ELF: Extensive, Lightweight and Flexible Framework for Game Research

Yuandong Tian  Qucheng Gong  Wenling Shang  Yuxin Wu  Larry Zitnick

Facebook AI Research
Reinforcement Learning: Ideal and Reality

[R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction]
Reinforcement Learning: Ideal and Reality

Design Choices:

CPU, GPU?
Simulation, Replays
Concurrency

[State $S_t$] $\rightarrow$ [Agent] $\rightarrow$ [Reward $R_t$] $\rightarrow$ [Action $a_t$] $\rightarrow$ [Environment] $\rightarrow$ [State $S_{t+1}$]

[R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction]
**ELF**: A simple for-loop

```python
while True:
    batched_states = GameContext.Wait()
    replies = model(batched_states)
    GameContext.Steps(replies)
```
ELF Characteristics

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 (half a day for a RTS game)

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

Go

ALE

Pong

Breakout

Mini-RTS

RTS Engine

Capture the Flag

Tower Defense
Lightweight

KFPS per CPU core for Pong (Atari)

- 1 core
- 2 cores
- 4 cores
- 8 cores
- 16 cores

OpenAI Gym
Lightweight

KFPS per CPU core for Pong (Atari)
Flexibility

while True:
    batched = GameContext.Wait()
    replies = model(batched)
    GameContext.Steps(replies)
Flexibility

```python
while True:
    ...
    if batch["type"] == "actor":
        ...
    elif batch["type"] == "train":
        ...
```

Training
Flexibility

while True:
    ...
    if batch["type"] == "actor0":
        ...
    elif batch["type"] == "actor1":
        ...
    ...
Flexibility

while True:
    ...
    for i in range(n):
        if batch["type"] == "actor%d" % i:
            ...

Multi-agent
Flexibility

Monte-Carlo Tree Search

while True:
    batched = GameContext.Wait()
    replies = model(batched)
    GameContext.Steps(replies)
ELF design

Game 1

History buffer

Producer (Games in C++)
ELF design

Producer (Games in C++)
ELF design

Producer (Games in C++)

Game 1

Game 2

Game N

History buffer

History buffer

History buffer

Collector
ELF design

Producer (Games in C++)

Game 1

History buffer

Game 2

History buffer

Game N

History buffer

Collector

Batch with History info

Distributor

Reply

Consumers (Python)
ELF design

Producer (Games in C++)

Game 1
History buffer
Collector
Batch with History info
Distributor
Reply
Actor
Model
Consumers (Python)

Game 2
History buffer

Game N
History buffer
ELF design

Producer (Games in C++)

Game 1
- History buffer

Game 2
- History buffer
- ...
- ...

Game N
- History buffer

Batch with History info

Distributor

A batch for actor

Actor

Model

Optimizer

A batch for optimizer

Consumers (Python)
ELF design

Producer (Games in C++)

Game 1
- History buffer

Game 2
- History buffer

Game N
- History buffer

Collector

Batch with History info

Distributor

Reply

Actor

Model

Optimizer

Consumers (Python)

A batch for actor

A batch for optimizer

Process
Gorilla

[Nair et al, Massively Parallel Methods for Deep Reinforcement Learning, ICML 2015]
Asynchronized Advantageous Actor-Critic (A3C)

Asynchronous Methods for Deep Reinforcement Learning

[Mnih et al, Asynchronous Methods for Deep Reinforcement Learning, ICML 2016]
GA3C / BatchA2C

[Babaeizadeh et al, Reinforcement Learning through Asynchronous Advantage Actor–Critic on a GPU, ICLR 2017]
ELF: A unified framework

Off-policy training
Deep Q-learning

One-to-One
Vanilla A3C

Many-to-One
BatchA2C, GA3C
ELF: A unified framework

Off-policy training
Deep Q-learning

One-to-One
Vanilla A3C

Many-to-One
BatchA3C, GA3C

One-to-Many
Self-Play,
Monte-Carlo Tree Search
Part II. MiniRTS Training
MiniRTS: A miniature RTS engine

<table>
<thead>
<tr>
<th>Platform</th>
<th>Frame per second</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALE</td>
<td>6,000</td>
</tr>
<tr>
<td>Open AI Universe</td>
<td>60</td>
</tr>
<tr>
<td>Malmo</td>
<td>120</td>
</tr>
<tr>
<td>DeepMind Lab</td>
<td>287*/866**</td>
</tr>
<tr>
<td>VizDoom</td>
<td>7,000</td>
</tr>
<tr>
<td>TorchCraft</td>
<td>2,000</td>
</tr>
<tr>
<td>MiniRTS</td>
<td>40,000</td>
</tr>
</tbody>
</table>

* Using CPU only  ** Using CPUs and GPU
MiniRTS

**Base**
Build workers and collect resources.

**Resource**
Contains 1000 minerals.

**Barracks**
Build melee attacker and range attacker.

**Worker**
Build barracks and gather resource. 
Low speed in movement and low attack damage.

**Melee Tank**
High HP, medium movement speed, short attack range, high attack damage.

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

Using Internal Game data and Actor-Critic Models. Reward is only available once the game is over.
9 Discrete Strategic 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>
Rule-based AIs

AI_SIMPLE
Build 5 tanks and attack

AI_HIT_AND_RUN
Build 2 tanks and harass

*MiniRTS trains with a single GPU and 6 CPUs in half a day.*
Trained AI
Win rate against rule-based AI

Frame skip (how often AI makes decisions)

<table>
<thead>
<tr>
<th>Opponent Frame skip</th>
<th>AI_SIMPLE</th>
<th>AI_HIT_AND_RUN</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>

*The frameskip of learned AI is always 50*
Win rate against rule-based AI

Network Architecture

<table>
<thead>
<tr>
<th>Win Rate (10K games)</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><strong>72.2</strong></td>
<td><strong>68.4(±4.3)</strong></td>
<td><strong>65.5</strong></td>
<td><strong>63.6(±7.9)</strong></td>
</tr>
</tbody>
</table>
Effect of Multi-step Training
Curriculum Training

<table>
<thead>
<tr>
<th>Win Rate</th>
<th>Without curriculum training</th>
<th>With curriculum training</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI_SIMPLE</td>
<td>66.0 (±2.4)</td>
<td>68.4 (±4.3)</td>
</tr>
<tr>
<td>AI_HIT_AND_RUN</td>
<td>54.4 (±15.9)</td>
<td>63.6 (±7.9)</td>
</tr>
</tbody>
</table>

First $k$ decisions made by AI_SIMPLE then made by trained AI

\[ k \sim \text{Uniform}[0, K] \]

\[ K \propto \beta^{-\#\text{game\_played}} \]
## Transfer Learning

<table>
<thead>
<tr>
<th>Win Rate</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</td>
<td>51.8 (±10.6)</td>
<td>54.7 (±11.2)</td>
<td>53.2 (±8.5)</td>
</tr>
</tbody>
</table>
Monte Carlo Tree Search

<table>
<thead>
<tr>
<th>Win Rate</th>
<th>AI_SIMPLE</th>
<th>AI_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>
<tr>
<td>Trained AI</td>
<td><strong>68.4 (±4.3)</strong></td>
<td><strong>63.6 (±7.9)</strong></td>
</tr>
</tbody>
</table>

* repeat on 1000 games, each using 800 rollouts.

MCTS uses complete information and perfect dynamics
Ongoing Work

• One framework for different games.
  • DarkForest remastered: https://github.com/facebookresearch/ELF/tree/master/go

• Richer game scenarios for MiniRTS.
  • LUA scripting support
  • Multiple bases (Expand? Rush? Defending?)
  • More complicated units.

• Realistic action space
  • One command per unit

• Model-based Reinforcement Learning

• Self-Play (Trained AI versus Trained AI)
Open Source

https://github.com/facebookresearch/ELF
LUA Interface for MiniRTS

• Easy to change game dynamics
  • Don’t need to touch C++.
• Comparable speed to C++
  • 1.5x slower than compiled code.

```lua
local g_funcs = { }
function g_funcs.attack(env, cmd)
    local target = env:unit(cmd.target)
    local u = env:self()
    if target:isdead() or not u:can_see(target) then
        -- c_print("Task finished!")
        return global.CMD_COMPLETE
    end
    local att_r = u:att_r()
    local in_range = env:dist_sqr(target:p()) <= att_r * att_r
    if u:cd_expired(global.CD_ATTACK) and in_range then
        -- print("Attacking ...")
        -- Then we need to attack.
        if att_r <= 1.0 then
            env:send_cmd_melee_attack(cmd.target, u:att())
        else
            env:send_cmd_emit_bullet(cmd.target, u:att())
        end
        env:cd_start(global.CD_ATTACK)
    else
        if not in_range then
            -- print("Moving towards target ...")
            env:move_towards(target)
        end
        end
    -- print("Done with Attacking ...")
end
```
# 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()
Questions?

Tonight Poster: #96

https://github.com/facebookresearch/ELF