A Graph Neural Network Reasoner for Game Description Language

Vortrag von Numan Tok
Frankfurt, 13.07.2023

INHALT

1 Einführung

Aktuelle Methoden, Einschränkungen, Grundlagen, Forschungsbeiträge

2 Neuronaler Reasoner

Darstellung des Spielzustandes, Erzeugung von Trainingsdaten, GNN-Architektur, Überblick

3 Ergebnisse

Einzeltraining, Transfer, gemischtes und sequentielles Training, RG-Abflachung

4 Dikussion & Fazit

Einschränkungen, Kritik, Fazit

1

Einführung

Aktuelle Methoden, Einschränkungen,
Grundlagen, Forschungsbeiträge

1.1 - Aktuelle Methoden

AlphaGo, AlphaGoZero, AlphaZero:

- Nutzen Monte-Carlo Baumsuche (MCTS), um sukzessiv zufällige Spielbäume zu erstellen und verschiedene Zugfolgen zu simulieren.
- Neuronale Netze (NN) erzeugen Wahrscheinlichkeitsverteilungen der nächsten Züge und bewerten diese anschließend
  →NN durch Reinforcement Learning (RL) trainiert
- Erreichen übermenschliche Leistungen

Warum MCTS & ML ?

1.1 - Aktuelle Methoden

TicTacToe

- Hat 765 mögliche Spielzustände
- Es existieren min. 72.657 Strategien, mit denen man nicht verlieren kann

Schach

- Hat Schätzungsweise 10^{40} bis 10^{50} mögliche Spielzustände
- Es existieren nach Schätzungen ca. 10^{120} mögliche Spielabläufe

Größenvergleich

- Nach Schätzungen beträgt die Anzahl aller Atome auf der Erde 10^{50} und im gesamten Universum 10^{80}

1.1 - Aktuelle Methoden

Warum MCTS und ML?

- → Spielbäume können extrem komplex werden
- → Erstellen vollständiger Spielbäume würde enorme Rechenressourcen benötigen
- → Weiterhin interessant für Anwendungen mit unvollständigen Informationen

1.1 - Aktuelle Methoden

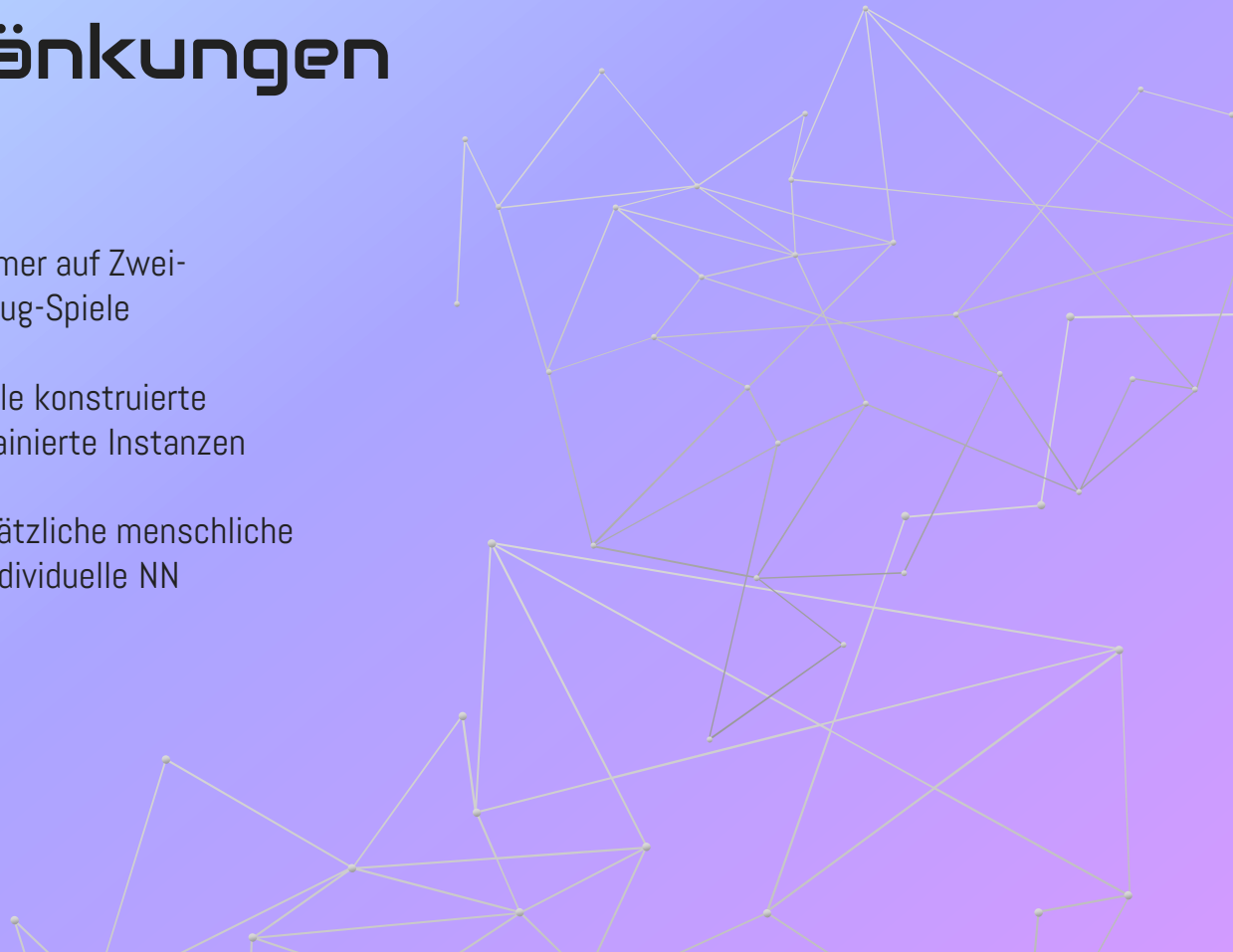
AlphaGo (2016)
- Lernte Go zunächst durch Expertenspiele.
- Lernte anschließend alleine durch RL weiter.

AlphaGoZero (2017)
- Lernte Go ausschließlich durch RL.

AlphaZero (2018)
- Allgemeinere Architektur, welche nicht mehr auf Go spezialisiert ist.
- Lernt Go, Schach und Shogi mit derselben Architektur, jedoch auf unterschiedlichen Instanzen (separat trainierte Netze).

1.2 - Einschränkungen

- Spieltypen begrenzen sich immer auf Zwei-Personen-, Nullsummen und Zug-Spiele
- Entweder für individuelle Spiele konstruierte Architekturen oder separat trainierte Instanzen
- Für andere Spiele werden zusätzliche menschliche Informationen benötigt, um individuelle NN zukonstruieren



1.3 - Grundlagen

GGP (General Game Playing):
Agenten beherrschen jedes Spiel mit:

- Nur Regeln gegeben
- Beliebiger Spielerzahl
- Simultanen Spielzügen
- Nicht-Nullsummenspielen

GGP-Agent (bisher):

1. Verarbeitet Spielregeln mit logikbasierten Reasonern, um Spielzustände zu extrahieren
2. Erstellt sukzessiv einen Spielbaum
3. Ermittelt besten Zug

1.3 - Grundlagen

GDL (Game Description Language)
= Logische Sprache, welche Spielregeln kodiert

- In GDL werden Spielzustände durch sogenannte “Fluents” dargestellt, welche dynamische Eigenschaften wie (cell 1 1 o), (cell 1 1 b), (control oplayer) sind und durch Aktionen beeinflusst werden können

- Der Nächste Spielzustand (next-state) kann logisch abgeleitet werden über die Kombination der

a. Fluents, welche im aktuellen Zustand gegeben sind
b. Aktionen der Spieler

1.3 - Grundlagen

GDL Tic-Tac-Toe Beispiel 1:

Im nächsten Zustand ist (cell 1 1 o) wahr, wenn oplayer Aktion (mark 1 1) ausführt. Im aktuellen Zustand ist (cell 1 1 b) wahr.

GDL Tic-Tac-Toe Beispiel 2:

Es ist oplayer erlaubt Aktion (mark 1 1) auszuführen, wenn (cell 1 1 b) und (control oplayer) im aktuellen Zustand wahr sind.

```
(<= (next (cell 1 1 o))
    (does oplayer (mark 1 1))
    (true (cell 1 1 b)))
```

```
(<= (legal oplayer (mark 1 1))
    (true (cell 1 1 b))
    (true (control oplayer)))
```

Figure 2: Fragment of GDL description of Tic-Tac-Toe.

Liste vollständiger GDL-Regeln

1.4 - Forschungsbeiträge

Zielfragestellung:

Kann die logische Schlussfolgerung in GDL mithilfe eines NN basierten Ansatzes mit hoher Genauigkeit approximiert werden?

1.4 - Forschungsbeiträge

Kann die logische Schlussfolgerung in GDL mithilfe eines NN basierten Ansatzes mit hoher Genauigkeit approximiert werden?

Darstellung

Eine allgemeine, Spiel unabhängige graphbasierte Darstellung von Spielzuständen, beschrieben in GDL

Trainingsdaten

Methode zur Erzeugung von Trainingsdatensätzen, um die GDL-Schlussfolgerungsaufgabe als ein auf NN basierendes ML-Problem zu gestalten

GNN-Reasoner

Ein GNN-basierter Reasoner, der verschiedene Spielzustände mit hoher Genauigkeit erlernen kann und eine gewisse Fähigkeit zum spielübergreifenden Lernen besitzt

2 Neuronaler Reasoner

Darstellung des Spielzustandes, Erzeugung von Trainingsdaten, GNN-Architektur, Überblick

2.1 – Darstellung des Spielzustands

Bisher:

- Zustände werden abhängig von der NN-Architektur in Vektor-, Matrix- oder Tensorform dargestellt
- Ein Schach- oder Go-Brett kann z.B. direkt in eine Matrix übersetzt und als CNN-Input genutzt werden

2.1 – Darstellung des Spielzustands

Bisher:

- Zustände werden abhängig von der NN-Architektur in Vektor-, Matrix- oder Tensorform dargestellt
- Ein Schach- oder Go-Brett kann z.B. direkt in eine Matrix übersetzt und als CNN-Input genutzt werden

Effektiv & effizient

2.1 – Darstellung des Spielzustands

Bisher:

- Zustände werden abhängig von der NN-Architektur in Vektor-, Matrix- oder Tensorform dargestellt
- Ein Schach- oder Go-Brett kann z.B. direkt in eine Matrix übersetzt und als CNN-Input genutzt werden

Effektiv & effizient

Haben feste Größe, dadurch nicht kompatibel zwischen verschiedenen Spielen

2.1 – Darstellung des Spielzustands

Bisher:

- Zustände werden abhängig von der NN-Architektur in Vektor-, Matrix- oder Tensorform dargestellt
- Ein Schach- oder Go-Brett kann z.B. direkt in eine Matrix übersetzt und als CNN-Input genutzt werden

Effektiv & effizient

Haben feste Größe, dadurch nicht kompatibel zwischen verschiedenen Spielen

Spielzustände verschiedener Spiele weisen unterschiedliche Eigenschaften auf → Einfache Spielfelddarstellungen sind nicht universell

2.1 – Darstellung des Spielzustands

Lösung: Instanziierter Regel-Graph (IRG)

- Regel-Graph (RG): graphenbasierte Darstellung der der Spielregeln in GDL
- IRG: RG wird durch Markierung bestimmter Bereich auf einen eindeutigen Zustand instanziiert

→ Erlaubt es, dass Zustände unterschiedlicher Spiele als gültige Eingaben für ein und dasselbe NN genutzt werden können

2.1 – Darstellung des Spielzustands

Regelgraphen:

- Gefärbte, gerichtete Graphen $G(V, E)$ mit Knoten V und Kanten E, die aus vier Knotentypen bestehen:
- **Schlüsselwortknoten** = stellen vordefinierten GDL-Schlüsselwort dar, wie z.B. legal, next, does, etc.
- **Prädikatsknoten** = repräsentieren Nicht-Schlüsselwortknoten, d.h. speziell für das Spiel deklarierte Prädikate (z.B. "mark", "cell")
- **Label-Knoten** = identifizieren Prädikate, die in der gesamten Beschreibung gleich sind, ohne lexikalische Labels zu behalten. Dazu wird eine Kante von Label-Knoten zu jedem Prädikat gezogen, das eine Instanz des besagten Labels ist.
- **Label-Argument-Knoten** = Identifizieren die Position von Argumenten in einem Prädikat, wobei die Positionsinformationen von Prädikaten in den Regelgraphen kodiert werden. Eine Kante wird zwischen dem Label-Argument-Knoten und jeder Instanz eines Arguments gezogen, das seine Position angibt. Kanten werden zwischen Prädikaten und ihren Argumenten gezeichnet, um einen Syntaxbaum zu erstellen

2.1 – Darstellung des Spielzustands

Regelgraph des vorherigen Beispiels:

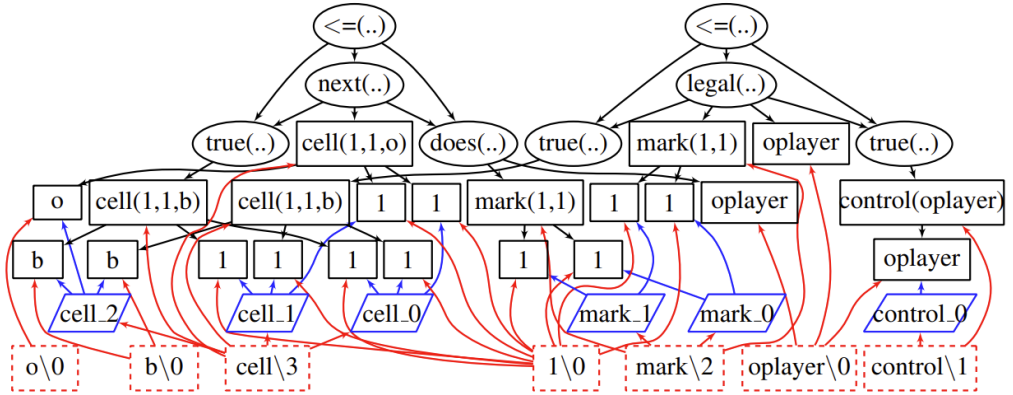


Figure 3: Rule graph of the fragment in Figure 2. Lexicographic information such as predicate names are not stored and only shown here to aid in visualisation.

Schlüsselwortknoten

Prädikatsknoten

Label-Knoten

Label-Argument-Knoten

2.1 – Darstellung des Spielzustands

Isomorphie in Regelgraphen:

Da nur Beziehungen und Positionen der verschiedenen Prädikate enthalten, ohne die tatsächlichen Prädikatnamen, ist die Darstellung isomorph zur Prädikatsverschlüsselung.

- → Zwei verschiedene Beschreibungen desselben Spiels, bei denen die Prädikatnamen geändert wurden, weisen dieselbe RG-Darstellung auf.
- → Ähnliche Merkmale, die in verschiedenen Spielen definiert sind, weisen ähnliche Untergraphen in ihren RG auf.

2.1 – Darstellung des Spielzustands

Instanziierte Regelgraphen:

- Allgemeine Zustandsdarstellung, erzeugt durch Instanziierung des RG eines Spiels
- Durch eindeutige Knotenmarkierung für jeden Zustand auf einen bestimmten Zustand lokalisiert
- Markierung basiert auf gegebenen Zustand $S = \{f_1, \dots, f_n\}$, bestehend aus Fluents f_i

Markierungsregel:

Überprüfe für jeden Knoten $n \in V$:

✓ n ist ein Prädikatsknoten
✓ $n \in S$
✓ Der Elternknoten e von n entspricht dem GDL-Schlüsselwort „$true$“
→ Falls alles zutreffend, markiere e, n und alle Kindknoten von n als wahr (1)
→ Verbleibende Knoten werden als falsch (0) markiert

2.1 – Darstellung des Spielzustands

Beispiel: $S = \{cell(1,1,b), \dots\}$

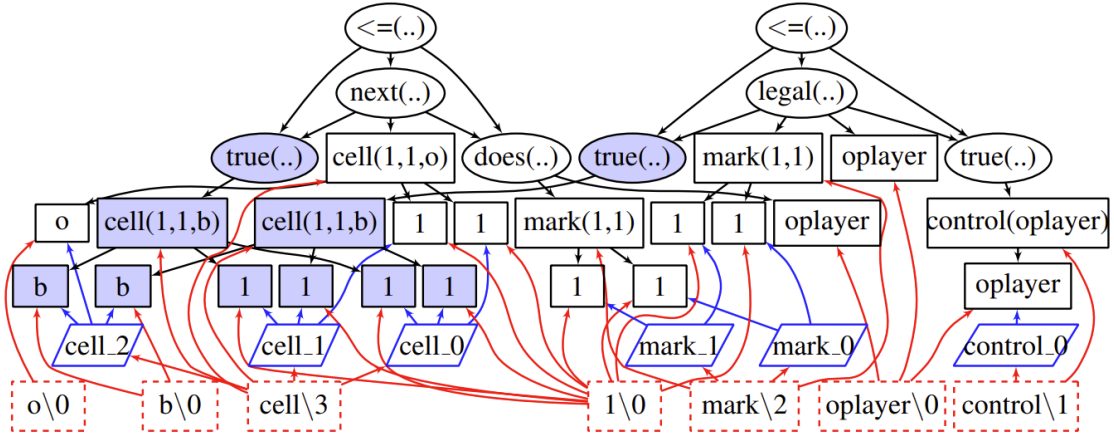


Figure 4: An example of an instantiated rule graph

Blau schattiert = wahr

2.1 – Darstellung des Spielzustands

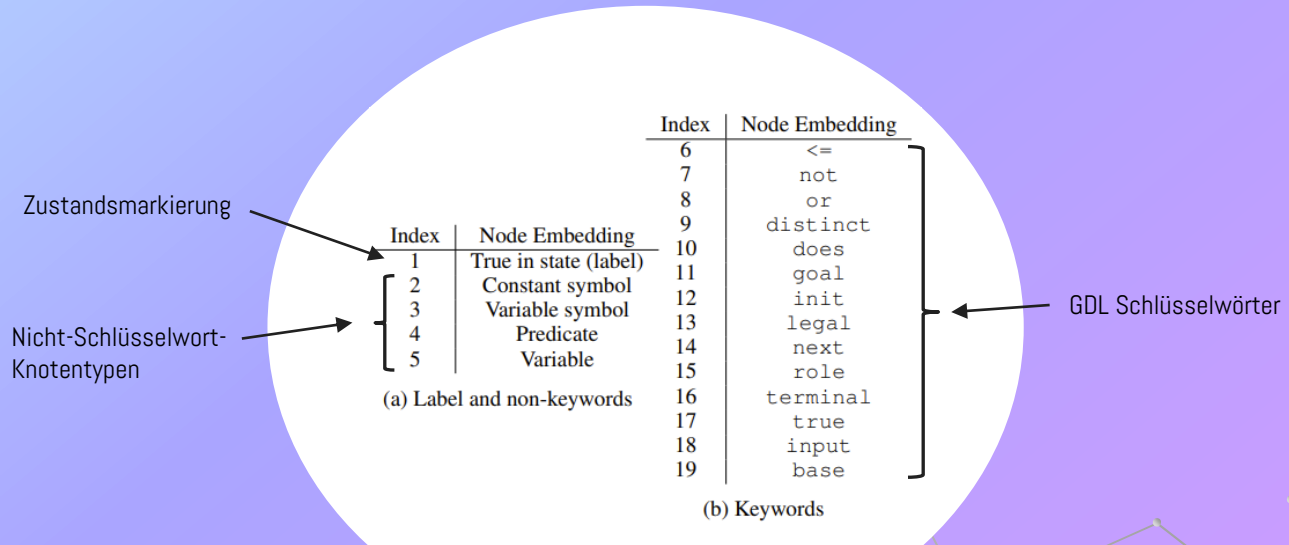
Knotenvektorisierung:

- Beschriftung und Knotentypen des IRG werden in einen Vektor umgewandelt
- Für jeden Knoten $n \in V$ konstruieren wir einen Vektor $\vec{v^{n}} \in \{0,1\}^{19}$
- Die vektorisierten Knoten ergeben dann gemeinsam eine Matrix $X \in \{0,1\}^{|V| \times 19}$
- X wird als GNN-Eingabe zusammen mit der Adjazenzmatrix A verwendet

2.1 – Darstellung des Spielzustands

Knotenvektorisierung:

- Beschriftung und Knotentypen des IRG werden in einen Vektor umgewandelt
- Für jeden Knoten $n \in V$ konstruieren wir einen Vektor $v^{\rightarrow n} \in \{0,1\}^{19}$
- Die vektorisierten Knoten ergeben dann gemeinsam eine Matrix $X \in \{0,1\}^{|V| \times 19}$
- X wird als GNN-Eingabe zusammen mit der Adjazenzmatrix A verwendet



Index	Node Embedding
1	True in state (label)
2	Constant symbol
3	Variable symbol
4	Predicate
5	Variable

(a) Label and non-keywords

Index	Node Embedding
6	<=
7	not
8	or
9	distinct
10	does
11	goal
12	init
13	legal
14	next
15	role
16	terminal
17	true
18	input
19	base

(b) Keywords

2.1 – Darstellung des Spielzustands

Knotenvektorisierung:

- Beschriftung und Knotentypen des IRG werden in einen Vektor umgewandelt
- Für jeden Knoten $n \in V$ konstruieren wir einen Vektor $v^{\rightarrow n} \in \{0,1\}^{19}$
- Die vektorisierten Knoten ergeben dann gemeinsam eine Matrix $X \in \{0,1\}^{|V| \times 19}$
- X wird als GNN-Eingabe zusammen mit der Adjazenzmatrix A verwendet

Index	Node Embedding
1	True in state (label)
2	Constant symbol
3	Variable symbol
4	Predicate
5	Variable

(a) Label and non-keywords

Index	Node Embedding
6	<=
7	not
8	or
9	distinct
10	does
11	goal
12	init
13	legal
14	next
15	role
16	terminal
17	true
18	input
19	base

(b) Keywords

Beispiel

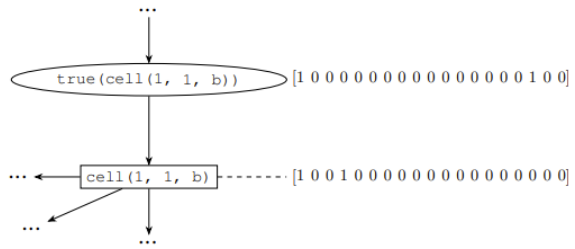


Figure 5: Example vector embedding of two nodes of the instantiated rule graph from Figure 4.

2.2 - Trainingsdaten

Automatische Datensatzerstellung:

- Es werden Trainingsbeispiele benötigt, welche die Spielzustände und die daraus abgeleiteten erlaubten Aktionen und nächsten Spielzustände enthalten.
- GDL-Beschreibung des ausgewählten Spiels wird zum Ausführen von Zufallsspielen verwendet
- Zustände werden als IRG gespeichert und ein Prolog-basierten Reasoner wird verwendet, um logische Schlussfolgerungen zu ziehen und die erlaubten Aktionen und nächsten Spielzustände abzuleiten

2.3 - GNN Architektur

Graph Neural Network (GNN):

- Können Graph-Datenstrukturen als Eingabe und Ausgabe verwenden
- Integrieren Informationen zu Knotenbeziehungen in das Lernen

Graph Attention Network (GAT):

- Eine bestimmte GNN-Architektur, für die sich die Autoren entschieden
- Soll das beste Gleichgewicht zwischen Leistung und Speicherbedarf geboten haben
- Experimente mit „Graph Convolutional Networks“ zeigten eine schlechtere Leistung
- Komplexere Architekturen wie Directed Acyclic Graph NNs schränkten die Größe der nutzbaren Spiele ein während sie nur eine minimale Verbesserung der Leistung boten

2.4 – Überblick

- Aufgabe des Reasoners: Gegeben der GDL sollen 2 Schlussfolgerungsaufgaben gelöst werden
    a) Was sind die erlaubten Aktionen im aktuellen Spielzustand
    b) Wie sehen die nächsten Spielzustände aus, nachdem die Spieler ihre Aktionen tätigen

- Die Traininsdaten mit den Zielwerten für a und b werden mithilfe eines logischen Reasoners und Selbstspiel erstellt

- Während des Trainings werden die Ausgabewahrscheinlichkeiten mit den Zielwerten verglichen und der Fehlerwert für die backpropagation berechnet.

2.4 – Überblick

1. **GDL in einen Regel-Graphen (RG) konvertieren**

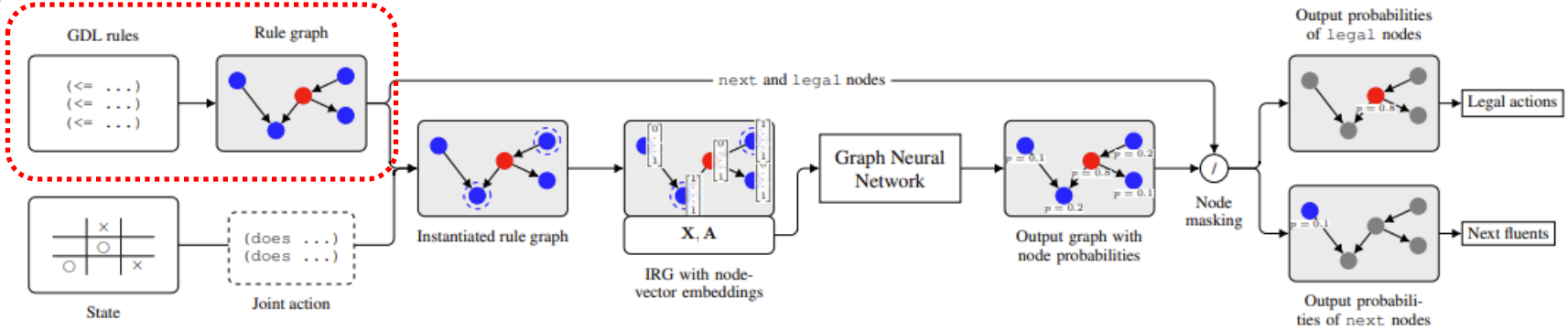


Figure 1: Overview of the Neural Reasoner

2.4 – Überblick

1. GDL in einen Regel-Graphen (RG) konvertieren
2. **RG mit einem Spielzustand kombinieren, um einen Instanziierten RG (IRG) zu erhalten**

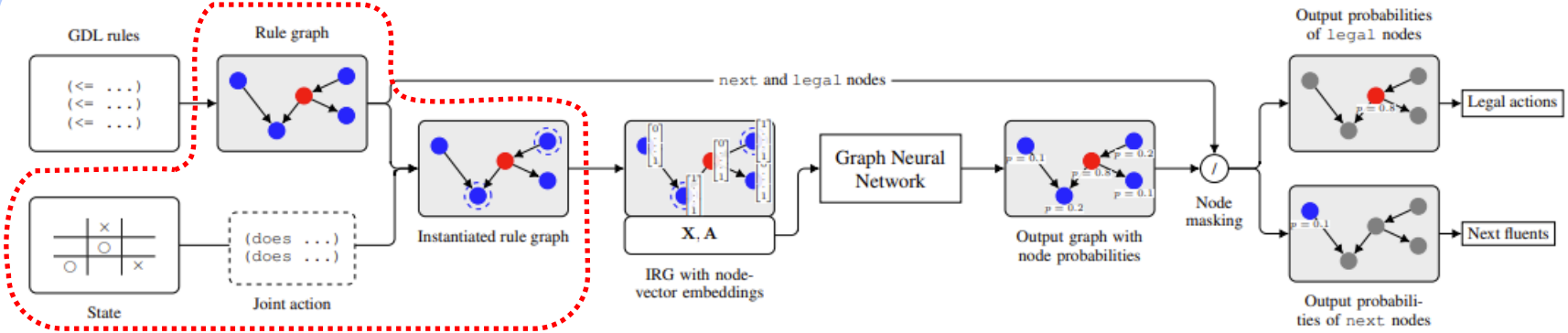


Figure 1: Overview of the Neural Reasoner

2.4 – Überblick

1. GDL in einen Regel-Graphen (RG) konvertieren
2. RG mit einem Spielzustand kombinieren, um einen Instanziierten RG (IRG) zu erhalten
3. **Knoten des IRG in Vektordarstellung übertragen und zu einer Matrix zusammenfügen**

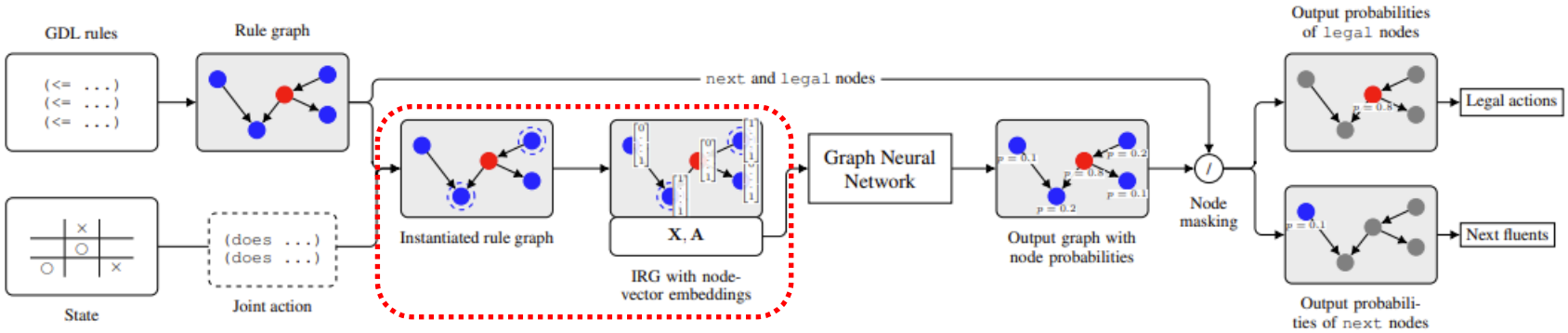


Figure 1: Overview of the Neural Reasoner

2.4 – Überblick

1. GDL in einen Regel-Graphen (RG) konvertieren
2. RG mit einem Spielzustand kombinieren, um einen Instanziierten RG (IRG) zu erhalten
3. Knoten des IRG in Vektordarstellung übertragen und zu einer Matrix zusammenfügen
4. **IRG-Matrix und Adjazenzmatrix werden nun als GNN-Eingabe genutzt**

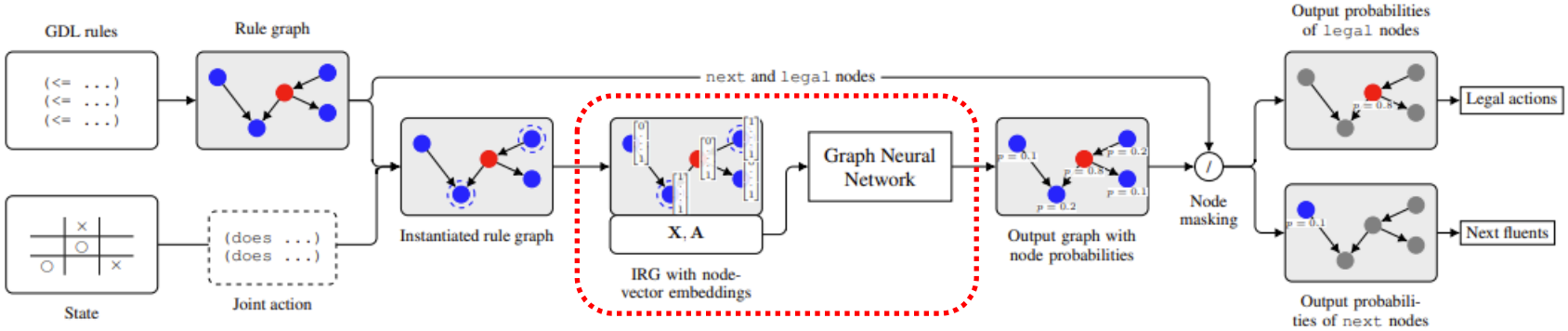


Figure 1: Overview of the Neural Reasoner

2.4 – Überblick

1. GDL in einen Regel-Graphen (RG) konvertieren
2. RG mit einem Spielzustand kombinieren, um einen Instanziierten RG (IRG) zu erhalten
3. Knoten des IRG in Vektordarstellung übertragen und zu einer Matrix zusammenfügen
4. IRG-Matrix und Adjazenzmatrix werden nun als GNN-Eingabe genutzt
5. **Ausgabe des GNN ist ein RG mit Knotenwahrscheinlichkeiten**

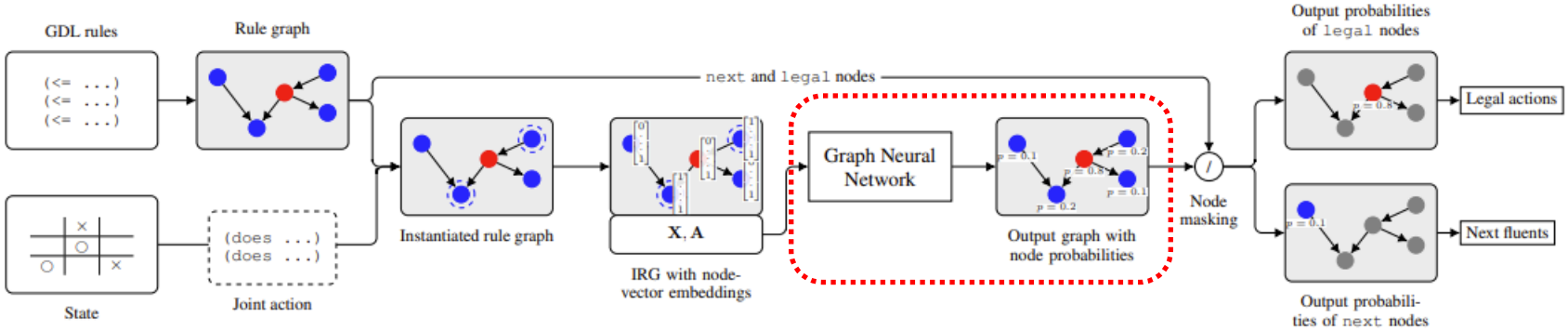


Figure 1: Overview of the Neural Reasoner

2.4 – Überblick

1. GDL in einen Regel-Graphen (RG) konvertieren
2. RG mit einem Spielzustand kombinieren, um einen Instanziierten RG (IRG) zu erhalten
3. Knoten des IRG in Vektordarstellung übertragen und zu einer Matrix zusammenfügen
4. IRG-Matrix und Adjazenzmatrix werden nun als GNN-Eingabe genutzt
5. Ausgabe des GNN ist ein RG mit Knotenwahrscheinlichkeiten
6. **Mithilfe des ursprünglichen RG werden die erlaubten und nächsten Knotentypen bestimmt**

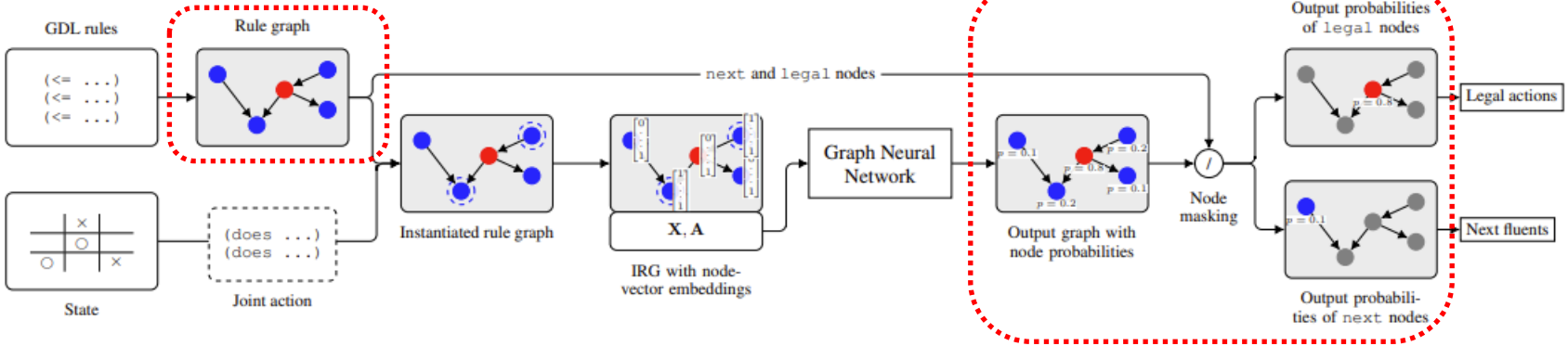


Figure 1: Overview of the Neural Reasoner

2.4 – Überblick

1. GDL in einen Regel-Graphen (RG) konvertieren
2. RG mit einem Spielzustand kombinieren, um einen Instanziierten RG (IRG) zu erhalten
3. Knoten des IRG in Vektordarstellung übertragen und zu einer Matrix zusammenfügen
4. IRG-Matrix und Adjazenzmatrix werden nun als GNN-Eingabe genutzt
5. Ausgabe des GNN ist ein RG mit Knotenwahrscheinlichkeiten
6. Mithilfe des ursprünglichen RG werden die erlaubten und nächsten Knotentypen bestimmt
7. **Knoten mit Wahrscheinlichkeiten über einem bestimmten Grenzwert werden ausgewählt**

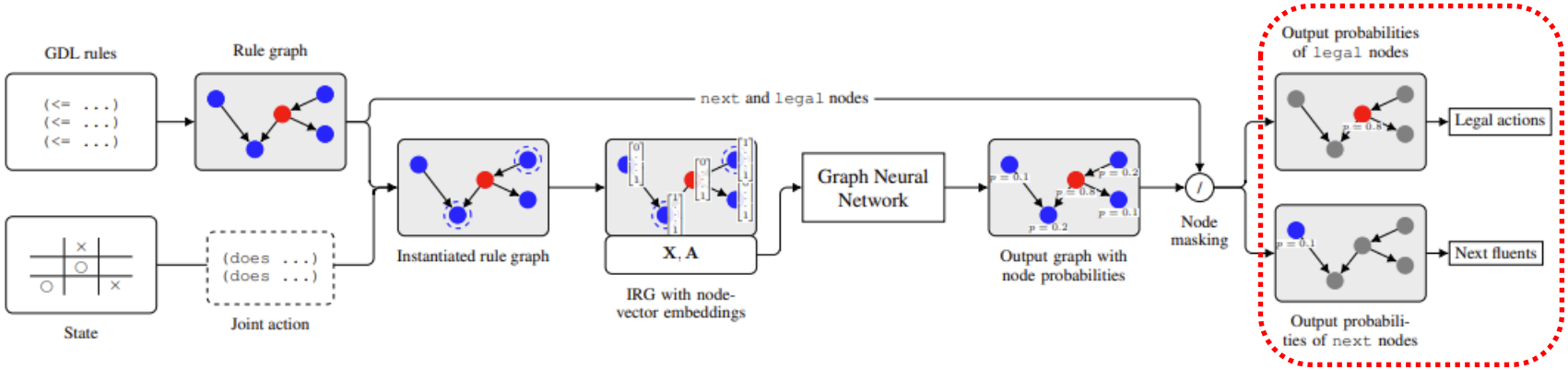


Figure 1: Overview of the Neural Reasoner

3 Ergebnisse & Einschränkungen

Einzeltraining, Transfer, gemischtes und sequentielles Training, RG-Abflachung

3.1 Einzelne Spielversuche

Game	Next fluents (%)	Legal actions (%)
parallelbuttonsandlights	84.35%	100.00%
tictactoe (TTT)	100.00%	100.00%
tictactoelarge (TTT_L)	100.00%	63.69%
doubletictactoe (TTT_D)	100.00%	91.96%
connectfour (C4)	93.58%	100.00%
connectfour3p ($C4_{3p}$)	95.50%	91.75%
blocker	88.94%	100.00%
knightstour	100.00%	100.00%
hamilton	94.94%	100.00%
hanoi6disks	87.55%	72.66%

Table 3: Neural reasoner accuracy over 100 games.

Genauigkeit über jeweils 100 zufällig gespielte Spiele

3.1 Einzelne Spielversuche

Game	Next fluents (%)	Legal actions (%)
parallelbuttonsandlights	84.35%	100.00%
tictactoe (TTT)	100.00%	100.00%
tictactoelarge (TTT_L)	100.00%	63.69%
doubletictactoe (TTT_D)	100.00%	91.96%
connectfour (C4)	93.58%	100.00%
connectfour3p ($C4_{3p}$)	95.50%	91.75%
blocker	88.94%	100.00%
knightstour	100.00%	100.00%
hamilton	94.94%	100.00%
hanoi6disks	87.55%	72.66%

Table 3: Neural reasoner accuracy over 100 games.

Genauigkeit über jeweils 100 zufällig gespielte Spiele

→ In den meisten Fällen bei 90% - 100%

3.1 Einzelne Spielversuche

Game	Next fluents (%)	Legal actions (%)
parallelbuttonsandlights	84.35%	100.00%
tictactoe (TTT)	100.00%	100.00%
tictactoelarge (TTT_L)	100.00%	63.69%
doubletictactoe (TTT_D)	100.00%	91.96%
connectfour (C4)	93.58%	100.00%
connectfour3p ($C4_{3p}$)	95.50%	91.75%
blocker	88.94%	100.00%
knightstour	100.00%	100.00%
hamilton	94.94%	100.00%
hanoi6disks	87.55%	72.66%

Table 3: Neural reasoner accuracy over 100 games.

Genauigkeit über jeweils 100 zufällig gespielte Spiele

→ In den meisten Fällen bei 90% - 100%

→ Auffällige Ausnahmen: Erlaubte Aktionen bei TTT_L und hanoi6disks

3.2 Transfer und gemischtes Lernen

Evaluierung von Spielen, welche der Reasoner nicht gelernt hat

Nächste Spielzüge

Train dataset	TTT	TTT_D	TTT_L	C4
TTT	100.00%	100.00%	100.00%	93.45%
TTT_D	100.00%	100.00%	94.77%	93.53%
TTT_L	40.66%	41.06%	100.0%	40.92%
C4	100.00%	94.68%	92.49%	93.58%

(a)

Table 4: Neural reasoner accuracy over 100 games for (a) `next` and (b) `legal` prediction.

Erlaubte Aktionen

Train dataset	TTT	TTT_D	TTT_L	C4
TTT	100.00%	1.35%	56.02%	100.00%
TTT_D	91.36%	91.96%	64.66%	89.71%
TTT_L	59.02%	14.18%	63.69%	12.85%
C4	100.00%	1.39%	56.47%	100.00%

(b)

Table 4: Neural reasoner accuracy over 100 games for (a) `next` and (b) `legal` prediction.

3.2 Transfer und gemischtes Lernen

Evaluierung von Spielen, welche der Reasoner nicht gelernt hat

→ Bei nächsten Spielzügen, mit Ausnahme von $\mathrm{TTT}_{\mathrm{L}}$, immer bei 90-100% Genauigkeit

→ Bei erlaubten Aktionen sind die Ergebnisse sehr unterschiedlich

Train dataset	TTT	TTT_D	TTT_L	C4
TTT	100.00%	100.00%	100.00%	93.45%
TTT_D	100.00%	100.00%	94.77%	93.53%
TTT_L	40.66%	41.06%	100.0%	40.92%
C4	100.00%	94.68%	92.49%	93.58%

(a)

Table 4: Neural reasoner accuracy over 100 games for (a) `next` and (b) `legal` prediction.

Train dataset	TTT	TTT_D	TTT_L	C4
TTT	100.00%	1.35%	56.02%	100.00%
TTT_D	91.36%	91.96%	64.66%	89.71%
TTT_L	59.02%	14.18%	63.69%	12.85%
C4	100.00%	1.39%	56.47%	100.00%

(b)

Table 4: Neural reasoner accuracy over 100 games for (a) `next` and (b) `legal` prediction.

3.2 Transfer und gemischtes Lernen

Zusätzlich, wird ein gemischtes Training durchgeführt (Spielzustände der 4 Spiele vermischt)

Nächste Spielzüge

Train dataset	TTT	TTT_D	TTT_L	C4
TTT	100.00%	100.00%	100.00%	93.45%
TTT_D	100.00%	100.00%	94.77%	93.53%
TTT_L	40.66%	41.06%	100.0%	40.92%
C4	100.00%	94.68%	92.49%	93.58%
Mixed	100.00%	100.00%	100.00%	98.00%

(a)

Table 4: Neural reasoner accuracy over 100 games for (a) `next` and (b) `legal` prediction.

Erlaubte Aktionen

Train dataset	TTT	TTT_D	TTT_L	C4
TTT	100.00%	1.35%	56.02%	100.00%
TTT_D	91.36%	91.96%	64.66%	89.71%
TTT_L	59.02%	14.18%	63.69%	12.85%
C4	100.00%	1.39%	56.47%	100.00%
Mixed	100.00%	82.62%	55.80%	100.00%

(b)

Table 4: Neural reasoner accuracy over 100 games for (a) `next` and (b) `legal` prediction.

3.2 Transfer und gemischtes Lernen

Zusätzlich, wird ein gemischtes Training durchgeführt (Spielzustände der 4 Spiele vermischt)

→ Das C4 trainierte Netz wird sogar im eigenen Spiel übertroffen
→ Ergebnisse sind in allen Fällen ausgewogen

Train dataset	TTT	TTT_D	TTT_L	C4
TTT	100.00%	100.00%	100.00%	93.45%
TTT_D	100.00%	100.00%	94.77%	93.53%
TTT_L	40.66%	41.06%	100.0%	40.92%
C4	100.00%	94.68%	92.49%	93.58%
Mixed	100.00%	100.00%	100.00%	98.00%

(a)

Table 4: Neural reasoner accuracy over 100 games for (a) `next` and (b) `legal` prediction.

Train dataset	TTT	TTT_D	TTT_L	C4
TTT	100.00%	1.35%	56.02%	100.00%
TTT_D	91.36%	91.96%	64.66%	89.71%
TTT_L	59.02%	14.18%	63.69%	12.85%
C4	100.00%	1.39%	56.47%	100.00%
Mixed	100.00%	82.62%	55.80%	100.00%

(b)

Table 4: Neural reasoner accuracy over 100 games for (a) `next` and (b) `legal` prediction.

3.3 Sequenziell vs. Gemischt

Sequenzielles Lernen verschiedener Spiele wurde ebenfalls getestet, da gemischtes Trainieren nicht immer praktikabel ist.

Training method		TTT	TTT_D	TTT_L	C4
Mixed	Next	100.00%	100.00%	100.00%	98.00%
	Legal	100.00%	82.62%	55.80%	100.00%
Sequential	Next	100.00%	100.00%	92.52%	93.63%
	Legal	100.00%	82.35%	64.45%	100.00%

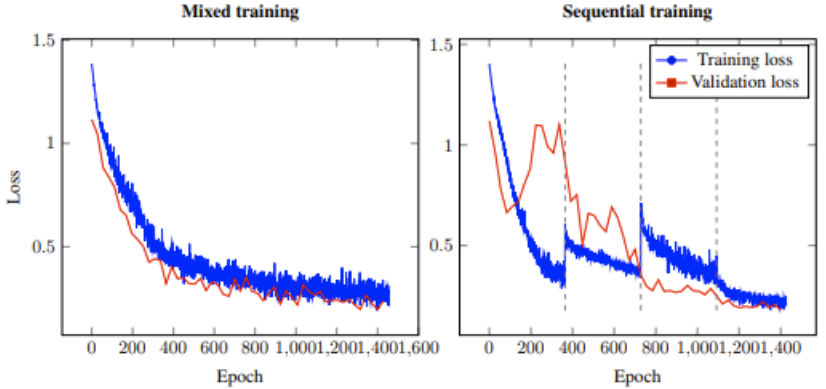
→ Performance ist ähnlich
→ Früher trainierte Spiele werden nicht vergessen

3.3 Sequenziell vs. Gemischt

Sequenzielles Lernen verschiedener Spiele wurde ebenfalls getestet,
da gemischtes Trainieren nicht immer praktikabel ist.

Training method		TTT	TTT_D	TTT_L	C4
Mixed	Next	100.00%	100.00%	100.00%	98.00%
	Legal	100.00%	82.62%	55.80%	100.00%
Sequential	Next	100.00%	100.00%	92.52%	93.63%
	Legal	100.00%	82.35%	64.45%	100.00%

→ Performance ist ähnlich
→ Früher trainierte Spiele werden nicht vergessen
→ Größere Schwankungen des validation loss



3.4 Abflachung von Regeln

Vermutung der Autoren:

Schlechtere Ergebnisse von $\mathrm{TTT}_{\mathrm{L}}$ könnten an der „unabgeflachten“ Form einiger GDL-Regeln liegen.

- → Bei „unabgeflachten“ Regeln liegt im Gegensatz zu sogenannten „abgeflachten“ Regeln keine direkte Abhängigkeit von grundlegenden Zustandseigenschaften vor, da sie stattdessen andere Regeln referenzieren.
- → Hierdurch können längere Abhängigkeiten unterschiedlicher Fluents und ein komplexerer Graph entstehen
- → Regeln können durch Einsetzen abgeflacht werden (falls nicht rekursiv)

3.4 Abflachung von Regeln

Beispiel Abflachung:

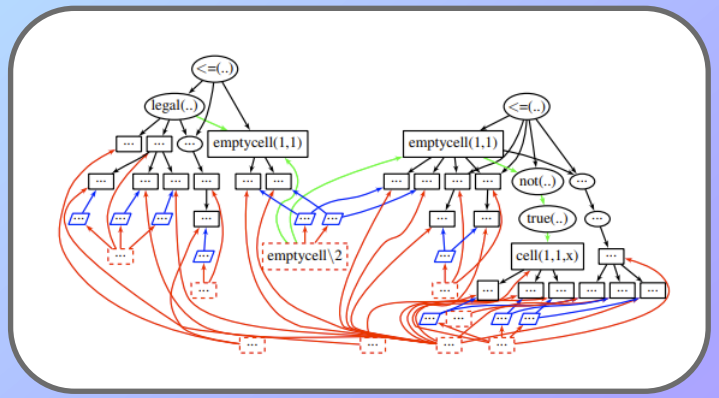
```
(<= (legal xplayer (mark 1 1 x))
    (true (control xplayer))
    (emptycell 1 1))

(<= (emptycell 1 1)
    (index 1)
    (index 1)
    (not (true (cell 1 1 x)))
    (not (true (cell 1 1 o))))
```

Abgeflacht

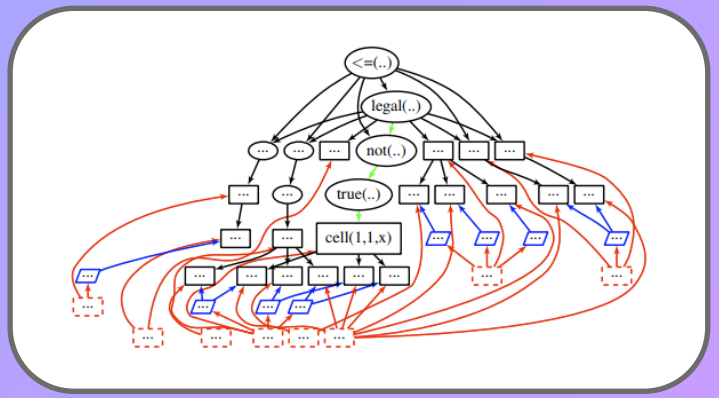
```
(<= (legal xplayer (mark 1 1 x))
    (true (control xplayer))
    (index 1)
    (index 1)
    (not (true (cell 1 1 x)))
    (not (true (cell 1 1 o))))
```

3.4 Abflachung von Regeln

Beispiel Abflachung:



Abgeflacht



→ Durchschnittlich kürzere Abhängigkeitspfade

3.4 Abflachung von Regeln

Resultat:

Game	Next fluents accuracy	Legal actions
TTT-L (standard)	100.00%	63.69%
TTT-L (flat)	100.00%	100.00%
TTT-D (standard)	100.00%	91.96%
TTT-D (flat)	100.00%	100.00%
C4 (standard)	93.58%	100.00%
C4 (flat)	100.00%	100.00%

Table 6: Terminal prediction for 100 games of standard and flattened games

3.4 Abflachung von Regeln

Resultat:
Deutliche Verbesserungen

Game	Next fluents accuracy	Legal actions
TTT-L (standard)	100.00%	63.69%
TTT-L (flat)	100.00%	100.00%
TTT-D (standard)	100.00%	91.96%
TTT-D (flat)	100.00%	100.00%
C4 (standard)	93.58%	100.00%
C4 (flat)	100.00%	100.00%

Table 6: Terminal prediction for 100 games of standard and flattened games

4
Diskussion & Fazit

Einschränkungen, Kritik, Fazit

4.1 – Einschränkungen

- Deutlich schlechtere Ergebnisse bei Spielen mit unabgeflachter GDL-Beschreibung
- Mögliche Schwierigkeiten bei komplexeren Spielen mit längeren Abhängigkeitspfaden
- Verbesserungsbedarf bei der Instanziierung des Regelgraphen und der Beschriftungsfunktion
- Unvollständigkeit des Reasoners in Bezug auf die Erkennung von Endzuständen und Zielwerten der Spieler

4.2 – Kritik

- Begrenzte Vielfalt und Komplexität der Spielauswahl → z. B. Schach, Go oder Shogi

- Geringe Anzahl von Spielen mit hoher Ähnlichkeit
→ Insbesondere bei Tests mit gemischtem/sequenziellem Training

- Unklare Motivation zur Entwicklung eines neuronalen Reasoners im Vergleich zu logikbasierten Reasonern

4.3 – Fazit

- Vorgestellte Methode zur Umwandlung von GDL-Beschreibungen in eine allgemeine Graphendarstellung ermöglicht Deep Learning Ansätze für GGP-fähige Reasoner
- GNN-basierter Reasoner zeigt gute Vorhersagegenauigkeit für verschiedene Trainingsmethoden und Transfertests
- Notwendigkeit weiterer Forschung zur Performanz bei komplexeren und vielfältigeren Spielen
- Potenzial zur Erweiterung und Verbesserung des Reasoners für zusätzliche Reasoner-Aufgaben und Lösung logikbasierter Probleme

Vielen Dank!

Seminar Künstliche Intelligenz
Bei Prof. Dr. Manfred Schmidt-Shauß

Votrag von: Numan Tok | 6927093
s3896299@uni-frankfurt.de

Frankfurt, 13.07.2023

Referenzen

Literatur:

- Gunawan, Alvaro, Ji Ruan, and Xiaowei Huang. "A Graph Neural Network Reasoner for Game Description Language." *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*. Vol. 19. No. 1. 2022.
- Silver, David, et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." *Science* 362.6419 (2018): 1140-1144.
- Bhatt, Anurag, Pratul Varshney, and Kalyanmoy Deb. "In search of no-loss strategies for the game of tic-tac-toe using a customized genetic algorithm." *Proceedings of the 10th annual conference on Genetic and evolutionary computation*. 2008.
- Steinerberger, Stefan. "On the number of positions in chess without promotion." *International Journal of Game Theory* 44.3 (2015): 761-767.
- Zakharov, Victor B., Michael G. Mal'kovskii, and A. I. Mostyaev. "On solving the problem of 7-piece chess endgames." *Programming and Computer Software* 45 (2019): 96-98.

Abbildungen und Tabellen:

Alle verwendeten Abbildungen und Tabellen stammen aus "A Graph Neural Network Reasoner for Game Description Language."

Online Referenzen:

- http://ggp.stanford.edu/gamemaster/homepage/showgames.php

CREDITS: Diese Präsentationsvorlage wurde von Slidesgo erstellt, inklusive Icons von Flaticon und Infografiken & Bilder von Freepik