Off-Policy Learning (Part 2)


Yugdeep Bangar
April 9, 2019
Leveraging Off-Policy methods

**IMPALA** - Distributed architecture for greater utilization of GPUs

**Soft-Actor Critic** - Practical RL for non-simulation environments
IMPALA
(Importance Weighted Actor-Learner Architectures)
Progress in Deep RL for expert agents

Silver, Schrittwieser, simonyan et al. (2017)
General Agents

Atari
General Agents

dm_contol
General Agents

DMLab-30
Objective

Solve large collection of tasks (e.g. DMLab-30), with a single reinforcement learning agent (network), and a single set of parameters
Challenges for a general agent

**Data efficiency** - (number of tasks) * (hundreds of millions of frames for each task) ?

**Stability** - multiple hyperparameters?

**Scale** - bigger networks?

**Task interference** - interference or positive transfer?
Approach

A scalable distributed agent (IMPALA)

Off-policy correction method V-trace
Asynchronous Advantage Actor-Critic (A3C)

- Agent learns a policy and a state value function
- Uses bootstrapped n-step return to reduce variance over REINFORCE with a baseline
- Distributed experience collection
- Adding more actor/learners leads to stale gradients
- Not GPU friendly
IMPALA - Single Learner

Centralized learner(s) and distributed actors

Actors receive parameters but send observations

Centralized learner can parallelize as much of the forward and backward passes as possible
Update timeline

Batched A2C

- Rendering time variance
- Low GPU utilisation

IMPALA - decoupled backward pass

- Acting decoupled from learning
- Actor parameters can lag by several updates
Update timeline

Batched A2C

- Rendering time variance
- Low GPU utilisation

IMPALA - decoupled backward pass

- Acting decoupled from learning
- Actor parameters can lag by several updates
Update timeline

Batched A2C
- Rendering time variance
- Low GPU utilisation

IMPALA - decoupled backward pass
- Acting decoupled from learning
- Actor parameters can lag by several updates

Easier to deal with stale experiences (using off-policy learning) than stale gradients
Actor-critic setup

- Learner off-policy $\pi$
- Baseline function $V^\pi$
- Local policy $\mu$

GPU-accelerated learner

Distributed actors

Problem: policy-lag between the actors and learner
V-trace

Principled off-policy advantage actor critic called V-Trace

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

where \( \rho_i \overset{\text{def}}{=} \min \left( \tilde{\rho}, \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)} \right) \) and \( c_i \overset{\text{def}}{=} \min \left( \tilde{c}, \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)} \right) \).

\( \rho_i \): which value function \( \mu \) or \( \pi \)

\( c_i \): speed of convergence

Weights are truncated (at most 1) to reduce variance
V-trace

Principled off-policy advantage actor critic called V-Trace

\[
v_s \overset{\text{def}}{=} V(x_s) + \sum_{t=s}^{s+n-1} \gamma^{t-s} \left( \Pi_{i=s}^{t-1} c_i \right) \frac{\rho_t (r_t + \gamma V(x_{t+1}) - V(x_t))}{\delta_t V}
\]

where \( \rho_i \overset{\text{def}}{=} \min \left( \tilde{\rho}_i, \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)} \right) \)
and \( c_i \overset{\text{def}}{=} \min (\tilde{c}, \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)}) \)

\( \rho_i \) : which value function \( \mu \) or \( \pi \)

\( c_i \) : speed of convergence

Weights are truncated (at most 1) to reduce variance
IMPALA - multiple learners

One or more GPU (or TPU) learners

Many CPU actors
Evaluation

Two networks

- Small CNN-LSTM
- Deep ResNet CNN-LSTM

Two test suites

- Atari-57
- DMLab-30 - language, memory, foraging, navigation
## Throughput

<table>
<thead>
<tr>
<th>Architecture</th>
<th>CPUs</th>
<th>GPUs(^1)</th>
<th>FPS(^2)</th>
</tr>
</thead>
<tbody>
<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 actors(^3)</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>

\(^1\) Nvidia P100 \(^2\) In frames/sec (4 times the agent steps due to action repeat). \(^3\) Limited by amount of rendering possible on a single machine.
### Throughput

<table>
<thead>
<tr>
<th>Architecture</th>
<th>CPUs</th>
<th>GPUs(^1)</th>
<th>FPS(^2)</th>
</tr>
</thead>
<tbody>
<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 9K</td>
</tr>
<tr>
<td>Batched A2C (sync step)</td>
<td>48</td>
<td>0</td>
<td>9K 5K</td>
</tr>
<tr>
<td>Batched A2C (sync step)</td>
<td>48</td>
<td>1</td>
<td>13K 5.5K</td>
</tr>
<tr>
<td>Batched A2C (sync traj.)</td>
<td>48</td>
<td>0</td>
<td>16K 17.5K</td>
</tr>
<tr>
<td>Batched A2C (dyn. batch)</td>
<td>48</td>
<td>1</td>
<td>16K 13K</td>
</tr>
<tr>
<td>IMPALA 48 actors</td>
<td>48</td>
<td>0</td>
<td>17K 20.5K</td>
</tr>
<tr>
<td>IMPALA (dyn. batch) 48 actors(^3)</td>
<td>48</td>
<td>1</td>
<td>21K 24K</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 50K</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>

\(^1\) Nvidia P100  
\(^2\) In frames/sec (4 times the agent steps due to action repeat).  
\(^3\) Limited by amount of rendering possible on a single machine.
Performance and stability

- IMPALA - 1 GPU - 200 actors
- A3C - Single Machine - 32 workers
- Batched A2C - Single Machine - 32 workers
- A3C - Distributed - 200 workers

Plots showing performance metrics for different environments and hyperparameter combinations.
DMLab-30

Mean Capped Normalized Score vs Environment Frames

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

Mean Capped Normalized Score vs Wall Clock Time (hours)
Shallow vs Deep networks

[Graph showing performance comparison between shallow and deep networks over time and wall clock time]
Integrating PBT

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

Mean Capped Normalized Score vs.
- Environment Frames
- Wall Clock Time (hours)
Scalability

![Graph showing time vs. environment frames and wall clock time for different algorithms.](image-url)
Multi-agent vs experts
Level Breakdown
Level Breakdown

<table>
<thead>
<tr>
<th>Task Description</th>
<th>A3C, deep</th>
<th>IMPALA-Experts, deep</th>
<th>IMPALA, deep, PBT</th>
</tr>
</thead>
<tbody>
<tr>
<td>language_select_described_object</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>language_answer_quantitative_question</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>language_select_located_object</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>explore_goal_locations_small</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rooms_collect_good_objects_train</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>explore_obstructed_goals_small</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>explore_object_locations_small</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>explore_object_locations_large</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>natlab_varying_map_randomized</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>natlab_varying_map_regrowth</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>explore_goal_locations_large</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>explore_obstructed_goals_large</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>explore_object_rewards_many</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rooms_watermaze</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rooms_select_nonmatching_object</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>explore_object_rewards_few</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pyschlab_continuous_recognition</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lasertag_three_opponents_small</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>skymaze_irreversible_path_varied</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rooms_keys_doors_puzzle</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>natlab_fixed_large_map</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Positive transfer
Level Breakdown

Positive transfer
Level Breakdown

Positive transfer

Not so much!
Takeaways

● **Efficient** and **scalable** Deep-RL agent
  ○ Efficient on single machine
  ○ Scales to 1000s of machines

● New off-policy correction (**V-trace**)  

● New level-suite **DMLab-30**

● **Strong multi-task performance** with **some positive transfer**

● **Deeper networks perform better**

● **Effective across wide range of RL problems**
Resources

TensorFlow Implementation: https://github.com/deepmind/scalable_agent

DeepMind Lab: https://github.com/deepmind/lab

DeepMind blog: https://deepmind.com/blog/impala-scalable-distributed-deeprl-dmlab-30/


Lectures

- ICLR 2018 (Koray Kavukcuoglu): https://www.youtube.com/watch?v=N5oZlO8pE40
Soft Actor Critic
Desired features for real world applications

Sample efficiency
No sensitive hyperparameters
Off-policy learning
Desired features for real world experimentation

Asynchronous sampling
Stop/resume training
Action smoothing
Simultaneously maximizing reward and entropy (MaxEnt)

\[ J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t, a_t) \sim \rho_\pi} \left[ r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot | s_t)) \right]. \]
Simultaneously maximizing reward and entropy (MaxEnt)

\[ J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t, a_t) \sim \rho_\pi} [r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot | s_t))] \]
Simultaneously maximizing reward and entropy

\[ J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t, a_t) \sim \rho_\pi} \left[ r(s_t, a_t) + \alpha H(\pi(\cdot | s_t)) \right]. \]

How to optimize it?
Optimizing the MaxEnt Objective

Soft Q-learning?
Optimizing the MaxEnt Objective

Soft Q-learning methods?

- Intractable in continuous domains
- Continuous solutions rely on biased approximations
Optimizing the MaxEnt Objective

Soft Q-learning (SQL)?

- Intractable in continuous domains
- Continuous solutions rely on biased approximations

Proposed solution: Soft actor-critic (SAC)
Optimizing the MaxEnt Objective

Soft Q-learning (SQL)?

- Intractable in continuous domains
- Continuous solutions rely on biased approximations

Proposed solution: **Soft actor-critic (SAC)**

- Learns the soft Q-function of policy and the policy jointly.
- Similar to DDPG, but with a stochastic policy
- Easy to implement, sample efficient, and stable
Soft Policy Iteration

1. **Soft policy evaluation**: Fix policy, apply soft Bellman backup until converges:

   \[
   T^\pi Q(s_t, a_t) \triangleq r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p} [V(s_{t+1})],
   \]
   \[
   V(s_t) = \mathbb{E}_{a_t \sim \pi} \left[ Q(s_t, a_t) - \log \pi(a_t | s_t) \right]
   \]

   This converges to \( Q^\pi \).

2. **Soft policy improvement**: Update the policy through information projection

   \[
   \pi_{\text{new}} = \text{arg min}_{\pi' \in \Pi} \text{D}_{\text{KL}} \left( \pi'(\cdot | s_t) \parallel \frac{\exp(Q^{\pi_{\text{old}}}(s_t, \cdot))}{Z^{\pi_{\text{old}}}(s_t)} \right)
   \]

   From the new policy, we have \( Q^{\pi_{\text{new}}}(s_t, a_t) \geq Q^{\pi_{\text{old}}}(s_t, a_t) \)
Soft Policy Iteration to Soft Actor-Critic

Use function approximators

Alternate optimization between Q-function ($V$ parameterized by $\psi$) and policy network ($\pi$ parameterized by $\phi$) with SGD

Additional network - Soft Q-function ($Q$ parameterized by $\theta$)
Soft Actor Critic

Initialize parameter vectors $\psi, \bar{\psi}, \theta, \phi$.

for each iteration do
  for each environment step do
    $a_t \sim \pi_{\phi}(a_t|s_t)$
    $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
    $D \leftarrow D \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
  end for
  for each gradient step do
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\phi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau \bar{\psi} + (1 - \tau)\psi$
  end for
end for
Simulated benchmarks
Real World experiments
Real World experiments
Resources

Implementations

- https://github.com/rail-berkeley/softlearning (by authors)
- https://github.com/vitchyr/rlkit
- https://github.com/openai/spinningup

Blogs/Tutorials

- https://bair.berkeley.edu/blog/2018/12/14/sac/

Questions?