LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR MOBILE UND VERTEILTE SYSTEME

BACHELORARBEIT

Kooperative Neuroevolution in RoboCup2D

Alexander Isenko

Entwurf vom 25. Oktober 2016

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR MOBILE UND VERTEILTE SYSTEME

BACHELORARBEIT

Kooperative Neuroevolution in RoboCup2D

Alexander Isenko

Aufgabensteller:	Prof. Dr. Claudia Linnhoff-Popien
Betreuer:	Thomas Gabor, M.Sc Lenz Belzner, Dipl.-Medieninf.
Abgabetermin:	11. November 2016

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

München, den 11. November 2016

. . . . . .
(Unterschrift des Kandidaten)

Abstract

Wir untersuchen in dieser Bachelorarbeit verschiedene Ansätze zur Entwicklung von neuronalen Netzen am Beispiel der Cross Entropy Method, genetische Algorithmen und CoSyNE unter Einschränkung von spärlichen Fitnesssignalen, hochdimensionalen kontinuierlichen Zustandsräumen und simulationsbasierter Optimierung.
Der Suchraum wird durch diskrete Cosinustransformationen (DCT) unter der Annahme reduziert, dass benachbarte Gewichte zueinander korelliert stehen. Die Domäne ist ein Fußballsimulator, Half Field Offense (HFO), der Teams aus dem weltweiten Wettbewerb RoboCup zum Vergleichen bereitstellt. Die Implementierung erfolgte in Haskell und Python.

Inhaltsverzeichnis

1. Einführung

- Finden von Lösungen ohne große Anpassung von Hyperparamteren
- Domänen mit starken Einschränkungen (sparse Fitness, hochdimensionale kontinuierlicher Zustandsraum)
- Maschinelles Lernen ist eine potenziell große Hilfe zur Entwicklung von besseren Systemen

Die Nützlichkeit von Automatisierung ist offensichtlich Viele benutzen Neuronale Netze mit Backpropagation, andere Möglichkeiten wie man Netze trainiert fallen ausm Fokus Ich spreche ein paar Möglichkeiten an und untersuche sie auf ihre Nützlichkeit in Fälle wo Backprop nicht möglich ist

1.1. Aufgabenstellung

- Verschiedene Techniken auf multi agenten systeme anzuwenden
- Kooperation und homogenität von Gewichten im Netz untersuchen
- Schauen ob ichs hinkriege

In dieser Arbeit beschäftige ich mich mit der Implementierung von verschiedenen Machine Learning Algorithmen für eine Domäne Half Field Offense mit sparsen Fitnesssignalen und einem hochdimensionalen kontinuierlichen Zustandsraum und stelle diese gegenüber. Aus dem Vergleich kann man die Nützlichkeit für andere Domänen mit ähnlichen Einschränkungen erahnen.

1.2. Motivation

- Praktikum - DARTS
- Gespräche mit Kommilitonen
- Abstrakte Zusammenhänge nutzen um Abhängigkeiten auf der Domänenseite zu abstrahieren

Im Siemens Praktikum bin ich das erste Mal mit Genetischen Algorithmen in Kontakt gekommen Es kam die Frage auf welche Domänen ein Schwierigkeiten für Machine Learning Tasks darstellen und in der Industrie anwendbar sind Verteilte Systeme, Roboter die zusammen etwas lernen, etc Fitness Engineering, also die Anpassung von dem Rewardsignal, ist oft nötig für Aufgaben wo der Reward von viele Zeitschritten und kontinuierlichen Aktionsketten abhängt Deshalb kam die Idee dies auf die Probe zu stellen, mit einem Fußballsimulator und neuen Algorithmen, die einen Durchbruch erreicht haben (CoSyNE)

1.3. Aufbau der Arbeit

- Erklärung der Grundlagen, GA, NN, DCT, CoSyNE, CE
- Aufbaut der Domäne, Implementierung der Algorithmen
- Präsentation der Resultate
- Ausblick
- Die Implementierung und Problematiken sind im Appendix
- Veränderungen während der Arbeit

Im Rahmen dieser Arbeit werden im Kapitel 2 die Grundlagen von Genetischen Algorithen und deren Verknüpfung zu neuronalen Netzen und der Cross Entropy Method erklärt und anschaulich dargestellt. Kapitel 3 beschäftigt sich mit der Domäne, Parametrisierung der Algorithmen und der jeweiligen Resultate. Das Kapitel 4 gibt einen Ausblick in weitere Verbesserungsmöglichkeiten und legt verwandte Felder dar. Im Appendix wird die Implementierung von dem gesamten System überschlagen, Problematiken angesprochen und Lehren die für eine zukünftige Arbeit gezogen werden können angesprochen.

2. Definitionen

Dieses Kapitel bietet Einblick in die Grundlagen von **Genetischen Algorithmen** im Zusammenhang mit **neuronalen Netzen** und der **Cross Entropy Method**. Außerdem werden einige Verbesserungen zu den naiven Methoden besprochen, wie die Reduzierung des Suchraums durch **Fouriertransformationen** und die Einführung von einer **kooperativen Evolution** durch Hinzufügen von einer neuen Methode zum GA.

2.1. Genetische Algorithmen

Ein genetischer Algorithmus ist ein Optimierungsverfahren, der meiner Meinung nach an einem natürlichen Vorgang am Besten zu erklären ist. Wir stellen uns beispielsweise zehn Antilopen und ein Gepard vor

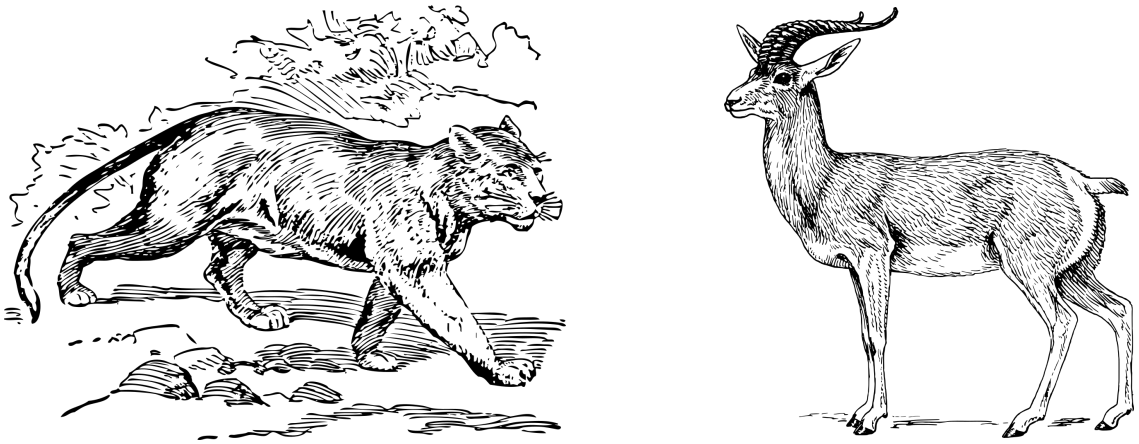

Abbildung 2.1.: Illustration von einem Geparden und einer Antilope

2.1.1. Selektion

Der Selektionsschritt kommt zustande, wenn wir eine bereits evaluierte Population haben und spaltet sie in zwei Teile. Es wird oft als fester Prozentsatz α von der Gesamtpopulation kodiert (citation needed).

Der Selektionsschritt teilt die Population and der Stelle $\alpha * \textit{Länge der Population}$ zwei Teile auf.

2.1.2. Kreuzung

“Es gibt viele Kreuzungsoperationen, wir behandeln nur welche am Beispiel von einer Liste”
“one-point, two-point und n-point crossover”

2.1.3. Mutation

“Mutation ist hilfreich um die Varianz der Population etwas zu erhöhen”

2.1.4. Repopulation

“Ist ein Schritt der oft implizit beim Crossover passiert, (zitat finden), wird nicht nur benutzt um Kinder und Eltern zu verknüpfen, sondern auch völlig neue Individuen hinzuzufügen”

2.2. Neuroevolution

“Neuroevolution beschäftigt sich mit der Verknüpfung von Genetischen Algorithmen und Neuronalen Netzn”

2.2.1. Neuronale Netze

“Neuronale Netze sind eine riesige Gleichung”
“Formel von Machine Learning zeigen”
“Backpropagation wird angesprochen aber nicht ausführlich erklärt”

Dense Ebene

“Kleines Beispielnetz aufmalen und durchrechnen”

LSTM Ebene

“Kleines Beispielnetz aufmalen und durchrechnen”

Softmax Ebene

“Erklärung bieten wieso es sowas gibt und was für eine Formel angewendet wird”

2.2.2. Verbindung mit genetischen Algorithmen

“Die Verknüpfung findet in der Kodierung von den Individuen statt - wir nehmen eine (naive) Darstellung von allen Gewichten” “Problematik -> Folgerung zu DCT”

2.3. Diskrete Cosinus Transformation - DCT

“Der Suchraum kann durch die sog. DCT auf beliebige Dimensionalität eingeschränkt werden, wenn bestimmte Annahmen getroffen werden können”
“Fouriertransformationen machen folgendes...”
“Es gibt eine Inverse die aus einem n-stelligen liste eine m-stellige macht, wo die Zahlen korelliert sind (Beispiele)”

2.3.1. Kodierung des Suchraums

“Die Kodierung besteht aus eine 20-stellingen Liste wie im Paper (Cosyne), wo damit 2k Gewichte entwickelt wurden”

2.4. Cooperative Synapsen Neuroevolution - CoSyNE

“CoSyNE wurde vom Prof.Dr.Schmidhuber an der ETH Zürich entwickelt und hat damit sehr viele anderen Algorithmen in den Schatten gestellt”
“Methodik, ist wie GA bloß mit einer Aktion mehr die statt spielbare ‘Policies’, nur im Koeffizientraum entwickle”
“Dies erlaubt eine (zitat) Kooperative Entwicklung von Koeffizienten für die nachfolgenden Inv.DCT”
“Kein Crossover, geringe Mutation”
“Beispiele vom Erfolg”

2.4.1. Permutation

“Wir transponieren, shuffeln und transponieren zurück”

2.5. Cross Entropy

“Cross Entropy ist eine andere Möglichkeit um die Individuen darzustellen, anstatt von Zahlen, hab ich nun pro Coeffizient eine Normalverteilung habe mit Mean und STD” (cite boer07)

2.5.1. Normalverteilung

“Was ist eine Normalverteilung”
“Wie programmiere ich eine Normalverteilung selber, Box-Muller, Randomness, Haskell-code Beispiel”

3. Umsetzung in RoboCup2D

RoboCup ist eine Fußball Simulator der seine Anfänge in 1993 in Japan, Tokyo gefunden hat. Eine Gruppe von Forschern, inklusive Minoru Asada, Yasuo Kuniyoshi und Hiroaki Kitano, haben als einen Wettbewerb unter dem Namen **Robot J-League** gestartet. Der Name stammt von einer professionellen japanischen Fußball Liga.

Nach einem Monat haben sie jedoch weltweit überwältigendes Feedback bekommen und haben die Initiative als internationales Projekt weitergeführt, daher kam die Umbenennung zur **Robot World Cup Initiative**, kurz RoboCup.

Die RoboCup Initiative hat betreibt derzeit sechs große Wettbewerbe, die sich jeweils wieder in Ligen und Subligen aufteilen lassen. Darunter fällt **RoboCup Soccer**, **RoboCup Rescue Rescue**, **RoboCup Junior**, **RoboCup Logistics**, **RoboCup @ Work** und **RoboCup @ Home**. Unsere Implementierung fällt in die Subliga **2D Soccer Simulation**, in der es darum geht in einer zweidimensionalen Welt zwei Fußballmannschaften gegeneinander antreten zu lassen.

Die Aufgabe die wir behandeln gehört zu einem Fragement von RoboCup2D, genannt **Half Field Offense**.
TODO: Bild vom gesamten Spielfeld

3.1. Half Field Offense

Die Domäne Half Field Offense grenzt das Spielfeld auf genau die Hälfte ein, sodass wir 4 Angreifer und 3 Verteidiger + Torwart haben. Diese Einschränkung vereinfacht den Zustandsraum immens und erlaubt potenziell eine Wiederverwendbarkeit der Agenten, wenn eine vollständige Mannschaft aufgebaut werden muss.

In unserer Implementierung haben wir lediglich ein 1v1 Szenario, also ein Angreifer gegen ein Torwart. Diese sieht jedoch explizit ein nahtlosen Skalierung auf ein 4vs4 Szenario vor, sodass weitere Parametrisierung ohne viel Aufwand ausprobiert werden können.

TODO: BILD vom halben Spielfeld mit 1v1 Szenario

3.1.1. Zustandsraum

Im Zustandsraum der HFO Domäne kann man zwischen dem **High Level State** und dem **Low Level State** unterscheiden. Wir haben den High Level Raum benutzt. Der High Level Zustandsraum wird durch folgende Formel aufgespannt.

Sei T die Anzahl der Teammitglieder, O die Anzahl der Gegner:

$$10 + 6T + 3O$$

TODO: Alles vom Paper abschreiben oder referenzieren?

In unserem 1v1 Setting haben wir damit 19 Zustandsparameter. Aus Angstgründen dass das Netz zu groß wird, wurden aber lediglich die *wichtigsten* 9 genommen wo ich mir gedacht habe was das mindeste an Information ist, was ein Spieler braucht um ein Tor zu schießen. Wir haben ihm folgendes gegeben:

- x Koordinaten (kontinuierlich -1 bis +1)
- y Koordinaten (kontinuierlich -1 bis +1)
- Sichtrichtung (kontinuierlich -1 bis +1)
- Nähe zum Ball (kontinulierlich 0 bis 180)
- Winkel zum Ball (kontinulierlich 0 bis 180)
- Kann eine Ballaktion (Schießen) ausgeführt werden (bool -1 oder +1)
- Wie nah bin im am Tor dran (kontinulierlich -1 bis +1)
- Was ist der Winkel zum Mittelpunkt des Tors (kontinulierlich 0 bis 180)
- Was ist der größte offene Winkel zwischen Torwart und Torpfosten (TODO Bild) (kontinulierlich 0 bis 180)

Winkel wurden so kodiert Koordinaten wurden so kodiert Bool'sche Werten wurden so kodiert Längen Werten wurden so kodiert

Der Low Level Zustandsraum ist hat folgenden Formel (sollte das überhaupt rein?)

3.1.2. Aktionsraum

Es gibt 6 nicht parametrisierte Aktionen und 8 parametrisierte. Wir haben die nicht parametrisierten benutzt, ohne Catch, weil Angreifer den Ball nicht fangen dürfen

Parametrisierte

- Dash(power, degrees)
- Turn(degrees)
- Tackle(degrees)
- Kick(power, degrees)
- Kick_To(x-coords, y-coords, speed)
- Move_To(x-coords, y-coords)
- Dribble_To(x-coords, y-coords)

Nicht parametrisierte

- Move
- Shoot
- Dribble
- Intercept
- Catch
- No-Op

3.1.3. Einschränkungen

Diese Domäne hat viele Einschränkungen wenn man sie mit XYZ vergleicht. Zum einen erlaubt sie uns nicht in die Zukunft zu propagieren und zu schauen wie gut eine Entscheidung ist. Wir haben ausschließlich den einen Simulator zur Hand der erst nachdem ein Spiel fertig ist uns ein Fitnesssignal sendet und wir daraufhin versuchen können davon abzuleiten ob die zahlreichen Aktionen die wir ausgeführt haben uns zum Erfolg geführt

haben. Auch genannt Sparse Fitness (Probleme in anderen Domänen suchen)

Wir haben ein sog. simulation based learning (Quellen suchen)

Leider haben wir einen sehr kontinuierlichen Zustandsraum, der von uns verlangt eine Abstraktionsebene zu benutzen...NNs

Diese Art von Machine Learning von nennt man auch Black Box Reinforcement Learning

3.2. Implementierung der Algorithmen

Der ausführliche Aufbau der Algorithmen wird näher im Appendix erklärt, hier gehen wir lediglich auf die Parametrisierung und wer für was zuständig war ein.

Der Simulationsserver ist in C++ geschrieben und wurde 1-zu-1 aus (src: https://github.com/LARG/HFO) genommen. Er wird durch Flags beim Starten parametrisiert.

Die Agenten sind in Python geschrieben und sind eine einfache Erweiterung von einem der Beispielskripte aus (same src). Diese werden nach meiner Implementierung auch mit Kommandozeilen Flags gestartet.

Der Koordinator der für die Umsetzung des GAs zuständig ist und den Server und die Agenten Skripts startet und überwacht ist in Haskell geschrieben.

Für alle folgenden Simulationen galten die selben Rahmenbedingungen:

Generationen	300
Populationsgröße	50
Teamepisoden	25
Episodenzeit	$500s$
Ball nicht berührt	$50s$

Das heißt pro Team haben wir 25 Spiele gespielt, pro Generation 1250 und die gesamte Simulation bestand aus 375000 Spielen. Die Episodenzeit wurde auf 500 Echtzeitsekunden beschränkt, da ansonsten die simulierte Zeit pro Spiel nicht praktikabel war.

3.2.1. Wahrscheinlichkeitsverteilung von Aktionen

“Einfache Kodierung von Aktionen durch eine Wahrscheinlichkeitsverteilung, der Agent sampelt raus”
“Parametrisierung”

Der erste Algorithmus hat als Kodierung der Individuen eine diskrete Wahrscheinlichkeitsverteilung über die 5 Aktionen.
Wenn diese and den Agenten weitergegeben wurde, hat er aus dieser Verteilung gesampelt, ohne Wissen über jeglichen den Zustand.

Die Kreuzung wurde auf zwei verschieden Arten umgesetzt, wobei sich keine als signifikant besser herausgestellt hatte.
Die erste Methode ...

Die zweite Methode hat beide Verteilungen genommen, die Wahrscheinlichkeiten für jeweiligen Aktionen addiert und normalisiert.

Seien $\mathcal{A}, \mathcal{B}$ die diskreten Wahrscheinlichkeitsverteilungen die verknüpft werden sollen:

$$l := |\mathcal{A}| : \mathcal{C} := \{\frac{(a_i + b_i)}{l} \mid a_i \in \mathcal{A}, b_i \in \mathcal{B}\}$$

Die Mutation wurde auf folgende Weise durchgeführt. δ wird in die Anzahl der Aktionen aufgeteilt, mit zufälligem Vorzeichen.

```
> splitDelta 20 5
[-4, +4, +4, -4, -4]
```

```
> splitDelta 100 4
[-25, +25, +25, -25]
```

Es wurde folgende Parametrisierung benutzt:

Selektion	25%
Mutationswahrscheinlichkeit	50%
Mutationsfaktor	20

3.2.2. Cross Entropy mit DCT

“Parametrisierung”

3.2.3. Neuroevolution mit DCT

“Parametrisierung”

3.2.4. CoSyNE mit DCT

“Parametrisierung”

3.3. Resultate

Im folgenden Teil finden sich alle Resultate. Durchschnittlich hat eine Trainigsphase mit 300 Generationen, Population 50 und 25 Episoden pro Team 30 Stunden gedauert. Alles wurde auf einem Laptop mit einem Intel Dualcore mit 2.9GHz und 4GB Arbeitsspeicher ausgeführt.

3.3.1. 1v1

“Domäne nochmal erklären, Zustandsraum spezifizieren, sodass alles replizierbar ist”

Wahrscheinlichkeitsverteilung

“Graph zeigen, Interpretation bzw. Erklärung” “Nach 25 Episoden stagniert”

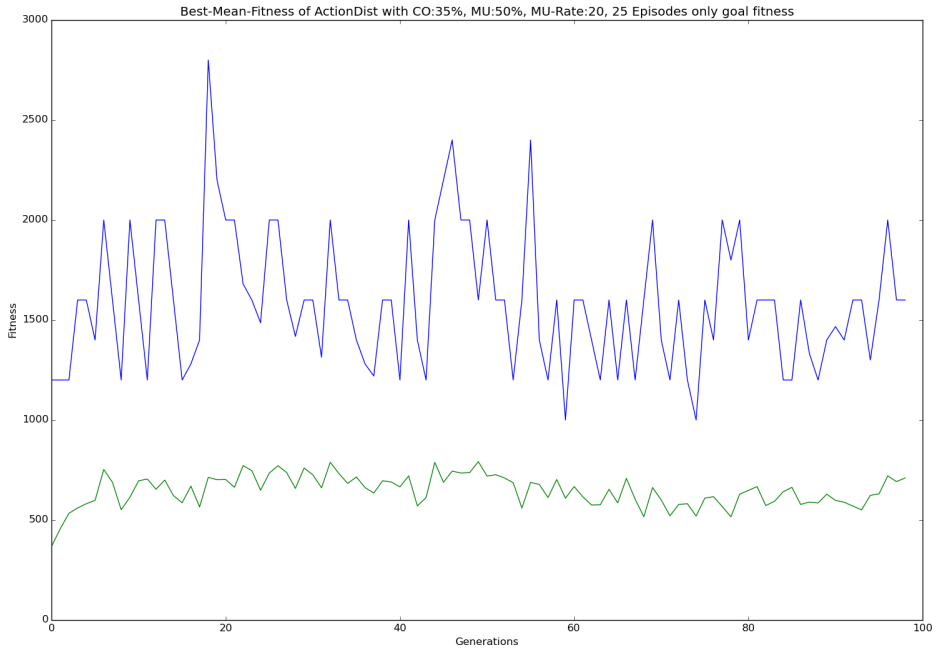

Abbildung 3.1.: Fitness Graph zur Wahrscheinlichkeitsverteilung

Cross Entropy

“Graph zeigen, Interpretation bzw. Erklärung” “Aggressivität” “Nach 50 Episoden stagniert”

Neuroevolution

“Graph zeigen, Interpretation bzw. Erklärung”

CoSyNE

“Graph zeigen, Interpretation bzw. Erklärung”

3.3.2. Vergleich

“Sicherheit <-> Aggressivität steht im Kontra zur Stabilität der Algorithmen”
“Wenn Zeit ein Faktor wäre”
“Wenn Sicherheit ein Faktor wäre”

4. Diskussion

“Im Folgenden bespreche ich die Nützlichkeit der Ergebnisse”

4.1. Anwendungsmöglichkeiten

“Wenn starke Einschränkungen bestehen sind, waren diese Algorithmen zielführend”

4.2. Coevolutionärer Aspekt

“Die Annahmen für Netze zu treffen ist interessant gewesen, hat anscheinend auch für 30k Gewichte geklappt, sollte man später zum checken”

4.3. Ausblick

“Im Folgenden gebe ich meinen Senf zu den Parametrisierungen an und wenn die Welt perfekt wäre, was ich dann machen würde”

4.3.1. Genetische Algorithmen

“Hab nicht an den Parameteren gespielt”

4.3.2. Aufbau des neuronalen Netzes

“Hab nicht verschiedene Netze ausprobiert”

4.3.3. Cross Entropy

“Hab die naivste Implementierung genommen, könnte viel viel besser werden, wenn man Grips reinsteckt”

4.3.4. Aktionsraum

“Hätte bessere Aktionen definieren können, die schlauere Spielzüge erlauben, da ich die Vermutung habe dass die jetzigen ein Hardcap an Qualität bieten”

4.3.5. Multi-Agenten Systeme

“Hatte noch keine Möglichkeit in diese Domäne ausführlich zu testen”

4.4. Verwandte Felder

“Folgende Reads sind nice”

4.4.1. Implementierung für OpenAI Gym

“Update für OpenAI Gym, meine Impementierung wird folgen”
“Keine Multi-Agenten Setting möglich, dsw ist meine Arbeit nicht umsonst gewesen”

4.4.2. Bestärkendes Lernen - Black Box RL

“Wenn man die Netze als Policy betrachtet ist meine Arbeit eine Anwendung im Reinforcement Learning Bereich gewesen”

4.4.3. Convolutional neuronale Netze und CoSyNE

“Warum ist die Annahme für homogene Netzstrukturen besser für Convolutional Networks”
“Evtl ist CoSyNE dort richtig geil”

Roboterarm

“Bilder, Graphen und Parametrisierung zeigen + Link + Zitat”

Rennspiel

“Bilder, Graphen und Parametrisierung zeigen + Link + Zitat”

A. Appendix

“Da der Fokus sehr stark auf dem Machine Learning Bereich lag, aber die Domäne viele Tücken hatte und die Implementierung einen Großteil der Zeit in Anspruch genommen hat, wird es hier behandelt”

A.1. Architektur

“Die Architektur des gesamten Projektes wurde in Haskell geplant, da ich mit externen und unsicheren Implementierungen arbeite”
“Automatische Docs, weil ich gute Kommentare schreibe” “Hat geholfen Fehler schneller zu finden und Codereuse ist super geil gewesen”
“Bild von der Kommunikation zeichnen”

A.1.1. Haskell Server

“Module”
“Typklassen sind geil”
“Globale Config ist nice”
“Automatische Serialisierung mit Aeson”

A.1.2. Python Agent

“Module”
“CMD Parser”
“Keras”

A.1.3. Kommunikation

“Haskell <-> Python: JSON ist von Python nativ als Dict unterstützt” “Python <-> Server: FFI Python to C++ (HFO) + (Zitat)”

A.1.4. Parallelisierungsmöglichkeiten

“Bottleneck ist die Kommunikation über JSON-Files”

A.2. Statistik

“Für Cross Entropy hab ich Varianz und STD mit Seed gebraucht, für MonadRandom gabs keine Implementierung, dsw hab ich eine eigne gemacht”

A.2.1. Lineares Laufzeit und Speicherkomplexität für Evaluation

“Folds sind supernice, minimale Erklärung, Links zu Gabriels Blog, Vorzeigen von Effizienz”

A.2.2. Stabile Varianzfunktion

“Catastrophic Cancellation, erste Lösung, zweite Lösung von Gabriel mit E-Mail Austausch”

A.3. Problematiken

“Server ist scheiße, nicht besonders gut dokumentiert”

A.3.1. HFO Server

“Erfolgreicher Durchlauf ist abhängig vom Computer, bzw. von der Leistung”
“Undefinierte Lags zwischen Step 2k-8k”
“Bedienung vom Visualizer ist nicht richtig erklärt”
“Visualizer produziert nicht benutzbare logs, nur das erste Spiel ist ‘abspielbar’, rest ist korrupt ”
“Einloggen von Spielern nimmt sehr viel Zeit in Kauf, dafür muss ein Austausch von Policies on-the-fly passieren”
“Visualzer bricht bei 24k Steps ab, aber ohne ihn funktioniert die Simulation nicht, laggt nur rum”
“Es gibt keine Zeitangaben wie lange ein Agent nicht aktiv sein darf, wenn er keine Entscheidung abgibt wird er ignoriert, dieser Fall sollte behandelt werden für kompetative Benutzung”

A.3.2. HFO Python Library

“Kodierung von Zuständen in denen sich der Agent befindet gibt ein Hexdump zurück, man muss per Hand die ENUMS herausfinden und hardcoden”