Ray - Scalability from a Laptop to a Cluster

Dean Wampler - EuroPython 2020
dean@anyscale.com
@deanwampler
ray.io
anyscale.com

https://anyscale.com/events
What We’ll Talk About:

- Ray demo
  - We’ll get into the mechanics of using the Ray API
- Why Ray Is Needed
- ML/AI Ray Libraries
- Ray for Microservices
- Adopting Ray and the Ray community
Demo

- Adapted from forthcoming tutorials:
  - [github.com/anyscale/academy](https://github.com/anyscale/academy)
- Contact Dean for details:
  - [dean@anyscale.com](mailto:dean@anyscale.com)
Why Ray?
Two Major Trends

Hence, there is a pressing need for robust, easy to use solutions for distributed Python.

Python growth driven by ML/AI and other data science workloads.

Model sizes and therefore compute requirements outstripping Moore’s Law.

Moore’s Law/Denard Scaling

(2x every 18 months)
The ML Landscape Today

All require distributed implementations to scale

- Featurization
  - Apache Spark
  - Hadoop

- Streaming
  - Apache Spark
  - Kafka

- Hyperparam Tuning
  - SIGOPT
  - Talos Autonomy

- Training
  - PyTorch
  - Learn

- Simulation
  - MPI
  - Akka
  - Orleans

- Model Serving
The Ray Vision: Sharing a Common Framework

Domain-specific libraries for each subsystem

Framework for distributed Python (and other languages…)

More libraries coming soon
Machine Learning with Ray-based Libraries
Ray Libraries

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

RAY

@deanwampler
Reinforcement Learning - Ray RLlib
Reinforcement Learning

<table>
<thead>
<tr>
<th>Games</th>
<th>Robotics, Autonomous Vehicles</th>
<th>Industrial Processes</th>
<th>System Optimization</th>
<th>Advertising, Recommendations</th>
<th>Finance</th>
<th>RL applications</th>
</tr>
</thead>
</table>

Decisions (actions)

Consequences (observations, rewards)

agent

environment
Reinforcement Learning

Decisions (actions)

Consequences (observations, rewards)

Agent

Environment

Games
Robotics, Autonomous Vehicles
Industrial Processes
System Optimization
Advertising, Recommendations
Finance

RL applications
Reinforcement Learning

- Decisions (actions)
- Consequences (observations, rewards)

- Games
- Robotics, Autonomous Vehicles
- Industrial Processes
- System Optimization
- Advertising, Recommendations
- Finance

RL applications
Reinforcement Learning

Decisions (actions)

Consequences (observations, rewards)

agent

environment

Games
Robotics, Autonomous Vehicles
Industrial Processes
System Optimization
Advertising, Recommendations
Finance

RL applications

Industrial Processes
System Optimization
Finance
Reinforcement Learning

Decisions (actions)

Consequences (observations, rewards)

agent

environment

Games
Robotics, Autonomous Vehicles
Industrial Processes
System Optimization
Advertising, Recommendations
Finance

RL applications

user history and context

video corpus

candidate generation

ranking

other candidate sources

video features

millions

hundreds

dozens
Reinforcement Learning

Decisions (actions)

Consequences (observations, rewards)

agent

environment

Industrial Processes
System Optimization
Advertising, Recommendations
Finance

Games
Robotics, Autonomous Vehicles
Industrial Processes
System Optimization
Advertising, Recommendations
Finance

RL applications
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
RLlib: A Scalable, Unified Library for RL

- Games
- Robotics, Autonomous Vehicles
- Industrial Processes
- System Optimization
- Advertising, Recommendations
- Finance

RL applications:

1. Application Support
   - OpenAI Gym
   - Multi-agent/Hierarchical
   - Policy Serving
   - Offline Data
2. Abstractions for RL
   - Custom Algorithms
   - RLlib Algorithms
3. Distributed Execution
   - RLlib Abstractions
   - Ray Tasks and Actors
Amazon SageMaker RL

Reinforcement learning for every developer and data scientist

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

<table>
<thead>
<tr>
<th>AWS Simulation Environments</th>
<th>Open Source Environments</th>
<th>Custom Environments</th>
<th>Commercial simulators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Sumerian</td>
<td>AWS RoboMaker</td>
<td>EnergyPlus</td>
<td>RoboSchool</td>
</tr>
</tbody>
</table>

OpenAI Gym

RL Toolkits that provide RL agent algorithm implementations

<table>
<thead>
<tr>
<th>RL-Coach</th>
<th>RL-Ray RLLib</th>
<th>OpenAI Baselines</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQN</td>
<td>APEX</td>
<td>TRPO</td>
</tr>
<tr>
<td>PPC</td>
<td>ES</td>
<td>GAIL</td>
</tr>
<tr>
<td>HER</td>
<td>IMPALA</td>
<td>...</td>
</tr>
<tr>
<td>Rainbow</td>
<td>ASC</td>
<td>...</td>
</tr>
</tbody>
</table>

SageMaker Deep Learning Frameworks

| TensorFlow | MxNet | PyTorch | Chainer |

Training Options

| Single Machine / Distributed | Local/Remote simulation | CPU/GPU Hardware |

SageMaker supported ✔️

Customer BYO ✗
Now in Azure

Reinforcement learning (preview) with Azure Machine Learning

05/05/2020 • 11 minutes to read • ![Icon](https://learn.microsoft.com/en-us/azure/machine-learning/docs/)


Note

Azure Machine Learning Reinforcement Learning is currently a preview feature. Only Ray and RLib frameworks are supported at this time.

In this article, you learn how to train a reinforcement learning (RL) agent to play the video game Pong. You will use the open-source Python library Ray RLlib with Azure Machine Learning to manage the complexity of distributed RL jobs.

In this article you will learn how to:
Diverse Compute Requirements Motivated Creation of Ray!

And repeated play, over and over again, to train for achieving the best reward

Simulator (game engine, robot sim, factory floor sim…)

Complex agent?

Neural network “stuff”

Decisions (actions)

Consequences (observations, rewards)

Environment

L0 (Input) 512x512

L1 256x256

L2 128x128

L3 64x64

L4 32x32

F5

F6 (Output)

Convolution

Fully connected

@deanwampler
Hyperparameter Tuning - Ray Tune

Featurization | Streaming | Hyperparam Tuning | Training | Simulation | Model Serving

RAY

tune.io

@deanwampler
Nontrivial Example - Neural Networks

How many layers? What kinds of layers?

Every number shown is a hyperparameter!

Stride of 4

Max pooling

Max pooling

Max pooling

Max pooling

Dense

Dense

Dense

Dense

1000
```python
tune.run(PytorchTrainable,
    config={
        "model_creator": PretrainBERT,
        "data_creator": create_data_loader,
        "use_gpu": True,
        "num_replicas": 8,
        "lr": tune.uniform(0.001, 0.1)
    },
    num_samples=100,
    search_alg=BayesianOptimization()
)
```
Tune is Built with Deep Learning as a Priority

Resource Aware Scheduling

Simple API for new algorithms

```python
class TrialScheduler:
    def on_result(self, trial, result): ...
    def choose_trial_to_run(self): ...
```

Seamless Distributed Execution

Framework Agnostic

tune.io
What about Ray for Microservices?
What Are Microservices?

- They partition the domain
- Conway's Law - Embraced
- Separate responsibilities
- Separate management
What Are Microservices?

- They partition the domain
- Conway's Law - Embraced
- Separate responsibilities
- Separate management

What's most interesting for our purposes today.
Separate Management

- Each team manages its own instances
- Each microservice has a different number of instances for scalability and resiliency
- But they have to be managed explicitly
Management - Simplified

- With Ray, you have one “logical” instance to manage and Ray does the cluster-wide scaling for you.
What about Kubernetes (and others…)?

- Ray scaling is very fine grained.
- It operates within the “nodes” of coarse-grained managers
- Containers, pods, VMs, or physical machines
Adopting Ray and the Ray community
If you’re already using...

- joblib
- multiprocessing.Pool
- Use Ray’s implementations
  - Drop-in replacements
  - Change import statements
  - Break the one-node limitation!

For example, from this:

```python
from multiprocessing.pool import Pool
```

To this:

```python
from ray.util.multiprocessing.pool import Pool
```

... And Ray is integrated with asyncio

See these blog posts:
Ray Community and Resources

- [ray.io](http://ray.io)
- Tutorials (free): [anyscale.com/academy](http://anyscale.com/academy)
- Need help?
  - Ray Slack: [ray-distributed.slack.com](http://ray-distributed.slack.com)
  - [ray-dev](http://ray-dev) Google group
Thanks for Listening

ray.io
anyscale.com - We’re Hiring!
dean@anyscale.com
@deanwampler

https://anyscale.com/events