What’s IMPALA?

- Single reinforcement learning agent with same parameters solves a multitude of tasks, with the aid of a bunch of computers.
What’s IMPALA?

- Single reinforcement learning agent with same parameters solves a multitude of tasks, with the aid of a bunch of computers
- Published by Deepmind in 2018
What’s IMPALA?

- Single reinforcement learning agent with same parameters solves a multitude of tasks, with the aid of a bunch of computers
- Published by Deepmind in 2018
- SINGLE AGENT, 30 3D WORLD TASKS, 57 ATARI GAMES
What’s IMPALA?

- Single reinforcement learning agent with same parameters solves a multitude of tasks, with the aid of a bunch of computers
- Published by Deepmind in 2018
- SINGLE AGENT, 30 3D WORLD TASKS, 57 ATARI GAMES
- Better performance with less hardware and data requirements
Background - Brief History

- SNARC (Stochastic Neural Analog Reinforcement Calculator)
- Tabular Q Learning
- Q Network
- DQN
- A3C
- IMPALA
Background - Actor Critic Learning

Actor $\equiv$ policy improvement

Critic $\equiv$ policy evaluation
Background - On Policy, Off Policy

- On Policy - Instant use of learned policy
- State is important
- Off Policy - Learns Value Mapping uses another policy to make decision (Epsilon greedy)
- Exploration is enough to train
**Background - Q Learning**

- Value Function vs Q Function
- Off Policy
- Learning baby steps:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t))$$
Related Work - DQN

- DNN to estimate Q values
- Experience Replay
- Target Network (Periodic Update with Q Network)
- Clipping Rewards (All positive rewards +1, negatives -1)
Related Work - A3C

- Three A’s of Asynchronous Advantage Actor Critic
- Async : Global Network, Multiple Agents, Own copy of network params and environment
- Advantage : Discounted Reward vs Advantage,
- Advantage = Q(s,a)(Estimation, DReward) - V(s)
- Actor : Actor of Actor Critic
- Shares Gradients not observations
Related Work - A2C

- Improvement over A3C
- Sync
- Waits till every actor to finish
- Better GPU utilization (IMPALA is better!)
IMPALA

- Actors collect experiences not gradients.
- Independent actors and learners.
- Decoupling the acting and learning causes the policy in the actor to lag behind the learner.
- Solution V-Trace.
IMPALA...

A3C

IMPALA - Single Learner

IMPALA - Multiple Learners
V-Trace : Target

\[ v_s \overset{\text{def}}{=} V(x_s) + \sum_{t=s}^{s+n-1} \gamma^{t-s} \left( \prod_{i=s}^{t-1} c_i \right) \delta_t V, \]

\[ v_s = V(x_s) + \sum_{t=s}^{s+n-1} \gamma^{t-s} \left( r_t + \gamma V(x_{t+1}) - V(x_t) \right) \]
\[ = \sum_{t=s}^{s+n-1} \gamma^{t-s} r_t + \gamma^n V(x_{s+n}), \quad (2) \]
V-Trace : Target

\[
\pi_{\bar{\rho}}(a|x) \overset{\text{def}}{=} \frac{\min (\bar{\rho}\mu(a|x), \pi(a|x))}{\sum_{b \in A} \min (\bar{\rho}\mu(b|x), \pi(b|x))}, \quad (3)
\]

Remark 1. V-trace targets can be computed recursively:

\[
v_s = V(x_s) + \delta_s V + \gamma c_s (v_{s+1} - V(x_{s+1})).
\]

Remark 2. Like in Retrace(\lambda), we can also consider an additional discounting parameter \( \lambda \in [0, 1] \) in the definition of V-trace by setting \( c_i = \lambda \min (\bar{c}, \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)}) \). In the on-policy case, when \( n = \infty \), V-trace then reduces to TD(\lambda).
V-Trace: Policy Gradient, On Policy

In the on-policy case, the gradient of the value function $V^\mu(x_0)$ with respect to some parameter of the policy $\mu$ is

$$\nabla V^\mu(x_0) = \mathbb{E}_\mu \left[ \sum_{s \geq 0} \gamma^s \nabla \log \mu(a_s | x_s) Q^\mu(x_s, a_s) \right],$$

where $Q^\mu(x_s, a_s) \overset{\text{def}}{=} \mathbb{E}_\mu \left[ \sum_{t \geq s} \gamma^{t-s} r_t | x_s, a_s \right]$ is the state-action value of policy $\mu$ at $(x_s, a_s)$. 
V-Trace: Policy Gradient, Off Policy

Now in the off-policy setting that we consider, we can use an IS weight between the policy being evaluated $\pi_{\tilde{\rho}}$ and the behaviour policy $\mu$, to update our policy parameter in the direction of

$$\mathbb{E}_{a_s \sim \mu(\cdot | x_s)} \left[ \frac{\pi_{\tilde{\rho}}(a_s | x_s)}{\mu(a_s | x_s)} \nabla \log \pi_{\tilde{\rho}}(a_s | x_s) q_s | x_s \right]$$  \hspace{1cm} (4)
V-Trace : L2 Loss

Consider a parametric representation $V_\theta$ of the value function and the current policy $\pi_\omega$. Trajectories have been generated by actors following some behaviour policy $\mu$. The V-trace targets $v_s$ are defined by (1). At training time $s$, the value parameters $\theta$ are updated by gradient descent on the $l2$ loss to the target $v_s$

$$(v_s - V_\theta(x_s)) \nabla_\theta V_\theta(x_s),$$
V-Trace: Update Policy with Grads

\[
\rho_s \nabla_\omega \log \pi_\omega(a_s|x_s) \left( r_s + \gamma v_{s+1} - V_\theta(x_s) \right).
\]

In order to prevent premature convergence we may add an entropy bonus, like in A3C, along the direction

\[-\nabla_\omega \sum_a \pi_\omega(a|x_s) \log \pi_\omega(a|x_s).\]

The overall update is obtained by summing these three gradients rescaled by appropriate coefficients, which are hyper-parameters of the algorithm.
**Testing Environment: DMLab, Atari**

- DMLab (DeepMind Lab) provides a suite of challenging 3D navigation and puzzle-solving tasks for learning agents.
- The Arcade Learning Environment (ALE) is a simple object-oriented framework that allows researchers and hobbyists to develop AI agents for Atari 2600 games.
2 Network Architectures

Model Architectures. Left: Small architecture used for training on individual levels, containing 2 convolutional layers and 1.2 million parameters. Right: Large architecture used for training on DMLab-30 multi-task challenge, containing 15 convolutional layers and 1.6 million parameters.
Single task training

- Training agent individually on 5 different DMLAB tasks.
- Tasks: Planning task, two maze navigation tasks, a laser tag task with scripted bots and a simple fruit collection task
- IMPALA is more robust to the choice of hyperparameters than A3C.
- IMPALA achieves higher scores over a larger number of combinations than A3C.
<table>
<thead>
<tr>
<th>Architecture</th>
<th>CPUs</th>
<th>GPUs</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Task 1</td>
</tr>
<tr>
<td><strong>Single-Machine</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3C 32 workers</td>
<td>64</td>
<td>0</td>
<td>6.5K</td>
</tr>
<tr>
<td>Batched A2C (sync step)</td>
<td>48</td>
<td>0</td>
<td>9K</td>
</tr>
<tr>
<td>Batched A2C (sync step)</td>
<td>48</td>
<td>1</td>
<td>13K</td>
</tr>
<tr>
<td>Batched A2C (sync traj.)</td>
<td>48</td>
<td>0</td>
<td>16K</td>
</tr>
<tr>
<td>Batched A2C (dyn. batch)</td>
<td>48</td>
<td>1</td>
<td>16K</td>
</tr>
<tr>
<td>IMPALA 48 actors</td>
<td>48</td>
<td>0</td>
<td>17K</td>
</tr>
<tr>
<td>IMPALA (dyn. batch) 48 actors¹</td>
<td>48</td>
<td>1</td>
<td>21K</td>
</tr>
<tr>
<td><strong>Distributed</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3C</td>
<td>200</td>
<td>0</td>
<td>46K</td>
</tr>
<tr>
<td>IMPALA</td>
<td>150</td>
<td>1</td>
<td>80K</td>
</tr>
<tr>
<td>IMPALA (optimised)</td>
<td>375</td>
<td>1</td>
<td>200K</td>
</tr>
<tr>
<td>IMPALA (optimised) batch 128</td>
<td>500</td>
<td>1</td>
<td>250K</td>
</tr>
</tbody>
</table>

¹ Limited by amount of rendering possible on a single machine. ² Nvidia P100
Figure 4. **Top Row:** Single task training on 5 DeepMind Lab tasks. Each curve is the mean of the best 3 runs based on final return. IMPALA achieves better performance than A3C. **Bottom Row:** Stability across hyperparameter combinations sorted by the final performance across different hyperparameter combinations. IMPALA is consistently more stable than A3C.
Multi task training

- Because of high throughput, feasible to train multiple tasks at the same time,
- Allocation of fixed number of actors per task,
- Multiple learners use different GPUs,
- Learners share gradients,
- A3C - 7.5 Days / IMPALA - 10 Hours (DMLab-30)
<table>
<thead>
<tr>
<th>Model</th>
<th>Test score</th>
</tr>
</thead>
<tbody>
<tr>
<td>A3C, deep</td>
<td>23.8%</td>
</tr>
<tr>
<td>IMPALA, shallow</td>
<td>37.1%</td>
</tr>
<tr>
<td>IMPALA-Experts, deep</td>
<td>44.5%</td>
</tr>
<tr>
<td>IMPALA, deep</td>
<td>46.5%</td>
</tr>
<tr>
<td>IMPALA, deep, PBT</td>
<td>49.4%</td>
</tr>
<tr>
<td>IMPALA, deep, PBT, 8 learners</td>
<td>49.1%</td>
</tr>
</tbody>
</table>

*Table 3. Mean capped human normalised scores on DMLab-30. All models were evaluated on the test tasks with 500 episodes per task. The table shows the best score for each architecture.*
The graph shows the mean capped normalized score over environment frames for different algorithms. The algorithms compared are:

- IMPALA, deep, PBT - 8 GPUs
- IMPALA, shallow
- IMPALA, deep
- IMPALA-Experts, deep
- A3C, deep

The x-axis represents the environment frames, while the y-axis represents the mean capped normalized score. The graph illustrates how each algorithm performs over time, with different lines indicating the performance of each algorithm.
<table>
<thead>
<tr>
<th>Human Normalised Return</th>
<th>Median</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>A3C, shallow, experts</td>
<td>54.9%</td>
<td>285.9%</td>
</tr>
<tr>
<td>A3C, deep, experts</td>
<td>117.9%</td>
<td>503.6%</td>
</tr>
<tr>
<td>IMPALA, shallow, experts</td>
<td>93.2%</td>
<td>466.4%</td>
</tr>
<tr>
<td>IMPALA, deep, experts</td>
<td>191.8%</td>
<td>957.6%</td>
</tr>
<tr>
<td>IMPALA, deep, multi-task</td>
<td>59.7%</td>
<td>176.9%</td>
</tr>
</tbody>
</table>

Table 4. Human normalised scores on Atari-57. Up to 30 no-ops at the beginning of each episode.
Conclusion

- IMPALA was better than A3C in performance
- First Deep-RL agent that has been tested on such large scale.
- Shows transfer learning is possible between multiple tasks
Thank You

Prabhant Singh and Basar Turgut