Reinforcement Learning

Jennifer Davis, Ph.D.
Andrea Lowe, Ph.D.
Domino Data Lab

WORKSHOP DEEP LEARNING

Jennifer Dawn Davis, PhD
Staff Field Data Scientist
Domino Data Labs

Andrea Lowe, PhD
Training and Enablement Engineer
Domino Data Labs

Practical Reinforcement Learning for Data Scientists

ODSC WEST

San Francisco | Nov 16-18

Introduction

- Reinforcement learning in a nutshell
- Reinforcement learning techniques are used in image analysis and generation, text generation, game-playing and many other applications
- Reinforcement learning models the aspect of human learning from interaction with the environment

What Reinforcement Learning is and isn’t

- Mapping situations to actions to maximize a numerical reward
- Distinguishing features
  - Trial-and-error search
  - Delayed Reward
- Differs from supervised learning
- Differs from unsupervised learning because it is not searching for hidden structure in collections of unlabeled data.

Research and Practical Uses

- Computational psychology
- Fraud detection
- Understanding market volatility
- Cancer detection on electronic radiographs
- Recommendation engines

Elements of Reinforcement Learning

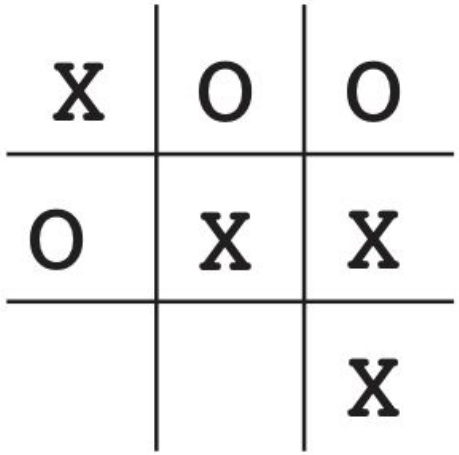
- Policy
- Reward Signal
- Value Function
- Model of the environment

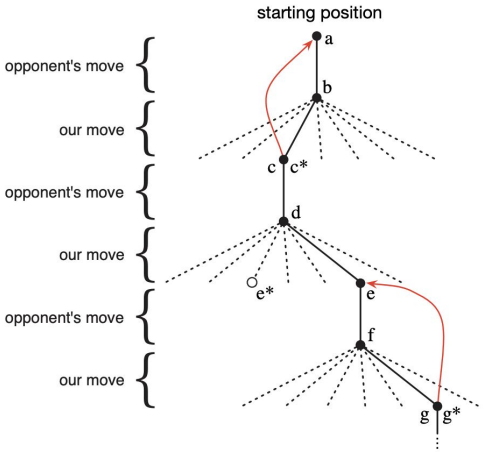
Theory by example

- Examples
  - A gazelle calf struggles to its feet minutes after being born. Half an hour later it runs at 20 miles per hour.
- Reinforcement learning involves interaction, active decision making and uncertainty about the environment
- A correct or incorrect choice may have delayed consequences, and affect future opportunities

Reinforcement Learning Algorithms

- Tic-tac-toe can be modeled as a reinforcement learning problem
- Example of temporal-difference learning method





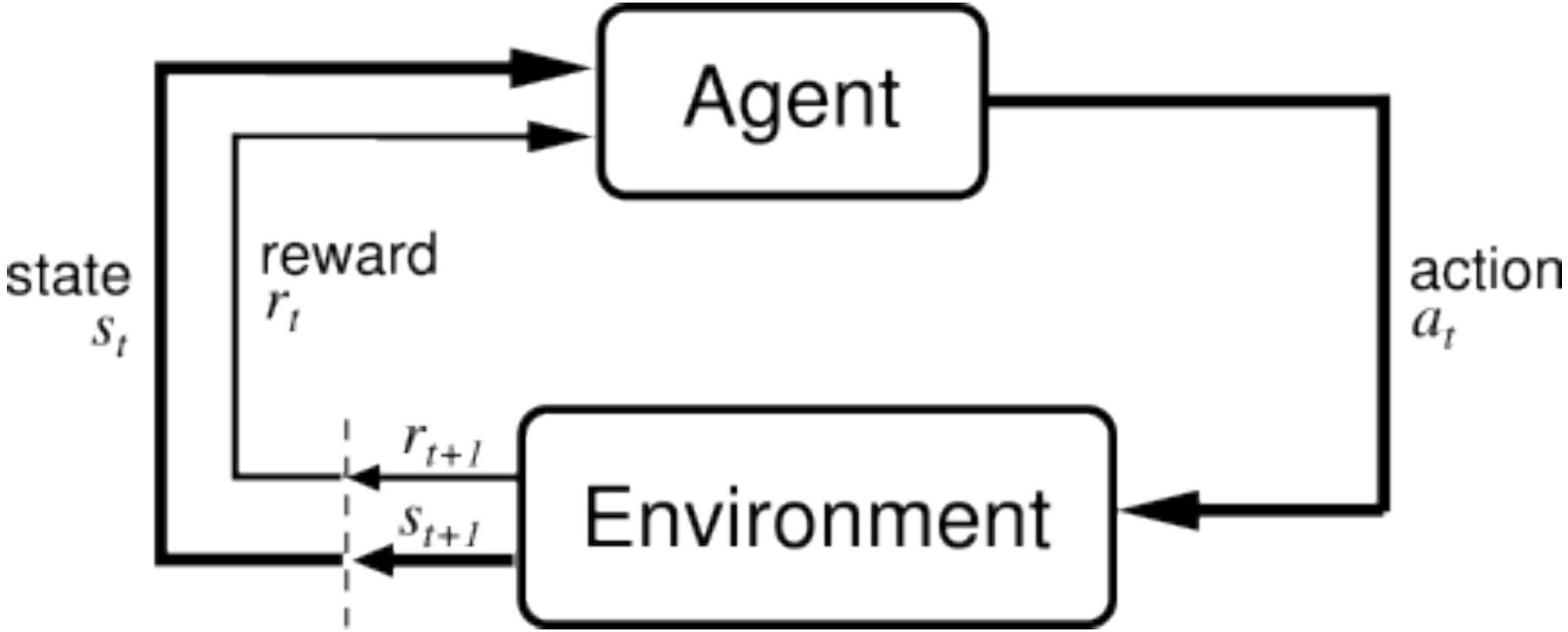
Example of other Common Reinforcement Learning Algorithms

- Q-learning
- Deep Q Learning
- Markov Decision Process and Finite Markov Decision Processes
- Multi-armed Bandits
- Dynamic Programming
- Monte Carlo Methods

Q-Learning Example

THE MOUSE AND CHEESE EXAMPLE

- A mouse (the agent) moves through a maze to find its cheese
- From the starting point to the end point the environment has obstacles
- The mouse much find the ending point along a path without obstacles
- The policy says the mouse must reach the cheese in the shortest amount of time needed to
- The mouse keeps a mental ‘ledger’ of what did and didn't work until he is able to get the cheese as fast as possible.
- The q-ledger consists of an action, and a relative reward
- The highest reward is getting the cheese



Reinforcement Learning

- Reinforcement learning is an iterative process
- The agent (learner) acts within an environment
- Each action comes with a reward
- The state of the agent is saved each time an action is taken

Basics of Q-Learning

- A **Q-value function (Q)** shows us how good a **certain action** is, given a state, for an agent following a policy. The optimal Q-value function (Q*) gives us maximum return achievable from a given state-action pair by any policy.
- "Q" refers to the function that the algorithm computes – the expected rewards for an action taken in a given state
- Q learning is one of the most intuitive RL algorithms and is similar to how we learn as humans

Types of Q-Learning Algorithms

- Q-learning algorithms all use the q-value as their operator
- Some algorithms include:
  - q-learning (canonical algorithm)
  - SARSA (state-action-reward-state-action)
  - Q-learning – Lambda
  - DQN – Deep Q Learning (what we will study today)
  - DDPG – Deep Deterministic Policy Gradient

Deep Q-Learning Applications

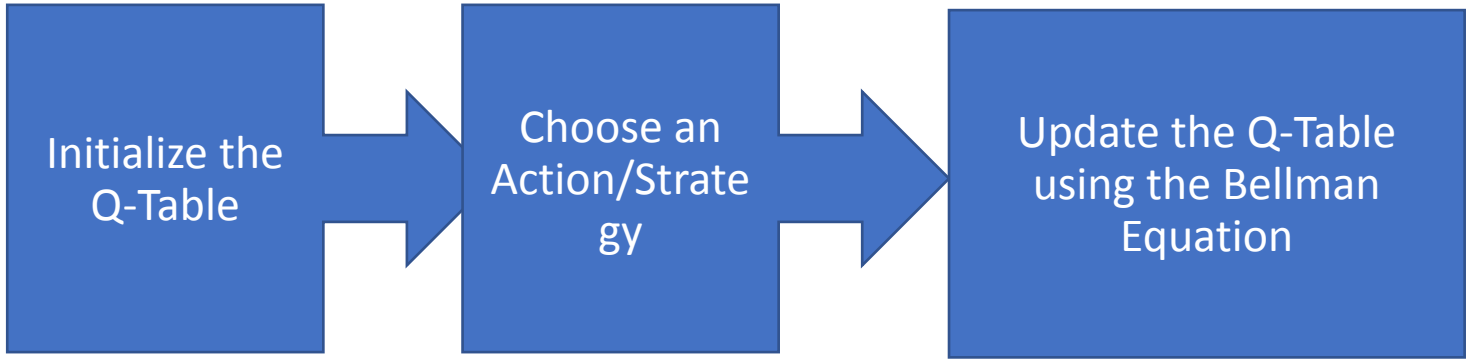
- Deep Q-learning Networks (DQN) uses neural networks to find the agent's next best action in a given environment
- Because of this the algorithm performs better than simple q-learning in response to complex or noisy environments

Tasks of Deep Q-Learning

- A neural network maps input states to action, q-value pairs
- First initialize the q-learning algorithm by
  - Initializing network weights
  - Choosing an action for a particular environment or state
  - Update the network weights using the Bellman Equation
  - The action with the largest q-value is the best action given a particular state

What goes into A Q-Learning Algorithm?

- Each q-table consists of an environment/state + action pair, and a Q-Value
- The State-Action with the highest q-value is considered the pair with the highest reward



State	Action	Q-Value
S1	A1	5
S2	A2	6
S3	A0	1
S5	A4	10

Bellman Equation

- Commonly used to calculate the reward for a particular action or strategy
- Used in q-learning as well as other types of reinforcement learning (such as decision processes)

Bellman Equation

$$Q(S_t, A_t) = (1 - \alpha)\, Q(S_t, A_t) + \alpha * (R_t + \lambda * max_a\, Q(S_{t+1}, a))$$

Figure 4: The Bellman Equation describes how to update our Q-table (Image by Author)

S = the State or Observation

A = the Action the agent takes

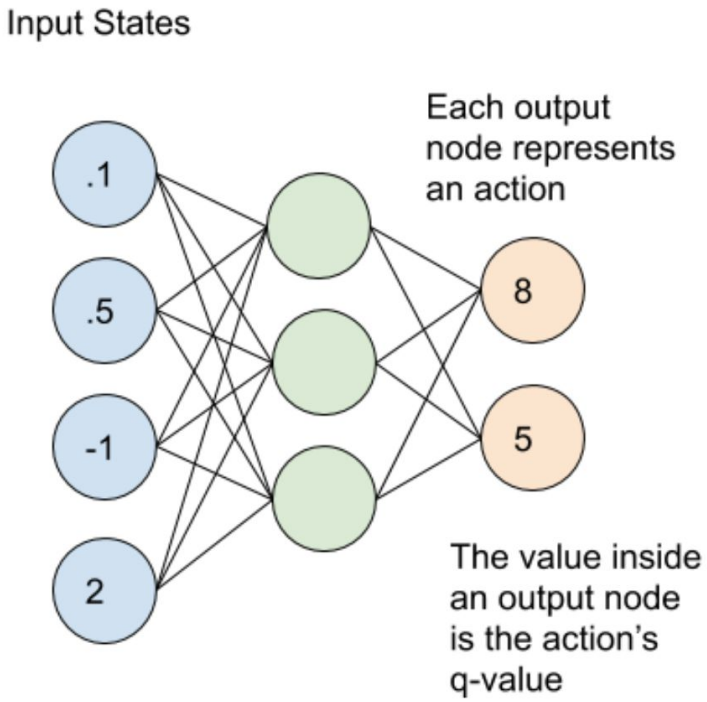
R = the Reward from taking an Action

t = the time step

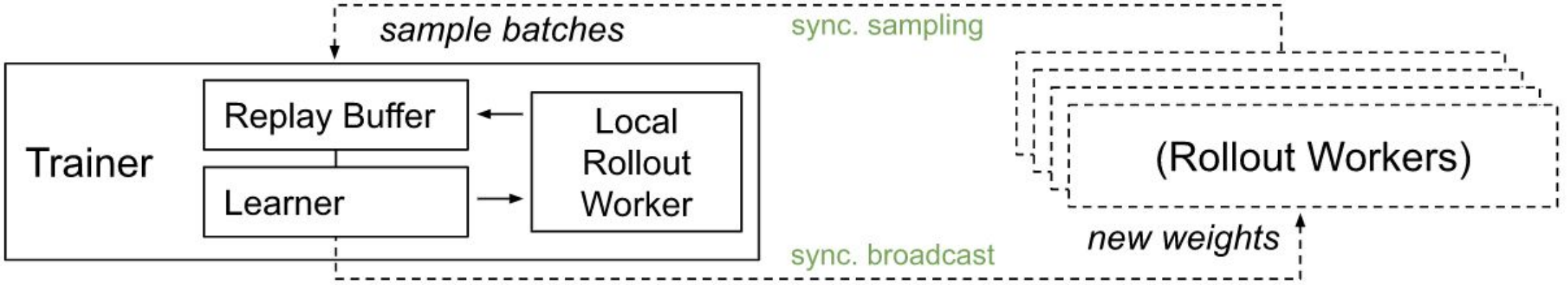
α = the Learning Rate

λ = the discount factor which causes rewards to lose their value over time so more immediate rewards are valued more highly

USING NEURAL NETWORKS FOR Q-LEARNING



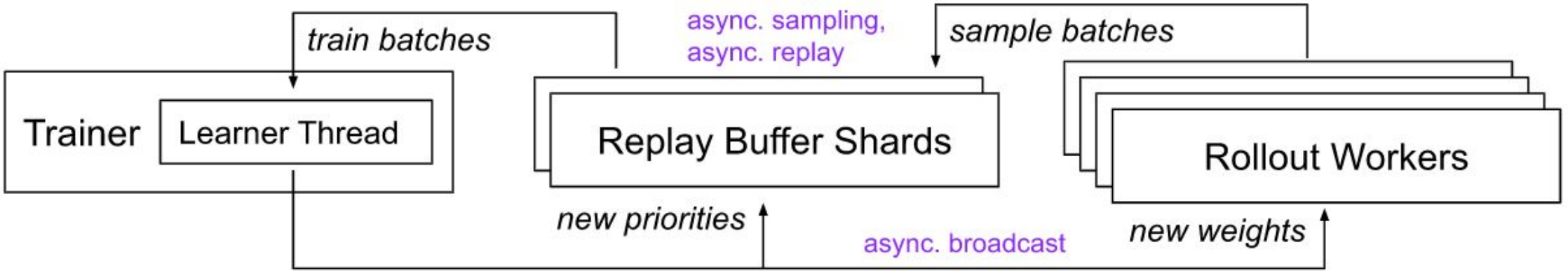
1) Initialize network weights
2) Choose an action
3) Update network weights using the Bellman Equation
4) The state-action pair with the largest q-value is the best action in a particular environment



DQN architecture

DQN: Algorithm Architecture

- This diagram shows how Ray distributed the DQN algorithm on its workers
- The trainer tracks the replay buffer and the learner
- Rollout workers do the calculations, adding new weights to the neural network for each q value



Ape-X architecture

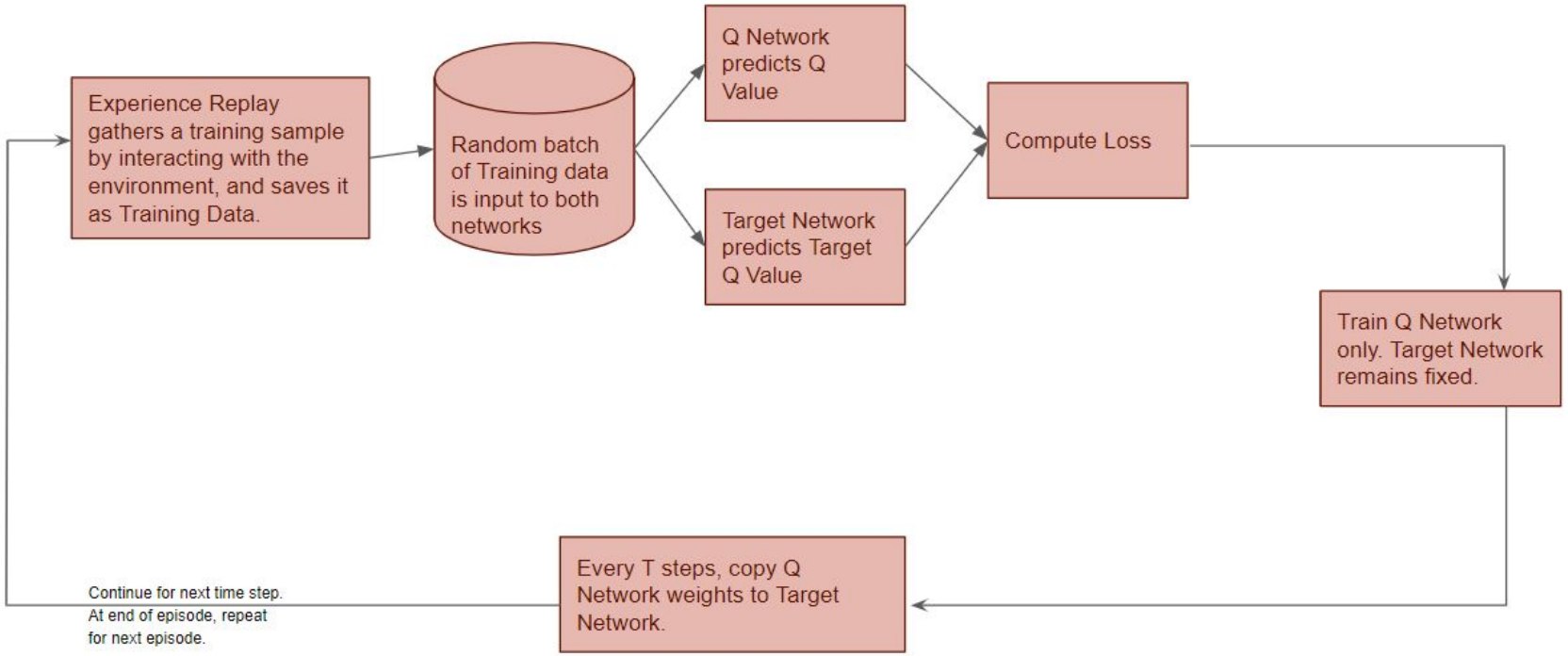
DQN: APEX Algorithm Architecture

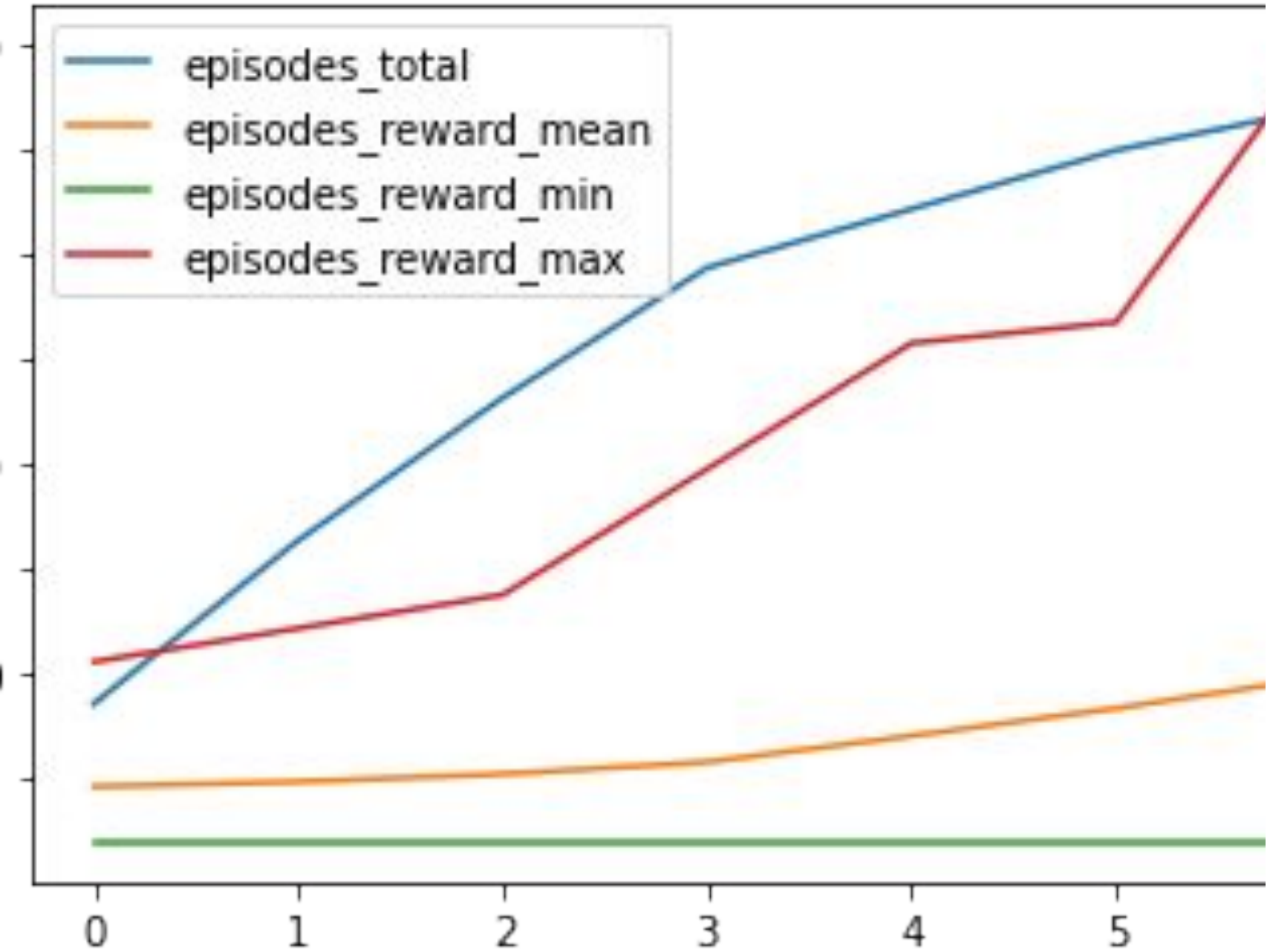
- The APEX algorithm was designed to speed up the training of DQN networks
- In Ray the replay buffer is sharded so that RL can be distributed over multiple workers
- Asynchronous sampling occurs
- This differs than regular DQN on Ray which does not include sharding

Deep Q-Learning Output Interpretation

- One difference is that we are looking for the best q-value in each action-space pair
- The q-value is contained in each neural node
- Additional calculations are required as the neural network is a function approximator
- Linear network are fine, the neural network does not have to be complex
- Neural networks are used to predict Q-values, a target Q-value, loss is computed, and a Q-network is trained, while the target network remains fixed
- The weights output from the q-network training are the q-values
- These q-values are fed a gain into the system to calculate the next set of q-values until a maximal reward is obtained

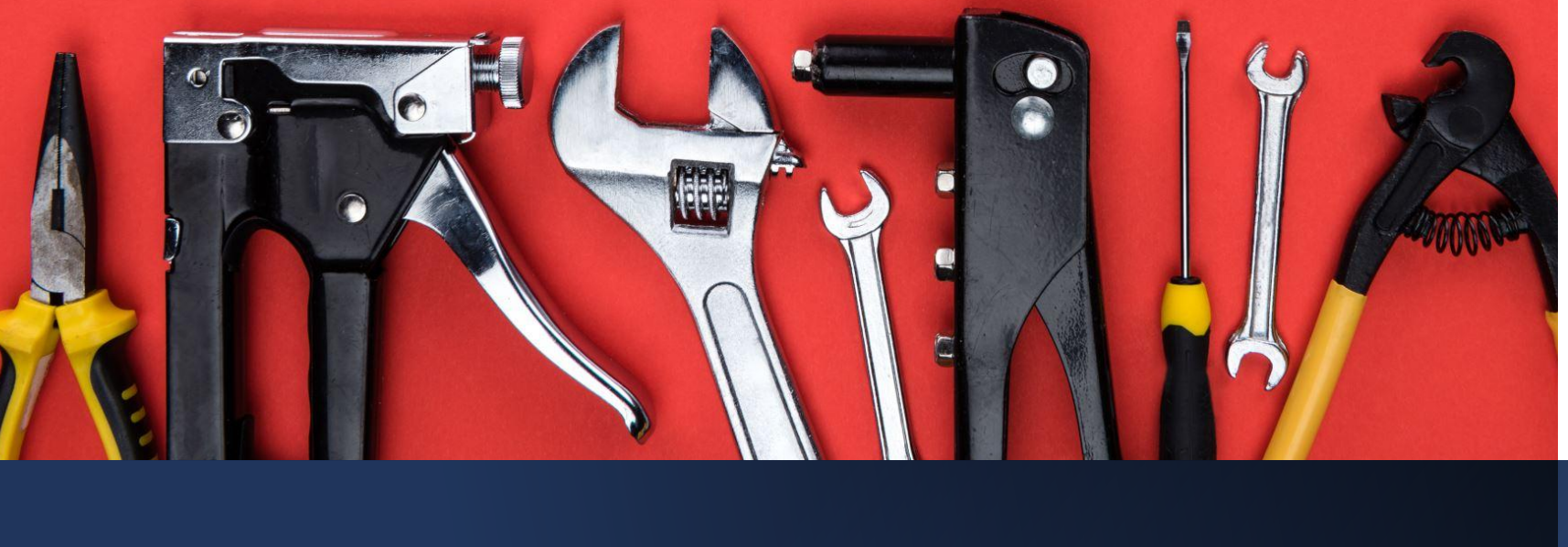
Diagram of DQN process





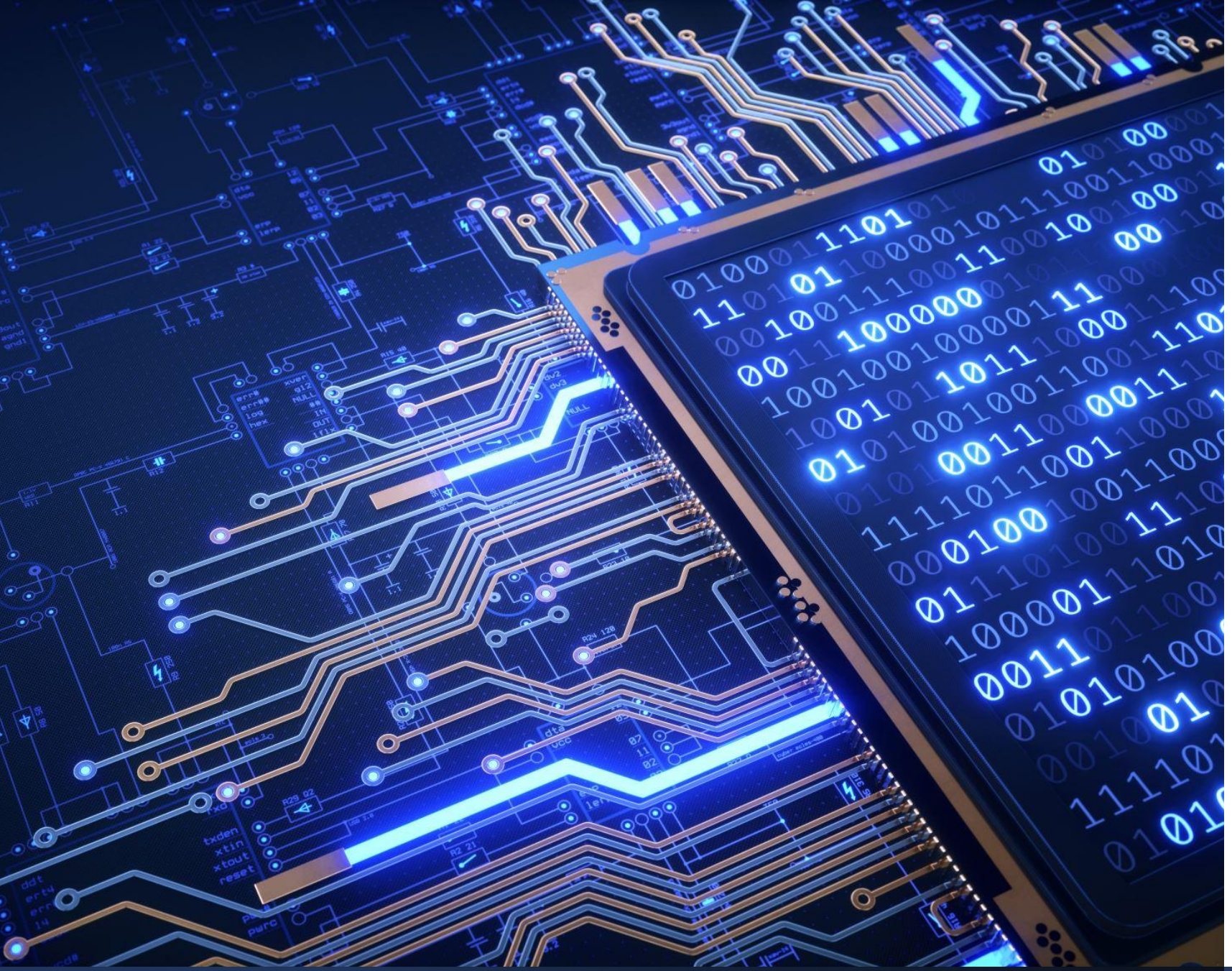
Result Example

- Maximal reward is highest reward at any point for a state action pair
- In this example the reward seems to level off and then increases again
- More episodes could be performed to find the highest reward



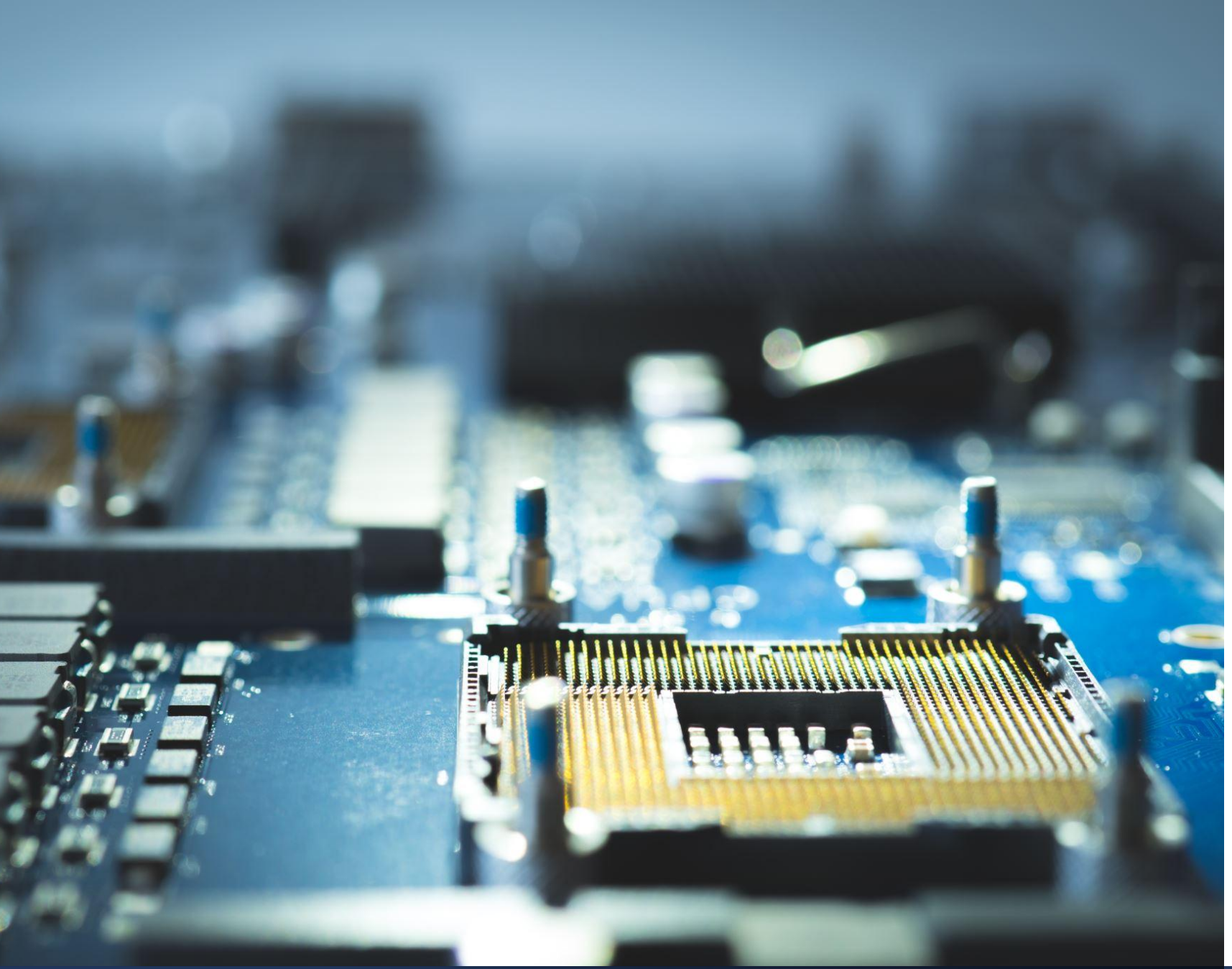
Tools we use in this Workshop

- Ray
- Pytorch
- TensorTrade
- Installing:
  - Locally
  - Using Domino Cluster



What is Parallel Computing?

- Type of computing in which many calculations or processes are carried out simultaneously
- Limits include:
  - Available parallelism
  - Load balancing – some processors work while others wait due to insufficient resources or unequal task size
  - Managing communication
- Pros
  - Faster speeds for computations
  - Enabling for big data and machine learning



What is distributed computing?

- When multiple computers are used, tasks can be done in parallel on multiple computers or work can be distributed among computer unevenly
- First rule: Don’t distribute the system until you have an observable reason to
- For reinforcement learning, data can be massive, and so a distributed system can be very useful
- For our course simple parallelism is sufficient

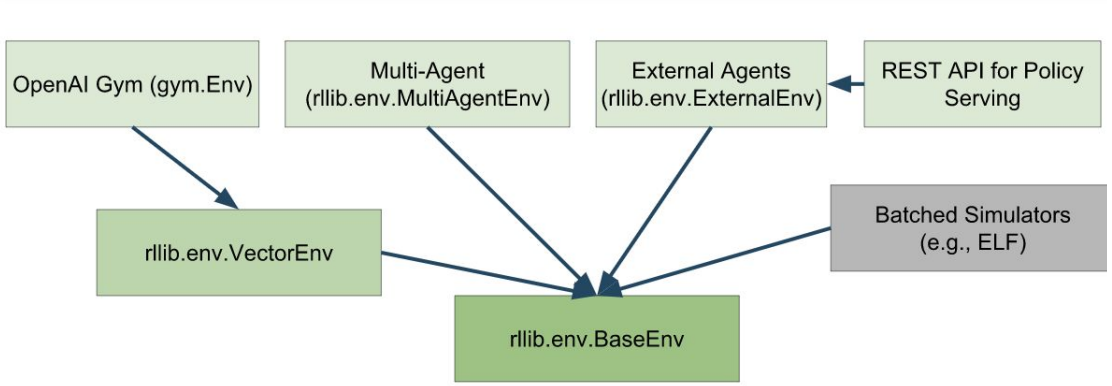
What is Pytorch?

- Pytorch is a python library which optimizes machine learning and deep learning while enabling parallel and distributed computing frameworks
- Pytorch allows a significant level of granularity when building deep learning pipelines

What is Ray?

- Universal API for building distributed applications
- Simple primitives for building and running applications
- Can run on single machine mode or with a cluster
- Has a large ecosystem of libraries and tools for reinforcement learning, and deep learning
- Is compatible with Pytorch and TensorFlow

Ray's Libraries - RLLib



- Reinforcement Learning in the past typically took many hours to train even a basic model
- Ray has sped up this process with parallel and distributed computing options
- In this tutorial we'll use Ray's RLLib, a reinforcement learning library

Ray’s Libraries – Ray Tune

- Tune is a Ray library with scalable hyper-parameter tuning
- We’ll use Tune to find the best configuration for a person walking down a hallway, i.e. the number of steps to take

What Does the Tensor-Trade Library Do?

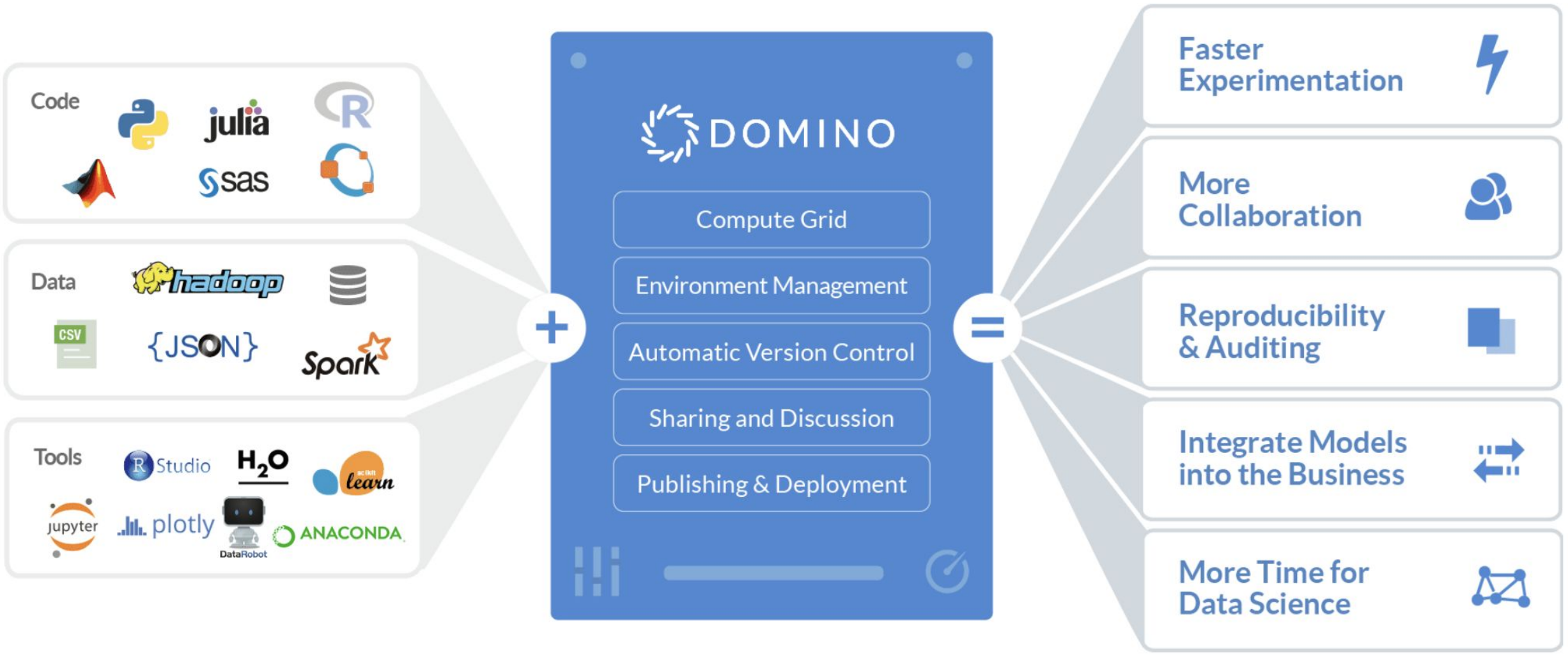
- Tensor Trade is a library built with TensorFlow. and Ray
- The goal of the library is to make bitcoin or financial stock trading more successful using reinforcement learning
- The library happens to be helpful for our purposes of looking at DQN for financial trading applications

Tutorial: Getting Started

- Download the libraries on your computer or use Domino to try your code – we will walk through how to use Domino
- Note the WI-FI may be spotty so try downloading and setting up the environment on your laptop
- The following libraries can be installed using the pip command: Ray, Pytorch, TensorTrade
- Suggest to use a jupyter notebook as your IDE as we have provided

Ray On-Demand On Domino Data Lab

Domino Data Lab

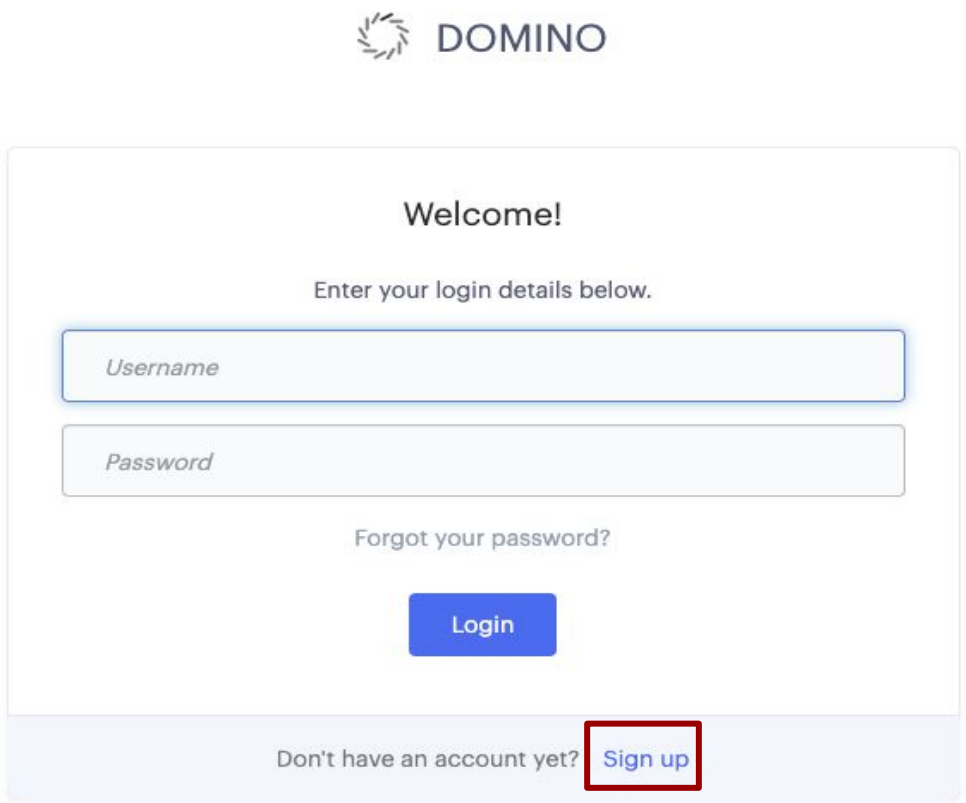


Ray On-Demand On Domino Data Lab

Sign up for account in the temporary Domino training instance:

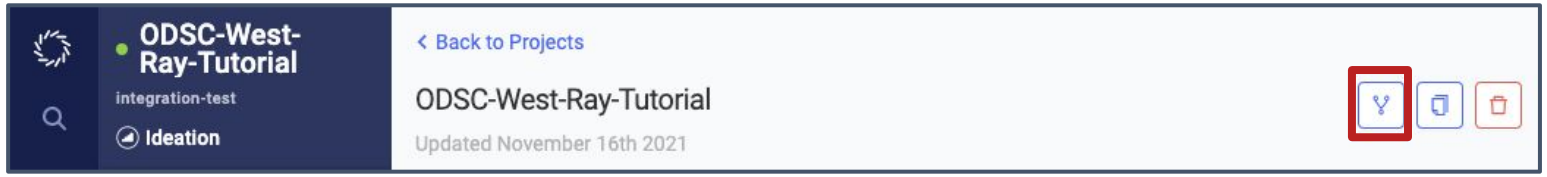
https://tutorial.workshop.domino.tech/

Note that this deployment will be taken down at the end of training



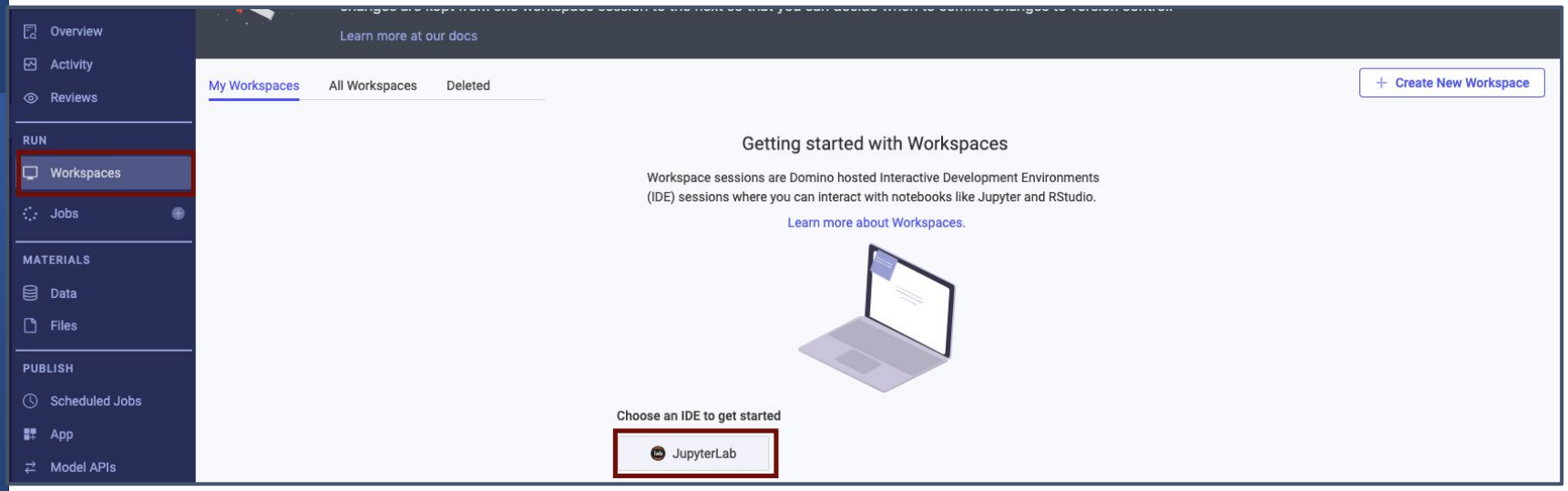
Ray On-Demand On Domino Data Lab

1. Sign into temporary training Domino instance
2. Open https://tutorial.workshop.domino.tech/u/integration-test/ODSC-West-Ray-Tutorial/overview
3. Fork the project from the Overview page and name it ‘<your name>-Training’



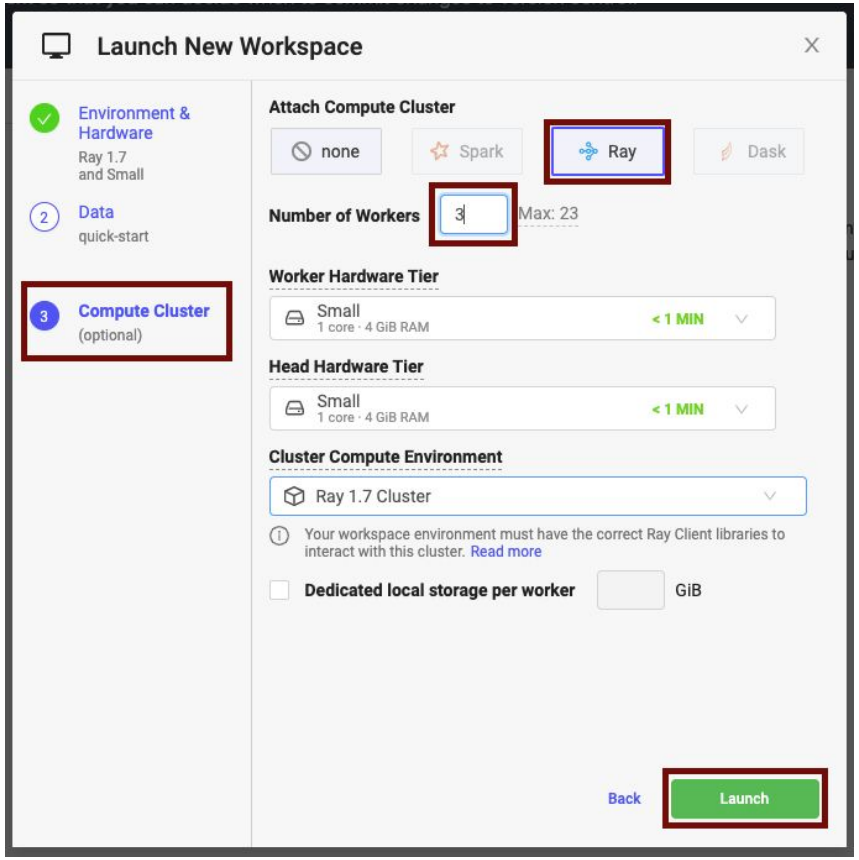
Ray On-Demand On Domino Data Lab

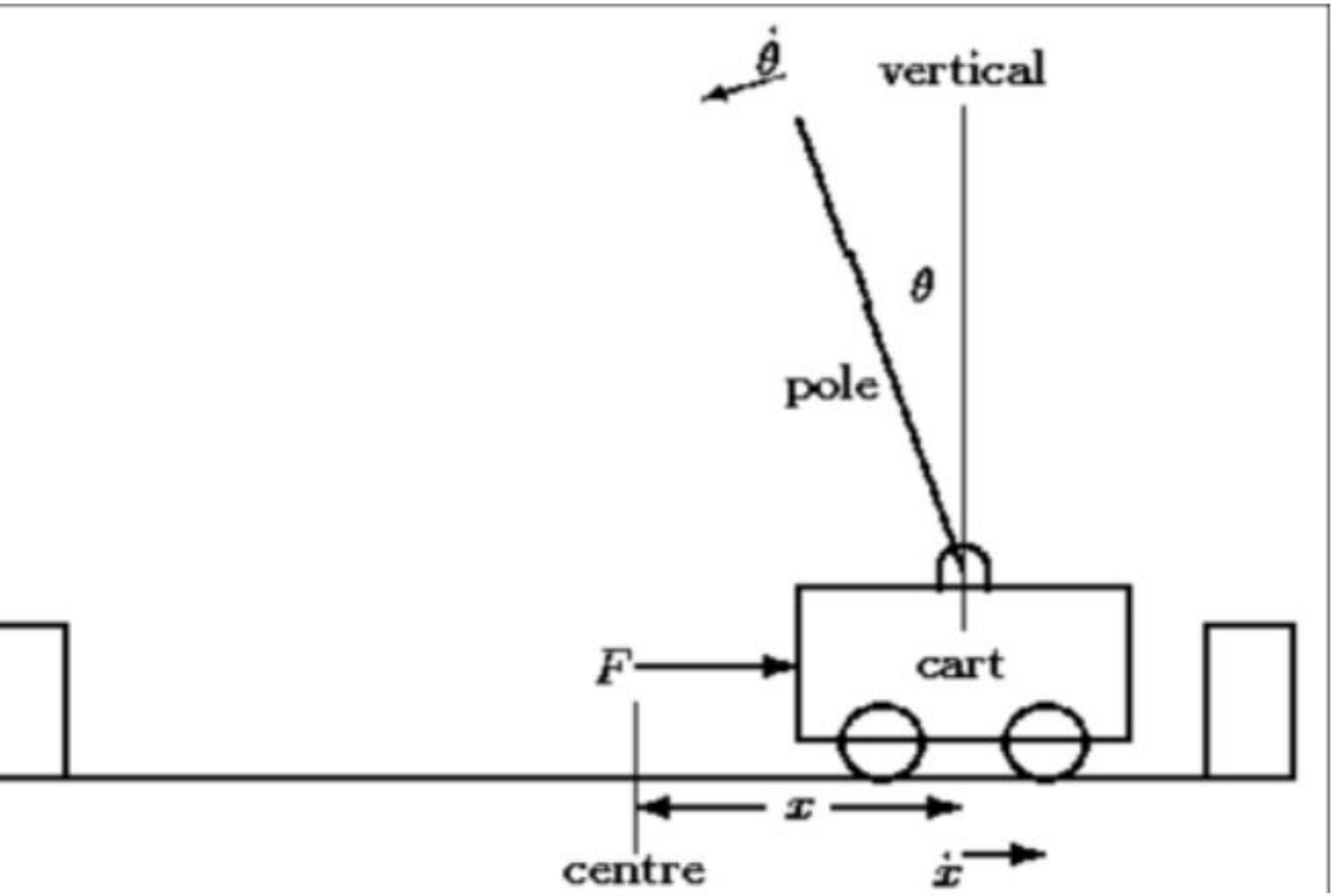
1. Navigate to the Workspaces tab
2. Click on Jupyterlab



Ray On-Demand On Domino Data Lab

1. Select ‘Compute Cluster’
2. Select ‘Ray’
3. Change the ‘Number of Workers’ to 3
4. Click the ‘Launch’ button





The Cart-Pole Example

- Canonical RL Problem
- A cart has a pole attached to it that can swing like a pendulum
- The cart moves back and forth along a frictionless surface
- The goal is to keep the pendulum from falling as the cart moves along the surface
- RL algorithm will learn how far back and forth it can move to ensure the pole does not fall

The Walking Through a Corridor Example

- Imagine a corridor of a fixed length
- The goal is to walk down the corridor with the minimum number of steps in the minimum amount of time
- What stride and pace should be used?
- The reward is the amount of time taken to walk through the corridor

Tensor Trade Exercises

- We use Deep Q Learning using Ray.
  - Training a deep q-network using Ray's Tune Library to optimize learning rate
  - Using Rllib to train a Deep Q Learning Stock Trading Agent and examining the results

Wrapping Up – What we learned today

- Distributed computing libraries: Ray and Pytorch
- Basics of Reinforcement Learning
- Examples of :
  - The Cart-Pole problem
  - Walking down a corridor
  - Stock Trading of US Dollars and Bitcoin

Appendix

RAY OVERVIEW

Ray.io provides a set of core low-level primitives (Ray Core) as well as pre-packaged libraries that take advantage of these primitives to enable solving powerful machine learning problems.

The following libraries come packaged with Ray:

- Tune: Scalable Hyperparameter Tuning
- RaySGD: Distributed Training Wrappers
- RLlib: Scalable Reinforcement Learning
- Ray Serve: Scalable and Programmable Serving

RAY ON DEMAND IN DOMINO: USE CASES

Distributed Multi-Node Training

Take existing PyTorch and Tensorflow models and scale them across multiple machines to dramatically reduce training times

Ray is suitable for both distributed CPU and GPU training

Hyperparameter Optimization

Launch a distributed hyperparameter sweep with just a few lines of code.

Take advantage of a large number of advanced parameter search algorithms.

Reinforcement Learning

Take advantage of a number of built-in reinforcement learning algorithms, along with a general framework for developing your own.

RAY ENVIRONMENT SETUP

When using on-demand Ray in Domino you will have two separate environments:

1. One for the workspace/job
    - Will contain configurations for `RAY_VERSION` and any data connections, see dockerfile instructions here
2. Ray cluster environment
    - Used on the cluster nodes, developed with a different base image - more info found here

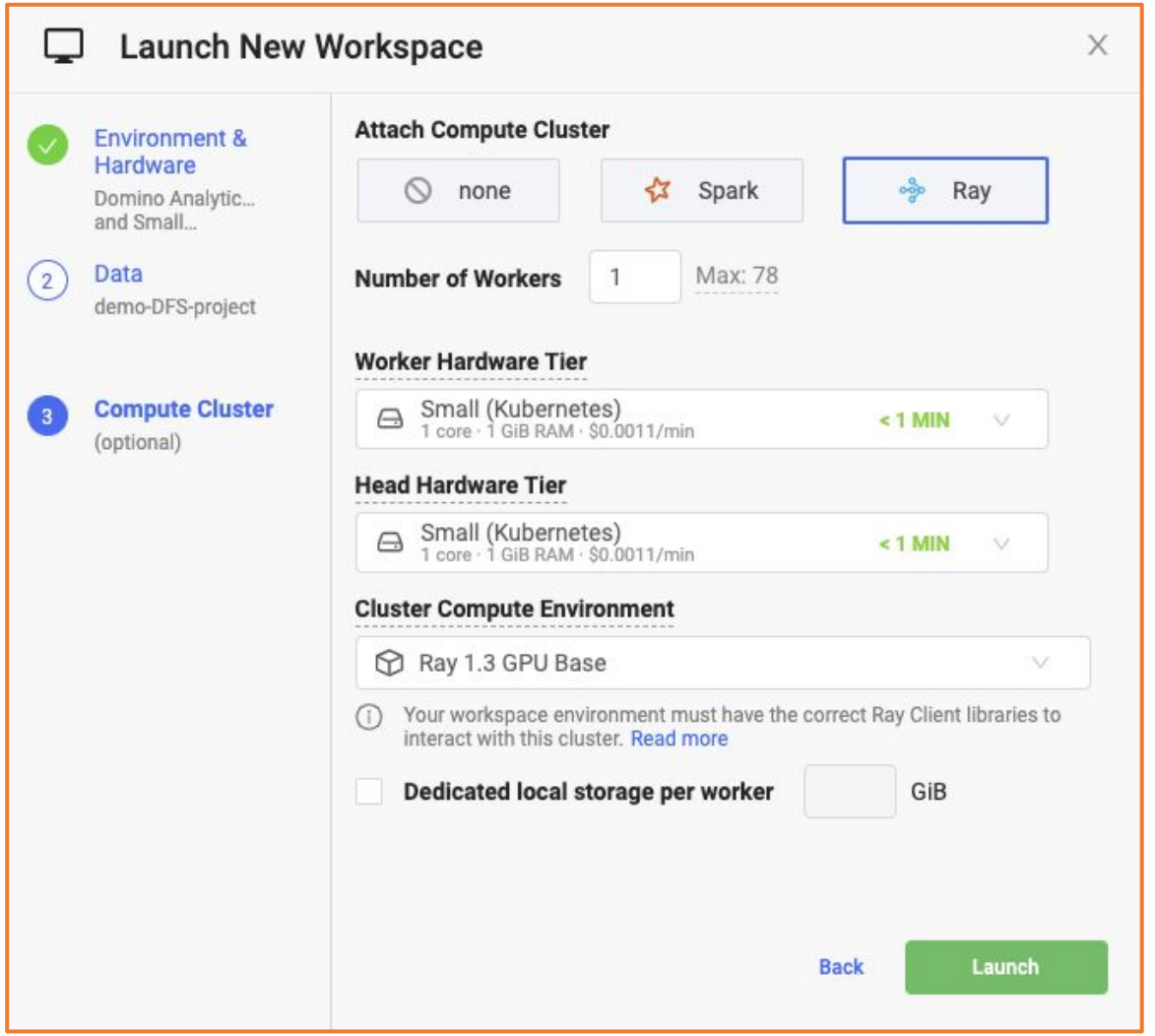
RAY IN DOMINO: CLUSTER SETTINGS

Number of Workers - Ray nodes in cluster

Worker Hardware Tier - amount of compute resources (CPU, GPU, and memory) that will be made available to each Ray node

Head Hardware Tier - responsible for coordinating the Ray workers, and does not need a significant amount of CPU resources. It will host the Ray Global Control Store, and the amount of required memory will depend on the complexity of your application

Dedicated local storage per executor - amount of dedicated storage that will be available to each worker. This will be automatically mounted to /tmp and de-provisioned when the cluster is shut down.



CONNECTING TO THE CLUSTER

Domino sets up key environment variables

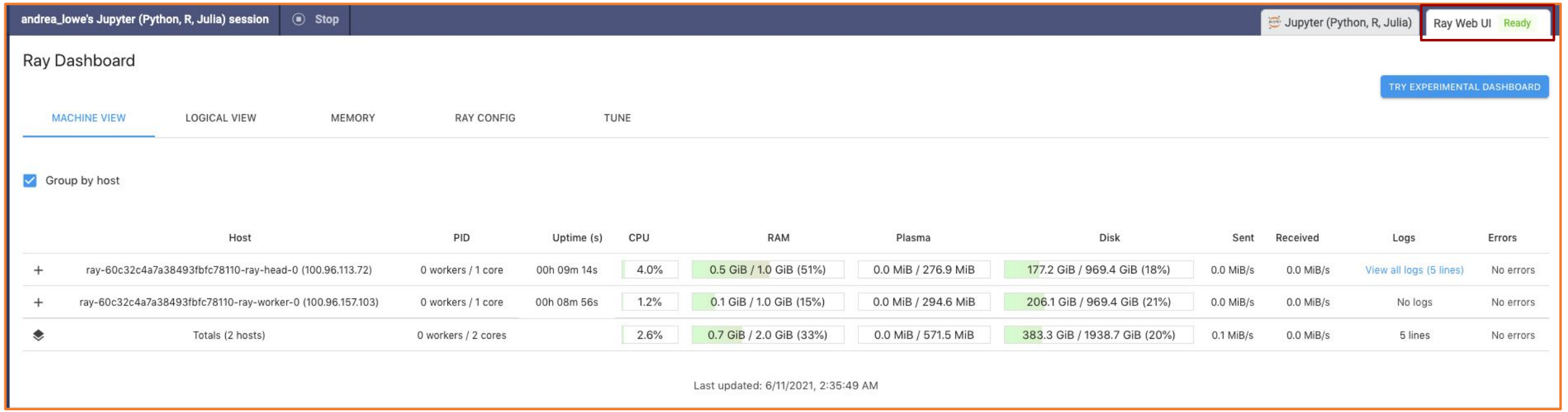
Use Ray Client instead of ray.init()

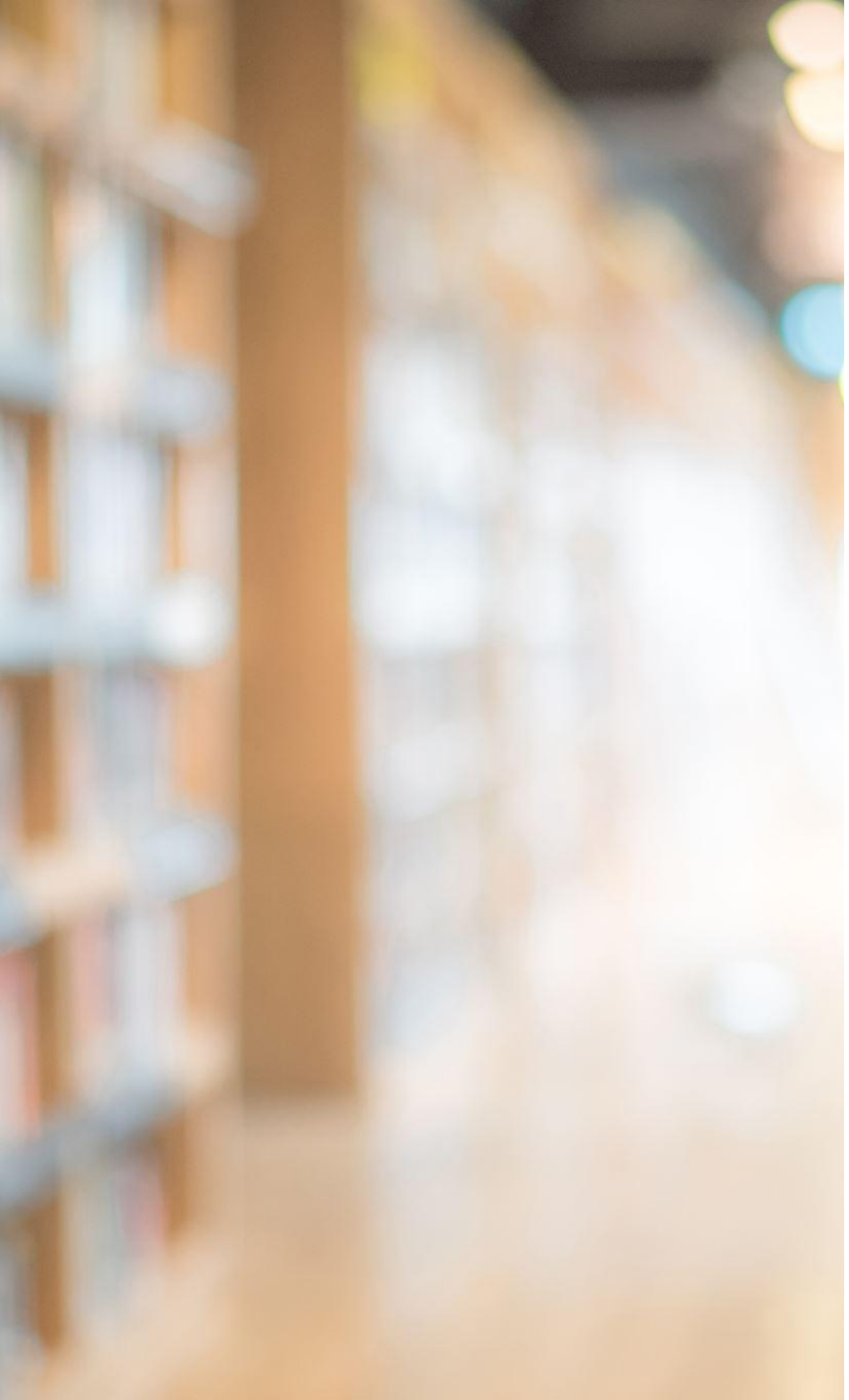
```
import ray
import ray.util
import os

if ray.is_initialized() == False:
   service_host = os.environ["RAY_HEAD_SERVICE_HOST"]
   service_port = os.environ["RAY_HEAD_SERVICE_PORT"]
    ray.util.connect(f"{service_host}:{service_port}")
```

ACCESSING THE RAY UI

The Ray dashboard is available as a dedicated tab within your workspace





Contents

- Introduction
- What Reinforcement Learning Is and Isn’t
- Theory by Example
- Elements of Reinforcement Learning
- Reinforcement Learning Algorithms
- Research

Works cited

- Machine Learning: The Art and Science of Algorithms that Make Sense of Data. Peter Flach, Cambridge University Press, 2012
- Wikipedia articles on reinforcement learning
- Data Mining: Practical Machine Learning Tools and Techniques, Third Edition. Witten, Frank and Hall, Elsevier, 2011
- Reinforcement Learning, 2nd Edition: An Introduction. Richard S. Sutton, MIT Press, 2020

Works Cited

- Heaton, Jeff. Artificial Intelligence for Humans, Volume 1: Fundamental Algorithms . Kindle Edition.
- Neural Network Concepts, “7 Types of Neural Network Activation Functions: How to Choose? Accessed January 14, 2020.
- Heaton, Jeff. Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks (p. 175). Heaton Research, Inc..