# Reinforcement Learning

## IMPALA - **Im**portance Weighted **A**ctor-**L**earner **A**rchitecture

Stefan Pantić

Faculty of Mathematics, University of Belgrade

06. November 2019.

# Contents

# Introduction

# Introduction

- Reinforcement learning is a machine learning approach for teaching **agents** how to act in an **environment** so that they maximize some given **reward**.
- Deep RL refers to the combination of RL algorithms with deep learning techiques. Most use the terms interchangeably.

# Learning methods comparison

- Supervised learning:
  - given a set of training examples $\{(x_i, y_i)\}_{i=1,\ldots,N}$ where $x_i$ is the feature vector of the $i$-th training example and $y_i$ is it's label, supervised learning learns a conditional probability distribution $p_X(y|x)$.
- Unsupervised learning:
  - given a set of training examples $\{x_i\}_{i=1,\ldots,N}$ where $x_i$ is the feature vector of the $i$-th training example, unsupervised learning learns a joint probability distribution $p_X(x)$ (note that this definition doesn't apply to all unsupervised learning tasks).
- Reinforcement learning:
  - RL is modeled as a **Markov decision process**.

# Markov Decision Processes

# Standard mathematical formalism behind RL

A **Markov Decision Process** (MDP) is a 5-tuple $\langle S, A, R, P, \rho_0 \rangle$ where:

- $S$ is the set of valid states
- $A$ is the set of valid actions
- $R : S \times A \times S \to \mathbb{R}$ is the reward function with $r_t = R(s_t, a_t, s_{t+1})$
- $P : S \times A \to \mathcal{P}$ is the transition probability function with $P(s^{'}|s, a)$ being the probability of transitioning to state $s^{'}$ if you take action $a$ in state $s$.
- $\rho_0$ is the initial state distribution

# Markov property

**Definition (Markov property)**

The conditional probability distribution of future states of the process depends only on the current state and not on the sequence of events that preceded it; that is, **given the present, the future does not depend on the past**.

# Key Concepts

# Agent-environment interface

- The main characters of RL are the **agent** and the **environment**.
- The **environment** is the world that the **agent** interacts with.

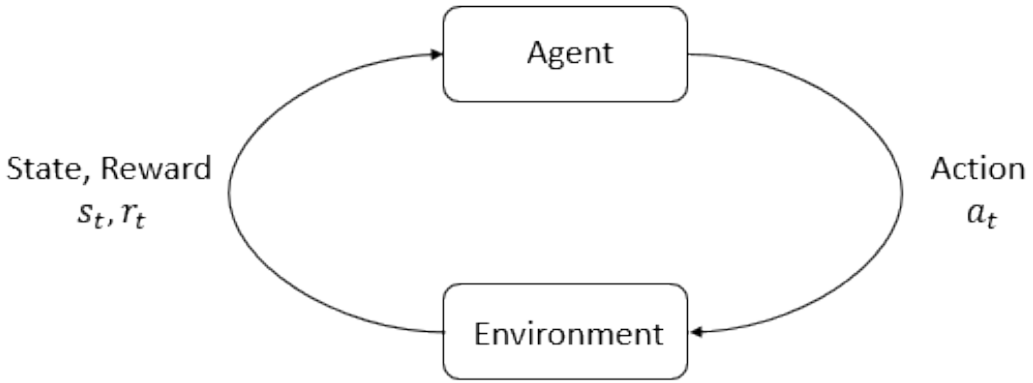


Figure: Agent-environment interaction loop

# States and Observations

- A **state** *s* is the complete description of the world (environment). No world information is omitted from the state.
- An **observation** *o* is a partial description of the state.

## A note on notation

In practice, the symbol *s* is used to denote both, specifically when discussing about how an agent decides which action to take conditioned on the current state. Most modern literature and research papers follow this notation and the symbol *o* is seldom used.

# Action spaces

- The set of all valid actions in a given environment is called the **action space**.
- Action spaces can be either **discrete**, where only a finite number of actions are avaliable to the agent, and **continuous** where actions are real-valued vectors.

## Application of algorithms

Some RL algorithms can only be directly applied to one class of action spaces.

# Policies

- A **policy** is a rule used by an agent to decide which actions to take.
- A policy can be either **deterministic**

$$a_t = \mu(s_t)$$

which emits an action to be executed by the agent, or **stochastic**

$$a_t \sim \pi(\cdot|s_t)$$

which is a probability distribution that is sampled to produce an action. The most common types of stochastic policies in RL are **categorical policies** (used for discrete action spaces) and **diagonal Gaussian policies** (used for continuous action spaces).

- In RL we deal with parameterized policies (eg. a neural network). Parameters are denoted with $\theta$ which is written as a subscript on the policy symbol to highlight the connection ($\mu_\theta$, $\pi_\theta$).

# Categorical policies

- A categorical policy is like a classifier over discrete actions.

**Definition (Log-Likelihood)**

Denote the last layer as a probability distribution $P_\theta(s)$. It is a vector of dimension $n$ where $n$ denotes the number of available actions. The log likelihood for an action $a$ can be obtained by indexing into the vector:

$$\log \pi_\theta(a|s) = \log [P_\theta(s)]_a$$

# Diagonal Gaussian policies

- A multivariate Gaussian distribution is described by a mean vector $\mu$ and a covariance matrix $\Sigma$. A diagonal Gaussian distribution is a special case where $\Sigma$ has entries only on the diagonal, so we can represent it using a vector.
- In a diagonal Gaussian policy we use a neural network that maps observations to mean actions $\mu_\theta(s)$, and the standard deviations (which are usually returned log-ed) are returned as $\log \sigma_\theta(s)$. $\sigma_\theta$ may share layers with the mean network.
- Actions are sampled using a vector of noise from a spherical multivariate Gaussian distribution $z \sim \mathcal{N}(0, E)$ using the following rule:

$$a = \mu_\theta(s) + \sigma_\theta(s) \odot z$$

# Policies

- Although all RL algorithms must follow some policy in order to decide which actions to take, the learning procedure of the algorithm doesn't always have to take it into account while learning.
- Algorithms which concern about the policy which yielded past action-state decisions are referred as **on-policy**, while those that ignore it are called **off-policy**.
- Algorithms whose goal is to learn an optimal policy are called **policy-based**.

# Trajectories

- A trajectory $\tau$ is a sequence of states and actions in the world:

$$\tau = (s_0, a_0, s_1, a_1, ...)$$

- State transitions (from $s_t$ to $s_{t+1}$) are governed by the natural laws of the environment and depend only on the most recent action $a_t$. State transitions occur via the transition function mentioned in Section 2.

# Reward and Return

- The reward function $R$ depends on the current state of the world $s_t$, the action just taken $a_t$, and the next state of the world $s_{t+1}$:

$$r_t = R(s_t, a_t, s_{t+1})$$

- The agents goal is to maximize the cumulative reward over a trajectory.
- **Infinite-horizon discounted return** is the sum of all rewards ever obtained by the agent, but discounted by how far into the future they are obtained, $\gamma \in (0, 1)$ represents the discount factor:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

# RL optimization problem

- The goal of RL is to select a policy $\pi$ that maximizes the **expected return** for an agent acting according to it.

## Definition (**RL optimization problem**)

Suppose that the environment transitions and the policy are stochastic. The probability of a $T$-step trajectory is:

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t)$$

If we denote the expected return of $\pi$ with $J(\pi)$ we get:

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)]$$

The central optimization problem of RL can then be expressed as:

$$\pi^* = \arg\max_{\pi} J(\pi)$$

where $\pi^*$ is the **optimal policy**.

# Value functions

- The **value** of a state (or state-action pair) is the expected return if we start at that state and act according to a particular policy from that point to infinity. Value functions represent a very important component in almost all RL algorithms.
- Algorithms whose goal is to learn an optimal value function are called **value-based**.

# Value functions

- There are four main value functions:
  1. The **on-policy value function** $V^\pi(s)$ which gives the expected return if we start at state $s$ and act according to policy $\pi$:
  $$V^\pi(s) = \mathop{\mathbb{E}}_{\tau \sim \pi}[R(\tau)|s_0 = s]$$
  2. The **on-policy state-action value function** $Q^\pi(s, a)$ which gives the expected return if you start in state $s$, take an arbitrary action $a$ (which was not necessarily sampled from $\pi$), and then act according to policy $\pi$:
  $$Q^\pi(s, a) = \mathop{\mathbb{E}}_{\tau \sim \pi}[R(\tau)|s_0 = s, a_0 = a]$$
  3. The **optimal value function** $V^*(s)$ which gives the expected return if we start at state $s$ and act according to the optimal policy for that environment:
  $$V^*(s) = \max_\pi \mathop{\mathbb{E}}_{\tau \sim \pi}[R(\tau)|s_0 = s]$$
  4. The **optimal state-action value function** $Q^*(s, a)$ which gives the expected return if you start in state $s$, take an arbitrary action $a$ (which was not necessarily sampled from $\pi$), and then act according to the optimal policy for that environment:
  $$Q^*(s, a) = \max_\pi \mathop{\mathbb{E}}_{\tau \sim \pi}[R(\tau)|s_0 = s, a_0 = a]$$

# Policy Gradients

# Policy Gradients

- **Policy gradient methods** refers to a family of methods for learning the parameters of a parameterized policy $\pi_\theta$ based on the gradient of the expected return:

$$J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi} [R(\tau)]$$

- These methods seek to maximize performance so their updates of the policy parameters $\theta$ approximate gradient ascent in $J$ via the rule:

$$\theta_{t+1} = \theta_t + \widehat{\nabla J(\pi_{\theta_t})}$$

where $\widehat{\nabla J(\pi_{\theta_t})}$ is a stochastic estimate whose expectation approximates the gradient of the performance measure with respect to $\theta_t$.

# Policy Gradients

## Definition (**Probability of a trajectory**)

The probability of a trajectory $\tau = (s_0, a_0, ..., sT + 1)$ given a policy $\pi_\theta$ that an agent acts by is:

$$P(\tau|\theta) = \rho_0(s_0) \prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Analogously, the **log-probability** of a trajectory is:

$$\log P(\tau|\theta) = \log \rho_0(s_0) + \sum_{t=0}^{T} \big( \log P(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t) \big)$$

## Definition (**The Log-Derivative trick**)

Using the log derivative rule $log^{'} f(x) = \frac{1}{f(x)} f^{'}(x)$ we can get the following identity:

$$\nabla_\theta P(\tau|\theta) = P(\tau|\theta) \nabla_\theta \log P(\tau|\theta)$$

# Policy Gradients

- Using the given definitions, we can compute the **gradient-log-probability** of a trajectory:

$$\nabla_\theta \log P(\tau|\theta) = \nabla_\theta \log \rho_0(s_0) + \sum_{t=0}^{T} \left(\nabla_\theta \log P(s_{t+1}|s_t, a_t) + \nabla_\theta \log \pi_\theta(a_t|s_t)\right)$$

- Notice that $\rho_0(s_0)$ and $\log P(s_{t+1}|s_t, a_t)$ relate to the environment and have no dependence on the parameters $\theta$, thus the gradients of both terms with regards to $\theta$ are 0.
- Removing unnecessary terms gives the following identity:

$$\nabla_\theta \log P(\tau|\theta) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

# Policy Gradients

- Using the given identities we can derive the expression for the basic policy gradient.

## Policy gradient derivation

$$
\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[R(\tau)] \\
&= \nabla_\theta \int_\tau P(\tau|\theta)R(\tau) \\
&= \int_\tau \nabla_\theta P(\tau|\theta)R(\tau) \\
&= \int_\tau P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)R(\tau) \\
&= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[\nabla_\theta \log P(\tau|\theta)R(\tau)] \\
&= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau)\right]
\end{aligned}
$$

# Policy Gradients

- We can estimate the derived expression using a sample mean.

## Estimating policy gradient

Given a set of trajectories $\mathcal{D} = \{\tau_i\}_{i=1,..,N}$ where each is obtained by an agent acting in the environment using a parametrized policy $\pi_\theta$, the policy gradient can be estimated as a sample mean over the given set of trajectories:

$$\widehat{\nabla_\theta J(\pi_\theta)} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)$$

where $|\mathcal{D}|$ is the number of trajectories in $\mathcal{D}$.

# Policy Gradients

- There is one caveat with our derived formula for the policy gradient :

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \left[ \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau) \right]$$

- It takes a gradient step in proportion to $R(\tau)$ for each action.

**Reminder**

$R(\tau)$ is the (discounted) sum of all rewards **ever** obtained:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

# Policy Gradients

- Agents should **only** reinforce actions based on their consequences. Only rewards that come after an action can give an image on how good that action was.
- Keeping this in mind an equivalent expression for the policy gradient can be derived without changing the gradient in expectation:

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \left[ \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) \right]$$

- Approximating this expectation leads to a significant reduction in variance compared to the previous of expression.

# Policy Gradients

**Lemma (1)**

*$P_\theta$ is a parameterized policy distribution over a random variable $x$, then:*

$$\mathop{\mathbb{E}}_{x \sim P_\theta} [\nabla_\theta \log P_\theta(x)] = 0$$

- A direct consequence of Lemma 1 is that for any function $b$ which only depends on the state the following is true:

$$\mathop{\mathbb{E}}_{x \sim P_\theta} [\nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t)] = 0$$

# Policy Gradients

- Using the Lemma and its consequence we can conclude that we are able to add or subtract terms from out expression without changing it in expectation. This gives us the following expression:

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \left[ \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} \left( R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t) \right) \right]$$

- Any function $b$ used in this way is called a **baseline**.
- The goal of introducing baselines is to reduce the variance of our mean estimate. Empirically, the choice of $b(s_t) = V^\pi(s_t)$ achieves the desired effect.

# Policy Gradients

- Along with the policy, the value function also has to be approximated. This is usually done using a neural network. The value function network can (and usually does) share layers with the policy network.
- The simplest method of learning $V_\phi^\pi(s_t)$ is minimizing a MSE objective:

$$\phi_k = \arg\min_\phi \mathop{\mathbb{E}}_{s_t, \hat{R}_t \sim \pi_k} \left[ (V_\phi^\pi(s_t) - \hat{R}_t)^2 \right]$$

where $\pi_k$ is the policy at the $k^{th}$ training epoch.

# IMPALA

# Problems in RL

- Some of the main problems with RL are:
  - Low sample efficiency
  - Scalability of training
  - Balance of exploration vs. exploitation
  - Hard to generalize on unseen environments
  - etc.

# Actor-Critic methods

- The main types of RL algorithms are:
  1. **Value-Based** where they try to approximate the optimal value function
  2. **Policy-Based** where they try to find the optimal policy directly
- Merging the families we get the **Actor-Critic** family of algorithms.
- The **actor** takes an input state $s_t$ and outputs the best action according to $\pi_\theta$. It controls how the agent behaves by learning the optimal policy (**policy-based component**). The **critic** evaluates the action by computing the value function (**value-based component**).

# A3C - Asynchronous Advantage Actor-Critic

- Released in 2016 by DeepMind.
- It is simple, robust, fast and achieved higher scores in baseline RL tasks than classic policy gradient and Q-learning methods.
- Instead of learning the state-value function, A3C learns the advantage function $A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$.
- Among the first to introduce the notion of having **multiple independent agents**, each with their own weights, who interact with their own local copy of the environment in parallel.
- Each agent computes its own gradients which are then sent to a **global parameter server** which averages them and updates its weights which are then projected to the agents.

# Problems with A3C

- Some agents will be working with an older version of the parameters. Only guarantees that this won't cause divergence are empirical results.
- Scalability is still relatively limited - gradient computation is expensive.

# IMPALA

- When talking about IMPALA we are actually referring to two components:
  1. **IMPALA**: a distributed agent architecture
  2. **V-trace**: an off-policy correction method

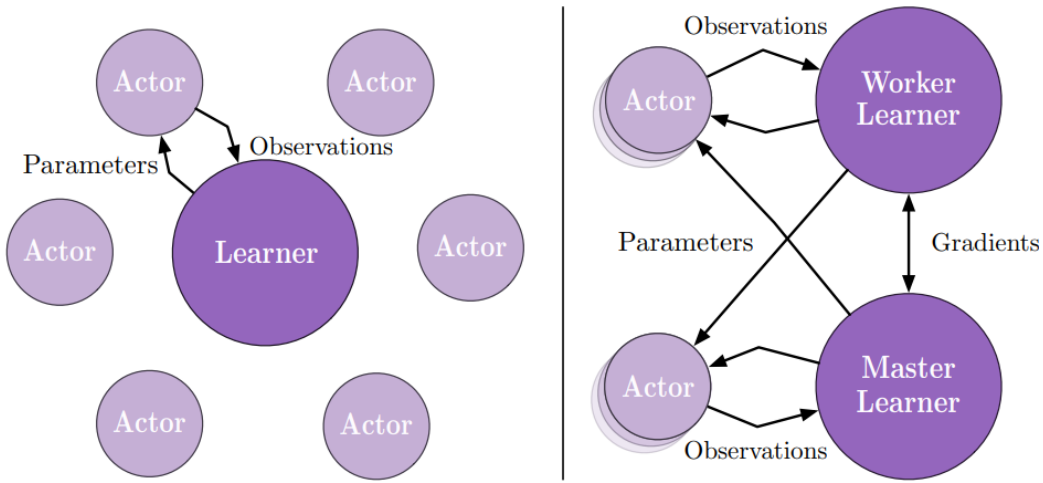


Figure: IMPALA scalable learning architecture

# Learning architecture

- Unlike A3C-based agents, IMPALA **actors** don't communicate gradients to the parameter server. Rather, they communicate trajectories of experience to a centralized **learner**. The learner then computes gradient updates on mini-batches of trajectories, allowing for very high throughput.
- The policy used to generate a trajectory can lag behind the learners policy so learning becomes off-policy, and therefore, the **V-trace** algorithm is required to correct for this discrepancy.

# IMPALA

- IMPALA uses an actor-learner setup to learn a parameterized policy $\pi_\theta$ and baseline function $V_\phi^\pi(s_t)$. The architecture consists of a set of actors generating trajectories and one or more learners using the experiences to learn $\pi_\theta$ (and $V_\phi^\pi(s_t)$) off-policy.
- Before each **unroll**, an actor updates its local policy $\mu_\theta$ to the latest learner policy $\pi_\theta$ and runs it for $n$ steps in the environment. After $n$ steps, the actor sends a trajectory of states, actions and rewards $\tau = (s_t, a_t, r_t)_{t=1,\dots,n}$, along with their policy distributions $\mu_\theta(a_t|s_r)$, $t = \overline{1,n}$ to a learner via a queue.
- The learner updates its policy on mini-batches of collected experience while correcting for **policy-lag** using V-trace.

# V-trace

**Definition (Behaviour Policy and Target Policy)**

The goal of an off-policy RL algorithm is to use trajectories generated by some policy $\mu$, called the **behaviour policy**, to learn the value function of another policy $\pi$, called the **target policy**.

# V-trace

## V-trace actor-critic algorithm

Consider parameterized representations of the policy $\pi_\theta$ and value function $V^\pi_\phi$. Trajectories have been generated by actors using the behaviour policy $\mu_\theta$. We denote the **V-trace targets** as $v_t$. At training time $t$, the value parameters $\phi$ are updated by gradient descent on the MSE loss to the target $v_t$, i. e. in the direction of:

$$\big(v_t - V^\pi_\phi(s_t)\big)\nabla_\phi V^\pi_\phi(s_t)$$

and the policy parameters $\theta$ are updated in the direction of the policy gradient:

$$\rho_t \nabla_\theta \log \pi_\theta(a_t|s_t)\big(r_t + \gamma v_{t+1} - V^\pi_\phi(s_t)\big)$$

An additional **entropy regularization bonus** may be added to promote exploration and prevent premature convergence:

$$-\nabla_\theta \sum_a \pi_\theta(a|s_t) \log \pi_\theta(a|s_t)$$

The overall update rule is obtained by summing these gradients scaled by appropriate coefficients (which are hyperparameters of the algorithm).

# V-trace targets

**Definition (V-trace target for $V_\phi^\pi(s_t)$)**

Consider a trajectory $\tau = (s_t, a_t, r_t)_{t=s,\dots,s+n}$ generated by the actor following some policy $\mu_\theta$. We define the $n$-step V-trace target for $V_\phi^\pi(s_s)$, our value approximation at state $s_s$ as:

$$v_s \overset{\text{def}}{=} V_\phi^\pi(s_s) + \sum_{t=s}^{s+n-1} \gamma^{t-s} \Big(\prod_{i=s}^{t-1} c_i\Big) \delta_t V_\phi^\pi$$

where $\delta_t V_\phi^\pi \overset{\text{def}}{=} \rho_t\big(r_t + \gamma V_\phi^\pi(s_{t+1}) - V_\phi^\pi(s_t)\big)$ is a **temporal difference** for $V_\phi^\pi$, and $\rho_t \overset{\text{def}}{=} \min(\overline{\rho}, \frac{\pi_\theta(a_t|s_t)}{\mu_\theta(a_t|s_t)})$ and $c_i \overset{\text{def}}{=} \min(\overline{c}, \frac{\pi_\theta(a_i|s_i)}{\mu_\theta(a_i|s_i)})$ are truncated **importance sampling weights**. We assume $\prod_{i=s}^{t-1} c_i = 1$ for $s = t$ and $\overline{\rho} \geq \overline{c}$. The formula for the V-trace target can be rewritten recursively:

$$v_s = V_\phi^\pi(s_s) + \delta_s V_\phi^\pi + \gamma c_s \big(v_{s+1} - V_\phi^\pi(s_{s+1})\big)$$

# V-trace targets

- $\rho_t$ is the fixed point in the update rule. It represents a fixed point between the policies $\mu_\theta$ and $\pi_\theta$. The truncation parameter $\overline{\rho}$ represents the nature of the value function we converge to.
- The product $c_s ... c_{t-1}$ measures how much a temporal difference $\delta_t V_\phi^\pi$ observed at time $t$ impacts the update of the value function at a previous time $s$. $\overline{c}$ represents the speed at which we converge to that function.

# Conclusion

# Conclusion

- We introduced reinforcement learning and its key concepts.
- We introduced a robust and efficient way of learning a parameterized agent policy in the way of policy gradients.
- We introduced the notion of actor-critic methods as an alternative approach to policy gradients.
- Finally, we introduced A3C and IMPALA as scalable, efficient and robust actor-critic methods for complex RL tasks.

# Code

- Source code for the IMPALA paper:
  - `https://github.com/deepmind/scalable_agent`
- ray (RLlib) - a scalable distributed machine learning framework where you can find an implementation of IMPALA, A3C, and many more bleeding edge RL algorithms:
  - `https://ray.readthedocs.io/en/latest/rllib.html`
  - `https://github.com/ray-project/ray`

# References I


Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al.
Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures.
In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu.
Asynchronous methods for deep reinforcement learning.
*CoRR*, abs/1602.01783, 2016.

OpenAI.
OpenAI spinning up, 2019.

Richard S. Sutton and Andrew G. Barto.
*Reinforcement Learning: An Introduction*.
The MIT Press, second edition, 2018.