ML and RL at Any Scale with Ray

https://github.com/ray-project/ray

Edward Oakes
Peter Schafhalter
Kristian Hartikainen
Outline

- Ray Overview
- Ray Tutorial
- Break (10:30am)
- RLlib Overview
- RLlib Tutorial
- Wrap-up (12:30pm)
The Landscape Today

- Distributed systems: Training
  - PyTorch
- Distributed systems: Model Serving
  - learn
- Distributed systems: Hyperparam Tuning
  - SIGOPT
- Distributed systems: Streaming
  - kafka
- Distributed systems: Simulation
  - MPI
- Distributed systems: Featurization
  - Spark
Sharing a Common Framework

Libraries
- Training
- Model Serving
- Hyperparam Tuning
- Streaming
- Simulation
- Featurization

RAY
def f(x):
    # compute for 1s
    return result
def f(x):
    # compute for 1s
    return result

x = f(a)
def f(x):
    # compute for 1s
    return result

x = f(a)  # 1 second
def f(x):
    # compute for 1s
    return result

x = f(a)
y = f(b)
def f(x):
    # compute for 1s
    return result

x = f(a)  # 1 second
y = f(b)  # 1 second
@ray.remote
def f(x):
    # compute for 1s
    return result
@ray.remote
def f(x):
    # compute for 1s
    return result

x_id = f.remote(a)
y_id = f.remote(b)
@ray.remote
def f(x):
    # compute for 1s
    return result

x_id = f.remote(a)
y_id = f.remote(b)
ray.get([x_id, y_id])
```python
def f(x):
    # compute for 1s
    return result

x_id = f.remote(a)
y_id = f.remote(b)
ray.get([x_id, y_id])
```
```python
@ray.remote
def f(x):
    # compute for 1s
    return result

x_id = f.remote(a)
y_id = f.remote(b)
ray.get([x_id, y_id])
```
```python
@ray.remote
def f(x):
    # compute for 1s
    return result

x_id = f.remote(a)
y_id = f.remote(b)
ray.get([x_id, y_id])
```
@ray.remote
def f(x):
    # compute for 1s
    return result

x_id = f.remote(a)
y_id = f.remote(b)
ray.get([x_id, y_id])
@ray.remote

def f(x):
    # compute for 1s
    return result

x_id = f.remote(a)
y_id = f.remote(b)
ray.get([x_id, y_id])
```python
@ray.remote
def f(x):
    # compute for 1s
    return result

x_id = f.remote(a)
y_id = f.remote(b)
ray.get([x_id, y_id])
```
@ray.remote
def f(x):
    # compute for 1s
    return result

x_id = f.remote(a)
y_id = f.remote(b)
ray.get([x_id, y_id])
```python
@ray.remote
def f(x):
    # compute for 1s
    return result

x_id = f.remote(a)
y_id = f.remote(b)
ray.get([x_id, y_id])
```
@ray.remote
def f(x):
    # compute for 1s
    return result

x_id = f.remote(a)
y_id = f.remote(b)
ray.get([x_id, y_id])
```python
@ray.remote
def f(x):
    # compute for 1s
    return result

x_id = f.remote(a)
y_id = f.remote(b)
ray.get([x_id, y_id])
```
```python
@ray.remote
def f(x):
    # compute for 1s
    return result

x_id = f.remote(a)
y_id = f.remote(b)
ray.get([x_id, y_id])
```
Example: Online Learning

Featurization → Training → Model Serving

Model updated every 1 day

+ 5% CTR

Model updated every 1 hour (using state-of-the-art solution)
Example: Online Learning

Model updated every 1 day
- 5% CTR

Model updated every 1 hour (using state-of-the-art solution)
- 1% CTR

Model updated every 5 min using Ray
Ray Features

- Remote **stateless tasks** and **stateful actors**
- Scale from your laptop to a cluster **with the same API**
- Scalable, **resource-aware scheduling**
- High-performance shared-memory object store
- Single-command cluster setup and autoscaling
Distributed applications

RAY General-purpose distributed computing framework for Python (and Java)
RAY

General-purpose distributed computing framework for Python (and Java)
Distributed applications

RAY General-purpose distributed computing framework for Python (and Java)
Tutorial

● Navigate to github.com/ray-project/tutorial
  ○ Click on tutorials under “Try Ray on Google Colab”
● Ray documentation can be found at: ray.readthedocs.io/en/latest
Break
(Will resume at 11:00)
github.com/ray-project/ray
Scalable Reinforcement Learning with Ray RLlib

Kristian Hartikainen
Peter Schafhalter
Edward Oakes
Contents

1. Motivation for and the basics of Reinforcement Learning
2. Motivation for and the basics of RLlib

rllib.io:
- Documentation, examples, tutorials, etc.
Distributed applications

RAY General-purpose distributed computing framework for Python (and Java)
Background: Supervised Learning
Background: Reinforcement Learning
Background: Reinforcement Learning

Decisions (actions)

Consequences (observations, rewards)

agent

environment
Go as a Reinforcement Learning Problem

AlphaGo (Silver et al. 2016)

- **Observations:**
  - board state

- **Actions:**
  - where to place the stones

- **Rewards:**
  - 1 if win
  - 0 otherwise
Growing number of RL applications

Robotics  Industrial Control  Advertising  System Optimization  Finance  RL applications
The anatomy of an RL algorithm

- generate samples (i.e. run the policy)
- use the samples (i.e. fit model or estimate the return)
- improve the policy
Many practical RL loop decompositions

Async DQN (Mnih et al, 2016)

Ape-X DQN (Horgan et al, 2018)
Many practical RL loop decompositions

Async DQN (Mnih et al, 2016)

Ape-X DQN (Horgan et al, 2018)

Policy $\pi_\theta(o_t)$
Trajectory postprocessor $\rho_\theta(X)$
Loss $L(\theta, X)$
Many practical RL loop decompositions

Async DQN (Mnih et al, 2016)

Ape-X DQN (Horgan et al, 2018)

Policy $\pi_\theta(o_t)$

Trajectory postprocessor $\rho_\theta(X)$

Loss $L(\theta,X)$
Many practical RL loop decompositions

No existing system can effectively meet all the varied demands of RL workloads.
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 implement, compare, and reproduce algorithms
RLlib: A scalable, unified library for RL

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

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

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

Distributed Execution with Ray
General Purpose APIs

Training in Simulation

Policy Serving

Multi-Agent

Actor Network
- State
- Action
- State
- Action

Environment

RLlib Policy Server
Broad range of scalable algorithms

- High-throughput architectures
  - Distributed Prioritized Experience Replay (Ape-X)
  - Importance Weighted Actor-Learner Architecture (IMPALA)
  - Asynchronous Proximal Policy Optimization (APPO)

- Gradient-based
  - Soft Actor-Critic (SAC)
  - Advantage Actor-Critic (A2C, A3C)
  - Deep Deterministic Policy Gradients (DDPG, TD3)
  - Deep Q Networks (DQN, Rainbow, Parametric DQN)
  - Policy Gradients
  - Proximal Policy Optimization (PPO)

- Derivative-free
  - Augmented Random Search (ARS)
  - Evolution Strategies

- Multi-agent specific
  - QMIX Monotonic Value Factorisation (QMIX, VDN, IQN)

- Offline
  - Advantage Re-Weighted Imitation Learning (MARWIL)
Unified framework for scalable RL

Distributed PPO (vs. OpenMPI)

Evolution Strategies (vs. Redis-based)

Ape-X Distributed DQN, DDPG
Thanks!

http://rllib.io

Comments and feedback:

Questions!
RLlib Tutorial

- Navigate to github.com/ray-project/tutorial
  - Click on tutorial under “Try Ray on Google Colab”

Please fill out our survey! bit.ly/ray-london-2019