Deep Reinforcement Learning and its Applications in Games

Yuandong Tian
Facebook AI Research
Overview

• Introduction: AI and Games
• Basic knowledge in Reinforcement Learning
  • Q-learning
  • Policy gradient
  • Actor-critic models
  • Game related approaches
• Case study
  • AlphaGo Fan and AlphaGo Zero
• Our work
  • DarkForest
  • Doom AI bot
  • ELF platform
Part I: Introduction
AI works in a lot of situations:

- **Object Recognition**
- **Medical**
- **Translation**
- **Speech Recognition**

- **Personalization**
- **Surveillance**
- **Smart Design**
- **Board game**
What AI still needs to improve

Less supervised data
Complicated/Unknown environments with lots of corner cases.
Common Sense
The Charm of Games

Complicated long-term strategies.

Realistic Worlds
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

Abstract game to real-world

Require a lot of resources.

Hard to benchmark the progress

Better Algorithm/System

Better Environment
What’s new in Game environment?

• Data are generated on the fly
• Agent not only learns from the data, but also choose which data to learn.
Part II: Reinforcement Learning
What is Reinforcement Learning?

Agent

State s

Reward r

Environment

Action a
What is Reinforcement Learning?

State:
where you are?

Action:
left/right/up/down

Next state:
where you are after the action?
What is Reinforcement Learning?

State: 
\((x, y) = (6, 0)\)

Actions:
- Left: \(x -= 1\)
- Right: \(x += 1\)
- Up: \(y -= 1\)
- Down: \(y += 1\)
What is Reinforcement Learning?

State:

\[ s = (x, y) = (6, 0) \]

Actions:

Left: \( x \leftarrow x - 1 \)
Right: \( x \leftarrow x + 1 \)
Up: \( y \leftarrow y - 1 \)
Down: \( y \leftarrow y + 1 \)
What is Reinforcement Learning?
Goal of Reinforcement Learning
Goal of Reinforcement Learning

**Agent**

- **State** $s_t$ → **Agent** → **Environment** → **State** $s_{t+1}$
- **Reward** $r_t$ → **Agent**

**Environment**

- **Action** $a_t$ → **State** $s_{t+1}$
- **Action** $a_{t+1}$ → **State** $s_{t+2}$
- **Action** $a_{t+2}$ → **State**

Maximize long-term reward: $\max \sum_{t'=t}^{+\infty} \gamma^{t'-t} r_{t'}$
Key Quantities

\[ V(s) \] Maximal reward you can get starting from \( S \)

"Value" of state \( S \)
Key Quantities

\[ Q(s, a) \] Maximal reward you can get starting from state \( S \) and action \( a \)

\( Q \)-function of state \( S \) and action \( a \)
Bellman Equations

\[ Q^*(s, a) = r(s, a) + \gamma \max_{a'} Q^*(s'(s, a), a') \]

\[ V^*(s) = \max_a r(s, a) + \gamma V^*(s'(s, a)) \]

Optimal solution
Algorithm

Tabular Q-learning

\[
Q^{(n)}(s, a) \leftarrow r(s, a) + \gamma \max_{a'} Q^{(n-1)}(s'(s, a), a')
\]

Value Iteration

\[
V^{(n)}(s) \leftarrow \max_{a} r(s, a) + \gamma V^{(n-1)}(s'(s, a))
\]

As long as we can enumerate all possible states and actions
Q-learning

\[ Q^{(n)}(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \max_{a'} Q^{(n-1)}(s_{t+1}, a') \]
On trajectories

Q-learning

$$Q^{(n)}_{\theta}(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \max_{a'} Q^{(n-1)}_{\theta}(s_{t+1}, a')$$

$Q_{\theta}(s, a)$ now have generalization capability

How could you take the gradient w.r.t $\theta$?
On trajectories

Q-learning

\[ Q_{\theta}^{(n)}(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \max_{a'} Q_{\theta'}^{(n-1)}(s_{t+1}, a') \]
On trajectories

Q-learning

\[ Q_\theta(s_t, a_t) \leftarrow (1 - \alpha) Q_\theta(s_t, a_t) + \alpha \left[ r(s_t, a_t) + \gamma \max_{a'} Q_{\theta'}(s_{t+1}, a') \right] \]

Sample trajectories

Q-learning

\[
Q^{(n)}_{\theta}(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \max_{a'} Q^{(n-1)}_{\theta}(s_{t+1}, a')
\]

How could we sample a trajectory in the state space?

*Behavior policy* $\beta(\cdot | s)$
On-policy versus Off-policy approaches

Off-policy, sampled by some behavior policy $\beta(\cdot|s)$
- Expert behaviors (imitation learning)
- Supervised learning

On-policy, sampled by the current models $Q^{(n)}(s, a) \pi(\cdot|s)$

Agent not only learns from the data, but also chooses which data to learn.
Policy gradient

$J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} \left[ r(\tau) \right]$ 

$\pi_{\theta}(a|s)$ Probability of taking action $a$ given state $s$

$r(\tau)$ Cumulative reward along a trajectory $\tau$
Policy gradient

\[ J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} [r(\tau)] \]

\[ \nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} [r(\tau) \nabla_\theta \log p_\theta(\tau)] \]
Policy gradient

\[ \nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ r(\tau) \nabla_{\theta} \log p_\theta(\tau) \right] \]

\[ \log p_\theta(\tau) = \log p(s_1) + \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \sum_{t=1}^{T} \log p(s_{t+1}|s_t, a_t) \]

Independent of \( \theta \)
Policy gradient

\[ \nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ r(\tau) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t | s_t) \right] \]

Estimated by sampling \( \pi_\theta(a | s) \)
Baseline

$$\mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \nabla_\theta \log p_\theta(\tau) \right] \equiv 0$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ r(\tau) \nabla_\theta \log p_\theta(\tau) \right]$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ (r(\tau) - b) \nabla_\theta \log p_\theta(\tau) \right]$$

Can be any function that only depends on state.
REINFORCE

\[ r(\tau) \quad \text{Actual reward obtained by rolling out from } S \]

\[ S \quad \pi_\theta(a|s) \]

\[ \tau_1, r(\tau_1) \]
\[ \tau_2, r(\tau_2) \]
\[ \tau_3, r(\tau_3) \]

\[ \nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[ r_i(\tau) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) \right] \]
REINFORCE

\[ r(\tau) \quad \text{Actual reward obtained by rolling out from } S \]

\[ \pi_\theta(a|s) \]

\[ \tau_1, r(\tau_1) \quad + \]
\[ \tau_2, r(\tau_2) \quad + \]
\[ \tau_3, r(\tau_3) \quad - \]
\[ \tau_4, r(\tau_4) \quad + \]

Too many positive rewards.
We only want to pick the best of the best.
REINFORCE

\[ r(\tau) \quad \text{Actual reward obtained by rolling out from} \ S \]

\[ \tau_1, r(\tau_1) \quad + \quad - \quad b \]
\[ \tau_2, r(\tau_2) \quad + \quad - \quad b \]
\[ \tau_3, r(\tau_3) \quad - \quad - \quad b \]
\[ \tau_4, r(\tau_4) \quad + \quad - \quad b \]

\[ b \quad \text{mean of all the rewards} \]
Actor-Critic Models

\[ r(\tau) \approx Q^\pi_\theta(s, a) \] Rollout return as a parametric function (critic)

\[ b(s) = V_\theta(s) \] Use the value function as the baseline

\[ r(\tau) - b(s) \approx Q^\pi_\theta(s, a) - V_\theta(s) = A^\pi_\theta(s, a) \]

Estimated from TD difference in the data

“Advantageous Actor-Critic”
A2C / A3C

\[ \nabla \log \pi(a|s_t)(R_t - V(s_t)) \]

"Update Policy network"

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
Part III: Algorithm used in Games
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.

Lufei Ruan vs. Yifan Hou (2010)

Current game situation

Extensive search → Evaluate → Consequence

Black wins  Black wins  Black wins
White wins  White wins  Black wins
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

Actor

Extensive search

Evaluate

Consequence

Black wins
White wins
Black wins
White wins
Black wins
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
- Random game play with simple rules [Zen, CrazyStone, DarkForest]
- Deep Value network [AlphaGo, DeepStack]

Current game situation

Extensive search

Evaluate

Consequence

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
Alpha-beta Pruning

A good counter move eliminates other choices.

Move order is important!

---

Fix depth

---

P1

P2

good move for P1

bad move for P1

good move for P2

---

---
Monte Carlo Tree Search

Aggregate win rates, and search towards the good nodes.

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

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

PUCT

(a) Tree policy

(b) Default policy

(c) Path to the root
Part IV: Case Study
AlphaGo Fan

• 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 Fan

- 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>
<tr>
<td>192</td>
<td>8</td>
<td>20</td>
<td>55.8</td>
<td>58.4</td>
<td>42</td>
<td>3</td>
<td>36.8</td>
</tr>
</tbody>
</table>

AlphaGo Fan

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

AlphaGo Fan

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

### AlphaGo Fan

- 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 Fan

AlphaGo Zero
AlphaGo Zero

\[(s_1, \pi_1, z)\]

\[(s_2, \pi_2, z)\]

\[(s_3, \pi_3, z)\]

Training samples for \(\theta_{i+1}\)

MCTS

\(\theta_i\)

\(\pi_1\)

\(\pi_2\)

\(\pi_3\)

\(z\)
AlphaGo Zero

\[ J(\theta) = (z - V_{\theta})^2 - \pi^T \log p_{\theta} + c\|\theta\|^2 \]
AlphaGo Zero
Using ResNet and shared network
AlphaGo Zero Strength

• 3 days version
  • 4.9M Games, 1600 rollouts/move
  • 20 block ResNet
  • Defeat AlphaGo Lee by 100:0.

• 40 days version
  • 29M Games, 1600 rollouts/move
  • 40 blocks ResNet.
  • Defeat AlphaGo Master by 89:11
Computation Time

Game Playing: 4 TPUs

Supervised network training
64 GPU (32 batchsize/GPU)
0.7 million mini-batch of size 2048 (370ms per batch)

Training data generation
4.9 M Games * 1600 rollouts/move (0.4s) * (~250 move/game)
= 15.5 years

15.5 years / 3 days = 1890 machines

Using TPU, single rollout 0.25ms

4.7k game situations/sec
18.9 games / sec
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

Better Algorithm/System

Better Environment
Our work

**Better Algorithm/System**

- DarkForest Go Engine
  (Yuandong Tian, Yan Zhu, ICLR16)

- Doom AI
  (Yuxin Wu, Yuandong Tian, ICLR17)

**Better Environment**

- ELF: Extensive Lightweight and Flexible Framework
  (Yuandong Tian et al, arXiv)
Our computer Go player: DarkForest

- DCNN as a tree policy
  - Predict next k moves (rather than next move)
  - Trained on 170k KGS dataset/80k GoGoD, \textbf{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>
<th>Feature used for DCNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our/enemy liberties</td>
<td></td>
</tr>
<tr>
<td>Ko location</td>
<td></td>
</tr>
<tr>
<td>Our/enemy stones/empty place</td>
<td></td>
</tr>
<tr>
<td>Our/enemy stone history</td>
<td></td>
</tr>
<tr>
<td>Opponent rank</td>
<td></td>
</tr>
</tbody>
</table>
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)
Win Rate analysis (using DarkForest)
First Person Shooter (FPS) Game

Yuxin Wu, Yuandong Tian, ICLR 2017

Play the game from the raw image!
Network Structure

Simple Frame Stacking is very useful (rather than Using LSTM)
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>
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>45</td>
<td>36</td>
<td>39</td>
<td>n/a</td>
<td>33</td>
<td>36</td>
<td>413</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 (NIPS 2017 Oral)

https://github.com/facebookresearch/ELF

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

• Lightweight
  • Fast. Mini-RTS (40K FPS per core)
  • Minimal resource usage (1GPU+several CPUs)
  • Fast training (a couple of hours for a RTS game)

• Flexible
  • Environment-Actor topology
  • Parametrized game environments.
  • Choice of different RL methods.

Arxiv: https://arxiv.org/abs/1707.01067
How RL system works

Game 1 → Process 1 → Actor → Model → Optimizer → Consumers (Python)

Game 2 → Process 2

Game N → Process N

Replay Buffer
ELF design

Plug-and-play; no worry about the concurrency anymore.
Possible Usage

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

• Complicated RL algorithms.

• Discrete/Continuous control
  • Robotics

• Dialog and Q&A System
Initialization

# 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', 'V' and 'pi', which is to be filled from the Python side.
# Their definitions are defined in the C++ wrapper of the game.
desc = dict(
    actor = dict(
        batchsize=args.batchsize,
        input=dict(T=1, keys=set(['s', 'last_r', 'last_terminal'])),
        reply=dict(T=1, keys=set(['pi', 'V', 'a']))
    )
)

GameContext = InitializeGame(num_games, batchsize, desc)
Main Loop

# Start all game threads
GameContext.Start()

while True:
    # Wait until a batch of game states are returned.
    # Note that these game instances will be blocked.
    batch = GameContext.Wait()
    if batch.desc == "actor":
        # Apply a model to the game state. You can do forward/backward propagation here.
        output = model(batch)

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

    # Resume games.
    GameContext.Steps()

# Stop all game threads.
GameContext.Stop()
Training

desc = dict(
    actor = dict(
        batchsize=args.batchsize,
        input=dict(T=1, keys=set(['s', 'last_r', 'last_terminal'])),
        reply=dict(T=1, keys=set(['pi', 'v', 'a']))
    ),
    train = dict(
        batchsize=args.batchsize,
        input=dict(T=6, keys=set(['s', 'last_r', 'last_terminal', 'a', 'pi'])),
        reply=None
    )
)

while True:
    ...

    if batch['desc'] == "actor":
        # Act given the current states to move the game environment forward.
        # It could be an act for a game, for its internal MCTS search, etc.
        # 
    elif batch['desc'] == "train":
        # Train your model. All the previous actions of the games and
        # their probabilities can be made available.
        ...

Self-Play

desc = dict(
    actor0 = dict(
        batchsize=args.batchsize,
        input=dict(T=1, keys=set(['s', 'last_r', 'last_terminal'])),
        reply=dict(T=1, keys=set(['pi', 'v', 'a'])),
        filter=dict(id=0)
    ),
    actor1 = dict(
        batchsize=args.batchsize,
        input=dict(T=1, keys=set(['s', 'last_r', 'last_terminal'])),
        reply=dict(T=1, keys=set(['pi', 'v', 'a'])),
        filter=dict(id=1)
    ),
    train = dict(
        batchsize=args.batchsize,
        input=dict(T=6, keys=set(['s', 'last_r', 'last_terminal', 'a', 'pi'])),
        reply=None,
        filter=dict(id=0)
    )
)

while True:
    ...
    if batch['desc'] == 'actor0':
        # Act for player 0
    elif batch['desc'] == 'actor1':
        # Act for player 1
    elif batch['desc'] == 'train':
        # Train your model only for player 0.
    ...
    ...


Multi-Agent

desc = {}
for i in range(num_agents):
    desc['actor%d' % i] = dict(
        batchsize=argv.batchsize,
        input=dict(T=1, keys=set(['s', 'last_r', 'last_terminal'])),
        reply=dict(T=1, keys=set(['pi', 'v', 'a'])),
        filter=dict(id=i)
    )

while True:
    ...
    for i in range(num_agents):
        if batch['desc'] == 'actor%d' % i:
            # Act for player i
    ...

Monte-Carlo Tree Search

desc = dict(
    actor = dict(
        batchsize=argv.batchsize,
        input=
        dict(T=1,
             keys=set(['s', 'last_r', 'last_terminal'])),
        reply=dict(T=1, keys=set(['pi', 'v', 'a'])),
    )
)

while True:
    batch = GameContext.Wait()
    if batch['desc'] == 'actor':
        # Act for player. During MCTS search, one
        # game instance could send multiple requests
        # for python side to respond.
        GameContext.Step()
Flexible Environment-Actor topology

(a) One-to-One
Vanilla A3C

(b) Many-to-One
BatchA3C, GA3C

(c) One-to-Many
Self-Play,
Monte-Carlo Tree Search
# A3C

def update(self, batch):
    ''' Actor critic model '''
    R = deepcopy(batch["V"][T - 1])
    batchsize = R.size(0)
    R.resize_(batchsize, 1)

    for t in range(T - 2, -1, -1):
        # Forward pass
        curr = self.model_interface.forward("model", batch.hist(t))

        # Compute the reward.
        R = R * self.args.discount + batch["r"][t]
        # If we see any terminal signal, do not backprop
        for i, terminal in enumerate(batch["terminal"][t]):
            if terminal: R[t][i] = curr["V"].data[i]

        # We need to set it beforehand.
        self.policy_gradient_weights = R - curr["V"].data

        # Compute policy gradient error:
        errs = self._compute_policy_entropy_err(curr["pi"], batch["a"][t])
        # Compute critic error
        value_err = self.value_loss(curr["V"], Variable(R))

        overall_err = value_err + errs["policy_err"]
        overall_err += errs["entropy_err"] * self.args.entropy_ratio
        overall_err.backward()
Architecture Hierarchy

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

<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>Mini-RTS</td>
</tr>
<tr>
<td>FPS</td>
<td>287(C) / 866(G)</td>
<td>7,000</td>
<td>2,000 (Frameskip=50)</td>
<td>40,000</td>
</tr>
</tbody>
</table>

## KFPS per CPU core for Mini-RTS

<table>
<thead>
<tr>
<th>Threads</th>
<th>1 core</th>
<th>2 cores</th>
<th>4 cores</th>
<th>8 cores</th>
<th>16 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>65</td>
<td>62</td>
<td>60</td>
<td>58</td>
<td>55</td>
</tr>
<tr>
<td>128</td>
<td>64</td>
<td>62</td>
<td>60</td>
<td>58</td>
<td>55</td>
</tr>
<tr>
<td>256</td>
<td>62</td>
<td>60</td>
<td>58</td>
<td>55</td>
<td>53</td>
</tr>
<tr>
<td>512</td>
<td>60</td>
<td>58</td>
<td>55</td>
<td>53</td>
<td>51</td>
</tr>
<tr>
<td>1024</td>
<td>58</td>
<td>55</td>
<td>53</td>
<td>51</td>
<td>49</td>
</tr>
</tbody>
</table>

## KFPS per CPU core for Pong (Atari)

<table>
<thead>
<tr>
<th>Threads</th>
<th>1 core</th>
<th>2 cores</th>
<th>4 cores</th>
<th>8 cores</th>
<th>16 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>5.8</td>
<td>5.5</td>
<td>5.2</td>
<td>4.9</td>
<td>4.6</td>
</tr>
<tr>
<td>128</td>
<td>5.7</td>
<td>5.4</td>
<td>5.1</td>
<td>4.8</td>
<td>4.5</td>
</tr>
<tr>
<td>256</td>
<td>5.5</td>
<td>5.2</td>
<td>4.9</td>
<td>4.6</td>
<td>4.3</td>
</tr>
<tr>
<td>512</td>
<td>5.3</td>
<td>4.9</td>
<td>4.6</td>
<td>4.3</td>
<td>4.0</td>
</tr>
<tr>
<td>1024</td>
<td>5.0</td>
<td>4.7</td>
<td>4.4</td>
<td>4.1</td>
<td>3.8</td>
</tr>
</tbody>
</table>

**Notes:**
- Frameskip = 50
- 6CPU + 1GPU
Training AI

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 (median)</th>
<th>AI_SIMPLE (mean/±std)</th>
<th>AI_HIT_AND_RUN (median)</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>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>61.4±5.8</td>
<td>55.4±4.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>52.8±2.4</td>
<td>51.1±5.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Network Architecture

<table>
<thead>
<tr>
<th></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>Mixture of SIMPLE_Ai and Trained AI</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>

Highest win rate against AI_SIMPLE: 80%

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

<table>
<thead>
<tr>
<th></th>
<th>MiniRTS (AI_SIMPLE)</th>
<th>MiniRTS (Hit_and_Run)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>24.2 (±3.9)</td>
<td>25.9 (±0.6)</td>
</tr>
<tr>
<td>MCTS</td>
<td>73.2 (±0.6)</td>
<td>62.7 (±2.0)</td>
</tr>
</tbody>
</table>

MCTS evaluation is repeated on 1000 games, using 800 rollouts. MCTS uses complete information and perfect dynamics.
Thanks!