IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures

Huizhuo Yuan

Peking University

March 25, 2019

Multi-task DeepRL

- General agents that are able to do many tasks simultaneously.
- Going from one network per task to one network for tens of tasks with many challenges.
- Data Efficienecy- Hundreds of millions of frames for a single task.
- Stability: Do we need task-specific hyperparameters?
- Scale: More complicated architecture and slower to train.
- Task Interference: Will multiple tasks cause interference or positive transfer.

RL Formulation

- Agent interacting with the environment. At each step t:
  1. Agent takes action a_t
  2. Environment returns reward r_{t+1} and state s_{t+1}
- Maximize total future reward

$$r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots$$

- For a policy π the action value function Q:

$$Q^\pi(s,a) = \mathbb{E}_[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \mid s_t = s, a_t = a]$$

$$= \mathbb{E}[r_t + \gamma Q^\pi(s_{t+1}, a_{t+1}) \mid s_t = s, a_t = a]$$

- Q represents how good an action a is given state s.

Optimal Value Functions

- An optimal value function give the maximal achievable value:

$$Q^*(s,a) = \max_\pi Q^\pi(s,a)$$

- Given an optimal value function we can get an optimal policy:

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

- Optimal value functions also obay a Bellman Equation.

$$Q^*(s_t,a_t) = \mathbb{E}[r_t + \gamma \max_{s'} Q^*(s_{t+1},a')]$$

DQN

- High-level idea is to make Q-learning look like supervised learning
- Optimize the Q-learning loss with minibatch SGD
- Apply Q-learning updates on batches of past experience instead of online
    1. Experience replay
    2. Previously used for better data efficiency
    3. Makes the data distribution more stationary
- Use an older set of weights to compute the targets, keeps the target function from changing too quickly

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim D}(r + \gamma max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2$$

Policy Gradient Methods

- An alternatively class of methods directly optimize the expected return of a policy:

$$\nabla_\theta J(\theta) = \nabla E[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots]$$

- For all differentiable policies

$$\nabla_\theta J(\theta) = E[\nabla_\theta log \pi_\theta(a \mid s) Q^\pi(s, a)]$$

where expectations is over states and actions.

- There is an sample based easy unbiased estimation (REINFORCE)

$$\nabla_\theta log \pi_\theta(a|s) R_t$$

where

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

- Start with a guess for each $Q(s,a)$
- Interact with the environment using some policy based on Q collecting tuples of experience $\{s_t, a_t, r_t, s_{t+1}, \cdots\}$, $e.g.\varepsilon$-greedy.
- Apply updates based on the Bellman equation

$$Q(s,a) \leftarrow Q(s,a) + (r + \gamma \max_a Q(s', a') - Q(s,a))$$

- $Q(s,a)$ is guaranteed to converge to the optimal value function Q^* under some reasonable assumptions.

Asynchronous Advantage Actor-Critic (A3C)

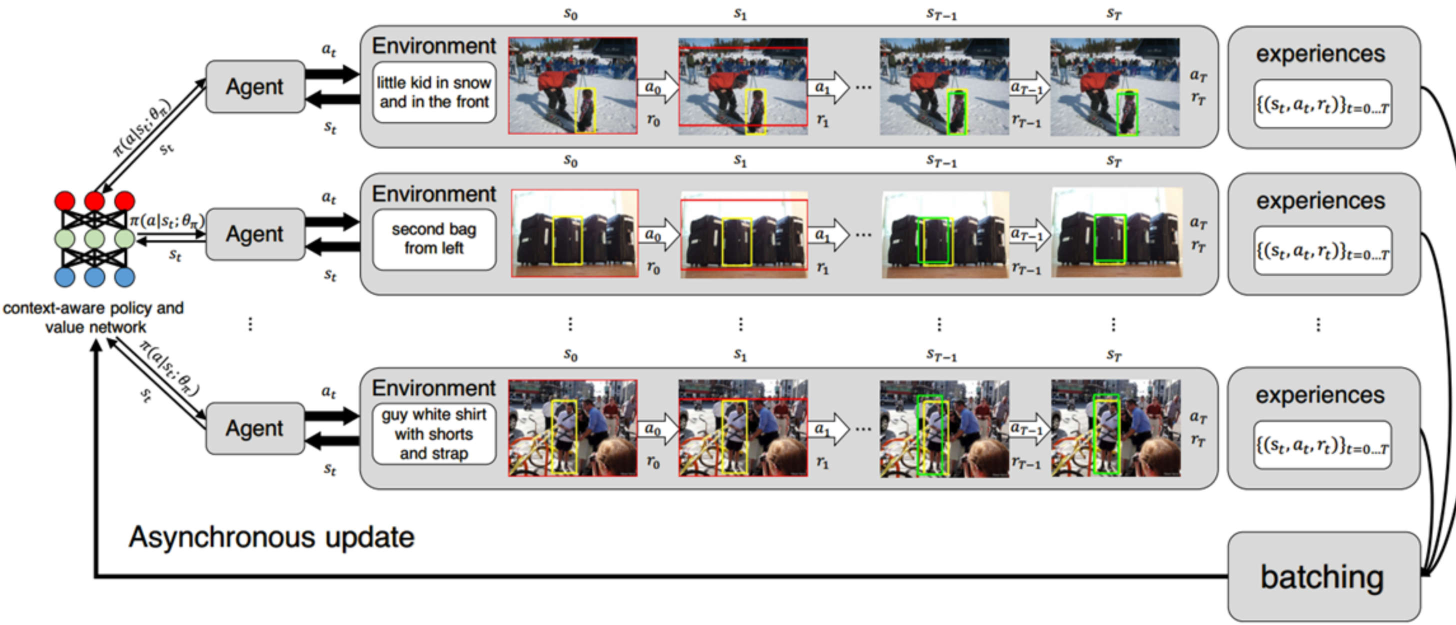
- The agent learns a policy and a state value function
- Uses bootstrapped n-step returns to reduce variance over REINFORCE with a baseline
- The policy gradient multiplied by an estimate of the advantage. Similar to Generalized Advantage Estimation (Schulman et al. 2015)

$$\nabla_\theta log\pi(a_t \mid s_t, \theta)(\sum_{k=0}^{N} \gamma^k r_{t+k} + \gamma^{N+1} V(s_{t+N+1}) - V(s_t))$$

- The critic/value function is trained with n-step TD learning. i.e. by minimizing the MSE

$$(\sum_{k=0}^{N} \gamma^k r_{t+k} + \gamma^{N+1} V(s_{t+N+1;\theta^-}) - V(s_t;\theta))^2$$

Asynchronous Advantage Actor-Critic (A3C)

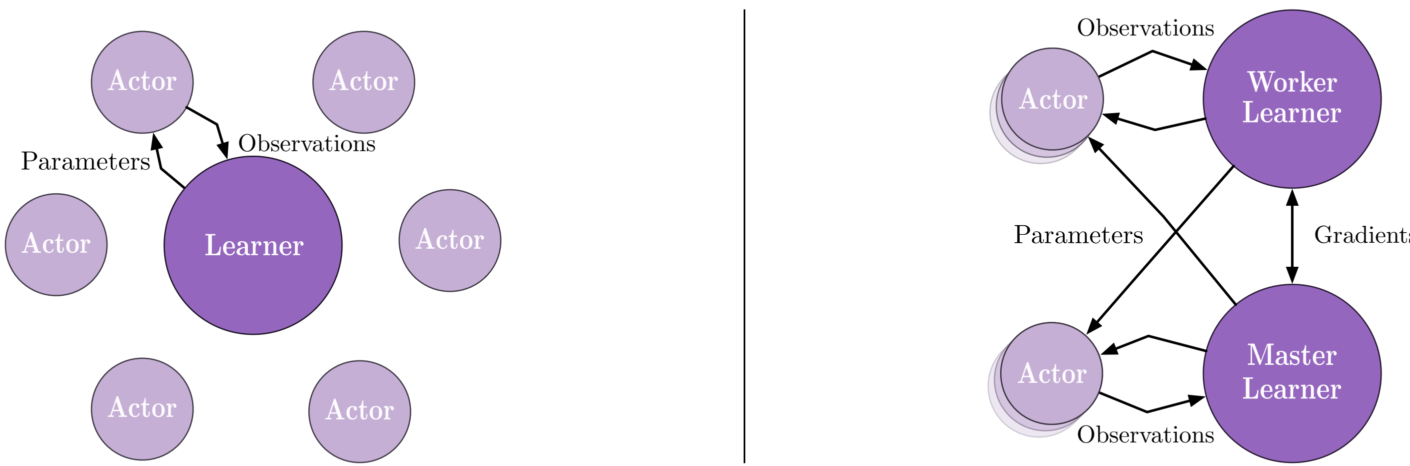


Scaling Up Distributed RL

- Our goal was to scale up A3C since it has more of the desired properties of a good multi-task agent
- Adding more actor/learners does not scale
- Distributed experience collection is good
- Communicating gradients is bad

A Better Architecture

- It is better to use a centralized learner(s) and distribute the acting
- Actors receive parameters but send observations
- The centralized learner can parallelize as much of the forward and backward passes as possible



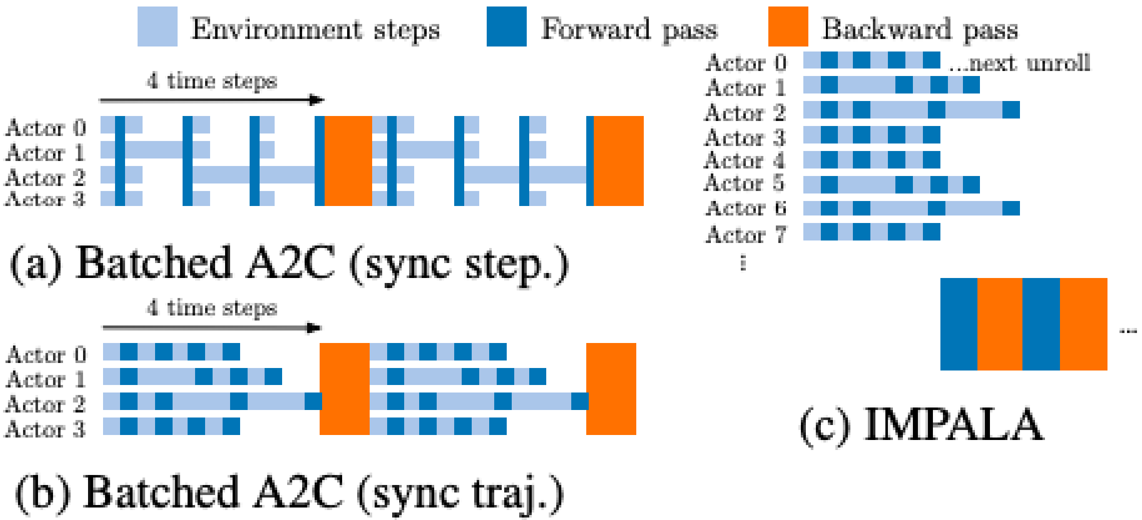


Figure 2. Timeline for one unroll with 4 steps using different architectures. Strategies shown in **(a)** and **(b)** can lead to low GPU utilisation due to rendering time variance within a batch. In **(a)**, the actors are synchronised after every step. In **(b)** after every n steps. IMPALA **(c)** decouples acting from learning.

Decoupled Backward Pass

- It is more efficient to decouple the backward pass
- Actors generate trajectories/unrolls and place them into a queue
- The learner continuously dequeues batches of trajectories and performs parameter updates
- Key Challenge:
  1. Decoupling the backwards pass requires off-policy learning
  2. Actor parameters can lag by several updates
- POD architecture-Parallel Off-policy Decoupled

V-Trace

- The experience generated by the actors can lag behind the learner's policy
- We introduce a principled off-policy advantage actor critic called V-Trace
- The V-Trace corrected estimate for the value $V(x_s)$ is:

$$v_s \stackrel{\text{def}}{=} V(x_s) + \sum_{t=s}^{s+n-1} \gamma^{t-s} \Big(\prod_{i=s}^{t-1} c_i \Big) \rho_t \big(r_t + \gamma V(x_{t+1}) - V(x_t) \big), \quad (1)$$

where $\rho_t \stackrel{\text{def}}{=} \min\left(\bar{\rho}, \frac{\pi(a_t|x_t)}{\mu(a_t|x_t)}\right)$ and $c_i \stackrel{\text{def}}{=} \min\left(\bar{c}, \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)}\right)$

- The V-Trace update for the value function is:

$$\big(v_s - V_\theta(x_s)\big) \nabla_\theta V_\theta(x_s)$$

- The V-Trcace update for the policy is:

$$\rho_s \nabla_\omega \log \pi_\omega(a_s|x_s) \big(r_s + \gamma v_{s+1} - V_\theta(x_s)\big)$$

- In the on-policy case reduces to the on-policy n-steps Bellman target.
- V-trace targets can be computed recursively:

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

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

$$
\begin{aligned}
\frac{1}{\alpha}\Delta V_t^{\lambda}(s_t) &= R_t^{\lambda} - V_t(s_t) \\
&= -V_t(s_t) + (1-\lambda)\lambda^0[r_{t+1} + \gamma V_t(s_{t+1})] \\
&\qquad + (1-\lambda)\lambda^1[r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2})] \\
&\qquad + (1-\lambda)\lambda^2[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V_t(s_{t+3})]
\end{aligned}
$$

$$
\begin{aligned}
\frac{1}{\alpha}\Delta V_t^{\lambda}(s_t) &= -V_t(s_t) \\
&\quad + (\gamma\lambda)^0\left[r_{t+1} + \gamma V_t(s_{t+1}) - \gamma\lambda V_t(s_{t+1})\right] \\
&\quad + (\gamma\lambda)^1\left[r_{t+2} + \gamma V_t(s_{t+2}) - \gamma\lambda V_t(s_{t+2})\right] \\
&\quad + (\gamma\lambda)^2\left[r_{t+3} + \gamma V_t(s_{t+3}) - \gamma\lambda V_t(s_{t+3})\right] \\
&\quad \vdots \\
&= \quad (\gamma\lambda)^0\left[r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)\right] \\
&\quad + (\gamma\lambda)^1\left[r_{t+2} + \gamma V_t(s_{t+2}) - V_t(s_{t+1})\right] \\
&\quad + (\gamma\lambda)^2\left[r_{t+3} + \gamma V_t(s_{t+3}) - V_t(s_{t+2})\right]
\end{aligned}
$$

Figure 1

- The weight ρ_t appears in the definition of the temporal difference $\delta_t V$ and defines the fixed point of this update rule. The weights c_i are similar to the "trace cutting" coefficients in Retrace.
- Notice that (**??**) estimates the policy gradient for $\pi_{\bar{\rho}}$ which is the policy evaluated by the V-trace algorithm when using a truncation level $\bar{\rho}$. However assuming the bias $V^{\pi_{\bar{\rho}}} - V^{\pi}$ is small (e.g. if $\bar{\rho}$ is large enough) then we can expect q_s to provide us with a good estimate of $Q^{\pi}(x_s, a_s)$.

$$\begin{aligned}\mathbb{E}[q_s|x_s,a_s] &= r_s + \gamma\mathbb{E}\big[V^{\pi_{\bar\rho}}(x_{s+1}) + \delta_{s+1}V^{\pi_{\bar\rho}} + \gamma c_{s+1}\delta_{s+2}V^{\pi_{\bar\rho}} + \ldots\big] \\ &= r_s + \gamma\mathbb{E}\big[V^{\pi_{\bar\rho}}(x_{s+1})\big] \\ &= Q^{\pi_{\bar\rho}}(x_s,a_s)\end{aligned}$$

whereas

$$\begin{aligned}\mathbb{E}[v_s|x_s,a_s] &= V^{\pi_{\bar\rho}}(x_s) + \rho_s\Big(r_s + \gamma\mathbb{E}\big[V^{\pi_{\bar\rho}}(x_{s+1})\big] - V^{\pi_{\bar\rho}}(x_s)\Big) + \gamma c_s\delta_{s+1}V^{\pi_{\bar\rho}} \\ &= V^{\pi_{\bar\rho}}(x_s) + \rho_s\Big(r_s + \gamma\mathbb{E}\big[V^{\pi_{\bar\rho}}(x_{s+1})\big] - V^{\pi_{\bar\rho}}(x_s)\Big) \\ &= V^{\pi_{\bar\rho}}(x_s)(1-\rho_s) + \rho_s Q^{\pi_{\bar\rho}}(x_s,a_s),\end{aligned}$$

which is different from $Q^{\pi_{\bar\rho}}(x_s,a_s)$ when $V^{\pi_{\bar\rho}}(x_s) \neq Q^{\pi_{\bar\rho}}(x_s,a_s)$.

Define the V-trace operator $\mathcal{R}$:

$$\mathcal{R}V(x) \stackrel{\text{def}}{=} V(x) + \mathbb{E}_\mu\Big[\sum_{t\geq 0}\gamma^t\big(c_0\ldots c_{t-1}\big)\rho_t\big(r_t + \gamma V(x_{t+1}) - V(x_t)\big)\Big], \quad (2)$$

Theorem

Let $\rho_t = \min\left(\bar{\rho}, \frac{\pi(a_t|x_t)}{\mu(a_t|x_t)}\right)$ and $c_t = \min\left(\bar{c}, \frac{\pi(a_t|x_t)}{\mu(a_t|x_t)}\right)$ be truncated importance sampling weights, with $\bar{\rho} \geq \bar{c}$. Assume that there exists $\beta \in (0,1]$ such that $\mathbb{E}_\mu \rho_0 \geq \beta$. Then the operator $\mathcal{R}$ defined by (2) has a unique fixed point $V^{\pi_{\bar{\rho}}}$, which is the value function of the policy $\pi_{\bar{\rho}}$ defined by

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

Furthermore, $\mathcal{R}$ is a η-contraction mapping in sup-norm, with

$$\eta \stackrel{\text{def}}{=} \gamma^{-1} - (\gamma^{-1} - 1)\mathbb{E}_\mu\Big[\sum_{t\geq 0}\gamma^t\Big(\prod_{i=0}^{t-2} c_i\Big)\rho_{t-1}\Big] \leq 1 - (1-\gamma)\beta < 1.$$

First notice that we can rewrite $\mathcal{R}$ as

$$\mathcal{R}V(x) = (1 - \mathbb{E}_\mu \rho_0) V(x) + \mathbb{E}_\mu \left[\sum_{t \geq 0} \gamma^t \Big(\prod_{s=0}^{t-1} c_s \Big) \Big(\rho_t r_t + \gamma [\rho_t - c_t \rho_{t+1}] V(x_{t+1}) \Big) \right].$$

Thus

$$\mathcal{R}V_1(x) - \mathcal{R}V_2(x) = \mathbb{E}_\mu \left[\sum_{t \geq 0} \gamma^t \Big(\prod_{s=0}^{t-2} c_s \Big) \underbrace{[\rho_{t-1} - c_{t-1}\rho_t]}_{\alpha_t} \big[V_1(x_t) - V_2(x_t) \big] \right],$$

Theorem

Assume a tabular representation, i.e. the state and action spaces are finite. Consider a set of trajectories, with the k^{th} trajectory $x_0, a_0, r_0, x_1, a_1, r_1, \ldots$ generated by following μ: $a_t \sim \mu(\cdot|x_t)$. For each state x_s along this trajectory, update

$$V_{k+1}(x_s) = V_k(x_s) + \alpha_k(x_s) \sum_{t \geq s} \gamma^{t-s} \big(c_s \ldots c_{t-1}\big) \rho_t \big(r_t + \gamma V_k(x_{t+1}) - V_k(x_t)\big) \tag{4}$$

with $c_i = \min\left(\bar{c}, \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)}\right)$, $\rho_i = \min\left(\bar{\rho}, \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)}\right)$, $\bar{\rho} \geq \bar{c}$. Assume that (1) all states are visited infinitely often, and (2) the stepsizes obey the usual Robbins-Munro conditions: for each state x, $\sum_k \alpha_k(x) = \infty$, $\sum_k \alpha_k^2(x) < \infty$. Then $V_k \to V^{\pi_{\bar{\rho}}}$ almost surely.

Network Architecture

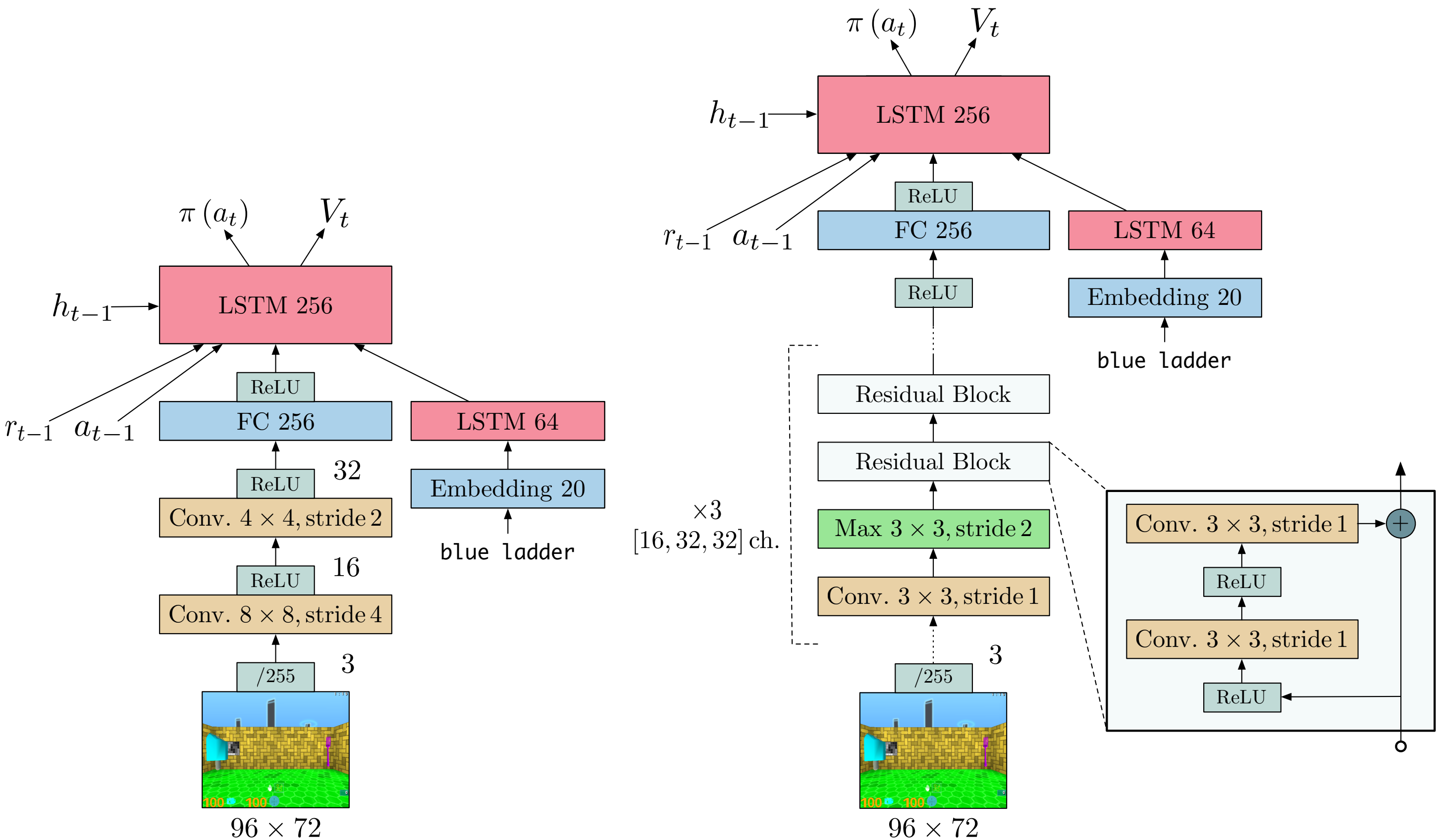


Figure 2: Model Architectures. **Left:** Small architecture, 2 convolutional layers and 1.2 million parameters. **Right:** Large architecture, 15 convolutional layers

Throughput Comparison

Architecture	CPUs	GPUs[1]	FPS[2]	
Single-Machine			Task 1	Task 2
A3C 32 workers	64	0	6.5K	9K
Batched A2C (sync step)	48	0	9K	5K
Batched A2C (sync step)	48	1	13K	5.5K
Batched A2C (sync traj.)	48	0	16K	17.5K
Batched A2C (dyn. batch)	48	1	16K	13K
IMPALA 48 actors	48	0	17K	20.5K
IMPALA (dyn. batch) 48 actors[3]	48	1	21K	24K
Distributed				
A3C	200	0	46K	50K
IMPALA	150	1	80K	
IMPALA (optimised)	375	1	200K	
IMPALA (optimised) batch 128	500	1	250K	

[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.

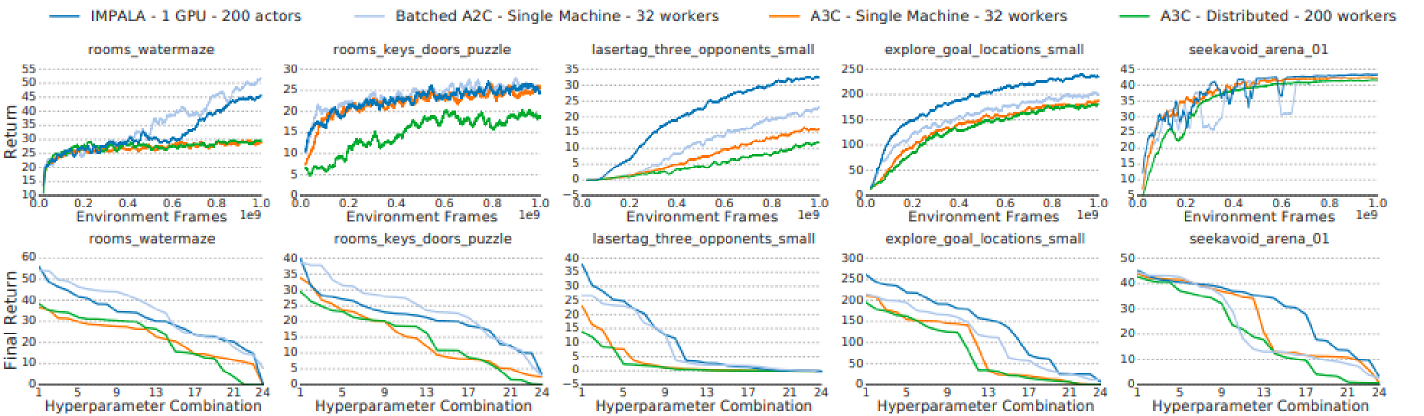
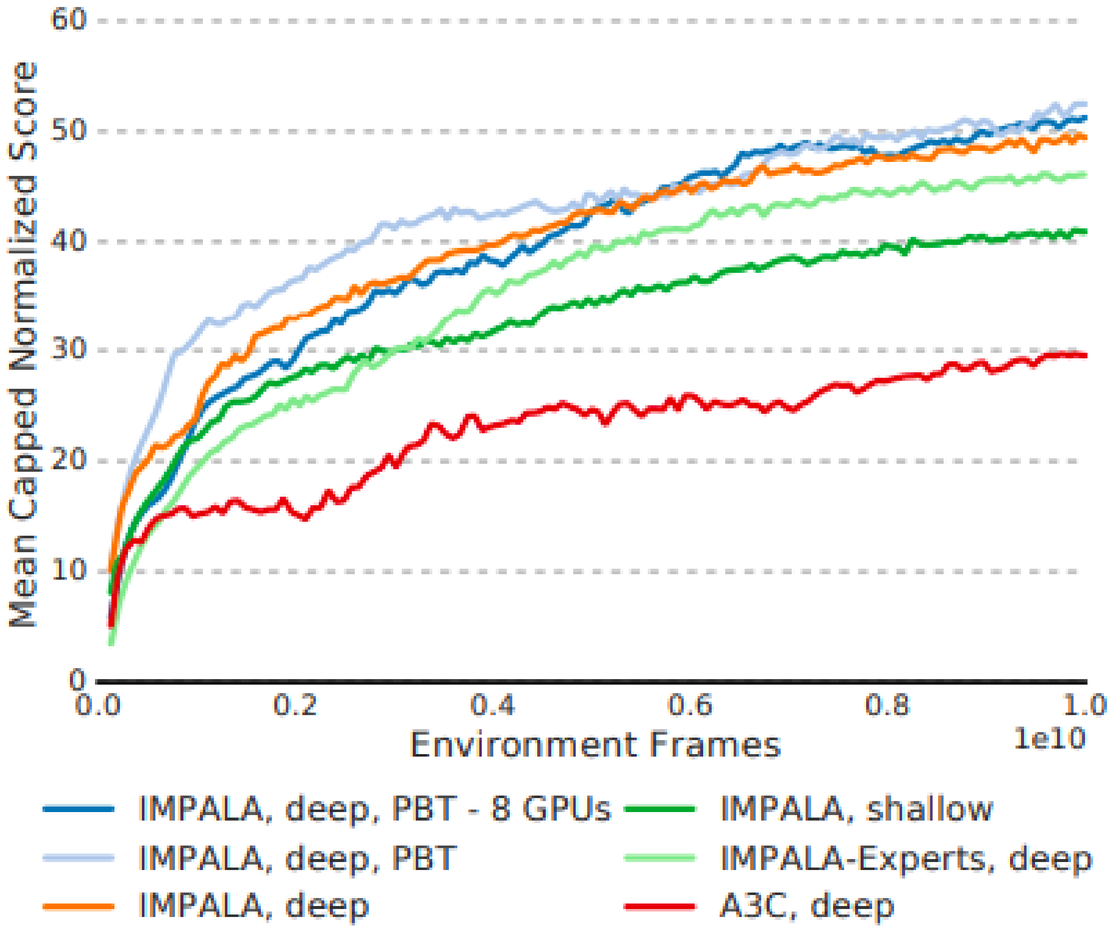


Figure 4. **Top Row:** Single task training on 5 DeepMind Lab tasks. Each curve is the mean of the best 3 runs based on final return. IMPALA achieves better performance than A3C. **Bottom Row:** Stability across hyperparameter combinations sorted by the final performance across different hyperparameter combinations. IMPALA is consistently more stable than A3C.

DMLab-30 Multi-Task

DMLab-30 Multi-Task Results



Conclusions

- New distributed RL architecture allows for stable learning with very high throughput
- Especially well-suited to the multi-task deep RL setting
- Synchronous batch learning is more robust to hyperparameters than async SGD
- Multi-task RL on the DMLab-30:
  1. Positive transfer
  2. Deep ResNets finally outperforms 3 layer ConvNets (Atari was too simple)