Jass Queen

Joel L.

1 Zusammenfassung Bot Jass Queen

Der Code für das Projekt ist auf GitHub (Joelius300/dl4g_hslu) zu finden.

1.1 Trumpfstrategie

Die verwendete Trumpfstrategie ist die Graf Heuristik [1], die im Unterricht vorgestellt wurde. Sie erwies sich als die beste von allen evaluierten Ansätzen, darunter sind viele Variationen von Supervised Machine Learning. Mehr dazu steht in Abschnitt 2.1 geschrieben.

1.2 Kartenstrategie

Die Strategie für die Wahl der Karte basiert auf Information Set Monte Carlo Tree Search (ISMCTS) [2] mit Root Parallelization [3].

Da Jass über nicht-perfekte Informationen verfügt, wird bei ISMCTS immer als erstes ein Information Set aus der Verteilung aller möglichen Information Sets genommen. Auf diesen unsicheren, aber perfekten Informationen können anschliessend die vier Schritte von Monte Carlo Tree Search, nämlich selection, expansion, rollout und backpropagation, durchgeführt werden. Knoten im Suchbaum, die nicht mit dem aktuellen Information Set kompatibel sind, werden ignoriert. Am Ende wird der vielversprechendste Kindknoten des Wurzelknotens ausgewählt und gespielt.

Dieser bereits sehr starke Algorithmus kann auf verschiedene Arten parallelisiert werden [4]. In diesem Projekt wird Root Parallelization umgesetzt, wobei mehrere Workers je einen Search Tree aufbauen und ISMCTS darauf betreiben. Am Ende werden alle Bäume zusammengeführt und die vielversprechendste Option über alle Bäume gesehen genommen. Solange die verwendeten Policies und Rollouts nicht vollständig deterministisch sind, erlaubt dies eine breitere Abdeckung des Suchraums, was zu einer genaueren Abschätzung der erwarteten Payoffs führt.

2 Experimente

2.1 Supervised Machine Learning für Trumpfstrategie

Um eine stärkere Trumpfstrategie als die Graf-Heuristik zu bauen, werden verschiedene Supervised Machine Learning Ansätze ausgearbeitet.

2.1.1 Evaluation

Ein wichtiger Punkt ist die Auswertung dieser Strategien. Es ist klar, dass nicht die Metriken auf den Daten verwendet werden dürfen, da diese nur aussagen, wie gut das Modell die Daten imitiert. Wie aber bereits am Fall von AlphaZero [5] gezeigt, kann es gut sein, dass eine neuartige Taktik, die tiefe Korrelation mit menschlichem Spiel aufweist, besser ist als diejenigen, die lediglich Menschen imitieren. Aus diesem Grund wird jede Trumpfstrategie wie folgt ausgewertet: Es werden 128 Jass Partien auf jeweils 1000 Punkte gegen die Graf Heuristik gespielt und die Gewinnrate berechnet. Die Kartenstrategie ist für alle Spieler ISMCTS. So kann aufgezeigt werden, ob diese Trumpfstrategie im Spiel tatsächlich die Überhand hat, was der Fall ist, wenn sie eine Gewinnrate von über 50% erzielt.

2.1.2 Methode

Die Experimente werden mit DVCLive [6] getrackt. Dafür wird eine DVC Pipeline [7] erstellt, welche sowohl die Vorbereitung der Daten, die verschiedenen Trainingsschritte und die Evaluation beinhaltet.

Während dem Training werden zudem Metriken wie der Loss und Klassifikationsmetriken wie Accuracy und F1 geloggt.

Die Daten werden vor der Verwendung mit undersampling balanciert, da Schieben gerade in den Swisslos Daten klar am häufigsten vorkommt. Normalisierung ist nicht nötig, da mit One-Hot Encoding gearbeitet wird und jedes Feature Binär (1 oder 0) ist.

2.1.3 Ansätze

Der einfachste Ansatz ist die alleinige Verwendung der Swisslos Daten. Dies erzielte eine Gewinnrate von 40% gegen Graf. Ein spannenderer Ansatz ist ein Pre-Training Ansatz, wo zuerst die Graf Heuristik gelernt wird (mit einem generierten Datensatz) und anschliessend nur mit den besten Spielern der Swisslos Daten ein Fine-Tuning gemacht wird.

- Training auf gesamten balancierten Swisslos Daten: 40% Gewinnrate gegen Graf
- Pre-Training auf Graf, Fine-Tuning auf Swisslos Daten (top 20%): 43% Gewinnrate gegen Graf

2.1.4 Modell

Das Modell nimmt die One-Hot kodierten Handkarten (36x1 binär) plus ob geschoben werden kann oder nicht (1x1 binär). Diese 37 Features werden in mehrere Linear Layers mit ReLU non-linearity gegeben. Bei jedem Layer werden die originalen Inputs hinzugefügt, um Skip-Connections zu ermöglichen. Am Ende ist ein Linear Layer, welcher die Logits für die 7 Trumpf Optionen (4 Farben + Uneufe + Obe-abe + Schieben) ausgibt. Der verwendete Loss ist Cross-Entropy. Zusätzlich wird Dropout Regularisierung verwendet.

2.1.5 Tuning

Folgende Hyperparameter wurden ausgiebig getuned:

- Anzahl Layers
- Dimension der hidden Layers
- Learning rate
- Dropout rate
- Art/Implementation der Residual Connections
- Data
  - Nur Swisslos
  - Graf + Swisslos
  - Graf + Top 30% Swisslos
  - Graf + Top 20% Swisslos
  - Graf + Top 10% Swisslos

2.2 Parameter Tuning für Monte Carlo Tree Search

Zwei Payoff Funktionen werden verglichen. Binärer Sieger (1 für Gewinner, 0 für Verlierer) schlägt punktebasierte Payoffs (tatsächliche Punkte auf [0,1] skaliert) nicht nur in Anzahl gewonnen Spielen sondern auch in Partien auf 1000 Punkte.

Der C Parameter für UCB1 wird ebenfalls getuned. Am Ende wurde 8 als beste Option gewählt, was erstaunlich war, da bisher $\sqrt{2}$ sehr gut zu performen schien, aber es zeigt sich, dass $1 < \sqrt{2} < 5 < 10 < 7 < 8$ gilt. Diese Tests werden im finalen Setting, das heisst auf dem Deployment Server mit Root Parallelized ISMCTS mit 15 Workers und 9.75 Sekunden Zeitbudget getestet.

2.3 Kurzer Abstecher in Reinforcement Learning

Mit den Unterlagen zu Reinforcement Learning gelang es leider nicht einen Bot mit RL zu trainieren. Es wurde zwar versucht ein Gymnasium Environment [8] Wrapper aufbauend auf dem Jass-Kit GameSim zu erstellen, jedoch ohne Erfolg.

3 Nennenswertes

Zusätzlich zum bisher genannten habe ich einige Dinge gemacht, die eventuell auch spannend sein könnten. Diese können im Code angeschaut werden.

- Ausführliche ML Pipeline mit DVC
- Model Training mit PyTorch Lightning und DVCLive
- Generierung und Balancierung des Graf Datasets
- Verbesserungen/Anpassungen der jass-kit Bibliothek in Open-Source Fork (Joelius300/jass-kit-py)
- Evaluation von Agents auf mehreren Kernen
- Deployment mit Docker und Gunicorn

Bibliographie

[1] D. Graf, „Jassen auf Basis der Spieltheorie“. Zugegriffen: 22. Dezember 2023. [Online]. Verfügbar unter: https://dwolleb.ch/jass/

[2] P. I. Cowling, E. J. Powley, und D. Whitehouse, „Information Set Monte Carlo Tree Search“, *IEEE Transactions on Computational Intelligence and AI in Games*, Bd. 4, Nr. 2, S. 120–143, Juni 2012, doi: 10.1109/TCIAIG.2012.2200894.

[3] T. Cazenave und N. Jouandeau, „On the Parallelization of UCT“.

[4] G. M. J. B. Chaslot, M. H. M. Winands, und H. J. Van Den Herik, „Parallel Monte-Carlo Tree Search“, *Computers and Games*, Bd. 5131. Springer Berlin Heidelberg, Berlin, Heidelberg, S. 60–71, 2008. doi: 10.1007/978-3-540-87608-3_6.

[5] D. Silver *u. a.*, „Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm“. Zugegriffen: 22. Dezember 2023. [Online]. Verfügbar unter: http://arxiv.org/abs/1712.01815

[6] „DVCLive“. Zugegriffen: 22. Dezember 2023. [Online]. Verfügbar unter: https://dvc.org/doc/dvclive

[7] „DVC Pipelines“. Zugegriffen: 22. Dezember 2023. [Online]. Verfügbar unter: https://dvc.org/doc/user-guide/pipelines

[8] „Gymnasium Documentation“. Zugegriffen: 22. Dezember 2023. [Online]. Verfügbar unter: https://gymnasium.farama.org/index.html