Ray RLlib

A scalable and unified library for reinforcement learning

https://rllib.io

Eric Liang

Riselab
UC Berkeley
Overview

- RLlib the open source project
- System challenges building a scalable RL library ("abstractions for RL")
Background: What is reinforcement learning?

Supervised Learning

Reinforcement Learning

policy

agent

environment

actions

observation + reward

Train

Errors
Growing number of RL applications

Robotics
Industrial Control
Advertising
System Optimization
Finance

RL applications

SELECT ?name FROM ?email WHERE {?person a Person; ?person foaf:firstName ?name; OPTIONAL {?person foaf:email ?email}}

High PUE  ML Control On  ML Control Off

Low PUE

Image of robotic arms and AlphaGo screen with Go board and move history.
A scalable, unified library for reinforcement learning

RL applications
- Robotics
- Industrial Control
- Advertising
- System Optimization
- Finance

RL approaches
- Single-Agent
- Multi-Agent
- Hierarchical
- Offline Batch

RLlib Training API
- PPO
- IMPALA
- QMIX
- ...
- Custom Algorithms

Distributed Execution with Ray
## Performance

**IMPALA and A2C vs A3C after 1 hour of training:**

<table>
<thead>
<tr>
<th>env</th>
<th>RLlib IMPALA 32-workers</th>
<th>RLlib A2C 5-workers</th>
<th>Mnih et al A3C 16-workers</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeamRider</td>
<td>3181</td>
<td>874</td>
<td>~1000</td>
</tr>
<tr>
<td>Breakout</td>
<td>538</td>
<td>268</td>
<td>~10</td>
</tr>
<tr>
<td>QBert</td>
<td>10850</td>
<td>1212</td>
<td>~500</td>
</tr>
<tr>
<td>SpaceInvaders</td>
<td>843</td>
<td>518</td>
<td>~300</td>
</tr>
</tbody>
</table>
User growth in 2018

Filtering GitHub and ray-dev@ issues for "rllib":
- user engagement is increasing
- couple dozen companies and research labs using RLlib!
Amazon SageMaker RL

Reinforcement learning for every developer and data scientist

Amazon SageMaker RL

End-to-end examples for classic RL and real-world RL applications

- Robotics
- Industrial Control
- HVAC
- Autonomous Vehicles
- Operations
- Finance
- Games
- NLP

RL Environments to model real-world problems

AWS Simulation Environments
- Amazon Sumerian
- AWS RoboMaker

Open Source Environments
- EnergyPlus
- RoboSchool
- PyBullet

Custom Environments
- Bring Your Own

Commercial simulators
- MATLAB & Simulink

Open AI Gym

RL Toolkits that provide RL agent algorithm implementations

- RL-Coach
- RL-Ray RLlib

SageMaker Deep Learning Frameworks
- TensorFlow
- MXNet
- PyTorch
- Chainer

Training Options
- Single Machine / Distributed
- Local/Remote simulation
- CPU/GPU Hardware

SageMaker supported

Customer BYO
Project status

• Goal: be the best library for RL applications and RL applications research
• Continuing development (https://github.com/ray-project/ray)
  • new algorithms
  • cross-cutting features (env modeling, AutoRL)
  • better performance
• Documentation at https://rllib.io
Abstractions for Distributed Reinforcement Learning
RL research scales with compute

Fig. courtesy NVidia Inc.

Fig. courtesy OpenAI

http://rllib.io
How do we leverage this hardware?

(a) Supervised Learning

(b) Reinforcement Learning

scalable abstractions for RL?
Example

```
rlib train --run=PPO --env=Pong-v0 --config={"num_workers": 1}'

rlib train --run=PPO --env=Pong-v0
   --config={"num_workers": 4, "num_gpus": 1}'

rlib train --run=PPO --env=Pong-v0
   --config={"num_workers": 256, "num_gpus": 8}'
   --redis-address=localhost:6379
```
Systems for RL today

• Many implementations (7000+ repos on GitHub!)
  – how general are they (and do they scale)?
    PPO: multiprocessing, MPI
    AlphaZero: custom systems
    Evolution Strategies: Redis
    IMPALA: Distributed TensorFlow
    A3C: shared memory, multiprocessing, TF

• Huge variety of algorithms and distributed systems used to implement, but little unification of different architectures
Challenges to unification

1. Wide range of physical execution strategies for one "algorithm"
Challenges to unification

2. Tight coupling with deep learning frameworks

Different parallelism paradigms:
- Distributed TensorFlow vs TensorFlow + MPI?
Challenges to unification

3. Large variety of algorithms with different structures

<table>
<thead>
<tr>
<th>Algorithm Family</th>
<th>Policy Evaluation</th>
<th>Replay Buffer</th>
<th>Gradient-Based Optimizer</th>
<th>Other Distributed Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQNs</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Policy Gradient</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Off-policy PG</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Model-Based Planning</td>
</tr>
<tr>
<td>Model-Based/Hybrid</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Multi-Agent</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Derivative-Free Optimization</td>
</tr>
<tr>
<td>Evolutionary Methods</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>MCTS, Derivative-Free Optimization</td>
</tr>
<tr>
<td>AlphaGo</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
We need abstractions for RL

*Good abstractions decompose RL algorithms into reusable components.*

Goals:

- Code reuse across deep learning frameworks
- Scalable execution of algorithms
- Easily compare and reproduce algorithms
Structure of RL computations

Agent

Policy: state $\rightarrow$ action

Environment

action ($a_{i+1}$)

state ($s_i$) (observation)

reward ($r_i$)
Structure of RL computations

- **Agent**
  - Policy improvement (e.g., SGD)
  - Policy evaluation (state → action)
  - Trajectory $X: s_0, (s_1, r_1), \ldots, (s_n, r_n)$

- **Environment**
  - Action ($a_{i+1}$)
  - State ($s_i$) (observation)
  - Reward ($r_i$)

Policy evaluation (state → action)
Many RL loop decompositions

Async DQN (Mnih et al; 2016)

- Param Server
- Actor-Learner
- Actor-Learner
- Actor-Learner

\[
X \leftarrow \text{rollout()}
\]
\[
d\theta \leftarrow \text{grad}(L, X)
\]
\[
sync(d\theta)
\]

Ape-X DQN (Horgan et al; 2018)

- Actor
- Actor
- Actor

\[
\text{Replay}
\]
\[
\text{Learner}
\]

\[
X \leftarrow \text{replay()}
\]
\[
\text{apply(}\text{grad}(L, X))
\]

\[
\theta \leftarrow \text{sync()}
\]
\[
\text{rollout()}
\]
Common components

Async DQN (Mnih et al; 2016)
- Param Server
- Actor-Learner
- Actor-Learner
- Actor-Learner

Policy \( \pi_\theta(o_t) \)
Trajectory postprocessor \( \rho_\theta(X) \)
Loss \( L(\theta,X) \)

Ape-X DQN (Horgan et al; 2018)
- Replay
- Learner
- Actor
- Actor
- Actor
- Actor
Common components

Async DQN (Mnih et al; 2016)

Policy $\pi_\theta(o_t)$
Trajectory postprocessor $\rho_\theta(X)$
Loss $L(\theta,X)$

Ape-X DQN (Horgan et al; 2018)

Replay
Actor
Actor
Actor
Structural differences

Async DQN (Mnih et al; 2016)
- Asynchronous optimization
- Replicated workers
- Single machine

Ape-X DQN (Horgan et al; 2018)
- Central learner
- Data queues between components
- Large replay buffers
- Scales to clusters

...and this is just one family!

→ No existing system can effectively meet all the varied demands of RL workloads.

+ Population-Based Training (Jaderberg et al; 2017)
- Nested parallel computations
- Control decisions based on intermediate results
Requirements for a new system

Goal: Capture a broad range of RL workloads with high performance and substantial code reuse

1. Support stateful computations
   - e.g., simulators, neural nets, replay buffers
   - big data frameworks, e.g., Spark, are typically stateless

2. Support asynchrony
   - difficult to express in MPI, esp. nested parallelism

3. Allow easy composition of (distributed) components
Ray System Substrate

- RLlib builds on Ray to provide higher-level RL abstractions
- Hierarchical parallel task model with stateful workers
  - flexible enough to capture a broad range of RL workloads (vs specialized sys.)

Hierarchical Task Model

http://rllib.io
Hierarchical Parallel Task Model

1. Create Python class instances in the cluster (stateful workers)
2. Schedule short-running tasks onto workers
   - Challenge: High performance: 1e6+ tasks/s, ~200us task overhead

```
Top-level worker (Python process)

Sub-worker (process)

Sub-sub worker processes
```

-Ray Cluster-

"collect experiences"
"do model-based rollouts"
"allreduce your gradients"
exchange weight shards through Ray object store

"run K steps of training"
Unifying system enables RL Abstractions

**Policy Optimizer Abstraction**

- SyncSamples
- SyncReplay
- AsyncGradients
- AsyncSamples
- MultiGPU
- ... 

**Policy Graph Abstraction**

\[ \{ \pi_\theta, \rho_\theta : L(\theta, X) \} \]

- Examples: \{Q-func, n-step, adv. calc, Q-loss\}
- \{LSTM, PG loss\}

**Hierarchical Task Model**
RLlib Abstractions in Action

Policy Optimizers

Policy Graphs

{Q-func, n-step, Q-loss}

{LSTM, adv. calc, PG loss}

+actor-critic loss, GAE

+clipped obj.

+V-trace

SyncSamples → SyncReplay → AsyncGradients → AsyncSamples → MultiGPU → ...


Summary: RLlib addresses challenges in providing scalable abstractions for reinforcement learning.

RLlib is open source and available at http://rllib.io
Thanks!