Reinforcement learning in Tensorflow

Krzysztof Suwada

Agenda

- Introduction
- Few examples
- Very very brief introduction to RL
- What is TF-Agents
- Example 1: Cart Pole
- Example 2: Atari
- Replay buffers

Introduction

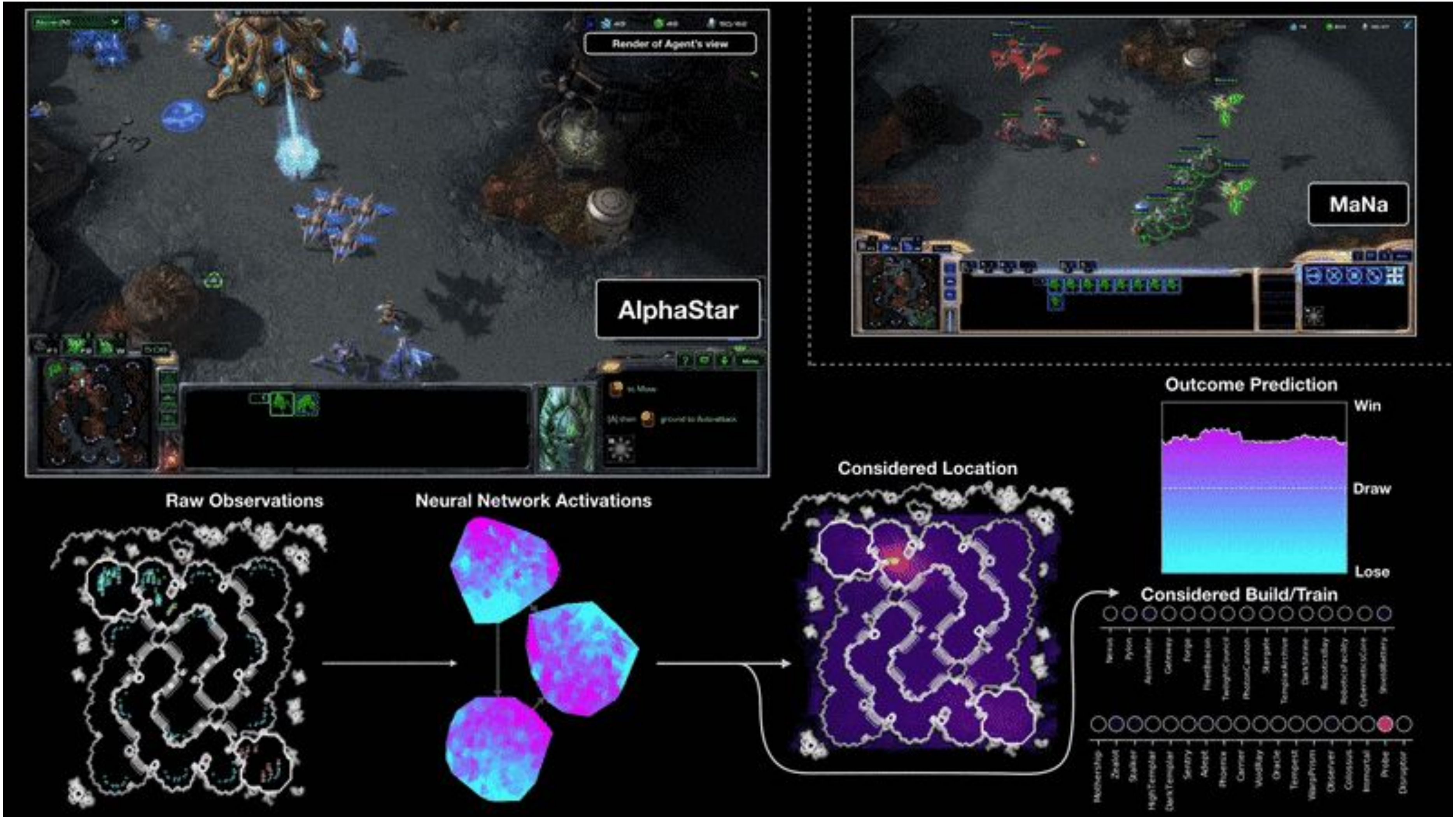
- Reinforcement learning (RL) is hot
  - AlphaGo Zero
  - Mastering Atari games
  - Mastering and defeating human progamers
    - SC2
    - Dota

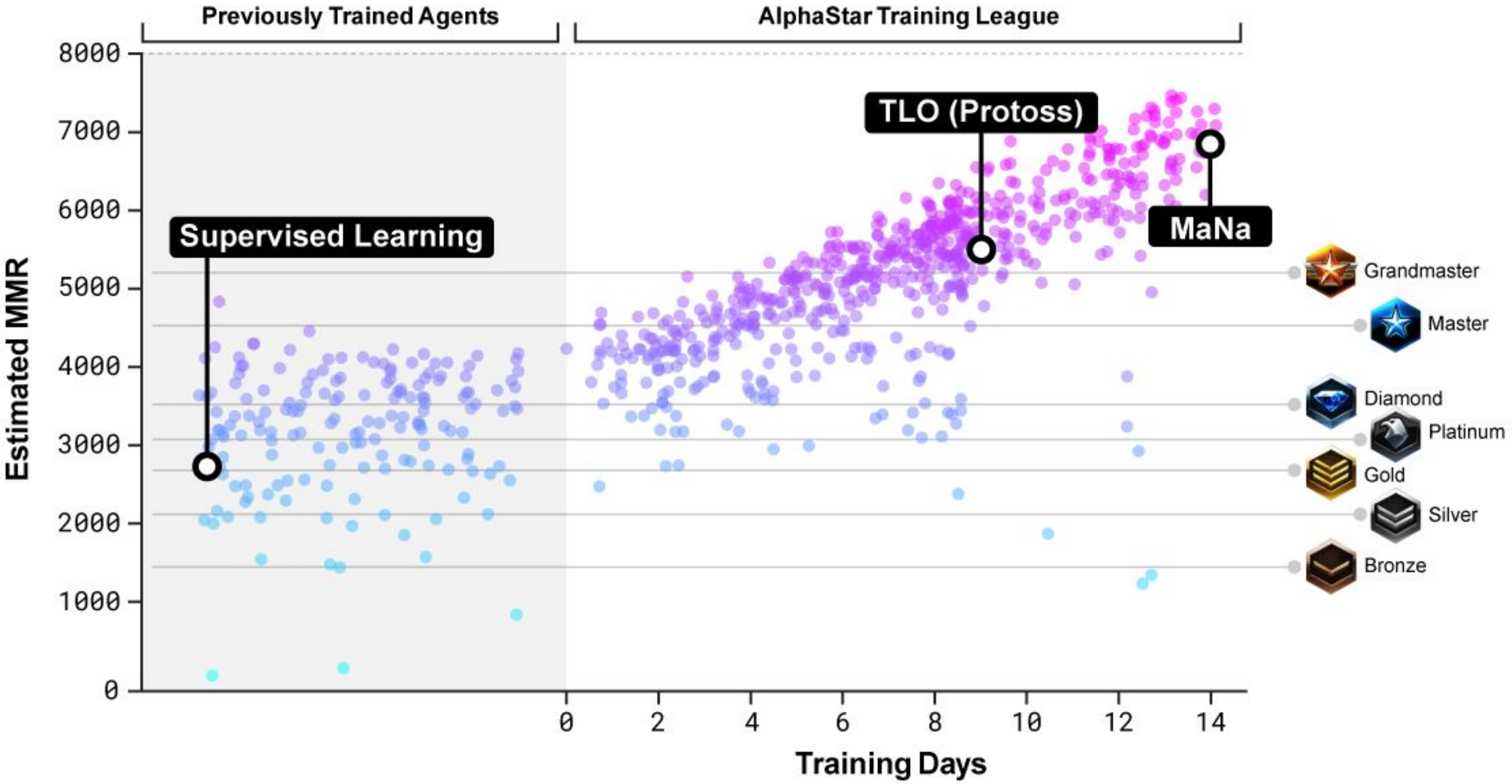
AlphaGo Zero

AlphaGo Zero

What AlphaGo Zero accomplished:

- Defeating the original AlphaGo 100 games to nil.
- Teaching itself to play Go without human knowledge.
- Achieved world class Go proficiency in 3 days.
- Achieved master class with less hardware resources.
- Required less training (4.9 millions games vs. 30 million)


Render of Agent's view
AlphaStar
MaNa
Outcome Prediction
Win
Draw
Lose
Raw Observations
Neural Network Activations
Considered Location
Considered Build/Train

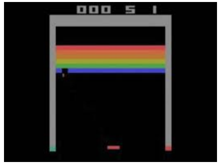

Previously Trained Agents
AlphaStar Training League
Estimated MMR
8000
7000
6000
5000
4000
3000
2000
1000
0
Supervised Learning
TLO (Protoss)
MaNa
Grandmaster
Master
Diamond
Platinum
Gold
Silver
Bronze
0
2
4
6
8
10
12
14
Training Days

Reinforcement learning

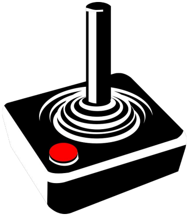
Agent

Environment

Observation



Action



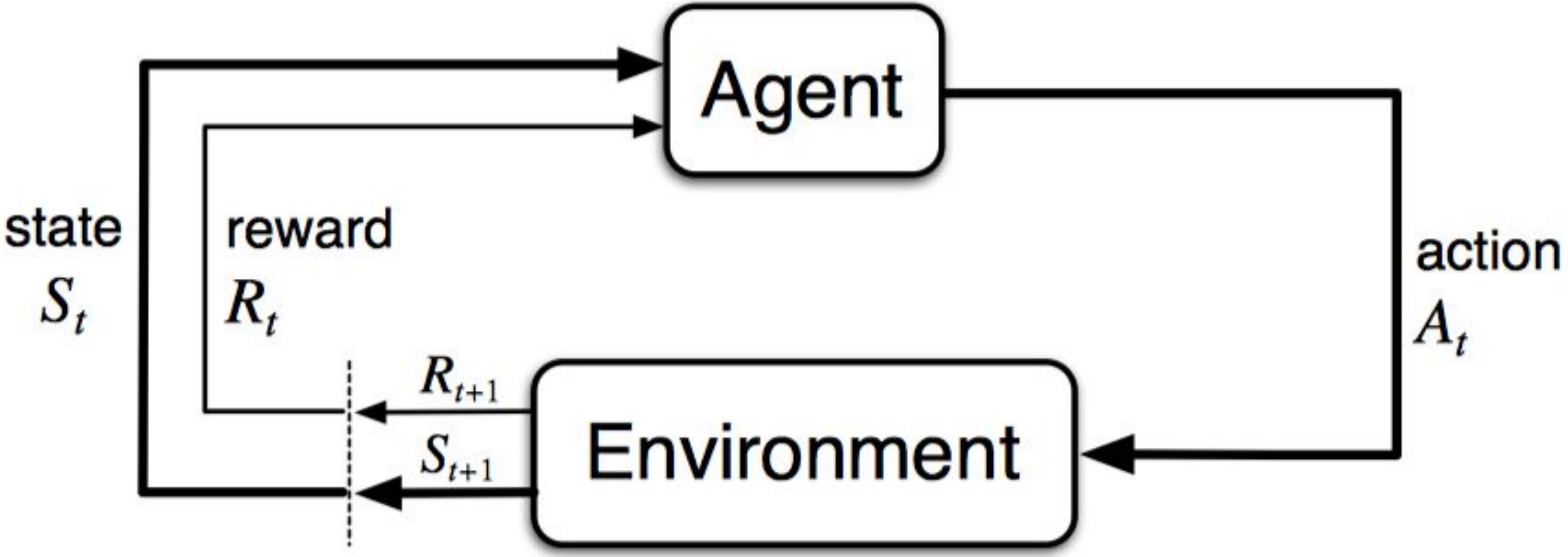
Reward

+1 ? -1 ? +50 ? -100

Rewards

- Hit a brick +1
- Clear the screen +100
- Drop the ball - 50
- Ball drops 3 times - Game Over

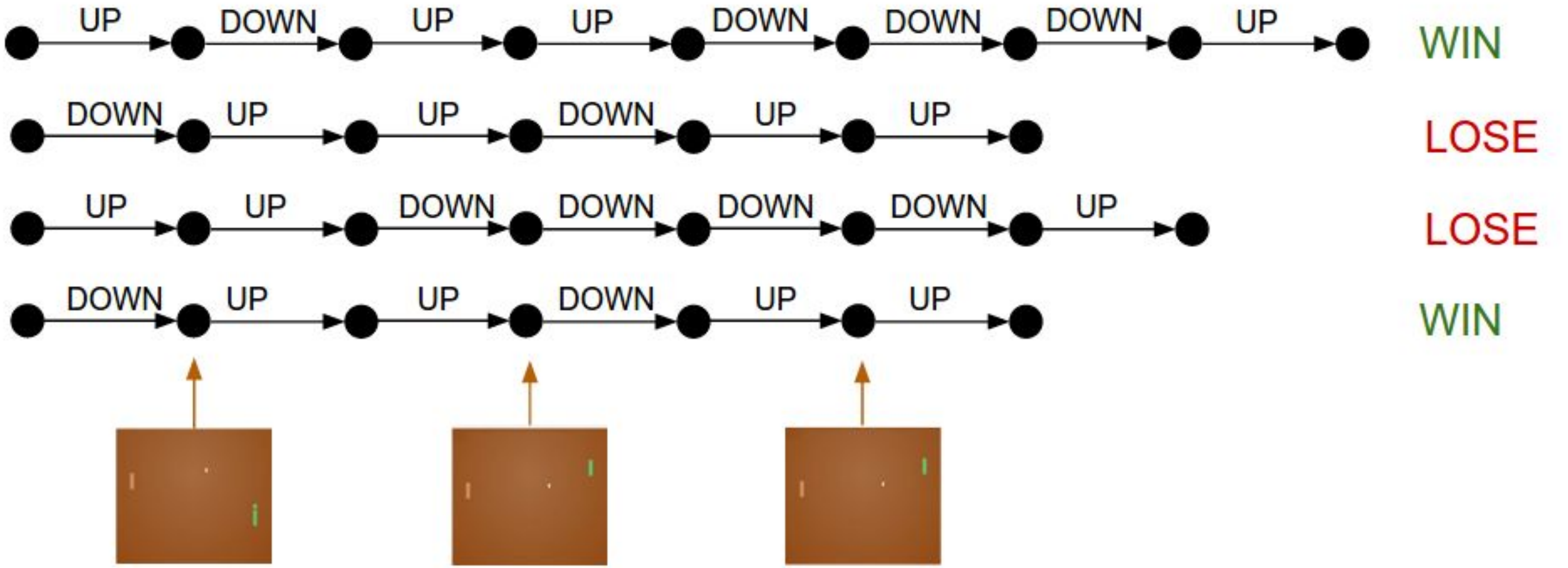
Canonical Agent–Environment Loop



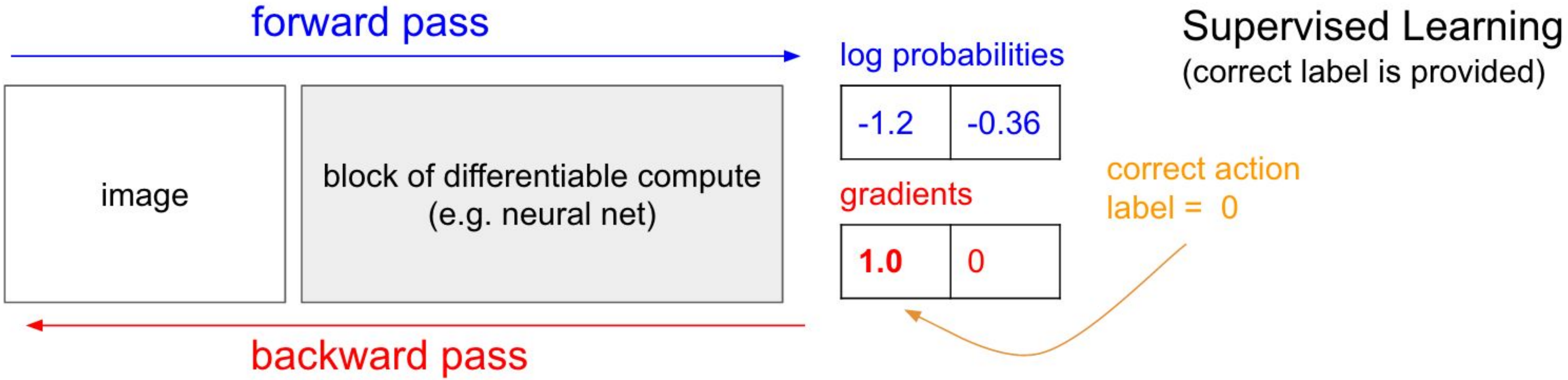
Markov Decision Process

- Agent
- Environment
- State
- Action
- Reward

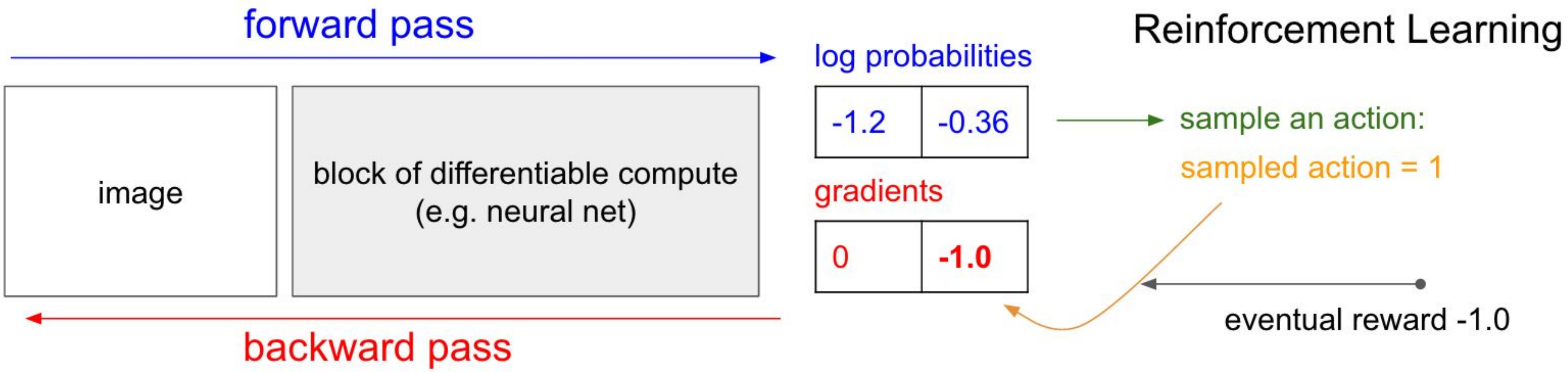
Trajectories



Supervised learning



Reinforcement learning



TF-Agents

What is TF-Agents?

A robust, scalable and easy to use Reinforcement Learning Library for TensorFlow

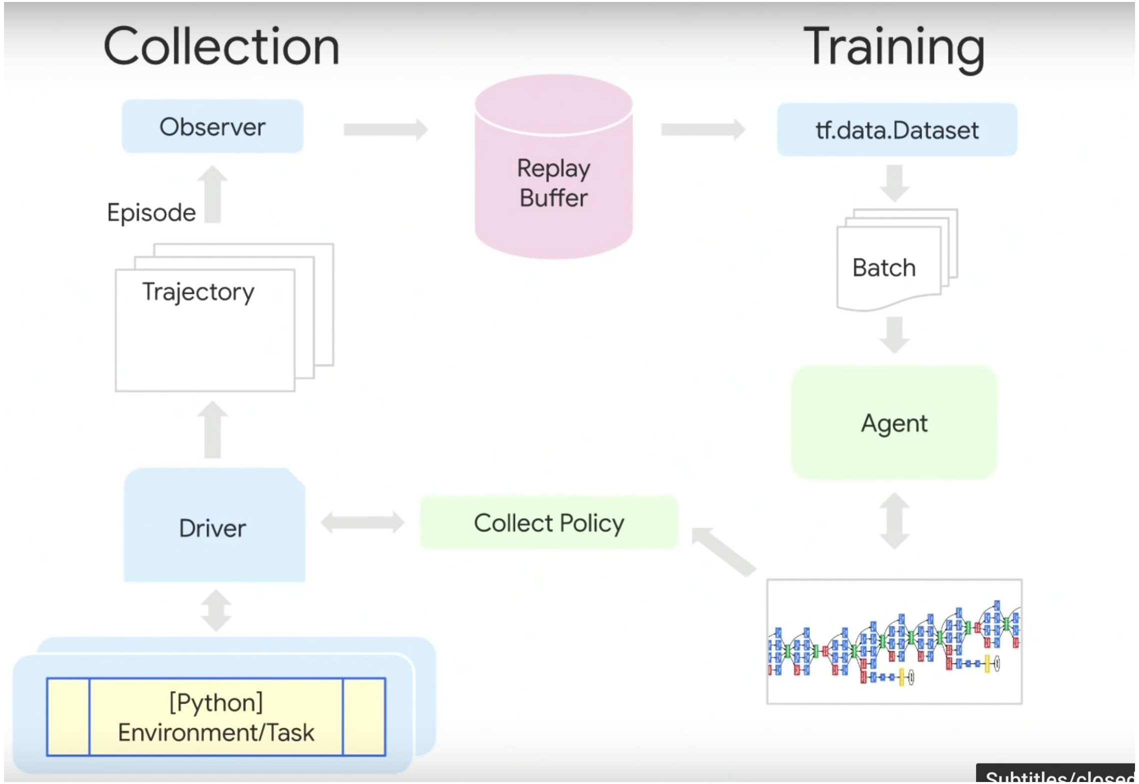
Why use TF-Agents?

- Great for learning RL: Jupyter notebooks, examples, documentation
- Well suited for solving complex problems
- Develop new RL algorithms quickly
- Well tested and easy to configure with gin-config

TF-Agents Focus: Ease of Use

- Built for TF 2.0
    - Develop and debug quickly with TF-Eager
    - Use tf.keras to define your networks
    - Use tf.function to speed up computations
    - Modular and extensible
- Work with TF.1.14 if you are not ready to upgrade

RL Landscape



Cart Pole

- Inverted pendulum
- Move left or move right

Environment implementation

Implementing an Environment

```
class CartPole(tf_agents.py_environment.PyEnvironment):

  def observation_spec(self): "Defines the Observations”

  def action_spec(self): "Defines the Actions”

  def _reset(self):
   """Reset the environment and return an initial time_step(reward, observation)."""

  def _step(self, action):
   """Apply the action and return the next time_step(reward, observation)."""
```

OpenAI Gym

Loading Gym CartPole

```
env = suite_gym.load("CartPole-V1")

print('Observation Spec:\n', env.observation_spec())
Observation Spec:
  BoundedArraySpec(
    shape=(4,), dtype=dtype('float32'), name=None,
    minimum=[-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38],
    maximum=[4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38])

print('Action Spec:\n', env.action_spec())
Action Spec:
  BoundedArraySpec(
    shape=(), dtype=dtype('int64'), name=None, minimum=0, maximum=1)
```

Policy gradients

Trying to balance the Pole

```
# Load the Environment
env = suite_gym.load("CartPole-V1")

# Define a Policy
policy = ActorPolicy(...)

time_step = env.reset()
episode_return = 0.0

# Start playing
while not time_step.is_last():
 policy_step = policy.action(time_step)
 time_step = env.step(policy_step.action)
 episode_return += time_step.reward
```

Policy-based RL

Goal: Learn a **stochastic policy** $\pi_\theta(a_t|s_t) = \Pr[a_t|s_t;\theta]$

to maximize **expected return**

$$\max_\theta \; \mathbb{E}_{\tau \sim \pi_\theta}[\mathcal{R}(\tau)]$$

Trajectory generated by following the policy $\tau = \{(s_1, a_1), \ldots, (s_T, a_T)\}$

Cumulative reward of the trajectory $\mathcal{R}(\tau) = \sum_t r(s_t, a_t)$

All made simple with TF-Agents

Preparing to Train with TF Agents

```
# Create the Enviroment
tf_env = tf_py_environment.TFPyEnvironment(suite_gym.load("CartPole-V1"))

# Create the Network
actor_net = actor_distribution_network.ActorDistributionNetwork(
    tf_env.observation_spec(), tf_env.action_spec(),
    fc_layer_params=[32, 64])

# Create the Agent
tf_agent = reinforce_agent.ReinforceAgent(
    tf_env.time_step_spec(),
    tf_env.action_spec(),
    actor_network=actor_net,
    optimizer=AdamOptimizer(learning_rate=learning_rate))
```

Replay buffes

Collect Experience and Train with TF Agents

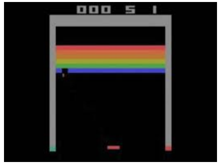
```
replay_buffer = TFUniformReplayBuffer()
driver = DynamicEpisodicDriver(
    tf_env, agent.collect_policy,
    observers=[replay_buffer.add_batch],
    num_episodes=1)

for _ in range(num_iterations):
    # Get experience
    driver.run()
    # Train the Agent
    experience = replay_buffer.gather_all()
    agent.train(experience)
    replay_buffer.clear()
```

Breakout

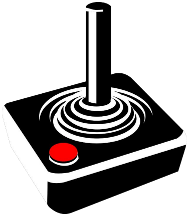
Agent

Environment

Observation

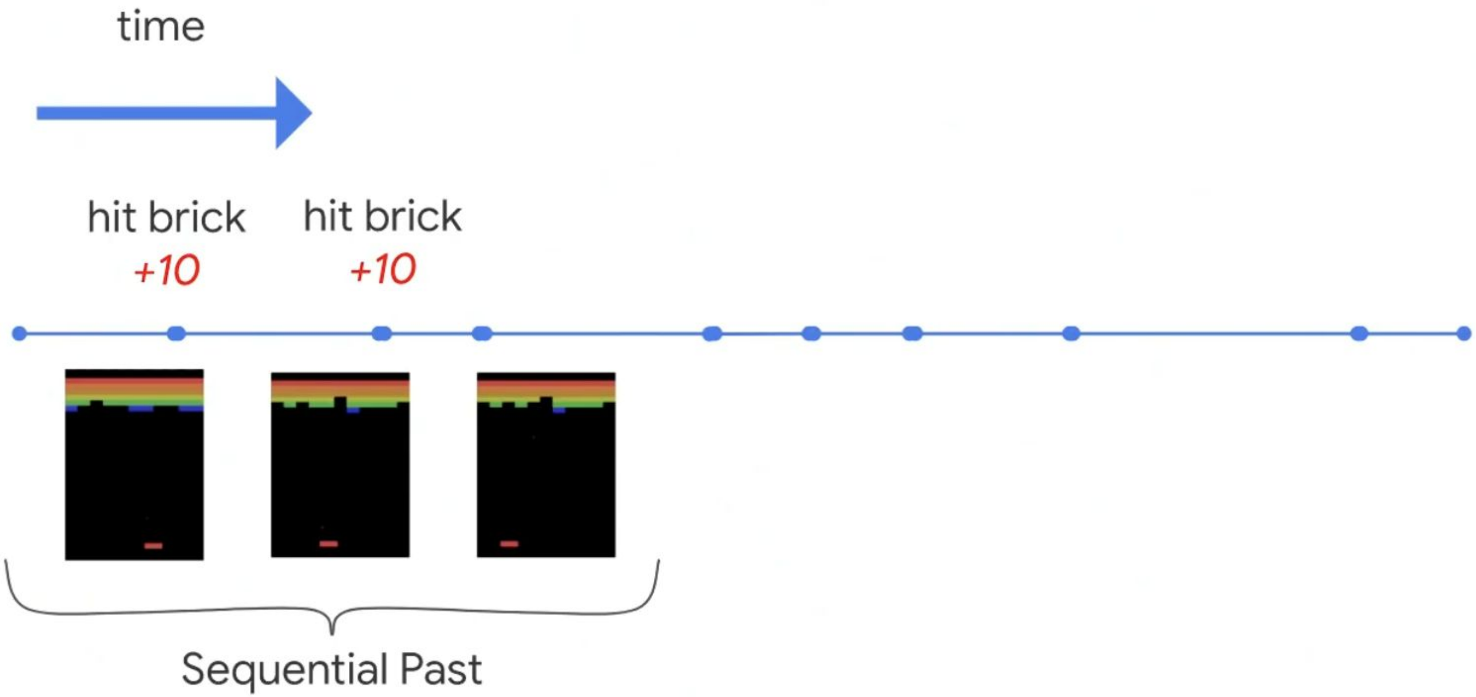


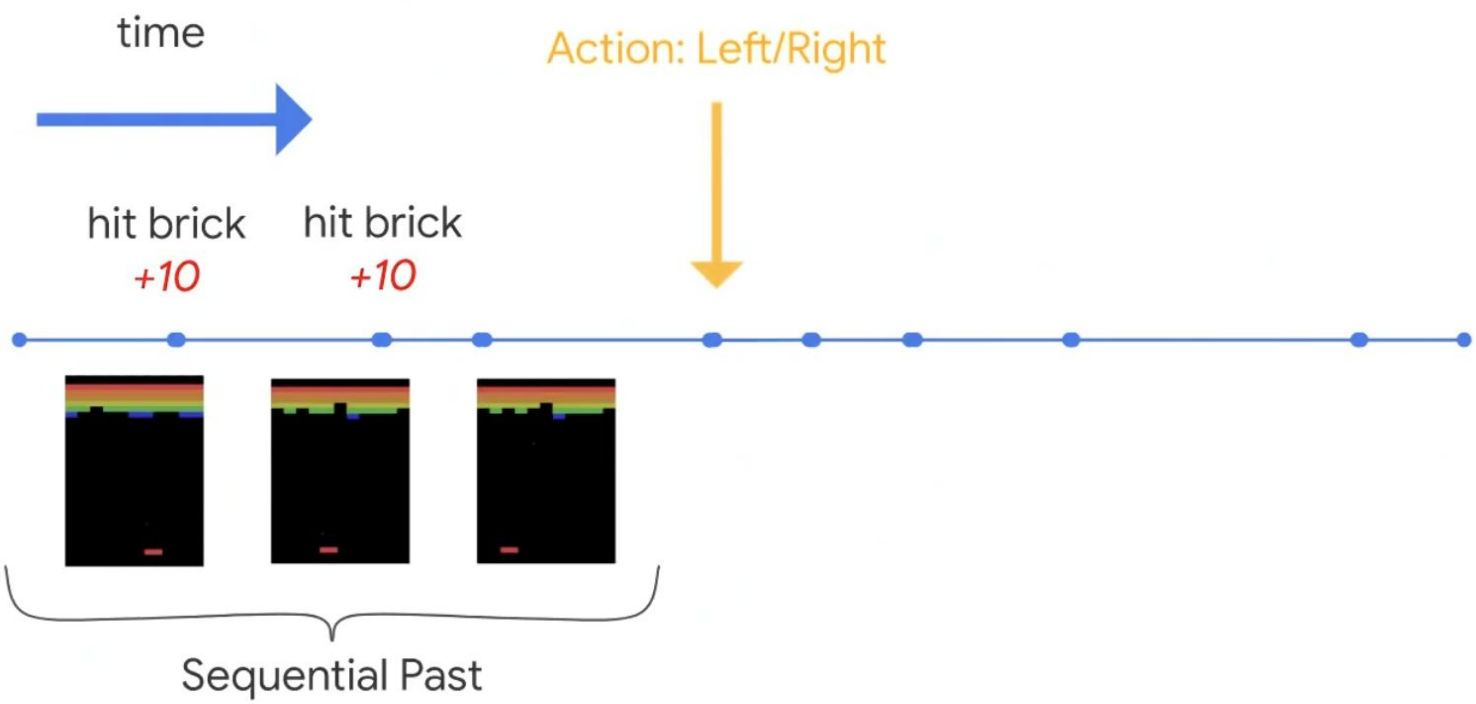
Action

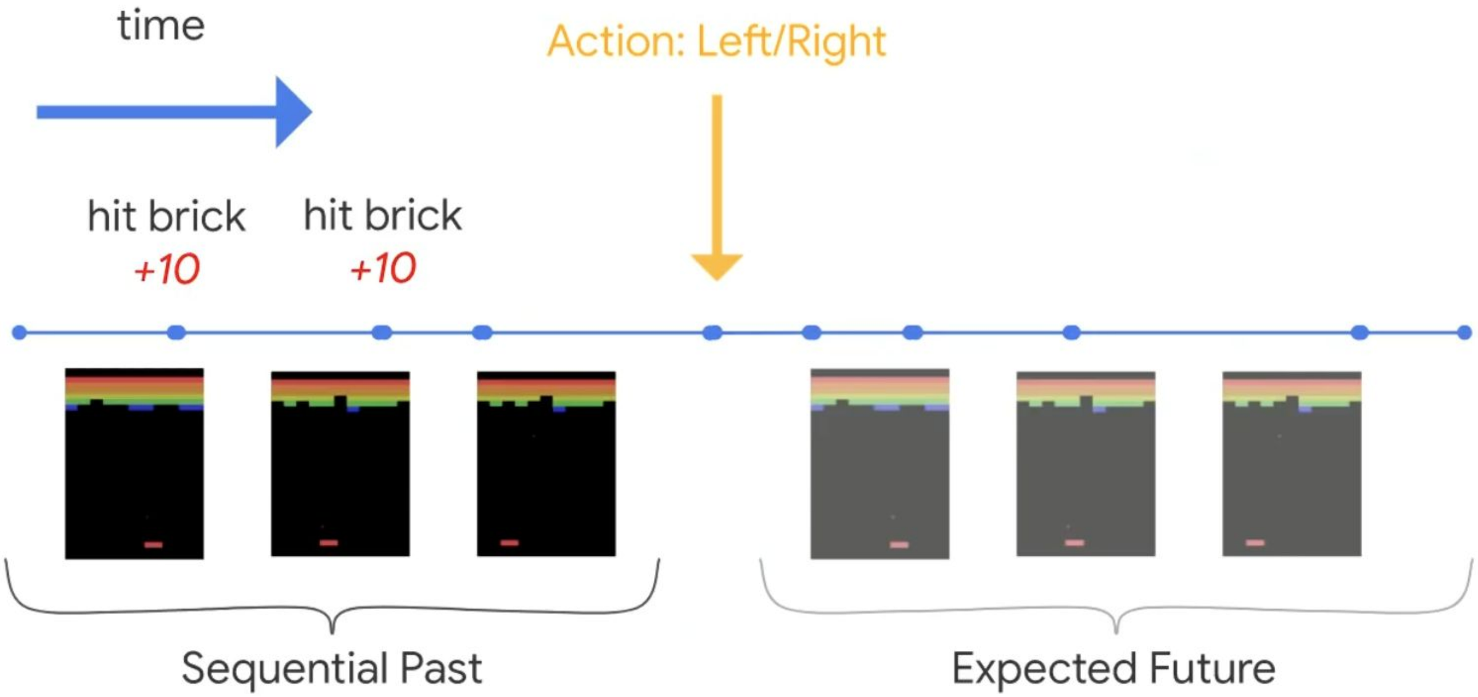


Reward +1 ? -1 ? +50 ? -100

Breakout timeline




time
Action: Left/Right
hit brick
+10
hit brick
+10
Sequential Past


time
Action: Left/Right
hit brick
+10
hit brick
+10
Sequential Past
Expected Future

Q-Learning

Action-Value Function: Expected return for this state and action pair.

$$q_\pi(s,a) = \mathbb{E}_\pi\left[R(s_t,a_t) + \gamma R(s_{t+1},a_{t+1}) + \cdots \,|\, S_t = s, A_t = a\right]$$

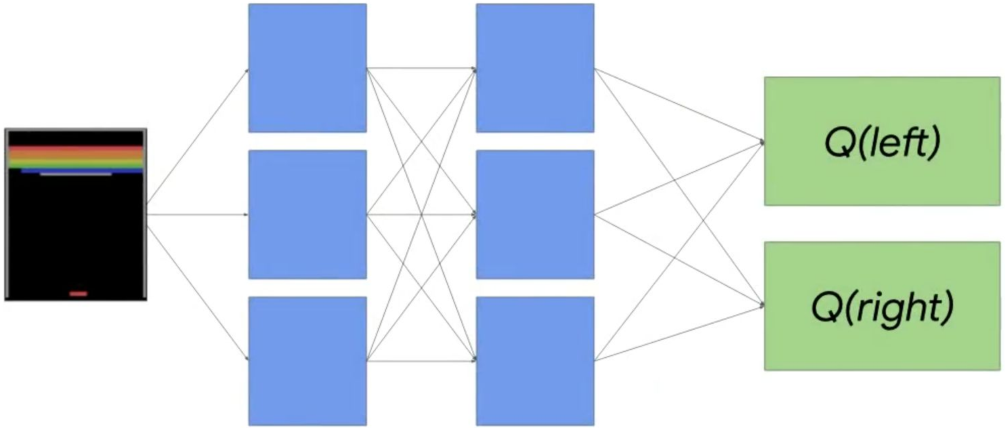
Optimal Action-Value Function and Policy

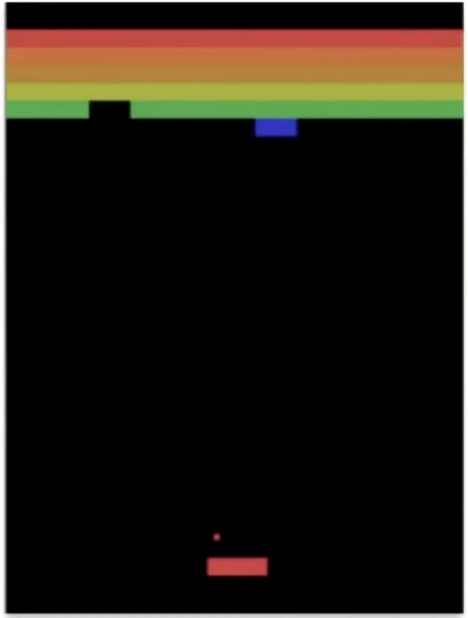
$$q_*(s,a) = \max_\pi q_\pi(s,a) \qquad \pi_*(s) = \arg\max_a \; q_*(s,a)$$

Goal: Directly approximate q_* with neural network *Q(s, a)*.

Q Networks

First introduced by DeepMind to play Atari games





To exploit: Take action with largest predicted Q value.

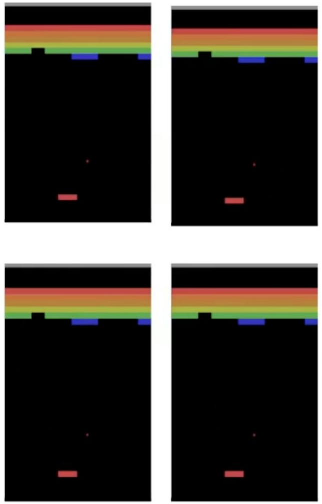
Loading Atari Environments (Breakout)

```
env = suite_atari.load("Breakout-v0")

print('Observation Spec:\n', env.observation_spec())
  Observation Spec:
    BoundedArraySpec(shape=(84, 84, 1), dtype='uint8', name=None, minimum=0, maximum=255)


print('Action Spec:\n', env.action_spec())
  Action Spec:
    BoundedArraySpec(shape=(), dtype=dtype('int64'), minimum=0, maximum=8)
```

Code: Build QNetwork. Build DQNAgent. Train it.

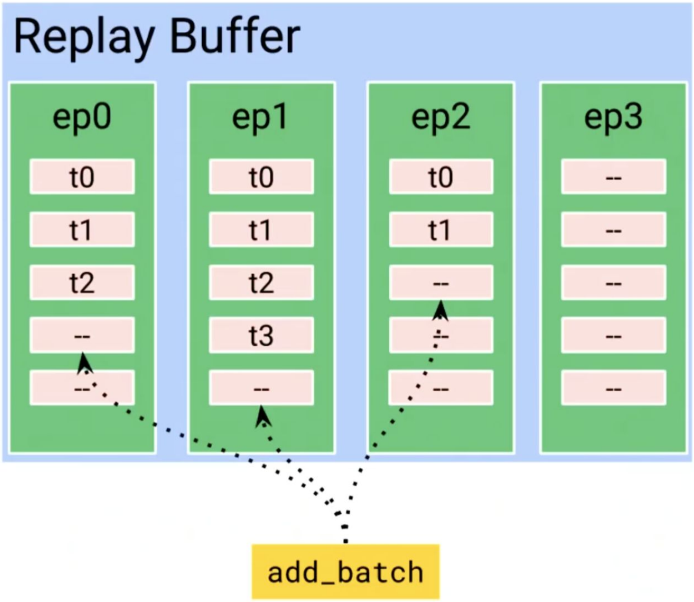
```
# Create a Network
q_net = q_network.QNetwork(observation_spec, action_spec, ...)

# Create the Agent
agent = dqn_agent.DqnAgent(...
  q_network=q_net,
  optimizer=AdamOptimizer(learning_rate=learning_rate))

# Get experience as a dataset
dataset = replay_buffer.as_dataset(num_steps=2).prefetch(3)

# Train the Agent
for batched_experience in dataset:
  agent.train(batched_experience)
```

Playing Breakout faster in TF

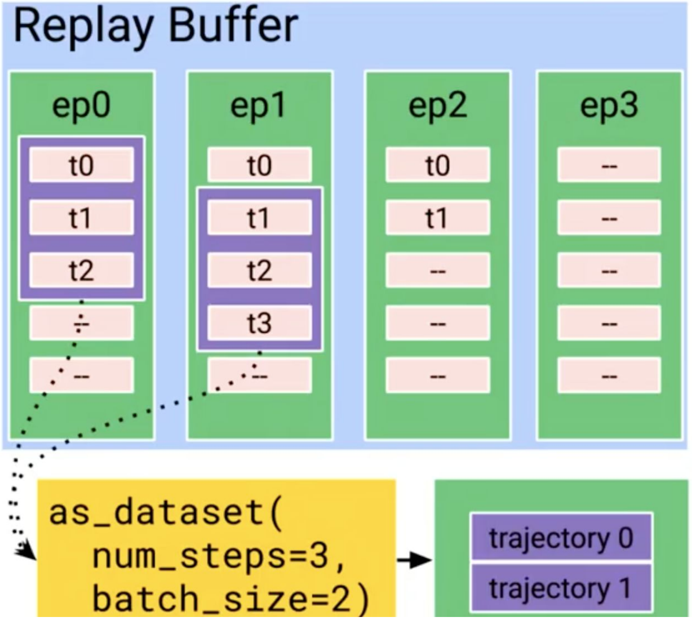


```
# Define multiple copies of the Environment
parallel_env = ParallelPyEnvironment(
    [BreakoutEnv() for _ in range(4)])
tf_env = TFPyEnvironment(parallel_env)

# Define a Policy
policy = QPolicy(...)

# Play in parallel
time_step = tf_env.reset()
episode_return = tf.zeros([4])
for _ in range(num_steps):
 policy_step = policy.action(time_step)
 time_step = tf_env.step(policy_step.action)
 episode_return += time_step.reward
```

Using Replay Buffers



Using Replay Buffers



Thank you!

- https://github.com/tensorflow/agents/tree/master/tf_agents/colabs
- https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf
- http://karpathy.github.io/2016/05/31/rl/
- https://www.youtube.com/watch?v=oPGVsoBonLM
- http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html