

# BACHELORARBEIT

## Learning to Play Pommerman with Emergent Communication

Patricia Gschoßmann





# BACHELORARBEIT

## Learning to Play Pommerman with Emergent Communication

Patricia Gschoßmann

Aufgabensteller: Prof. Dr. Claudia Linnhoff-Popien

Betreuer: Thomy Phan  
Thomas Gabor

Abgabetermin: 1. April 2020





Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 1. April 2020

.....  
*(Unterschrift des Kandidaten)*



## **Abstract**

In vielen Aufgaben künstlicher Intelligenz müssen Agenten in der Lage sein, ihr Verhalten untereinander zu koordinieren. Koordination dieser Art benötigt Kommunikation. Nach dieser Definition ist Kommunikation nicht nur Austausch von Information, sondern involviert auch die *Verständigung*. Die partiell beobachtbare „Team-Radio“-Variante der Multi-Agenten Domäne Pommerman ist in dieser Arbeit das Environment, in dem die Agenten mit und gegeneinander agieren. Hier treten vier Agenten in Zweierteams gegeneinander an, wobei Agenten innerhalb eines Teams durch einen diskreten Kanal miteinander kommunizieren können. Der Fokus dieser Arbeit liegt auf der Frage, ob Kommunikation in Pommerman erlernbar ist und, wenn ja, ob die Agenten diese auch gewinnbringend nutzen. Dafür wurde ein Team entwickelt und mittels Deep Recurrent Q-Learning und Value Decomposition trainiert. Das Modell besteht aus zwei rekurrenten Netzen pro Agent, wovon eines für die Nachrichten- und eines für die Aktionsauswahl verwendet wird. Das Training erfolgt gegen unterschiedlich anspruchsvolle Gegner. Die Ergebnisse werden mit der Performance verschiedener Teams verglichen, die nicht miteinander kommunizieren. Des Weiteren wird das Kommunikationsprotokoll der Agenten anhand ausgewählter Metriken analysiert. Im Detail werden unter anderem Speaker Consistency und Instantaneous Coordination betrachtet.

Es konnte gezeigt werden, dass die Agenten lernen, einfache Gegner zu besiegen, die Performance nicht-kommunikationsfähiger Modelle konnte im Allgemeinen allerdings nicht übertroffen werden. Die Performance in Spielen gegen schwierigere Teams wurde verbessert, ist aber noch weit von bestmöglichen Ergebnissen entfernt. Die Agenten lernen, den Kommunikationskanal zu nutzen. Auch wenn das Potenzial davon nicht vollständig ausgeschöpft wurde, konnte gezeigt werden, dass Kommunikation in Pommerman erlernbar ist.



# Inhaltsverzeichnis

<b>Inhaltsverzeichnis</b>	<b>i</b>
<b>1. Einleitung</b>	<b>1</b>
1.1. Motivation . . . . .	1
1.2. Zielsetzung . . . . .	1
1.3. Struktur dieser Arbeit . . . . .	2
<b>2. Grundlagen</b>	<b>3</b>
2.1. Reinforcement Learning . . . . .	3
2.1.1. Markov Decision Process . . . . .	3
2.1.2. Partiell beobachtbare MDPs . . . . .	6
2.1.3. Kooperative Multi-Agenten Systeme . . . . .	6
2.1.4. Nicht-kooperative Multi-Agenten Systeme . . . . .	8
2.2. Künstliche neuronale Netze . . . . .	8
2.3. Temporal-Difference Learning . . . . .	11
2.3.1. Q-Learning . . . . .	12
2.3.2. Deep Q-Learning . . . . .	13
2.3.3. Deep Recurrent Q-Learning . . . . .	15
2.3.4. Deep Q-Learning in kooperativen Multi-Agenten Systemen . . . . .	15
<b>3. Pommerman</b>	<b>17</b>
3.1. Motivation . . . . .	17
3.2. Spielfeldaufbau und Verlauf . . . . .	18
3.2.1. Aufbau . . . . .	18
3.2.2. Regeln und Verlauf des Spiels . . . . .	18
3.3. Herausforderungen . . . . .	19
3.4. Agenten . . . . .	20
<b>4. Verwandte Arbeiten</b>	<b>21</b>
4.1. Deep Q-Learning . . . . .	21
4.1.1. Einzel-Agenten Systeme . . . . .	21
4.1.2. Multi-Agenten Systeme ohne Kommunikation . . . . .	23
4.2. Emergente Kommunikation . . . . .	23
4.3. Ansätze zu Pommerman . . . . .	30
4.3.1. Non-Deep Learning Methoden . . . . .	30
4.3.2. Deep Learning Methoden . . . . .	31
<b>5. Herangehensweise</b>	<b>37</b>
5.1. Modellaufbau . . . . .	37
5.1.1. Datenvorverarbeitung . . . . .	37

## **INHALTSVERZEICHNIS**

5.1.2. Architektur ohne Kommunikation . . . . .	37
5.1.3. Architektur mit Kommunikation . . . . .	38
5.2. Trainings- und Testsetup . . . . .	41
5.3. Metriken zur Analyse der Kommunikation . . . . .	41
<b>6. Ergebnisse und Evaluation</b>	<b>43</b>
6.1. Reward und Task Completion mit und ohne Kommunikation . . . . .	43
6.1.1. Lernkurven im Vergleich . . . . .	43
6.1.2. Gewinnraten im Vergleich . . . . .	46
6.2. Qualitative Analyse der Kommunikation . . . . .	48
6.2.1. Zusammenhang zwischen Nachrichten und Spielzuständen . . . . .	48
6.2.2. Speaker Consistency . . . . .	52
6.2.3. Instantaneous Coordination . . . . .	55
6.3. Interpretation der Nachrichten . . . . .	57
6.4. Performance bei Entfernen der Kommunikation . . . . .	58
<b>7. Fazit und Ausblick</b>	<b>61</b>
7.1. Zusammenfassung . . . . .	61
7.2. Weitere Überlegungen zur Forschung . . . . .	62
<b>A. Anhang</b>	<b>65</b>
<b>Abbildungsverzeichnis</b>	<b>73</b>
<b>Tabellenverzeichnis</b>	<b>75</b>
<b>Liste der Algorithmen</b>	<b>77</b>
<b>Literaturverzeichnis</b>	<b>79</b>

# 1. Einleitung

## 1.1. Motivation

Kommunikation ist ein zentraler Aspekt menschlicher Intelligenz. Sie ermöglicht es uns, unser Handeln zu koordinieren, indem wir Information teilen. Nach dieser Definition ist Kommunikation nicht nur der Austausch von Daten, sondern involviert auch die *Verständigung*. Dadurch sind wir im Stande, nicht nur als Individuum, sondern auch als Gruppe zu agieren [SSF16]. Das ist vor allem in Situationen mit unzureichenden Informationen und limitierten Fähigkeiten notwendig.

Environments mit diesen Eigenschaften stellen eine besondere Herausforderung im Reinforcement Learning [SB98] dar, denn nicht nur die nächsten Aktionen der anderen Agenten sind ungewiss, sondern auch der zugrundeliegende Zustand des Environments. Kommunikation in partiell beobachtbaren, kooperativen Multi-Agenten Systemen ist demnach essentiell, um effektiv den maximalen Reward zu erreichen.

Die Entwicklung kommunikationsfähiger Agenten ist eines der Hauptziele künstlicher Intelligenz. Die Forschung ist teilweise durch den Gedanken motiviert, dass Kommunikation durch Interaktion mit der Umgebung entstehen und sich so menschlicher Sprache annähern kann [MJB15], [LUTG17]. Denn jede praktische Umsetzung einer intelligenten Maschine muss mit uns kommunizieren, damit wir die Ziele komplexer Operationen spezifizieren und die Ausgabe der Maschine verstehen können [MJB15].

## 1.2. Zielsetzung

Die partiell beobachtbare Multi-Agenten Domäne Pommerman [REH<sup>+</sup>18], ein qualitativ hochwertiger Benchmark im Multi-Agenten Reinforcement Learning, ist in dieser Arbeit das Environment. Hier treten vier Agenten in Zweierteams gegeneinander an, wobei Agenten innerhalb eines Teams via eines diskreten Kanals kostenlos miteinander kommunizieren können. Der ausgesprochen große Zustandsraum und eine spärliche Rewardfunktion stellen neben der partiellen Beobachtbarkeit weitere Herausforderungen des Spiels dar.

Ziel dieser Arbeit ist es, herauszufinden, ob Kommunikation in Pommerman erlernbar ist und, wenn ja, ob die Agenten diese auch zu ihren Gunsten nutzen. Hierfür werden mittels Deep Recurrent Q-Learning und Value Decomposition Teams aus zwei Agenten zentral trainiert und deren Kommunikationsprotokoll aus dezentral ausgeführten Tests analysiert. Jeder Agent verwendet zwei rekurrente Netze, eines für die Kommunikation und eines für die nächste Aktion. Beide Netze werden unabhängig voneinander optimiert. Das Training erfolgt gegen drei verschiedene Gegner unterschiedlichen Schwierigkeitsgrades. Die Idee dahinter ist, herauszufinden, inwiefern sich das Kommunikationsverhalten abhängig vom Komplexitätsniveau ändert bzw. als hilfreich erweist. Die Ergebnisse werden mit der Performance verschiedener Teams verglichen, unter anderem einem Team

## *1. Einleitung*

mit zwei Agenten, die ein Aktionsnetz mit ähnlicher Architektur verwenden, aber nicht miteinander kommunizieren.

## **1.3. Struktur dieser Arbeit**

Die Arbeit ist wie folgt strukturiert:

- Kapitel 2 definiert theoretische Grundlagen, die für das Verständnis dieser Arbeit notwendig sind.
- Kapitel 3 präsentiert die Regeln und Herausforderungen der Domäne Pommerman im Detail.
- Kapitel 4 liefert eine Zusammenfassung bisheriger Forschungen zu den Themen Deep Q-Learning, emergente Kommunikation und Pommerman.
- Kapitel 5 beschreibt die Architektur des Modells, Trainings- und Testsetup sowie angewandte Metriken zur Analyse der Kommunikation.
- In Kapitel 6 werden die Ergebnisse der durchgeführten Experimente präsentiert und evaluiert.
- Abschließend wird in Kapitel 7 die Arbeit zusammengefasst und ein Ausblick über künftige Forschungsmöglichkeiten gegeben.

## 2. Grundlagen

Dieses Kapitel definiert das grundlegende Konzept und begleitende Notationen, die für den restlichen Teil der Arbeit relevant sind. Die Notation orientiert sich mit einigen Anpassungen stark an [SB98], [OA16], [HS15].

### 2.1. Reinforcement Learning

Reinforcement Learning (RL) ist ein Teilbereich des maschinellen Lernens und bedeutet Lernen durch Interaktion, um ein bestimmtes Ziel zu erreichen. Das Ziel wird durch einen sog. *Reward* formuliert. Ein *Agent* (Entscheidungsträger und lernende Instanz) interagiert in diskreten Zeitschritten mit seinem *Environment*, um eine Strategie zu lernen, die die zu erwartende Summe der Rewards maximiert (*reward hypothesis*) [SB98].

#### 2.1.1. Markov Decision Process

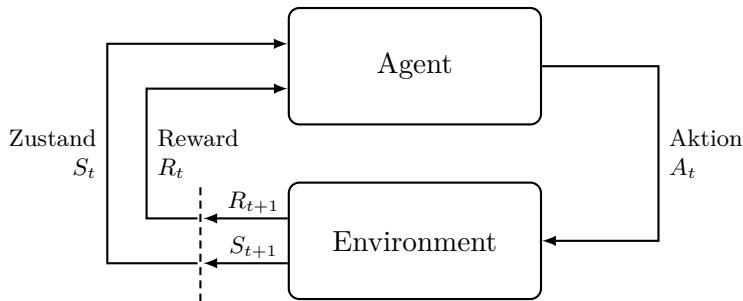


Abbildung 2.1.: Agent-Environment-Interaktion in einem Markov Decision Process (Angelehnt an [SB98, Abb. 3.1])

Das Reinforcement Learning Problem wird mathematisch üblicherweise durch ein *Markow Entscheidungsproblem* oder *Markov Decision Process* (MDP) beschrieben. Formal ist ein MDP ein Quintupel  $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$  [SB98], [RN09], mit

- der Menge der Zustände  $\mathcal{S}$ ,
- der Menge der Aktionen  $\mathcal{A}$ ,
- der Transitionsfunktion oder dem Modell  $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ ,
- der Rewardfunktion  $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ ,
- dem Discountfaktor  $\gamma \in [0, 1]$ .

Abb. 2.1 stellt das Zusammenspiel zwischen Agent und Environment in einem MDP graphisch dar. Zu jedem Zeitpunkt  $t$  beobachtet der Agent den Zustand  $S_t \in \mathcal{S}$  des En-

## 2. Grundlagen

vironments, auf dessen Basis er seine nächste Aktion  $A_t \in \mathcal{A}$ <sup>1</sup> auswählt. Als Konsequenz seiner Aktion beobachtet der Agent im nächsten Zeitschritt einen neuen Zustand  $S_{t+1} \in \mathcal{S}$  des Environments und einen Reward  $R_{t+1} \in \mathbb{R}$ , ermittelt durch die Rewardfunktion  $r$ .  $r(s, a)$  gibt den zu erwartenden unmittelbaren Reward in Form einer reellen Zahl an, wenn Aktion  $a$  in Zustand  $s$  ausgeführt wird. Es entsteht eine Sequenz aus Zuständen, Aktionen und Rewards, die *Trajektorie* genannt wird [SB98]:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots, S_{T-1}, A_{T-1}, R_T, S_T \quad (2.1)$$

$T$  beschreibt hier den finalen Zeitpunkt.

Die Transitionsfunktion  $p(s, a, s')$  liefert die Wahrscheinlichkeit, mit der Zustand  $s'$  ausgehend von Zustand  $s$  durch Aktion  $a$  erreicht wird. Alternativ lässt sich auch  $p(s'|s, a)$  schreiben. Damit ein System als MDP modelliert werden kann, muss es die *Markov Eigenschaft* erfüllen. Das bedeutet, dass die Wahrscheinlichkeit  $p$  nur vom aktuellen Zustand und nicht von der Historie vorheriger Zustände abhängig ist. Der Zustand muss dafür alle nötigen Informationen über vergangene Agent-Environment Interaktionen repräsentieren, die relevant für die Zukunft sind [SB98].

Die kumulative Summe aller Rewards innerhalb einer Trajektorie nennt man *Gewinn*:

$$G = R_1 + R_2 + R_3 + \dots + R_T \quad (2.2)$$

Diesen gilt es wie erwähnt zu maximieren. Ist  $T = \infty$ , wird zur Berechnung des Gewinns die diskontierte Summe der Rewards verwendet:

$$G = R_1 + \gamma R_2 + \gamma^2 R_3 + \dots = \sum_{t=1}^T \gamma^{t-1} R_t, \quad (2.3)$$

anderenfalls wäre der Gewinn allgemein auch unendlich. Der Discountfaktor  $\gamma \in [0, 1]$  bestimmt den Wert zukünftiger Rewards zum gegenwärtigen Zeitpunkt. Je näher  $\gamma$  an 1 ist, desto stärker werden zukünftige Rewards gewichtet.

Der zu erwartende Gewinn eines Agenten ist abhängig von seinen Aktionen. Eine Strategie oder *Policy* ist eine Funktion  $\pi : \mathcal{S} \rightarrow \mathcal{A}$ , die jedem Zustand  $s$  eine Aktion  $a$  zuordnet und dadurch das Verhalten eines Agenten bestimmt [KLC98]. Der Wert  $v_\pi(s)$  eines Zustands  $s$  unter einer Policy  $\pi$  ist der zu erwartende Gewinn eines Agenten, der von  $s$  ausgehend  $\pi$  folgt.  $v_\pi(s)$  wird *State-Value Function* einer Policy  $\pi$  genannt und ist formal wie folgt definiert:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right] \quad (2.4)$$

Nach allgemeinem Verständnis gibt die Value Function an, wie hilfreich ein Zustand für einen Agenten ist. Der Wert eines Endzustands ist immer 0.

Ähnlich dazu wird der Wert  $q_\pi(s, a)$  eines Zustands  $s$  und einer Aktion  $a$  wie folgt definiert:

---

<sup>1</sup>Es ist möglich, dass der Aktionsraum vom Zustand abhängig ist, man schreibt dann  $\mathcal{A}(s)$ . Zur Vereinfachung wird in dieser Arbeit angenommen, dass in allen Zuständen alle Aktionen erlaubt sind:  $\forall s, s' \in \mathcal{S} : \mathcal{A}(s) \equiv \mathcal{A}(s') \equiv \mathcal{A}$ .

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right] \quad (2.5)$$

$q_\pi(s, a)$  wird *Action-Value Function* (oder auch Q-Value) einer Policy  $\pi$  genannt und berechnet den zu erwartenden Gewinn eines Agenten, der ausgehend von Zustand  $s$  Aktion  $a$  ausführt und anschließend Policy  $\pi$  folgt. Die Action-Value Function misst vereinfacht gesagt, wie gut eine Aktion in einem bestimmten Zustand ist.  $v_\pi(s)$  lässt sich durch  $q_\pi(s, a)$  ausdrücken [WSH<sup>+</sup>16]:

$$v_\pi(s) = \mathbb{E}_{a \sim \pi(s)}[q_\pi(s, a)] \quad (2.6)$$

Zusätzlich gilt folgender Zusammenhang:

$$q_\pi(s, a) = v_\pi(s) + A_\pi(s, a) \quad (2.7)$$

$A_\pi(s, a)$  heißt *Advantage Function* und dient als ein relatives Maß für die Bedeutung einzelner Aktionen; im Grunde genommen bestimmt  $A_\pi(s, a)$ , wie viel besser eine Aktion  $a$  in Zustand  $s$  im Vergleich zu allen anderen Aktionen ist. Aus (2.6) und (2.7) folgt:

$$\mathbb{E}_{a \sim \pi(s)}[A_\pi(s, a)] = 0 \quad (2.8)$$

Eine Policy  $\pi$  ist mindestens genauso gut wie eine Policy  $\pi'$ , wenn sie den gleichen oder einen höheren Gewinn für jeden Zustand erzielt. Formal bedeutet das:

$$\pi \geq \pi' \Leftrightarrow \forall s \in \mathcal{S} : v_\pi(s) \geq v_{\pi'}(s) \quad (2.9)$$

Eine Policy, die mindestens genauso gut wie alle anderen Policies ist, heißt *optimale Policy*  $\pi_*$ . State- und Action-Value Function einer optimalen Policy nennt man *optimale State-Value Function*  $v_*(s)$  und *optimale Action-Value Function*  $q_*(s, a)$ . Beide Funktionen erfüllen die *Bellman Optimality Equation*:

$$\begin{aligned} v_*(s) &= \max_{\pi} v_{\pi}(s) \\ &= \max_{a \in \mathcal{A}} q_*(s, a) \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \end{aligned} \quad (2.10)$$

$$= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_*(s')] \quad (2.11)$$

$$\begin{aligned} q_*(s, a) &= \max_{\pi} q_{\pi}(s, a) \\ &= \mathbb{E} \left[ R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \middle| S_t = s, A_t = a \right] \end{aligned} \quad (2.12)$$

$$= \sum_{s', r} p(s', r | s, a) \left[ r + \gamma \max_{a'} q_*(s', a') \right] \quad (2.13)$$

Ein MDP gilt als gelöst, wenn man eine optimale Policy gefunden hat.  $\pi_*$  lässt sich direkt

## 2. Grundlagen

aus  $q_*$  konstruieren:

$$\pi_*(s) = \operatorname{argmax}_{a \in \mathcal{A}} q_*(s, a) \quad (2.14)$$

Das Lösen der Bellman Optimality Equation wäre demnach ein Weg, eine optimale Policy zu finden. In der Realität ist dieser Ansatz in den meisten Fällen aufgrund fehlender Rechenleistung allerdings nicht umsetzbar; für das Spiel „Backgammon“ mit  $10^{20}$  Zuständen bräuchte man aktuell mehrere tausend Jahre, um die Bellman Equation zu lösen. Im Reinforcement Learning beruft man sich deswegen auf Strategien zur Approximation, mehr dazu in Unterabschnitt 2.3.2.

### 2.1.2. Partiell beobachtbare MDPs

In der Realität kann es vorkommen, dass der Zustand des Environments für einen Agenten nur teilweise sichtbar ist, man sagt das Environment ist *partiell beobachtbar* [OA16], [RN09]. Partiell beobachtbare Domänen werden durch ein *Partially Observable Markov Decision Process* (POMDP) repräsentiert, eine Erweiterung des MDPs. Formal ist ein POMDP ein 8-Tupel  $(\mathcal{S}, \mathcal{A}, \mathcal{O}, p, r, z, b_0, \gamma)$  [SB98], mit

- der Menge der Observationen  $\mathcal{O}$ ,
- der Observationsfunktion  $z : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \rightarrow [0, 1]$ ,
- dem initialen Belief State  $b_0 \in \mathcal{B}$ .

$\mathcal{S}$ ,  $\mathcal{A}$ ,  $p$ ,  $r$  und  $\gamma$  sind genauso wie in einem MDP definiert (Unterabschnitt 2.1.1).

In einem POMDP wird angenommen, dass sich das Environment zu jedem Zeitpunkt in einem wohldefinierten, aber versteckten Zustand befindet, der für einen Agenten i.d.R. nicht vollständig sichtbar ist. Der Agent erhält eine Observation, die den zugrundeliegenden Zustand des Environments nicht eindeutig repräsentiert [Spa12]. Die Observationsfunktion  $z(o|s, a)$  liefert die Wahrscheinlichkeit, mit der Observation  $o$  ausgehend von Zustand  $s$  durch Aktion  $a$  erreicht wird. Die Sequenz aus Aktionen, Observationen und Rewards wird *Historie* genannt:

$$H_t = A_0, O_1, R_1, \dots, A_{t-1}, O_t, R_t \quad (2.15)$$

Ein Agent kann seine Historie nutzen, um einen sog. *Belief State* abzuleiten. Ein Belief State  $b_h : \mathcal{S} \rightarrow [0, 1]$  für eine gegebene Historie  $h$  ist eine Wahrscheinlichkeitsverteilung über alle möglichen, versteckten Zustände  $\mathcal{S}$  eines Environments, die die Markov Eigenschaft erfüllt [KLC98]. Sie dient Agenten als Entscheidungsgrundlage für ihre nächsten Aktionen [SB98], [OA16].  $\mathcal{H}$  repräsentiert die Menge der Historien und  $\mathcal{B}$  die Menge der Belief States. Eine Policy in einem POMDP hat die Form  $\pi : \mathcal{B} \rightarrow \mathcal{A}$ .

Für weitere Informationen zu POMDPs im Allgemeinen und wie optimale Value Functions in POMDPs berechnet werden können verweise ich auf [KLC98] und [Spa12].

### 2.1.3. Kooperative Multi-Agenten Systeme

Bisherige Definitionen sind nur auf Situationen anwendbar, in denen das Environment nur durch einen einzigen Agenten beeinflusst wird, sog. *Einzel-Agenten Systeme*. Eine Domäne, in der mehrere Agenten gleichzeitig Einfluss auf das (üblicherweise partiell beobachtbare) Environment haben, nennt man *Multi-Agenten System* (MAS). Verfolgen die

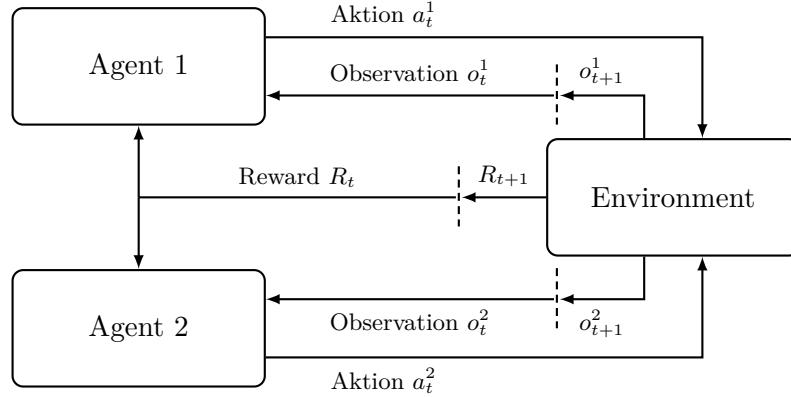


Abbildung 2.2.: Agenten-Environment-Interaktion in einem Dec-POMDP

Agenten ein gemeinsames Ziel, ist das MAS kooperativ. Unabhängig davon können die Agenten identisch (homogen) sein oder sich in ihrer Art und ihren Fähigkeiten unterscheiden (heterogen). MAS ermöglichen zudem Kommunikation zwischen den Agenten, vorausgesetzt es gibt diesbezüglich keine Einschränkungen durch das Environment. Zur Modellierung eines kooperativen Multi-Agenten Systems in einem partiell beobachtbaren Environment verwendet man ein *Decentralized POMDP* (Dec-POMDP). Ein Dec-POMDP ist formal ein 9-Tupel  $(\mathcal{D}, \mathcal{S}, \mathcal{A}, \mathcal{O}, r, p, z, b_0, \gamma)$  [OA16], [BGIZ02], mit:

- der Menge der Agenten  $\mathcal{D} = \{1, \dots, n\}$ ,
- der Menge der gemeinsamen Aktionen  $\mathcal{A} = \times_{d \in \mathcal{D}} \mathcal{A}_d$ ,
- der Menge der gemeinsamen Observationen  $\mathcal{O} = \times_{d \in \mathcal{D}} \mathcal{O}_d$ ,
- der initialen Wahrscheinlichkeitsverteilung  $b_0$  der Zustände  $\mathcal{S}$ .

$\mathcal{S}$ ,  $p$ ,  $r$ ,  $z$  und  $\gamma$  sind wie in den Unterabschnitten 2.1.1 und 2.1.2 definiert.  $\mathcal{O}_d$  ist die Menge der möglichen Observationen eines Agenten  $d \in \mathcal{D}$ .  $\mathcal{A}_d$  ist die Menge der Aktionen, die dem Agenten  $d$  zur Verfügung steht. Abbildung 2.2 zeigt die Interaktion zwischen den Agenten und dem Environment in einem Dec-POMDP. Zu jedem Zeitpunkt  $t$  sendet das Environment eine gemeinsame Observation  $O_t = \langle o_t^1, \dots, o_t^n \rangle \in \mathcal{O}$ , von der jeder Agent  $d$  nur seine individuelle Observation  $o_t^d \in \mathcal{O}_d$  erfährt. Anschließend wählt jeder Agent  $d$  eine Aktion  $a_t^d \in \mathcal{A}_d$  aus, die zusammen eine gemeinsame Aktion  $A_t = \langle a_t^1, \dots, a_t^n \rangle \in \mathcal{A}$  bilden [Bou96].<sup>2</sup> Jeder Agent kennt nur seine eigene Aktion; die Aktionen der anderen Agenten sind ungewiss. Die Rewardfunktion  $r$  liefert auf Basis von  $A_t$  einen gemeinsamen Reward  $R_{t+1}$ , der für alle Agenten gleich ist.

Analog zu partiell beobachtbaren Einzel-Agenten Systemen nutzt jeder Agent  $d$  seine Historie  $h^d \in \mathcal{H}_d$ , um ausgehend von  $b_0$  einen Belief State  $b_{h^d} \in \mathcal{B}_d$  herzuleiten. Das Tupel  $\bar{b} = (b_{h^1}, \dots, b_{h^n})$  setzt sich aus den Belief States der Agenten zusammen [SLG<sup>+</sup>17]. Das Ziel der Agenten ist, eine gemeinsame optimale Policy  $\pi : \mathcal{B}^n \rightarrow \mathcal{A}^n$  zu finden, die den Reward maximiert.

Dec-POMDPs erlauben *explizite Kommunikation* [OA16], wenn den Agenten eine separate Menge an Nachrichten zur Kommunikation untereinander zur Verfügung steht. Das Modell wird durch den Kommunikationskanal um ein Alphabet  $\Sigma$  erweitert. Es entsteht

<sup>2</sup>Im Allgemeinen repräsentiert  $A_t$  die gemeinsame Aktion aller Agenten zum Zeitpunkt  $t$ , während  $a^d$  für die Aktion von Agent  $d$  (ohne eines festgelegten Zeitpunkts  $t$ ) verwendet wird.

## 2. Grundlagen

ein *Dec-POMDP-Comm*.  $\Sigma$  enthält alle möglichen Wörter, die die Agenten verschicken können. Die Länge der Nachricht (i.e. die Anzahl der Wörter in einer Nachricht) ist von der Domäne vorgegeben und kann fest oder variabel sein. Eine Kostenfunktion  $C_\Sigma$  bestimmt die Kosten einer Nachricht. Die Agenten befinden sich in einem *Cheap Talk Setting*, wenn Kommunikation kostenlos ist [FR96].

Kommunikation kann unidirektional oder bidirektional ablaufen. Nachrichten von anderen Spielern sind Teil der Observation eines Agenten. Das Dec-POMDP-Comm macht keine Aussagen darüber, ob Kommunikation simultan oder sequentiell zu den anderen Aktionen abläuft. Weitere Details zu Dec-POMDPs und Dec-POMDP-Comms sind in [OA16] zu finden.

### 2.1.4. Nicht-kooperative Multi-Agenten Systeme

Das Dec-POMDP eignet sich nur für die Modellierung von MAS, in denen Agenten kooperativ interagieren, da nur ein einziger (Team) Reward spezifiziert wird. Nicht-kooperative MAS werden stattdessen als *Partially Observable Stochastic Game* (POSG) modelliert [OA16]. POSGs sind Teil der Spieltheorie und spezifizieren, anders als Dec-POMDPs, eine Menge von Rewardfunktionen (eine für jeden Agenten). Die restlichen Komponenten entsprechen denen in Dec-POMDPs. Das Ziel jedes Agenten besteht darin, seinen eigenen individuellen Reward zu maximieren; die Agenten handeln selbst-interessiert. Optimalität ist nicht mehr festgelegt, als Konsequenz existiert keine gemeinsame optimale Policy mehr. Der Begriff der optimalen gemeinsamen Policy wird durch den des *Nash-Gleichgewichts* (NE) ersetzt. Als NE bezeichnet man einen Zustand, in dem es für keinen Spieler von Vorteil ist, seine Strategie als einziger (unilateral) zu ändern [BS10]. Ein NE ist *pareto optimal*, wenn kein anderes Spielende existiert, das alle Spieler präferieren würden [RN09]. Strategien zum Lösen eines MDPs sind nicht direkt auf POSGs anwendbar.

## 2.2. Künstliche neuronale Netze

*Künstliche neuronale Netze* (KNNs) sind mathematische Modelle zur Informationsverarbeitung, deren Funktionalität echten neuronalen Netzen des menschlichen Nervensystems nachempfunden ist [CHK95], [Gur97] [SKP97]. Sie sind eine Form der künstlichen Intelligenz und damit zentraler Bestandteil maschinellen Lernens [ON15]. Durch KNNs lassen sich beliebige funktionale Beziehungen realisieren und approximieren, wodurch sie vielseitig einsetzbar sind [CHK95], [SKP97].

Es existieren verschiedene Arten von KNNs, aber die grundlegenden Prinzipien sind gleich: Ein KNN ist eine vernetzte Anordnung vieler datenverarbeitender Elemente (*Neuronen*), die, analog zu synaptischen Verbindungen im Nervensystem, über sog. *Gewichte* miteinander verbunden sind. Neuronen werden in Schichten unterteilt, sog. „Layer“. Ein KNN besteht grundsätzlich aus einem Input- und einem Output-Layer und optional mehreren Hidden-Layern (vgl. Abbildung 2.3). Über das Input-Layer werden Signale<sup>3</sup> empfangen (üblicherweise in der Form eines mehrdimensionalen Vektors), die über die gewichteten Verbindungen an die nachfolgenden Layer weitergeleitet werden. Die *Aktivierungsfunktion* eines Neurons bestimmt anhand der gewichteten Summe aller eingehen-

---

<sup>3</sup>Im Kontext von RL stellen die Observationen des Agenten die Signale dar.

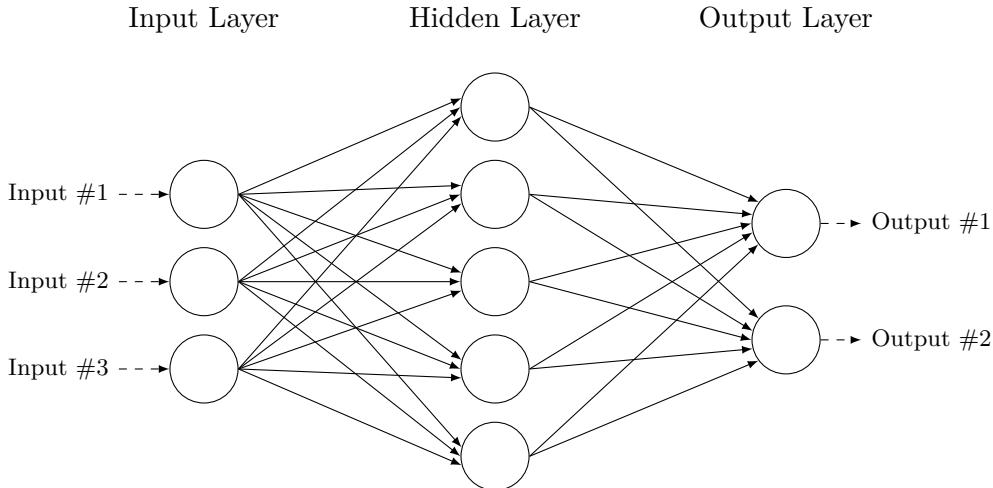


Abbildung 2.3.: Ein einfaches Multi-Layer Perceptron mit einem Input Layer, einem Hidden Layer und einem Output Layer. Jeder Kreis repräsentiert ein einzelnes Neuron, durchgehende Pfeile repräsentieren gewichtete Verbindungen. Gestrichelte Pfeile stehen für Ein- bzw. Ausgaben.

den Signale (der „Aktivierung“ des Neurons) dessen Ausgabe. Im einfachsten Fall ist die Ausgabe 1, wenn die Aktivierung einen Grenzwert überschreitet, ansonsten 0. Neuronen, die diese Aktivierungsfunktion nutzen, nennt man *Threshold Logic Unit* (TLU).

Das „Wissen“ eines KNNs ist in seinen Gewichten gespeichert und dadurch über das gesamte Netz verteilt (anders als bei klassischen Computerprogrammen). *Lernen* im Zusammenhang mit KNNs bedeutet das Anpassen der Gewichte zwischen den Neuronen anhand eines Inputs; das KNN wird *trainiert*. *Deep Learning* beschreibt eine Methode, die KNNs mit mehreren Hidden Layern einsetzt (sog. *Deep Neural Networks* (DNNs)) [ON15]. *Deep Reinforcement Learning* kombiniert RL mit DNNs.

Wie die Gewichte während des Trainings optimiert werden, hängt vom verwendeten Lernalgorithmus ab. Das gängigste Verfahren heißt *Stochastic Gradient Descent* (SGD) [Rud16] [Bot12], [BP92]. SGD ist im Allgemeinen ein Algorithmus zum Lösen von Optimierungsproblemen. Im Kontext künstlicher neuronaler Netze bestimmt SGD die Gewichte des KNNs so, dass sie eine *Loss-Funktion*  $\mathcal{L}$  minimieren. Abbildung 2.4 visualisiert das grundlegende Prinzip des Verfahrens. Zu Beginn des Trainings werden die Gewichte willkürlich initiali-

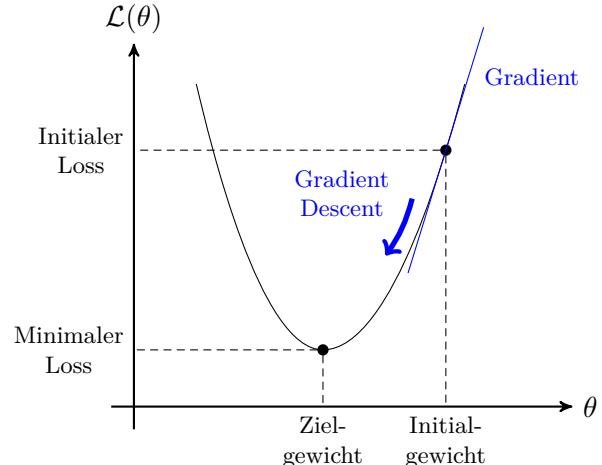


Abbildung 2.4.: Schematische Darstellung von Stochastic Gradient Descent für eine Loss-Funktion  $\mathcal{L}$ , die von einer Variable (Gewicht  $\theta$ ) abhängt (Angelehnt an [Gur97, Abb. 5.4])

9

## 2. Grundlagen

siert [SKP97], [CHK95]. Das KNN aktualisiert dann anhand der Signale aus Trainingsschritt  $i$  die Gewichte (gekennzeichnet durch  $\theta$ ) entlang einer Abstiegsrichtung, bis keine Verbesserung mehr erzielt wird:

$$\theta_{i+1} = \theta_i + \alpha \nabla_{\theta} \mathcal{L}(\theta_i) \quad (2.16)$$

$\alpha \in (0, 1]$  wird *Lernrate* oder *Step Size* genannt und bestimmt die Geschwindigkeit des Lernprozesses [Rud16].  $\nabla_{\theta} \mathcal{L}$  ist der Gradient der Loss-Funktion und stellt die Abstiegsrichtung dar. Die Loss-Funktion hängt von der Strategie ab, die zum Lösen des RL Problems verwendet wird. Zur Berechnung der Gradienten wird ein Verfahren eingesetzt, das *Back Propagation* (BP) genannt wird. In [Cil10] wird der BP Algorithmus genau beschrieben.

Die Loss-Funktion muss für BP kontinuierlich und differenzierbar sein. TLUs produzieren nicht-differenzierbare Gewichte und sind somit für dieses Verfahren nicht geeignet [BP92]. Standardmäßig werden *Rectified Linear Units* (ReLUs) eingesetzt, deren Aktivierungsfunktion  $\sigma(x) : \max(0, x)$  ist [RZL17].

In dieser Arbeit wird eine Variante des SGD-Verfahrens namens *Adam* [KB14], [Rud16] verwendet. Adam ist ein adaptiver Lernalgorithmus, der für verschiedene Gewichte individuelle Lernraten berechnet. Der Algorithmus ist dafür bekannt, selbst bei großen KNNs mit einer hohen Anzahl an Hidden Layern und Neuronen schnell gute Ergebnisse zu erzielen. Eine ausführliche Beschreibung und weitere Details zu Adam können in [KB14] nachgelesen werden.

Folgende Arten künstlicher neuronaler Netze sind für diese Arbeit relevant:

**Multi-Layer Perceptron** MLPs [SKP97] bestehen aus mehreren 1-dimensionalen Layern. Jedes Neuron eines Layers ist mit jedem Neuron des nachfolgenden Layers verbunden (vgl. Abbildung 2.3); die Layer sind „fully connected“. MLPs sind klassische *feedforward* KNNs. Die Signale wandern nur in eine Richtung (*forward*). Es gibt keine Zyklen. Sie sind die simpelste Form von KNNs, aber eine der am weitest verbreitetsten.

**Long Short-Term Memory** LSTMs [HS97] sind sog. *rekurrente* KNNs (RNNs), die zur Verarbeitung sequentieller Datensätze eingesetzt werden. Anders als in klassischen feedforward Architekturen, können Neuronen in RNNs auch Verbindungen zu Neuronen aus demselben oder vorherigen Layern haben (vgl. Abbildung 2.5). Neuronen mit rekurrenten Verbindungen erhalten als Input nicht nur Informationen aus dem Datensatz zum Zeitpunkt  $t$ , sondern auch aus vorherigen Zeitschritten. Diese Information wird *Hidden State*  $h_{t-1}$  genannt und ist eine Repräsentation vorheriger Inputs. Informationen aus vorher gesehenen Signalen bleiben dadurch erhalten (man sagt das RNN hat ein Gedächtnis). Lernen erfolgt nicht nur auf dem Signal des aktuellen Trainingsschritts, sondern auch auf den Daten vorheriger Schritte [LBE15]. Dadurch sind RNNs in der Lage, zeitlich codierte Informationen in sequentiellen Daten zu erkennen.

RNNs leiden unter „Short-Term Memory“: Informationen zu Beginn langer Sequenzen werden „vergessen“. Das Problem resultiert aus „Vanishing Gradients“, die während der Back Propagation entstehen [HS97]. Vanishing Gradients sind Gradienten, deren Wert verschwindend gering wird, sodass sie keinen Einfluss mehr auf die Ge-

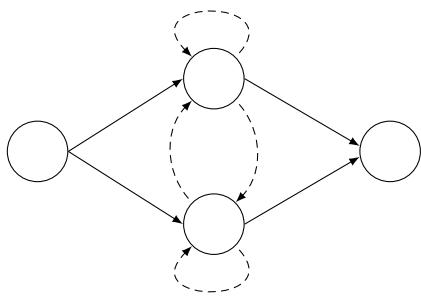


Abbildung 2.5: Ein einfaches rekurrentes neuronales Netz (anglehnt an [LBE15, Abb. 3]). Zum Zeitpunkt  $t$  werden Signale entlang durchgehender Pfeile übertragen. Gestrichelte Pfeile repräsentieren rekurrente Verbindungen, die den Hidden State  $h_t$  im nächsten Zeitschritt  $t + 1$  weiterleiten.

wichte des RNNs ausüben. LSTMs lösen das Problem, indem Neuronen der Hidden Layer durch sog. „Memory Cells“ ersetzt werden. Memory Cells können lernen, welche Informationen zum Hidden State hinzugefügt oder entfernt werden sollen. Als Aktivierungsfunktion wird standardmäßig der *Tangens hyperbolicus* ( $\tanh$ ) genutzt:  $\tanh x = 1 - \frac{2}{e^{2x}+1}$ . Für weitere Informationen zu RNNs und LSTMs verweise ich auf [HS97], [LBE15] und [She20].

**Convolutional NN** CNNs [ON15] werden hauptsächlich zur Verarbeitung von Bilddaten verwendet, insbesondere im Bereich der Mustererkennung und Objektklassifizierung. CNNs bestehen aus mehreren 3-dimensionalen Layern (sog. „Convolutional Layer“ (ConvLayer)), gefolgt von einem oder mehreren fully-connected Layern. Die Anzahl der Neuronen im letzten fully-connected Layer entsprechen üblicherweise der Anzahl an Objektklassen. Convolutional Layer extrahieren relevante Merkmale, um die Bilddatei auf eine Größe zu reduzieren, die für die nachfolgenden Layer leichter zu verarbeiten ist. Neuronen zweier aufeinanderfolgender Convolutional Layer sind nur über eine kleine Region miteinander verbunden (vgl. Abbildung 2.6). Eine Faltungsmatrix wird schrittweise über die Eingabe (oder den Output der Neuronen des vorherigen Layers) bewegt, um durch diskrete Konvolution den Input jedes Neurons des nachfolgenden Layers zu berechnen. Die Anzahl der verwendeten Faltungsmatrizen bestimmt die Tiefe eines Convolutional Layers. Die Größe der Faltungsmatrizen innerhalb eines Layers ist immer gleich. Die Schrittgröße gibt an, wie weit eine Faltungsmatrix nach jeder Operation verschoben wird. Padding ist eine Methode, die Dimensionen des Ergebnisses der Konvolution zu kontrollieren. CNNs können bei Bedarf mit LSTMs kombiniert werden.

## 2.3. Temporal-Difference Learning

Wie erwähnt, ist die Lösung eines MDPs eine optimale Policy. Man unterscheidet zwei Kategorien von Methoden zum Generieren optimaler Policies: In *modellfreien* Methoden lernen Agenten direkt aus ihren Erfahrungen durch „Trial and Error“ Verfahren [SB98]. Es wird kein Modell benötigt, das die Dynamiken innerhalb des Environments beschreibt. In *modellbasierten* Methoden generieren Agenten auf Basis ihrer Erfahrungen ein solches Modell, das anschließend genutzt wird, um die Konsequenz ihrer Aktionen abzuleiten und dadurch eine optimale Policy zu generieren.

*Temporal-Difference* Learning (TD) [SB98] beschreibt eine Menge modellfreier Lernmethoden, in denen bisher gelernte Näherungen zur eigenen Überarbeitung genutzt werden

## 2. Grundlagen

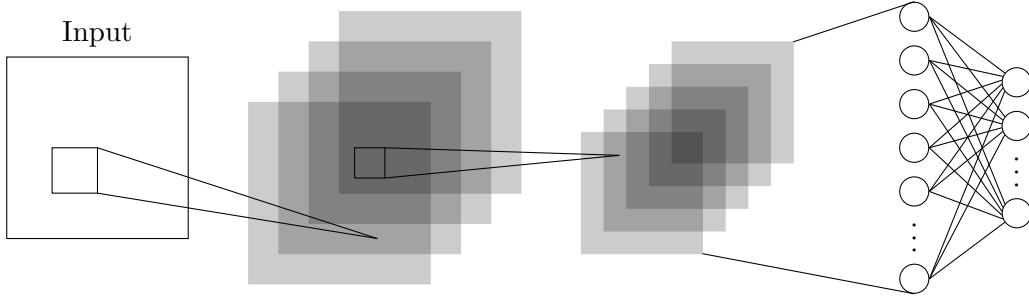


Abbildung 2.6.: Aufbau eines klassischen Convolutional Neural Networks mit zwei Convolutional und zwei fully-connected Layern. Das erste Convolutional Layer verwendet vier Filterkernel. Padding ist so gewählt, dass die Dimensionen des Inputs nicht reduziert werden. Das zweite Convolutional Layer hat fünf Filterkernel, die Größe des vorherigen Layers wird verringert; auf Padding wurde verzichtet oder es ist zu gering, um die Dimensionen zu erhalten. Dimension und Schrittgröße der Filtermatrizen sind in beiden ConvLayern unbekannt.

(„Bootstrapping“). TD Methoden können in *On-Policy* und *Off-Policy* Methoden unterteilt werden. In *On-Policy* Methoden approximieren Agenten eine optimale Policy anhand der Policy, die ihnen auch als Entscheidungsgrundlage für ihre nächsten Aktionen dient. In *Off-Policy* Methoden nutzen Agenten zur Approximation eine Policy, die nicht parallel als Entscheidungsgrundlage verwendet wird.

### 2.3.1. Q-Learning

Q-Learning (einer der ersten Durchbrüche im RL) ist ein Off-Policy TD Control Algorithmus, der die Action-Value Function (Q-Values) einer optimalen Policy approximiert [SB98], [WD92], [RB19]. Die Idee dahinter ist, den geschätzten Wert der Q-Values iterativ an den wahren Wert  $Q_*$  (berechnet durch die Bellman Equation (2.13)) anzupassen. Dazu wird der gewichtete Durchschnitt des alten Q-Values und der neuen Information (i.e. der Gewinn) verwendet:

$$\begin{aligned} Q_{i+1}(S_t, A_t) &\leftarrow (1 - \alpha) Q_i(S_t, A_t) + \alpha G_t \\ &\leftarrow Q_i(S_t, A_t) + \alpha (G_t - Q_i(S_t, A_t)) \\ &\leftarrow Q_i(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q_i(S_{t+1}, a) - Q_i(S_t, A_t) \right] \end{aligned} \quad (2.17)$$

$Q_i$  konvergiert mit einer Wahrscheinlichkeit 1 zu  $Q_*$  für  $i \rightarrow \infty$  [MKS<sup>+</sup>13].

Die aktuelle Näherung  $Q_i$  wird genutzt, um eine Policy abzuleiten, die die nächste Aktion  $A_t$  vorgibt (vgl. (2.14)).

$$A_t = \operatorname{argmax}_{a \in \mathcal{A}} Q_i(S_t, a) \quad (2.18)$$

Es wird die Aktion ausgewählt, die nach aktuellem Wissensstand/aktueller Policy den größten zu erwartenden Gewinn liefert („Exploitation“/Ausbeuten des aktuellen Wissens). Der Agent erhält nach Ausführen der Aktion im nächsten Schritt einen Reward

$R_{t+1}$  und den neuen Zustand  $S_{t+1}$ . Der Q-Value kann anschließend unter der Annahme, dass der Agent weiterhin derselben *greedy* Policy folgt, aktualisiert werden (vgl. (2.17)). Die neue Approximation  $Q_{i+1}$  wird verwendet, um im nächsten Schritt  $A_{t+1}$  abzuleiten. Anders als beim Update angenommen, folgt der Agent nicht derselben (*greedy*) Policy. Die optimale Action-Value Function  $Q_*$  wird dadurch in jedem Schritt direkt durch die (momentan) gelernte Action-Value Function  $Q$  approximiert, unabhängig von der tatsächlich im nächsten Schritt verwendeten (neu approximierten) Policy (i.e. Off-Policy).

Wählt der Agent in (2.18) keine der *greedy* Aktionen, spricht man von „Exploration“/Erkunden. Exploitation (das Auswählen der *greedy* Aktion) ist eine sinnvolle Strategie, um den Gewinn innerhalb eines Schritts zu maximieren. Exploration ermöglicht aber die Näherung der non-*greedy* Aktionen zu verbessern und kann auf lange Sicht eine bessere Wahl finden [SB98]. Ein *Explorations-Exploitations-Dilemma* (EE-Dilemma) entsteht.  $\epsilon$ -*greedy* Methoden sind eine simple Alternative, um das EE-Dilemma zu lösen. Der Agent wählt mit einer geringen Wahrscheinlichkeit  $\epsilon$  zufällig eine der möglichen Aktion aus, unabhängig von deren aktuellen Q-Value Approximation. Mit einer Wahrscheinlichkeit von  $\epsilon - 1$  werden *greedy* Aktionen ausgewählt. Dadurch wird sichergestellt, dass die Q-Values aller Aktionen optimiert werden. In der Praxis ist es üblich, mit  $\epsilon = 1$  zu starten und  $\epsilon$  schrittweise zu verringern [LC16].

### 2.3.2. Deep Q-Learning

In der Realität ist es aufgrund sehr großer Zustands- und Aktionsräume nicht umsetzbar, den Q-Value aller Zustands- und Aktionspaare  $(S, A)$  zu berechnen. Zur Umsetzung fehlt ausreichend Rechenleistung. Stattdessen kann die Action-Value Function durch ein Modell approximiert werden [MKS<sup>+</sup>13], [CYL16]:

$$Q_\theta(s, a) \approx Q_*(s, a) \quad (2.19)$$

Im RL wird für ein solches Modell üblicherweise ein KNN verwendet (vgl. Abschnitt 2.2).

*Deep Q-Learning* nutzt Deep Learning (KNNs mit mehreren Hidden Layern) zur Approximation von  $Q_*$ . Das verwendete Netz wird „(Deep) Q-Network“ (DQN) genannt. Das Ziel ist es, die Gewichte  $\theta$  des DQNs in jedem Trainingsschritt  $i$  so zu optimieren, bis (2.19) erfüllt ist. Folgende Loss-Funktion wird verwendet [HS15]:

$$\mathcal{L}(s, a | \theta_i) \approx \left[ r + \gamma \max_a Q_{\theta_i}(s', a) - Q_{\theta_i}(s, a) \right]^2 \quad (2.20)$$

$r + \gamma \max_a Q_{\theta_i}(s', a)$  wird *Target y* genannt [RB19]. Durch das DQN ist es nicht mehr nötig, alle Q-Values einzeln zu berechnen.

Mit (2.20) als Loss-Funktion können Divergenzen und Instabilitäten auftreten. Verschiedene Strategien helfen, den Lernprozess zu stabilisieren [HS15], [LC16], [RB19]:

**Experience Replay** Der Agent speichert in jedem Zeitschritt seine Erfahrungen als Tupel  $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$ <sup>4</sup> in einem „Replay Memory“  $\mathcal{D}$ . Die Q-Updates erfolgen

---

<sup>4</sup>bzw.  $e_t = (o_t, a_t, r_{t+1}, o_{t+1})$  für partiell beobachtbare Environments.

## 2. Grundlagen

auf einer Stichprobe bestehend aus mehreren Tupeln, die zu jeder Trainingsphase  $i$  mit gleicher Wahrscheinlichkeit aus  $\mathcal{D}$  entnommen werden. Korrelationen zwischen aufeinanderfolgenden Erfahrungstupeln können dadurch umgangen und bereits Gelehrtes erneut wiederholt werden; selten auftretende und kostspielige<sup>5</sup> Erfahrungen können effektiv wiederverwendet werden [Lin93]. Übliche Stichprobengrößen sind 32 oder 64.

**Fixed Q-Target** löst das Problem, dass sich das Target  $y$  mit jedem Update der Q-Values auch verändert. Um Oszillationen im Training zu vermeiden, wird zur Berechnung von  $y$  ein anderes Netz („Target-Network“  $Q_{\theta^-}$ ) verwendet als das Q-Network  $Q_{\theta_i}$ . Die Gewichte  $\theta^-$  des Target-Networks werden alle  $\tau$  Trainingsschritte zu  $\theta_i$  aktualisiert. In der Zwischenzeit bleiben sie unverändert [MKS<sup>+</sup>15], [RB19]. Alternativ kann man  $\theta^-$  in jedem Trainingsschritt gleich  $\tau\theta_i$  setzen.

**Double DQL** Der max-Operator im Standard Q-Learning und Deep Q-Learning approximiert den maximal zu erwartenden Gewinn durch den höchsten Wert der momentan approximierten Q-Values der Aktionen. Das kann dazu führen, dass Aktionen überschätzt werden. Indem man Aktionsauswahl und -bewertung voneinander entkoppelt, kann das Problem verhindert werden [vH10], [vHGS15]: Das Q-Network  $Q_{\theta_i}$  wird verwendet, um die beste Aktion des nächsten Zustands zu bestimmen (die Aktion mit dem höchsten Q-Value). Das Target-Network  $Q_{\theta^-}$  bestimmt dann den zu erwartenden Gewinn dieser Aktion im nächsten Zustand.

Unter Verwendung oben genannter Strategien entsteht folgende Loss-Funktion:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{U}(\mathcal{D})} \left[ \left( r + \gamma Q_{\theta^-}(s', \operatorname{argmax}_{a'} Q_{\theta_i}(s', a')) - Q_{\theta_i}(s, a) \right)^2 \right] \quad (2.21)$$

Klassisches Deep Q-Learning berechnet für jeden Zustand den Wert der Aktionen, unabhängig davon, wie wertvoll dieser Zustand ist. Das ist besonders für Zustände unnötig, in denen die Aktionen keine relevanten Auswirkungen zur Folge haben.<sup>6</sup> Wang et al. [WSH<sup>+</sup>16] entwickelten dadurch motiviert eine weitere Strategie namens **Dueling Network Architektur**, die auf (2.7) aufbaut. Das DQN wird dafür in zwei „parallele Ströme“ von Layern zerlegt, die jeweils die State-Value Function und die Advantage Function separat voneinander approximieren. Beide Ströme werden anschließend kombiniert, um die Q-Values zu generieren. Für die Back Propagation ist es notwendig, dass  $v(s)$  und  $A(s, a)$  eindeutig aus  $Q(s, a)$  berechnet werden können. Dafür wird (2.7) um eine Bedingung ergänzt: Der Q-Value der gewählten Aktion muss gleich dem Wert des Zustands sein. Zusammen mit (2.18) ergibt das:

$$Q_{\theta, \alpha, \beta}(s, a) = v_{\theta, \beta}(s) + \left[ A_{\theta, \alpha}(s, a) - \max_{a' \in \mathcal{A}} A_{\theta, \alpha}(s, a') \right] \quad (2.22)$$

$\theta$  repräsentiert nach wie vor die Gewichte des ursprünglichen DQNs.  $\alpha$  und  $\beta$  beschreiben die Gewichte der neuen Layer. Aus Gründen der besseren Lesbarkeit wurde auf den Index

---

<sup>5</sup>im Sinne von Schaden involvierend.

<sup>6</sup>In Zuständen, in denen ein negativer Reward unausweichlich ist (e.g. Tod des Agenten), macht es beispielsweise keinen Unterschied, welche Aktion als nächstes gewählt wird.

$i$  der Trainingsphase verzichtet. Nun gilt:

$$A_t = \operatorname{argmax}_{a \in \mathcal{A}} Q_{\theta, \alpha, \beta}(S_t, a) = \operatorname{argmax}_{a \in \mathcal{A}} A_{\theta, \alpha}(S_t, a) \quad (2.23)$$

$$Q_{\theta, \alpha, \beta}(S_t, A_t) = v_{\theta, \alpha}(S_t) \quad (2.24)$$

Der Wert eines Zustands kann bestimmt werden, ohne die Auswirkungen der einzelnen Aktionen berechnen zu müssen. Der Agent lernt schneller und zuverlässiger. Für zusätzliche Stabilität kann der max-Operator in (2.22) durch den Durchschnitt ersetzt werden:

$$Q_{\theta, \alpha, \beta}(s, a) = v_{\theta, \beta}(s) + \left[ A_{\theta, \alpha}(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A_{\theta, \alpha}(s, a') \right] \quad (2.25)$$

### 2.3.3. Deep Recurrent Q-Learning

Vanilla Deep Q-Learning setzt voraus, dass der zugrundeliegende Zustand des Environments für den Agenten vollständig sichtbar ist und ist demnach ohne weitere Anpassungen nicht für POMDPs geeignet [RB19], [HS15], [LC16]:  $Q_{\theta_i}(o, a) \neq Q_{\theta_i}(s, a)$ . Hausknecht et al. [HS15] entwickelten eine Variante des Deep Q-Learnings, in der das DQN um eine rekurrente Schicht erweitert wird. Das resultierende *Deep Recurrent Q-Network* (DRQN) approximiert  $Q_{\theta_i}(o_t, h_{t-1}, a_t)$  an Stelle von  $Q_{\theta_i}(s_t, a_t)$ .  $h_{t-1}$  ist der Hidden State des DRQNs. Im Kontext von Reinforcement Learning entspricht der Hidden State eines RNNs der komprimierten Historie des Agenten. Die Autoren verwenden für die Umsetzung der rekurrenten Schicht ein LSTM. Die (komprimierte) Historie des Agenten wird dann auf Basis der neuen Observation durch  $h_t = LSTM(o_t, h_{t-1})$  aktualisiert und verwendet, um  $Q(h_t, a_t)$  zu berechnen.

### 2.3.4. Deep Q-Learning in kooperativen Multi-Agenten Systemen

Es existieren verschiedene Möglichkeiten Deep Q-Learning in kooperativen Multi-Agenten Systemen anzuwenden. Im Folgenden werden zwei Methoden vorgestellt.

#### Independent Deep (Recurrent) Q-Learning

*Independent Deep (Recurrent) Q-Learning* [Tan93], [FAdFW16], [TMK<sup>+</sup>15] ist die simpelste Adaption des DQL Algorithmus in MAS. Jeder Agent  $d$  des Environments lernt unabhängig von den anderen Agenten seine eigene Action-Value Function  $Q_{\theta_i}^d(o_t^d, h_{t-1}^d, a_t^d)$ . Jedes DRQN wird dezentral trainiert; die Gradienten werden individuell gebildet. Das Multi-Agenten Problem wird als eine Menge paralleler Einzel-Agenten Probleme interpretiert, die sich ein Environment teilen [RSdW<sup>+</sup>18]; alle anderen Spieler sind Teil des Environments. Das hat im Allgemeinen zur Folge, dass das Environment für jeden Agenten nicht-stationär erscheint; andere Agenten ändern während des eigenen Lernprozesses ihr Verhalten, wodurch sich die Dynamiken des Environments verändern. Zudem können Agenten gestörte Rewardsignale erhalten, die durch das (nicht beobachtete) Verhalten ihrer Mitspieler verursacht wurden. Der Reward kann nicht erklärt werden. Das kann zu Konvergenzproblemen führen.

## 2. Grundlagen

### Value Decomposition (Networks)

*Value Decomposition* (VD) [SLG<sup>+</sup>17] ist eine einfache Methode Independent Deep Q-Learning zu erweitern, um bessere Ergebnisse zu erzielen. Die Idee basiert auf der Annahme, dass in MAS die gemeinsame Action-Value Function  $Q_{tot}$  der Agenten in einzelne Komponenten zerlegt werden kann; jede Komponente entspricht der individuellen Action-Value Function  $Q_{\theta_i}^d$  eines Agenten  $d$ :

$$Q_{tot}\left(\left(h^1, \dots, h^n\right), \left(a^1, \dots, a^n\right)\right) \approx \sum_{d=1}^n Q_{\theta_i}^d\left(h^d, a^d\right) \quad (2.26)$$

Jede Action-Value Function gibt an, wie gut die Aktion des Agenten im aktuellen Zustand ist. Damit kann bestimmt werden, wie viel jeder Agent für den gemeinsamen Reward beigetragen hat [VVB<sup>+</sup>20]. Für die Umsetzung von VD werden die Q-Values der ausgewählten Aktionen jedes Agenten addiert.  $Q$  wird in (2.21) durch  $Q_{tot}$  ersetzt [RSdW<sup>+</sup>18]. Bei der Berechnung der Gradienten wird dadurch sichergestellt, dass jeder Agent entsprechend seiner Leistung belohnt wird [VVB<sup>+</sup>20]. Das Ziel ist, eine optimale lineare Zerlegung des gemeinsamen Rewardsignals zu finden.

Es ist möglich, für die Zusammensetzung der Q-Values ein zusätzliches KNN zu verwenden [VVB<sup>+</sup>20]. *Value Decomposition Networks* (VDN) [SLG<sup>+</sup>17] eignen sich für Systeme, in denen der Erfolg eines Agenten vom Erfolg eines anderen Agenten abhängt oder um heterogene Agenten zu trainieren.

### 3. Pomberman



Abbildung 3.1.: Partiell beobachtbares Pomberman Spielfeld, dunkelgrauer Nebel verdeckt die Felder außerhalb der Sicht der Agenten. Hellbraune Holz- und dunkelbraune Steinwände versperren den Weg über hellgraue Passagen.

Pomberman [REH<sup>+</sup>18] ist eine Multi-Agenten Domäne, basierend auf dem Nintendo-Spiel *Bomberman*. Das Spiel involviert vier Agenten, die je nach Spielkonfiguration alle gegeneinander antreten oder Zweierteams bilden. Das Ziel ist, gegnerische Spieler durch Bomben, die jeder Agent legen kann, zu zerstören.

Es existieren drei verschiedene Spielmodi. In der simpelsten Variante „Free for all“ (FFA) treten die Agenten alleine an. Das Spielfeld ist für jeden vollständig sichtbar und Sieger ist derjenige, der am längsten überlebt. In den Team Varianten „Team“ und „Team Radio“ treten die Agenten in Zweierteams gegeneinander an. Das Spielfeld ist für jeden nur teilweise sichtbar, d.h. jeder Agent kann bis zu vier Felder weit in jede Richtung sehen. Das Spiel endet, wenn beide Agenten des gegnerischen Teams zerstört wurden. Kommunikation zwischen Agenten ist lediglich in der „Team Radio“ Konfiguration zulässig, weswegen im Folgenden nicht weiter auf die Varianten FFA und „Team“ eingegangen wird. Wenn von nun an von Pomberman die Rede ist, ist dieser Modus implizit gemeint.

#### 3.1. Motivation

Bislang lag der Fokus der Multi-Agenten Forschung im Reinforcement Learning auf Zwei-Personen-Nullsummenspielen, wie Schach und Go. Resnick et al. [REH<sup>+</sup>18] entwickelten Pomberman, um die Entwicklung im Bereich der Nicht-Nullsummenspiele und des Multi-

### 3. Pommerman

Agenten Learnings mit mehr als zwei Spielern voranzutreiben. Das Fehlen eines qualitativ hochwertigen Benchmarks sei einer der Gründe für die fehlenden Durchbrüche und den vergleichsweise langsamen Fortschritt in diesen Bereichen.

Um die Ziele der Forscher zu unterstützen, werden auf der Konferenz für „Neural Information Processing Systems“ (NeurIPS) jährlich Wettbewerbe zu Pommerman veranstaltet. Die Pommerman FFA und Team Competitions fanden im Mai und November 2018 statt. Im November 2019 wurde die Pommerman Team-Radio Competition ausgetragen. Es konnte gezeigt werden, dass bereits große Fortschritte in der Entwicklung starker Agenten für Pommerman gemacht wurden, diese sind jedoch nicht mit den Erfolgen zu Backgammon, Schach, Atari Videospielen, Poker und Go zu vergleichen [OT19].

## 3.2. Spielfeldaufbau und Verlauf

### 3.2.1. Aufbau

Die Agenten starten zu Beginn des Spiels in den Ecken des Spielfelds, wobei gegenüberliegende Agenten ein Team bilden. Das  $11 \times 11$  große Spielfeld wird zufällig initialisiert, unter der Bedingung, dass sich alle Agenten gegenseitig erreichen können. Teile des Spielfelds, die die Agenten nicht sehen können, sind durch Nebel ausgegraut. Abb. 3.1a zeigt eine mögliche Startkonfiguration inklusive der unterschiedlichen Sichtfelder der jeweiligen Agenten auf das Feld.

Das Spielfeld besteht aus drei verschiedenen Arten von Feldern:

-  Steinwand – kann nicht zerstört werden und ist nicht begehbar
-  Holzwand – versteckt Passagen oder Powerups, nicht begehbar bis sie gesprengt wird
-  Passage – begehbares Feld, kann andere Objekte enthalten:
  -  Agent
  -  Bombe – durch einen Agenten gelegt, Zeit bis zur Explosion als rote Linie am unteren Rand des Feldes dargestellt
  -  Flamme – durch Explosion einer Bombe ausgelöst, tötet Agenten, zerstört Holzwände und Powerups, hat eine Lebensdauer von mehreren Schritten
  -  ExtraBomb-Powerup – erhöht die Munition um eins
  -  IncreaseRange-Powerup – erhöht den Sprengradius um eins
  -  CanKick-Powerup – ermöglicht das Verschieben von Bomben
-  Nebel – verbirgt Felder außerhalb der Sicht eines Agenten

### 3.2.2. Regeln und Verlauf des Spiels

Pro Runde hat jeder Agent die Möglichkeit, eine der sechs Aktionen STOP, UP, LEFT, DOWN, RIGHT und BOMB auszuwählen, durch die er sich entsprechend auf dem Spielfeld bewegt oder eine Bombe legt. Entscheidet sich ein Spieler für einen nicht begehbar Pfad, kann die Aktion nicht ausgeführt werden und er bleibt in seiner vorherigen Position stehen. Legt er eine Bombe, verringert sich seine Munition um eins. Sobald die Bombe explodiert, wird der Zähler wieder um eins erhöht. Zu Beginn des Spiels startet jeder Agent mit einer Munition, durch das Einsammeln von *ExtraBomb*-Powerups kann diese

jeweils um eins erhöht werden.

Eine Bombe hat bis zu ihrer Explosion eine Lebensdauer von zehn Schritten. Der Explosionsradius einer Bombe in horizontaler und vertikaler Richtung wird von der Detonationsstärke des Agenten bestimmt. Initial beträgt diese drei Felder, wird aber durch *IncreaseRange*-Powerups jeweils um eins erhöht. Durch das Einsammeln von *CanKick*-Powerups hat jeder Agent außerdem die Möglichkeit, eine Bombe zu bewegen, wenn er auf sie trifft. Eine gekickte Bombe bewegt sich, bis sie gegen einen Spieler, eine Wand oder eine andere Bombe stößt. Die Spieler haben zu jedem Zeitpunkt Zugriff auf die verbleibende Lebensdauer, den Explosionsradius und die Bewegungsrichtung aller Bomben, die sich auf dem Spielfeld innerhalb ihres Sichtfelds befinden.

Zusätzlich kommunizieren die Agenten in jeder Runde über einen diskreten Kanal mit begrenzter Bandbreite. Jede Nachricht besteht aus zwei Zahlen („Wörter“), von denen beide 0 sind, wenn der Agent tot ist, ansonsten sind sie zwischen 1 und 8:  $\Sigma = [0, 8]$ . Die Bedeutung der Zahlenwerte sind (mit Ausnahme der 0) noch nicht festgelegt und müssen somit gelernt werden. Eine Nachricht wird im nächsten Schritt als Teil der Observation an das Teammitglied weitergeleitet. Die Wahl der Nachricht erfolgt parallel zu der Wahl der nächsten Aktion. Die Kommunikation ist kostenlos.

Das Spiel endet, wenn beide Agenten des gegnerischen Teams zerstört wurden. Unentschieden sind möglich, wenn das Spiel vor Ablauf der maximalen Anzahl an Schritten (i.e. 800) nicht zu Ende ist oder wenn die letzten beiden überlebenden Agenten in derselben Runde sterben.

### 3.3. Herausforderungen

Gao et al.<sup>1</sup> nennen in [GHLKT19] unter anderem folgende Eigenschaften als Gründe, warum Pommerman ein herausfordernder Benchmark für Reinforcement Learning und Multi-Agenten Systeme ist:

**Größe des Zustandsraums** Das Spielfeld hat aufgrund seiner Größe und möglichen Eigenschaften einen enorm großen Zustandsraum.

**Spärliche und nicht repräsentative Rewardfunktion** Die Agenten starten mit einem Reward von 0, der sich während des Spiels nicht verändert, das heißt innerhalb einer Episode erhalten die Agenten nach jedem Schritt einen Reward von 0. Erst am Ende einer Episode erhalten sie einen von 0 verschiedenen Wert als Reward, welcher bei einem Sieg 1 beträgt und bei einer Niederlage und einem Unentschieden -1. Das ermöglicht den Agenten erst sehr spät eine qualitative Bewertung ihrer Spielzüge. Der Reward ist außerdem nicht aussagekräftig, wenn sich Gegner selbst unfreiwillig durch ihre eigenen Bomben töten. Der Agent erhält dadurch einen positiven Reward für eine Aktion, die nicht maßgeblich zum Besiegen des Gegners beigetragen hat, was den Lernfortschritt des Agenten behindern kann [MKS<sup>+</sup>13].

**Verspätete Auswirkungen einer Bombe** Die einzige Möglichkeit, das Environment zu verändern, ist, Wände oder Gegner durch Bombenlegen zu sprengen. Auswirkungen der Aktion BOMB sind durch die Lebensdauer einer Bombe bis zu ihrer Explosion

---

<sup>1</sup>Zweitplatzierter in der Kategorie „Learning Agents“ bei der NeurIPS 2018.

### 3. Pommerman

erst verspätet zu beobachten, Zusammenhänge zwischen der Aktion und einer veränderten Environment sind dadurch schwerer zu erkennen. Andere Spieler können diesen Effekt verstärken, indem sie Bomben auf andere Positionen kicken.

**Partielle Beobachtbarkeit** Damit Agenten lediglich anhand ihrer aktuellen Observation Entscheidungen über ihre nächste Aktion treffen können, müssen ihre Beobachtungen die komplette Information des Spiels enthalten [RB19]. Jeder Agent besitzt jedoch nur ein begrenztes Sichtfeld. Das Ableiten des zugrundeliegenden Zustands des Environments anhand einer Observation ist dadurch nicht möglich; der vollständige Zustand des Environments bleibt unklar. Um die optimale Aktion bestimmen zu können, müssen Agenten demnach in der Lage sein, sich an vorherige Spielzustände zu erinnern [LC16, RN09, KLC98].

**Leistungszuordnung innerhalb eines Teams** Mitglieder eines Teams erhalten am Ende einer Episode denselben Reward. Die individuellen Leistungen der Agenten sind dadurch kaum differenzierbar. Beispielsweise könnte ein Spieler beide Gegner töten, während sein Teammitglied „campiert“, das heißt wiederholt die Aktion STOP ausführt, und nichts zum Sieg beiträgt. Beide Agenten würden einen positiven Reward erhalten, der ihr Verhalten bestärkt; ein *lazy* Agent wird trainiert.

## 3.4. Agenten

Pommerman stellt einen suchbasierten Standardagenten als Baseline für eigene Agenten zur Verfügung. Der **SimpleAgent** handelt nach einer vorgefundenen Heuristik, die in Algorithmus 1 vereinfacht skizziert ist.

Treten vier **SimpleAgents** in einem FFA-Spiel gegeneinander an, beträgt die Gewinnrate jedes Agenten 18%. In 28% der Fälle endet das Spiel mit einem Unentschieden.<sup>2</sup> Es wird explizit darauf hingewiesen, dass der **SimpleAgent** sich selbst zerstören kann. Das muss bei der Auswertung berücksichtigt werden.

In dieser Arbeit werden zusätzlich Agenten verwendet, die zufällig eine Aktion auswählen. Dem **RandomAgent** stehen alle sechs möglichen Aktionen zur Verfügung. Der **RandomNoBombAgent** legt keine Bomben; er wählt zufällig eine der Aktionen STOP, UP, LEFT, DOWN und RIGHT aus.

---

#### Algorithmus 1: Heuristik des **SimpleAgents**

---

```
1 act(Observation obs, Actionspace  $\mathcal{A}$ ):  
2     berechne Pfade zu Objekten in der Nähe  
         (mit Dijkstra und Radius  $\leq 10$ )  
3     if in Reichweite einer Bombe then  
4         return gehe Richtung sicheres Feld  
5     if neben Gegner and maybeBomb() then  
6         return Bombe  
7     if Gegner im Radius  $\leq 3$  then  
8         return gehe Richtung Gegner  
9     if Item im Radius  $\leq 2$  then  
10        return gehe Richtung Item  
11    if Holzwand im Radius  $\leq 1$  then  
12        if maybeBomb() then  
13            return Bombe  
14        else  
15            return Stop  
16    if Holzwand im Radius  $\leq 2$  then  
17        return gehe Richtung Holzwand  
18    return zufällige sichere Aktion  
19 maybeBomb():  
20    if Munition > 1 and Bombe legen sperrt  
        mich nicht ein then  
        return True  
22    return False
```

---

<sup>2</sup>Laut <https://github.com/MultiAgentLearning/playground/tree/master/docs>.

## 4. Verwandte Arbeiten

In diesem Kapitel werden Arbeiten aus den Themengebieten Deep Q-Learning und emergente Kommunikation in MAS zusammengefasst, um den aktuellen Stand der Forschung und etwaige Lücken aufzuzeigen. Des Weiteren werden die bisher erfolgreichsten Ansätze zu Pommerman skizziert.

### 4.1. Deep Q-Learning

Folgende Arbeiten sollen einen Überblick über die bisherigen Erfolge von Deep Q-Learning in Einzel- und Multi-Agenten Systemen ohne Kommunikation geben. Es kann beobachtet werden, dass Deep Q-Learning und Erweiterungen vielversprechende RL Verfahren sind.

#### 4.1.1. Einzel-Agenten Systeme

Mnih et al. (DeepMind) [MKS<sup>+</sup>13] präsentierten erstmalig ein Deep Learning Modell, das erfolgreich in der Lage war, hoch-dimensionale Einzel-Agenten Domänen mittels Reinforcement Learning und Q-Learning zu bewältigen. Das Modell erreichte in einer Vielzahl verschiedener Atari Videospiele „Superhuman Performance“ [MKS<sup>+</sup>15]. Das Netz setzt sich aus zwei Convolutional Layern zusammen, gefolgt von zwei weiteren fully-connected Layern. Diese Architektur wird bis heute (in abgewandelten Formen) standardmäßig für Deep Q-Networks in ähnlichen Problemstellungen eingesetzt. Im Zusammenhang mit Videospielen erhielt das DQN den momentanen Spielzustand in Form von rohen Pixeldaten (dem aktuellsten Frame) als Input.

Die Applikation von Vanilla DQNs ist, wie unter 2.3.3 erläutert, auf total beobachtbare Environments beschränkt. In der Praxis verwendeten Mnih et al. die letzten vier Frames des Videospiels als Input für ihr Netz, um in partiell beobachtbaren Ataridomänen den vollständigen Zustand abzuleiten. Dieser Ansatz hat den Nachteil, dass die Anzahl der verwendeten Frames fest ist; Informationen vorheriger Spielzustände werden nicht berücksichtigt.

Kempka et al. untersuchten in ihrer Arbeit [KWR<sup>+</sup>16] den Erfolg von Deep Q-Networks innerhalb komplexer 3D-Environments wie das Ego-Shooter Game Doom. Die Auswertung erfolgte anhand von zwei einfachen Anwendungsbeispielen:

- Ein Move'n'Shoot Szenario mit einem stationären Objekt als Ziel, dessen Position zu jeder Zeit sichtbar ist.
- Ein Maze Game, in dem der Agent bestimmte Objekte einsammeln und anderen ausweichen muss.

Im ersten Szenario erhielt das DQN das aktuellste Frame als Input. Die Ergebnisse zeigten, dass die Leistung des Modells stark von der Anzahl der Frames abhängt, die während des Trainings übersprungen werden. Die schnellsten Lernerfolge wurden mit vier bis zehn

#### 4. Verwandte Arbeiten

Frames erzielt.

Im zweiten Anwendungsbeispiel wurden analog zu der Arbeit von DeepMind die vier letzten Frames als Input für das DQN verwendet.<sup>1</sup> Trotz des Erfolges dieser Methode in Atari Environments, zeigen Kempka et al., dass das Modell nicht in der Lage ist, optimale Strategien für das Maze Game zu lernen.

Hausknecht et al. [HS15] zeigten, dass DRQNs die Leistungen von 4-Frames DQNs und sogar 10-Frames DQNs in partiell beobachtbaren Environments übertreffen.

Chen et al. [CYL16] trainierten verschiedene DRQN-Modelle anhand der Ataridomäne Q\*bert, eines der Spiele, in denen die Ergebnisse klassischer DQN-Modelle nur knapp über deren menschlicher Spieler lagen. Ihre Arbeit bestätigt, dass rekurrente Layer helfen können, bessere Ergebnisse zu erzielen.

Lample et al. [LC16] analysierten ebenfalls die Performance von Deep Q-Learning im Rahmen der 3D-Environment Doom. Mit Hilfe von DRQNs erreichten sie in verschiedenen Deathmatch<sup>2</sup> Szenarien deutlich bessere Resultate als integrierte Bots und menschliche Spieler. Diese Ergebnisse decken sich unter Berücksichtigung von [KWR<sup>+</sup>16] mit denen von Hausknecht et al.

Romac et al. [RB19] lieferten mit ihrer Arbeit einen weiteren Vergleich zwischen einem klassischen DQN, einem 4-Frames DQN und einem DRQN. Die Untersuchungen sind motiviert durch die Frage, ob ein DRQN in partiell beobachtbaren Environments immer bessere Leistungen als ein DQN erbringt, selbst wenn das Ziel simpel ist.

Alle drei Modelle wurden anhand zwei verschiedener Szenarien innerhalb des 3D-Videospiels Minecraft trainiert.

- Ein einfaches Found-the-Goal Problem: In einem  $7 \times 7 \times 7$  großen Raum muss ein Stück Gold gefunden werden.
- Das klassische Cliff-Walking Problem: Das Ziel ist von A nach B zu gelangen, ohne von der Klippe zu fallen.

Das Environment wurde für die Experimente so modifiziert, dass es partiell beobachtbar ist.

Die Ergebnisse des ersten Szenarios zeigten, dass das 4-Frames DQN deutlich effizienter und schneller lernte als das klassische DQN. Das Training des DRQNs dauerte im Schnitt am längsten, erzielte aber ähnlich gute Ergebnisse. Darüber hinaus wies die Lernkurve des DRQN-Modells den steilsten Anstieg auf. Der größte Unterschied zwischen den Modellen bestand darin, dass sich das 4-Frames DQN kontinuierlich verbessern konnte, während die Performance des klassischen DQNs mit der Zeit zu schwanken begann und die des DRQNs sogar schlechter wurde.

Das DRQN erzielte bessere Ergebnisse im zweiten Szenario, nach einiger Zeit kam es aber auch hier zu Performanceeinbrüchen. Die Erfolge des klassischen DQNs konnten

---

<sup>1</sup>Das Szenario ist partiell beobachtbar in dem Sinne, dass nur die Position der Objekte innerhalb des Sichtfelds des Agenten bekannt sind.

<sup>2</sup>Ein Spielmodus mit dem Ziel, die Anzahl der Kills zu maximieren.

nicht übertroffen werden. Das Modell lernte deutlich schneller und stabiler.<sup>3</sup>

Die Kernaussage der Arbeit ist, dass ein DRQN nicht immer die beste Wahl für partiell beobachtbare Environments ist. Ein DQN kann abhängig von der Komplexität der Domäne bereits ausreichend sein. Dennoch gilt: Ist es notwendig, sich an ältere Spielzustände zu erinnern, sind Vanilla DQNs nicht geeignet.

Die Ergebnisse von Romac et al. decken sich in weiten Teilen mit denen von Moreno-Vera [MV19]. Der Autor verglich die Performance von Vanilla DQNs in verschiedenen Atari Domänen mit deren von Deep Recurrent Q-Networks, Double Deep Q-Networks und einer Kombination aus beidem: Deep Recurrent Double Q-Networks (DRDQNs). Es konnte gezeigt werden, dass die Erweiterungen von Deep Q-Networks einzeln nicht immer zu Verbesserungen führten. DRDQNs erzielten im Gegensatz dazu jedes Mal die höchste Punktzahl.

#### 4.1.2. Multi-Agenten Systeme ohne Kommunikation

Tampuu et al. [TMK<sup>+</sup>15] setzten Independent Deep Q-Learning in einer Multi-Agenten Version des Atarispiels Pong ein und verglichen die Performance von kooperativen und nicht-kooperativen Agenten. Das Environment des Zwei-Personen Spiels ist für beide Agenten total beobachtbar. Die Agenten sind in der Lage effiziente Strategien zu erlernen. Independent Deep Q-Learning erzielt in weiteren empirischen Berichten jedoch meist schlechtere Ergebnisse als andere Ansätze [SLG<sup>+</sup>17], [CB98], [FAdFW16].

Sunehag et al. [SLG<sup>+</sup>17] testeten Independent Q-Learning, Value Decomposition und zentralisierte Ansätze in mehreren partiell-beobachtbaren Domänen mit dem Ergebnis, dass VD (und Erweiterungen davon) deutlich besser performt.

Egorov [Ego16] verfolgt eine andere Methode: In der untersuchten Domäne treten die Agenten ähnlich zu Pommerman in Teams gegeneinander an, mit dem Unterschied, dass das Environment total beobachtbar ist. Für jedes Team wurde insgesamt nur ein Netz verwendet. Jeder Agent wurde nacheinander trainiert, die Policies der anderen Agenten blieben für diese Zeit unverändert; die Policy des lernenden Agenten wurde anschließend an dessen Teammitglieder weitergegeben. Die Agenten lernten in einem 1 vs. 1 Szenario (ohne Verteilung der Policy) eine nahezu optimale Strategie; in 2 vs. 2 und 3 vs. 3 Settings übertrafen sie heuristische Verfahren.

## 4.2. Emergente Kommunikation

Folgende Arbeiten begrenzen sich auf partiell beobachtbare Multi-Agenten Probleme, in denen Kommunikation mittels Deep Q-Learning gelernt werden soll (i.e. emergente Kommunikation). Das bedeutet, es ist vorab kein Kommunikationsprotokoll gegeben.

Foerster et al. [FAdFW16] konzentrierten sich in ihrer Arbeit auf zwei Environments

---

<sup>3</sup>Diese Ergebnisse entsprachen nicht denen des 4-Frames DQNs, dessen Lernerfolge nach  $\frac{2}{5}$  der Trainingszeit stoppten und im Laufe der nächsten Episoden um mehr als die Hälfte sanken. Die Autoren untersuchten die Ursache nicht weiter.

#### 4. Verwandte Arbeiten

mit diskretem Kommunikationskanal, dessen Bandbreite zur Ausführungszeit limitiert ist. Eine Nachricht in den Domänen „Switch Riddle“ und „MNIST Games“ hat die Form eines einzelnen Bits. Beide Domänen sind vollständig kooperativ. Die Anzahl der Agenten ist variabel. Sie präsentierten zwei DQN-Modelle namens *Reinforced Inter-Agent Learning* (RIAL) und *Differentiable Inter-Agent Learning* (DIAL), die von zentralem Training und dezentraler Ausführung (ein gängiges Paradigma in MAS, vgl. [OSV11]) profitieren. Das bedeutet, die Agenten werden zusammen trainiert, aber getrennt voneinander evaluiert [JKJG16]. Ein Ansatz ohne Kommunikation (NoComm) diente in den Experimenten als Ausgangswert und zusätzliche Vergleichsmöglichkeit.

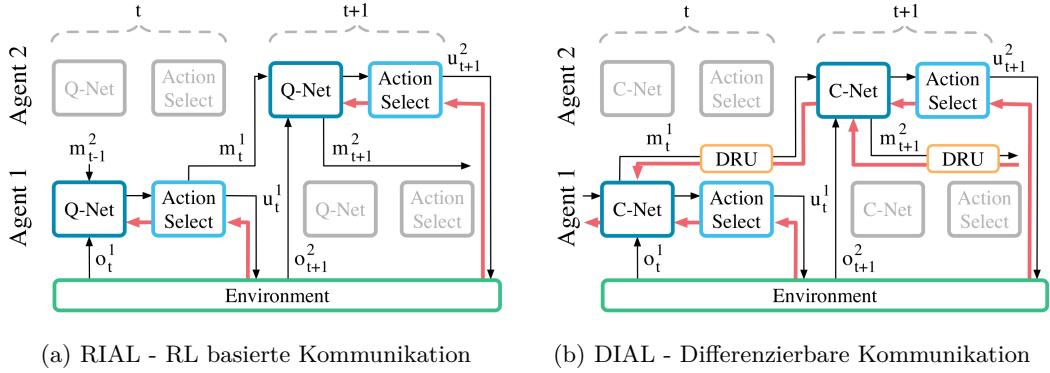


Abbildung 4.1.: RIAL und DIAL Architektur im Vergleich (entnommen aus [FAdFW16]).

In RIAL (a) werden sowohl Aktion als auch Nachricht auf Basis der Q-Values bestimmt. Back Propagation (in rot dargestellt) erfolgt nur über das Netz eines Agenten. In DIAL (b) verarbeitet eine „Discretise/Regularise Unit“ (DRU) die Nachricht zu einem differenzierbaren Vektor während des Trainings, bevor sie an das Teammitglied weitergeleitet wird. Die Gradienten durchqueren bei BP die Netze beider Agenten.

RIAL kombiniert DRQNs mit Independent Q-Learning. Um die Anzahl der Q-Values zu verringern, verwendet jeder Agent für die Auswahl der Aktionen und Nachrichten zwei separate Netze.<sup>4</sup> Für die Testphase sind keine Anpassungen nötig, da das Training dezentral erfolgt. In einer Erweiterung von RIAL wird während des Trainings für alle Agenten ein gemeinsames Netz verwendet, sog. *Parameter Sharing*. Aufgrund unterschiedlicher Observationen verhalten sich die Agenten dennoch nicht gleich. In den dezentralen Testdurchläufen erhält jeder Agent eine eigene Kopie des trainierten Netzes.

In DIAL wird der Kommunikationskanal zwischen den Agenten während des Lernprozesses durch eine stetige Verbindung zwischen dem Output eines Netzes und dem Input des anderen Netzes ersetzt. Der neuen Kanal ermöglicht die Übertragung jeglicher reellwertiger Nachrichten, auch Gradienten. Back Propagation erfolgt nunmehr über ein gesamtes Netz. Es entsteht ein System, das End-to-End-Training der Agenten ermöglicht. Die zentrale Verbindung zwischen den Netzen wird in den Tests aufgehoben; reellwertige Nachrichten werden diskretisiert und auf die Menge der erlaubten Wörter abgebildet. DIAL kann auch für die Umsetzung von Parameter Sharing angepasst werden.

<sup>4</sup>Statt  $|\mathcal{A}||\Sigma|$  Outputs werden pro Netz nur noch  $|\mathcal{A}| + |\Sigma|$  Outputs benötigt. In (2.18) wird nicht über  $\mathcal{A} \times \Sigma$  maximiert, sondern separat über  $\mathcal{A}$  und  $\Sigma$ .

Abbildung 4.1 zeigt den Informationsaustausch innerhalb beider Modelle (ohne Parameter Sharing) im Vergleich.

Experience Replay wurde in beiden Ansätzen „ausgeschaltet“ mit der Begründung, dass dieses Verfahren für simultan lernende Agenten nicht geeignet ist (aufgrund der Nicht-Stationarität des Environments). Die Erfahrungstupel im Replay Memory sind dadurch unbrauchbar.

Die Experimente zeigten, dass DIAL (mit und ohne Parameter Sharing) und RIAL (mit Parameter Sharing) in jedem Setting schneller und besser lernten als Independent Q-Learning und RIAL (ohne Parameter Sharing). Die Erkenntnisse vorheriger Arbeiten wurden bestätigt: Zentrale Modelle sind erfolgreicher als dezentrale Ansätze. Parameter Sharing beschleunigte zudem für beide Modelle den Lernfortschritt. RIAL und NoComm waren nicht immer in der Lage, eine Policy zu lernen. Grundsätzlich gilt aber, dass RIAL mit Parameter Sharing bessere Ergebnisse erzielte als NoComm. Diese Ergebnisse deuten darauf hin, dass Kommunikation und zentralisiertes Training für MAS essentiell sind, um optimale Strategien zu erlernen.

Vanneste et al. [VVB<sup>+</sup>20] erweiterten DIAL mit Value Decomposition. Sie präsentierten die Modelle DIAL-VD und DIAL-VDN. Das VDN besteht aus zwei fully-connected Layer mit ReLU Aktivierung dazwischen. Abbildung 4.2 zeigt die Architektur beider Modelle.

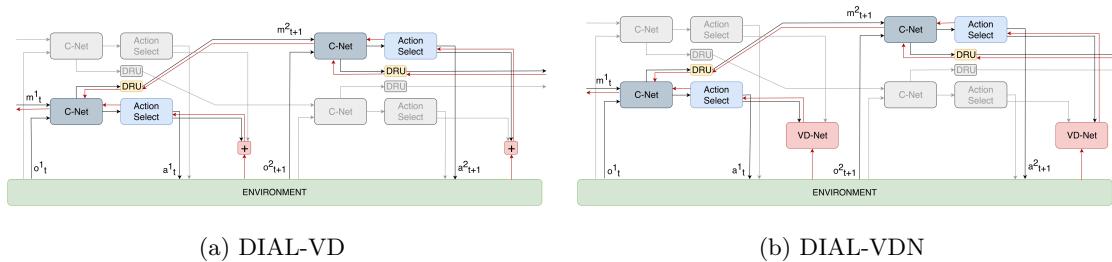


Abbildung 4.2.: Die Architektur von DIAL mit Value Decomposition (Networks) erweitert (entnommen aus [VVB<sup>+</sup>20]). BP erfolgt über die Summe der Q-Values der ausgewählten Aktionen.

Die Autoren untersuchten die Performance von DIAL-VD und DIAL-VDN im Vergleich zu klassischem DIAL. Alle Modelle setzen Parameter Sharing um. Die Auswertung erfolgte anhand des Zwei-Spieler OpenAI-Environments „Simple Reference“.<sup>5</sup> Eine Nachricht besteht aus zwei Bits.

Die Experimente zeigten, dass DIAL-VD am besten performte. Das Modell lernte nicht nur schneller als DIAL und DIAL-VDN, sondern erzielte auch bessere Ergebnisse. DIAL-VDN führte im Gegensatz dazu kaum zu Verbesserungen. Zu Beginn des Trainings lernte das Modell zwar schneller als DIAL, konnte dessen Ergebnisse aber nicht übertreffen. Das Training beider Value Decomposition Methoden ist instabil. Es kommt zu extremen Performanceeinbrüchen nach Erreichen der Höchstleistung. Die Performance von DIAL sinkt ebenfalls.

<sup>5</sup>Code verfügbar auf <https://github.com/openai/multiagent-particle-envs>.

#### 4. Verwandte Arbeiten

Bei Sukhbaatar et al. [SSF16] lernen Agenten, miteinander zu kommunizieren, *bevor* sie eine Aktion auswählen. Das Modell namens *CommNet* fungiert als zentrale Instanz, die das Verhalten aller Agenten steuert. CommNet erhält die Observationen aller Agenten und bildet diese auf eine gemeinsame Aktion ab. Jeder Agent besitzt ein eigenes Modell mit individuellem Hidden State, das Zugang zu einem stetigen Kommunikationskanal hat. Über diesen Kanal wird eine Zusammensetzung der Nachrichten aller Agenten weitergeleitet. Die Nachricht eines Agenten entspricht dem Hidden State seines Moduls. Jedes Modul verwendet diese Information, um den eigenen Hidden State anzupassen. CommNet erlaubt mehrere Kommunikationsschritte zwischen den Agenten vor Auswahl der Aktionen, wodurch die Hidden States der Module mehrmals angepasst werden. Abbildung 4.3 zeigt den Kommunikationsfluss im gesamten Modell.

Es ist nicht vorgegeben, inwiefern die anderen Hidden States bei der Berechnung des eigenen berücksichtigt werden sollen. CommNet versucht, das zu lernen. Das Modell kann mit üblichen Einzel-Agenten RL Verfahren trainiert werden, da der stetige Kommunikationskanal die Übertragung von Gradienten erlaubt (analog zu DIAL). Dazu wird eine Policy Gradient Methode verwendet.

Die Autoren zogen unter anderem folgende Grundmodelle als Vergleichsobjekte heran:

- Independent Policy Gradient – analog zu Independent Q-Learning, Kommunikation nicht möglich
- Diskrete Kommunikation – Agenten kommunizieren über diskrete Symbole, deren Bedeutung während des Trainings gelernt wird

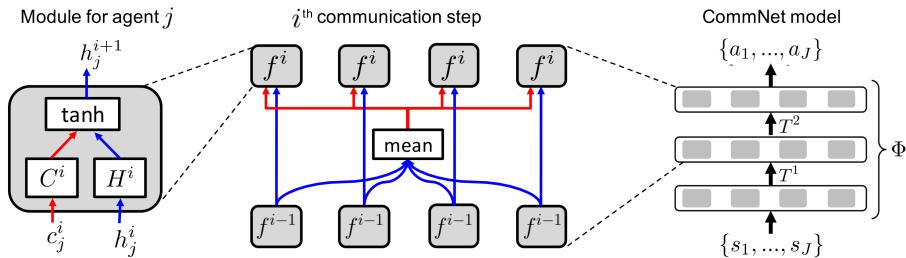


Abbildung 4.3.: Die Architektur des CommNets (entnommen aus [SSF16]). Links: Das Modul eines Agenten. Mitte: Ein einzelner Kommunikationsschritt. Ein stetiger Kommunikationskanal überträgt eine Zusammensetzung der Nachrichten (i.e. die Hidden States der Module) an alle Agenten. Rechts: Das gesamte Modell mit zwei Kommunikationsschritten vor Auswahl der Aktionen.

Die Verfahren wurden an mehreren Domänen getestet. Die Experimente zeigten, dass diskrete Kommunikation in der einfachen Version fast so gut performte wie CommNet. Die Ergebnisse diskreter Kommunikation kommen aber vor allem in schwierigeren Environments nicht an die des CommNets heran. Die Autoren beobachteten außerdem, dass Kommunikation umso vorteilhafter ist, je kleiner das Sichtfeld der Agenten.

In weiteren Analysen wurden Hidden States und entsprechende Kommunikationsvektoren miteinander verglichen, um zu verstehen, was die Agenten kommunizieren (i.e. inwiefern ein Agent Information der anderen berücksichtigt). Eine 2D Hauptkomponentenanalyse zeigt, dass die Projektionen der Hidden States breit gefächert sind; das deutet darauf hin,

dass die Vektoren Informationen enthalten. Im Gegensatz dazu sind viele der Kommunikationsvektoren fast null; die Agenten scheinen gelernt zu haben, nur dann zu sprechen, wenn es notwendig ist. Aufgrund des gemeinsamen Kommunikationskanals ist es hilfreicher, wenn möglichst wenig gleichzeitig sprechen.

Cao et al. [CLL<sup>+</sup>18] stellten sich die Frage, welches Environment die Entstehung von Kommunikation begünstigt. Sie betrachten dazu ein semi-kooperatives Verhandlungsproblem, in dem zwei Agenten mehrere Objekte zwischen sich aufteilen müssen. Das Environment ist partiell beobachtbar in dem Sinne, dass keiner der Agenten weiß, welche Objekte sein Gegenüber möchte. Über den einen Kanal können Agenten eine konkrete Aufteilung vorschlagen („Proposal Channel“). Ein anderer ermöglicht die Übertragung diskreter Symbole, deren Bedeutung nicht vordefiniert ist („Linguistic Channel“). Ein Agent kann die Verhandlung jederzeit beenden, wodurch er den Vorschlag des anderen Agenten annimmt.

Der Reward ist prosozial.<sup>6</sup> Cao et al. unterscheiden zwischen prosozialen und eigennützigen Agenten.<sup>7</sup> Beide Agenten werden mittels *REINFORCE* Algorithmus trainiert.

In ihrem ersten Experiment widmeten sich die Autoren der Frage, ob eigennützige Agenten lernen zu verhandeln. Unter Verwendung des Proposal Channels lernten die Agenten die Objekte fair (aber nicht optimal) aufzuteilen. Stand nur der Linguistic Channel zur Verfügung, konnten sich die Agenten nicht einigen. Der Nachrichtenaustausch deutet darauf hin, dass die Agenten nicht in der Lage sind, relevante Informationen für die Aufteilung auszutauschen. Das unterstützt die Hypothese, dass Kooperation eine Voraussetzung für die Entstehung von Kommunikation ist.

Das zweite Experiment drehte sich um die Frage, ob prosoziale Agenten lernen, die optimale Verteilung herauszufinden. Dazu muss jeder seinen Nutzen an den Objekten offenbaren. Die Agenten konnten die Aufgabe unter Verwendung des Linguistic Channels am besten lösen. Die Bandbreite des Kanals ist unbegrenzt, was den effektiven Austausch relevanter Informationen begünstigt. Weitere Untersuchungen zeigen, dass die Agenten gelernt haben, den Symbolen eine semantische Bedeutung zu geben. Ihr Sprachverhalten lässt vermuten, dass sie unterschiedliche Rollen einnehmen: Ein Agent (der Sprecher) teilt seinen Nutzen, den der andere (der Zuhörer) mit seinem vergleicht und die optimale Verteilung ableitet. Im Vergleich dazu erzielten die Agenten bessere Ergebnisse, wenn nur der Proposal Channel zur Verfügung steht, als wenn sie gar nicht kommunizieren. Das deutet darauf hin, dass sie lernten den Proposal Channel umzufunktionieren, um Informationen auszutauschen.

Lazaridou et al. [LPB16] untersuchten das Kommunikationsverhalten von zwei Agenten in einem „Referential Game“. Ein Agent (der Sprecher) muss aus mehreren Bildern ein Target beschreiben, das der andere Agent (der Zuhörer) identifizieren soll. Die Objekte in den Bildern werden in unterschiedliche Kategorien gegliedert. Das Target kann ein Objekt sein, das für beide Agenten gleich ist, oder aber nur derselben Kategorie angehören (e.g. Target ist ein Bild mit „Hund“, beide Agenten sehen Bilder mit unterschiedlichen Hunden).

---

<sup>6</sup>Das heißt er wird auf beide Agenten aufgeteilt:  $R = \alpha R_i + \beta R_j$ .

<sup>7</sup>Für einen eigennützigen Agenten  $i$  ist  $\alpha = 1$  und  $\beta = 0$ , während für einen prosozialen Agenten  $\alpha = \beta = 1$  ist. Prosoziale Agenten streben eine optimale Verteilung der Objekte an (i.e. jedes Objekt geht zu dem Agenten, der den größten Nutzen dafür hat).

#### 4. Verwandte Arbeiten

Die Nachricht enthält ein Symbol aus einem Alphabet fester Länge. Ein zuvor trainiertes VGG CNN<sup>8</sup> verarbeitet jedes Bild, bevor es den Agenten gezeigt wird. Die Agenten wurden mittels *REINFORCE* Algorithmus trainiert.

Die Ergebnisse zeigten, dass die Agenten erfolgreich miteinander kommunizieren. Eine Analyse der Nachrichten (vgl. Abbildung A.1 im Anhang) bestätigte, dass benachbarte (i.e. optisch ähnliche) Objekte mit demselben Symbol beschrieben wurden. Das deutet darauf hin, dass die Symbole allgemeine konzeptuelle Eigenschaften des Bildes (keine detaillierten Merkmale) beschreiben.

Um Symbole besser interpretieren zu können, erweitern die Autoren das Modell mit Supervised Learning, sodass der Sprecher lernen muss, die konventionellen Namen der Objekte zu verwenden. Die Kommunikation nähert sich dadurch der natürlichen Sprache an.

Havrylov et al. [HT17] betrachteten ebenfalls ein Referential Game, mit folgenden Unterschieden: Die Nachricht besteht aus einer Sequenz von Symbolen und der Sprecher sieht nur das Target. Die maximale Länge der Nachrichten ist vorgegeben. Die Experimente zeigten, dass die Performance der Agenten höher ist je länger die Nachrichten sind. Weitere Analysen verraten, dass die Reihenfolge der Symbole eine Rolle spielt. Die Agenten lernten genauer betrachtet eine Art hierarchische Codierung. Abbildung A.2 im Anhang zeigt Beispielbilder, die bestimmten Sequenzen entsprechen.

Lazaridou et al. führten ihre Forschung in [LHTC18] fort. Der Fokus ihrer Arbeit liegt auf der Frage, ob die entstandene Sprache das *Kompositionalsprinzip* natürlicher Sprache erfüllt.<sup>9</sup> Sowohl Sprecher als auch Zuhörer sehen mehreren Bilder, aber nur der Sprecher kennt das Target. Das Setting gleicht ansonsten dem in [HT17]. Folgende Szenarien wurden untersucht:

- (i) Die Objekte in den Bildern wurden durch mehrere Eigenschaften symbolisch repräsentiert (e.g. das Objekt „Auto“ hat die Eigenschaften „hat Sitze“, „hat Reifen“ und „hergestellt aus Metall“).
- (ii) Das Referential Game musste auf Basis roher Pixeldaten bewältigt werden. Die Bilder wurden zuvor nicht verarbeitet. Verschiedene Varianten des Spiels wurden betrachtet.<sup>10</sup>

Die Autoren verwendeten REINFORCE für das End-to-End-Training der Agenten.

Die Ergebnisse von (i) deckten sich mit denen von Havrylov et al.: Der Erfolg der Agenten nahm zu je größer die maximale Länge der Nachrichten war. Ähnliche Objekte produzierten ähnliche Nachrichten. Die ersten beiden Fragmente einer Nachricht scheinen kategorische Informationen zu enthalten. Weitere Testdurchläufe zeigten, dass die Agenten neue Nachrichten für Bilder generierten, die sie während des Trainings nicht gesehen haben – ein Hauptmerkmal des Kompositionalsprinzips.

In (ii) bewältigten die Agenten ebenfalls alle Variationen des Problems erfolgreich. Die

---

<sup>8</sup>Eine gängige CNN-Architektur zur Objekterkennung in Bildern.

<sup>9</sup>Das Kompositionalsprinzip besagt, dass die Bedeutung eines zusammengesetzten Ausdrucks durch die Bedeutung seiner Teile, und wie sie kombiniert wurden, bestimmt ist [JP97]. Dadurch können aus einem endlichen Wörterbuch und endlichen Kombinationsregeln prinzipiell unendlich viele Ausdrücke generiert werden [LHTC18].

<sup>10</sup>Unterschiedliche Anzahl der gezeigten Bilder, unterschiedliche Position des Targetbildes für Sprecher und Zuhörer, gleiche Anzahl der Farben und Objekte.

Nachrichten enthielten strukturell einheitliche Informationen, wenn den Agenten viele Bilder vorgelegt wurden: Das erste und letzte Fragment kennzeichneten die horizontale und vertikale Position der Objekte, d.h. ähnliche Szenarien (i.e. Positionen der Objekte) produzierten ähnliche Nachrichten – ein weiteres Merkmal des Kompositionalsprinzips. Ein Zusammenhang zwischen Nachrichten und Objekten ist allerdings nicht immer eindeutig. Die Fähigkeit, Eigenschaften der Objekte zu erkennen, ist entscheidend für ein Protokoll nach Art des Kompositionalsprinzips. Das geschah sonst durch die Vorverarbeitung der Bilder.

Jorge et al. [JKJG16] analysierten die Entstehung von Kommunikation in einer kooperativen Adaption des Spiels „Wer ist es?“. Das Environment ähnelt dem klassischen Referential Game. Die Länge der Konversation ist festgelegt. Die Bilder der Charaktere wurden anders als in [LPB16] und [HT17] zuvor nicht verarbeitet.

Die Autoren präsentierten ein Modell ähnlich zu DIAL ohne Parameter Sharding (vgl. Abbildung 4.4).<sup>11</sup> Die Nachrichten wurden in Form eines One-Hot Encodings ausgetauscht. Pro Experiment standen den fragenden Agenten unterschiedlich viele Wörter zur Verfügung (ein Wort symbolisiert eine Frage). Die Ergebnisse zeigten, dass die Performance der Agenten nicht von der Größe des Datensatzes abhängt. Ein größeres Alphabet ist jedoch hilfreicher je weniger Antwortmöglichkeiten zur Auswahl stehen. Eine Analyse der Nachrichten (vgl. Abbildung A.3 im Anhang) deutet darauf hin, dass ähnliche Bilder dieselbe Antwort produzierten. Weitere Experimente zeigten, dass die zweite Frage von Spieler 2 zu 93% von der Antwort auf seine vorherigen Frage abhängt. Die Agenten scheinen eine Art Konversation gelernt zu haben.

Evtimova et al. [EDKC17] änderten das klassische Referential Game weiter ab. Spieler 1 sieht als einziger die graphische Repräsentation der Bilder. Jedes Bild wurde zuvor von ResNet-34<sup>12</sup> verarbeitet. Spieler 2 hat nur Zugang zu textuellen Beschreibungen. Die Nachrichten sind binär. Der Austausch ist bidirektional und symmetrisch (i.e. beiden Agenten können dieselben Nachrichten verschicken). Die Länge des Dialogs ist variabel. Die Autoren konnten herausfinden, dass die Länge der Konversation zunahm je schwieriger die Kategorie der Objekte war. Die Gewissheit von Spieler 2 über seine Prognose nahm im Verlauf der Konversation zu. Parallel dazu wurden seine Fragen mit der Zeit immer spezifischer. Dadurch sank die Sicherheit von Spieler 1, die Frage richtig zu beantworten.

<sup>11</sup> Anders als in [FAdFW16] generierte das Modell keine Aktion, sondern nur eine Nachricht.

<sup>12</sup> Eine gängige CNN-Architektur zur Bildverarbeitung und Objekterkennung.

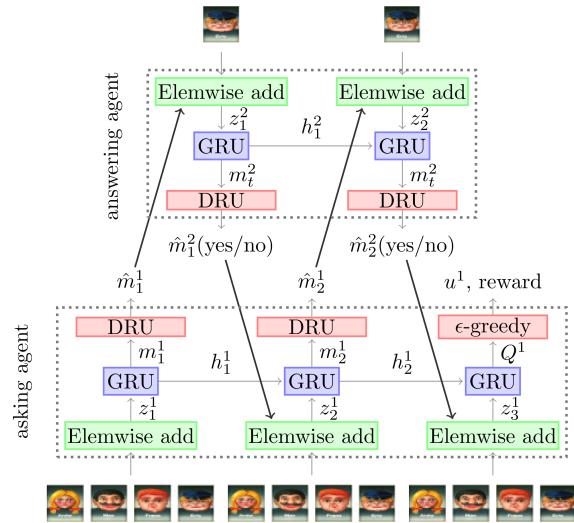


Abbildung 4.4.: Schematische Illustration des Spiels „Wer ist es?“ (entnommen aus [JKJG16]). Die Agenten verwenden ein Modell ähnlich zu DIAL.

## 4.3. Ansätze zu Pommerman

Momentan stehen keine näheren Informationen zu den Siegerimplementierungen der Pommerman Team-Radio Competition von 2019 zur Verfügung. Das folgende Kapitel beschränkt sich deswegen auf die Gewinner der FFA und Team Competitions, die im Mai und November 2018 stattfanden, sowie weitere empirische Arbeiten, deren Ansätze Pommerman erfolgreich bewältigten.

### 4.3.1. Non-Deep Learning Methoden

#### Hybrid Search Agent

Zhou et al. [ZGM<sup>+</sup>18] kombinierten in ihrer Arbeit suchbasierte Verfahren mit tiefenbeschränkten Suchbäumen. Ein endlicher Automat bestimmt, welche Methode zum Einsatz kommt. Der Agent verwendet eine feste Explorationsheuristik, wenn er sich in einem sicheren Zustand befindet. Wenn sich Gegner oder Bomben, deren Lebensdauer unter einem bestimmten Grenzwert liegt, in der Nähe befinden, werden tiefenbeschränkte Suchbäume eingesetzt. Die Suchbäume sind aufgrund des Zeitlimits zur Auswahl der nächsten Aktion (i.e. 10ms) tiefenbeschränkt. Es wurden mehrere Varianten mit unterschiedlichen Suchalgorithmen für die Baumsuche implementiert. Jedes Modell wurde in 300 FFA-Spielen gegen drei `SimpleAgents` getestet. Die Gewinnraten aller Modelle betrugen 60 – 80%.

Yichen Gong (Co-Autor der Arbeit) gewann mit `Agent47Agent` die Pommerman FFA Competition von 2018.<sup>13</sup> Der Agent entsprach vermutlich einem der Modelle aus [ZGM<sup>+</sup>18]. Welcher Suchalgorithmus verwendet wurde, ist nicht bekannt.

#### Real-Time Tree Search with Pessimistic Scenarios

Osogami et al. [OT19] entwickelten für die Pommerman Team Competition die Agenten `hakozaki` und `dypm`. Beide Agenten verwenden Suchbäume mit einer Tiefe gleich eins. Ausgehend von diesem Zustand, simulieren sie ein pessimistisches Szenario. Diese Idee basiert auf der Annahme, dass gute Aktionen sich dadurch auszeichnen, dass sie selbst in den schlechtesten Bedingungen gut performen. Die Szenarien sind deterministisch, wodurch die Suchbäume nach der ersten Ebene nicht mehr verzweigt sind. Dadurch wird keine Rechenzeit in die Evaluation von Pfaden investiert, die nicht besucht werden. Abbildung 4.5 visualisiert einen Suchbaum dieser Art mit einer Tiefe von zwei.

In den meisten Fällen ist das „Worst Case“ Szenario nicht bekannt. Die Agenten konstruieren ein pessimistisches Szenario, indem gegnerische Spieler mehrere Aktionen gleichzeitig ausführen (unabhängig davon, ob das realistisch oder erlaubt ist). Jedes Szenario wird anhand der Überlebensfähigkeit des eigenen bzw. gegnerischen Teams bewertet. `hakazaki` macht seine Überlebensfähigkeit an der Anzahl der Positionen fest, die erreicht werden können, ohne auf einen anderen Agenten zu treffen. `dypm` misst seine Überlebenschancen anhand der Anzahl der Felder, in denen der bis zum Ende des Szenarios überlebt.

Osogami et al. erreichten mit `hakozaki` den ersten und mit `dypm` den dritten Platz des Wettbewerbs.

---

<sup>13</sup>Laut <https://www.pommerman.com/competitions>.

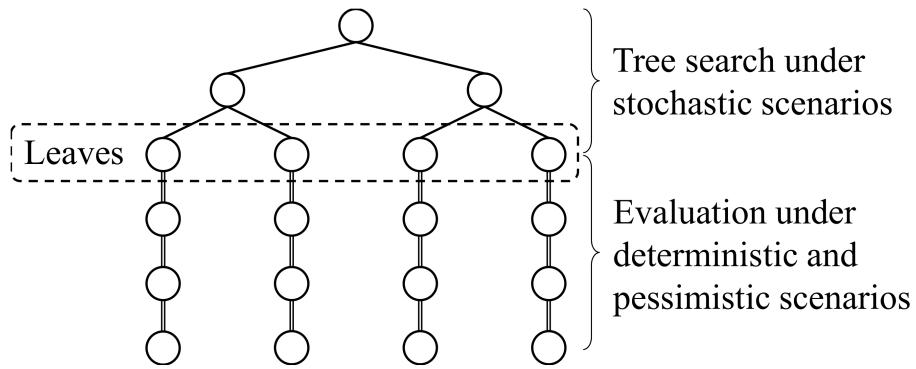


Abbildung 4.5.: Baumsuche mit pessimistischem Szenario (entnommen aus [OT19])

### 4.3.2. Deep Learning Methoden

#### Deep Reinforcement Learning und MCTS

Kartal et al. [KHLT18] präsentierte ein Framework, das Planner Imitation (PI) mit *Asynchronous Advantage Actor-Critic* (A3C) kombiniert. Die Idee von PI-A3C ist, eine Policy zu optimieren und parallel MCTS zu imitieren. Jeder Worker besitzt analog zu Vanilla A3C ein eigenes Actor-Critic Netz. Einer der Worker wird zusätzlich mit MCTS ausgestattet, um damit die nächsten Aktionen zu bestimmen. Der Loss dieses Workers wird auf Basis der ausgewählten Aktion und der Aktion, die sein Actor-Critic vorgegeben hätte, bestimmt. Abbildung 4.6 veranschaulicht die Unterschiede zwischen Vanilla A3C und PI-A3C graphisch.

Kartal et al. verwenden in ihren Experimenten eine vereinfachte Variante der Pommerman FFA-Konfiguration mit zwei Spielern und einem  $8 \times 8$  großen Spielfeld. Das Framework wurde gegen zwei verschiedene Agenten getestet:

- Ein **Static Agent**, der sich weder bewegt, noch Bomben legt.
- Ein **SimpleAgent**.

Beide Experimente zeigten, dass PI-A3C deutlich schneller konvergiert als Vanilla PI-A3C und eine bessere Policy erlernt.

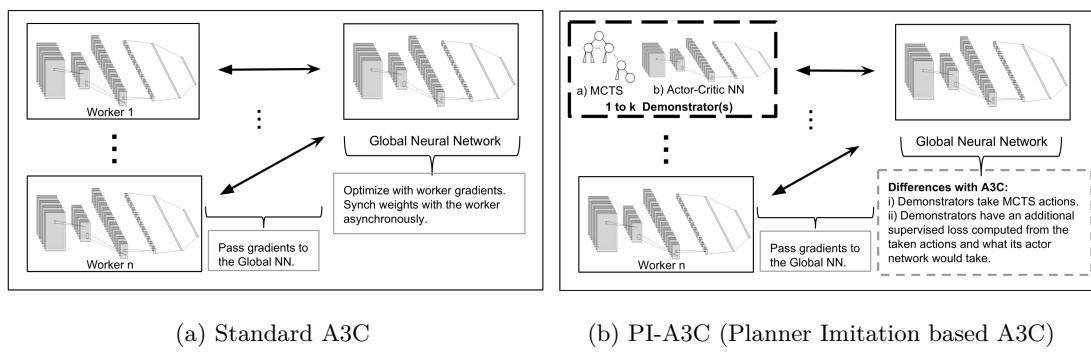


Abbildung 4.6.: Standard A3C vs. PI-A3C (entnommen aus [KHLT18])

#### 4. Verwandte Arbeiten

##### **Skynet955**

Gao et al. [GHLKT19] präsentierte in ihrer Arbeit das **Skynet955**-Team, das in der Pommerman Team Competition den zweiten Platz in der Kategorie „Learning Agents“ belegte. Das Team besteht aus zwei Deep Neural Networks, die mit *Proximal Policy Optimization* (PPO) und den Konzepten Reward Shaping, Curriculum Learning und Action Pruning trainiert wurden. Trotz partieller Beobachtbarkeit enthalten die DNNs keine rekurrenten Schichten. Jeder Agent verwendet stattdessen ein „Retrospective Board“, um sich an die Belegung aller Felder zu erinnern. Für Felder außerhalb der Sicht eines Agenten enthält das Board den zuletzt gesehenen Wert. Der Input der Netze setzt sich aus 14 Kanälen der Größe  $11 \times 11$  zusammen, wovon die ersten zehn aus der Observation eines Agenten extrahiert werden und die restlichen vier aus dem Retrospective Board. Gao et al. begründeten ihre Entscheidung damit, dass das Team mit Retrospective Boards ähnlich gut, aber deutlich schneller als mit LSTMs performte.

Um den Lernprozess zu beschleunigen, wurde jeder Agent mit einem Aktionsfilter ausgestattet (vgl. Tabelle 4.1). Die spärliche Rewardfunktion wurde während des Trainings modifiziert und um weitere Rewards ergänzt (vgl. Tabelle 4.2).

Das Training erfolgte in zwei Etappen gegen zwei verschiedene Gegner:

- (i) Ein **Static** Team.
- (ii) Ein Team aus **SmartRandomNoBomb** Agenten, die den Aktionsfilter nutzen und keine Bomben legen.

Die Gewinnrate des **Skynet955**-Teams gegen ein Team aus **SimpleAgents** lag bereits nach der ersten Phase des Trainings bei 70% [GKHLT19].

<i>Selbstmord vermeiden</i>	Gehe nicht zu Positionen, die im nächsten Schritt Flammen sind.
	Gehe nicht zu „todgeweihten“ Positionen, i.e. Positionen, aus denen es keinen Ausweg gibt. Todgeweihte Positionen können für jede Bombe anhand ihrer Lebensdauer und ihres Explosionsradius zusammen mit der Umgebung bestimmt werden.
<i>Platzierung der Bomben</i>	Lege keine Bomben, wenn ein Teammitglied in der Nähe ist, i.e., wenn ihre Manhattan Distanz kleiner als die Summe eurer Explosionsradien ist.
	Lege keine Bomben, wenn du dich im Explosionsradius einer anderen Bombe befindest.

Tabelle 4.1.: Aktionsfilter eines **Skynet955**-Agenten (entnommen aus [GHLKT19])

0.001 für das Betreten von Feldern, in denen man noch nicht war	0.5 wenn man Teil des Gewinnerteams ist, aber bereits tot
0.02 für das Aufheben von <i>CanKick</i>	0 bei einem Unentschieden
0.01 für das Aufheben von <i>ExtraBomb</i>	0.5 bei dem Tod eines Gegners
0.01 für das Aufheben von <i>IncreaseRange</i>	−0.5 bei dem Tod eines Teammitglieds

Tabelle 4.2.: Reward Shaping eines **Skynet955**-Agenten (entnommen aus [GHLKT19])

## COMBAT Framework

Peng et al. verwendeten Continual Learning, um für die Pommerman Team Competition eine Population aus Advantage-Actor-Critic (A2C) Agenten zu trainieren [PPYG18], [PLGD<sup>+</sup>19]. In ihrem Framework namens *Continual Match BAseD Training* (COMBAT) treten die Agenten der Population in mehreren Spielen gegeneinander an. Auf Basis seiner Leistungen erhält jeder Agenten einen bestimmten Rang. Die Netze der Agenten werden anschließend durch den Optimierungsalgorithmus A2C aktualisiert. COMBAT eliminiert im nächsten Schritt die Agenten der niedrigsten Ränge. Parallel erfolgt ein Fein-tuning des Discountfaktors für Netze, die bereits konvergiert sind. Der Vorgang wird anschließend wiederholt. Abbildung 4.7 veranschaulicht das Verfahren graphisch. Das Sichtfeld der Agenten in den partiell beobachtbaren Konfigurationen ist im Vergleich zum Spielfeld relativ groß. Peng et al. interpretierten das System aufgrund dessen als MDP und verzichteten auf rekurrente Schichten in den DNNs.

Navocado (ein Agent, der mit dem COMBAT Framework trainiert wurde) belegte den ersten Platz der Pommerman Team Competition in der Kategorie „Learning Agents“.

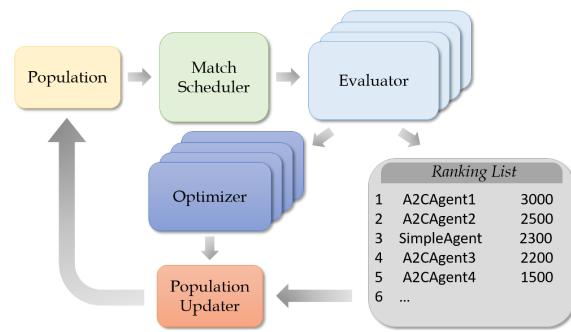


Abbildung 4.7.: Infrastruktur des COMBAT Frameworks (entnommen aus [PPYG18])

## MAGNet

Malysheva et al. [MSS<sup>+</sup>18] stellten in ihrer Arbeit ein Verfahren namens *MAGNet* vor. MAGNet trainiert für jeden Agenten ein neuronales Netz, um einen sog. „Relevance Graph“ in Form einer  $|\mathcal{D}| \times (|\mathcal{D}| + |\mathcal{E}|)$  großen Matrix zu generieren.  $\mathcal{D}$  entspricht der Anzahl an Agenten und  $\mathcal{E}$  der maximalen Anzahl an Objekten des Environments. Die Netze verwenden für die Erzeugung der Graphen einen Self-Attention Mechanismus. Die Graphen repräsentieren Beziehungen zwischen Agenten untereinander und zu Objekten des Environments und sollen helfen zu erkennen, was wichtig für den Sieg ist. Eine Methode, die durch NerveNet [WLBF18] inspiriert ist, generiert im nächsten Schritt auf Basis des Relevance Graphs eine akkumulierte Summe an Nachrichten. Ein weiteres neuronales Netz entscheidet dann auf Basis der Nachrichten und der aktuellen Observation des Agenten, welche Aktion ausgeführt werden soll. Alle Netze wurden mit der *Deep Deterministic Policy Gradient* Actor-Critic Methode trainiert.

Alles Experimente wurden innerhalb des Pommerman Team Environments gegen einen heuristischen Standardagenten ausgetragen. Nach einer Trainingsperiode von 50000 Episoden betrug die Gewinnrate von MAGNet  $71,3 \pm 0,7\%$ . MAGNet übertraf damit State-Of-The-Art Multi-Agenten RL Methoden wie MCTS, MADDPG und auch DQN.

## Backplay

Deep Reinforcement Learning kann vor allem in Environments mit spärlicher Reward-funktion (e.g. Pommerman) einige Zeit dauern; der Agent benötigt viele Episoden, um

#### 4. Verwandte Arbeiten

positive Rewards zu erhalten und etwas lernen zu können. Resnick et al. [RRK<sup>+</sup>18] entwickelten eine Methode namens *Backplay*, die deutlich weniger Episoden benötigt. Dadurch wird der Lernprozess des Agenten beschleunigt. Die Idee ist, die Trajektorie einer guten (aber nicht unbedingt optimalen) Episode im Verlauf des Trainings rückwärts bis zum Startzustand zu traversieren. Das Training beginnt nicht in dem Startzustand  $s_0$  der Demonstration, sondern in der Nähe des Endzustands  $s_T$ . Der Agent bewegt sich nun ausgehend von diesem Zustand entlang der Trajektorie, bis er den Endzustand erreicht. Währenddessen wird untersucht, ob angrenzende Positionen/Zustände schneller zum Ziel führen. Nach Abschluss der Episode wird der Startpunkt weiter zurück gesetzt und die Demonstration erneut durchlaufen. Dieser Vorgang wird wiederholt, bis der Startzustand erreicht wurde.

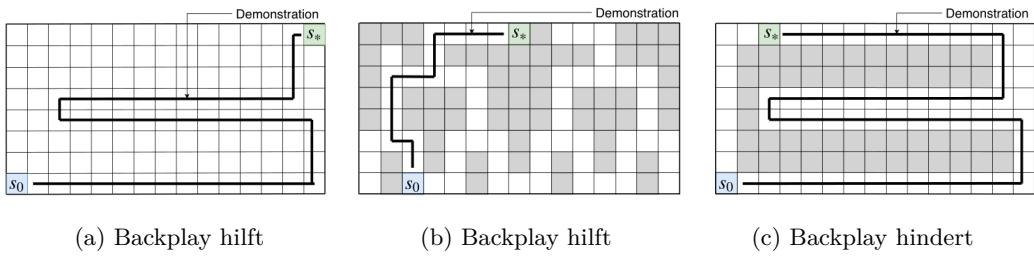


Abbildung 4.8.: Qualitative Analyse verschiedener Demonstration für Backplay in einer Labyrinthdomäne (entnommen aus [RRK<sup>+</sup>18]). In (c) verhindert die Demonstration den schnelleren Weg über die linke Kante zu finden.

Resnick et al. testeten Backplay in Kombination mit PPO unter anderem in der FFA Konfiguration von Pommerman. Das Training erfolgte gegen drei **Agent47Agents** (vgl. Abschnitt 4.3.1). Die Demonstrationen stammten von Spielen des **Agent47Agents**. Die Experimente zeigten, dass Backplay im Gegensatz zu Vanilla PPO in der Lage ist, die Gegner erfolgreich zu bewältigen. Die Autoren weisen allerdings explizit darauf hin, dass der Erfolg von Backplay stark von der Demonstration abhängt. Abbildung 4.8 illustriert, wann Backplay hilfreich oder hinderlich ist.

#### Proximal Policy Optimization vs. Deep Q-Learning from Demonstrations

Shah et al. [SST18] analysierten die Performance von PPO in der Pommerman FFA Konfiguration. Ihre Untersuchungen waren motiviert durch die Tatsache, dass PPO eine geeignete Strategie für Domänen mit großen Zustands- und Aktionsräumen ist. Die Evaluation zeigte, dass der Agent eine defensive Spielstrategie lernte. Das Legen von Bomben wurde vermieden, die Gegner wurden nicht attackiert. Der Agent konnte im Schnitt 17,5% der Spiele gegen drei **SimpleAgents** gewinnen.

Darüber hinaus implementierten die Autoren einen weiteren Agenten unter Verwendung von *Deep Q-Learning from Demonstrations* (DQfD)<sup>14</sup>. Der Agent wurde anhand von 1000 Demonstrationen trainiert. Die Demonstrationen stammten von Spielen, in denen vier **SimpleAgents** gegeneinander antraten. DQfD performte im Gegensatz zu PPO deutlich offensiver. Der Agent konnte Bomben erfolgreich ausweichen, während er das Spielfeld

<sup>14</sup>DQfD ist eine Strategie, die auf Imitation Learning basiert.

effizient explorierte. Parallel dazu lernte er gegnerische Spieler in eine Ecke zu treiben und zu töten. Mit einer durchschnittlichen Gewinnrate von 22,6% schneidet er ähnlich gut ab wie ein `SimpleAgent`.

### Deep Q-Learning vs. Deep Q-Learning from Demonstrations

Singh et al. [SDS<sup>+</sup>19] evaluieren in ihrer Arbeit unter anderem die Performance von klassischem Deep Q-Learning im Vergleich zu DQfD. Die Demonstrationen stammen analog zu [SST18] von Spielen des `SimpleAgents`. Die Agenten wurden gegen drei `SimpleAgents` im FFA Modus von Pommerman getestet.

Die Autoren konnten zeigen, dass Deep Q-Learning zu ähnlichen Ergebnissen wie PPO führte (vgl. [SST18]). Der Agent lernte weder das Spielfeld zu explorieren noch Bomben zu legen. Stattdessen versteckte er sich in den Ecken des Spielfelds oder zwischen Wänden, um nicht zu sterben. Die Ergebnisse von DQfD deckten sich mit denen von [SST18]. Der Agent lernte erfolgreich das Spielfeld zu erkunden und Bomben zu legen. Das Performance Level entsprach dem eines `SimpleAgents`.



# 5. Herangehensweise

Für diese Arbeit wurde ein Team aus zwei Agenten für die „Team Radio“-Variante in Pommerman implementiert und mittels Deep Recurrent Q-Learning und Value Decomposition trainiert. Aufgrund des partiell beobachtbaren Environments wurden rekurrente Netze verwendet. Jedes Netz realisiert Experience Replay<sup>1</sup>, Fixed Q-Targets und Double DQL, sowie eine Dueling Network Architektur (vgl. Unterabschnitt 2.3.2). Das Training erfolgte via VD, um die Leistungen innerhalb des Teams besser zuordnen zu können. Anschließend wurde das Kommunikationsprotokoll der Agenten aus dezentral ausgeführten Tests anhand ausgewählter Metriken analysiert. Das bedeutet Value Decomposition wird nur während den Trainingsphasen verwendet. Für die Zusammensetzung der Q-Values wurde kein zusätzliches neuronales Netz (VDN) verwendet, da in dieser Arbeit Agenten eines Teams homogen sind [VVB<sup>+</sup>20].

## 5.1. Modellaufbau

Insgesamt wurden zwei Teammodelle implementiert. Das erste Modell (das NoCom-Team) kann Aktionen ausführen, aber nicht kommunizieren; Agenten dieses Teams verschicken in jedem Zeitschritt die Nachricht  $[0, 0]$ . Lernen erfolgt anhand vorverarbeiteter Daten zum Spielfeld, die Nachricht des Teammitglieds wird nicht verwendet. Agenten des zweiten Modells (das Com-Team) sind zusätzlich in der Lage miteinander zu kommunizieren.  $[0, 0]$  wird als Nachricht nur verschickt, wenn der Agent bereits tot ist. Sowohl die Informationen zum Spielfeld, als auch die Nachricht des Teammitglieds werden für den Lernprozess verwendet.

### 5.1.1. Datenvorverarbeitung

Alle Informationen, die das Spielfeld betreffen, erhalten die Agenten in einem  $11 \times 11 \times 1$  großen Array. Zur Verarbeitung der wichtigsten Informationen, werden relevante Observations der Agenten vor jeder Entscheidung zusammengefasst. Spielfeld, Explosionsradius, Lebensdauer und Bewegungsrichtung der Bomben, sowie Lebensdauer der Flammen werden überlagert; es entsteht ein  $11 \times 11 \times 5$  großes Array. Agenten des Com-Teams verarbeiten zusätzlich die Nachricht des Teammitglieds in Form eines One-Hot Encodings.

### 5.1.2. Architektur ohne Kommunikation

Das NoCom-Team verwendet insgesamt zwei DRQNs zur Verarbeitung der Informationen des Spielfelds. Jeder Spieler besitzt ein Aktionsnetz, das wie folgt aufgebaut ist:  
Der  $11 \times 11 \times 5$  große Input wird über vier aufeinander folgende Convolutional Layer mit

<sup>1</sup>mit Erfahrungstupel der Form  $e_t = (o_t, u_t, r_{t+1}, o_{t+1})$ , wobei  $u_t$  sich aus Aktion  $a_t$  und *der eigenen* Nachricht  $m_t$  zusammensetzt. Das Ausschalten von Experience Replay führte zu überangepassten Modellen. Dieses Ergebnis deckt sich mit denen in [FNF<sup>+</sup>17].

## 5. Herangehensweise

den Dimensionen  $11 \times 11 \times 16$ ,  $11 \times 11 \times 32$ ,  $11 \times 11 \times 64$  und  $11 \times 11 \times 64$  verarbeitet. Die Größe der Filtermatrizen beträgt  $3 \times 3$ , Schrittgröße und Padding sind jedes Mal gleich 1. Padding wurde so gewählt, dass die Größe des Inputs nach jeder Konvolution unverändert bleibt, da  $11 \times 11$  bereits sehr klein ist. Jedes Layer nutzt ReLUs zur Aktivierung. Die Ausgabe wird geglättet und an ein fully-connected Layer mit 128 ReLUs weitergeleitet. Anschließend verarbeitet ein LSTM mit 128 Zellen den Output auf Basis seines vorherigen Hidden States. Das Layer verwendet standardmäßig tanh als Aktivierungsfunktion. Das Netz wird anschließend in zwei separate Value und Advantage Ströme geteilt, die je zwei fully-connected Layer enthalten mit ReLU Aktivierung dazwischen. Das erste Layer beider Ströme umfasst 128 Neuronen. Das zweite Layer des Advantage Stroms besteht aus 6 Neuronen (eines für jede Aktion), das des Value Stroms aus einem. Der Output beider Ströme wird unter Verwendung des Durchschnitts (vgl. (2.25)) zusammengefasst; ein Q-Value für jede mögliche Aktion wird berechnet, auf dessen Basis die Aktionsauswahl erfolgt. Abbildung 5.1 zeigt den Aufbau eines Aktionsnetzes graphisch. Während des Trainings werden die Q-Values der ausgewählten Aktionen addiert; basierend auf dieser Summe werden beide Netze zentral optimiert (vgl. Unterabschnitt 2.3.4). Die gesamte Architektur des Teams ist in Abbildung 5.2 dargestellt.

### 5.1.3. Architektur mit Kommunikation

Die Grundidee des Com-Modells ist die Verwendung von zwei separaten DRQNs für die Aktions- und Nachrichtenauswahl. Das Team setzt insgesamt vier Netze ein. Pro Agent werden zwei Netze verwendet: Ein DRQN<sub>Act</sub> produziert für jeden Agenten die Q-Values seiner Aktionen. Das Netz erhält neben den Spielfelddaten zusätzlich die Nachricht des Teammitglieds (in Form eines One-Hot Encodings) als Eingabe. Ein separates DRQN<sub>Com</sub> berechnet auf Basis der Spielfeldinformationen die Q-Values der Wörter 0 bis 8. Für die Wahl der Nachricht sind nur die Q-Values der Wörter 1 bis 8 relevant; eine Nachricht setzt sich aus den zwei Wörtern mit den höchsten Q-Values zusammen. Beide DRQNs bestehen aus denselben Grundbausteinen wie die Netze des NoCom-Teams. Folgende Änderungen wurden vorgenommen:

**DRQN<sub>Com</sub>** Das letzte Layer des Advantage Stroms besteht nicht aus 6, sondern 9 Neuronen (eines für jedes Wort, vgl. Abbildung 5.1).

**DRQN<sub>Act</sub>** Vier Convolutional Layer und ein fully-connected Layer verarbeiten nach wie vor die Informationen des Spielfelds. Die Nachricht des Teammitglieds wird parallel einem zusätzlichen fully-connected Layer mit 128 ReLUs übergeben. Die Ausgaben beider fully-connected Layer werden addiert und bilden den neuen Input des LSTMs. Abbildung 5.3 illustriert das Modell.

Größe und Aktivierungsfunktionen jedes Layers entsprechen (wenn nicht anders angegeben) dem NoCom-Modell.

Die Aktionsnetze werden separat von den Kommunikationsnetzen trainiert. Dafür werden die Q-Values der Aktionsnetze ( $Total_{Act}$ ) und die Q-Values der Kommunikationsnetze ( $Total_{Com}$ ) addiert. Das Training der Aktionsnetze erfolgt bezüglich  $Total_{Act}$ . Für das Training der Kommunikationsnetze wird  $Total_{Com}$  verwendet. Abbildung 5.4 zeigt die gesamte Architektur des Teams.

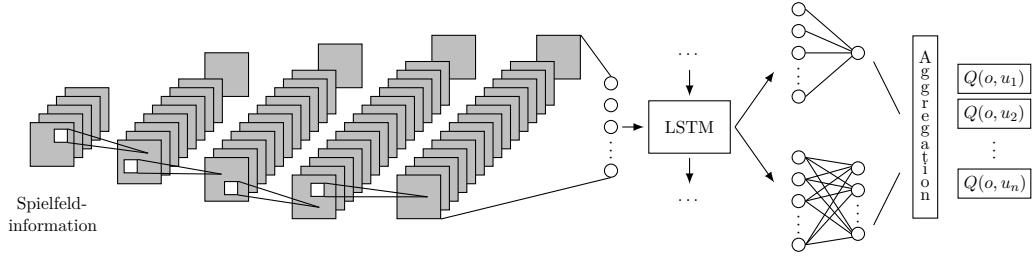


Abbildung 5.1.: Architektur der Aktionsnetze des NoCom-Teams mit  $n = 6$  oder der Kommunikationsnetze des Com-Teams mit  $n = 9$  (Illustration angelehnt an [Sim18]). Vier Convolutional Layer verarbeiten die Spielfeldinformation. Der geglättete Output wird an ein fully-connected Layer weitergeleitet, dem ein LSTM folgt. Die Ausgabe des LSTMs wird in zwei Ströme unterteilt (oben: Value-Strom, unten: Advantage Strom), die je aus zwei fully-connected Layer bestehen. Value- und Advantage-Strom werden aggregiert und produzieren die Q-Values der sechs Aktionen bzw. neun Wörter.

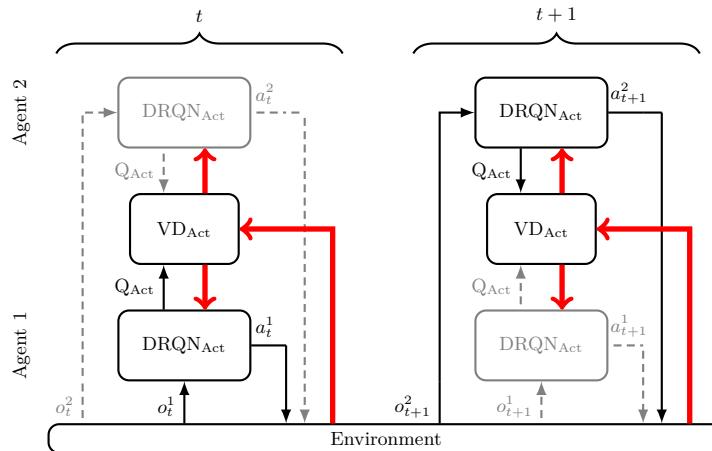


Abbildung 5.2.: Informationsfluss im NoCom-Team für zwei Zeitschritte (angelehnt an [FAdFW16, Abb. 1]). Beide Agenten wählen ihre Aktion auf Basis der Q-Values. Back Propagation (in rot dargestellt) erfolgt über die Summe der Q-Values der ausgewählten Aktionen.

## 5. Herangehensweise

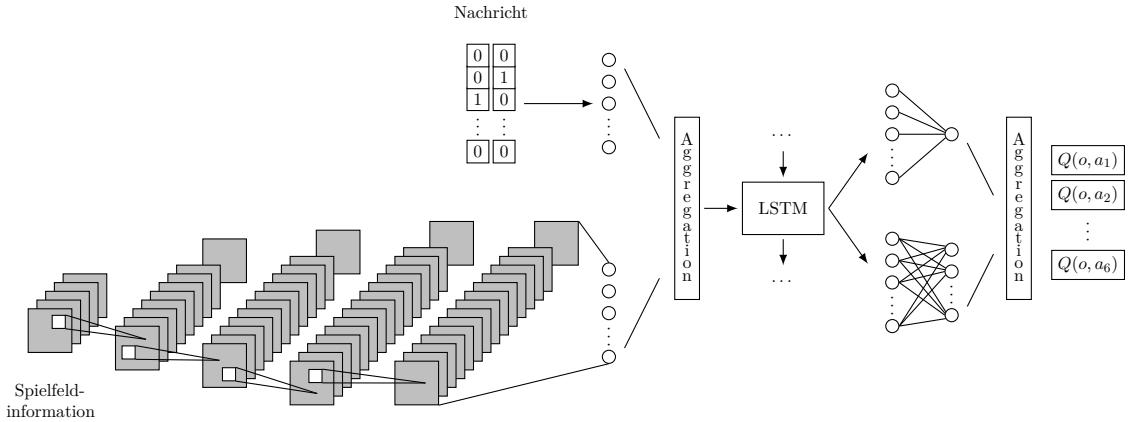


Abbildung 5.3.: Architektur der Aktionsnetze des Com-Teams (Illustration angelehnt an [Sim18]). Vier Convolutional Layer verarbeiten die Spielfeldinformation. Der geglättete Output wird an ein fully-connected Layer weitergeleitet. Parallel dazu wird die Nachricht des Teammitglieds (bestehend aus zwei Wörtern in Form eines One-Hot Encodings) von einem fully-connected Layer verarbeitet. Der Output beider fully-connected Layer wird aggregiert und einem LSTM übergeben. Die Ausgabe des LSTMs wird in zwei Ströme unterteilt (oben: Value-Strom, unten: Advantage Strom), die je aus zwei fully-connected Layern bestehen. Value- und Advantage-Strom werden aggregiert und produzieren die Q-Values der sechs Aktionen.

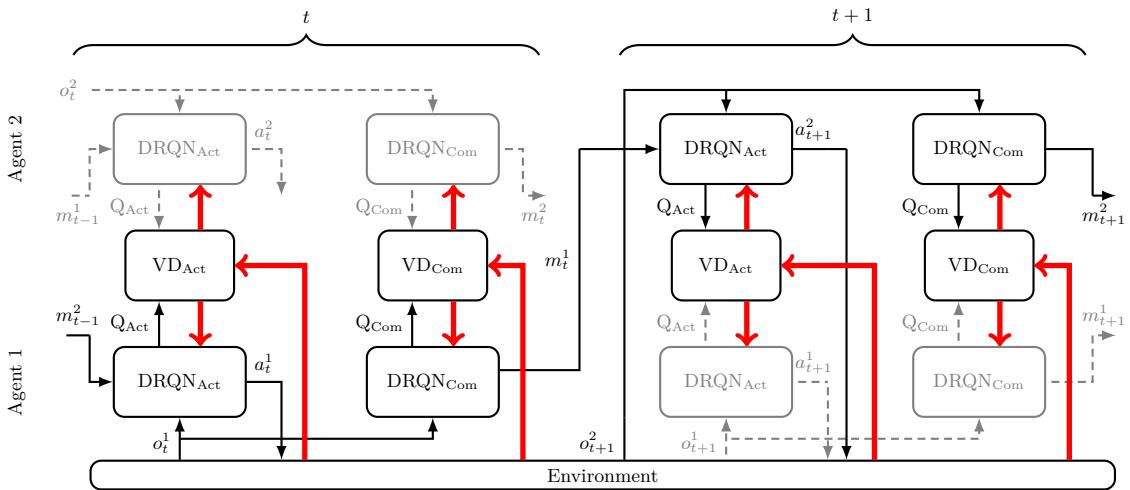


Abbildung 5.4.: Informationsfluss im Com-Team für zwei Zeitschritte (angelehnt an [FAdFW16, Abb. 1]). Beide Agenten wählen ihre Aktion und Nachricht unabhängig voneinander auf Basis der entsprechenden Q-Values. Die Gradienten der Aktionen und Nachrichten werden separat berechnet. Back Propagation (in rot dargestellt) erfolgt über die Summe der Q-Values der ausgewählten Aktionen bzw. Nachrichten.

## 5.2. Trainings- und Testsetup

Beide Modelle wurden jeweils gegen drei verschiedene Gegner unterschiedlichen Schwierigkeitsgrades trainiert: Ein Team aus `RandomAgents`, ein Team aus `RandomNoBombAgents` und ein Team aus `SimpleAgents`. Das Training fand jedes Mal unter denselben Bedingungen statt. Im Anschluss der Trainingsphase wurden alle Versionen der `NoCom`- und `Com`-Teams gegeneinander getestet. Folgendes Setup wurde verwendet:

### Trainingsphase

Jedes Team startet in der linken oberen und rechten unteren Ecke des Spielfelds. Die Modelle verwenden Adam als Lernalgorithmus mit der Default Step Size  $\alpha = 0.001$ . Das Training erfolgt über 10000 Episoden. Die Hidden States der DRQNs werden zu Beginn jeder Episode zurückgesetzt. Die Erfahrungstupel einer Episode werden gebündelt gespeichert. Das Replay Memory speichert insgesamt die 5000 aktuellsten Episoden. Während des Experience Replays wird eine Stichprobe von 32 Episoden aus dem Replay Memory entnommen; analog zu [RB19] wird von jeder Episode eine zufällige Sequenz von acht aufeinanderfolgenden Zeitschritten (Erfahrungstupeln, „Time Steps“) verwendet, auf denen die Berechnung der Gradienten erfolgt. Das `Com`-Team nutzt für beide Netzarten dieselbe Stichprobe. Die Gewichte der Q-Networks werden in jeder Episode nach jedem Zeitschritt aktualisiert, sobald genügend Episoden verfügbar sind. Anschließend werden die Gewichte der Target-Networks mit  $\tau = 0.001$  angepasst. Der Discountfaktor beträgt  $\gamma = 0.99$ . Sowohl die Wahl der Nachricht als auch die der nächsten Aktion erfolgen  $\epsilon$ -greedy mit  $\epsilon = 0.1$ . Um das Training zu beschleunigen, wurde für die Exploration der Aktionsfilter des `Skynet955`-Teams<sup>2</sup> verwendet (vgl. Abschnitt 4.3.2). Gleichzeitig wurde die Rewardfunktion so angepasst, dass alle Agenten bei einem Unentschieden einen Reward von 0 bekommen. Pro Team werden 18 unabhängige Durchläufe à 10000 Episoden gestartet.

Die Werte der Parameter  $\gamma$ ,  $\tau$  und Time Step wurden durch Hyperparameter Tuning (Grid Search Methode) bestimmt.

### Testphase

Um die Performance der Modelle richtig einschätzen zu können, ist  $\epsilon$  während der Testdurchläufe gleich 0, das heißt der Aktionsfilter wird nicht weiter verwendet. Der Reward bleibt gleich, d.h. ein Unentschieden wird weiterhin mit 0 bepunktet.<sup>3</sup> Insgesamt werden für jede Konstellation 200 unabhängige Episoden ausgeführt. Für jede Episode wird zufällig eines der 18 Modelle pro Version ausgewählt, um zu testen, wie robust die Implementierung ist. Alle Tests wurden zwei Mal durchgeführt, sodass jedes Team einmal in den Positionen des Trainings starten kann.

## 5.3. Metriken zur Analyse der Kommunikation

Kommunikation hat zwei Voraussetzungen [LFB<sup>+</sup>19]:

---

<sup>2</sup>Verfügbar auf <https://github.com/BorealisAI/pommerman-baseline>.

<sup>3</sup>Das spielt lediglich für die Auswertung der Performance im nachfolgenden Kapitel eine Rolle, da die Agenten den Reward in der Testphase nicht nutzen.

## 5. Herangehensweise

**Positive Signaling** Ein Agent muss ein Signal produzieren, das in irgendeiner Weise mit seinen Observationen oder geplanten Aktionen korreliert.

**Positive Listening** Ein anderer Agent muss nach Erhalt des Signals seinen Belief State aktualisieren oder sein Verhalten ändern.

Es ist wünschenswert, dass beide Agenten sowohl über ihre Umwelt oder Intentionen sprechen als auch die Nachrichten anderer interpretieren können und entsprechend darauf reagieren. Es ist möglich, dass beides unabhängig voneinander auftritt [EBL<sup>+</sup>19]: Der Sprecher kann Informationen über seinen Zustand teilen, ohne dass der Zuhörer diese Informationen verwendet; der Zuhörer kann seine nächste Aktion abhängig von der Nachricht bestimmen, ohne dass der Sprecher ein gutes Kommunikationsprotokoll erlernt hat. Letzteres ist natürlich wenig sinnvoll und gilt es zu vermeiden.

Das Kommunikationsverhalten des Com-Teams wird im Verlauf des Trainings und in den Tests anhand folgender Metriken analysiert:

- **Reward und Task Completion** Vergleich der Performance des NoCom-Teams und des Com-Teams. Zeigt, ob das Hinzufügen eines Kommunikationskanals den Reward erhöht. Misst Positive Signaling und Positive Listening.
- **Zusammenhang zwischen Nachrichten und Spielzuständen** Interpretiert, was der Agent sagt, d.h. misst Positive Signaling. Wird anhand der Verteilung der Nachrichten innerhalb bestimmter Situationen/Zustände bewertet.
- **Zusammenhang zwischen Nachrichten und Aktionen**
  - **Speaker Consistency** (SC) Liefert den Zusammenhang zwischen der Nachricht eines Agenten und seiner nächsten Aktion, d.h. misst Positive Signaling:

$$SC = \sum_{a \in \mathcal{A}} \sum_{m \in \Sigma} p(a, m) \log \frac{p(a, m)}{p(a)p(m)}, \quad (5.1)$$

wobei  $p(a, m) = \frac{1}{N} \sum_{i=1}^N \mathbb{1}_{\{\text{act}=a, \text{comm}=m\}}$  empirisch berechnet wird, indem man über (Wort, Aktion) Kookkurrenzen pro Durchlauf mittelt.

- **Instantaneous Coordination** (IC) Liefert den Zusammenhang zwischen der Nachricht eines Agenten und der nächsten Aktion des *anderen* Agenten, d.h. misst Positive Listening. Wird analog zu SC berechnet.

Da SC und IC auf Basis eines Wortes berechnet werden und eine Nachricht in Pommerman aus zwei Wörtern besteht, ist die Aussage der Werte verfälscht. Sind SC und/oder IC niedrig, bedeutet das nicht, dass kein Zusammenhang zwischen Nachricht und Aktion besteht. Das muss bei der Betrachtung der Darstellungen berücksichtigt werden und wird bei der Evaluation erneut ausführlich erklärt.

# 6. Ergebnisse und Evaluation

Folgendes Kapitel evaluiert die Ergebnisse des Trainings und der Testdurchläufe jedes Modells. Abbildungen zeigen, sofern nicht anders angegeben, die Daten mit einem 95%-Konfidenzintervall. Zur Übersichtlichkeit wird in allen Darstellungen der Zusatz „Agent“ in den Namen der Agenten weggelassen; Indices geben den Gegner während des Trainings an: R = `RandomAgent`, NB = `RandomNoBombAgent` und S = `SimpleAgent`.

## 6.1. Reward und Task Completion mit und ohne Kommunikation

Der Reward und das Bewältigen der Domäne sind wie erwähnt nicht die aussagekräftigsten Parameter für die Messung von Positive Signaling und Positive Listening [LFB<sup>+</sup>19]. Ein Anstieg des Rewards durch Hinzufügen eines Kommunikationskanals ist allerdings ein Indiz dafür, dass der Kanal in irgendeiner Weise sinnvoll genutzt wird, i.e. dass Kommunikation entstanden ist.

### 6.1.1. Lernkurven im Vergleich

Abbildung 6.1 zeigt den durchschnittlichen Reward im Verlauf des Trainings für beide Modelle gegen die verschiedenen Gegner. Bei der Betrachtung jeder Lernkurve müssen vorab einige Dinge beachtet werden:

Der `RandomAgent` ist ein Agent mit extrem hoher „Selbstmordrate“. Durch die zufällige Aktionsauswahl wird er sich mit großer Wahrscheinlichkeit schnell selbst zerstören. Spielverläufe von Episoden, in denen `RandomAgents` involviert sind, zeigen, dass die Agenten nicht in der Lage sind, ihre Gegner zu zerstören, bevor sie sich selbst töten. `RandomAgents` können nur gewinnen, wenn alle Gegner durch ihre eigenen Bomben zuerst sterben. Ein Unentschieden tritt extrem selten auf. Der `RandomAgent` ist demnach ein sehr einfacher Gegner.

Der `RandomNoBombAgent` ist im Vergleich dazu deutlich anspruchsvoller. Zu einem Selbstmord kommt es nie. Ein Sieg gegen den `RandomNoBombAgent` ist nur möglich, wenn man den Agenten zerstört. Der Agent ist sehr passiv. Das Spiel endet in einem Unentschieden, wenn man nicht gegen den Agenten vorgeht und sich nicht selbst tötet.

Der `SimpleAgent` ist von allen drei Agenten die größte Herausforderung, da er im Gegensatz zu den anderen aggressiv gegen seine Gegner vorgeht. Ein Vorteil gegenüber dem `RandomNoBombAgent` ist, dass der `SimpleAgent` sich selbst zerstören kann (s.o. Kapitel 3.4).

#### Auswertung gegen `RandomAgents`

Im Training gegen den `RandomAgent` (vgl. Abbildung 6.1a) startet das Com-Team mit einem durchschnittlichen Reward, der um 0,2 geringer ist als der des NoCom-Teams. Com-

## 6. Ergebnisse und Evaluation

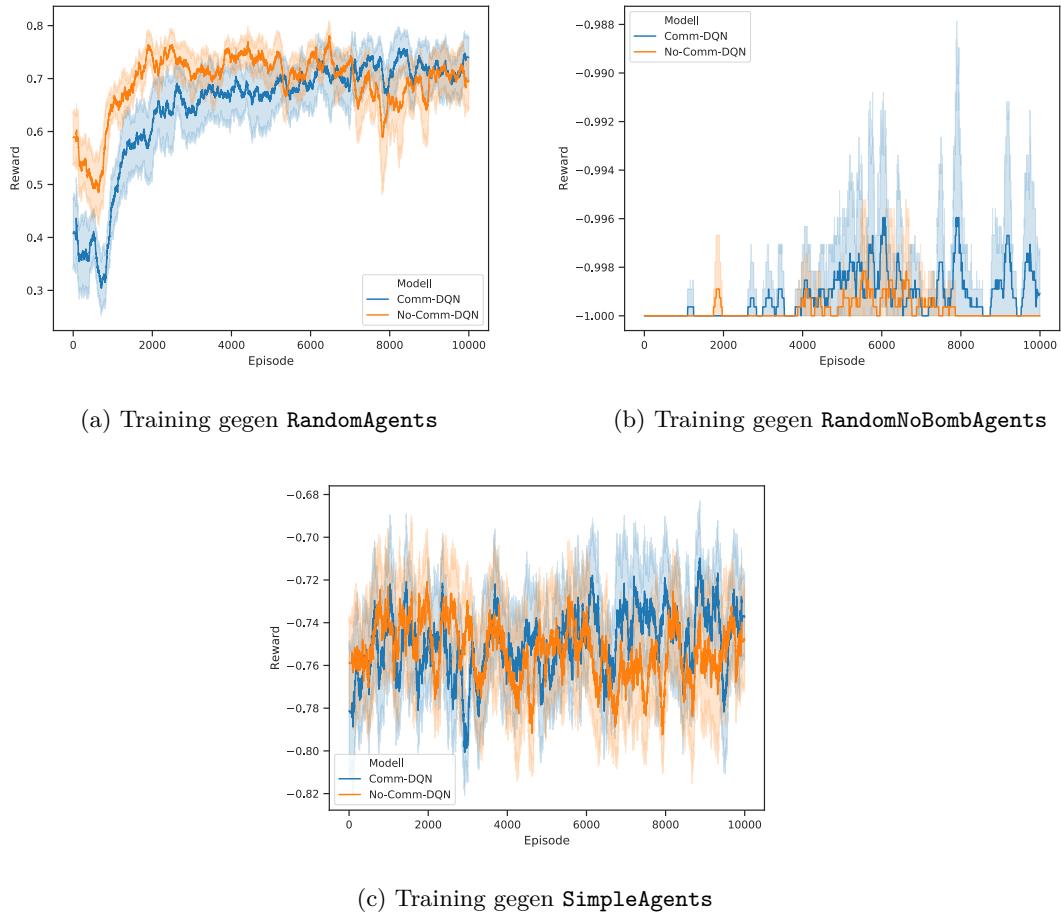


Abbildung 6.1.: Durchschnittlicher Reward im Verlauf des Trainings gegen verschiedene Gegner.

Agenten scheinen sich demnach zu Beginn des Trainings häufiger selbst zu zerstören als NoCom-Agenten. Beide Modelle können mit einem durchschnittlichen Reward von 0,4 und 0,6 bereits zu Beginn des Trainings relativ hohe Gewinnraten vorweisen. Der durchschnittliche Reward beider Modelle verringert sich jedoch rapide (i.e. in den ersten 500 – 1000 Episoden). Außerdem ist deutlich zu erkennen, dass der Lernprozess des Com-Teams im Vergleich zu dem des NoCom-Teams um einiges langsamer verläuft: Erst ab  $\sim$  Episode 6500 erzielt das Com-Modell mit einem durchschnittlichen Reward knapp über 0,7 ein ähnlich gutes Ergebnis, wie es das NoCom-Modell bereits ab  $\sim$  Episode 2000 erreichten konnte. Das ist vermutlich darauf zurückzuführen, dass die Agenten über den Kommunikationskanal am Anfang zufällige bzw. nicht optimierte Daten übertragen, die die Aktionsauswahl und den Lernprozess beeinflussen und in diesem Fall stören. Der weitere Verlauf der Lernkurve deutet darauf hin, dass das Com-Modell dafür deutlich stabiler lernt. Bis auf vier Einbrüche in der Performance (am Anfang, um Episode 3000 und kurz vor und kurz nach Episode 8000) steigt der Reward kontinuierlich an, und das lässt vermuten, dass eine längere Trainingszeit zu noch besseren Ergebnissen führen könnte. Die Leistung des

## 6.1. Reward und Task Completion mit und ohne Kommunikation

NoCom-Modells beginnt im Gegensatz dazu ab Erreichen eines durchschnittlichen Rewards von  $\sim 0,75$  (ca. bei Episode 2000) zu oszillieren. Der durchschnittliche Reward sinkt zwischen Episode 6000 – 7500 sogar bis auf  $\sim 0,6$ . Kommunikation scheint die Performance in der Konstellation gegen zwei RandomAgents zu stabilisieren.

Eine stichprobenartige Analyse der Spiele ergab, dass sowohl NoCom-Agenten als auch Com-Agenten lernten, zu warten, bis sich beide RandomAgents selbst zerstören. Bis auf vereinzelte Bewegungen bleiben die Agenten jeweils auf einer Position, ohne eine Bombe zu legen.

### Auswertung gegen RandomNoBombAgents

Das Training gegen RandomNoBombAgents (vgl. Abbildung 6.1b) verlief wie zu erwarten deutlich schlechter. Der durchschnittliche Reward beider Teams blieb durchgehend nahezu  $-1$ . Zwar konnte vor allem das Com-Team die Spiele vereinzelt mit einem Unentschieden beenden, insgesamt scheitern allerdings beide Modelle daran, in dieser Zeit zu lernen, die Gegner zu besiegen. Zu beachten gilt, dass der RandomNoBombAgent im Grunde genommen kein schwieriger Gegner ist. Tatsächlich müssen die Modelle zuerst lernen, dass sie durch Bomben sterben können, was zu einem negativen Reward führt. Anschließend muss allerdings gelernt werden, dass der RandomNoBombAgent ohne Legen von Bomben nicht besiegt werden kann. Jedes Spiel würde ansonsten immer in einem Unentschieden enden. Erst nach Erlangen dieses Verständnisses, kann erlernt werden, wie der Gegner am effektivsten zerstört werden kann (e.g. durch das Einsammeln von Powerups). Das Verhalten der Agenten in den Testdurchläufen zeigt, dass die erste „Lernphase“ bereits erreicht werden konnte: Das Modell nach Episode 10000 legt deutlich weniger Bomben. Diese Tatsache führt zu dem Schluss, dass ein Training über 10000 Episoden lediglich zu kurz war, um zu lernen, den RandomNoBombAgent effektiv zu besiegen. Ein Vergleich zwischen der Trainingszeit in dieser Arbeit und derer in verwandten Arbeiten verstärkt die Vermutung.

### Auswertung gegen SimpleAgents

Die Lernkurven beider Modelle gegen den SimpleAgent (vgl. Abbildung 6.1c) starten bei einem durchschnittlichen Reward von  $-0,78$  bzw.  $-0,76$ . Dieser Wert kann (analog zu dem Training gegen RandomAgents) auf Selbstmorde der Gegner zurückgeführt werden – ein SimpleAgent zerstört sich im Gegensatz zu einem RandomAgent nur viel seltener selbst. Dass die initialen Rewards des Com-Teams und NoCom-Teams zu Beginn nahezu gleich sind (anders als gegen RandomAgents), kann daran liegen, dass die Lebenserwartung des SimpleAgents höher ist als die beider Modelle. Von Anfang an oszillieren beide Lernkurven mit kleiner Amplitude um einen ähnlichen Reward. Es ist kein dauerhafter Anstieg des Rewards erkennbar. Das deutet darauf hin, dass die Modelle ständig ihre Strategien ändern oder sich (zu) schnell an eine Strategie anpassen, die nicht der optimalen entspricht. In diesem Fall ist das eine Anpassung an den positiven Reward, der schnell und leicht bei einem Selbstmord des SimpleAgents erzielt wird: Die Modelle lernen zu warten und greifen nicht an, bis der Agent sich selbst tötet. Der SimpleAgent ist allerdings kein Gegner, der durch diese Strategie erfolgreich besiegt werden kann. Com- und NoCom-Agenten müssen zumindest lernen, vor dem gegnerischen Team zu fliehen, sobald

## 6. Ergebnisse und Evaluation

dieses sich in der Nähe befindet. Nur dann kann das Warten auf den Selbstmord des **SimpleAgents** eine effektive Strategie sein. Es kann nicht sicher gesagt werden, ob das mit den Modellen aus dieser Arbeit durch eine längere Trainingsdauer im Rahmen des Möglichen wäre.

### 6.1.2. Gewinnraten im Vergleich

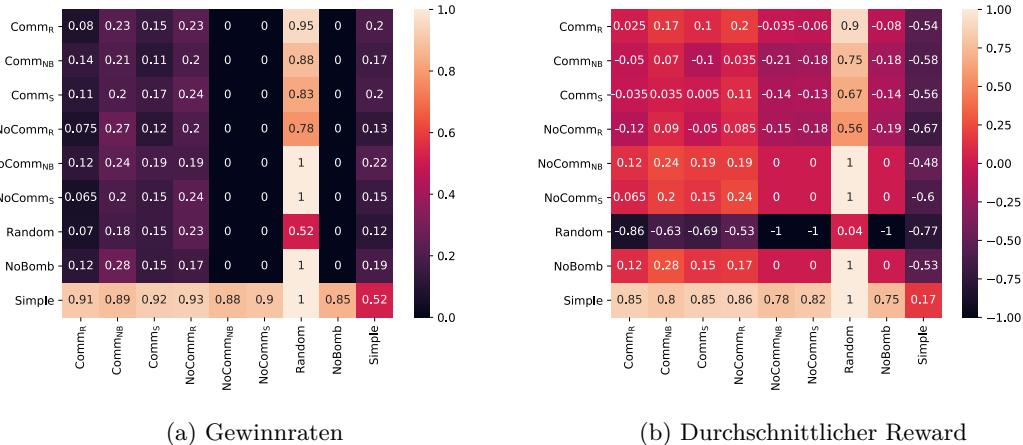


Abbildung 6.2.: Finale Performance aller Modelle im Vergleich. Alle Tests wurden zwei Mal durchgeführt, sodass die Performance jedes Teams in beiden Startpositionen evaluiert werden kann. Zeilen enthalten die Teams, die in den „normalen“ Positionen starteten. (a) visualisiert die Gewinnraten aus Sicht der Teams in den Zeilen, (b) visualisiert analog den durchschnittlichen Reward.

Abbildung 6.2 visualisiert die finale Performance aller Modelle (nach Episode 10000) und Agenten aus Pommerman im Vergleich. Tabelle A.1a im Anhang enthält genaue Angaben über die Zahl der Siege, Unentschieden und Niederlagen. Darin ist zu erkennen, dass die Performance der trainierten Modelle unabhängig von der Startposition ist.

Es überrascht nicht, dass das **ComRandom**-Team im Vergleich zu seinen Variationen am besten gegen seinen Gegner aus dem Training performte: **ComNoBomb** und **ComSimple** verloren gegen den **RandomAgent** durchschnittlich ca. 20 Spiele mehr. Das **ComRandom**-Modell erzielte in beiden Startpositionen eine Gewinnrate von über 90% und damit sogar ein besseres Ergebnis als am Ende der Trainingsphase. Das ist darauf zurückzuführen, dass  $\epsilon$  während des Trainings durchgehend 0,1 ist und in den Tests gleich 0. In der Explorationsphase wurde während des Trainings zwar der Aktionsfilter verwendet, die Agenten befanden sich allerdings nie in der Nähe des Teammitglieds, weswegen das Legen von Bomben nicht gefiltert wurde. Aufgrund des niedrigen Wertes von  $\epsilon$ , konnte der Aktionsfilter in nachfolgenden Schritten selten den Selbstmord des Agenten verhindern. Da die Agenten nicht lernten, Bomben und Flammen auszuweichen, führte das zu ihrem Tod. Mit  $\epsilon = 0$  tritt diese Situation nicht auf.

### 6.1. Reward und Task Completion mit und ohne Kommunikation

Das kommunikationsfähige Modell performte ebenfalls besser als sein nicht kommunikationsfähiges Pendant. Das **NoCom<sub>Random</sub>**-Team verlor im Vergleich 34 Spiele mehr. Mit einem durchschnittlichen Reward von 0,56 liegt das Modell damit deutlich unter dem zuletzt erreichten Reward gegen Ende des Trainings. Demnach erzielten die Agenten sogar schlechtere Ergebnisse als die kommunikationsfähigen Modelle, die nicht gegen den **RandomAgent** trainiert wurden. Eine Analyse einzelner Episoden, in denen **Com<sub>Random</sub>**- bzw. **NoCom<sub>Random</sub>**-Agenten gegen **RandomAgents** antreten, zeigt auf den ersten Blick keine signifikanten Unterschiede im Verhalten der Modelle.

Episoden, in denen das **Com<sub>Random</sub>**-Team gegen sich selbst spielte, verliefen sehr ausgewogen. 173 Spiele endeten in einem Unentschieden, ansonsten gewann das Team in der „gewöhnlichen“ Startposition vier Spiele mehr als das Team, das in den anderen Ecken startete. Die Spiele gegen das **NoCom<sub>Random</sub>**-Team endeten ebenfalls größtenteils in einem Unentschieden (73%), das kommunikationsfähige Modell gewann allerdings durchschnittlich 31 Spiele mehr. Ein Indiz dafür, dass das **Com<sub>Random</sub>**-Modell, das Legen von Bomben stärker vermeidet und demnach seltener durch eigene Bomben stirbt.

**NoCom**-Agenten, die gegen **RandomNoBombAgents** oder **SimpleAgents** trainiert wurden, konnten in Spielen gegen den **RandomAgent** die Ergebnisse aller anderen übertreffen: Beide Modelle gewannen 200 von 200 Spielen. Beide **NoCom**-Varianten konnten darüber hinaus von niemandem außer den **SimpleAgents** besiegt werden. Entsprechende **Com**-Teams verloren hingegen durchaus Partien, obwohl die Lernkurven vermuten ließen, dass beide Modelle ähnlich performen. Eine mögliche Ursache könnte sein, dass die entsprechenden **Com**-Modelle im Training deutlich weniger optimiert werden konnte als die **NoCom**-Modelle. **NoCom**-Agenten lernten bereits das Legen von Bomben eher zu vermeiden. Das deckt sich mit der Gewinnrate von 100% gegen den **RandomAgent**. Entsprechende **Com**-Agenten konnten im Gegensatz dazu ihre Aktionsauswahl innerhalb von 10000 Episoden nicht mit gleichem Erfolg optimieren.

Gegen **RandomNoBombAgents** erzielen sowohl **Com<sub>NoBomb</sub>**-Agenten als auch **NoCom<sub>NoBomb</sub>**-Agenten einen auffällig höheren Reward als während des Trainings. Zwar konnte keines der Modelle den Gegner besiegen, dennoch endete der Großteil der Spiele in einem Unentschieden. Das **Com**-Team erzielte mit 37 bzw. 55 Niederlagen einen Reward von -0,18 bzw. -0,28. Das nicht-kommunikationsfähige Modell performte sogar noch besser: Die Agenten überlebten in jedem Spiel bis zum Schluss, der Reward lag in beiden Konstellationen bei 0. Dieses Verhalten kann ebenfalls durch das Ausschalten der Exploration in der Testphase erklärt werden. Die Auswirkungen der Exploration machen sich gegen **RandomNoBombAgents** stärker bemerkbar, da sie sich nicht selbst zerstören. Das Legen einer Bombe ohne gelernt zu haben, dieser auszuweichen, führt viel eher zu einer Niederlage als im Training gegen **RandomAgents**, die sich zur selben Zeit selbst töten könnten.

Analog dazu erzielen **Com<sub>Simple</sub>**- und **NoCom<sub>Simple</sub>**-Agenten in den Tests einen höheren Reward gegen **SimpleAgents**, da sie durch längeres Überleben die Möglichkeit erhöhen, dass sich die Gegner selbst zerstören.

## 6.2. Qualitative Analyse der Kommunikation

Dieses Kapitel deckt durch die Analyse des Nachrichtenaustauschs zwischen  $\text{Com}_{\text{Random}}$ -Agenten nach Abschluss des Trainings etwaige Gründe für dessen stabilere Performance auf. Parallel dazu wird versucht, die Frage zu beantworten, warum sich  $\text{NoCom}_{\text{NoBomb}}$ -Agenten gegen  $\text{RandomNoBombAgents}$  besser schlagen als  $\text{Com}_{\text{NoBomb}}$ -Agenten, sowie Gründe dafür zu finden, weswegen  $\text{NoCom}_{\text{NoBomb}}$ - und  $\text{NoCom}_{\text{Simple}}$ -Agenten in Spielen gegen  $\text{RandomAgents}$  besser abschneiden als ihre kommunikationsfähigen Gegenstücke. Auf beide Fragen konnte jedoch keine zufriedenstellende Antwort gefunden werden.

### 6.2.1. Zusammenhang zwischen Nachrichten und Spielzuständen

Aus dem Umstand, dass die Nachrichtenauswahl der  $\text{Com}$ -Agenten auf Basis ihrer Observation erfolgt, folgt intuitiv, dass die Agenten lokale Informationen über den Kommunikationskanal teilen. Im Folgenden wird die Verteilung der Nachrichten in sechs verschiedenen Szenarien betrachtet, die aus menschlicher Sicht entscheidend für den Spielverlauf sind:

- (i) Eine Bombe befindet sich im Sichtfeld des Agenten.
- (ii) Eine Flamme befindet sich im Sichtfeld des Agenten.
- (iii) Ein Gegner befindet sich im Sichtfeld des Agenten.
- (iv) Der Agent befindet sich in Reichweite einer Bombe.
- (v) Der Agent befindet sich neben einem Gegner.<sup>1</sup>
- (vi) Ein Gegner befindet sich in einem Radius<sup>2</sup>  $\leq 3$ .

Es könnte demnach hilfreich sein, wenn ein Agent seinem Teammitglied signalisieren würde, dass er sich in einer solchen Situation befindet. Es werden nur die Nachrichten lebender Agenten betrachtet, weswegen die 0 in nachfolgenden Verteilungen fehlt.

#### Auswertung gegen RandomAgents

Abb. 6.3 zeigt die Verteilung der Nachrichten des  $\text{Com}_{\text{Random}}$ -Teams pro Szenario in den Spielen gegen  $\text{RandomAgents}$ . In keiner der 400 Episoden befanden sich die Agenten auch nur in der Nähe eines Gegners, weswegen Szenario (v) und (vi) in der Abbildung fehlen. Auffällig sind die großen Konfidenzintervalle der Nachrichten, die sich in nahezu jedem Szenario überlappen. Die Nachrichten symbolisieren die Zustände dem zu Folge nicht eindeutig.

Bei der Betrachtung ausgewählter Nachrichtenverläufe ist schnell erkennbar, dass die Agenten in jedem Schritt größtenteils dieselben zwei Wörter austauschen. Das könnte daran liegen, dass sich die Observationen beider Agenten nicht großartig unterscheiden (und sich nicht oft ändern), da sich beide Spieler ähnlich verhalten und in der Nähe ihrer Startposition bleiben. Zudem spielen Informationen über die lokale Umgebung des Teammitglieds für die gemeinsame Strategie in Episoden gegen  $\text{RandomAgents}$  nur bedingt eine Rolle. Die relative Häufigkeit der Szenarien zeigt, dass der Großteil der Spiele endet, ohne dass sich eine Bombe, eine Flamme oder ein Gegner in den Sichtfeldern der Agenten befand. Die Verteilung „Gegner im Sichtfeld“ deutet darauf hin, dass sich die Agenten

---

<sup>1</sup>Manhattan-Distanz = 1.

<sup>2</sup>gemessen mit Manhattan-Metrik  $\|r\|_1$ .

## 6.2. Qualitative Analyse der Kommunikation

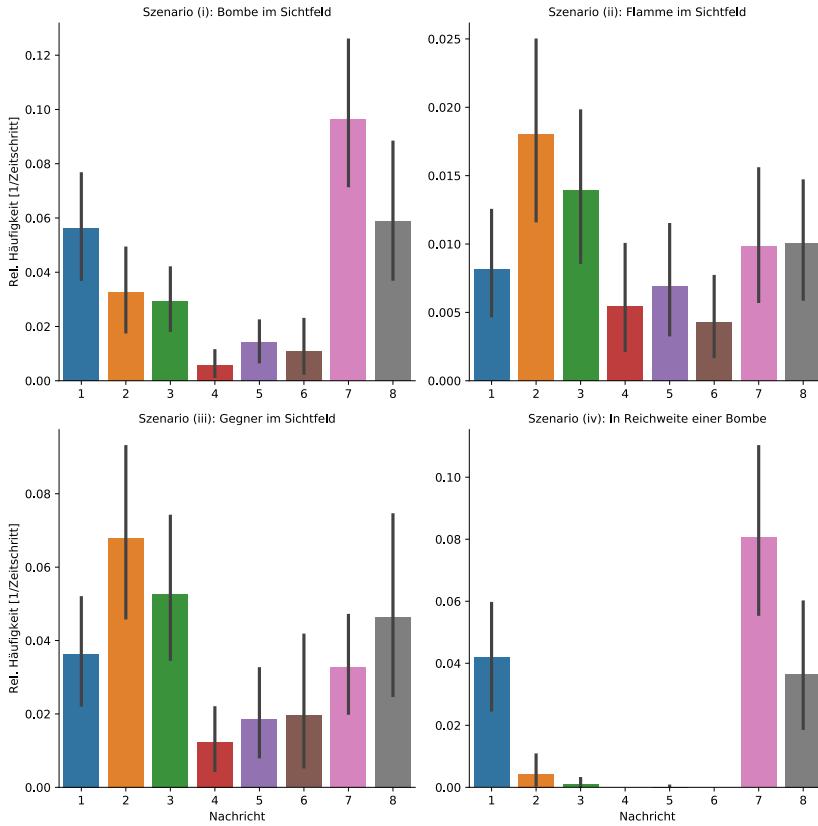


Abbildung 6.3.: Verteilung der Nachrichten von `ComRandom`-Agenten in bestimmten Szenarien aus Spielen gegen `RandomAgents`.

nicht weit bewegen.<sup>3</sup> Die Verteilung „In Reichweite einer Bombe“ untermauert, wie selten die Agenten Bomben legen. Die Daten, die zu diesem Szenario vorliegen, stammen mit großer Wahrscheinlichkeit von den Spielen, die das `Com`-Team verlor. Dass in diesem Zustand nur die Nachrichten 1, 3, 7 und 8 verschickt wurden, liegt lediglich daran, dass nur Agenten, die wiederholt diese Zahlen verschicken, in diese Situation kamen. Das erklärt auch, warum in Szenario (i) vermehrt die Nachrichten 1, 7 und 8 verschickt wurden. Ein Vergleich der Verteilungen (ii) und (vi) bestätigt, dass die Agenten nicht gelernt haben, Flammen auszuweichen: Die meisten Agenten wurden durch Flammen getötet, weswegen das Szenario nicht oft erfasst wurde. Allgemein kann beobachtet werden, dass 4, 5 und 6 im Vergleich zu den restlichen Nachrichten seltener verwendet wurden. Die meisten Agenten verwendeten die Nachrichten 1, 2, 3, 7 oder 8 (in beliebiger Kombination). Dass keiner der oben genannten Zustände im Verlauf des Spiels häufig vorkommt, macht deutlich, dass die Anpassung des Nachrichtenaustauschs auf bestimmte Situationen für die Strategie der Spieler nicht notwendig ist. Zudem ist wichtig zu beachten, dass mehrere Szenarien gleichzeitig auftreten können. Z.B.: Nach Ausführen der Aktion BOMB befindet sich die Bombe zwangsläufig im Sichtfeld des Agenten und der Agent in Reichweite der Bombe. Genauso gilt: Sobald ein `RandomAgent` in Sichtweite der Agenten ist, sind das

<sup>3</sup>Gegner könnten auch vor `Comm`-Agenten fliehen und aufgrund dessen nicht im Sichtfeld der Agenten erscheinen. Das ist für `RandomAgents` allerdings unwahrscheinlich.

## 6. Ergebnisse und Evaluation

(aufgrund der Verhaltensweise des `RandomAgents`) mit großer Wahrscheinlichkeit auch Bomben und/oder Flammen. Das erklärt, warum die Verteilung der Nachrichten in Szenario (iii) ähnlich wie in (ii) ist bzw. die Verteilung in (i) der in (iv) ähnelt.

Zu beachten gilt außerdem, dass die Modelle dieser Arbeit die Information über die Gegner nicht direkt verarbeiten. Wer Gegner ist und wer nicht, muss erst gelernt werden. Allerdings spielt diese Frage für `ComRandom`-Agenten im Spiel gegen `RandomAgents` keine Rolle. Was zählt ist, dass (selbst gelegte) Bomben zum Tod führen, weswegen die Agenten nicht eindeutig signalisieren, ob sich ein Gegner in Sichtweite befindet. Es könnte dennoch gut sein, dass „Flammen im Sichtfeld“ selbst nach einer längeren Trainingsperiode immer noch keine eindeutigen Nachrichten produzieren, weil die Agenten ihren Fokus auf andere Eigenschaften der Observation legen.

### Auswertung gegen RandomNoBombAgents

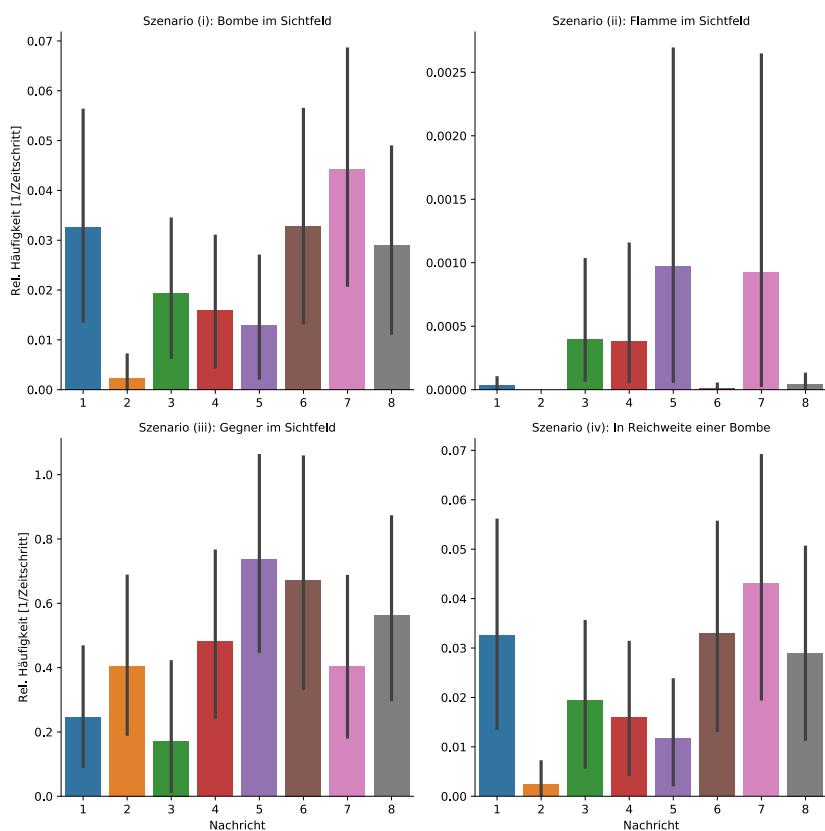


Abbildung 6.4.: Verteilung der Nachrichten von `ComNoBomb`-Agenten in bestimmten Szenarien aus Spielen gegen `RandomNoBombAgents`.

Abb. 6.4 zeigt die Verteilung der Nachrichten des `ComNoBomb`-Teams in den Spielen gegen `RandomNoBombAgents`. Die Agenten näherten sich in keiner der 400 Episoden einem Gegner, weswegen die Szenarien (v) und (vi) analog zu Abb. 6.3 in der Darstellung fehlen. Jede Bombe, die sich in Sichtweite der Agenten befindet, stammt von ihrem eigenen

## 6.2. Qualitative Analyse der Kommunikation

Team, da `RandomNoBombAgents` keine Bomben legen. Aufgrund dessen sind Verteilung (i) und (iv) identisch (unter Berücksichtigung, dass sich `ComNoBomb`-Agenten kaum bewegen). Für „Flammen im Sichtfeld“ gelten dieselben Folgerungen wie zuvor: Sobald eine Bombe explodiert (und Flammen sichtbar werden), wurden die meisten Agenten augenblicklich getötet, weswegen das Szenario nicht oft erfasst wurde. Verteilung (iii) deutet an, dass `ComNoBomb`-Agenten den Gegner deutlich häufiger sehen als `ComRandom`-Agenten. Das Szenario weist die meisten Daten auf, das bedeutet, die Verteilung repräsentiert den Nachrichtenaustausch am besten. Abweichungen in selten auftretenden Szenarien sind darauf zurückzuführen, dass wiederholt die Modelle mit selbem Nachrichtenaustausch in diese Situation kommen.

Die Konfidenzintervalle der Nachrichten, wenn sich Gegner im Sichtfeld befinden, sind dennoch sehr groß und überlappen sich. Das liegt daran, dass `ComNoBomb`-Agenten ähnlich wie `ComRandom`-Agenten kommunizieren und wiederholt dieselben Nachrichten austauschen. Hier wäre das Teilen lokaler Information zwischen den Agenten allerdings von Vorteil: Mit dem geteilten Wissen über die Positionen der `RandomNoBombAgents` könnten beide Spieler beispielsweise gemeinsam versuchen die Gegner zu umzingeln. Es kann nicht gesagt werden, ob die Agenten das in einer längere Trainingsphase hätten lernen können.

### Auswertung gegen SimpleAgents

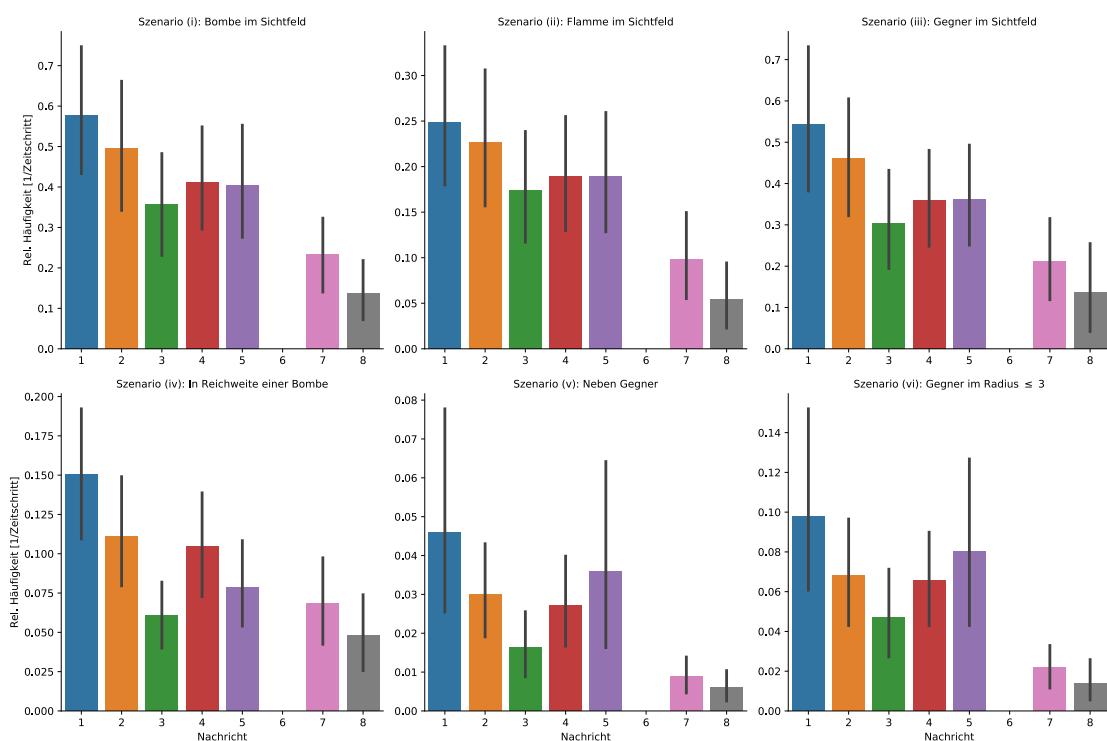


Abbildung 6.5.: Verteilung der Nachrichten von `ComSimple`-Agenten in bestimmten Szenarien gegen `SimpleAgents`.

Abb. 6.5 zeigt die Verteilung der Nachrichten des `ComSimple`-Teams in den Spielen gegen

## 6. Ergebnisse und Evaluation

**SimpleAgents.** Jedes Szenario tritt aufgrund der Beschaffenheit des **SimpleAgents** mehrmals auf. Auffällig ist, dass Nachricht 6 nie verwendet wurde. Allerdings ist die Verteilung der Nachrichten in den Szenarien (i) - (iv), die meistens gemeinsam auftreten, ähnlich. Das gleiche gilt für Szenario (v) und (vi). Das zeigt, dass die Agenten unabhängig von ihrer Observation dieselben Nachrichten verschicken – das entstandene Kommunikationsprotokoll gleicht dem, das im Training gegen die anderen Agenten erlernt wurde. Diese Beobachtung überrascht nicht: Das **Com**-Modell erfuhr wie im Training gegen den **RandomAgent** frühzeitig einen positiven Reward durch den Selbstmord des Gegners. Gleichzeitig gewinnen die Agenten ohne weitere Anpassungen des Nachrichtenaustauschs bereits 20% der Spiele. Unter Berücksichtigung des Kommunikationsprotokolls deckt sich das mit der These, dass die Agenten ihre Strategie (inkl. Nachrichtenaustausch) frühzeitig auf Basis des positiven Rewards optimiert haben. Zwischen Nachrichten und Observationen ist ansonsten kein Zusammenhang erkennbar.

### 6.2.2. Speaker Consistency

Dieses Kapitel gibt Auskunft darüber, ob ein Zusammenhang zwischen den Nachrichten und Aktionen eines **Com**-Agenten besteht. Zusätzlich wird der Verlauf der Speaker Consistency während des Trainings analysiert. Es werden nur die Nachrichten lebender Agenten betrachtet, weswegen die 0 in nachfolgenden Abbildungen fehlt.

Ausgehend von der Architektur des Modells ist intuitiv keine hohe SC zu erwarten, da die Nachrichtenauswahl nicht auf Basis der Aktionen erfolgt. Da Aktions- und Kommunikationsnetz allerdings denselben Input erhalten und auf dieselbe Art und Weise optimiert werden, ist eine Korrelation beobachtbar.

Wie bereits festgestellt werden konnte, unterscheiden sich die Kommunikationsprotokolle der Modelle, die gegen **RandomNoBombAgents** und **SimpleAgents** trainiert wurden, kaum von den Protokollen der **ComRandom**-Modelle. Die Metriken führen aus diesem Grund zu ähnlichen Ergebnissen. Der Vollständigkeit halber, wird die Auswertung im Folgenden an allen Varianten fortgeführt. Im Fokus stehen jedoch **Com**- und **NoCom**-Team, die gegen den **RandomAgent** trainiert wurden. Die Auswertungen bzgl. **ComNoBomb**- und **ComSimple**-Modellen sind weniger detailliert.

#### Auswertung gegen RandomAgents

Abbildung 6.6 zeigt den Verlauf der SC im Training gegen **RandomAgents** jeweils für beide Agenten. Beide Kurven verlaufen nahezu gleich. Zwar sind Parallelen zwischen der Entwicklung des durchschnittlichen Rewards (Abb. 6.1a) und der SC zu erkennen, diese müssen aber nicht kausal zusammenhängen.<sup>4</sup> Bis zum Ende des Trainings konvergiert der Wert der Metrik zu  $\sim 0,1$ . Zu beachten gilt, dass eine Nachricht in Pommerman aus zwei Zahlen besteht, während die Berechnung von SC in dieser Arbeit auf der Menge des Alphabets  $\Sigma$  (ohne 0) erfolgt (und nicht auf allen 64 möglichen Nachrichten/Zahlenkombinationen). Unter dieser Berücksichtigung deckt sich der Wert mit dem beobachteten (Kommunikations-)Verhalten: Wenn ein Agent in jedem Zeitschritt immer dieselbe Aktion ausführt und parallel dazu dieselbe Nachricht, bestehend aus zwei *voneinander verschiedenen* Zahlen, verschickt, ist  $SC = 0$ . Dass die Kurve nicht auf 0 sinkt, ist auf  $\epsilon$

---

<sup>4</sup>Zur Erinnerung: Positive Signaling kann ohne Positive Listening vorliegen. Es ist (noch) ungewiss, ob der Zuhörer die Nachricht verwendet und Kommunikation dadurch den Reward beeinflusst.

## 6.2. Qualitative Analyse der Kommunikation

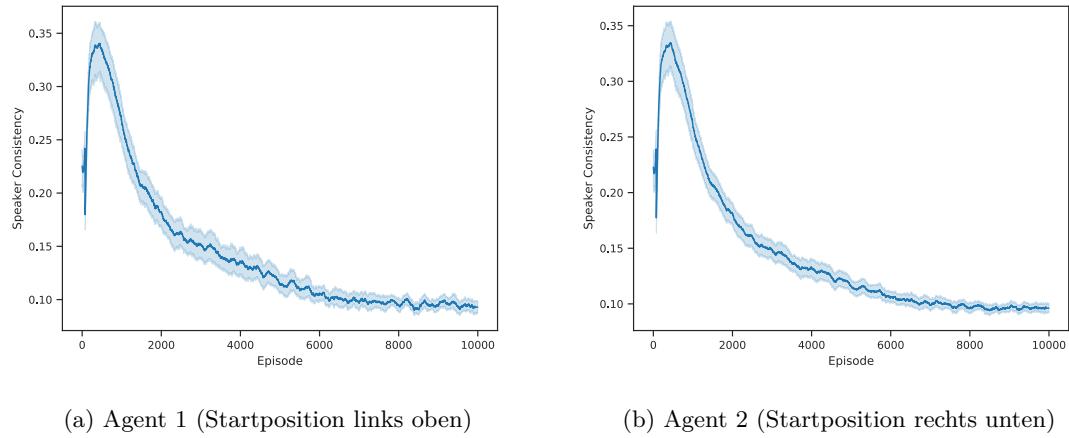


Abbildung 6.6.: Durchschnittliche Speaker Consistency beider Com-Agenten im Verlauf des Trainings gegen RandomAgents.

zurückzuführen.

Das bedeutet nicht, dass die Agenten gelernt haben, zu signalisieren, welche Aktion sie als Nächstes ausführen. Derselbe Input und dasselbe Optimierungsverfahren führen dazu, dass sich die Netze gleich Verhalten, i.e. in jedem Zeitschritt denselben Output generieren.

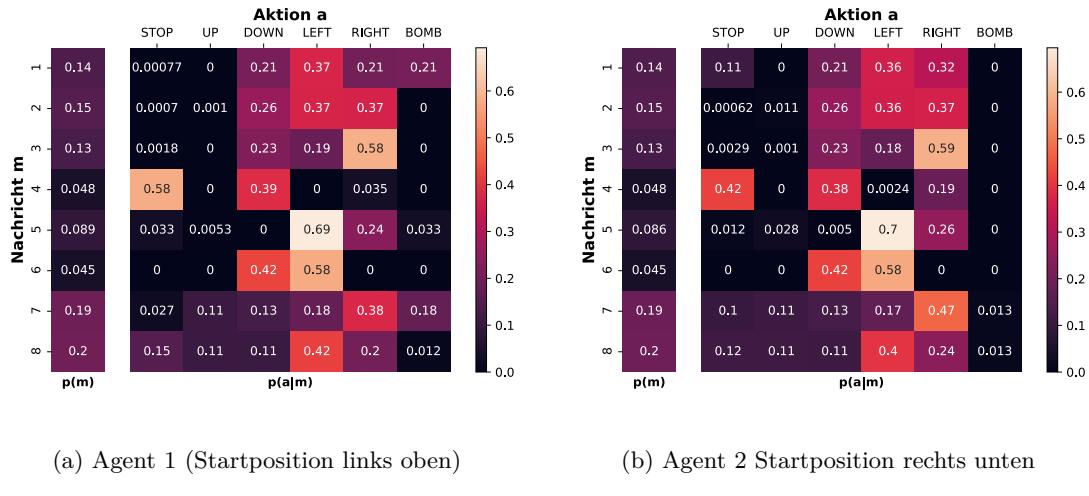


Abbildung 6.7.: Zusammenhang zwischen Nachrichten und Aktionen der Com-Agenten, die gegen RandomAgents trainiert wurden.

Abbildung 6.7 zeigt die Wahrscheinlichkeit  $p(m)$ , mit der jeder Agent eine Nachricht  $m$  verschickt, sowie die Wahrscheinlichkeit  $p(a|m)$ , mit der jeder Agent Aktion  $a$  basierend auf seiner eigenen Nachricht  $m$  ausgewählt.<sup>5</sup> Es fällt auf, dass die Wahrscheinlichkeit

<sup>5</sup>Wie zuvor erwähnt muss beachtet werden, dass die Aktion des Sprechers in Wirklichkeit nicht auf Basis der eigenen Nachricht ausgewählt wird.  $p(a|m)$  wurde anstelle von  $p(a, m)$  gewählt, da ansonsten die Werte der Wahrscheinlichkeiten unübersichtlich klein wären. Es gilt  $p(a, m) = p(a|m)p(m)$ .

## 6. Ergebnisse und Evaluation

$p(m)$  beider Agenten bis auf minimale Abweichungen gleich ist. Es hat sich keine „Frage-Antwort“-Beziehung entwickelt, da die Agenten Nachricht  $m$  ansonsten nicht mit gleicher Wahrscheinlichkeit wählen würden. Die erlernte Strategie beider Agenten setzt nicht das Teilen individueller Informationen voraus.  $p(m)$  deckt sich zudem mit der Verteilung der Nachrichten aus Unterabschnitt 6.2.1. Die Nachrichten 4, 5 und 6 werden mit einer geringeren Wahrscheinlichkeit verschickt als der Rest. In Kombination mit  $p(a|m)$  scheint das jedoch auf den ersten Blick nicht im Einklang mit der Feststellung, dass jeder Agent fast immer dieselbe Aktion ausführt und dieselben Nachrichten verschickt: Das liegt daran, dass sich die Agenten jedes trainierten  $\text{Com}_{\text{Random}}$ -Modells auf unterschiedliche Nachrichten „geeinigt“ haben. Aus diesem Grund zeigt die Analyse für Nachrichten mit geringer Wahrscheinlichkeit eindeutigere Aktionen (die wenigen Agenten, die Nachricht [5, 6] verschicken, wählen primär Aktion LEFT).

Interessant zu beobachten ist, dass Aktion STOP im Vergleich zu DOWN, LEFT und RIGHT relativ selten ausgeführt wird. Angesichts der Tatsache, dass wenig Bomben gelegt und somit auch kaum Wände gesprengt werden, bedeutet das, dass die Agenten wiederholt „ungültige“ Aktionen ausführen. Anders als bei der Analyse des Spielverlaufs vermutet, bleiben die Agenten nicht die ganze Zeit stehen. Stattdessen laufen sie wiederholt gegen Wände, ohne ihre Position zu ändern. Da dieses Verhalten ebenfalls zu einem Sieg führt, solange keine Bomben gelegt werden, haben die Agenten nicht gelernt, das zu vermeiden.

### Auswertung gegen RandomNoBombAgents

Abb. A.4 im Anhang zeigt, wie sich die Speaker Consistency beider Agenten im Training gegen  $\text{RandomNoBombAgents}$  entwickelt. Beide Kurven verlaufen ähnlich zu denen aus vorherigem Abschnitt: Zu Beginn des Trainings ist ein Anstieg der SC zu beobachten, allerdings mit geringerem Ausmaß. Außerdem startet das Team mit einem vergleichsweise niedrigeren Wert und beendet das Training mit einem minimal höheren. Die Kurve flacht deutlich schneller ab. Nach Episode 3000 sind keine Änderungen mehr zu erkennen. Ab diesem Zeitpunkt werden (unter Berücksichtigung von  $\epsilon$ ) weitest gehend dieselben Nachrichten verschickt und dieselben Aktionen ausgeführt.

Abbildung A.5 im Anhang zeigt  $p(m)$  und  $p(a|m)$  beider Agenten des  $\text{Com}_{\text{NoBomb}}$ -Teams in Spielen gegen  $\text{RandomNoBombAgents}$ .  $p(m)$  ist analog zur vorherigen Analyse für beide Agenten sehr ähnlich, auch wenn die Werte diesmal für mehr als nur eine Nachricht minimal abweichen. Die Wahrscheinlichkeit deckt sich außerdem ebenso mit der Verteilung der Nachrichten, die in Unterabschnitt 6.2.1 beobachtet werden konnte. Unter Berücksichtigung des konstanten Nachrichtenaustauschs folgt: Jedes Wort (i.e. jede Zahl) ist fester Bestandteil von ungefähr zwei der 18 trainierten Kommunikationsprotokolle. Es treten nahezu alle (Nachricht, Aktion) Paare auf. Das erklärt den (minimal) größeren Wert der Metrik im Vergleich zu  $\text{Com}_{\text{Random}}$ -Agenten und das größere Konfidenzintervall am Ende des Trainings. Im Gegensatz zu  $p(m)$  sind in  $p(a|m)$  größere Unterschiede zwischen den Agenten erkennbar. Es stellt sich heraus, dass Agent 2 dazu tendiert, Aktion RIGHT auszuführen. Bedingt durch seine Startposition führt das dazu, dass er sich selten bewegt (da er in wenigen Schritten das Ende des Spielfeldes erreicht hat). Agent 1 wählt zwar vermehrt dieselbe Aktion, neigt aber dazu, auch STOP, UP, DOWN und LEFT auszuführen.

### Auswertung gegen SimpleAgents

Abb. A.8 im Anhang zeigt den Verlauf der SC im Training gegen **SimpleAgents** jeweils für beide Agenten. Es sind ähnliche Entwicklungen wie zuvor erkennbar: Der initiale Wert der Metrik gleicht dem im Training gegen **RandomAgents**. Die Kurve flacht nach einem anfänglichen Anstieg ab. Bis Episode 4000 wurde ein Wert von 0,1 erreicht – ca. 2000 Episoden früher als gegen **RandomAgents**. Die Ursache dafür ist nicht eindeutig, da der Wert nicht nur vom Kommunikationsprotokoll, sondern auch von den gewählten Aktionen abhängt. Nach Episode 4000 sind weiterhin nur minimale Änderungen erkennbar.

Abbildung A.9 im Anhang zeigt  $p(m)$  und  $p(a|m)$  beider Agenten des **ComSimple**-Teams in Spielen gegen **SimpleAgents**. Auch hier gilt, dass sich die Wahrscheinlichkeitsverteilung der Nachrichten mit den Beobachtungen aus Unterabschnitt 6.2.1 decken und für beide Agenten gleich ist. Analog zum **ComRandom**-Team unterscheidet sich  $p(a|m)$  beider Agenten kaum. Dementsprechend gilt auch hier zu beachten: Jedes der 18 Modelle verwendet zwei unterschiedliche Nachrichten, weswegen die Abbildung für Nachrichten mit geringer Wahrscheinlichkeit eindeutigere Aktionen zeigt (vgl. Nachricht 7 und Aktion UP, Nachricht 8 und Aktion DOWN). Es ist die Tendenz erkennbar, dass mehrere Modelle gelernt haben, die Nachrichten 2, 3, 4, oder 5 (in beliebiger Kombination) zu verschicken und parallel dazu Aktion RIGHT auszuführen. Außerdem fällt auf, dass die Agenten im Vergleich zu den anderen Modellen mehr Bomben legen. Das Vermeiden von Aktion BOMB kristallisierte sich während des Trainings dementsprechend nicht als Schlüssel zum Sieg heraus. Beide Agenten wählen dennoch hauptsächlich Aktion RIGHT – Agent 2 führt dementsprechend wiederholt eine ungültige Aktion aus. Warum das so ist (und warum nicht stattdessen Aktion LEFT ausgewählt wird, sodass Agent 1 in eine Wand läuft), kann auf Basis dieser Analyse nicht gesagt werden.

### 6.2.3. Instantaneous Coordination

Analog zum vorherigen Abschnitt wird im Folgenden analysiert, ob Aktionen eines **Com**-Agenten von den Nachrichten des Teammitglieds abhängen. Dazu werden  $p(m)$  und  $p(a|m)$  (mit Nachricht  $m$  des Teammitglieds und eigener Aktion  $a$ ) aller **Com**-Versionen aus Spielen gegen entsprechende Gegner untersucht und der Verlauf der Instantaneous Coordination während des Trainings betrachtet. Da das Aktionsnetz unter anderem die Nachricht des Teammitglieds verarbeitet, kann der Nachrichteninput die gewählte Aktion direkt beeinflussen. Anhand dieser Analyse kann das jedoch nicht sicher bestätigt werden.

### Auswertung gegen RandomAgents

Abbildung 6.8 zeigt den Verlauf der IC im Training gegen **RandomAgents** jeweils für beide Agenten. Analog zur SC verlaufen beide Kurven nahezu gleich. Vergleicht man die Entwicklung mit dem Verlauf der Lernkurve (Abb. 6.1a), erkennt man, dass IC bei beiden Agenten zu Beginn des Trainings deutlich ansteigt, während der Reward sinkt. Ab ca. Episode 1000 sinkt IC stetig, gleichzeitig steigt der Reward. Es ist (noch) unbekannt, ob diese Korrelation kausal zusammenhängt oder zufällig ist (i.e. inwiefern die Nachricht bei der Aktionsauswahl berücksichtigt wird). IC sinkt bis zum Ende des Trainings auf ungefähr 0,1, was zu erwarten war: Wenn jeder Agent zwei *verschiedene* Zahlen erhält,

## 6. Ergebnisse und Evaluation

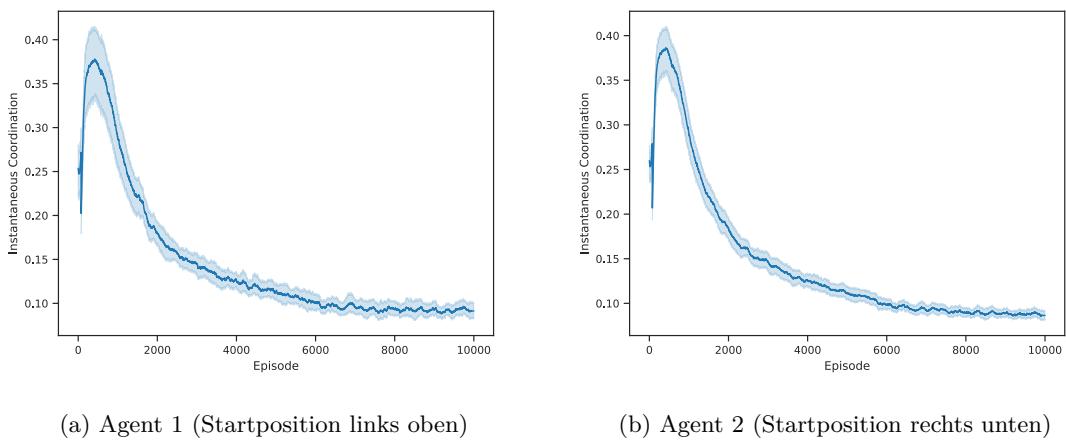


Abbildung 6.8.: Durchschnittliche Instantaneous Coordination beider Com-Agenten im Verlauf des Trainings gegen `RandomAgents`.

die aber in jedem Zeitschritt unverändert sind, und parallel immer dieselbe Aktion produziert, ist  $IC = 0$ . Abweichungen von 0 sind auch hier durch  $\epsilon$  begründet.

Abbildung 6.9 zeigt  $p(m)$  und  $p(a|m)$  für beide Agenten des `ComRandom`-Teams. Die Wahrscheinlichkeiten sind anders als im vorherigen Abschnitt wie folgt zu verstehen:

- $p(m)$  in Abb. 6.9a ist die Wahrscheinlichkeit, mit der *Agent 2* eine Nachricht verschickt;  $p(a|m)$  ist die Wahrscheinlichkeit, mit der *Agent 1* basierend auf der Nachricht  $m$  des Teammitglieds handelt.
- $p(m)$  in Abb. 6.9b ist die Wahrscheinlichkeit, mit der *Agent 1* eine Nachricht verschickt;  $p(a|m)$  ist die Wahrscheinlichkeit, mit der *Agent 2* basierend auf der Nachricht  $m$  des Teammitglieds handelt.

Die Wahrscheinlichkeit  $p(m)$  beider Darstellungen ist im Vergleich zur entsprechenden Abbildung aus dem vorherigen Kapitel (Abb. 6.7) vertauscht. Minimale Abweichungen der Werte bzw. Rundungen sind auf die Art der Messung zurückzuführen: Die Dokumentation der Nachrichten und Aktionen begann hier erst bei  $t = 1$ , da jeder Agent zum Zeitpunkt  $t = 0$  die Nachricht  $[0, 0]$  verschickte.

Im Vergleich zu 6.7 sind keine großen Unterschiede erkennbar. Das deckt sich mit der Beobachtung, dass beide Agenten gleich handeln und kommunizieren: Betrachtet man Agent 1, hängen die Aktionen des Spielers genau so von seiner eigenen Nachricht ab wie von der seines Teammitglieds (das gleiche gilt für Agent 2). Dass der Nachrichteninput nahezu konstant ist, erschwert die Einschätzung, wie stark die Nachricht die Aktion beeinflusst.

### Auswertung gegen `RandomNoBombAgents`

Abb. A.6 im Anhang zeigt wie sich die Instantaneous Coordination beider Agenten im Training gegen `RandomNoBombAgents` entwickelt. Beide Kurven verlaufen ähnlich zu SC: Die Kurve flacht ähnlich schnell ab, bis Episode 3000 ist das größte Gefälle zu beobachten. Vergleicht man die Kurve mit der Entwicklung des durchschnittlichen Rewards

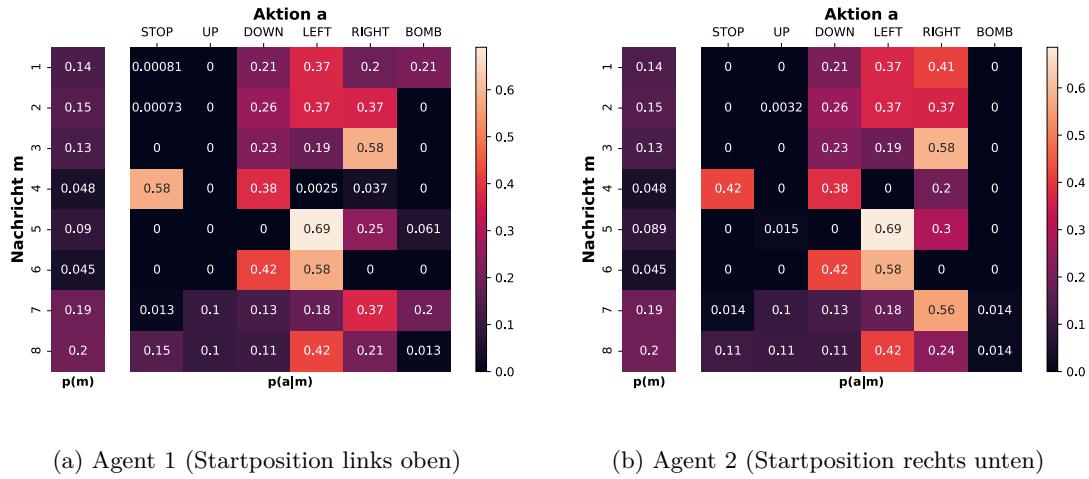


Abbildung 6.9.: Zusammenhang zwischen Nachrichten des Teammitglieds und eigenen Aktionen für Com-Agenten, die gegen RandomAgents trainiert wurden.

(Abb. 6.1b), ist zu erkennen, dass das Com-Team ab diesem Zeitpunkt die Spiele vereinzelt mit einem Unentschieden beenden konnte. Der Tiefpunkt der IC ist bei  $\sim$  Episode 7500 erreicht.

Abbildung A.7 im Anhang zeigt  $p(m)$  und  $p(a|m)$  für beide Agenten des ComNoBomb-Teams. Die Wahrscheinlichkeiten sind entsprechend Abb. 6.9 zu interpretieren. Unter der Berücksichtigung, dass ComNoBomb-Agenten ähnlich wie ComRandom-Agenten kommunizieren und agieren, überraschen die Ergebnisse nicht. Zu entsprechender Abbildung aus vorherigem Abschnitt (vgl. Abb. A.5 im Anhang) sind dementsprechend kaum Unterschiede zu erkennen.

### Auswertung gegen SimpleAgents

Abb. A.10 im Anhang stellt den Verlauf der Metrik für beide Agenten im Training gegen SimpleAgents dar. Es ist kein Zusammenhang zwischen IC und dem durchschnittlichen Reward erkennbar (vgl. Abb. 6.1c). Analog zu vorherigen Abschnitten entwickeln sich IC und SC gleich.

Dieselbe Beobachtung gilt analog für  $p(a|m)$  in Abbildung A.11 (berechnet nach Art der IC) und in Abb. A.9 (berechnet nach Art der SC). Es bestätigt sich erneut, dass beide Agenten des ComSimple-Teams ähnlich handeln.

## 6.3. Interpretation der Nachrichten

Eine mögliche Interpretation der Nachrichten beider Agenten, die in allen Versionen des Modells wiederholt gleich sind, wäre, dass jeder seinem Teammitglied signalisiert, dass er noch lebt. Die Nachricht wäre eine Art „Lebenszeichen“. Es könnte sein, dass ein Agent mit dieser Information während des Trainings einen positiven Reward begründet hat,

## 6. Ergebnisse und Evaluation

obwohl er im Spiel gestorben ist. Wie genau allerdings das Teammitglied diese Information verarbeitet, konnte mit der bisherigen Analyse nicht gesagt werden. Zwar korrelieren gewählte Aktion und die *gesamte* Nachricht (nicht einzelne Wörter) des Teammitglieds stark, beides muss aber nicht kausal zusammenhängen. Das Aktionsnetz könnte unabhängig von der Nachricht auf Basis der Spielfeldinformation dieselbe Aktion ausgeben. Ein Verhalten, das mit der Interpretation der Nachricht übereinstimmen würde: Das Ändern der Strategie, wenn ein Teammitglied gestorben ist, macht in Spielen gegen **RandomAgents** keinen Sinn, weswegen die Information darüber nicht stark gewichtet wird. Um das zu überprüfen, wurde in einem weiteren Experiment die Kommunikation zwischen den Agenten komplett entfernt. Die Ergebnisse dazu werden im nächsten Kapitel präsentiert. Immerhin ist klar, dass die Möglichkeiten des Kommunikationsprotokolls so nicht komplett ausgeschöpft wurden. Es ist selbsterklärend, dass bei Erhalt einer Nachricht ungleich 0 der Sprecher noch lebt. Abgesehen davon hätte dieses Ergebnis bereits mit der Übermittlung einer Zahl erreicht werden können.

Damit kann die stabilere Performance der **Com<sub>Random</sub>**-Agenten im Vergleich zu Agenten des **NoCom<sub>Random</sub>**-Teams noch nicht erklärt werden. Da im Training ebenfalls sehr gute Ergebnisse erreicht wurden, kann man davon ausgehen, dass das nicht-kommunikationsfähige Modell durch Value Decomposition auch in der Lage war, den Reward anhand der Leistungen des gesamten Teams zu begründen. Zu beachten gilt allerdings, dass mit [0, 0] eine Nachricht übermittelt wird, die Agenten ansonsten nie beobachten. Im Gegensatz dazu erfasst das Environment für tote Agenten die Aktion STOP.<sup>6</sup> Im Falle eines Todes erfolgt der Optimierungsschritt im **NoCom**-Modell auf Basis von Aktionen, die auch von lebenden Agenten ausgeführt werden können. Der schwankende Erfolg im Training kann eventuell darauf zurückgeführt werden, dass **NoCom<sub>Random</sub>**-Agenten auf zusätzliche Stabilität, die durch Kommunikation erlangt wird, verzichten müssen.

Weswegen **NoCom<sub>NoBomb</sub>**-Agenten in Spielen gegen **RandomNoBombAgents** dennoch besser abschneiden als das **Com<sub>NoBomb</sub>**-Team, ist unklar. Eine mögliche Erklärung wäre, dass beide Modelle im Training noch nicht vollständig optimiert werden konnten. Wie im Training gegen **RandomAgents** erzielte das nicht-kommunikationsfähige Modell zu Beginn bessere Ergebnisse, die in einer längeren Lernphase vom **Com<sub>NoBomb</sub>**-Team aufgeholt werden könnten. Andererseits könnte es auch sein, dass die verwendete Architektur nicht für dieses Setup geeignet ist. Ohne das Training fortzuführen bzw. alternative Modelle zu testen, kann das nicht eindeutig gesagt werden.

### 6.4. Performance bei Entfernen der Kommunikation

Im vorherigen Kapitel wurde festgestellt, dass Agenten des **Com**-Modells lernen, wiederholt dieselben zwei Zahlen untereinander auszutauschen. Zwar wurde dadurch das Potenzial des Kommunikationskanals nicht vollständig ausgenutzt, doch solange die Leistungen stimmen, ist das nichts grundsätzlich Schlechtes.

Zur Erinnerung: Das kommunikationsfähige Modell lernt zwar langsamer, aber scheinbar

---

<sup>6</sup>Theoretisch würde das bedeuten, dass **Com**-Agenten auch ohne Value Decomposition lernen könnten, den Reward als Teamreward zu interpretieren – trotz partieller Beobachtbarkeit. **NoCom**-Agenten im Gegensatz dazu nicht.

#### 6.4. Performance bei Entfernen der Kommunikation

auch stabiler als das Modell ohne Kommunikation. Zwischen Nachrichten und Observationen ist kein direkter Zusammenhang erkennbar und durch vorherige Analysen kann nicht genau gesagt werden, ob und wie die Zahlen interpretiert werden. Um das herauszufinden, wurde ein weiteres Experiment durchgeführt, in dem beide Agenten weiterhin jeweils eine Nachricht  $m$  produzieren, die vor dem Abschicken durch  $[0, 0]$  ersetzt wird.

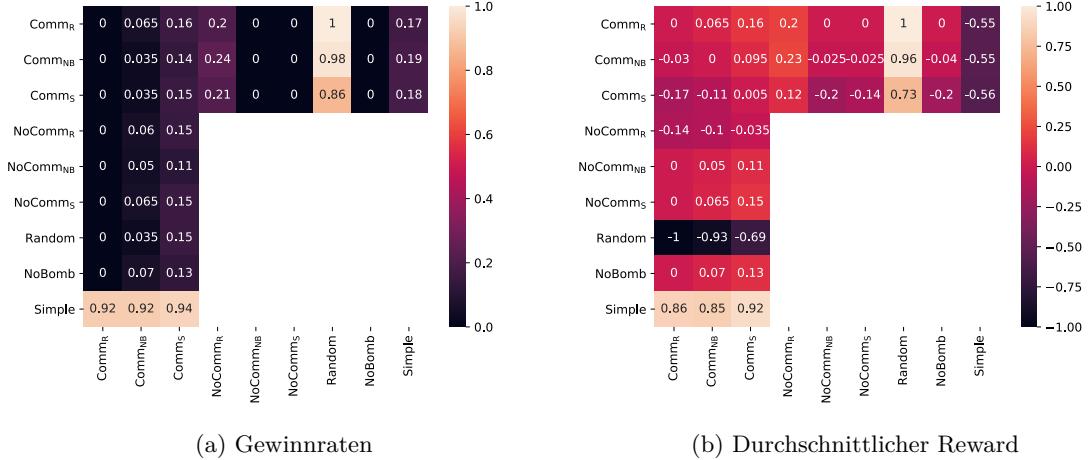


Abbildung 6.10.: Performance der Com-Modelle ohne Kommunikation. Jede Nachricht wurde durch  $[0, 0]$  ersetzt.

Abbildung 6.10 visualisiert die Performance der Kommunikationsmodelle in diesem Experiment. Tabelle A.1b im Anhang enthält genaue Angaben über die Zahl der Siege, Unentschieden und Niederlagen. Es ist eindeutig erkennbar, dass das  $\text{Com}_{\text{Random}}$ - und  $\text{Com}_{\text{NoBomb}}$ -Team gegen nahezu jeden Gegner (außer den **SimpleAgent**) eine höhere Gewinnrate und einen höheren Reward erzielen. Was auf den ersten Blick überrascht, kann mit Blick auf Tabelle A.1b im Anhang erklärt werden: In Spielen gegen **RandomAgents** ist die Zahl der Siege höher, gegen alle anderen Gegner (**SimpleAgent** ausgenommen) konnten mehrere Unentschieden erzielt werden. Das bedeutet, dass sich die Agenten weniger selbst töten und folglich weniger Bomben gelegt werden. Das Aktionsnetz berechnet mit  $[0, 0]$  als Nachrichteninput für die Aktion BOMB einen geringeren Q-Value. Das macht Sinn: Im Training hat jeder Agent gelernt, dass er beim Legen einer Bombe stirbt (da das Ausweichen von Flammen nicht gelernt wurde). Wenn das Teammitglied nun bereits tot war (i.e. die Nachricht  $[0, 0]$  verschickt hat), hatte das einen negativen Reward zur Folge – beide Agenten sind gestorben, das Team hat verloren. Gleichzeitig lernten die Agenten aber, dass sie bei Erhalt einer von  $[0, 0]$  verschiedenen Nachricht sterben und trotzdem einen positiven Reward erzielen können (nämlich dann, wenn das Teammitglied den Rest der Partie gewinnt). Die Agenten lernten, vorsichtiger zu spielen, sobald sie nur noch alleine auf dem Spielfeld standen. Das erklärt die Ergebnisse der Tests: Durch die Nachricht  $[0, 0]$  nahm jeder Agent an, dass das Team verlieren würde, sobald er eine Bombe legt. Beide Agenten vermieden die Aktion BOMB dadurch mehr als ohnehin schon – dadurch konnten mehrere Spiele gegen den **RandomAgent** gewonnen bzw. weniger Spiele gegen die anderen Agenten verloren werden. Für die These spricht außerdem, dass Spiele zwischen beiden Modellen untereinander häufiger in einem Unentschieden endeten.

## *6. Ergebnisse und Evaluation*

Auffällig ist, dass gegen **ComSimple**-Agenten nicht nur die Zahl der Unentschieden höher ist, sondern auch die Zahl der Gewinne. Das zeigt, dass **ComSimple**-Agenten ohne Kommunikation öfter durch ihre eigenen Bomben sterben als mit Kommunikation. Das ist auch in Spielen gegen **NoComNoBomb**-Agenten und **RandomNoBombAgents** erkennbar: **ComSimple**-Agenten verloren durchschnittlich ca. 9 Spiele mehr als sonst. Das Modell reagiert auf das Entfernen der Kommunikation deutlich empfindlicher. Eine mögliche Erklärung ist, dass die Wahrscheinlichkeit gegen den **SimpleAgent** zu gewinnen, obwohl ein Agent des Teams bereits gestorben ist, deutlich geringer ist als gegen **RandomAgents** oder **RandomNoBombAgents**. **ComSimple**-Agenten erfuhrten im Training unabhängig von der Nachricht in den meisten Fällen einen negativen Reward. Das Modell lernte nicht zwischen [0, 0] und anderen Nachrichten zu differenzieren.

Gewinnrate und Reward aller Modelle in Spielen gegen den **SimpleAgent** zeigen allgemein keine eindeutigen Tendenzen und sind wenn überhaupt nur minimal verändert. Das Vermeiden von Aktion BOMB führt in diesem Fall zu nichts, außer, dass die Episoden etwas länger dauern. **SimpleAgents** können **Com**-Agenten früher oder später dennoch besiegen.

# **7. Fazit und Ausblick**

Zum Schluss werden die zentralen Ergebnisse dieser Arbeit zusammengefasst und Möglichkeiten für zukünftige Forschungsarbeiten präsentiert.

## **7.1. Zusammenfassung**

Das Ziel dieser Arbeit war herauszufinden, ob Kommunikation in Pommerman erlernbar ist und, wenn ja, ob die Agenten diese auch zu ihren Gunsten nutzen. Dazu wurde ein Team aus zwei Agenten entwickelt und mittels Deep Recurrent Q-Learning und Value Decomposition gegen drei Gegner unterschiedlichen Schwierigkeitsgrades zentral trainiert. Anschließend wurden die Ergebnisse mit der Performance verschiedener Teams verglichen, u.a. eines nicht-kommunikationsfähigen Modells ähnlicher Architektur, sowie das Kommunikationsprotokoll der Agenten aus dezentral ausgeführten Tests anhand ausgewählter Metriken analysiert. Vorab wurden die Erweiterung von Deep Q-Learning auf partiell beobachtbare Multi-Agenten Domänen ausführlich behandelt sowie die Herausforderungen von Pommerman beschrieben. Des Weiteren wurden State-Of-The-Art und bisherige Erkenntnisse über Deep Q-Learning, emergente Kommunikation und Ansätze zu Pommerman vorgestellt.

Es konnte gezeigt werden, dass das kommunikationsfähige Modell in der Lage war, zu lernen, ein Team aus einfachen Gegnern zu besiegen. Nach Abschluss des Trainings schlug es sich besser als sein nicht-kommunikationsfähiges Gegenstück. Das konnte im Training allerdings deutlich schneller ähnlich gute Ergebnisse erzielen, musste im Verlauf jedoch Performanceeinbrüche hinnehmen. Folglich ist Kommunikation nicht immer zwingend notwendig, um als Gruppe erfolgreich zu agieren. Die Performance der (kommunikationsfähigen) Agenten konnte in Spielen gegen schwierigere Teams verbessert werden, ist aber noch weit von bestmöglichen Ergebnissen entfernt. Nicht-kommunikationsfähige Modelle schnitten etwas besser ab, das Optimum konnte aber ebenso nicht erreicht werden. Allgemein konnte beobachtet werden, dass alle trainierten Agenten sehr defensiv agieren. Das Verhalten zeigte sich auch im Kommunikationsprotokoll: Die Agenten tauschten wiederholt dieselbe Zahlenkombination untereinander aus. Nachrichten und Aktionen korrelierten dadurch indirekt. Die Analyse der Kommunikation konnte, unabhängig von den Gegnern, keinen eindeutigen Zusammenhang zwischen der Nachricht und lokalen Observationen aufdecken. In einem weiteren Experiment, in dem die Kommunikation entfernt wurde, konnten die Agenten bessere Ergebnisse erzielen. Die Nachricht wurde als eine Art „Lebenszeichen“ der Agenten interpretiert, das ein Agent im Training als Erklärung für (nicht repräsentative) Rewards verwenden konnte. Das Resultat des Experiments wurde damit begründet, dass ein Agent unter der Annahme, dass sein Teammitglied bereits tot ist, vorsichtiger handelt. Das bedeutet, dass der Kommunikationskanal genutzt wird. Auch wenn die Agenten daran scheitern, das Potenzial davon vollständig auszuschöpfen, konnte gezeigt werden, dass Kommunikation in Pommerman erlernbar ist.

## 7.2. Weitere Überlegungen zur Forschung

Abschließend werden mögliche Verbesserungsvorschläge skizziert, durch die aggressivere Strategien trainiert und der Kommunikationskanal vielseitiger genutzt werden könnten.

Um offene Fragen zu beantworten, wäre der nächste logische Schritt, das Training von Com- und NoCom-Modell gegen **RandomNoBombAgents** und **SimpleAgents** fortzuführen. Angesichts der momentan erreichten Performance und der Trainingsdauer in verwandten Arbeiten wären mindestens 20000 weitere Episoden empfehlenswert.

Alternativ könnte man beide Modelle gegen ein Team aus Agenten unterschiedlichen Schwierigkeitsgrades trainieren. **RandomAgent**, **RandomNoBombAgent** und **SimpleAgent** sind dabei beliebig kombinierbar. Dadurch kann die Herausforderung im Vergleich zu einem Team aus **RandomAgents** vergrößert werden, ohne direkt gegen zwei **SimpleAgents** antreten zu müssen. Das Idee kann erweitert werden, indem die Startposition der Gegner pro Episode variiert, sodass die Modelle nicht davon ausgehen können, dass z.B. der schwächere Agent in der linken unteren Ecke startet. In einem Training, das beispielsweise einen **RandomNoBombAgent** und einen **SimpleAgent** involviert, wäre es unter anderem interessant zu beobachten, ob Com-Agenten kommunizieren würden, in welchen Teilen des Spielfelds sich der Agent, der Bomben legen kann, befindet.

Eine weitere Überlegung wäre der Einsatz von Curriculum Learning, um die Modelle in mehreren Trainingsphasen gegen unterschiedlich schwere Gegner zu trainieren. Diese Herangehensweise lässt sich einfach mit vorher genannter Strategie verbinden, indem pro Trainingsphase jeweils nur ein Agent des gegnerischen Teams durch einen schwieriger zu besiegenden ersetzt wird.

Des Weiteren bieten sich mehrere Optionen an, durch die die Agenten den Kommunikationskanal im Com-Modell möglicherweise effektiver nutzen. Anstatt Aktions- und Kommunikationsnetz getrennt voneinander zu optimieren, könnte man beide Netze innerhalb eines Back Propagation-Schritts gleichzeitig optimieren. Dafür verwendet man bei der Optimierung anhand eines Erfahrungstupels  $e_t = (o_t, u_t, r_{t+1}, o_{t+1})$  aus dem Replay Buffer nicht Nachricht  $m_{t-1}$  aus Observation  $o_t$ , sondern die Nachricht, die das Kommunikationsnetz auf Basis der Spielfeldinformationen (aus  $o_t$ ) berechnet. Back Propagation erfolgt dann ausgehend von  $VD_{Act}$  (vgl. Abbildung 5.4 aus Kapitel 5) bis über beide Kommunikationsnetze.  $VD_{Comm}$  ist dadurch nicht mehr notwendig.

Das Kommunikationsnetz könnte weiterhin so angepasst werden, dass es nicht die Q-Values jedes einzelnen Wortes berechnet, sondern die Q-Values aller Tupel/Wortkombinationen. Die Idee dahinter ist, dass Wörter, die einzeln einen niedrigen Q-Value haben, kombiniert einen höheren Reward geben könnten.

Alternativ könnte man für die Nachrichten- und Aktionsauswahl ein gemeinsames Netz verwenden. Das daraus resultierende Modell würde RIAL mit Value Decomposition kombinieren. Eine weitere, nahe liegende Überlegung wäre demnach die Umsetzung von DIAL (mit VD).

Um die Modelle im Allgemeinen besser auf die Domäne Pommerman abzustimmen, könnte man die Implementierung mit verschiedenen Verfahren aus Abschnitt 4.3 verknüpfen. Deep Q-Learning from Demonstrations würde helfen, das Training zu beschleunigen, sowie aggressivere Strategien zu erlernen. Backplay könnte als Erweiterung von DQfD zudem

## 7.2. Weitere Überlegungen zur Forschung

das Problem des verzögerten Rewards lösen.



## A. Anhang

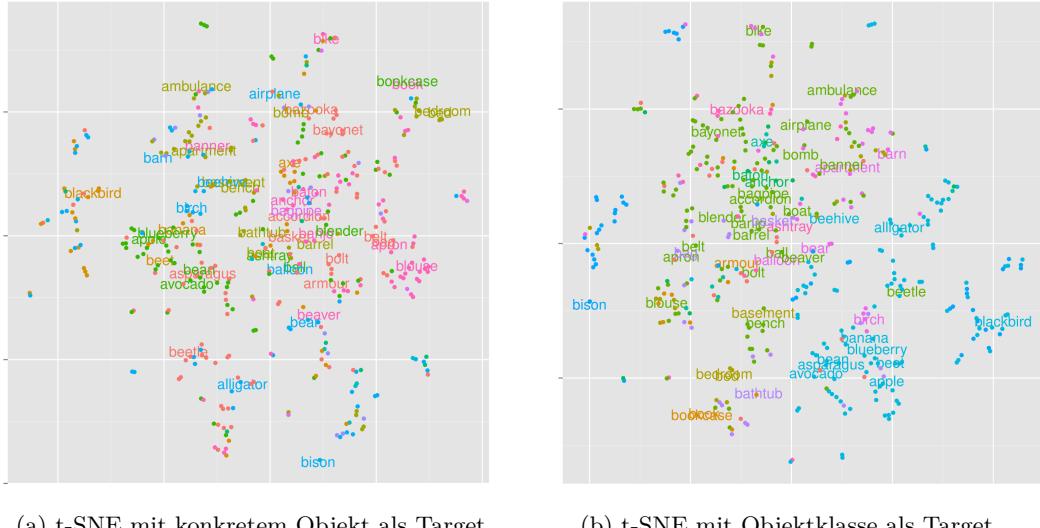


Abbildung A.1.: Entnommen aus [LPB16]: 2-dimensionale Vektorrepräsentationen der Objekte in den Bildern, ermittelt durch t-SNE. Jedes Objekt ist farblich nach dem Symbol gekennzeichnet, mit dem es am häufigsten assoziiert wurde.

## A. Anhang

	Comm <sub>R</sub>	Comm <sub>NB</sub>	Comm <sub>S</sub>	No-Comm <sub>R</sub>	No-Comm <sub>NB</sub>	No-Comm <sub>S</sub>	Random	NoBomb	Simple
Comm <sub>R</sub>	(16, 173, 11)	(46, 142, 12)	(30, 160, 10)	(47, 145, 8)	(0, 193, 7)	(0, 188, 12)	(190, 0, 10)	(0, 184, 16)	(40, 12, 148)
Comm <sub>NB</sub>	(28, 134, 38)	(41, 123, 36)	(22, 135, 43)	(40, 127, 33)	(0, 158, 42)	(0, 163, 37)	(175, 0, 25)	(0, 163, 37)	(34, 15, 151)
Comm <sub>S</sub>	(22, 149, 29)	(41, 125, 34)	(33, 135, 32)	(49, 124, 27)	(0, 171, 29)	(0, 174, 26)	(166, 1, 33)	(0, 171, 29)	(39, 11, 150)
No-Comm <sub>R</sub>	(15, 147, 38)	(54, 110, 36)	(23, 144, 33)	(41, 135, 24)	(0, 169, 31)	(0, 163, 37)	(156, 1, 43)	(0, 162, 38)	(26, 14, 160)
No-Comm <sub>NB</sub>	(25, 175, 0)	(49, 151, 0)	(38, 162, 0)	(38, 162, 0)	(0, 200, 0)	(0, 200, 0)	(200, 0, 0)	(0, 200, 0)	(44, 16, 140)
No-Comms	(13, 187, 0)	(40, 160, 0)	(31, 169, 0)	(48, 152, 0)	(0, 200, 0)	(0, 200, 0)	(200, 0, 0)	(0, 200, 0)	(31, 18, 151)
Random	(14, 0, 186)	(36, 2, 162)	(31, 0, 169)	(47, 0, 153)	(0, 0, 200)	(0, 0, 200)	(103, 2, 95)	(0, 0, 200)	(23, 0, 177)
NoBomb	(23, 177, 0)	(55, 145, 0)	(30, 170, 0)	(35, 165, 0)	(0, 200, 0)	(0, 200, 0)	(200, 0, 0)	(0, 200, 0)	(38, 19, 143)
Simple	(182, 6, 12)	(177, 6, 17)	(184, 2, 14)	(185, 2, 13)	(176, 3, 21)	(179, 6, 15)	(200, 0, 0)	(171, 8, 21)	(104, 26, 70)

(a) Performance aller Teams im Vergleich

	Comm <sub>R</sub>	Comm <sub>NB</sub>	Comm <sub>S</sub>	No-Comm <sub>R</sub>	No-Comm <sub>NB</sub>	No-Comms	Random	NoBomb	Simple
Comm <sub>R</sub>	(0, 200, 0)	(13, 187, 0)	(32, 168, 0)	(41, 159, 0)	(0, 200, 0)	(0, 200, 0)	(200, 0, 0)	(0, 200, 0)	(35, 21, 144)
Comm <sub>NB</sub>	(0, 194, 6)	(7, 186, 7)	(28, 163, 9)	(49, 147, 4)	(0, 195, 5)	(0, 195, 5)	(196, 0, 4)	(0, 192, 8)	(38, 15, 147)
Comm <sub>S</sub>	(0, 166, 34)	(7, 164, 29)	(31, 139, 30)	(43, 137, 20)	(0, 159, 41)	(0, 171, 29)	(173, 1, 26)	(0, 160, 40)	(37, 14, 149)
No-Comm <sub>R</sub>	(0, 172, 28)	(12, 155, 33)	(30, 133, 37)	—	—	—	—	—	—
No-Comm <sub>NB</sub>	(0, 200, 0)	(10, 190, 0)	(22, 178, 0)	—	—	—	—	—	—
No-Comms	(0, 200, 0)	(13, 187, 0)	(31, 169, 0)	—	—	—	—	—	—
Random	(0, 0, 200)	(7, 1, 192)	(31, 0, 169)	—	—	—	—	—	—
NoBomb	(0, 200, 0)	(14, 186, 0)	(26, 174, 0)	—	—	—	—	—	—
Simple	(184, 4, 12)	(184, 2, 14)	(188, 7, 5)	—	—	—	—	—	—

(b) Comm-DQN Performance ohne Kommunikation

Tabelle A.1.: Testergebnisse: (a) zeigt die Performance aller Modelle und Pommerman Agenten. (b) zeigt die Performance aller Comm-DQN-Modelle ohne Kommunikation. Jede Nachricht wurde durch [0, 0] ersetzt. Alle Tests wurden zwei Mal durchgeführt, sodass die Performance jedes Teams in beiden Startpositionen evaluiert werden kann. Zeilen enthalten die Teams, die in den „normalen“ Positionen starteten. Jede Zelle enthält die Verteilung von (Sieg, Unentschieden, Niederlage) der 200 Tests aus Sicht der Teams in den Zeilen.

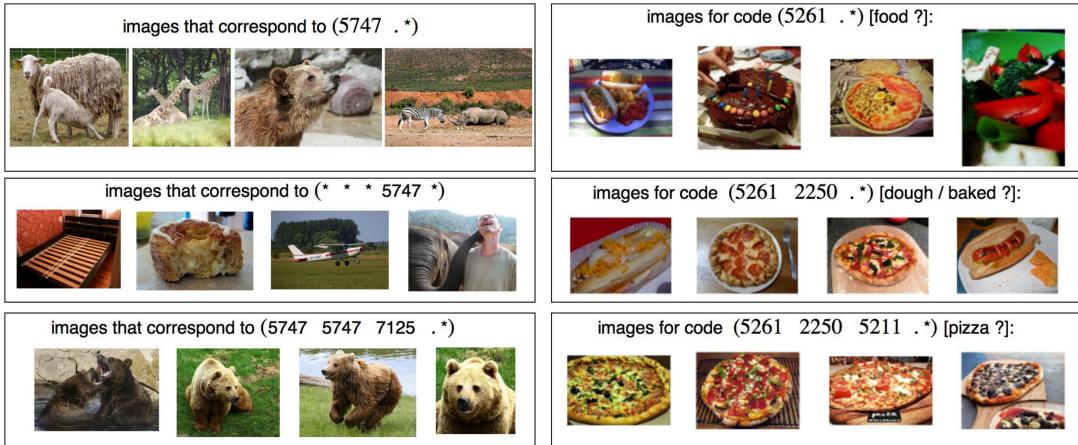


Abbildung A.2.: Entnommen aus [HT17]. Links: Beispielbilder, die der Sequenz (5757 .\*) entsprechen. Die zweite Reihe zeigt, dass Nachrichten mit demselben Wort an einer anderen Stelle mit anderen Objekten assoziiert werden. Rechts: Beispielbilder, die der Sequenz (5261 .\*) entsprechen. Reihe zwei und drei zeigen eine hierarchische Kodierung innerhalb der assoziierten Kategorie.

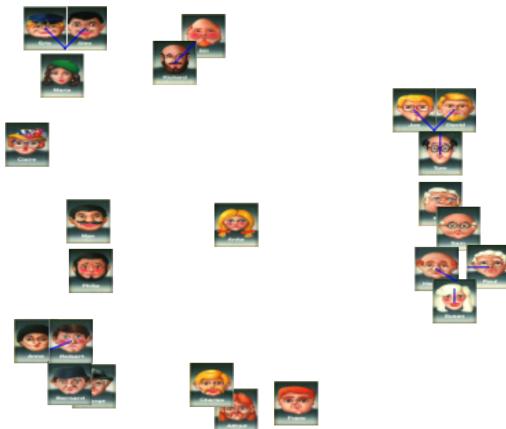


Abbildung A.3.: Entnommen aus [JKJG16]: Die Beziehungen zwischen Bildern aus „Wer ist es?“ als t-SNE Mapping. Die Distanz zwischen zwei Bildern wird daran gemessen, wie oft eine Frage zu den Bildern gleich beantwortet wird.

## A. Anhang

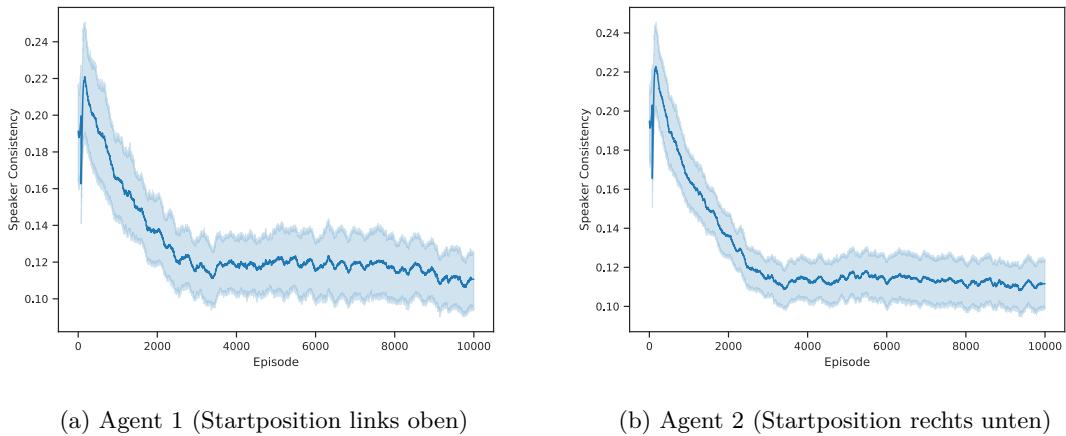


Abbildung A.4.: Durchschnittliche Speaker Consistency beider Comm-DQN-Agenten im Verlauf des Trainings gegen RandomNoBombAgents.

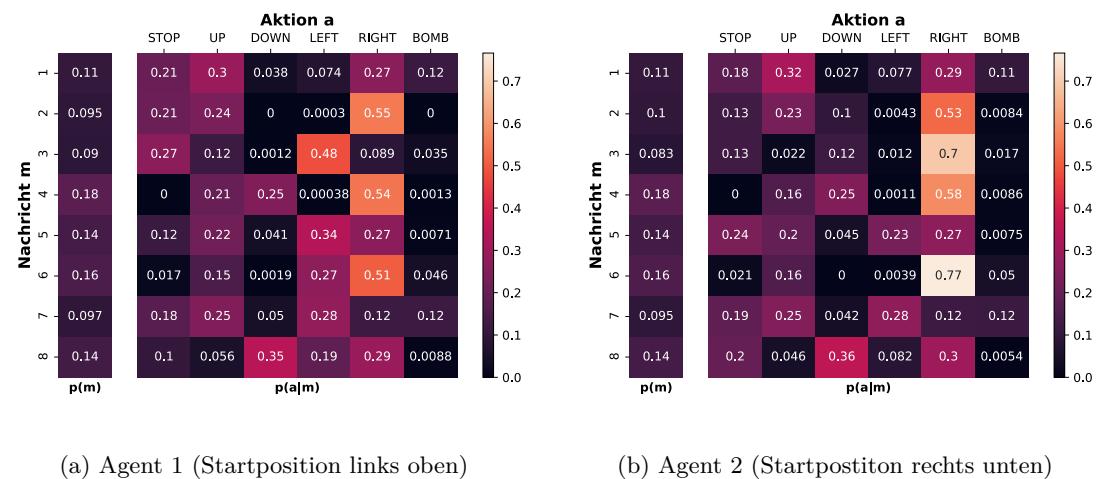


Abbildung A.5.: Zusammenhang zwischen Nachrichten und Aktionen der Comm-DQN-Agenten, die gegen RandomNoBombsAgents trainiert wurden.

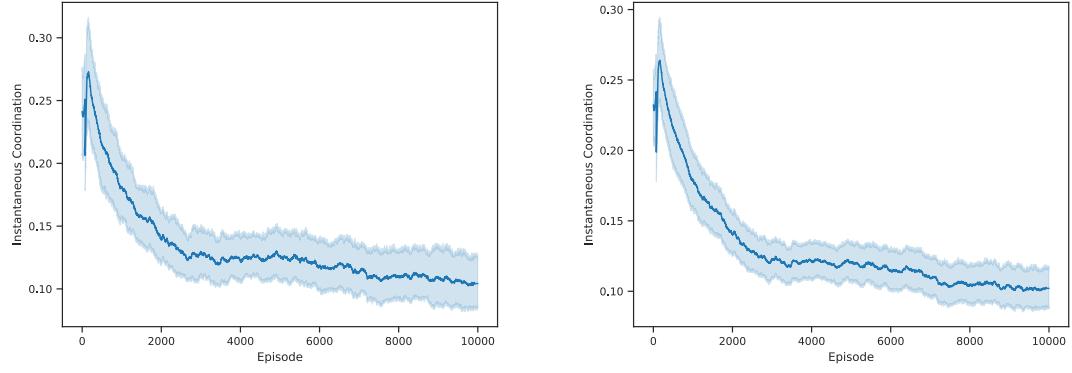


Abbildung A.6.: Durchschnittliche Instantaneous Coordination beider Comm-DQN-Agenten im Verlauf des Trainings gegen RandomNoBombAgents.

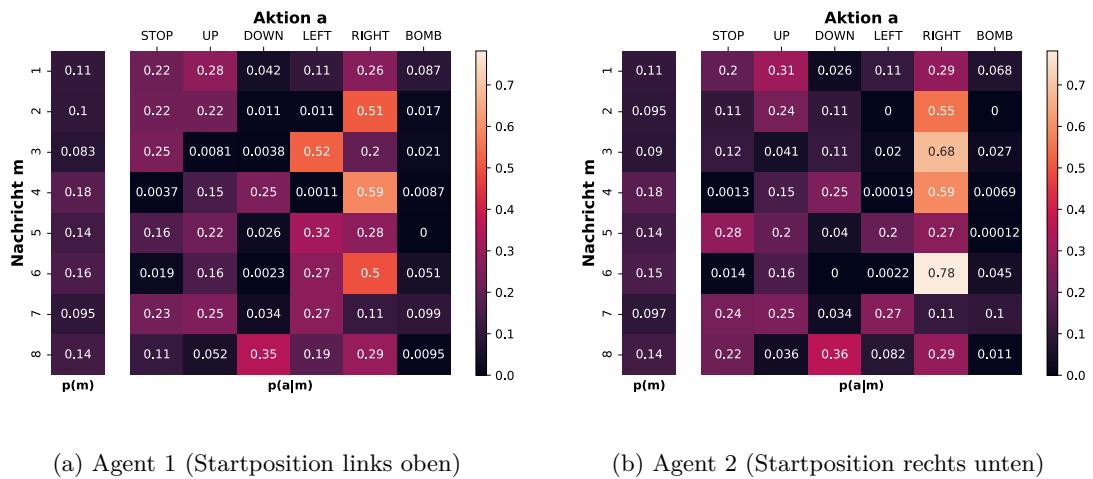


Abbildung A.7.: Zusammenhang zwischen Nachrichten des Teammitglieds und eigenen Aktionen für Comm-DQN-Agenten, die gegen RandomNoBombAgents trainiert wurden.

## A. Anhang

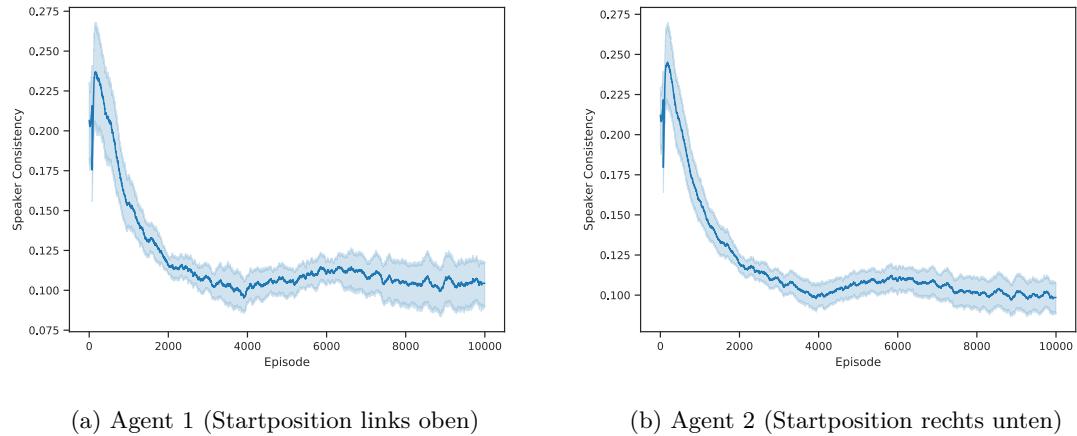


Abbildung A.8.: Durchschnittliche Speaker Consistency beider Comm-DQN-Agenten im Verlauf des Trainings gegen SimpleAgents.

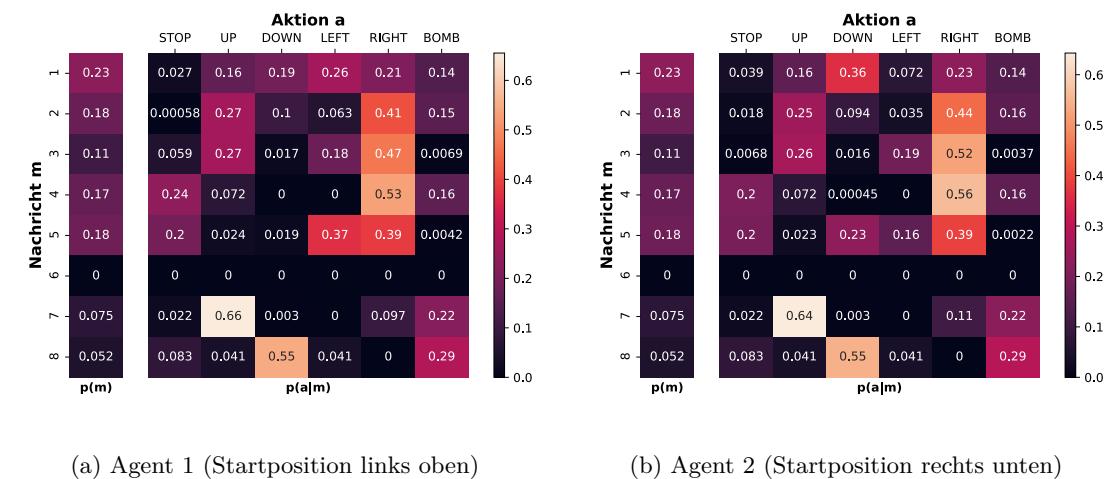
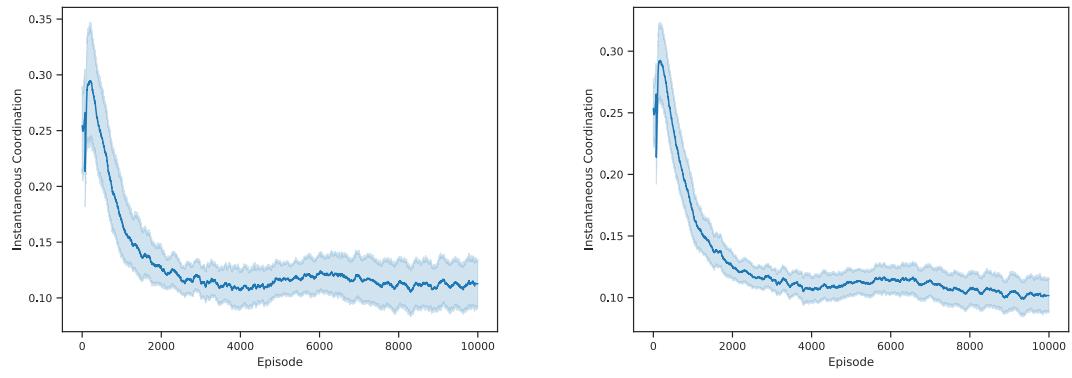


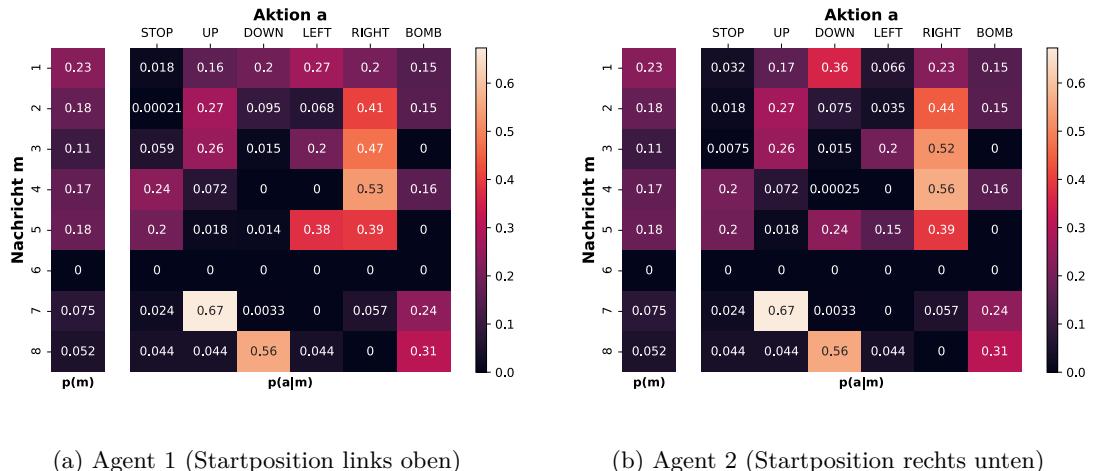
Abbildung A.9.: Zusammenhang zwischen Nachrichten und Aktionen der Comm-DQN-Agenten, die gegen SimpleAgents trainiert wurden.



(a) Agent 1 (Startposition links oben)

(b) Agent 2 (Startposition rechts unten)

Abbildung A.10.: Durchschnittliche Instantaneous Coordination beider Comm-DQN-Agenten im Verlauf des Trainings gegen SimpleAgents.



(a) Agent 1 (Startposition links oben)

(b) Agent 2 (Startposition rechts unten)

Abbildung A.11.: Zusammenhang zwischen Nachrichten des Teammitglieds und eigenen Aktionen für Comm-DQN-Agenten, die gegen SimpleAgents trainiert wurden.



# Abbildungsverzeichnis

2.1.	Agent-Environment-Interaktion in einem MDP . . . . .	3
2.2.	Agenten-Environment-Interaktion in einem Dec-POMDP . . . . .	7
2.3.	Multi-Layer Perceptron . . . . .	9
2.4.	Stochastic Gradient Descent . . . . .	9
2.5.	Rekurrentes neuronales Netz . . . . .	11
2.6.	Convolutional Neural Network . . . . .	12
3.1.	Pommerman Spielfeld . . . . .	17
4.1.	RIAL vs. DIAL Architektur . . . . .	24
4.2.	DIAL mit Value Decomposition (Networks) . . . . .	25
4.3.	Kommunikationsfluss im CommNet . . . . .	26
4.4.	Illustration des Spiels „Wer ist es?“ und Architektur des Modells . . . . .	29
4.5.	Baumsuche mit pessimistischem Szenario . . . . .	31
4.6.	Planner Imitation mit A3C . . . . .	31
4.7.	COMBAT Framework . . . . .	33
4.8.	Backplay in einer Labyrinthdomäne . . . . .	34
5.1.	DRQN <sub>Act</sub> des NoCom-Teams und DRQN <sub>Com</sub> des Com-Teams . . . . .	39
5.2.	Illustration des NoCom-Teams . . . . .	39
5.3.	DRQN <sub>Act</sub> des Com-Teams . . . . .	40
5.4.	Illustration des Com-Teams . . . . .	40
6.1.	Reward im Verlauf des Trainings . . . . .	44
6.2.	Performance aller Modelle im Vergleich . . . . .	46
6.3.	Verteilung der Nachrichten pro Szenario gegen RandomAgents . . . . .	49
6.4.	Verteilung der Nachrichten pro Szenario gegen RandomNoBombAgents . . . . .	50
6.5.	Verteilung der Nachrichten pro Szenario gegen SimpleAgents . . . . .	51
6.6.	SC im Verlauf des Trainings gegen RandomAgents . . . . .	53
6.7.	SC: $p(m)$ und $p(a m)$ gegen RandomAgents . . . . .	53
6.8.	IC im Verlauf des Trainings gegen RandomAgents . . . . .	56
6.9.	IC: $p(m)$ und $p(a m)$ gegen RandomAgents . . . . .	57
6.10.	Performance des Com-Teams ohne Kommunikation . . . . .	59
A.1.	t-SNE Plots eines Referential Games . . . . .	65
A.2.	Beispielbilder und entsprechende Nachrichten eines Referential Games . . . . .	67
A.3.	t-SNE Plot zu „Wer ist es?“ . . . . .	67
A.4.	SC im Verlauf des Trainings gegen RandomNoBombAgents . . . . .	68
A.5.	SC: $p(m)$ und $p(a m)$ gegen RandomNoBombAgents . . . . .	68
A.6.	IC im Verlauf des Trainings gegen RandomNoBombAgents . . . . .	69

## ABBILDUNGSVERZEICHNIS

A.7. IC: $p(m)$ und $p(a m)$ gegen <b>RandomNoBombAgents</b> . . . . .	69
A.8. SC im Verlauf des Trainings gegen <b>SimpleAgents</b> . . . . .	70
A.9. SC: $p(m)$ und $p(a m)$ gegen <b>SimpleAgents</b> . . . . .	70
A.10. IC im Verlauf des Trainings gegen <b>SimpleAgents</b> . . . . .	71
A.11. IC: $p(m)$ und $p(a m)$ gegen <b>SimpleAgents</b> . . . . .	71

# **Tabellenverzeichnis**

4.1. Aktionsfilter des Skynet955-Teams . . . . .	32
4.2. Reward Shaping des Skynet955-Teams . . . . .	32
A.1. Performance aller Modelle im Vergleich . . . . .	66



# **Liste der Algorithmen**

1. Heuristik des SimpleAgents . . . . .	20
---	----



# Literaturverzeichnis

- [BGIZ02] BERNSTEIN, DANIEL S., ROBERT GIVAN, NEIL IMMERMAN und SHLOMO ZILBERSTEIN: *The Complexity of Decentralized Control of Markov Decision Processes*. Mathematics of Operations Research, 27(4):819–840, nov 2002.
- [Bot12] BOTTOU, LÉON: *Stochastic Gradient Descent Tricks*. In: *Neural Networks, Tricks of the Trade, Reloaded*, Seiten 421–436. 2012.
- [Bou96] BOUTILIER, CRAIG: *Planning, learning and coordination in multiagent decision processes*. Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge, Seiten 195–210, 1996.
- [BP92] BENVENUTO, N. und F. PIAZZA: *On the complex backpropagation algorithm*. IEEE Transactions on Signal Processing, 40(4):967–969, apr 1992.
- [BS10] BALAJI, P. G. und D. SRINIVASAN: *An Introduction to Multi-Agent Systems*. In: *Studies in Computational Intelligence*, Band 310, Seiten 1–27. 2010.
- [CB98] CLAUS, CAROLINE und CRAIG BOUTILIER: *The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems*. AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence, Seiten 746–752, feb 1998.
- [CHK95] CROSS, S.S., R.F. HARRISON und R.L. KENNEDY: *Introduction to neural networks*. The Lancet, 346(8982):1075–1079, oct 1995.
- [Cil10] CILIMKOVIC, MIRZA: *Neural Networks and Back Propagation Algorithm*. 2010.
- [CLL<sup>+</sup>18] CAO, KRIS, ANGELIKI LAZARIDOU, MARC LANCTOT, JOEL Z LEIBO, KARL TUYLS und STEPHEN CLARK: *Emergent Communication through Negotiation*. apr 2018.
- [CYL16] CHEN, CLARE, VINCENT YING und DILLON LAIRD: *Deep Q-Learning with Recurrent Neural Networks*. Stanford CS229 Course Report, 2016.
- [EBL<sup>+</sup>19] ECCLES, TOM, YORAM BACHRACH, GUY LEVER, ANGELIKI LAZARIDOU und THORE GRAEPEL: *Biases for Emergent Communication in Multi-agent Reinforcement Learning*. (NeurIPS 2019), dec 2019.
- [EDKC17] EVTIMOVA, KATRINA, ANDREW DROZDOV, DOUWE KIELA und KYUNG-HYUN CHO: *Emergent Communication in a Multi-Modal, Multi-Step Referential Game*. may 2017.
- [Ego16] EGOROV, MAXIM: *Multi-agent deep reinforcement learning*. CS231n: Convolutional Neural Networks for Visual Recognition, jul 2016.
- [FAdFW16] FOERSTER, JAKOB N, YANNIS M ASSAEL, NANDO DE FREITAS und SHIMON WHITESON: *Learning to Communicate with Deep Multi-Agent Reinforcement Learning*. may 2016.

## LITERATURVERZEICHNIS

- [FNF<sup>+</sup>17] FOERSTER, JAKOB, NANTAS NARDELLI, GREGORY FARQUHAR, TRIANTAFYLLOS AFOURAS, PHILIP H. S. TORR, PUSHMEET KOHLI und SHIMON WHITESON: *Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning*. feb 2017.
- [FR96] FARRELL, JOSEPH und MATTHEW RABIN: *Cheap Talk*. Journal of Economic Perspectives, 10(3):103–118, aug 1996.
- [GHLKT19] GAO, CHAO, PABLO HERNANDEZ-LEAL, BILAL KARTAL und MATTHEW E. TAYLOR: *Skynet: A Top Deep RL Agent in the Inaugural Pommerman Team Competition*. apr 2019.
- [GKHLT19] GAO, CHAO, BILAL KARTAL, PABLO HERNANDEZ-LEAL und MATTHEW E. TAYLOR: *On Hard Exploration for Reinforcement Learning: a Case Study in Pommerman*. jul 2019.
- [Gur97] GURNEY, KEVIN: *An Introduction to Neural Networks*. oct 1997.
- [HS97] HOCHREITER, SEPP und JÜRGEN SCHMIDHUBER: *Long Short-Term Memory*. Neural Computation, 9(8):1735–1780, nov 1997.
- [HS15] HAUSKNECHT, MATTHEW und PETER STONE: *Deep Recurrent Q-Learning for Partially Observable MDPs*. jul 2015.
- [HT17] HAVRYLOV, SERHII und IVAN TITOV: *Emergence of Language with Multi-agent Games: Learning to Communicate with Sequences of Symbols*. may 2017.
- [JKJG16] JORGE, EMILIO, MIKAEL KÅGEBÄCK, FREDRIK D. JOHANSSON und EMIL GUSTAVSSON: *Learning to Play Guess Who? and Inventing a Grounded Language as a Consequence*. nov 2016.
- [JP97] JANSSEN, THEO M V und BH PARTEE: *Compositionality*. Handbook of Logic and Linguistics, Seiten 417–473, 1997.
- [KB14] KINGMA, DIEDERIK P. und JIMMY BA: *Adam: A Method for Stochastic Optimization*. dec 2014.
- [KHLT18] KARTAL, BILAL, PABLO HERNANDEZ-LEAL und MATTHEW E. TAYLOR: *Using Monte Carlo Tree Search as a Demonstrator within Asynchronous Deep RL*. nov 2018.
- [KLC98] Kaelbling, Leslie Pack, Michael L Littman und Anthony R Cassandra: *Planning and acting in partially observable stochastic domains*. Artificial Intelligence, 101(1-2):99–134, may 1998.
- [KWR<sup>+</sup>16] KEMPKA, MICHAL, MAREK WYDMUCH, GRZEGORZ RUNC, JAKUB TOCZEK und WOJCIECH JASKOWSKI: *ViZDoom: A Doom-based AI research platform for visual reinforcement learning*. In: *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, Seiten 1–8. IEEE, sep 2016.
- [LBE15] LIPTON, ZACHARY C., JOHN BERKOWITZ und CHARLES ELKAN: *A Critical Review of Recurrent Neural Networks for Sequence Learning*. may 2015.
- [LC16] LAMPLE, GUILLAUME und DEVENDRA SINGH CHAPLOT: *Playing FPS Games with Deep Reinforcement Learning*. sep 2016.

- [LFB<sup>+</sup>19] LOWE, RYAN, JAKOB FOERSTER, Y-LAN BOUREAU, JOELLE PINEAU und YANN DAUPHIN: *On the Pitfalls of Measuring Emergent Communication.* mar 2019.
- [LHTC18] LAZARIDOU, ANGELIKI, KARL MORITZ HERMANN, KARL TUYLS und STEPHEN CLARK: *Emergence of Linguistic Communication from Referential Games with Symbolic and Pixel Input.* apr 2018.
- [Lin93] LIN, LONG-JI: *Reinforcement learning for robots using neural networks.* Doktorarbeit, 1993.
- [LPB16] LAZARIDOU, ANGELIKI, ALEXANDER PEYSAKHOVICH und MARCO BARONI: *Multi-Agent Cooperation and the Emergence of (Natural) Language.* dec 2016.
- [LUTG17] LAKE, BRENDEN M., TOMER D. ULLMAN, JOSHUA B. TENENBAUM und SAMUEL J. GERSHMAN: *Building machines that learn and think like people.* Behavioral and Brain Sciences, 40, apr 2017.
- [MJB15] MIKOLOV, TOMAS, ARMAND JOULIN und MARCO BARONI: *A Roadmap towards Machine Intelligence.* Seiten 29–61, nov 2015.
- [MKS<sup>+</sup>13] MNIIH, VOLODYMYR, KORAY KAVUKCUOGLU, DAVID SILVER, ALEX GRAVES, IOANNIS ANTONOGLOU, DAAN WIERSTRA und MARTIN RIEDMILLER: *Playing Atari with Deep Reinforcement Learning.* dec 2013.
- [MKS<sup>+</sup>15] MNIIH, VOLODYMYR, KORAY KAVUKCUOGLU, DAVID SILVER, ANDREI A. RUSU, JOEL VENESS, MARC G. BELLEMARE, ALEX GRAVES, MARTIN RIEDMILLER, ANDREAS K. FIDJELAND, GEORG OSTROVSKI, STIG PETERSEN, CHARLES BEATTIE, AMIR SADIK, IOANNIS ANTONOGLOU, HELEN KING, DHARSHAN KUMARAN, DAAN WIERSTRA, SHANE LEGG und DEMIS HASSABIS: *Human-level control through deep reinforcement learning.* Nature, 518(7540):529–533, feb 2015.
- [MSS<sup>+</sup>18] MALYSHEVA, ALEKSANDRA, TEGG TAEKYONG SUNG, CHAE-BONG SOHN, DANIEL KUDENKO und ALEKSEI SHPILMAN: *Deep Multi-Agent Reinforcement Learning with Relevance Graphs.* nov 2018.
- [MV19] MORENO-VERA, FELIPE: *Performing Deep Recurrent Double Q-Learning for Atari Games.* aug 2019.
- [OA16] OLIEHOEK, FRANS A. und CHRISTOPHER AMATO: *A Concise Introduction to Decentralized POMDPs.* SpringerBriefs in Intelligent Systems. Springer International Publishing, Cham, oct 2016.
- [ON15] O'SHEA, KEIRON und RYAN NASH: *An Introduction to Convolutional Neural Networks.* nov 2015.
- [OSV11] OLIEHOEK, FRANS A., MATTHIJS T. J. SPAAN und NIKOS VLASSIS: *Optimal and Approximate Q-value Functions for Decentralized POMDPs.* 2, oct 2011.
- [OT19] OSOGAMI, TAKAYUKI und TOSHIHIRO TAKAHASHI: *Real-time tree search with pessimistic scenarios.* feb 2019.

## LITERATURVERZEICHNIS

- [PLGD<sup>+</sup>19] PEREZ-LIEBANA, DIEGO, RALUCA D GAINA, OLVE DRAGESET, ERCÜMEN ILHAN, MARTIN BALLA und SIMON M LUCAS: *Analysis of Statistical Forward Planning Methods in Pommerman*. Proceedings of the Artificial intelligence and Interactive Digital Entertainment (AIIDE), 2019.
- [PPYG18] PENG, PENG, LIANG PANG, YUFENG YUAN und CHAO GAO: *Continual Match Based Training in Pommerman: Technical Report*. dec 2018.
- [RB19] ROMAC, CLÉMENT und VINCENT BÉRAUD: *Deep Recurrent Q-Learning vs Deep Q-Learning on a simple Partially Observable Markov Decision Process with Minecraft*. mar 2019.
- [REH<sup>+</sup>18] RESNICK, CINJON, WES ELDRIDGE, DAVID HA, DENNY BRITZ, JAKOB FOERSTER, JULIAN TOGELIUS, KYUNGHYUN CHO und JOAN BRUNA: *Pommerman: A Multi-Agent Playground*. sep 2018.
- [RN09] RUSSELL, STUART und PETER NORVIG: *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, 3rd Auflage, nov 2009.
- [RRK<sup>+</sup>18] RESNICK, CINJON, ROBERTA RAILEANU, SANYAM KAPOOR, ALEXANDER PEYSAKHOVICH, KYUNGHYUN CHO und JOAN BRUNA: *Backplay: 'Man muss immer umkehren'*. jul 2018.
- [RSdW<sup>+</sup>18] RASHID, TABISH, MIKAYEL SAMVELYAN, CHRISTIAN SCHROEDER DE WITT, GREGORY FARQUHAR, JAKOB FOERSTER und SHIMON WHITESON: *QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning*. mar 2018.
- [Rud16] RUDER, SEBASTIAN: *An overview of gradient descent optimization algorithms*. sep 2016.
- [RZL17] RAMACHANDRAN, PRAJIT, BARRET ZOPH und QUOC V LE: *Searching for Activation Functions*. oct 2017.
- [SB98] SUTTON, R.S. und A.G. BARTO: *Reinforcement Learning: An Introduction*. IEEE Transactions on Neural Networks, 9(5), sep 1998.
- [SDS<sup>+</sup>19] SINGH, Ms. NAVYA, MR. ANSHUL DHULL, MR. BARATH MOHAN. S, MR. BHAVISH PAHWA und Ms. KOMAL SHARMA: *Automated Gaming Pommerman: FFA*. 2019.
- [She20] SHERSTINSKY, ALEX: *Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network*. Physica D: Nonlinear Phenomena, 404:132306, mar 2020.
- [Sim18] SIMONINI, THOMAS: *Improvements in Deep Q Learning: Dueling Double DQN, Prioritized Experience Replay, and fixed...* <https://www.freecodecamp.org/news/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682/>, 2018. [Online; aufgerufen am 11. März 2020].
- [SKP97] SVOZIL, DANIEL, VLADIMIR KVASNICKA und JIRI POSPICHAL: *Introduction to multi-layer feed-forward neural networks*. Chemometrics and Intelligent Laboratory Systems, 39(1):43–62, nov 1997.

- [SLG<sup>+</sup>17] SUNEHAG, PETER, GUY LEVER, AUDRUNAS GRUSLYS, WOJCIECH MARIAN CZARNECKI, VINICIUS ZAMBALDI, MAX JADERBERG, MARC LANCTOT, NICOLAS SONNERAT, JOEL Z. LEIBO, KARL TUYLS und THORE GRAEPEL: *Value-Decomposition Networks For Cooperative Multi-Agent Learning*. jun 2017.
- [Spa12] SPAAN, MATTHIJS T. J.: *Partially Observable Markov Decision Processes*. Seiten 387–414. 2012.
- [SSF16] SUKHBAATAR, SAINBAYAR, ARTHUR SZLAM und ROB FERGUS: *Learning Multiagent Communication with Backpropagation*. may 2016.
- [SST18] SHAH, DHRUV, NIHAL SINGH und CHINMAY TALEGAONKAR: *Multi-Agent Strategies for Pommerman*. 2018.
- [Tan93] TAN, MING: *Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents*. In: *Machine Learning Proceedings 1993*, Seiten 330–337. Elsevier, 1993.
- [TMK<sup>+</sup>15] TAMPUU, ARDI, TAMBET MATIISEN, DORIAN KODELJA, ILYA KUZOVKIN, KRISTJAN KORJUS, JUHAN ARU, JAAN ARU und RAUL VICENTE: *Multi-agent Cooperation and Competition with Deep Reinforcement Learning*. nov 2015.
- [vH10] HASSELT, HADO VAN: *Double Q-learning*. In: CULOTTA, J. D. LAFFERTY, C. K. I. WILLIAMS, J. SHAWE-TAYLOR, R. S. ZEMEL und A. CULOTTA (Herausgeber): *Advances in Neural Information Processing Systems 23*, Seiten 2613 – 2621. Curran Associates, Inc., 2010.
- [vHGS15] HASSELT, HADO VAN, ARTHUR GUEZ und DAVID SILVER: *Deep Reinforcement Learning with Double Q-learning*. sep 2015.
- [VVB<sup>+</sup>20] VANNESTE, SIMON, ASTRID VANNESTE, STIG BOSMANS, SIEGFRIED MERCELIS und PETER HELLINCKX: *Learning to Communicate with Multi-agent Reinforcement Learning Using Value-Decomposition Networks*. Seiten 736–745. 2020.
- [WD92] WATKINS, CHRISTOPHER J. C. H. und PETER DAYAN: *Q-learning*. Machine Learning, 8(3-4):279–292, may 1992.
- [WLBF18] WANG, TINGWU, RENJIE LIAO, JIMMY BA und SANJA FIDLER: *Nervenet: Learning structured policy with graph neural networks*. 6th International Conference on Learning Representations, 2018.
- [WSH<sup>+</sup>16] WANG, ZIYU, TOM SCHAUL, MATTEO HESSEL, HADO VAN HASSELT, MARC LANCTOT und NANDO DE FREITAS: *Dueling Network Architectures for Deep Reinforcement Learning*. nov 2016.
- [ZGM<sup>+</sup>18] ZHOU, HONGWEI, YICHEN GONG, LUVNEESH MUGRAI, AHMED KHALIFA, ANDY NEALEN und JULIAN TOGELIUS: *A hybrid search agent in pommerman*. In: *Proceedings of the 13th International Conference on the Foundations of Digital Games - FDG '18*, Seiten 1–4, New York, New York, USA, 2018. ACM Press.