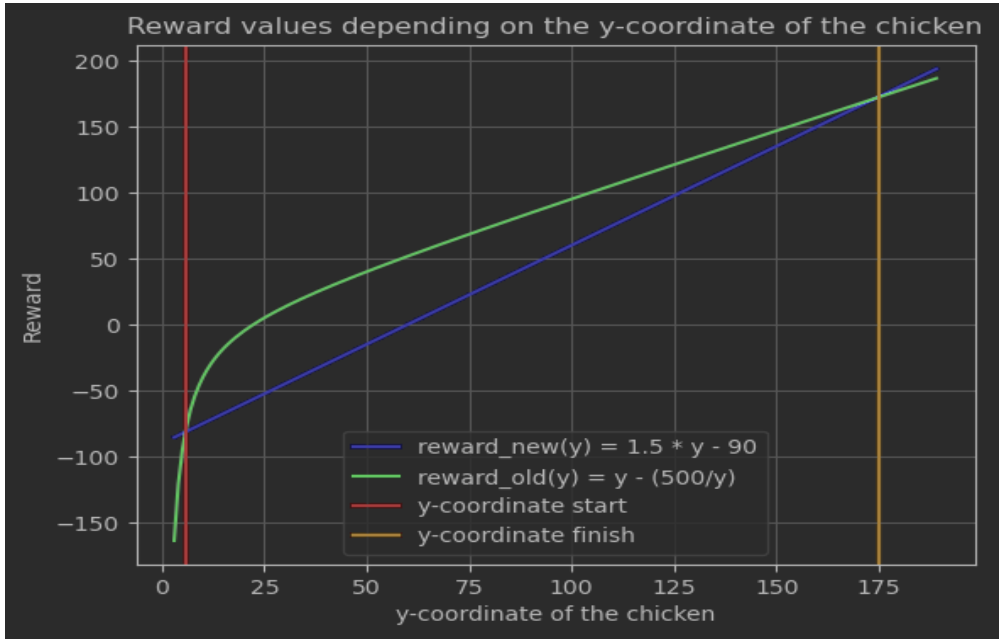


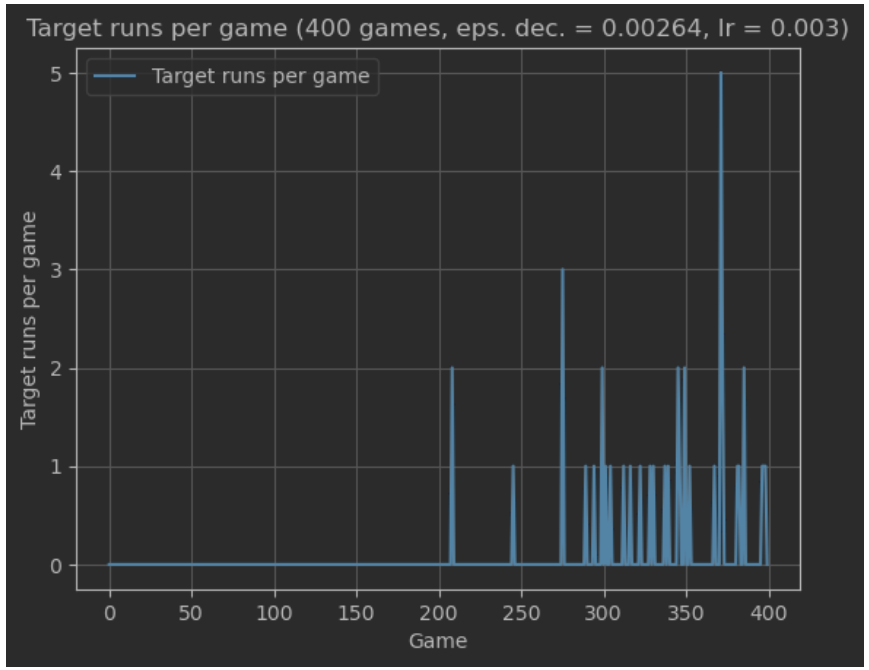


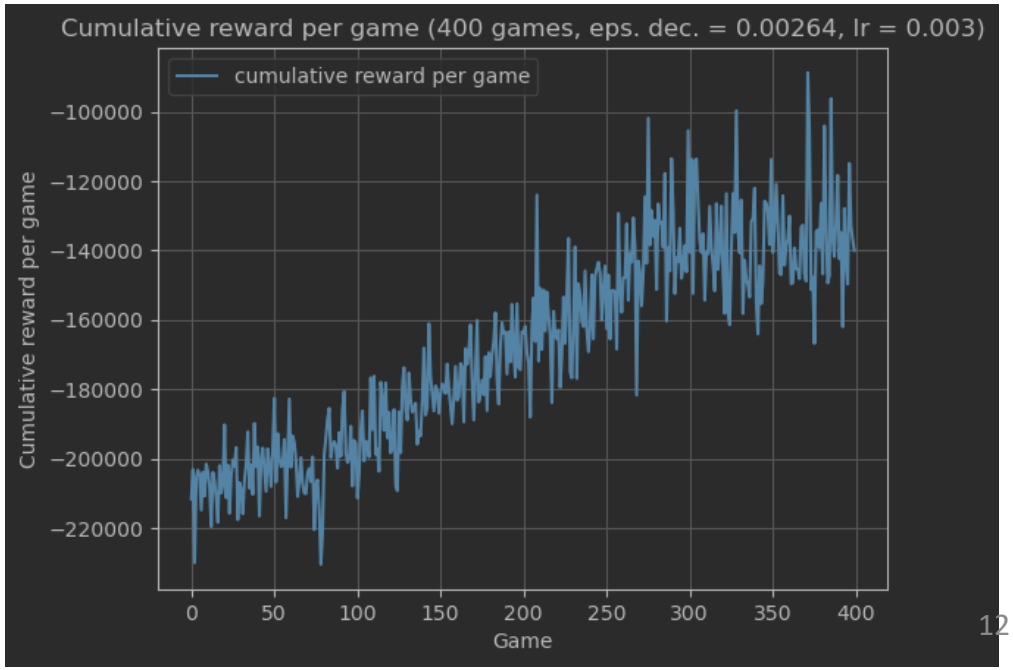


Baseline – Versuch 16

- Epsilon Decrease: 0.00264
- Learning rate: 0.003
- Durchläufe (Games): 400
- Reward function: Linear







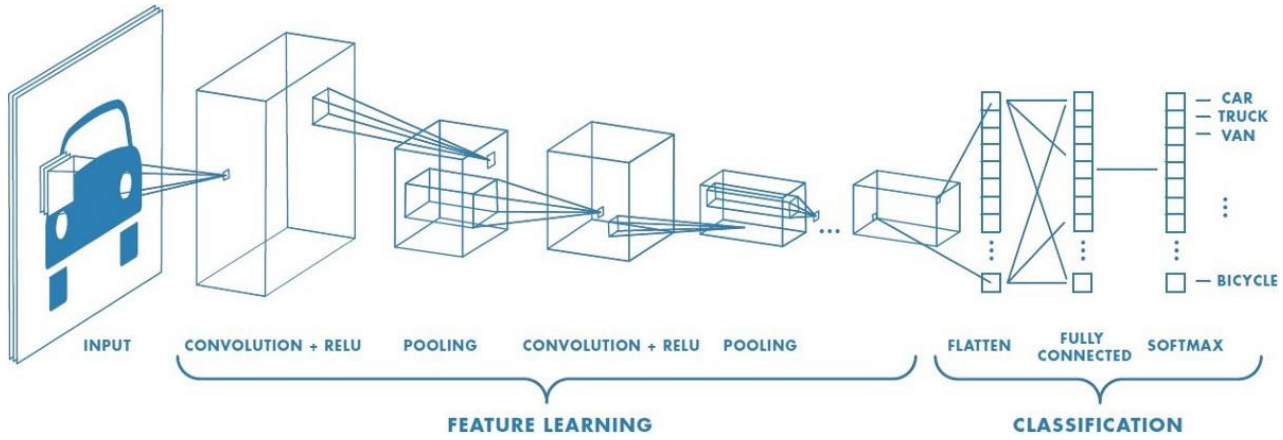
Ansatz – Cropped RAM

- Observation Space begrenzt auf
  - Position d. Autos
  - Position d. Hühnchens
  - Cooldown der Hühnchen

- Input Dimension: 13 vs 128
  - Schneller

Ansatz – CNN mit grayscale image

- Convolutional Layer
  - ReLU Funktion
- MaxPooling
- Convolutional Layer
  - ReLU Funktion
- MaxPooling
- Flattening
- Fully connected Layer
- Fully connected Layer
- Softmax



Ansatz – CNN mit rgb image

Observation Typ: RGB Array

Neuronales Netz:

- Convolutional Layer (Conv2D)
- BatchNorm2D
- ReLU (Aktivierungsfunktion)

ME-TRPO

ME-TRPO Wiederholung

- Model Ensemble
  - Model des Environments wird als Simulation des Environments verwendet
  - Mehrere Models werden zusammengeführt
    - Hohe Robustheit
    - Erhöhter Zeitaufwand
- Trust Region Policy Optimization
  - Policy Gradient
  - Beschränkung wie weit sich die Policy bei jedem Update verändern darf
  - Library von stable-baselines3

Model Ensemble – Neuronales Netzwerk

- Mögliche Arten ein Model zu erstellen
  - Statistische Modelle
    - Gauß Prozess
  - Maschinelles Lernen Modelle
    - Neuronale Netze
    - Support Vector Regression
- Neuronale Netze
  - Vorteile
    - Komplexe Beziehungen in Daten erkennen
    - Verhalten von Systemen genau beschreiben
  - Nachteile
    - Erfordern viele Daten

Model Ensemble – Daten

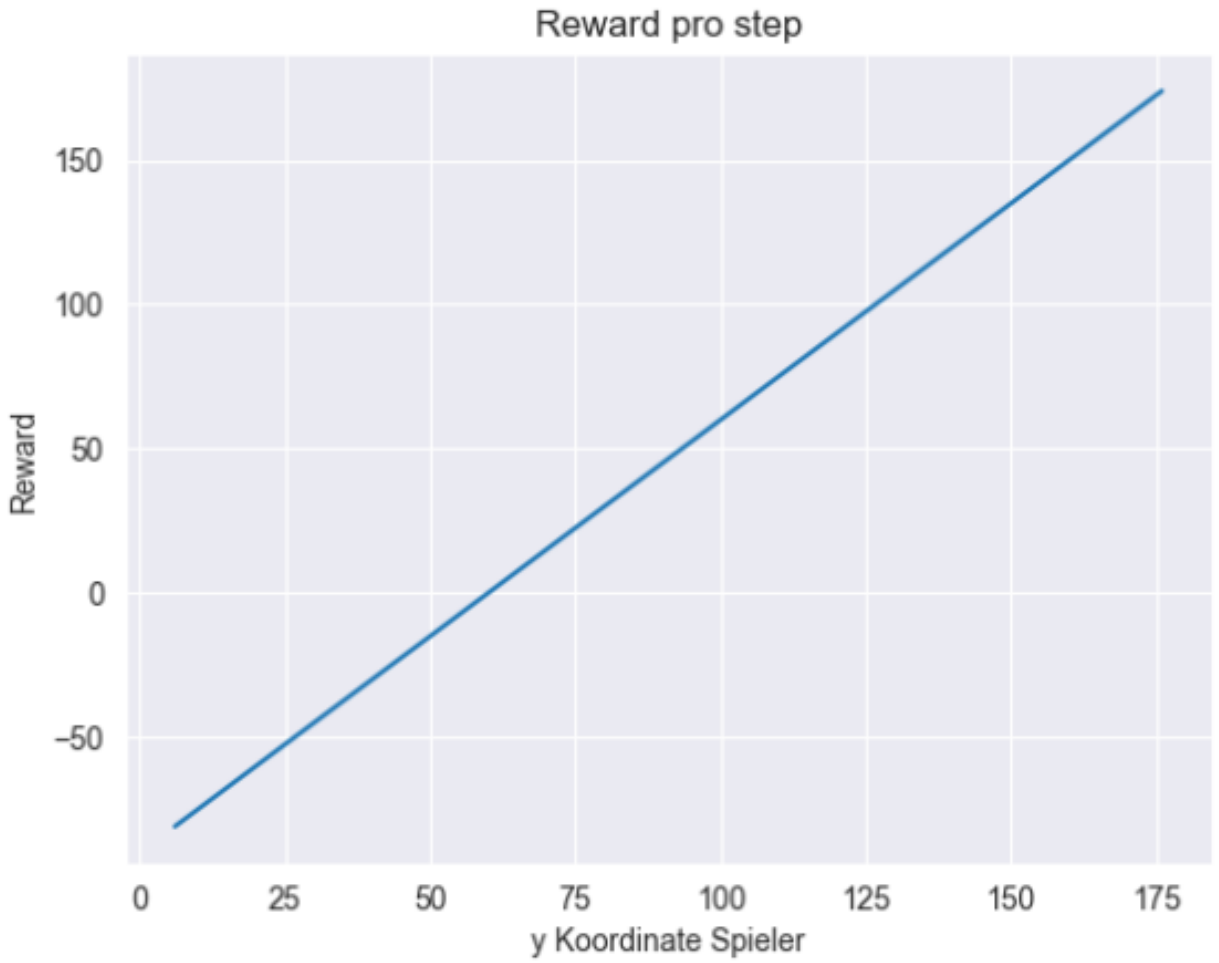
- Pro Epoche werden Daten von 4 Spielen gesammelt (8192 Samples)
- Numerische Werte
- State
  - Y Koordinate Spieler
  - X Koordinate Autos
  - Score
  - Cooldown
- Actions
  - Oben
  - Unten
  - Nichts
- Neuronales Netz: $f(s, a) = \hat{s}$

Model Ensemble – Training & Testen

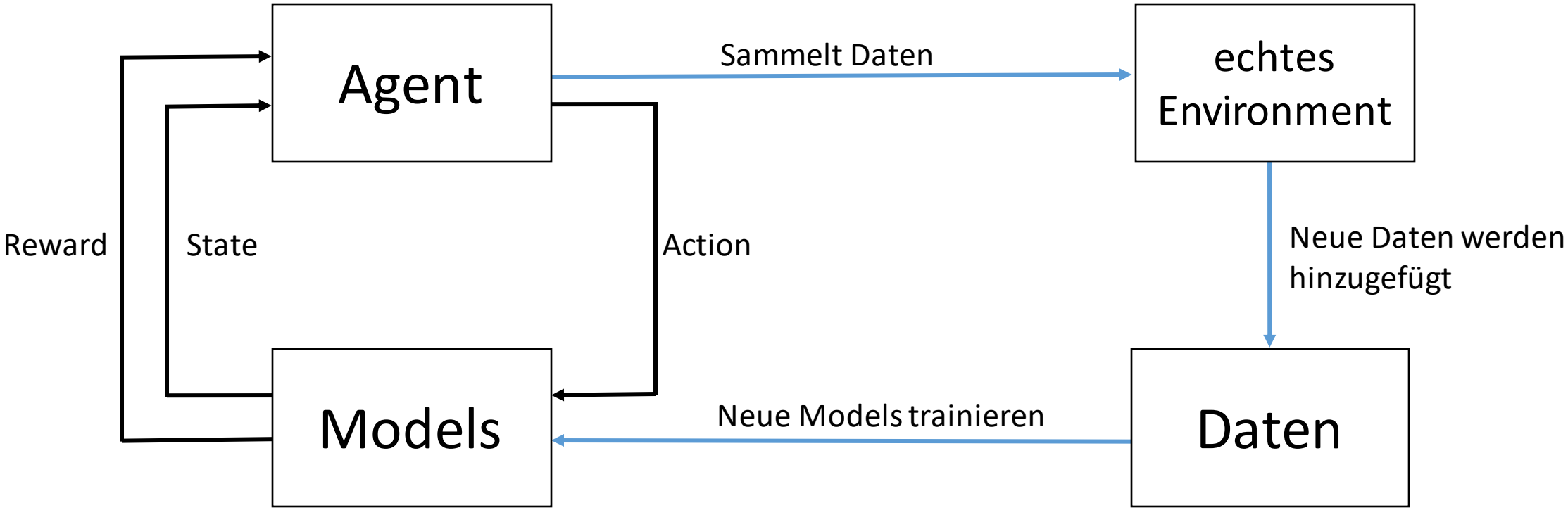
- Neuronales Netz:
  - 14 Input Neuronen
  - 1 Hidden Layer mit 256 Neuronen
  - 13 Output Neuronen
- 70/30 Train Test Split
- Loss Function: Mean Squared Error
  - $\frac{1}{n} * \sum(y_i - y_i')^2$
- Optimizer: Adam
- Batchsize: 234
  - Ganzer Batch wird auf GPU gespeichert um schneller zu trainieren
- Learning Rate: 0.0014

Reward Function

- Für jeden step einen Grundreward $r = \frac{3}{2}y - 90$
- 10.000 jedes mal wenn man ins Ziel läuft
- -10.000 jedes mal wenn man von einem Auto angefahren wird



ME-TRPO Aufbau

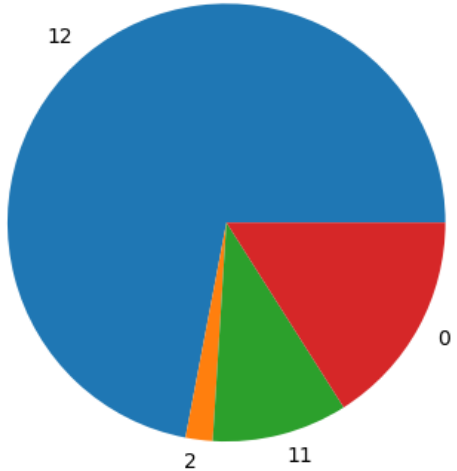


Ergebnisse

Ergebnisse – Ansatz Cropped RAM

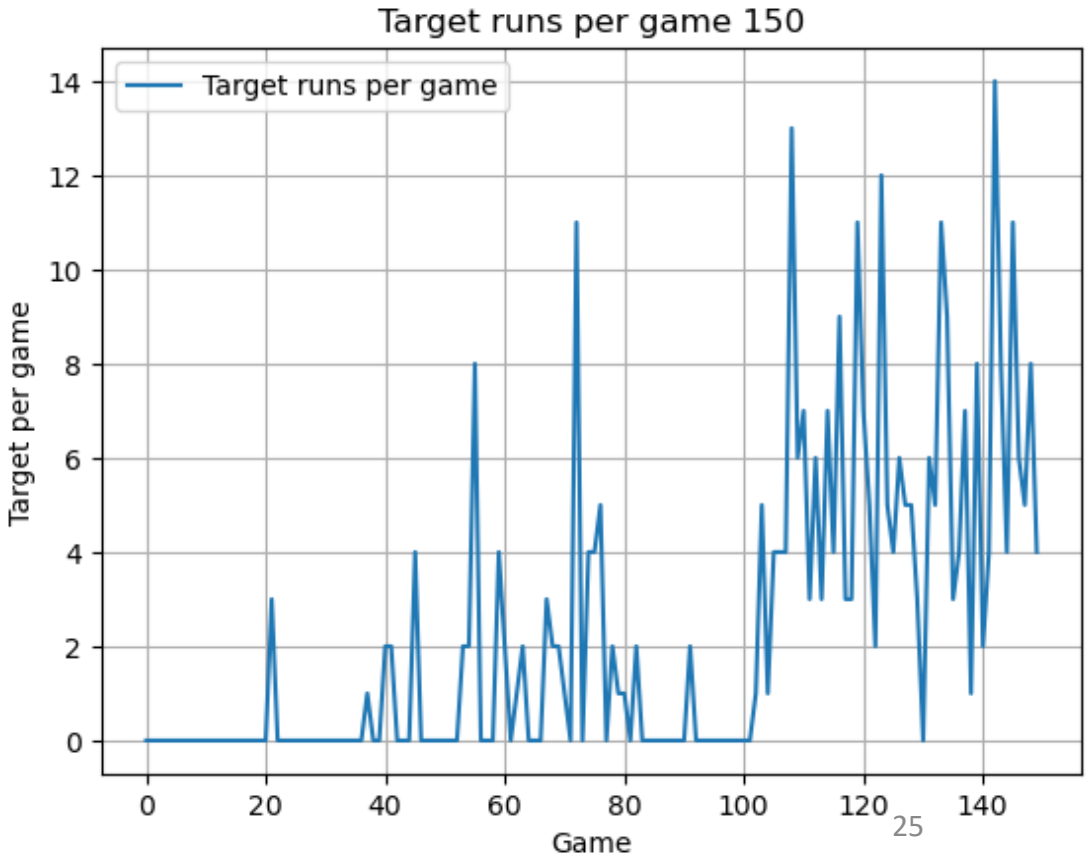
- Gelernt wurde in 400 Spielen
  - Timesteps = Spiele * 2048 = 819.200
- Getestet wurde in 50 Spielen

Score	Häufigkeit	Crashes
0	8	71
2	1	10
11	5	53 oder 54
12	36	52



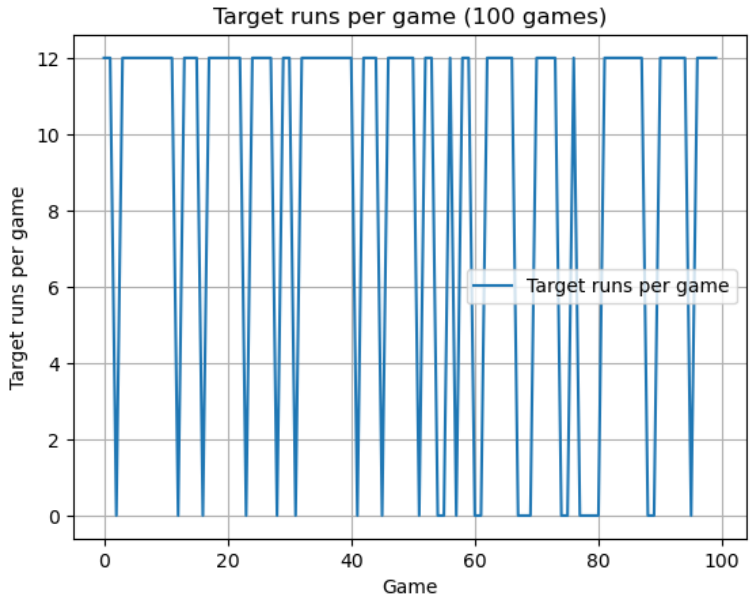
Ergebnisse – Ansatz CNN mit RGB image

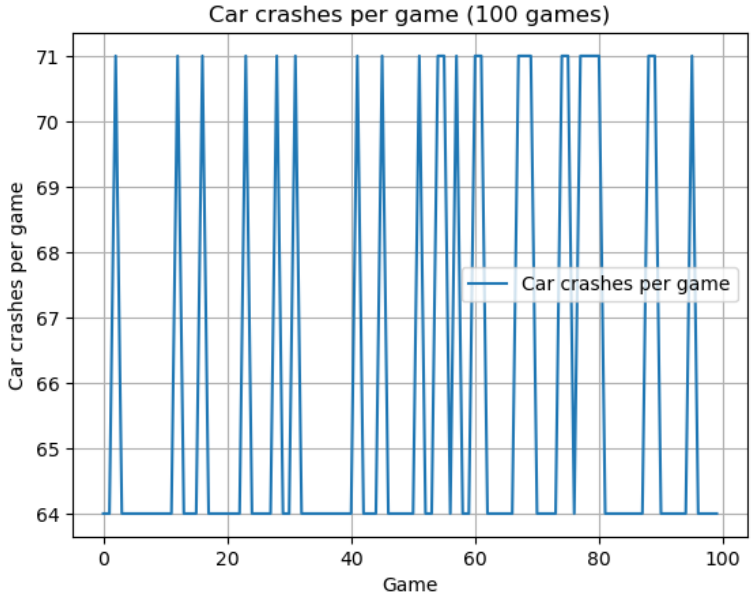
- Durchschnitt (Score letzten 30 Games): 6.03
- Maximal Score: 14
- Minimal Score: 0
- Timesteps: 307.200



Ergebnisse – Ansatz CNN mit grayscale image

- Getestet wurde in 100 Spielen
  - Durchschnittlicher Score: 8,88
  - Minimaler Score: 0 (x26) mit jeweils 71 Kollisionen
  - Maximaler Score: 12 (x74) mit jeweils 64 Kollisionen
  - Durchschnittliche Anzahl an Kollisionen: 65,82
- Timesteps (Training) = 100 * 2048 = 204800



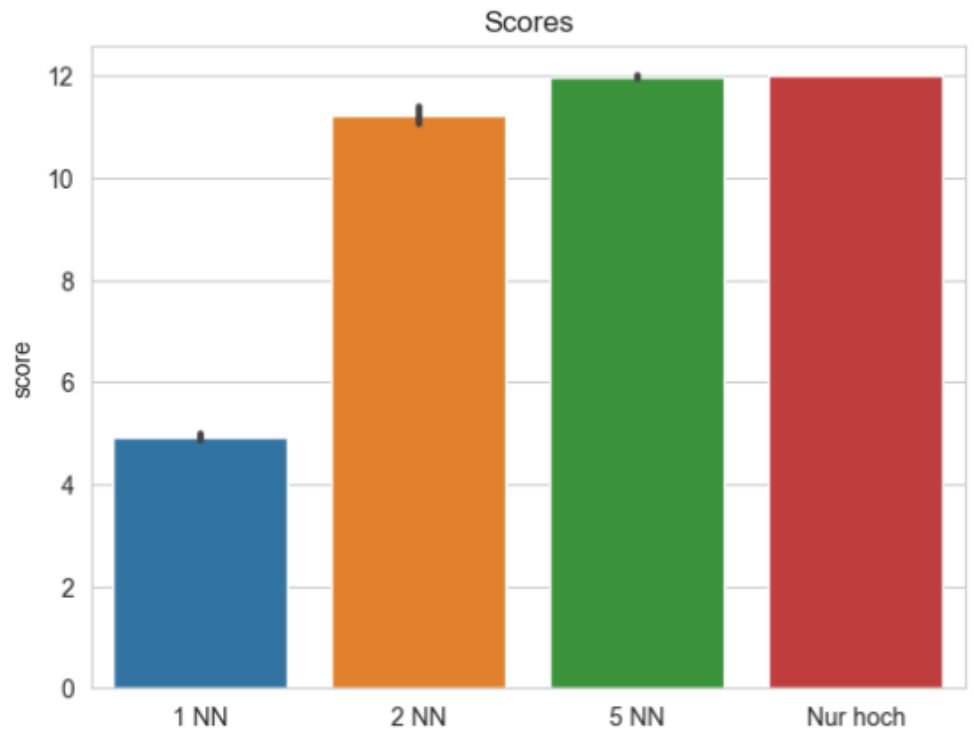


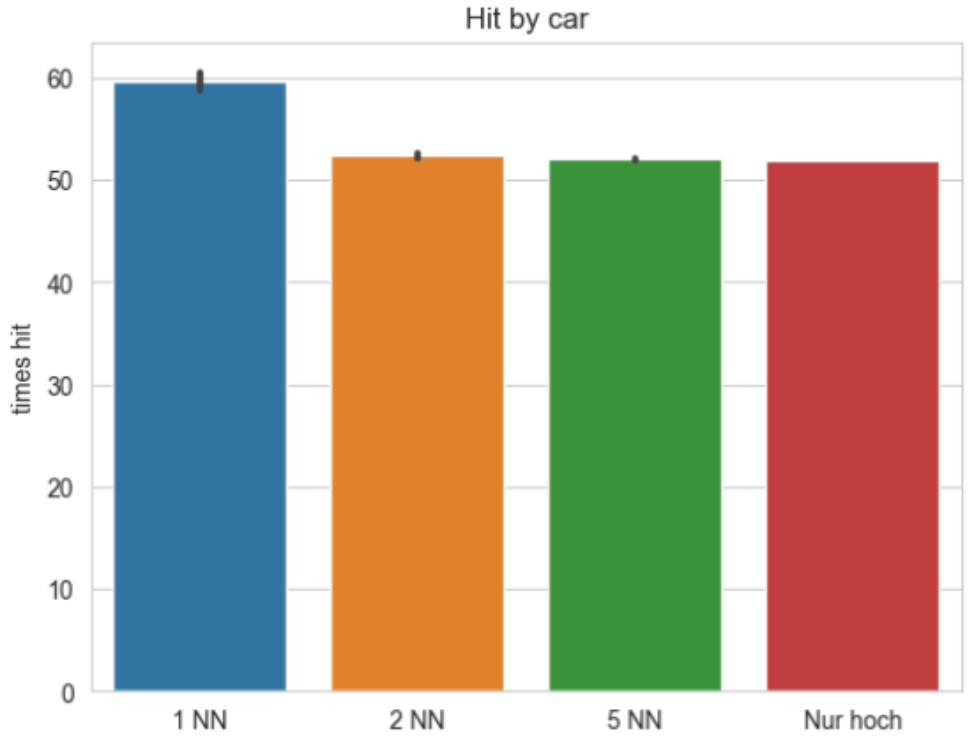
Ergebnisse – Ansatz CNN mit grayscale image

- Ausblick
  - Optimierung bzgl. Performance
  - Umfangreicheres Training (mehr Spiele)
  - Änderungen im Aufbau des CNN
  - Alternative Funktionen zur Berechnung des Loss (statt MSE)
  - Alternative Algorithmen zur Optimierung (statt Adam)

Ergebnisse – ME-TRPO

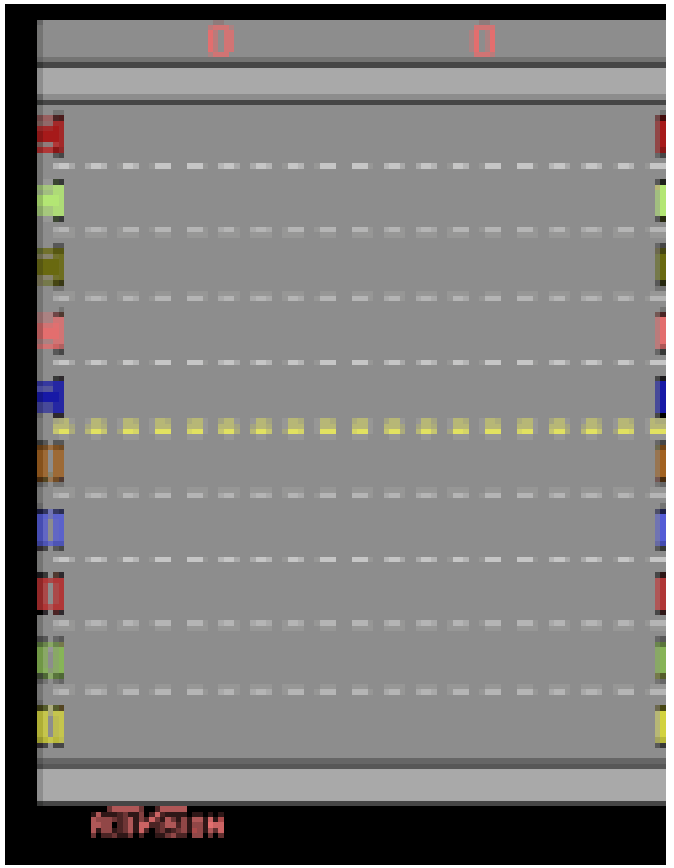
- Timesteps (echtes Environment): 204.800 (100 Spiele)
- Timesteps (Simulation): 12.800.000 (6250 Spiele)





Ergebnisse - ME-TRPO – Fazit

- Was hat funktioniert?
  - Reward Function
  - Verkleinerter Observation space
- Woran hat es gelegen?
  - Models nicht genau genug
    - Score
    - Hit
- Ausblick
  - Trainingszeit und Genauigkeit der Neuronalen Netze optimieren



Gemeinsamer Vergleich

Score	Paul	Simon	Jan Felix	Tobias 1NN	Tobias 2NN	Tobias 5NN
Minimum	0	0	0	4	10	11
Maximum	14	12	12	6	12	12
Average	6.03	9.78	8.88	4.92	11.22	11.98

GitHub Repositories

- Zum Repository von Simon, Jan Felix und Paul:
  - TobiasPapen/projekt-seminar-deep-q-learning (github.com)
- Zum Repository von Tobias:
  - TobiasPapen/projekt-seminar-me-trpo (github.com)