Hacking Reinforcement Learning

Guillem Duran Ballester

Guillemmdb

@Miau_DB

europython
Edinburgh 23-29 July 2018
A tale about hacking AI-Corp
AI-Corp

"The Env" API

"Algorithms" API

Research

Money$$

Data

WOW SCIENCE
MUCH DATA
SUCH PAPERS
VERY HYPE
MANY LIKES

Data

Money$$
1. Information gathering
2. Scanning
3. Exploitation & privilege escalation
4. Maintaining access & covering tracks
What is RL?
Our Hobby:
Developing FractalAI

"Study hard what interests you the most in the most undisciplined, irreverent and original manner possible.”

R. P. Feynman

Sergio Hernández
@EntropyFarmer

Guillem Duran
@Miau_DB
Causal entropic forces

- Paper by Alex. Wissner-Gross (2013)
- Intelligence is a thermodynamic process
- No neural networks → Equations
Intelligent decision

Number of future possible outcomes

Direction of maximum

Given your current state

\[ F(X_0, \tau) = T_c \nabla_X S_c(X, \tau) \bigg|_{X_0} \]
Until you reach the time horizon

Map them to a score

\[ S_c(X, \tau) = -k_B \int_{x(t)} \Pr(x(t)|x(0)) \ln \Pr(x(t)|x(0)) \, \mathcal{D}x(t) \]
Cone: Space of future possible outcomes

Sample random walks

Move away from the wall so fewer walks get 0 score

Present

Zero score
Nobody likes entropic forces

- All rewards equal 1
- NP hard!
FractalAI

- Finds low probability points and paths
- Constrained resources
- Total control of exploration process
- Linear time
A set of rules for:

1. Defining a cloud of points (Swarm)
2. Moving a Swarm in any Cone
3. Measuring and comparing Swarms
4. Analyzing the history of a Swarm
Hacking RL

1. Information gathering
2. Finding vulnerabilities & Scanning
3. Exploitation & privilege escalation
4. Covering tracks & Maintaining access
Finding an attack vector

reward, end, info.update({"action": action})

FractalAI
- Scalar diff 2 states
- Scalar reward 1 state
- A prior action prob
- openai.gym

0 Days: FMC & SW

TensorFlow

env.get_state()

OpenAI

action

observation
Swarms are cool

- They move in linear time.
- Pixels/RAM + Reward.
- They guess density distributions
- They follow useful paths
Cunningham's Law

"The best way to get the right answer on the Internet is not to ask a question; it's to post the wrong answer."

Figure 5.1.: Theorem 5.3.3 rules out simple but powerful artificial intelligence algorithms, as indicated by the greyed out region in the upper left. Theorem 5.6.1 upper bounds how powerful an algorithm can be before it can no longer be proven to be a powerful algorithm. This is indicated by the horizontal line separating the region of provable algorithms from the region of Gödel incompleteness.
Using a Swarm to generate data

- **Swarm Wave (SW)**
  - Move a **Swarm** → Sample state space
  - **Cone** → Tree of visited states
  - Efficient → Only one tree
Using a Swarm to generate data

- **Swarm Wave (SW)**
  - Move a **Swarm** → Sample state space
  - **Cone** → Tree of visited states
  - Efficient → Only one tree

- **Fractal Monte Carlo (FMC)**
  - 1 Cone per action
  - Robust → Stochastic/difficult envs
  - Distribution of action utility
Hardcore Lunar Lander

2 Continuous DoF

FIRE

02+ 01+ 02- 01-

HP

Fuel

Rubber band

Hook
The Gameplay

Reward
- Health + Fuel level
- Closer to target → +0.2
- Reach target → +100

Catch rock outside this circle

Bring rock here

Don’t crash!
- Grey lines: Rocket Paths
- Colored lines: Hook’s Path
- Colored change: New target (Pick up/drop rock)
Hacking RL

1. Information gathering
2. Scanning
3. Exploitation & privilege escalation
4. Maintaining access & covering tracks
Demo time!
Hacking RL

1. Information gathering
2. Scanning
3. Exploitation & privilege escalation
4. Maintaining access & managing tracks
Performance of the Swarm Wave
Robust to sparse rewards
Solving Atari games is easy
SW is useful in virtually all environments
Fractal Monte Carlo
Control swarms of agents
Multi objective environments
Hacking OpenAI Baselines

Run_atary.py → inject hacked env.

```python
from baselines.common.cmd_util import atari_arg_parser,
from fractalai.datasets.baselines import make_atari_env
import ray
ray.init()
```

A2c.py → recover action

```python
obs, rewards, dones, infos = self.env.step(actions)
actions = [inf['action'] for inf in infos]
mb_actions.append(actions)
```
- PyData Mallorca co-organizer
- Telecomm. Engineer
- My hobby: hacking AI stuff
- RL Researcher Wannabe

Let’s coauthor papers or hire me!

- SW & FMC are simple
- I learn stuff super fast
- I like teaching & sharing
- Save tons of money!

Guillem Duran Ballester
Guillemdb
Thank You!

Please Hack us:

1. Talk repo: [Guillemdb/hacking-rl](https://github.com/Guillemdb/hacking-rl)
2. Code: [FragileTheory/FractalAI](https://github.com/FragileTheory/FractalAI)
3. More than 100 videos
4. PDFs on arXiv.org

@Miau_DB
@Entropyfarmer
Additional Material

- How the algorithm works
- An overview of the FractalAI repository
- Reinforcement Learning as a supervised problem
- Hacking OpenAI baselines
- Papers that need some love
- Improving AlphaZero
- Combining FractalAI with neural networks
The Algorithm

1. Random perturbation of the walkers
2. Calculate the virtual reward of each walker
   a. Distance to 1 random walker
   b. Reward of current state
3. Clone the walkers → Balance the Swarm
Random perturbation

Causal slice $X_H(x_0, t=5^*dt) =$
Set of all feasible system futures.

Initial action was “A”

Initial action was “B”

System’s initial state $x_0$
Walkers & Reward density
Cloning Process

Walkers jump to better positions by cloning other walker’s states.
Cloning balances both densities
Choose the action that most walkers share.

Causal slice $X_H(x_0, t=6*\text{dt})$.

State space $E_S$.

- Action “A” $W_A = 1/6$.
- Action “B” $W_B = 5/6$.

System’s initial state $x_0$. 
RL is training a DNN model

- ML without labels → Environment
- Sample the environment
- Dataset of games → Map states to scores
- Predict good actions
Which Envs are compromised?

- Atari games → Solved 32 Games!
- Sega games → Good performance
- dm_control → x1000+ with tricks
- I hope soon in DoTA 2 & challenging environments
If you run it on your laptop in 50 games

- Pwns planning SoTA
- Cheaper than a human (No Pitfall)
- 17+ games with max scores (1M Bug)
- Beats human record → 56.36% games
RL as a supervised task

- Train autoencoder with a SW
- Generate 1M Games and overfit on them
- Use a GAN to mimic a fractal
- Use FMC to calculate Q vals/Advantages
- Trained model as a prior
Give love to papers!

- Reproducing world models
- Playing Atari from demonstrations (OpenAI)
- Playing Atari from YouTube Videos (Deepmind)
- RUDDER
Efficiency on MsPacman

SW vs. UCT & p-IW (Assuming 2 x M4.16xlarge)

<table>
<thead>
<tr>
<th></th>
<th>UCT 150k</th>
<th>p-IW 150k</th>
<th>p-IW 0.5s</th>
<th>p-IW 32s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>x1.25</td>
<td>x0.91</td>
<td>x1.85</td>
<td>x1.21</td>
</tr>
<tr>
<td>Sampling</td>
<td>x1260</td>
<td>x1260</td>
<td>x1848</td>
<td>x29581</td>
</tr>
<tr>
<td>Efficiency</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When UCT(AlphaZero) finishes $\frac{2}{3}$ of its first step, SW has already beaten by 25% its final score.

An example run:
- 128 walkers
- 14.20 samples / action
- Scored 27971 points
- Game len 6892
- 97894 samples
- 1min 38s. Runtime
- 70.34 fps
Improving Alphazero

- Change UTC for SW → sample x1000 + faster
- Stones as reward → SW jumps local optima
- Embedding of conv. layers for distance
- Use FMC to get better Q-values
- Heuristics only valid in Go
SW: Presenting an unfair benchmark

- A fair benchmark requires sampling 1M score at 150k samples / step
  - 10 min play: 12000 steps - One step: 400 µs
  - 1 core game: 4.8s x 150k x 50 rounds -> 416 days
  - Ideal M4.16xlarge: $3.20 / Hour → **500$ per game** running 1 instance for 6.5 days
  - 26,500$ on 53 games → Sponsors are welcome
Counting Paths vs. Trees

- Samples / step: confusing → Tree of games

Traditional Planning

Swarm Wave