AI in Games: Achievements and Challenges

Yuandong Tian
Facebook AI Research
The Game of Go

“A minute to learn, a lifetime to master”

AlphaGo versus LeeSedol (2016)

Master versus Ke Jie (2017)
Is this useful?
Game as a Vehicle of AI

- Infinite supply of fully labeled data
- Controllable and replicable
- Low cost per sample
- Faster than real-time
- Less safety and ethical concerns
- Complicated dynamics with simple rules.
Game as a Vehicle of AI

Algorithm is slow and data-inefficient

Require a lot of resources.

Abstract game to real-world

Hard to benchmark the progress
Game as a Vehicle of AI

Algorithm is slow and data-inefficient

Require a lot of resources.

Better Games

Abstract game to real-world

Hard to benchmark the progress
Good old days 1970s 1980s 1990s 2000s 2010s
Game Spectrum

Good old days 1970s 1980s 1990s 2000s 2010s

Go Chess Poker
Game Spectrum

- Good old days
- 1970s
- 1980s
- 1990s
- 2000s
- 2010s

Games:
- Pong (1972)
- Breakout (1978)
Game Spectrum

Good old days | 1970s | 1980s | 1990s | 2000s | 2010s

Game Spectrum

Good old days 1970s 1980s 1990s 2000s 2010s

Game Spectrum

Game Spectrum

Game as a Vehicle of AI

Algorithm is slow and data-inefficient

Abstract game to real-world

Better Algorithm/System

Require a lot of resources.

Hard to benchmark the progress

Better Environment
Our work

**Better Algorithm/System**

DarkForest Go Engine  
(Yuandong Tian, Yan Zhu, ICLR 2016)

**Better Environment**

ELF: Extensive Lightweight and Flexible Framework  
(Yuandong Tian et al, submitted to NIPS 2017)

Doom AI  
(Yuxin Wu, Yuandong Tian, ICLR 2017)
How Game AI works

Even with a super-super computer, it is not possible to search the entire space.
How Game AI works

Even with a super-super computer, it is not possible to search the entire space.

Current game situation

Lufei Ruan vs. Yifan Hou (2010)
How Game AI works

How many action do you have per step?

- Checker: a few possible moves
- Poker: a few possible moves
- Chess: 30-40 possible moves
- Go: 100-200 possible moves
- StarCraft: 50^100 possible moves

Alpha-beta pruning + Iterative deepening [Major Chess engine]
Counterfactual Regret Minimization [Libratus, DeepStack]
Monte-Carlo Tree Search + UCB exploration [Major Go engine]
???

Current game situation

Extensive search
Evaluate
Consequence
How Game AI works

How complicated is the game situation? How deep is the game?

- Chess
- Go
- Poker
- StarCraft

Rule-based
- Linear function for situation evaluation [Stockfish]
- End game database [Libratus]
- Random game play with simple rules [Zen, CrazyStone, DarkForest]
- Deep Value network [AlphaGo, DeepStack]

Current game situation

Extensive search

Evaluate
- Black wins
- White wins
- Black wins
- White wins
- Black wins
How to model Policy/Value function?

Non-smooth + high-dimensional
Sensitive to situations. One stone changes in Go leads to different game.

Traditional approach
• Many manual steps
• Conflicting parameters, not scalable.
• Need strong domain knowledge.

Deep Learning
• End-to-End training
  • Lots of data, less tuning.
• Minimal domain knowledge.
• Amazing performance
Case study: AlphaGo

• Computations
  • Train with many GPUs and inference with TPU.

• Policy network
  • Trained supervised from human replays.
  • Self-play network with RL.

• High quality playout/rollout policy
  • 2 microsecond per move, 24.2% accuracy. ~30%
  • Thousands of times faster than DCNN prediction.

• Value network
  • Predicts game consequence for current situation.
  • Trained on 30M self-play games.

## AlphaGo

- Policy network SL (trained with human games)

<table>
<thead>
<tr>
<th>Filters</th>
<th>Symmetries</th>
<th>Features</th>
<th>Test accuracy %</th>
<th>Train accuracy %</th>
<th>Raw wins %</th>
<th>AlphaGo wins %</th>
<th>Forward time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>1</td>
<td>48</td>
<td>54.6</td>
<td>57.0</td>
<td>36</td>
<td>53</td>
<td>2.8</td>
</tr>
<tr>
<td>192</td>
<td>1</td>
<td>48</td>
<td>55.4</td>
<td>58.0</td>
<td>50</td>
<td>50</td>
<td>4.8</td>
</tr>
<tr>
<td>256</td>
<td>1</td>
<td>48</td>
<td>55.9</td>
<td>59.1</td>
<td>67</td>
<td>55</td>
<td>7.1</td>
</tr>
<tr>
<td>256</td>
<td>2</td>
<td>48</td>
<td>56.5</td>
<td>59.8</td>
<td>67</td>
<td>38</td>
<td>13.9</td>
</tr>
<tr>
<td>256</td>
<td>4</td>
<td>48</td>
<td>56.9</td>
<td>60.2</td>
<td>69</td>
<td>14</td>
<td>27.6</td>
</tr>
<tr>
<td>256</td>
<td>8</td>
<td>48</td>
<td>57.0</td>
<td>60.4</td>
<td>69</td>
<td>5</td>
<td>55.3</td>
</tr>
<tr>
<td>192</td>
<td>1</td>
<td>4</td>
<td>47.6</td>
<td>51.4</td>
<td>25</td>
<td>15</td>
<td>4.8</td>
</tr>
<tr>
<td>192</td>
<td>1</td>
<td>12</td>
<td>54.7</td>
<td>57.1</td>
<td>30</td>
<td>34</td>
<td>4.8</td>
</tr>
<tr>
<td>192</td>
<td>1</td>
<td>20</td>
<td>54.7</td>
<td>57.2</td>
<td>38</td>
<td>40</td>
<td>4.8</td>
</tr>
<tr>
<td>192</td>
<td>8</td>
<td>4</td>
<td>49.2</td>
<td>53.2</td>
<td>24</td>
<td>2</td>
<td>36.8</td>
</tr>
<tr>
<td>192</td>
<td>8</td>
<td>12</td>
<td>55.7</td>
<td>58.3</td>
<td>32</td>
<td>3</td>
<td>36.8</td>
</tr>
</tbody>
</table>
| 192     | 8          | 20       | 55.8            | 58.4             | 42         | 3             | 36.8             

AlphaGo

• Fast Rollout (2 microsecond), ~30% accuracy

Monte Carlo Tree Search

Aggregate win rates, and search towards the good nodes.

\[ a_t = \arg\max_a (Q(s_t, a) + u(s_t, a)) \]

\[ u(s, a) \propto \frac{P(s, a)}{1 + N(s, a)} \]

Tree policy

Default policy

PUCT
AlphaGo

- Value Network (trained via 30M self-played games)
- How data are collected?

AlphaGo

- Value Network (trained via 30M self-played games)

<table>
<thead>
<tr>
<th>Short name</th>
<th>Policy network</th>
<th>Value network</th>
<th>Rollouts</th>
<th>Mixing constant</th>
<th>Policy GPUs</th>
<th>Value GPUs</th>
<th>Elo rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_{rvp}$</td>
<td>$p_\sigma$</td>
<td>$v_\theta$</td>
<td>$p_\pi$</td>
<td>$\lambda = 0.5$</td>
<td>2</td>
<td>6</td>
<td>2890</td>
</tr>
<tr>
<td>$\alpha_{vp}$</td>
<td>$p_\sigma$</td>
<td>$v_\theta$</td>
<td>$-$</td>
<td>$\lambda = 0$</td>
<td>2</td>
<td>6</td>
<td>2177</td>
</tr>
<tr>
<td>$\alpha_{rp}$</td>
<td>$p_\sigma$</td>
<td>$-$</td>
<td>$p_\pi$</td>
<td>$\lambda = 1$</td>
<td>8</td>
<td>0</td>
<td>2416</td>
</tr>
<tr>
<td>$\alpha_{rv}$</td>
<td>$[p_\tau]$</td>
<td>$v_\theta$</td>
<td>$p_\pi$</td>
<td>$\lambda = 0.5$</td>
<td>0</td>
<td>8</td>
<td>2077</td>
</tr>
<tr>
<td>$\alpha_{v}$</td>
<td>$[p_\tau]$</td>
<td>$v_\theta$</td>
<td>$-$</td>
<td>$\lambda = 0$</td>
<td>0</td>
<td>8</td>
<td>1655</td>
</tr>
<tr>
<td>$\alpha_{r}$</td>
<td>$[p_\tau]$</td>
<td>$-$</td>
<td>$p_\pi$</td>
<td>$\lambda = 1$</td>
<td>0</td>
<td>0</td>
<td>1457</td>
</tr>
<tr>
<td>$\alpha_{p}$</td>
<td>$p_\sigma$</td>
<td>$-$</td>
<td>$-$</td>
<td>$-$</td>
<td>0</td>
<td>0</td>
<td>1517</td>
</tr>
</tbody>
</table>

AlphaGo

Our work
Our computer Go player: DarkForest

Yuandong Tian and Yan Zhu, ICLR 2016

- DCNN as a tree policy
  - Predict next k moves (rather than next move)
  - Trained on 170k KGS dataset/80k GoGoD, 57.1% accuracy.
  - KGS 3D without search (0.1s per move)
  - Release 3 month before AlphaGo, < 1% GPUs (from Aja Huang)
Our computer Go player: DarkForest

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our/enemy liberties</td>
</tr>
<tr>
<td>Ko location</td>
</tr>
<tr>
<td>Our/enemy stones/empty place</td>
</tr>
<tr>
<td>Our/enemy stone history</td>
</tr>
<tr>
<td>Opponent rank</td>
</tr>
</tbody>
</table>

Feature used for DCNN

feature type: standard

![Graph showing winrate against Pachi 10k over epochs for different nstep values (nstep=1, nstep=2, nstep=3)]
Pure DCNN

*darkforest*: Only use top-1 prediction, trained on KGS
*darkfores1*: Use top-3 prediction, trained on GoGoD
*darkfores2*: *darkfores1* with fine-tuning.

<table>
<thead>
<tr>
<th></th>
<th>GnuGo (level 10)</th>
<th>Pachi 10k</th>
<th>Pachi 100k</th>
<th>Fuego 10k</th>
<th>Fuego 100k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clark &amp; Storkey (2015)</td>
<td>91.0</td>
<td>-</td>
<td>-</td>
<td></td>
<td>14.0</td>
</tr>
<tr>
<td>Maddison et al. (2015)</td>
<td>97.2</td>
<td>47.4</td>
<td>11.0</td>
<td>23.3</td>
<td>12.5</td>
</tr>
<tr>
<td><strong>darkforest</strong></td>
<td>98.0 ± 1.0</td>
<td>71.5 ± 2.1</td>
<td>27.3 ± 3.0</td>
<td>84.5 ± 1.5</td>
<td>56.7 ± 2.5</td>
</tr>
<tr>
<td><strong>darkfores1</strong></td>
<td>99.7 ± 0.3</td>
<td>88.7 ± 2.1</td>
<td>59.0 ± 3.3</td>
<td>93.2 ± 1.5</td>
<td>78.0 ± 1.7</td>
</tr>
<tr>
<td><strong>darkfores2</strong></td>
<td>100 ± 0.0</td>
<td>94.3 ± 1.7</td>
<td>72.6 ± 1.9</td>
<td>98.5 ± 0.1</td>
<td>89.7 ± 2.1</td>
</tr>
</tbody>
</table>

Win rate between DCNN and open source engines.
# DCNN + MCTS

**darkfmcts3**: Top-3/5, 75k rollouts, ~12sec/move, KGS 5d

<table>
<thead>
<tr>
<th></th>
<th>darkforest+MCTS</th>
<th>darkforest1+MCTS</th>
<th>darkforest2+MCTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vs pure DCNN (1000rl/top-20)</td>
<td>84.8%</td>
<td>74.0%</td>
<td>62.8%</td>
</tr>
<tr>
<td>Vs pure DCNN (1000rl/top-5)</td>
<td>89.6%</td>
<td>76.4%</td>
<td>68.4%</td>
</tr>
<tr>
<td>Vs pure DCNN (1000rl/top-3)</td>
<td>91.6%</td>
<td>89.6%</td>
<td>79.2%</td>
</tr>
<tr>
<td>Vs pure DCNN (5000rl/top-5)</td>
<td>96.8%</td>
<td>94.3%</td>
<td>94.2%</td>
</tr>
<tr>
<td>Vs Pachi 10k (pure DCNN baseline)</td>
<td>71.5%</td>
<td>88.7%</td>
<td>94.3%</td>
</tr>
<tr>
<td>Vs Pachi 10k (1000rl/top-20)</td>
<td>91.2% (+19.7%)</td>
<td>92.0% (+3.3%)</td>
<td>95.2% (+0.9%)</td>
</tr>
<tr>
<td>Vs Pachi 10k (1000rl/top-5)</td>
<td>88.4% (+16.9%)</td>
<td>94.4% (+5.7%)</td>
<td>97.6% (+3.3%)</td>
</tr>
<tr>
<td>Vs Pachi 10k (1000rl/top-3)</td>
<td>95.2% (+23.7%)</td>
<td>98.4% (+9.7%)</td>
<td>99.2% (+4.9%)</td>
</tr>
<tr>
<td>Vs Pachi 10k (5000/top-5)</td>
<td>98.4%</td>
<td>99.6%</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

Win rate between DCNN + MCTS and open source engines.
Our computer Go player: DarkForest

• DCNN+MCTS
  • Use top3/5 moves from DCNN, 75k rollouts.
  • Stable KGS 5d. Open source. [https://github.com/facebookresearch/darkforestGo](https://github.com/facebookresearch/darkforestGo)
  • 3rd place on KGS January Tournaments
  • 2nd place in 9th UEC Computer Go Competition (Not this time 😊)

---

DarkForest versus Koichi Kobayashi (9p)
Win Rate analysis (using DarkForest) (AlphaGo versus Lee Sedol)
First Person Shooter (FPS) Game

Yuxin Wu, Yuandong Tian, ICLR 2017
Network Structure

Simple Frame Stacking is very useful (rather than Using LSTM)
Encourage actions leading to states with high-than-expected value.
Encourage value function to converge to the true cumulative rewards.
Keep the diversity of actions
Curriculum Training

From simple to complicated
Curriculum Training

<table>
<thead>
<tr>
<th></th>
<th>Class 0</th>
<th>Class 1</th>
<th>Class 2</th>
<th>Class 3</th>
<th>Class 4</th>
<th>Class 5</th>
<th>Class 6</th>
<th>Class 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>0.2</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
<td>0.6</td>
<td>0.8</td>
<td>0.8</td>
<td>1.0</td>
</tr>
<tr>
<td>Health</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>80</td>
<td>100</td>
</tr>
</tbody>
</table>

FlatMap
VizDoom AI Competition 2016 (Track1)

We won the first place!

<table>
<thead>
<tr>
<th>Rank</th>
<th>Bot</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>Total frags</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F1</td>
<td>56</td>
<td>62</td>
<td>n/a</td>
<td>54</td>
<td>47</td>
<td>43</td>
<td>47</td>
<td>55</td>
<td>50</td>
<td>48</td>
<td>50</td>
<td>559</td>
</tr>
<tr>
<td>2</td>
<td>Arnold</td>
<td>36</td>
<td>34</td>
<td>42</td>
<td>36</td>
<td>36</td>
<td>36</td>
<td>45</td>
<td>36</td>
<td>39</td>
<td>n/a</td>
<td>33</td>
<td>36</td>
</tr>
<tr>
<td>3</td>
<td>CLYDE</td>
<td>37</td>
<td>n/a</td>
<td>38</td>
<td>32</td>
<td>37</td>
<td>30</td>
<td>46</td>
<td>42</td>
<td>33</td>
<td>24</td>
<td>44</td>
<td>393</td>
</tr>
</tbody>
</table>

Videos:
https://www.youtube.com/watch?v=94EPSjQH38Y
https://www.youtube.com/watch?v=Qv4esGWOg7w&t=394s
Visualization of Value functions

Best 4 frames (agent is about to shoot the enemy)

Worst 4 frames (agent missed the shoot and is out of ammo)
ELF: Extensive, Lightweight and Flexible Framework for Game Research

Yuandong Tian, Qucheng Gong, Wendy Shang, Yuxin Wu, Larry Zitnick (Submitted to NIPS 2017)

• Extensive
  • Any games with C++ interfaces can be incorporated.

• Lightweight
  • Fast. Mini-RTS (40K FPS per core)
  • Minimal resource usage (1GPU+several CPUs)

• Flexible
  • Environment-Actor topology
  • Change of parameters in the game environments.
  • Choice of different RL methods.

Arxiv: https://arxiv.org/abs/1707.01067
Repository: https://github.com/facebookresearch/ELF
Possible Usage

• Game Research
  • Board game (Chess, Go, etc)
  • Real-time Strategy Game

• Discrete/Continuous control
  • Robotics

• Dialog and Q&A System
Sample Usage – Initialization

```python
# Sample Usage
# We run 1024 games concurrently.
num_games = 1024

# Every time we wait for an arbitrary set of 256 games and return.
batchsize = 256

# The return states contain key 's', 'r' and 'terminal'
# and the reply contains key 'a', which is to be filled from the Python side.
# Their definitions are defined in the C++ wrapper of the game.
input_spec = dict(s='', r='', terminal='')
reply_spec = dict(a='')

GameContext = InitializeGame(num_games, batchsize, input_spec, reply_spec)

# Start all game threads
GameContext.Start()
```
Sample Usage – Main Loop

```python
while True:
    # Wait until a batch of game states are returned.
    # Note that these game instances will be blocked.
    batch = GameContext.Wait()

    # Apply a model to the game state.
    # You can do forward/backward propagation here.
    # Assuming that the output has key 'pi'
    output = model(batch)

    # Sample from the output to get the actions of this batch.
    reply['a'][:] = SampleFromDistribution(output)

    # Resume games.
    GameContext.Steps()

    # Stop all game threads.
    GameContext.Stop()
```
RLPytorch

• A RL platform in PyTorch
• A3C in 30 lines.
• Interfacing with dict.
Plug-and-play; no worry about the concurrency anymore.
Flexible Environment-Actor topology

(a) One-to-One
(b) Many-to-One
(c) One-to-Many

Vanilla A3C
BatchA3C, GA3C
Self-Play, Monte-Carlo Tree Search
Architecture Hierarchy

ELF

- Go (DarkForest)
  - Pong
  - Breakout
- ALE
- RTS Engine
  - Mini-RTS
  - Capture the Flag
  - Tower Defense

An extensive framework that can host many games.

Specific game engines.

Environments
A miniature RTS engine

<table>
<thead>
<tr>
<th>Game Name</th>
<th>Descriptions</th>
<th>Avg Game Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mini-RTS</td>
<td>Gather resource and build troops to destroy opponent’s base.</td>
<td>1000-6000 ticks</td>
</tr>
<tr>
<td>Capture the Flag</td>
<td>Capture the flag and bring it to your own base.</td>
<td>1000-4000 ticks</td>
</tr>
<tr>
<td>Tower Defense</td>
<td>Builds defensive towers to block enemy invasion.</td>
<td>1000-2000 ticks</td>
</tr>
</tbody>
</table>
Simulation Speed

**KFPS per CPU core for Mini-RTS**

- 64 threads: 60,000
- 128 threads: 53,000
- 256 threads: 60,000
- 512 threads: 60,000
- 1024 threads: 60,000

**KFPS per CPU core for Pong (Atari)**

- 64 threads: 60,000
- 128 threads: 53,000
- 256 threads: 60,000
- 512 threads: 60,000
- 1024 threads: 60,000

<table>
<thead>
<tr>
<th>Platform</th>
<th>ALE</th>
<th>RLE</th>
<th>Universe</th>
<th>Malmo</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPS</td>
<td>6000</td>
<td>530</td>
<td>60</td>
<td>120</td>
</tr>
<tr>
<td>Platform</td>
<td>DeepMind Lab</td>
<td>VizDoom</td>
<td>TorchCraft</td>
<td><strong>Mini-RTS</strong></td>
</tr>
<tr>
<td>FPS</td>
<td>287(C) / 866(G)</td>
<td>7,000</td>
<td>2,000 (Frameskip=50)</td>
<td><strong>40,000</strong></td>
</tr>
</tbody>
</table>
Training AI

Game visualization

Game internal data (respecting fog of war)

Location of all range tanks
Location of all melee tanks
Location of all workers
HP portion
Resource

Using Internal Game data and A3C.
Reward is only available once the game is over.
MiniRTS

Building that can build workers and collect resources.

Resource unit that contains 1000 minerals.

Building that can build melee attacker and range attacker.

Worker who can build barracks and gather resource.
Low speed in movement and low attack damage.

Tank with high HP, medium movement speed, short attack range, high attack damage.

Tank with low HP, high movement speed, long attack range and medium attack damage.
**Training AI**

9 discrete actions.

<table>
<thead>
<tr>
<th>No.</th>
<th>Action name</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IDLE</td>
<td>Do nothing</td>
</tr>
<tr>
<td>2</td>
<td>BUILD WORKER</td>
<td>If the base is idle, build a worker</td>
</tr>
<tr>
<td>3</td>
<td>BUILD BARRACK</td>
<td>Move a worker (gathering or idle) to an empty place and build a barrack.</td>
</tr>
<tr>
<td>4</td>
<td>BUILD MELEE ATTACKER</td>
<td>If we have an idle barrack, build an melee attacker.</td>
</tr>
<tr>
<td>5</td>
<td>BUILD RANGE ATTACKER</td>
<td>If we have an idle barrack, build a range attacker.</td>
</tr>
<tr>
<td>6</td>
<td>HIT AND RUN</td>
<td>If we have range attackers, move towards opponent base and attack. Take advantage of their long attack range and high movement speed to hit and run if enemy counter-attack.</td>
</tr>
<tr>
<td>7</td>
<td>ATTACK</td>
<td>All melee and range attackers attack the opponent’s base.</td>
</tr>
<tr>
<td>8</td>
<td>ATTACK IN RANGE</td>
<td>All melee and range attackers attack enemies in sight.</td>
</tr>
<tr>
<td>9</td>
<td>ALL DEFEND</td>
<td>All troops attack enemy troops near the base and resource.</td>
</tr>
</tbody>
</table>
Win rate against rule-based AI

Frame skip (how often AI makes decisions)

<table>
<thead>
<tr>
<th>Frame skip</th>
<th>AI_SIMPLE (mean/std)</th>
<th>AI_HIT_AND_RUN (mean/std)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>68.4(±4.3)</td>
<td>63.6(±7.9)</td>
</tr>
<tr>
<td>20</td>
<td>61.4(±5.8)</td>
<td>55.4(±4.7)</td>
</tr>
<tr>
<td>10</td>
<td>52.8(±2.4)</td>
<td>51.1(±5.0)</td>
</tr>
</tbody>
</table>

Network Architecture

<table>
<thead>
<tr>
<th>Network Architecture</th>
<th>SIMPLE (median)</th>
<th>SIMPLE (mean/std)</th>
<th>HIT_AND_RUN (median)</th>
<th>HIT_AND_RUN (mean/std)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReLU</td>
<td>52.8</td>
<td>54.7(±4.2)</td>
<td>60.4</td>
<td>57.0(±6.8)</td>
</tr>
<tr>
<td>Leaky ReLU</td>
<td>59.8</td>
<td>61.0(±2.6)</td>
<td>60.2</td>
<td>60.3(±3.3)</td>
</tr>
<tr>
<td>ReLU + BN</td>
<td>61.0</td>
<td>64.4(±7.4)</td>
<td>55.6</td>
<td>57.5(±6.8)</td>
</tr>
<tr>
<td>Leaky ReLU + BN</td>
<td>72.2</td>
<td>68.4(±4.3)</td>
<td>65.5</td>
<td>63.6(±7.9)</td>
</tr>
</tbody>
</table>
Effect of T-steps

Large T is better.
Transfer Learning and Curriculum Training

<table>
<thead>
<tr>
<th></th>
<th>AI_SIMPLE</th>
<th>AI_HIT_AND_RUN</th>
<th>Combined (50%SIMPLE+50% H&amp;R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMPLE</td>
<td>68.4 (±4.3)</td>
<td>26.6 (±7.6)</td>
<td>47.5 (±5.1)</td>
</tr>
<tr>
<td>HIT_AND_RUN</td>
<td>34.6 (±13.1)</td>
<td>63.6 (±7.9)</td>
<td>49.1 (±10.5)</td>
</tr>
<tr>
<td>Combined (No curriculum)</td>
<td>49.4 (±10.0)</td>
<td>46.0 (±15.3)</td>
<td>47.7 (±11.0)</td>
</tr>
<tr>
<td>Combined</td>
<td>51.8 (±10.6)</td>
<td>54.7 (±11.2)</td>
<td>53.2 (±8.5)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>AI_SIMPLE</th>
<th>AI_HIT_AND_RUN</th>
<th>CAPTURE_THE_FLAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without curriculum training</td>
<td>66.0 (±2.4)</td>
<td>54.4 (±15.9)</td>
<td>54.2 (±20.0)</td>
</tr>
<tr>
<td>With curriculum training</td>
<td>68.4 (±4.3)</td>
<td>63.6 (±7.9)</td>
<td>59.9 (±7.4)</td>
</tr>
</tbody>
</table>
Videos
Future Work

• Richer game scenarios.
  • Multiple bases (Expand? Rush? Defending?)
  • More complicated units.
• More Realistic action space
  • Assign one action per unit
• Model-based Reinforcement Learning
  • MCTS with perfect information and perfect dynamics also achieves ~70% winrate
• Self-Play (Trained AI versus Trained AI)
The other direction: Data-Driven Methods of Non-convex Problems

• Idea
  • Current optimization assumes arbitrary data distribution
    • A convex function is always convex no matter what the input data is.
  • What can we guarantee, if we have some key information about data distributions?

• My Previous Works
  • Data-driven descent (CVPR 2010)
  • Hierarchical Data-Driven Descent (ICCV 2013, Marr Prize Honorable mention)
  • Analysis on 2-Layered ReLU network with Gaussian input (ICML 2017)
  • And more...
Thanks!